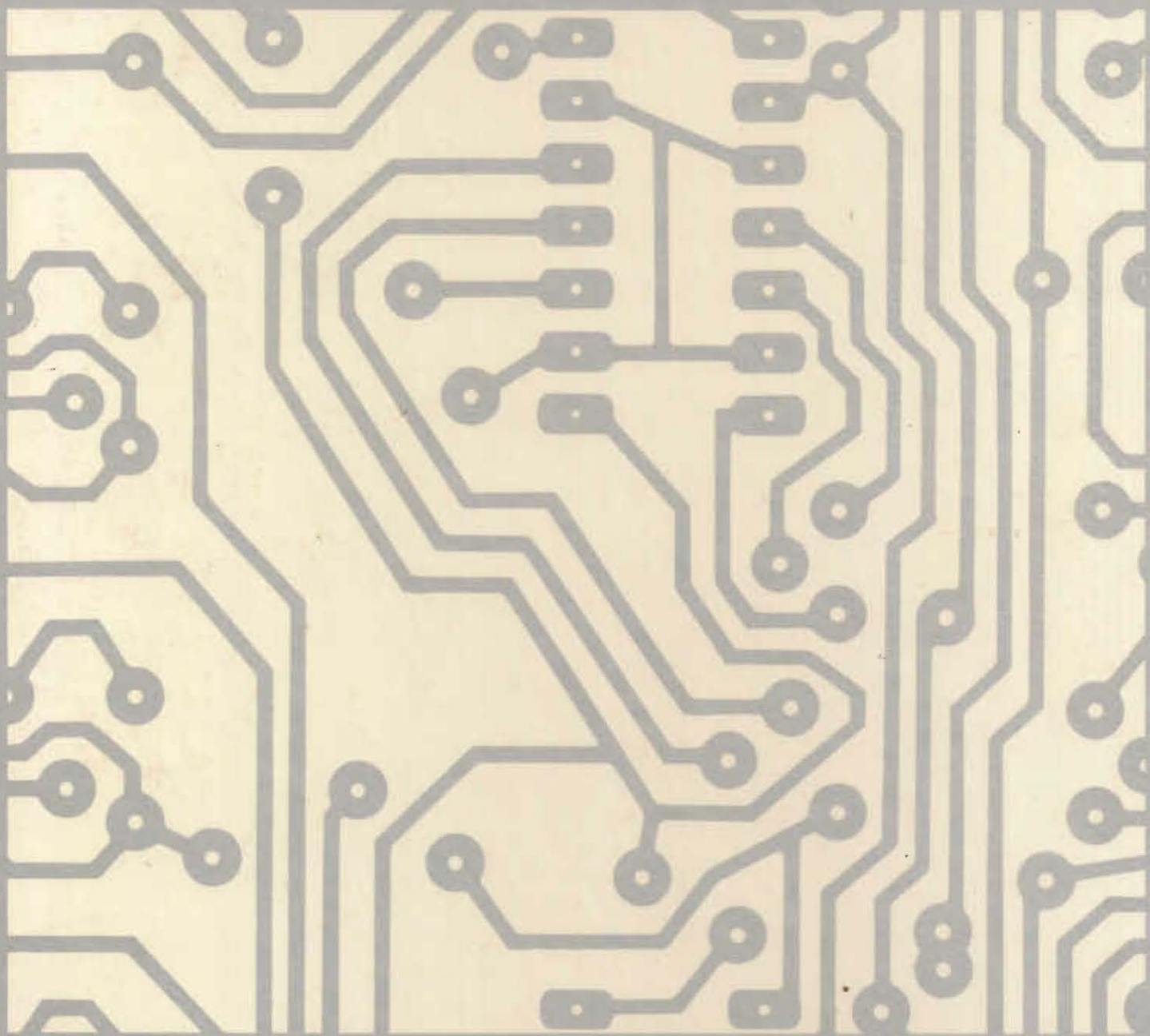
BASIC PLUS

ENCYCLOPEDIE D'INITIATION
A L'INFORMATIQUE ET AUX ORDINATEURS



EDITIONS DU HENNIN

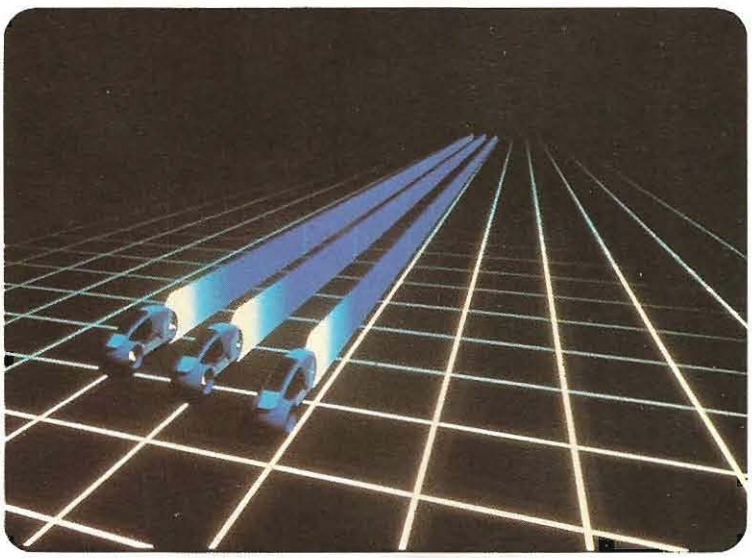
BASIC PLUS

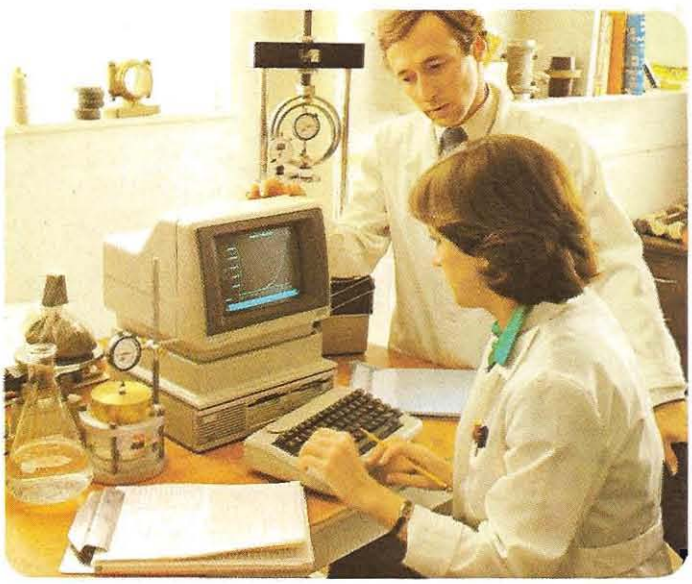
ENCYCLOPEDIE D'INITIATION A L'INFORMATIQUE ET AUX ORDINATEURS

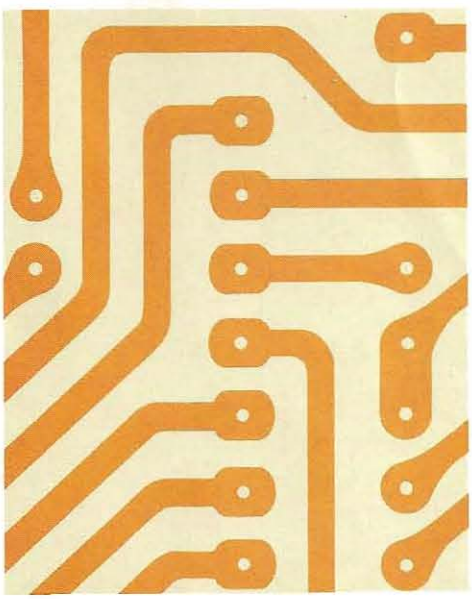


ENCYCLOPEDIE D'INITIATION A L'INFORMATIQUE ET AUX ORDINATEURS

EDITIONS
DU HENNIN







BASIC PLUS


FS02
MATERIALREKVISITION
KORT F-
KOD
ORDERNR
ARTIKELNR FÖR ENHET
UTLÄMNAT
ÅR MÅN DAG
SIGN
ARTIKELNUMMER
REKVIRERAD KVANT. SORT
UTLÄMNAD KVANT. SORT
FR.KONTO
STANDARDPRIS
KRONOR
ÖRE
5000273014
700 st
700 st
3968661
16 50
5000273019
140 st
85 st
3968661
16 25
5000273068
50 st
30 st
3968865
27 85
REKVIRERAD AV
ORS.BET. SIGN
MATERIAL SÄNDS TILL
4658/0004
Alfaskop

Stuart Franklin-Sygma

PRESENTATION

Jusqu'à ces dernières années, l'informatique a été le domaine réservé de quelques initiés qui, pour mener à bien des travaux très complexes, avaient besoin d'être secondés par des machines.
Imposantes, tant par leur taille que par l'aspect sérieux des hommes en blouses blanches qui les entouraient, ces machines n'équipaient que les centres d'industrie et de recherche les plus importants.

Aujourd'hui, les ordinateurs font partie de notre vie et sont en train de modifier notre culture. On les retrouve partout, dans les écoles, dans les usines, dans les bureaux et jusque dans les foyers, où ils sont parfois monopolisés par les enfants. Ils dirigent les trains, pilotent les avions et les engins spatiaux, transmettent nos communications téléphoniques. Ils remplissent aussi bien les fonctions de bibliothécaire et d'archiviste que de traducteur. Ils sont présents dans les blocs opératoires des hôpitaux, mais ils savent également jouer aux échecs, et tentent de prévoir la météo.

Nous sommes indiscutablement au seuil d'une ère nouvelle, dans laquelle l'ordinateur se banalisera au point d'être à la portée de tout le monde.
Il est facile de prévoir que d'ici peu un ordinateur familial, doté d'un écran et d'une imprimante ne coûtera pas plus cher qu'un magnétoscope et sera, par conséquent, très largement diffusé.
Avec l'ordinateur domestique nous pouvons, non seulement avoir des activités professionnelles, étudier, organiser des jeux et programmer la vie quotidienne de la famille, mais également aborder les sujets de l'information et de l'enseignement d'une manière tout à fait différente.
Mais se servir d'un ordinateur nécessite, comme toute technique nouvelle, un apprentissage. Il faut savoir dialoguer avec la machine, lui poser des questions intelligibles et pouvoir comprendre les informations qu'elle donne.
Une encyclopédie pratique de l'informatique répond à ces besoins.

Si vous avez décidé d'acquérir un ordinateur personnel, la lecture de BASIC PLUS orientera votre choix, mais avant tout, elle vous permettra d'apprendre le langage de l'ordinateur et ses applications.
Volontairement construite comme un cours et illustrée de schémas et de nombreuses photographies, l'encyclopédie BASIC PLUS guide le lecteur pas à pas dans sa découverte des structures et du fonctionnement d'un ordinateur, des services qu'il peut en attendre et jusqu'à la façon de concevoir et d'écrire un programme.
BASIC PLUS constitue un véritable cours adapté à l'usage de chacun.

Il est probable que d'ici quelques années celui qui ne sera pas capable d'utiliser un ordinateur se sente un marginal, comme celui qui, aujourd'hui, ne sait pas se servir d'un téléphone. Il était donc temps qu'un ouvrage comme BASIC PLUS voie le jour, et initie à l'informatique tous ceux qui, de plus en plus nombreux, comprennent l'intérêt de cette science dont les applications entrent tous les jours davantage dans notre vie quotidienne et dont l'influence sur notre civilisation ne manquera pas d'être décisive.

L'Editeur

Grazia Neri/Wheeler Pictures

SOMMAIRE

Comment est fait un ordinateur

Pour juger de la diversité et de la puissance des systèmes informatiques offerts par un marché en continuelle expansion, il est nécessaire de connaître la logique d'un ordinateur, ses fonctions, la manière dont elles sont mises en œuvre et leurs applications.

L'ordinateur, machine intelligente ?
Qu'est-ce qu'un programme et quel est l'intérêt d'une machine programmable. Les données : définition et utilisation. Le passé et l'avenir du calcul automatique : du boulier aux premiers essais pour concevoir une machine à calculer. Les calculatrices. L'idée d'une « programmation » : naissance du premier ordinateur. Développement de l'ordinateur : la miniaturisation mènera-t-elle aux machines « pensantes » ? Les principales applications professionnelles de l'ordinateur.

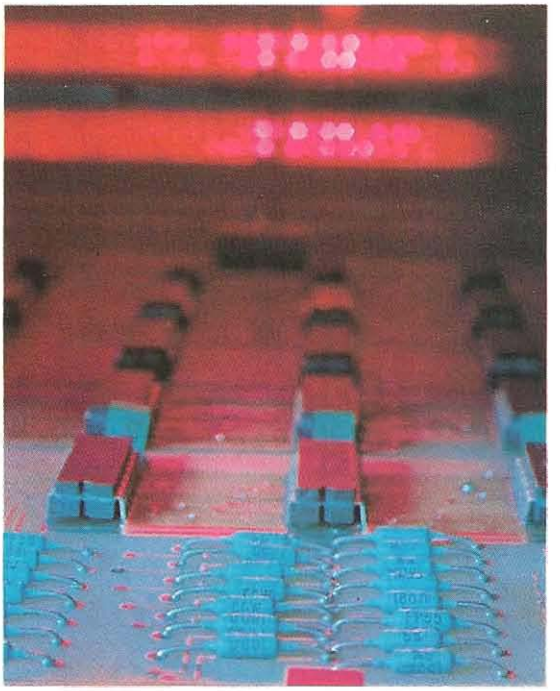
La logique de l'ordinateur
Comment « raisonne » un ordinateur. Machines analogiques et machines numériques. Un premier exemple d'application : la balance électronique. Classification des ordinateurs. Réseaux de transmission des données, liaisons entre ordinateurs. Organisation d'un centre de calcul.

Les systèmes de numération
Les impulsions : définition et exemples. Systèmes de numération dans le monde des ordinateurs. Traduction des nombres dans les diverses bases. Opérations arithmétiques en base 2. Tables de conversion des nombres en systèmes binaire, octal et hexadécimal.

Eléments de logique
Les opérateurs logiques : quels sont-ils et comment s'en sert-on. Les ensembles. Applications des opérateurs logiques : recherche et sélection dans les mémoires. Les circuits intégrés et leurs applications. Exemples de circuits intégrés qui développent des fonctions logiques. La protection des données : circuits pour la création des codes avec les opérateurs logiques.

Codes de transmission des données
Les codes : traduction de nos symboles alphabétiques et numériques en langage machine. Description du code ASCII. Les codes transparents, la codification des lettres et des nombres. Quelques codes de sécurité. Systèmes de transmission des données. Protocoles synchrones et asynchrones.

Microsystèmes
Structure d'un système à base de micro-ordinateur. Les principaux organes. Les Bus de liaison. Périphériques, méthodes et protocoles de transmission des données. La fonction de l'unité centrale (UC ou CPU) et les calculs qu'elle peut exécuter. Fonctionnement d'un système à microprocesseur : les cycles et les instructions. Compteurs de programme, registres, mémoires. Utilisation de la mémoire : cycles d'une instruction. Les sauts à l'intérieur d'un programme. Interaction avec le monde extérieur et gestion des interruptions. L'état de Hold. Technique DMA de transfert des données.



Archives Curcio/S. Petrelli

Traduction d'un problème pratique en un schéma logique

Qu'est-ce qu'un programme. Méthodes pour rationaliser et résoudre un problème pratique. Présentation des principaux langages symboliques. Les méthodes de stockage des données. Structuration des données. La logique à suivre pour la saisie des données.

La programmation
Les langages de programmation. Langages compilés et langages interprétés. La compilation. Langages évolués. Langages machine. Caractéristiques, champs d'utilisation et critères d'évaluation.

Analyse et organigrammes
Etablissement des programmes: analyse du problème et des données disponibles. Schématisation du problème et méthodes de structuration. Applications: définition des procédures de gestion d'un magasin. Organigrammes: définitions et symboliques. Emploi des symboles. Les index et les indicateurs d'état. Boucles et compteurs: méthodes d'utilisation, exemples d'applications. Informatique scientifique et informatique de gestion. Différentes méthodes de programmation.

Stockage des données
Principaux types de mémoire de masse. L'archivage pour l'emploi de micro et de mini-ordinateurs. Fichiers et enregistrements. Les divers types de fichiers: principales caractéristiques et champs d'utilisation. Structuration et organisation des données: quelques exemples. Archives complexes: méthodes de stockage. Les banques de données. Quelques exemples de procédures pour la gestion des banques de données.

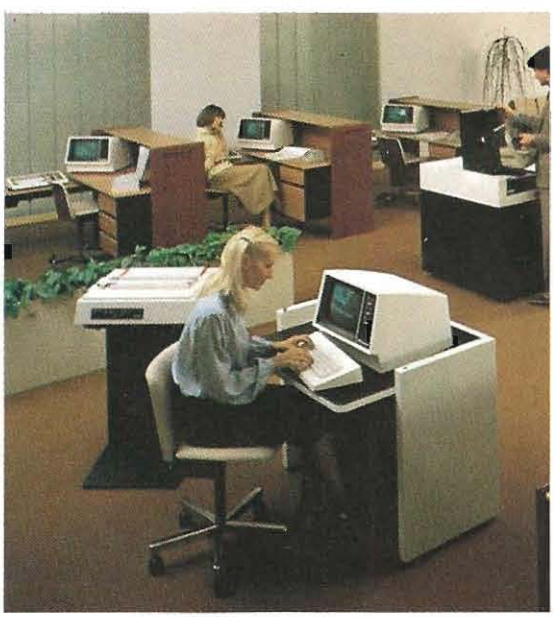
Etablissement des programmes
Présentation des principaux ordinateurs personnels et mini-ordinateurs. Tableaux comparatifs et critères d'évaluation. Segmentation des programmes en fonction de la complexité du problème. Critères de choix d'une machine.

Le langage de la programmation Basic

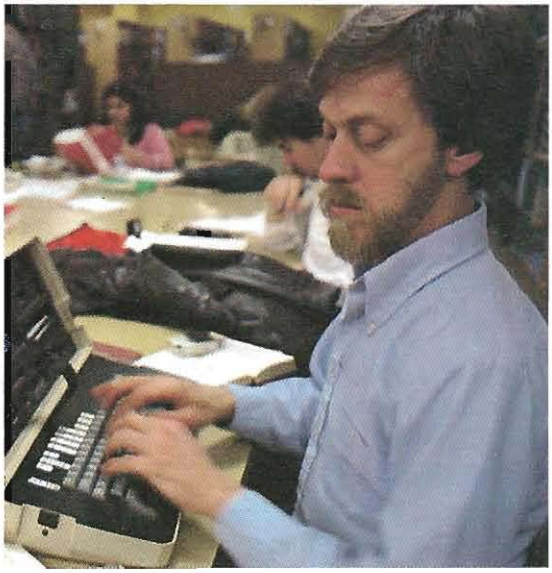
Théories et méthodes pour programmer en Basic ; le fonctionnement en mode conversationnel ; les fonctions mathématiques, ce qu'elles sont et leur développement en Basic ; méthodologies pour un usage correct des instructions Basic. Méthodes de division et de segmentation des programmes. Mise au point par segments de programme.

Généralités
Emploi du Basic interprété pour le déroulement des calculs: le Basic dans le mode conversationnel. Constantes et variables, définitions; précision dans le calcul. Représentation des nombres réels en format exponentiel. Représentation des nombres entiers et réels à l'intérieur de la machine. Les zones mémoire: mémoire programme et mémoire des données. Les calculs: priorités, précision et arrondis des calculs. Les variables de chaîne; ce qu'elles sont et comment on les utilise. Opérations avec les variables de chaîne. Différences de précision obtenue par les divers types d'ordinateurs personnels.

Les commandes
Les commandes du Basic. Ecriture et sauvegarde des programmes. Edition. Les instructions TRON et TROFF; la recherche des erreurs. Format des instructions Basic.



Honeywell



Grazia Neri/Contact

Les instructions
Instructions d'affectation pour les variables. Les boucles : utilisation et syntaxe. Les sauts : sauts d'instructions, appels de sous-programmes. Utilisation des sauts et des sous-programmes. Les opérateurs logiques dans la programmation. Les instructions conditionnelles. Usage de ces instructions pour le diagnostic. Applications : établissement et développement d'un programme pour le calcul des amortissements.

Les fonctions
Notion de fonction. Représentation graphique. Fonctions numériques en Basic : emploi, précision et limites. Fonctions numériques définies par l'utilisateur. Fonctions de chaîne en Basic. Fonctions de chaîne définies par l'utilisateur. Aperçus de statistique : génération des nombres aléatoires. Applications : établissement et écriture d'un programme pour résoudre des équations ; établissement et écriture de programmes pour la gestion des chaînes de caractères.

Les tableaux
Mémorisation des données. Définition des tableaux mono et multidimensionnels. Tableaux numériques et de chaîne. Les instructions d'attribution dans les tableaux. Application : établissement et écriture d'un programme pour effectuer des calculs avec un nombre quelconque de chiffres ; utilisation des tableaux pour la génération des messages d'erreurs ; établissement et écriture d'un programme pour le calcul des zones.

Fonctions d'entrée et de sortie des données
Instructions de lecture et d'impression des données. Instructions d'impression selon un format. Le curseur : commande de gestion de l'écran. Utilisation de l'imprimante, codes pour les fonctions particulières. Les tâches programmables : les instructions de programmation et leur utilisation. Applications : établissement et écriture d'un programme pour la mise en ordre des données. Les menus, systèmes de présentation et de lancement des programmes d'une procédure. Les instructions de manipulation des bits. Les instructions pour l'accès direct en mémoire.

Gestion des archives
Les formats de mémorisation des données sur disquette. Codification et transformation des données. Les instructions de lecture et d'écriture sur disquette. Fonctions numériques et de chaîne en corrélation avec la disquette. Méthodes de gestion des fichiers multivolumes. La gestion des fichiers complexes : description de quelques programmes pour la gestion des données. L'organisation de l'archivage ; problèmes connexes à cette fonction. Programmes utilitaires pour l'organisation des archives. Recherche et sélection des données : les sous-programmes de comparaison. Applications : établissement et écriture du programme de gestion d'un fichier d'adresses, de rubriques ou d'échéances ; établissement et écriture du programme de gestion d'un magasin ; établissement et écriture du sous-programme permettant d'arrondir les calculs.

Structuration des programmes
Principes de structuration des programmes. Technique d'approche Top-down. Généralisation des sous-programmes. Segmentation des programmes. Transfert des données entre programmes ou entre sous-programmes ; emploi de la disquette comme support. Les masques écran, exemples d'applications.

Conclusions
Méthodes de validation des programmes. Recherche des erreurs. Précis des instructions Basic. Applications : recours à plusieurs fichiers dans un même programme. Etablissement et préparation des organigrammes de premier niveau pour une procédure de facturation.

La programmation avancée

Trois autres langages importants du calculateur : l'Assembleur, le Cobol, le Fortran. Examen approfondi des instructions en Fortran, comparées à celles du Basic, et leurs applications scientifiques. Techniques de programmation en langages mixtes. L'emploi de l'ordinateur dans la réalisation des graphiques et des dessins. Les applications possibles, en particulier dans le domaine du contrôle automatique des outillages et des appareillages complexes.

L'Assembleur
Le langage Assembleur, structure et caractéristiques essentielles. Format des principaux types d'instructions en Assembleur. Quelques exemples. Le Macro-assembleur. Emploi de l'Assembleur ; quand et pourquoi il peut être utile. Applications : procédure de facturation, préparation des organigrammes.

Le Cobol
Le langage Cobol ; structure et caractéristiques essentielles. Quelques exemples d'instructions.

Le Fortran
Structure du langage Fortran. Champ d'utilisation. Les instructions Fortran. Comparaison entre les instructions Fortran et les instructions Basic. Précis des instructions Fortran. Applications : procédure de facturation ; écriture des programmes. Traduction en Fortran de quelques programmes Basic.

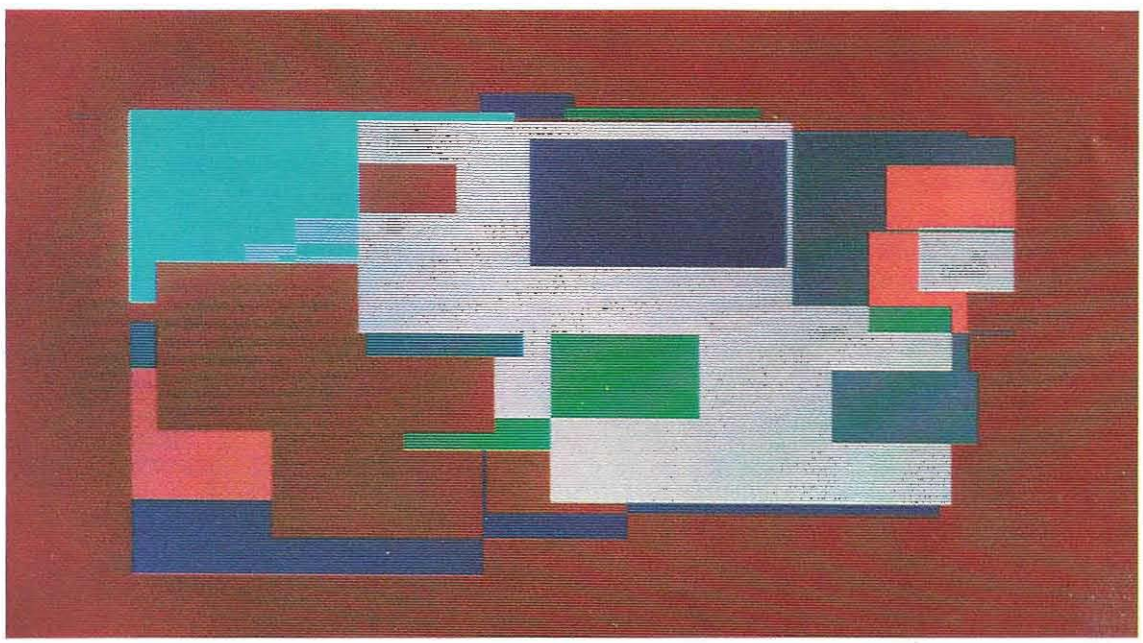
Programmes mixtes
Systèmes de programmation en plusieurs langages. Utilisation de programmes mixtes. Basic compilé : différences et avantages par rapport au Basic interprété. Systèmes d'exploitation : structure et fonctions. Principaux programmes utilitaires dans les systèmes d'exploitation. Systèmes d'exploitation en multiprogrammation. Application : programme de calcul des salaires.

L'infographie
Graphiques, histogrammes, échelles de représentation. Sous-programmes pour la préparation des histogrammes et des graphiques. Le vidéographique. Le traceur de courbes. Emploi de l'imprimante pour la préparation des histogrammes. Applications : écriture des sous-programmes graphiques ; écriture des sous-programmes pour la préparation des histogrammes. Représentation graphique de quelques fonctions.

Les applications spécifiques
L'avenir : développement des systèmes à microprocesseur. Schéma d'emploi des micro-ordinateurs dans les contrôles automatiques.

Glossaire général

Index analytique



Centre THC

PETIT GLOSSAIRE

ASCII (American Standard Code for Information Interchange). Code principal utilisé pour l'échange de données entre ordinateurs.

Bit (de l'angl. binary digit). Quantité minimale d'information : 0 ou 1.

Boucle (angl. loop). Partie de programme exécutée de manière répétitive, soit un nombre de fois prédéterminé, soit jusqu'à l'obtention du résultat attendu. Exemple : partie de programme lisant sans arrêt l'état du clavier dans l'attente de l'enfoncement d'une touche.

Bus. Ensembles de conducteurs électriques (pistes du circuit imprimé) permettant l'échange des informations.

Byte. Mot américain désignant un regroupement de 8 bits et utilisé dans le sens plus limité d'octet (nombre binaire de 8 chiffres).

Champ. Partie d'un enregistrement contenant une donnée ou une catégorie de données Un champ est numérique s'il ne contient que des chiffres, alphanumérique s'il contient aussi des lettres.

Circuit intégré. Circuit électronique rassemblant une grande quantité de composants sous un très petit volume et en mesure d'exécuter des fonctions très complexes

Clé. Information alphanumérique permettant de retrouver un ensemble de données On distingue les clés de recherche et les clés de protection.

Codes transparents. Les codes ASCII auxquels ne correspondent pas des caractères. On les utilise pour envoyer des commandes, ou des réponses à des commandes.

Compilateur. Programme utilitaire permettant de transformer un programme écrit dans un langage évolué (Basic généralement) en un programme dans un autre langage (machine) directement exécutable par le CPU (important gain de temps).

CPU (Central Processing Unit). En français : unité centrale (UC). Elle assure le traitement des informations C'est le circuit qui gouverne tout le fonctionnement d'un micro-ordinateur.

Curseur. Repère lumineux de l'écran, généralement matérialisé par un petit carré, à partir duquel la machine commence à émettre les messages et à accepter les données d'entrée Il marque l'emplacement dans lequel s'insère le caractère suivant.

Display. Affichage. Moyen visuel de communication avec la machine, généralement un écran, ou moniteur vidéo, mais aussi, par exemple, des afficheurs à cristaux liquides (micro-ordinateurs portables).

Disquette (angl. floppy disk). Nom courant du disque souple. Sa capacité de stockage est de l'ordre de 100000 à 1200000 caractères selon le type.

DMA (Direct Memory Access). Méthode de transfert des données avec accès direct à la mémoire.

Drapeau (angl. flag) Variable dont la valeur dépend de la vérification de situations déterminées. Elle sert à mémoriser une situation.

Enregistrement (angl. record). Caractérisé par son champ, un enregistrement est une partie d'un fichier. Par exemple, dans un fichier d'adresses, le code postal est un enregistrement numérique.

Fichier (angl. file). Zone de la mémoire de masse dans laquelle sont mémorisées des informations homogènes (archives, etc)

Hardware. Ensemble du matériel employé pour le traitement des informations.

Hold. Etat du CPU (UC) en attente de l'achèvement d'une opération.

Impulsion. Signal électrique de brève durée Une impulsion peut correspondre à un bit.

Interrupt. Interruption de programme provoquée par une cause externe reconnue et traitée par le CPU.

I/O (Input/Output). Fonction d'échange des informations entre un ordinateur et ses périphériques. Les termes français sont entrée et sortie (E/S)

Organigramme (angl. flowchart). Représentation graphique des opérations qui doivent s'effectuer dans un programme et de leur résolution.

Programme objet. Programme source ayant été assemblé, et difficilement modifiable, mais dont l'exécution par le CPU est immédiate.

Programme source. Programme écrit en langage symbolique qui ne sera utilisable par la machine que lorsqu'il aura été « assemblé » (traduit), devenant ainsi un programme objet, directement exécutable.

Puce (angl. chip). Petite pastille de matériau semi-conducteur (silicium le plus souvent) et portant un ou plusieurs circuits intégrés.

Saut (angl. jump). Instruction qui permet d'isoler, en la sautant, une partie de programme. Synonyme de rupture de séquence.

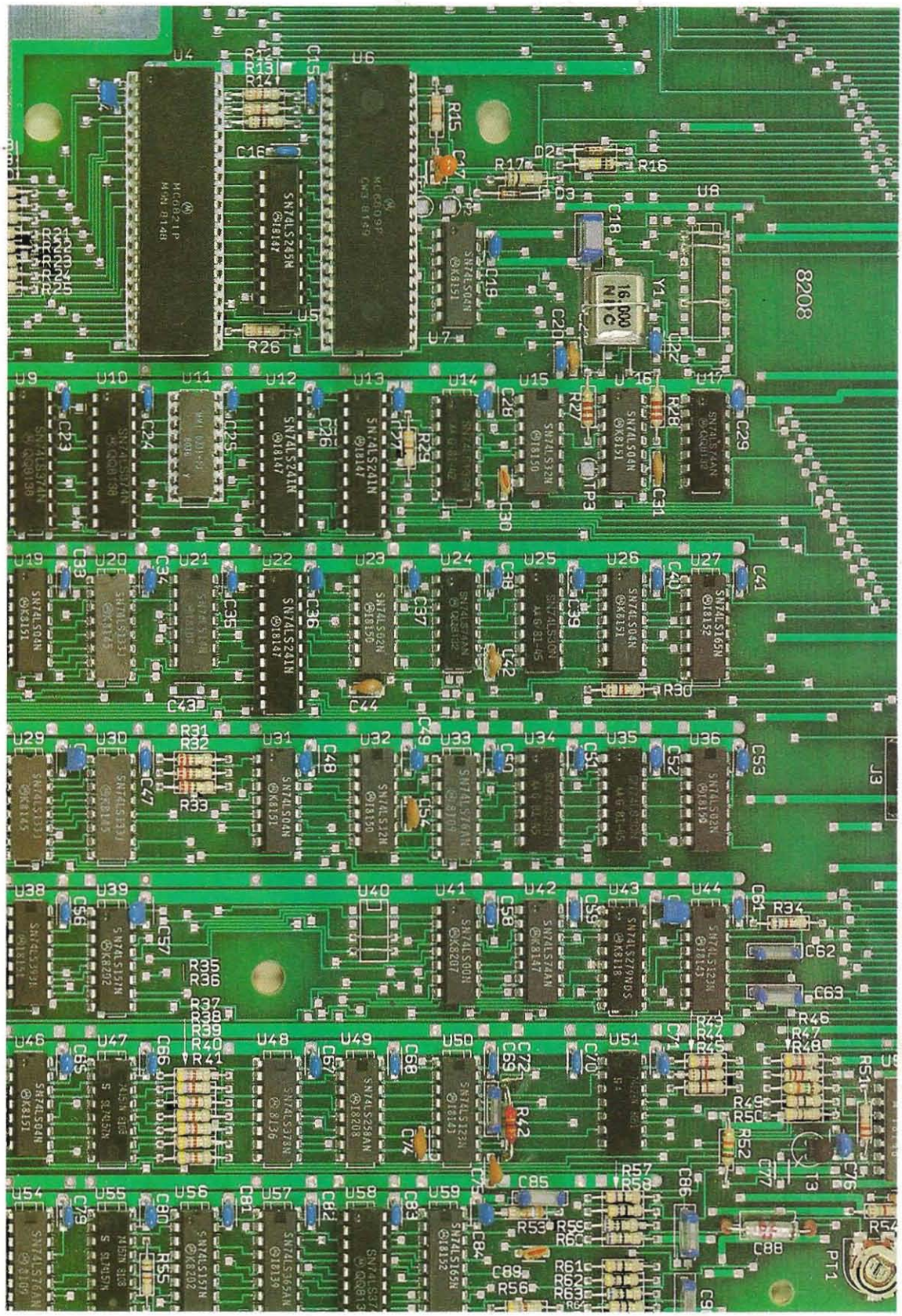
Segmentation. Technique avec laquelle sont divisés les programmes de grande masse en parties (segments) plus petites, plus faciles et plus rapides à utiliser et à mettre au point.

Software. Ensemble des programmes destinés au traitement par ordinateur. En français on parle de logiciel, par opposition à matériel (hardware)

Sous-programme (angl subroutine) Partie de programme qui exécute des fonctions spécifiques Il peut être utilisé à partir de n'importe quel point du programme principal.

Top-down. Analyse descendante. Technique de programmation qui consiste à développer l'analyse et à codifier des instructions, en subdivisant le problème en sous-problèmes plus simples et eux-mêmes décomposables.

Tri (angl. sorting). Programme particulier qui sert au classement des données. Les méthodes de tri sont nombreuses et présentent des difficultés et des intérêts divers.



Carte principale d'un micro-ordinateur T07 (D.R)

L'ordinateur, machine intelligente ?

Bien que de création relativement récente, les ordinateurs nous sont désormais assez familiers et il est peut-être bon de rappeler comment sont nés les premiers d'entre eux. Dérivant des calculatrices, les ancêtres de l'ordinateur ont d'abord été des appareils capables d'effectuer rapidement de très nombreux calculs. Mais ils n'étaient utilisés que par des chercheurs et restaient ignorés du grand public.

L'évolution rapide de l'électronique et la nécessité, pour l'industrie, de disposer de machines auxquelles on pouvait confier des opérations à caractère répétitif, ont donné un élan décisif au développement et à la commercialisation des ordinateurs.

Le rôle principal de ces premières machines était d'effectuer toutes les activités qui ne requièrent pas l'intervention de l'intelligence, mais qui présentent cependant une complexité nettement supérieure à l'exécution de calculs. C'étaient par conséquent des machines capables de traiter des informations.

En outre, l'ordinateur se différencie de la calculatrice par le fait qu'il est aussi capable d'« apprendre ». On peut en effet lui enseigner comment exécuter certaines tâches, c'est-à-dire qu'il peut être programmé. Cet « apprentissage » des premiers ordinateurs a, cependant, un caractère passif : la machine elle-même n'y contribue en aucune manière.

Ce type d'enseignement, la **programmation**, est assez proche des réflexes conditionnés.

L'intérieur d'un tel magasin fascinera toujours les passionnés d'informatique. Il leur offre, en effet, un large choix de documentation et de matériel.



Grazia Neri/Wheeler Pictures

Prenons un exemple. Nous pouvons dresser notre chien à aller chez le marchand de journaux, à s'y faire donner le journal et à nous l'apporter. Ces actions sont déterminées par des réflexes conditionnés acquis durant le dressage, et elles n'impliquent aucune contribution de l'intelligence. Si le chien trouve le kiosque fermé, ou si le journal n'a pas paru, il ne prendra pas de décision.
Le même concept s'applique au « dressage » (programmation) des ordinateurs. Ces derniers peuvent exécuter toutes les fonctions pour lesquelles ils ont été programmés, mais dès qu'apparaît un événement imprévu, l'intervention de l'homme est nécessaire.
Pourtant, certaines recherches en cours pourraient permettre de construire des appareils dotés d'une forme de « conscience ». Aujourd'hui, cette possibilité appartient encore à la science-fiction, bien que certains ordinateurs plus évolués soient déjà capables de détecter les erreurs, voire de diagnostiquer des pannes et d'isoler les circuits défectueux.
Les capacités actuelles des ordinateurs et leur développement prévisible nous les font apparaître comme des machines extrêmement sophistiquées. Pourtant les ordinateurs peuvent être utilisés par tous. C'est surtout vrai pour les petits systèmes comme les ordinateurs personnels qui, malgré leur complexité, remplacent de plus en plus les calculatrices.
L'emploi de ces dernières est désormais aussi indispensable aux ingénieurs et aux commerçants qu'aux élèves, qui trouvent normal de confier les calculs à la machine, réservant le temps ainsi libéré aux activités qui font appel à leur capacité de raisonner.
Pour faire un devoir de mathématique, l'élève doit d'abord poser le problème, puis en exécuter les calculs. Cette dernière partie du travail n'a rien à voir avec la rationalisation du problème et la solution apportée ; elle peut donc être confiée à la machine.
Si, par exemple, un élève veut calculer l'hypoténuse d'un triangle rectangle dont il connaît les côtés, il doit recourir au théorème de Pythagore qu'il posera ainsi :

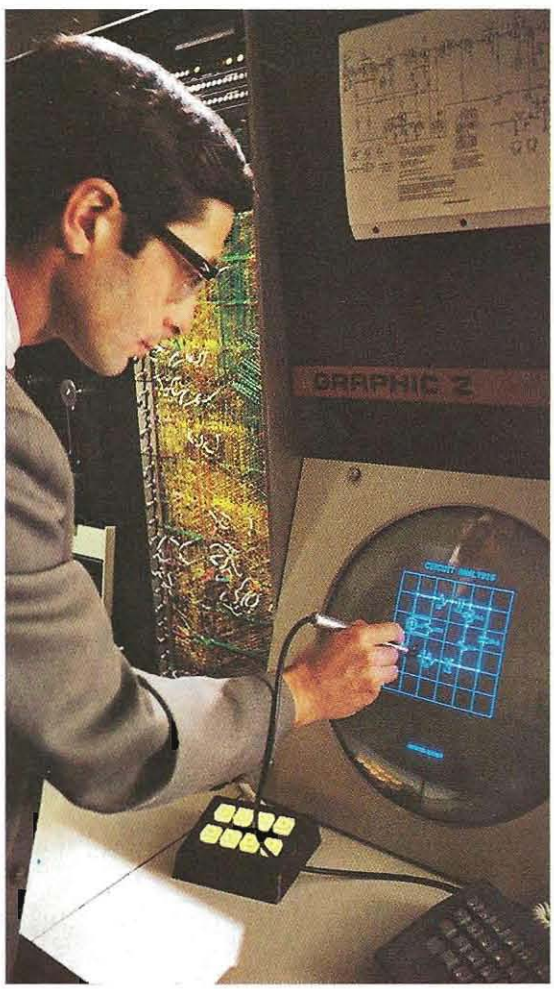
a) hypoténuse $= \sqrt{(\text{premier côté})^2 + (\text{second côté})^2}$.

Si les côtés valent 8 (premier côté) et 4 (second côté), la formule, obtenue à partir de la précédente en remplaçant les lettres par les valeurs numériques, est :

b) hypoténuse $= \sqrt{8^2 + 4^2} = \sqrt{64 + 16} = \sqrt{80} \cong 8{,}9$.

Pour trouver la valeur de l'hypoténuse et terminer l'exercice, l'élève devra extraire la racine carrée d'un nombre. La génération précédente lisait le résultat sur les tables. Aujourd'hui ce résultat est fourni par la calculatrice, et l'élève de demain le demandera à l'ordinateur. Voyons pourquoi. La solution du problème est dans la formule a), le stade suivant b) étant nécessaire pour trouver la valeur numérique. L'élève qui a effectué la première partie a prouvé qu'il est capable de poser le problème et donc de le résoudre ; la phase suivante est seulement un achèvement ennuyeux et peu significatif : on peut le déléguer à la machine. S'il existait un appareil

Le crayon optique sert à enregistrer directement des informations en les traçant sur l'écran.



Marka



Marka

L'enseignement assisté par l'ordinateur permet à chaque étudiant de travailler à son rythme.

auquel on puisse donner ces cinq instructions : « prends la valeur du premier côté et mets-la au carré », « prends la valeur du second côté et mets-la au carré », « additionne les deux carrés », « extrais la racine de la somme obtenue », « donne-moi le résultat », l'élève aurait, non seulement fini son travail après la première phrase, mais il aurait aussi préparé la réponse pour tous les autres problèmes appliquant le théorème de Pythagore.

Les données

Une telle machine existe : c'est l'ordinateur, auquel nous pouvons « enseigner », en le programmant, comment résoudre le théorème de Pythagore. Notre rôle consiste à lui fournir, la première fois, toutes les indications nécessaires au calcul. Les fois suivantes nous n'aurons plus qu'à lui donner les nouvelles valeurs des côtés, et la machine, connaissant déjà les opérations à accomplir, pourra fournir rapidement le résultat.

Quand nous effectuons une addition, nous connaissons les deux termes et nous cherchons le résultat. Les deux nombres à additionner sont les données du calcul.

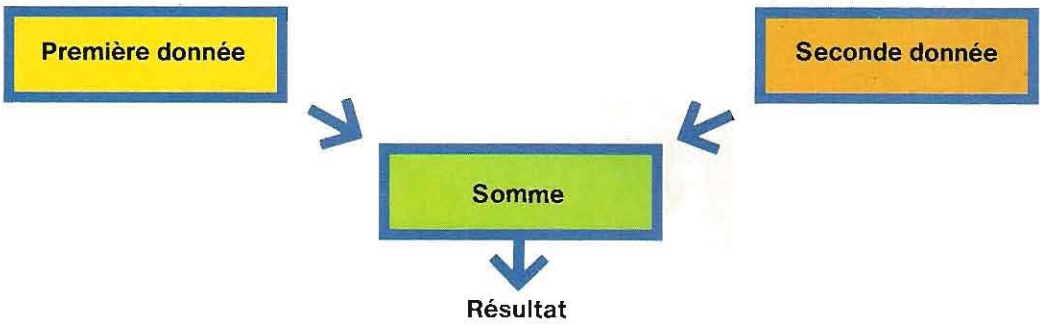
Si nous cherchons un numéro de téléphone sur l'annuaire, nous devons connaître un nom, un prénom et une adresse. Nous utilisons ces informations comme données pour retrouver le numéro de téléphone. Dans ce cas, l'action (recherche du numéro) se termine avec la lecture du numéro de téléphone lui-même ; celui-ci aussi est une donnée, puisqu'il est déjà contenu dans l'annuaire.

Dans le premier cas (calcul) on obtient une valeur (résultat) qui n'existait pas ; mais dans le second (recherche) on extrait une valeur déjà existante (numéro de téléphone). Ces deux méthodes d'utilisation des données sont schématisées page 16. Avec les ordinateurs, il convient de suivre la même logique. Pour exécuter un calcul il faut enregistrer toutes les valeurs numériques des données du problème ; dans une recherche il faut également connaître l'ensemble des données sur lequel s'effectue la sélection.

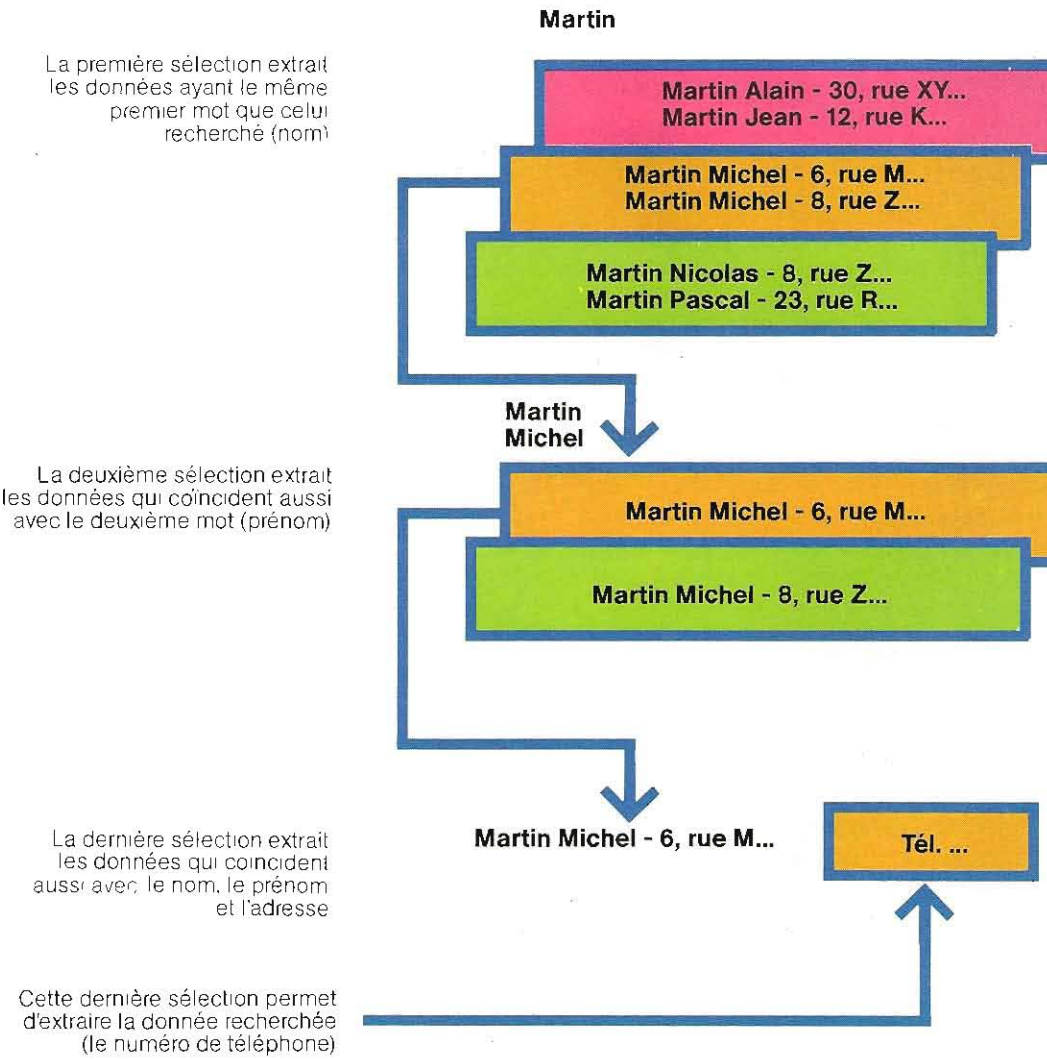
Lorsque nous parlons avec quelqu'un, nous procédons à un échange d'informations. Ces données (informations) proviennent de notre

EMPLOI DES DONNEES DE L'ORDINATEUR

Les données sont utilisées pour fournir un résultat



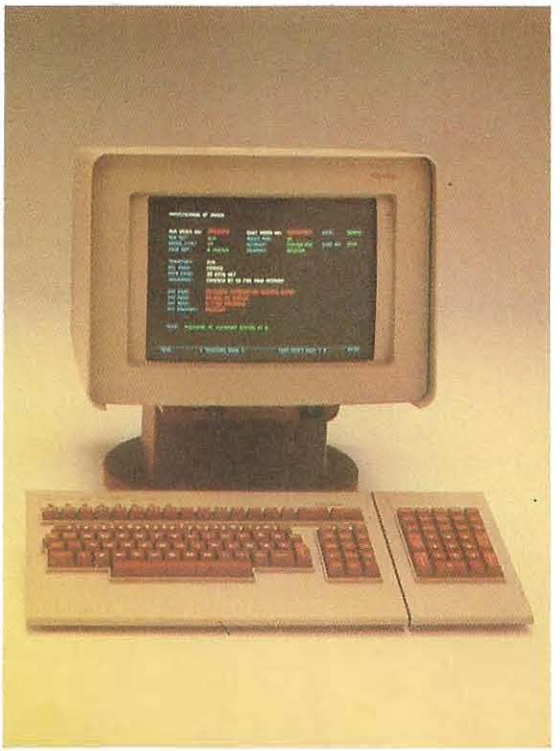
Les données sont utilisées pour opérer des choix (sélections)



mémoire, quand, par exemple, nous relatons un fait ; elles peuvent être le résultat d'une élaboration, si nous exprimons une opinion.
Pour relater un fait ou exprimer un avis, il faut en connaître les circonstances. De même, **l'ordinateur, pour fournir une réponse, doit posséder toutes les données.**
Par exemple, en regardant de l'eau qui bout, nous concluons que, si nous la touchions, elle nous brûlerait. Ce processus mental est possible si on nous a d'abord prévenus du danger et si nous avons emmagasiné l'information. Le raisonnement suivi est simple. Il associe deux données : eau qui bout et risque de brûlure.
Le même processus s'applique à l'ordinateur : pour commencer, il faut lui fournir (en « entrée ») toutes les données et les éventuelles règles d'association ; par la suite, au moment de faire une vérification, la machine sélectionnera parmi toutes les réponses possibles la bonne réponse et elle la présentera en « sortie ».
Cependant, nos possibilités d'emmagasiner des données dans notre cerveau ne se limitent pas à des échanges : nous apprenons aussi par nos expériences. Revenons à l'exemple de l'eau qui bout. Si aucun avertissement ne nous avait été donné, l'information aurait pu provenir d'une situation vécue. La première fois, nous aurions touché le récipient et nos récepteurs (nos terminaisons nerveuses) auraient envoyé des sensations de douleur à notre cerveau, qui aurait catalogué le danger. De cette mémorisation aurait résulté notre comportement futur en présence d'une situation analogue. Cette méthode d'apprentissage, qui nous enrichit constamment de nouvelles données, a été également employée avec les ordinateurs.
C'est une application qui donne d'excellents résultats et laisse entrevoir de remarquables perspectives.
Dans les exemples présentés sont contenues les deux formes possibles de raisonnement :

- **conception déterministe**
- **conception probabiliste**

La conception déterministe consiste à associer cause et effet au moyen d'une grande quantité de données, préalablement introduites dans l'ordinateur. Autrement dit, toutes les situations possibles doivent être prévues.



Ericsson

Ordinateur personnel haut de gamme doté d'un clavier ergonomique et d'un écran couleur.

Dans la conception probabiliste, la logique est plus complexe et sera mieux expliquée avec un exemple.
Supposons que nous voulons apprendre à l'ordinateur à jouer aux échecs.
Selon la méthode déterministe, nous devrons stocker dans sa mémoire une réponse à chaque situation possible. Un processus de ce genre, outre qu'il serait fort laborieux, serait très certainement incomplet.
L'alternative probabiliste consiste à lui faire mémoriser les règles selon lesquelles les figures peuvent être déplacées et à lui faire jouer un grand nombre de parties.
Durant chaque partie, si un coup (d'abord joué par hasard) donne de bons résultats, l'ordinateur mémorise la situation, et le coup réalisé est classé comme valide. Si, au contraire, le résultat est négatif, la situation est classée comme à éviter.
Naturellement, la machine perdra toutes les premières parties, mais augmentera peu à peu son expérience et donc le nombre de situations qu'elle peut identifier. De ce fait, il deviendra de plus en plus difficile de la battre. C'est ainsi que l'ordinateur peut « apprendre ».

L'ordinateur peut-il penser ?

Le savant américain Robert Jastrow, faisant un rapprochement entre les différentes caractéristiques du cerveau humain et de l'ordinateur, dans son livre *The Enchanted Loom*, a formulé quelques théories sur la capacité de « penser » de l'ordinateur, et de fascinantes hypothèses sur ses développements futurs.

Peu de personnes seraient disposées à nier la supériorité de l'ordinateur en ce qui concerne l'arithmétique et les calculs mathématiques en général, mais elles admettraient difficilement qu'un ordinateur puisse faire plus. Comment une machine peut-elle penser ou raisonner ? La chose paraît impossible, mais un intéressant programme d'ordinateur, exposé par A.L. Samuel d'IBM, illustre comment cela peut arriver. Le programme qui enseigne à un ordinateur le jeu d'échecs démontre que la machine est capable d'exprimer des raisonnements étonnamment proches des raisonnements humains, si elle a reçu la formation appropriée.

Samuel commença à dresser son ordinateur en lui enseignant les règles des échecs, qu'il emmagasinait en mémoire. La formule qu'il proposa analysait la position des pièces séparément, puis elle en additionnait les points forts et les points faibles, tels que le nombre d'occasions de prendre un pion de l'adversaire et les pièges éventuels dans lesquels les siens pouvaient tomber. La formule comprenait aussi des jugements plus subtils ; elle conseillait les coups qui permettaient l'échange d'un pion, dans le cas, bien sûr, où la machine avait l'avantage. Si, au contraire, elle était en difficulté, la formule déconseillait l'échange. N'importe quel joueur humain suit la même stratégie. Enfin, la formule exprimait la force de la position de la machine à l'aide d'un nombre : plus élevé était le nombre, plus forte était la position. La formule était pour la machine l'équivalent de l'expérience d'un joueur humain qui, en général, fonde sa stratégie sur un ensemble de règles empiriques issues de son expérience. N'importe qui, tenu de faire des choix entre diverses options – le conseiller financier ou le chef d'entreprise – affronte le problème de la même manière.

Après avoir enseigné à la machine les bases de la stratégie du jeu d'échecs, Samuel la programma pour qu'elle améliore progressivement sa tactique au cours du jeu. Il lui conféra ainsi la capacité d'apprendre à partir de l'expérience. Pour l'ordinateur cela impliquait une mise à jour permanente des données de la formule après chaque coup. Comment la machine pouvait-elle dépasser son maître en sagesse ? En comparant la valeur qu'on attendait d'un coup à celle effectivement totalisée après la réponse de l'adversaire. La différence consistait naturellement dans le fait que l'adversaire pouvait aussi ne pas jouer le coup que la machine lui avait déjà attribué. La comparaison disait à quels facteurs de la formule on devait attribuer un poids plus ou moins grand. La machine remplaçait ces « poids » dans la formule immédiatement après chaque coup, changeant chaque fois la formule et améliorant ainsi sa stratégie de jeu. Exactement comme un être humain, la machine apprenait sur le terrain.

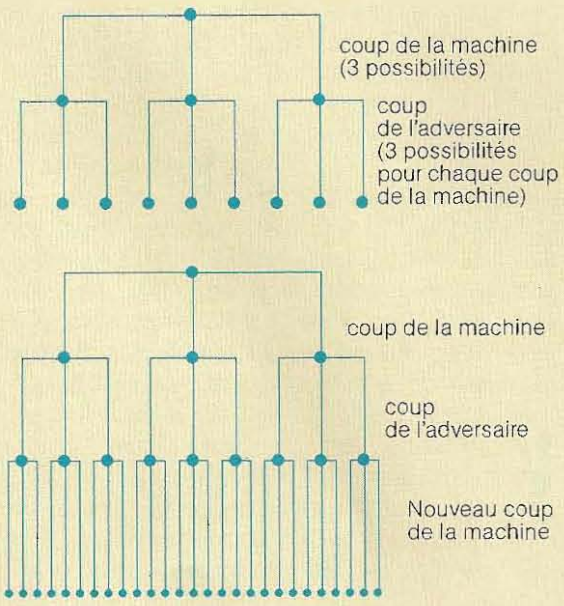
Comment jouer contre un ordinateur ? Au début de la partie, on dispose l'échiquier avec les pièces du jeu, qu'un assistant déplacera à chaque coup, ou bien échiquier et pièces apparaissent sur un écran de télévision. Tout cela est au bénéfice du joueur humain, car la machine n'a pas besoin de regarder l'échiquier puisque sa mémoire électronique a enregistré toutes les positions et tous les échanges du jeu.

Les deux adversaires jouent maintenant l'un après l'autre. Le joueur humain communique ses propres coups à la machine en les tapant sur un clavier. Quand la machine décide de la réponse, elle fait clignoter le sigle correspondant sur l'écran de télévision ou l'imprime sur du papier. Se référant constamment à sa formule, la machine élabore chaque coup à la manière d'un être humain ; elle cherche à prévoir les développements du jeu aussi loin que possible, ainsi que les mérites relatifs de chaque coup éventuel, considérant que l'adversaire répondra avec le meilleur coup possible pour lui. Ce qui est aussi ce que ferait un être humain.

Le style de l'ordinateur dans le jeu d'échecs est en fait très semblable à celui d'un débutant qui calcule ses coups : « Si je bouge cette pièce, il bougera cette autre. Si je bouge cette autre, il bougera... ». La différence entre le

novice et le joueur expérimenté est que ce dernier est familiarisé avec beaucoup de situations de jeu et avec les réactions probables de l'adversaire : il peut prévoir le résultat d'un coup très à l'avance, sans élaborer consciemment les coups intermédiaires. A ce point, un observateur extérieur dirait que le bon joueur a un sens intuitif des positions sur l'échiquier. En réalité, il élabore les coups de la même façon que l'ordinateur, mais la pensée se déroule au niveau inconscient.

Comment «pense» l'ordinateur ? Supposons qu'à un certain point du jeu la machine ait 3 coups possibles. Elle explore chaque possibilité, calcule la valeur d'un coup particulier à l'aide de la formule, puis cherche à établir la réponse la plus probable de l'adversaire. Supposons que l'adversaire aussi ait 3 réponses possibles pour chaque coup de l'ordinateur. L'ordinateur devra donc considérer un total de 9 coups. Le schéma suivant illustre les alternatives qui, à ce stade de la partie, s'offrent à l'ordinateur. Chaque bifurcation correspond à une alternative. Le diagramme porte le nom d'«arbre des décisions». Anticipant les coups, la machine voit maintenant neuf possibilités. Elle en tient compte grâce à sa mémoire électronique, et élabore pour chacune le coup suivant. Ce qui revient à ajouter une autre ramification à l'arbre. Il présente alors 3 possibilités pour chacune des 9 précédentes, soit 27 branches.



La phase ultérieure de ce processus porterait à 81 le nombre des possibilités, ce qui signifie que la machine devrait avoir en mémoire 81 échiquiers. Cela commence à devenir difficile, même pour un ordinateur, et surtout pour les anciens modèles dont disposait Samuel dans les années 50. Le programme de Samuel s'arrêtait là.

Aujourd'hui les machines ont une rapidité mille fois supérieure, et sont capables de prévoir un bien plus grand nombre de coups ; elles jouent donc beaucoup mieux aux échecs.

La subtilité de la stratégie du jeu confère à l'ordinateur une qualité pratiquement «biologique». Après avoir fait une partie d'échecs contre un ordinateur considéré comme le meilleur joueur «artificiel» mondial, le champion écossais David Levy – maître international et l'un des 500 meilleurs joueurs du monde – déclara : «on a tendance à considérer ces ordinateurs comme des personnes, surtout quand on voit qu'ils ont compris ce que vous faites et qu'ils cherchent à faire quelque chose d'intelligent». Certains spécialistes pensent d'ailleurs que le champion du monde des échecs 1990 ne sera pas un être humain.

Bien entendu, le jeu d'échecs est un jeu très abstrait. La machine pourvue d'une logique adaptée aux échecs ne possède pas l'inspiration créatrice qui caractérise les plus hautes sphères de la pensée humaine. A l'égal d'un jeune diplômé, elle a de prodigieuses capacités intellectuelles, mais il lui manque peut-être encore la sagesse. Et pourtant, cette supériorité du cerveau sur l'ordinateur est déjà remise en question par les progrès techniques les plus récents.

Ces nouveaux développements dépendent aussi de l'augmentation prodigieuse du nombre des composants que l'on peut regrouper dans un circuit intégré. Les circuits intégrés sont comme les neurones du cerveau humain ; très nombreux et tous reliés entre eux, ils constituent le «cerveau électronique moderne». Vers 1970, une «puce» (chip) ne pouvait recevoir qu'un nombre restreint de transistors et de composants électroniques. Par conséquent, elle pouvait penser ou se souvenir, mais pas les deux à la fois. Aujourd'hui les meilleurs fabricants américains et japonais de circuits intégrés savent



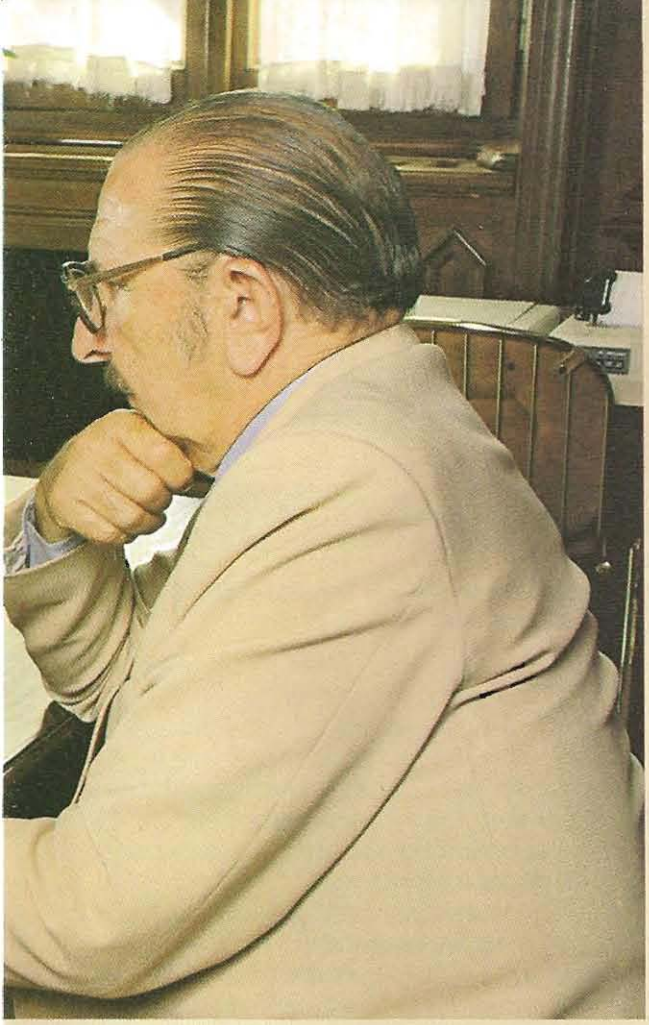
Dominique Fradin

L'ordinateur personnel joue aux échecs, mais le niveau du jeu est réglable.

regrouper tant de composants électroniques dans un circuit que, pour la première fois, il est possible de réunir sur une seule puce des circuits pensants et des unités de mémoire. Il en résulte qu'un tel circuit peut à la fois se souvenir, raisonner et prendre des décisions.

Cette combinaison de fonctions est extrêmement importante dans la mesure où, par exemple, un circuit de mémoire capable de penser peut être relié à des circuits voisins auxquels il pourra envoyer des instructions et qui lui en enverront. C'est pourquoi, quand on envoie une instruction aux unités de mémoire de l'ordinateur pour en extraire une information – le nom d'une personne par exemple – stockée dans un circuit déterminé, les circuits de mémoire directement concernés peuvent transmettre aux circuits voisins des informations connexes et offrir à l'utilisateur une plus grande quantité d'informations que ce qu'il en attendait.

On se rapproche ici du type de mémoire associative formée dans le cortex cérébral et qui constitue un puissant élément du raisonnement humain. Le cerveau répond à une simple demande d'information par une vaste quantité d'éléments, tous liés à l'information demandée par des associations qui puisent dans l'expérience passée. La pensée humaine est grandement facilitée par la richesse de la réponse du cerveau à de telles demandes, réponses qui se développent au niveau des milliers de connexions des cellules cérébrales. Un ordinateur qui présenterait les mêmes circuits, chacun relié à un grand nombre d'autres circuits, serait un vrai cerveau au silicium. Comme le cerveau humain, il fonctionnerait à la manière d'une unité indépendante, opérant au niveau inconscient. Les dernières améliorations techniques apportées aux ordinateurs rendent possible un autre type de pensée créatrice. La pensée de l'ordinateur avait tendance à être de type mécanique et dénuée d'imagination. C'est que les circuits d'un ordinateur sont fort simples par rapport à ceux du cerveau humain. Chaque circuit d'ordinateur n'est lié qu'à deux ou trois fils d'arrivée des autres parties de la



machine, tandis qu'une cellule microscopique du cerveau reçoit du reste du cerveau des dizaines de milliers de fibres nerveuses. Les innombrables liaisons d'une cellule à l'autre, combinées aux caractéristiques subtiles des circuits du cerveau, expliquent en grande partie l'extraordinaire puissance de cet organe.

En théorie, les ordinateurs auraient pu être construits depuis longtemps avec des circuits à entrées variables, exactement comme le cerveau humain. Cependant, même petit, un ordinateur de ce type aurait nécessité des centaines de milliers de fils distincts pour les connexions de circuit à circuit, ce qui serait impossible à construire.

Avec les nouveaux circuits intégrés, toutefois, le discours change. Les fils sont absents et les connexions – microscopiques – font partie de la structure même du circuit. Cette innovation, qui a l'apparence d'un simple progrès technologique, est au contraire une phase fondamentale dans l'évolution de l'ordinateur. Elle permet en effet la construction de machines dotées d'un fonctionnement semblable à celui du cerveau humain. Si l'on en croit les prévisions, de tels ordinateurs verront le jour dans les années 90. A maints égards, ils rivaliseront avec l'esprit humain et auront beaucoup des attributs d'une vie intelligente ; une réceptivité au monde extérieur, la capacité d'apprendre par l'expérience, une prompte compréhension des idées nouvelles.

La plupart des gens diront qu'un ordinateur ne sera jamais un organisme vivant, parce qu'il n'a ni sentiments, ni émotions. Il ne mange pas, ne bouge pas, ne grandit pas ; et il est fait de métal et de plastique, au lieu de chair et d'os. On pourrait pourtant lui ajouter des roues et un moteur. Et cet ordinateur sur roulettes pourrait être programmé de façon à se diriger vers une prise de courant et à s'y relier pour avaler une « gorgée » d'électricité dans le cas où ses batteries seraient déchargées et où ses voltmètres signaleraient les tiraillements de la faim. Il est vrai qu'il faudrait quelqu'un pour lui procurer l'électricité, mais la plupart des gens n'assurent pas eux-mêmes leur alimentation. D'ailleurs, si c'était nécessaire, il ne serait pas difficile de construire un ordinateur capable de dépister l'électricité et de se l'approprier de lui-même.

Même les sentiments et les émotions peuvent être, si nécessaire, intégrés à un ordinateur, exactement comme la nature les a intégrés dans les parties les plus antiques du cerveau humain pour assurer la survie de l'homme. Samuel fit cette découverte alors qu'il cherchait à pousser son ordinateur à apprendre plus rapidement et à se montrer meilleur élève. La machine jouait assez bien pour un débutant, mais elle ne se battait pas assez. Au lieu de pousser à une rapide victoire quand elle avait le dessus, comme aurait fait un joueur humain, elle s'attardait sur toutes les alternatives possibles. Samuel décida d'inverser sa psychologie. Il modifia le programme de façon que, lorsque la machine avait l'avantage, elle devenait agressive, choisissant le coup qui devait lui donner la victoire le plus rapidement possible. Quand elle était dans une mauvaise position, elle adoptait des tactiques dilatoires, et choisissait le coup qui faisait durer le jeu le plus longtemps possible.

(D'après THE ENCHANTED LOOM, de Robert Jastrow. © 1981 by Reader's Library, Inc. ; 1982 Editions Mazarine pour l'édition française, AU-DELA DU CERVEAU.)

De la calculatrice à l'ordinateur : le programme

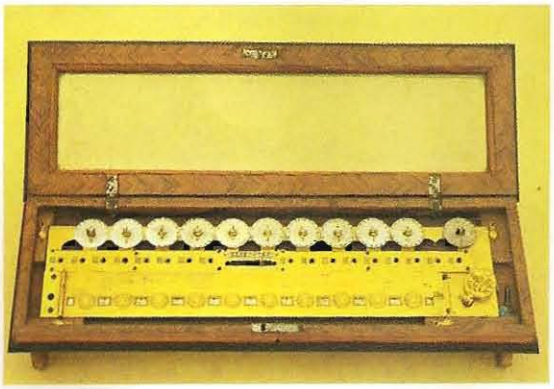
La première machine construite pour effectuer des calculs fut d'abord une calculatrice mécanique qui exécutait des additions et des soustractions. Elle a ensuite été perfectionnée, devenant électromécanique et acquérant la capacité de faire les quatre opérations. La limite essentielle de cette machine est qu'elle contraint à poser chaque fois l'opération à effectuer et qu'on ne peut pas lui donner d'« instructions » quant au travail à réaliser. Au contraire, les ordinateurs peuvent être « formés », c'est-à-dire programmés pour exécuter certaines tâches.

L'exemple qui vient immédiatement à l'esprit est celui, bien connu, de la machine à laver. Dans cet appareil électroménager, est monté un système qui, en fonction du temps passé et de la température de l'eau, provoque les différentes phases du lavage. L'utilisateur peut choisir entre divers types de lavage (plus ou moins longs, à température plus ou moins élevée), c'est-à-dire entre divers types de programmes.

Cette machine a donc, elle aussi, reçu des données (le temps et la température de l'eau) ; selon le programme choisi, c'est le traitement de ces données qui déclenche les cycles de lavage.

Le fonctionnement d'une machine programmable peut être ainsi schématisé :





Musée d'Histoire des Sciences, Florence

Machine arithmétique inventée en 1666 par Samuel Morland.

Passé et présent

Le désir de se débarrasser des calculs, surtout s'ils sont longs et ennuyeux, pour consacrer plus de temps à la réflexion, préoccupe l'homme depuis l'Antiquité.

Les premiers instruments d'aide au calcul – les bouliers – apparaissent en Chine vers le VIIIe siècle avant J.-C. Les Romains, en revanche, utilisaient l'abaque, planchette portant des rainures horizontales dans lesquelles on pouvait déplacer des sortes de rondelles. La première rainure représentait les unités, la deuxième les dizaines, etc. ; c'était là une autre forme de boulier. Le boulier est encore en usage aujourd'hui et des opérateurs particulièrement habiles sont capables de s'en servir pour extraire des racines carrées. Ce fut d'ailleurs pendant longtemps le seul moyen disponible.

En 1642, Pascal invente une machine arithmétique à engrenages. Le perfectionnement de cet appareil permettra de construire la première machine à faire les multiplications. Cette machine multipliait en procédant par additions successives.

Toutes ces machines restèrent toutefois des curiosités jusqu'en 1850 environ. On construisit alors des machines à additionner plus simples et plus fiables, qui eurent enfin une utilisation pratique.

A la fin du siècle (1889) fut fabriquée une machine à faire les divisions. Elle contenait une table de Pythagore et effectuait la division en sélectionnant le résultat dans la table.

Toutes ces machines, qui constituaient pourtant les fondements du calcul mécanique,

n'exécutaient qu'une seule opération à la fois et n'avaient aucune forme de mémoire. La première machine capable de faire plusieurs opérations, et dotée d'une mémoire, fut conçue en Angleterre en 1821 par le mathématicien Charles Babbage.

Sur le principe de fonctionnement conçu par Babbage, on fabriqua par la suite des machines plus évoluées, parfois même dotées d'imprimantes.

En 1929, la première application pour le calcul scientifique fut conçue aux Etats-Unis. Le fonctionnement mécanique, lent et lourd à manipuler, fut ensuite remplacé par des mécanismes électromagnétiques actionnés par des dispositifs électriques.

Cette innovation conduisit à la fabrication, vers 1944, du Mark 1, un appareil qui, pour l'époque, avait une potentialité énorme : dix opérations à la seconde. Ses dimensions étaient remarquables aussi : 18 mètres de long sur 2,50 mètres de hauteur ! A peine trois ans plus tard, fut construite la première machine à tubes électroniques. Les opérations étaient effectuées par des circuits électroniques, mais le principe de fonctionnement était toujours celui de Babbage.

La notion de programme apparut vers 1950 : on fournissait à la machine toutes les indications nécessaires aux calculs et elle les effectuait ensuite sans autre intervention de l'opérateur. Le premier ordinateur était né.

Depuis, ce secteur a connu un développement vertigineux, qu'on n'a observé dans aucun autre domaine scientifique.

Les tubes électroniques ont été remplacés par des *transistors,* capables de remplir les mêmes fonctions avec un encombrement des dizaines de fois moindre et avec peu de dissipation de chaleur. Le progrès suivant a été l'introduction des *circuits intégrés* (entre 1965 et 1970). Cette nouvelle technologie a permis la construction d'ordinateurs aux dimensions très réduites et dotés d'une grande vitesse de calcul.

Les calculateurs modernes peuvent effectuer un million d'opérations à la seconde contre une dizaine seulement pour Mark 1 !

Depuis 1970, d'énormes progrès ont été réalisés dans le domaine de la miniaturisation des composants et des circuits. Ils ont été essentiels pour la naissance des micro-ordinateurs.

Le tableau suivant résume les principales étapes de cette évolution.

VIIIe s. av. J.-C.	Bouliers
1642	Mach. à additionner
1850	Mach. à additionner dotées de mémoire
1889	Mach. à diviser
1929	Calcul scientifique Imprimantes
1944	Mark 1. Dispositifs électromagnétiques
1950	Programmation
1955	Transistors
1965-1970	Circuits intégrés
1970-1980	Grands progrès de la miniaturisation

Centre de calcul moderne organisé autour d'un ordinateur de grande puissance.



Intema

Le tisseur de nombres : histoire d'une invention

Le XVIIIe siècle, riche en bouleversements économiques, avait été relativement pauvre en découvertes scientifiques. Dans la dernière décennie, toutefois, naquit (en 1792) Charles Babbage, généralement considéré comme le père de l'ordinateur moderne.

Babbage se penchait depuis longtemps sur des problèmes de calculs et sur la compilation de données. L'une des tâches les plus ardues et sujettes à erreurs était la compilation des tables de logarithmes : Babbage continuait d'y trouver des erreurs banales. Pensant justement à la nature répétitive de ces calculs et au gaspillage de temps, Babbage comprit qu'une machine devrait être capable d'effectuer ce type d'opérations et le schéma de base d'un tel appareil se forma très vite dans son esprit.

En 1821 il se sentit assez sûr de son fait pour annoncer à la Société Royale Astronomique qu'il avait construit un prototype. Lors d'une démonstration, il expliqua que sa machine fonctionnait selon « la méthode des différences » et il fournit des détails très convaincants sur son fonctionnement.

Nous n'allons pas expliquer comment marchait la machine : qu'il suffise de dire qu'elle était en mesure de résoudre des équations de polynômes en calculant les différences successives entre ensembles de nombres. Il la montra à une assemblée de la Société en 1822 ; la présentation fut si bien acceptée que la médaille d'or de la Société fut attribuée à son écrit « Observations sur les applications des machines au calcul des tables mathématiques ».

Désormais enthousiaste, Babbage se mit à construire la version finale de la machine. Même le coup d'œil le plus superficiel aux plans de Babbage révèle combien ils étaient audacieux. Les principes fondamentaux du calcul, les petites roues et les mécanismes n'avaient pourtant rien de particulièrement nouveau. La machine était cependant beaucoup plus grosse et beaucoup plus compliquée que le prototype ; en outre, avec son optimisme, Babbage avait prévu qu'elle ne devait pas seulement calculer des tables, mais aussi les imprimer sur papier. Il demanda une subvention au gouvernement britannique qui lui accorda 1 500 livres sterling. Il fit construire une usine sur les terrains des Babbage, il embaucha d'habiles ouvriers et les travaux commencèrent.

La tâche principale était de créer, sur des tours spéciaux, avec la tolérance la plus étroite possible, les centaines d'arbres, de roues et d'engrenages qui devaient constituer les parties fonctionnantes de la machine à différences. Et c'est là qu'apparurent les premiers grands obstacles. Le modèle construit pour la Société Astronomique était seulement un prototype. Les très petites irrégularités de ses composants provoquaient bien une certaine instabilité du système, mais pas au point, cependant, de causer des arrêts ou de bloquer le fonctionnement global. Au contraire, dans la machine à différences, des séries de petites imperfections tendaient à s'additionner, provoquant de grandes vibrations et des grippages.

Babbage poussa ses ouvriers à travailler avec une meilleure précision. Une amélioration s'ensuivit, mais pas suffisamment appropriée à la complexité globale du système. Insensible aux premiers échecs, Babbage s'entêta, sans réaliser pourtant que c'était demander aux machines de l'époque des performances qu'elles ne pouvaient pas fournir.

Il demanda de plus en plus d'argent au gouvernement. On lui accorda jusqu'à 17 000 livres, puis il fut décidé que c'en était trop et le projet fut suspendu en 1833.

Si Babbage avait été une personne raisonnable, ce qu'il n'était pas, cela aurait été le moment de s'arrêter. Au contraire, son esprit infatigable commença à se consacrer à un schéma encore plus ambitieux : c'est à ce moment que naquit le concept d'ordinateur.

Bien que la machine à différences ait constitué un net progrès sur celles qui avaient été construites auparavant, elle n'était capable d'accomplir qu'une seule tâche : résoudre des équations de polynômes. Le résultat était obtenu à travers une série de mouvements convenus : un ensemble de petites roues en faisait tourner un autre, un levier en levait un autre et ainsi de suite, en une séquence plus ou moins prévisible.

En d'autres termes, la machine n'était capable d'exécuter que l'unique type de fonction pour laquelle elle avait été fabriquée.

En y réfléchissant bien, Babbage se rendit compte qu'il faisait fausse route. Il se dit qu'une machine capable d'effectuer un type de calculs devrait être aussi capable d'effectuer tous les autres calculs. Le bien-fondé de cette idée allait être mathématiquement démontré pratiquement un siècle plus tard par un autre génie anglais : Alan Turing.

Pourquoi ne pas concevoir une machine en la déterminant à l'avance, non pas pour une tâche définie, mais pour plusieurs tâches, afin que son propriétaire, au moment où il le voudrait, puisse lui en demander l'exécution. Une telle machine risquait d'être plus compliquée, et de coûter encore plus cher à construire, mais elle serait aussi beaucoup plus utile. Sans doute, il pouvait paraître évident de voir la nouvelle machine comme un ensemble de plusieurs autres, possédant un seul mécanisme d'entrée et une connexion commune à l'imprimante ou à tout autre type de sortie.

Babbage refusa ce qui venait d'abord à l'idée de tout le monde, et il aboutit à une intuition extraordinaire. Son projet devait permettre d'utiliser les organes internes de la machine de multiples façons. De cette manière, devant une tâche particulière, une seule séquence aurait été exécutée, et chaque tâche différente aurait été liée à des modèles uniques d'activité interne.

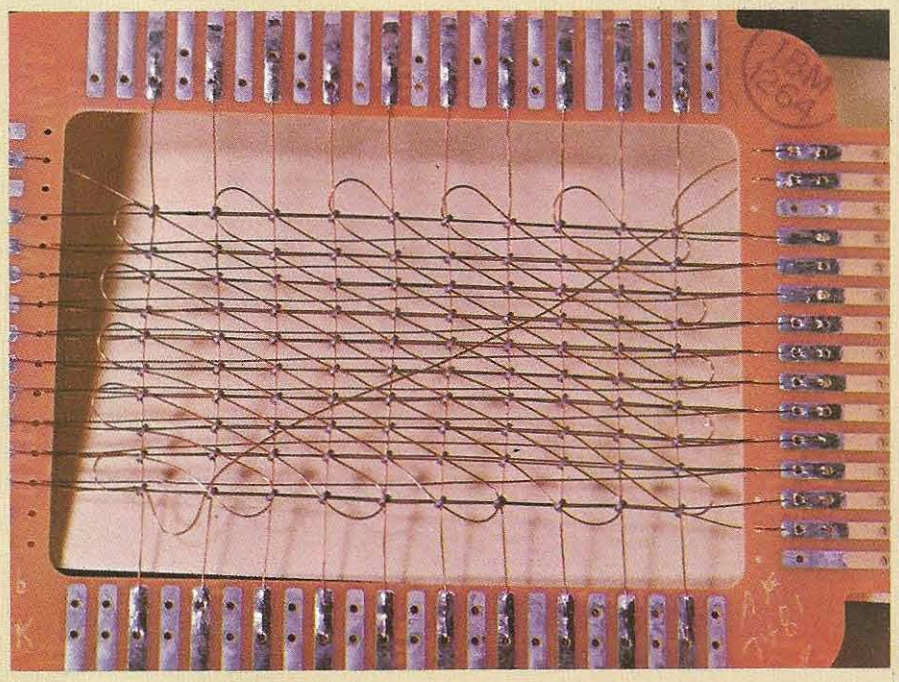
Il manquait cependant encore l'idée intelligente de savoir comment on pourrait « dire » à la machine quelle action elle avait à choisir parmi tout cet « ensemble d'actions » et à quel moment précis elle aurait à la déclencher.

Babbage appela ce dispositif « machine analytique ». Il convient de souligner ici qu'il parlait déjà d'un « calculateur programmable ».

En premier lieu, cette machine comprenait un ensemble de dispositifs d'entrée, qui permettaient d'y introduire des nombres ou des instructions. Puis il y avait une unité arithmétique, ou processeur, c'est-à-dire la partie de la machine qui exécutait vraiment les calculs. Babbage l'appela « la machine ». En troisième lieu, on trouvait une unité de contrôle assurant que le calculateur exécutait bien une tâche plutôt qu'une autre, et qu'il accomplissait bien tous les calculs dans la bonne séquence. C'est un magasin, ou mémoire, qui retenait les nombres en instance de traitement. Enfin, il y avait le dispositif de sortie proprement dit.

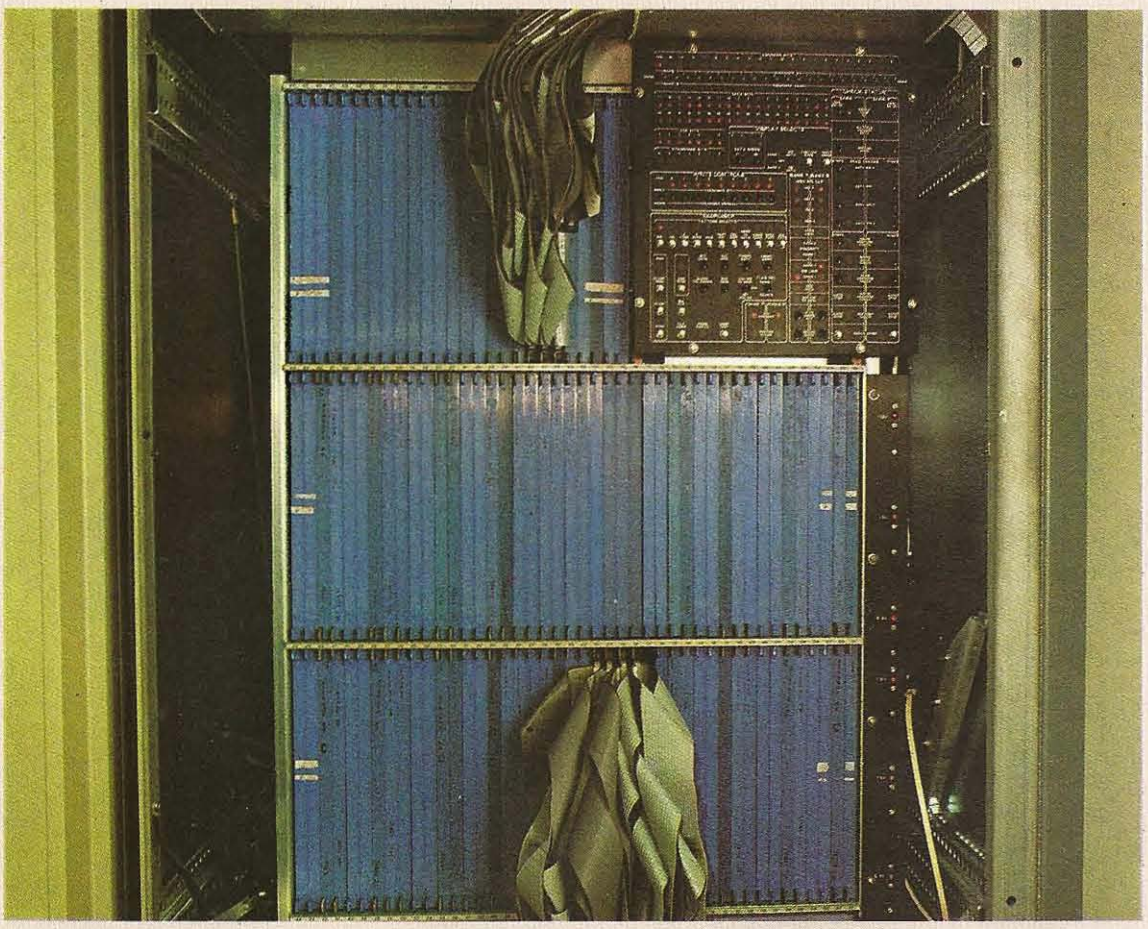
Bien qu'encore schématique, ces cinq parties sont les organes essentiels d'un ordinateur ancien ou moderne. Les dispositifs de calcul et de mémoire de Babbage étaient des colonnes de roues à dix dents, réunies par une incroyable forêt d'arbres et de joints.

Le prototype de la machine à différences était manœuvré à la main. On actionnait un levier pour faire bouger les engrenages ; une cloche sonnait quand le stade de calcul



Archives Curcio/IBM

La première forme de mémoire électronique était constituée de noyaux magnétiques, où l'élément de mémoire est un petit anneau (tore) de ferrite et où l'entrelacement des fils sert à sélectionner les "adresses", ainsi qu'à lire et écrire la donnée au moyen d'impulsions électroniques.



Dominique Fradin

Les mémoires des ordinateurs évoluent. Ici : intérieur d'un ordinateur moderne constitué d'un assemblage de cartes mémoire reliées par des câbles plats.

recherché était atteint. On introduisait alors ce qu'on voulait, on actionnait de nouveau le levier jusqu'à la sonnerie de la cloche et ainsi de suite. Babbage, qui avait conscience de l'absurdité de ces manœuvres, décida d'adopter l'alimentation à la vapeur pour sa dernière version, la machine analytique.

Pour charger et pour programmer les instructions de l'unité de contrôle, Babbage s'inspira d'une invention du Français Joseph Jacquard. Ce dernier s'était rendu compte que les tisserands accomplissaient devant leur métier une tâche délicate mais répétitive et qui aurait donc pu être automatisée. Il avait alors inventé un carton rigide perforé. Pendant le tissage, une série de tiges guidaient les fils de chaîne, et la fonction du carton perforé était d'en bloquer certaines tandis que les autres passaient à travers les trous du carton, continuant le tissage. A chaque passage de la navette, un carton perforé d'une série donnée de trous apparaissait sur le passage des tiges, contrôlant le dessin du tissu. En quelque sorte, il contrôlait aussi le métier à tisser. Babbage comprit que ce système aurait été autrement utile pour contrôler la séquence des calculs dans sa machine.

Le rapport avec le tissage fut également découvert par Ada, comtesse de Lovelace qui, dans un commentaire sur la machine de Babbage, écrivit : « la machine analytique tisse ensemble les algébriques comme le métier de Jacquard tisse les feuilles et les fleurs ». Cette femme était une mathématicienne exceptionnelle, et qui plus est, d'une beauté hors du commun. Un croquis de l'époque la montre ressemblant légèrement à Elizabeth Taylor, avec quelques traces des beaux traits de son père, Lord Byron. Quand elle fit connaissance de Babbage et se rendit compte de l'importance de ses tentatives, elle se mit à étudier à fond les plans de la

machine analytique et se fit donner des explications de vive voix. Elle avait de l'argent et son enthousiasme, mais il fallut plusieurs années avant d'arriver à quelque chose.

Finalement elle publia une longue série de notes intitulée « Observations sur la machine analytique de Monsieur Babbage ». C'est une excellente lecture pour qui veut connaître l'entreprise dans le détail, et on y voit que lady Lovelace avait conscience de quelques-uns des problèmes philosophiques qui se posent dans la construction d'une machine de ce type, problèmes dont certains n'ont toujours pas été résolus aujourd'hui. Par exemple, sur la question de savoir si la machine était créative ou non, elle écrivait : « la machine analytique n'a nullement la prétention de créer quoi que ce soit. Elle peut faire tout ce que nous sommes en mesure de programmer. Elle peut faire des analyses, mais elle ne peut avancer aucune vérité ni aucune relation analytique. Son but, c'est de rendre accessible ce que nous connaissons déjà ». Ce commentaire très intelligent semble avoir été la première déclaration sur les capacités intellectuelles d'un ordinateur. Un ordinateur peut faire seulement ce pour quoi il est programmé. Comme on le verra par la suite, la question est intéressante, bien que peut-être un peu secondaire. Le mérite de l'avoir formulée en revient entièrement à lady Lovelace.

Le plus important est l'effet psychologique que les écrits de cette dame eurent sur Babbage. Jusque-là, à part lors de son succès auprès de la Société Astronomique, il n'avait jamais été approuvé de personne, ni même compris. Le plus satisfaisant pour lui était que la comtesse ait pris la peine d'étudier ses théories sous leur aspect mathématique et qu'elle n'y ait trouvé aucune imperfection. Lui, il savait bien que la machine analytique devait fonctionner et voici que cette femme le croyait aussi. Alors il ne restait plus qu'à construire l'appareil en question.

Cependant, en Angleterre le gouvernement avait changé. Le ministre du Trésor avait renoncé à financer un projet qui paraissait désormais irréalisable. Le temps passait, la machine à différences restait un ensemble incomplet de roues et de cames ; la machine analytique n'était qu'une série de croquis, d'idées et de notes de lady Lovelace. Les choses allaient de mal en pis. Ada Lovelace mourut, âgée de trente-six ans, et Babbage continua seul, obtenant peu de résultats. Les gouvernements se succédaient, aucun ne sympathisant avec son idée. Disraeli écrivit même, faisant preuve d'un humour plutôt sarcastique, que, selon lui, le seul emploi qu'on pouvait faire de la machine était de s'en servir pour calculer combien elle avait coûté...

Pourtant, pendant ce temps, d'autres mathématiciens et ingénieurs feuilletaient avec grand intérêt les publications de Babbage et les notes de lady Lovelace. L'un d'eux, l'ingénieur suédois George Scheutz, fit construire sa propre version de la machine à différences. Contrairement à Babbage, il obtint un succès si retentissant qu'il construisit aussitôt un prototype qu'il montra comme le premier « modèle pour la production » à une Exposition d'Ingénierie en 1855. Dans la foule qui se pressait autour de ce curieux dispositif, se tenait Babbage, âgé de soixante-treize ans et fort découragé. Quand on lui demanda un commentaire sur la machine de Scheutz, il le fit courtoisement et ils se congratulèrent. Néanmoins, on peut imaginer les pensées qui lui ont alors traversé l'esprit.

Babbage devait mourir en 1871, à l'âge de soixante-dix-neuf ans. Il faut bien admettre qu'il est mort déçu, après avoir pourtant conçu un projet si révolutionnaire qu'il devait un jour changer le monde. Ses contemporains le considéraient comme un génie rêveur et sans jugement, mais en tout cas comme un génie. Ils étaient même tellement convaincus de la supériorité de son intelligence, qu'après sa mort ils soumirent son cerveau à un examen attentif pour voir s'il présentait des caractéristiques physiques différentes d'un cerveau commun. Victor Horsely, l'un des plus célèbres chirurgiens de l'époque, s'occupa de l'examen. Après avoir tourmenté des millions de neurones éteints, il déclara que ce cerveau ne lui semblait pas différent de tant d'autres qu'il avait déjà vus. Toutefois, personne ne se décida à détruire le cerveau de Babbage, et il est encore conservé dans deux bocaux, un pour chaque hémisphère, au musée Hunter du Collège Royal des Chirurgiens, où on vous le montrera si vous le désirez.

(D'après THE MIGHTY MICRO de Christopher Evans. © Christopher Evans 1979.)

Evolution des ordinateurs : applications

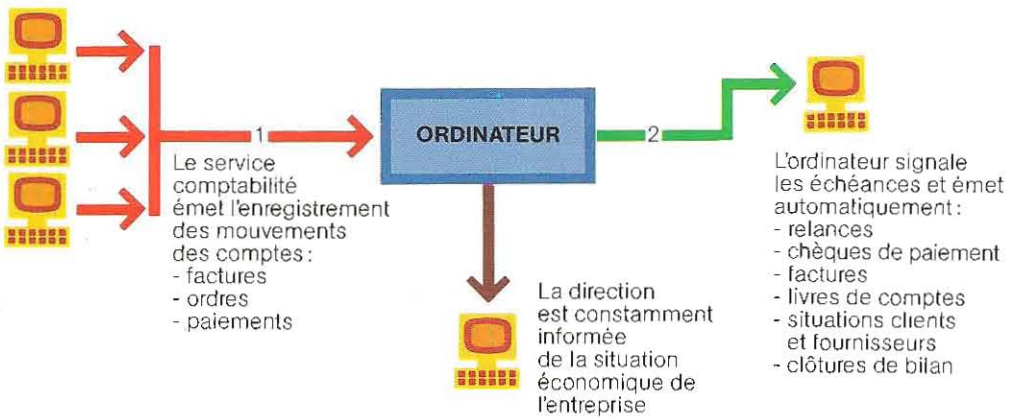
Dans les années 80, l'évolution de l'ordinateur n'a été marquée par aucun événement spectaculaire. Pourtant, les progrès de la miniaturisation allaient permettre de diminuer la taille et le prix des ordinateurs, les rendant accessibles à un plus grand nombre d'amateurs.
Pendant ce temps-là, un profond changement intervenait dans la société.
Dans un monde industrialisé, la nécessité première est l'information. L'échange d'informations au moment opportun peut signifier l'aboutissement d'une recherche scientifique, le juste choix d'une décision économique ou, plus simplement, un précieux gain de temps. L'ordinateur est l'instrument idéal pour ces applications. Sa capacité de mémoriser autorise à stocker de grandes quantités d'informations auxquelles l'utilisateur peut accéder très rapidement.

Les exemples sont multiples. Prenons d'abord celui de la gestion d'une bibliothèque. La recherche des volumes qui traitent d'un sujet déterminé est longue et ennuyeuse si elle est faite à la main. Au contraire, si la bibliothèque est gérée par un ordinateur, les informations sont acquises très rapidement, et sans contraindre l'utilisateur à une activité manuelle qui, par ailleurs, vole du temps à sa créativité.
Citons un autre exemple. Dans toute entreprise, les différents départements doivent connaître la situation générale de la société pour pouvoir programmer leurs propres activités. Ainsi, le stock de pièces finies doit être connu aussi bien du service de production que de celui des expéditions. Cette complexité d'informations ne peut être gérée avec succès que par l'intermédiaire d'un ordinateur. Les schémas reproduits dans cette page et la suivante donnent une idée des services que peut rendre l'ordinateur dans une entreprise.

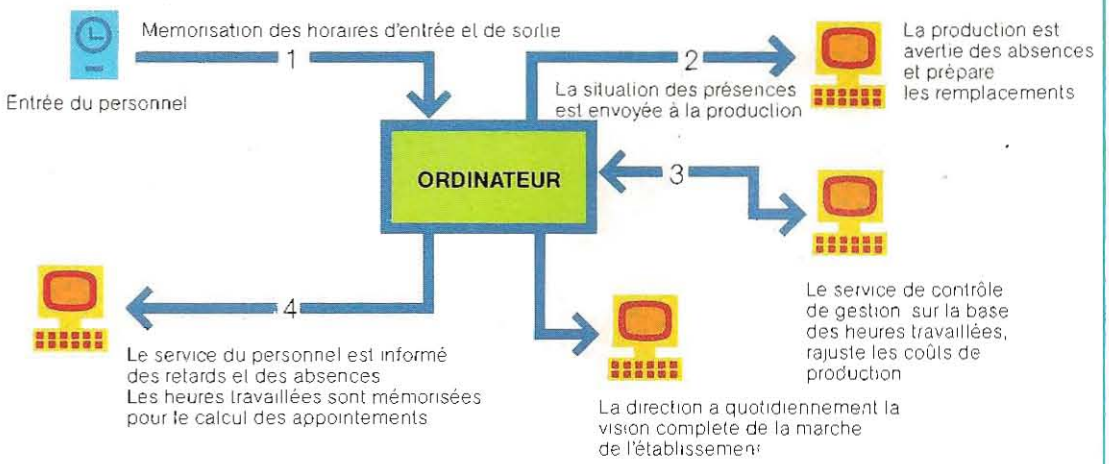
PRINCIPAUX CHAMPS D'APPLICATION DE L'ORDINATEUR DANS UNE ENTREPRISE



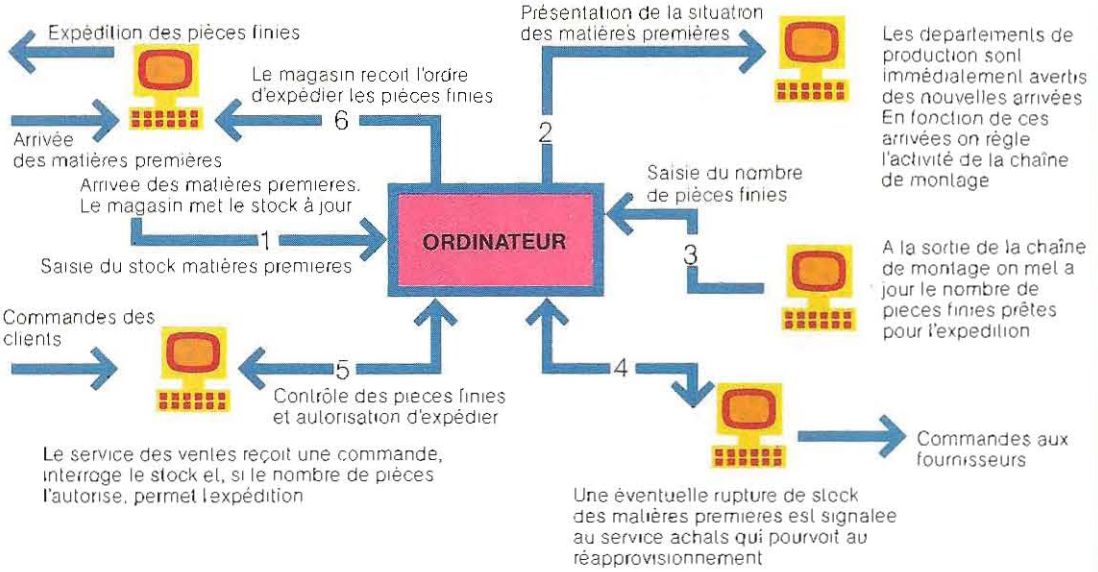
Gestion de la comptabilité



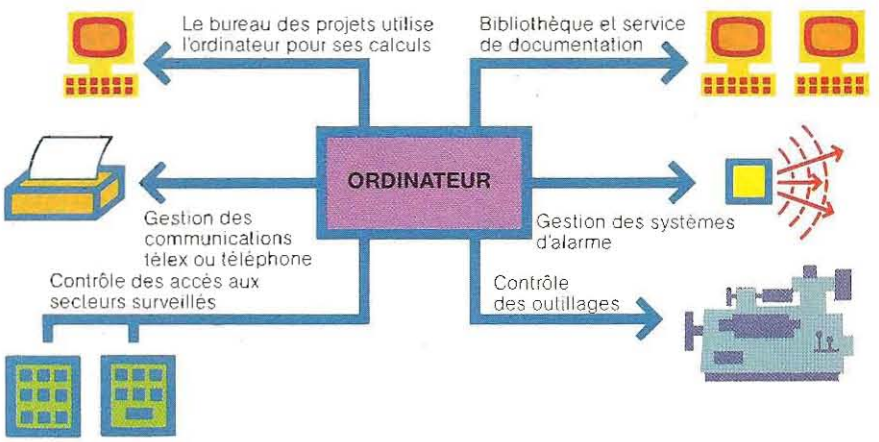
Gestion du personnel



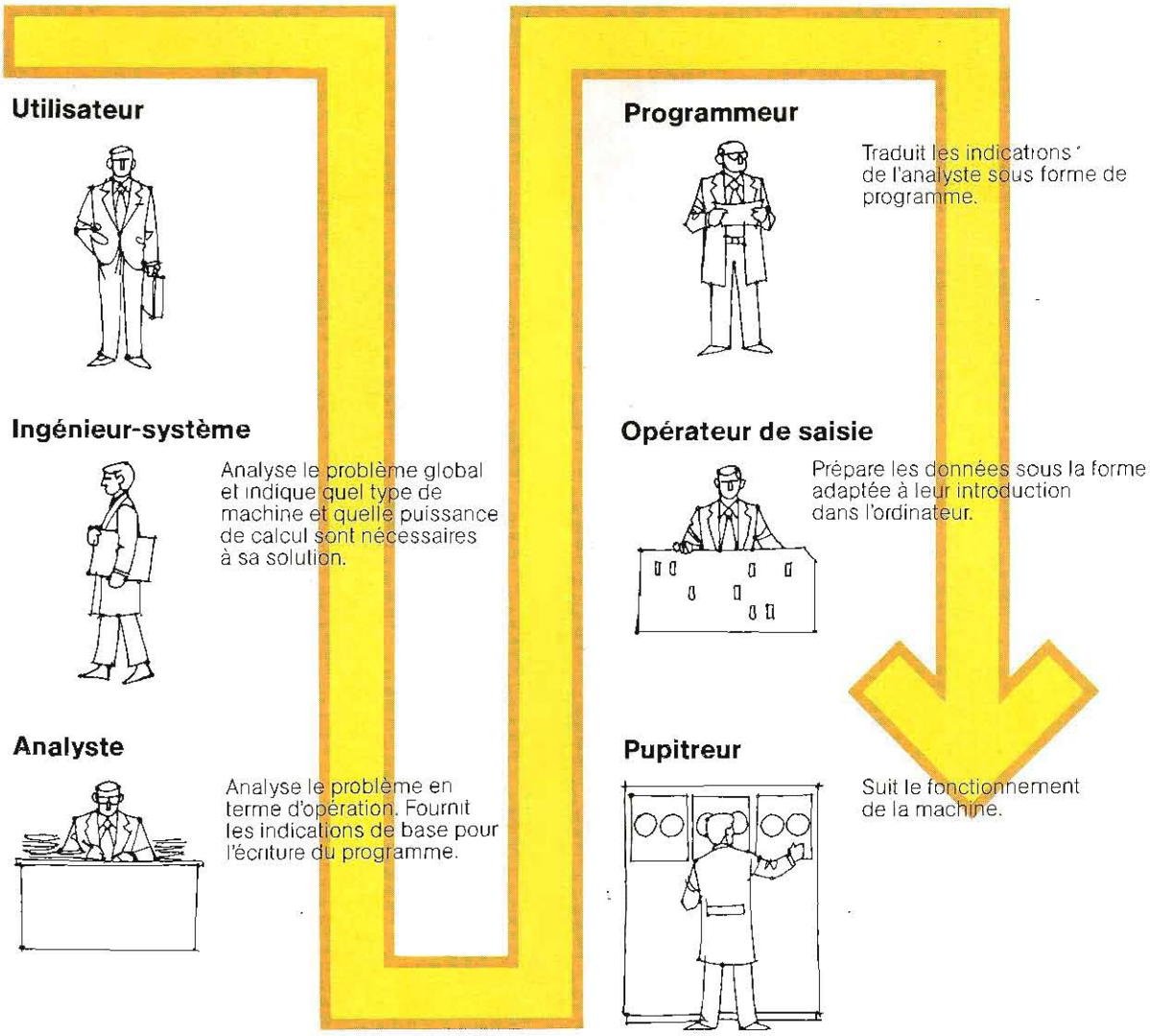
Système d'information pour la production



Applications particulières



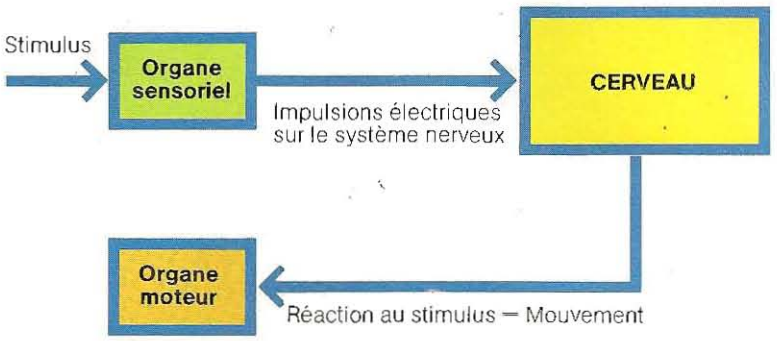
STRUCTURE D'UN CENTRE DE CALCUL



La science particulière qui étudie les formes et les modes de traitement automatique de l'information est l'**Informatique.**
Cette science a des aspects très complexes, et elle transformera, dans un proche futur, les procédures de la gestion d'entreprise.
L'informatique n'a pas seulement cet aspect rigoureusement scientifique applicable à une industrie complexe : elle peut également être une aide de la gestion familiale.
Combien d'entre nous connaissent, à tout moment, le montant de leur compte en banque ? La majorité des gens a en tête un chiffre approximatif, parfois assez éloigné de la réalité. Or il existe déjà des services bancaires à domicile qui permettent, à partir d'un terminal, d'interroger son propre compte, ou de faire des transferts d'un compte à l'autre, entre autres.
Il y a encore peu de temps, les ordinateurs étaient des instruments coûteux et compliqués, dont l'usage exigeait la compétence de spécialistes ayant chacun des fonctions bien définies (voir le schéma de cette page). La structure nécessaire au fonctionnement de ces machines constituait une charge économique considérable et, par conséquent, les ordinateurs ne pouvaient être vraiment utilisés que dans de grandes entreprises industrielles. L'entreprise de petite ou moyenne dimension n'avait aucun intérêt économique à adopter des procédures automatisées.
Ces dernières années, les développements

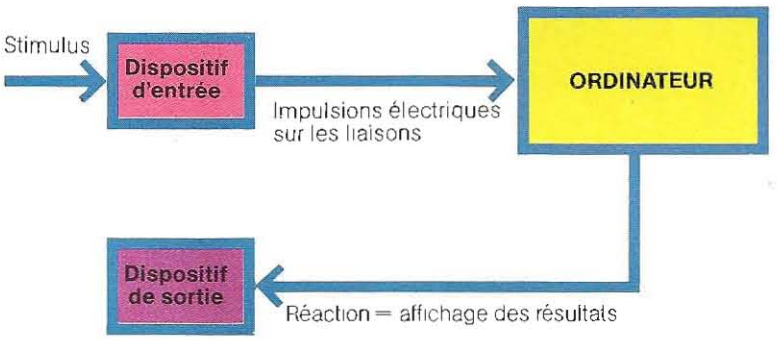
SCHEMA LOGIQUE DU PROCESSUS ACTION-REACTION

Processus action-réaction dans le cerveau



Un stimulus, la piqûre d'un insecte par exemple, produit une série d'impulsions électriques. Le cerveau les traite et commande, en réaction, une action motrice.

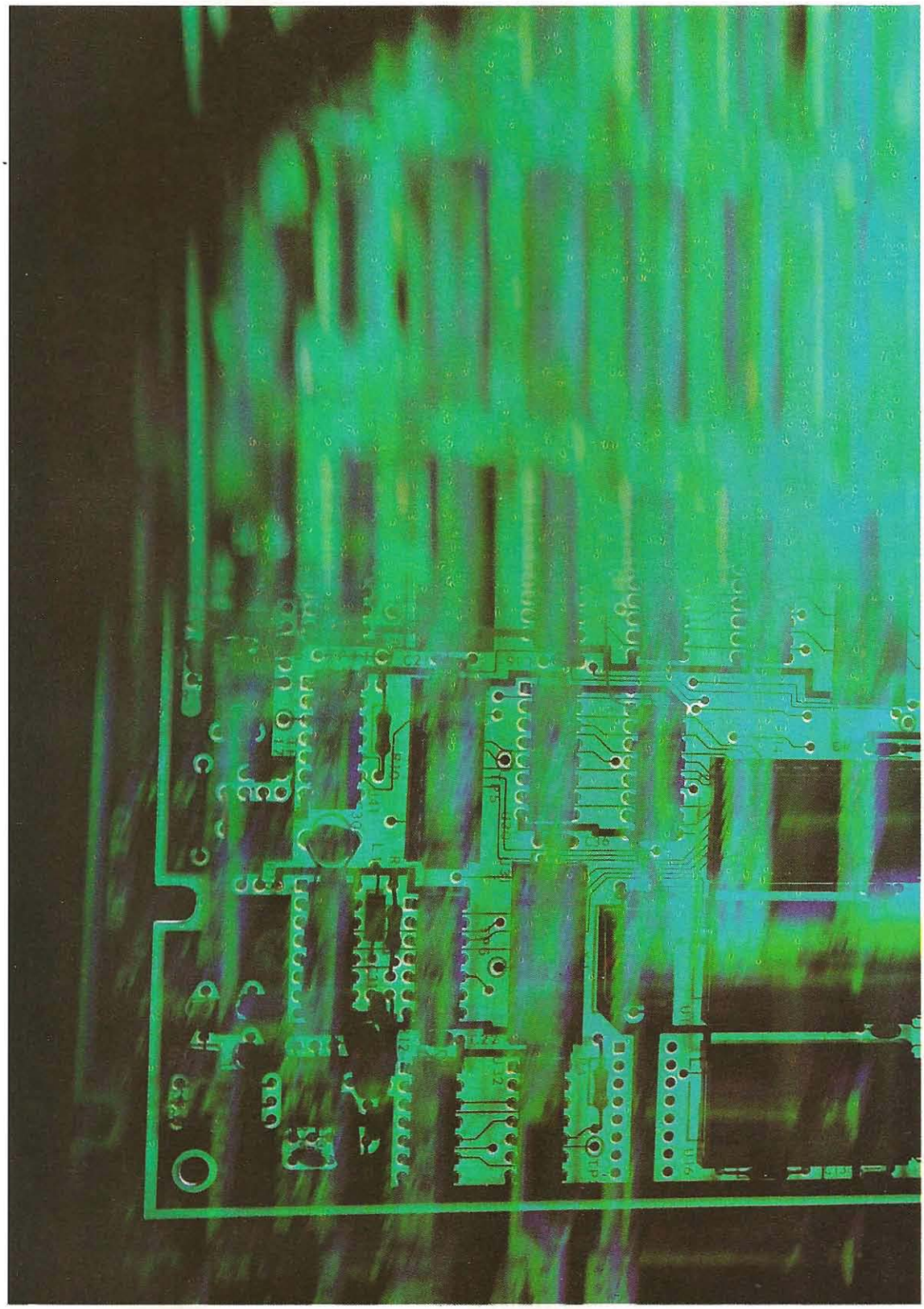
Processus action-réaction dans l'ordinateur



Dans l'ordinateur, le stimulus est l'ordre d'exécuter une certaine transaction. Cet ordre est converti en signaux électriques dans le dispositif d'entrée et envoyé, sur les liaisons appropriées, à l'ordinateur.
Les signaux sont interprétés et provoquent l'affichage des résultats (réaction)

technologiques ont au contraire rendu possible la production d'ordinateurs de dimensions très réduites, d'utilisation plus facile et d'un coût assez bas. On imagine facilement ce que pourra être leur développement. Déjà, certains ordinateurs parlent, sont en mesure de lire des textes, d'exécuter des dessins, de jouer aux échecs, de faire des traductions, de jouer de la musique. Quelques prototypes automobiles sont même équipés de microprocesseurs qui obéissent à la parole: ils savent reconnaître la voix de leur propriétaire. Le développement le plus intéressant interviendra peut-être dans le domaine de la médecine en raison des remarquables analogies entre les mécanismes de la perception des stimuli par le corps humain et la méthode d'acquisition des données dans un ordinateur (voir schéma de cette page). Le neurone (unité fondamentale du système nerveux) peut se trouver seulement dans deux états; excité ou non excité. Ces deux seuls états possibles sont à la base du fonctionnement des ordinateurs numériques.

Le développement des ordinateurs et les analogies avec le corps humain ont conduit à la naissance d'une nouvelle science: la **Robotique.** Les découvertes en ce domaine ont permis de produire des machines capables d'effectuer pratiquement tous les travaux humains et qui peuvent être programmées pour avoir des réactions semblables aux nôtres.



Mallio Falconi

Le circuit imprimé est l'un des éléments fondamentaux de toutes les machines électroniques.

La logique de l'ordinateur

Un ordinateur est constitué de composants électroniques simples ou de sous-ensembles permettant:
- la saisie des informations
- leur traitement
- leur mémorisation (stockage)
- leur diffusion (transmission, édition, etc.).

Le traitement est l'affaire du programme, lequel est exécuté par l'unité centrale, qui doit elle-même disposer d'une unité de calcul, d'une unité logique (qui assure les comparaisons entre données) et d'une mémoire centrale où est logé le programme.

Le déroulement d'un programme peut être séquentiel ou logique. Dans une série séquentielle, chaque fonction est toujours effectuée après la précédente et avant la suivante.

Dans une série logique, l'ordre normal peut être modifié selon des événements déterminés. Prenons des exemples hors de l'ordinateur:

- Des feux de circulation tricolores fonctionnent de façon séquentielle: les couleurs se succèdent toujours dans le même ordre.
- L'emploi du changement de vitesses d'une voiture est une séquence logique, puisque le passage d'un état (vitesse) à l'autre est réglé par des causes non liées au temps (voir schéma en bas de page).

Toutes les fonctions que l'ordinateur devra développer sont prédéfinies et mises en œuvre par le programme. Dans le programme il conviendra de prévoir toutes les voies possibles résultant d'éventuels choix logiques.

Les ordinateurs peuvent être classés en deux familles fondamentales:

- **Ordinateurs analogiques**
- **Ordinateurs numériques**

Les ordinateurs analogiques et les ordinateurs numériques

Le terme analogique dérive de leur principe de fonctionnement. Le calcul mathématique à résoudre est effectué au moyen d'une analogie avec des grandeurs physiques. La règle à calcul, par exemple, est un dispositif analogique puisqu'elle permet de trouver la solution d'opérations arithmétiques grâce à une analogie avec des longueurs (voir figure p. 34, en haut). Dans ce genre de machines, le résultat d'une opération est obtenu en exploitant des lois physiques adéquates. Pour chaque type d'opération il faut donc le circuit approprié, et ce sont des machines limitées à des emplois spécifiques de type essentiellement scientifique.

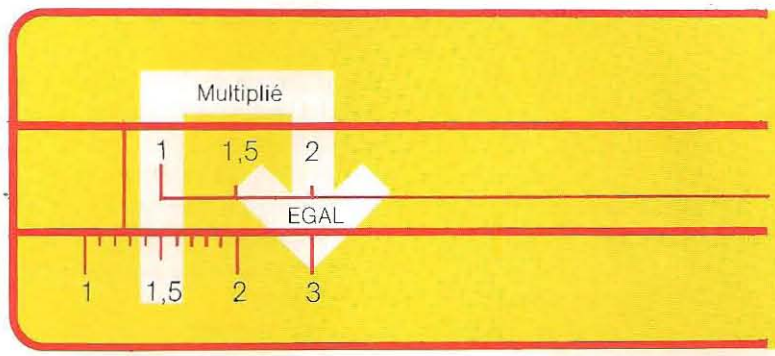
Nous nous limiterons à parler des ordinateurs électroniques actuels, même si notre démonstration peut s'appliquer à d'autres machines d'un type plus ancien (ordinateurs à fluide ou ordinateurs mécaniques).

Mode séquentiel: *les feux de signalisation*



Mode logique: *l'emploi du changement de vitesses*





REGLE A CALCUL

La règle à calcul est un instrument communément utilisé pour exécuter des calculs parfois complexes, exception faite des additions et des soustractions. C'est un exemple d'instrument analogique : en effet elle effectue des calculs par analogie avec les longueurs des graduations qu'elle porte.

Les circuits électroniques de l'ordinateur peuvent être de deux sortes :

- **circuits numériques**
- **circuits analogiques**

Un circuit numérique ne sait reconnaître que deux situations :

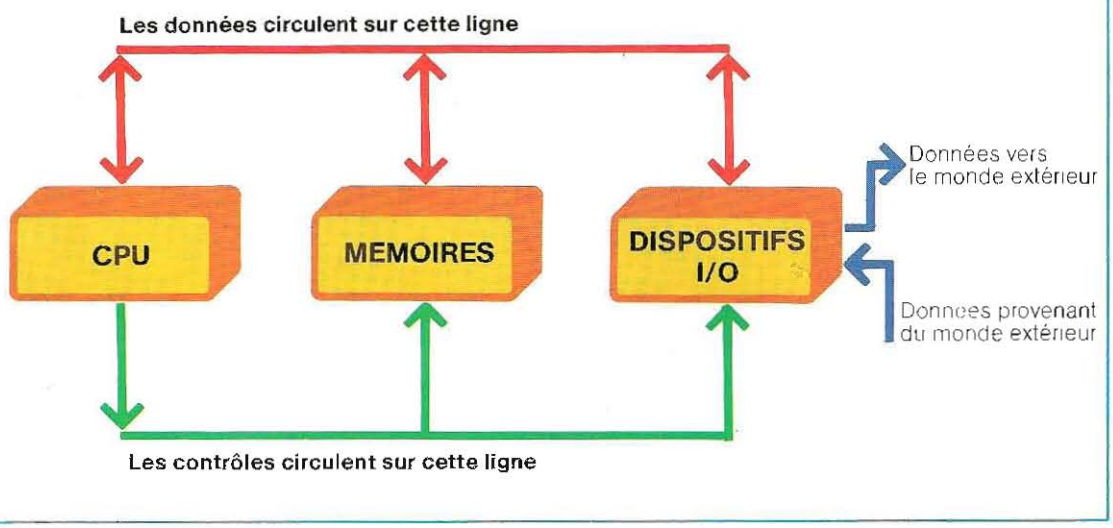
Présence de signal = ON

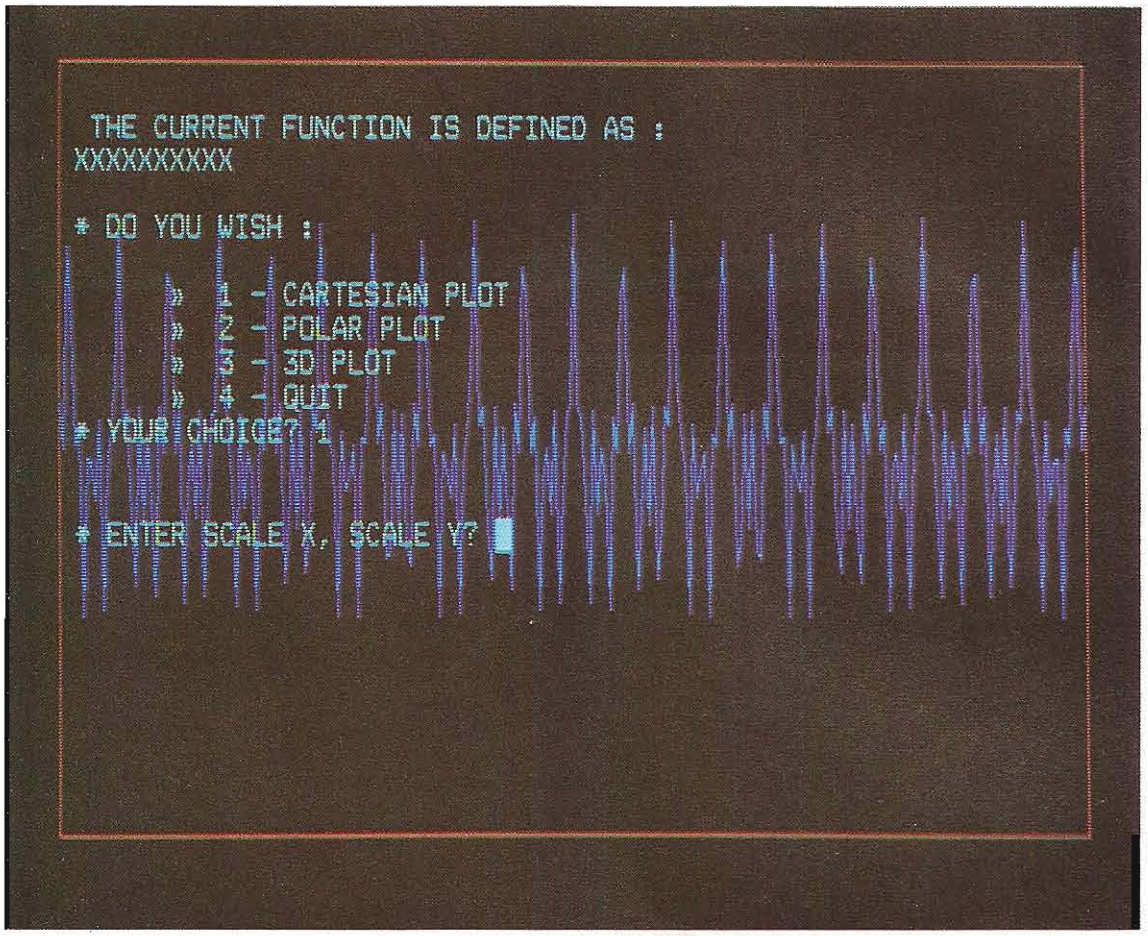
Absence de signal = OFF

En revanche, le circuit analogique est capable de mesurer la présence ou l'absence de signal, et également sa valeur.
La plaque photographique est un dispositif analogique dans la mesure où elle produit une donnée (photo) proportionnellement au signal (lumière).
Un interrupteur est un dispositif numérique qui n'a que deux possibilités : ouvert ou fermé.
Un poste radio, un téléviseur, une installation stéréo sont des appareils analogiques qui fournissent des images et des sons proportionnels aux signaux reçus.

Un ordinateur numérique est une machine capable d'exécuter des fonctions logiques à travers une série d'états :
ON/OFF
(Présence/Absence)

STRUCTURE PRINCIPALE D'UN SYSTEME A MICRO-ORDINATEUR





Centre THC

Type d'affichage écran (display) représentant le menu d'un programme particulier permettant l'affichage graphique d'une fonction choisie.

Un ordinateur peut exécuter les fonctions suivantes :

- Opérations arithmétiques entre deux données
- Comparaison entre deux données et choix consécutifs
- Transfert et mémorisation des données.

Bien que le nombre de ces fonctions principales soit très restreint, on peut tout de même, grâce à certaines combinaisons, effectuer n'importe quel calcul.

Le schéma de la page 34 (en bas) résume la structure d'un ordinateur, qui regroupe :

- Un dispositif capable d'exécuter les opérations arithmétiques et les comparaisons, c'est le CPU (unité centrale de traitement, ou UC).
- Un (ou plusieurs) dispositif capable de mémoriser les données, c'est la mémoire.
- Un dispositif capable de recevoir et de faire entrer les informations, c'est l'entrée (Input) et de les sortir pour les communiquer à l'homme, c'est la sortie (Output), le tout désigné par I/O, ou E/S en français.

L'unité centrale (CPU) assure l'exécution des calculs et envoie les ordres nécessaires au fonctionnement des autres unités.

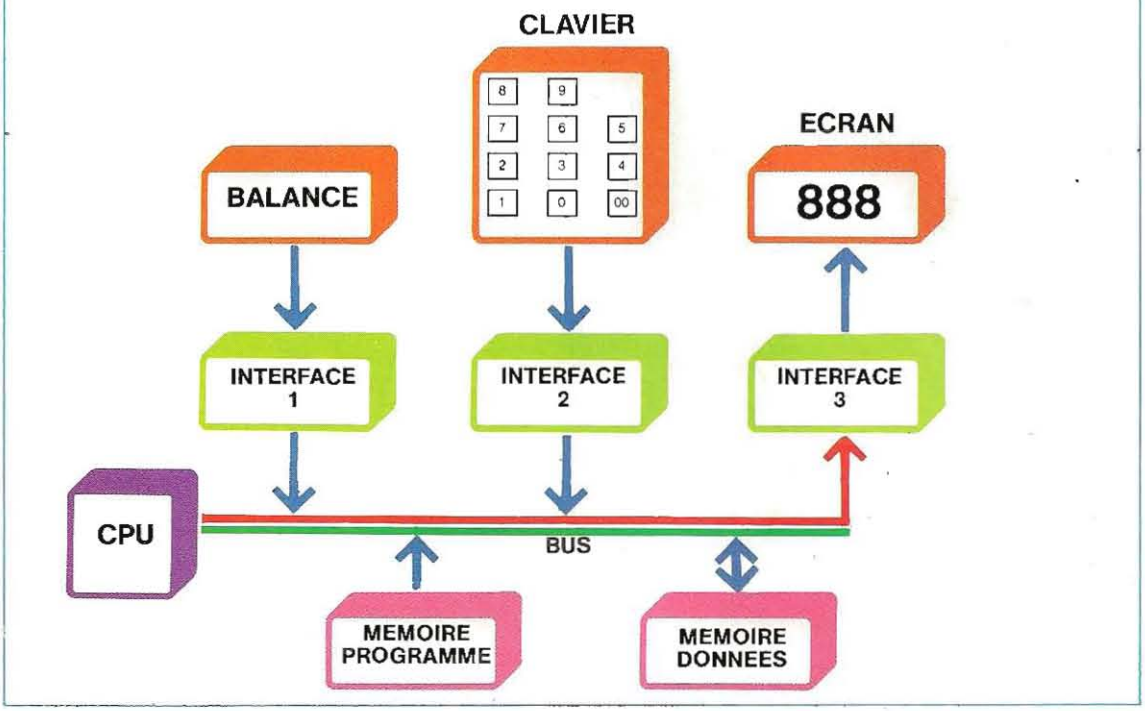
La mémoire est chargée de recevoir et de stocker les données, puis de les restituer quand on les lui demande.

Les dispositifs d'entrée et de sortie transforment les données pour les rendre compréhensibles à la machine et à l'homme.

Reportons-nous au schéma de la page 36 illustrant le cas de la balance électronique qu'utilisent beaucoup de commerçants.

Pour simplifier, la ligne de transmission des données et celle du contrôle sont représentées par une liaison unique, le Bus.

UN SYSTEME A ORDINATEUR : LA BALANCE ELECTRONIQUE



Les fonctions que doit exécuter le système sont :

A / Pesée d'un objet et saisie par la machine de la valeur de ce poids.
La pesée est assurée par la balance. Cette donnée est transformée en information numérique dans l'interface 1. L'interface 1 est un dispositif d'entrée (Input).

B / Saisie du prix (à l'unité) de l'objet.
La fonction est remplie par le clavier et l'interface 2. L'interface 2 est un autre dispositif d'entrée.

C / Calcul du prix total.
La fonction est exécutée à l'intérieur du CPU (ou UC).

D / Présentation du résultat.
La fonction est effectuée par l'ensemble interface 3 et display*. L'interface 3 est un dispositif de sortie (Output).

Dans le schéma, on voit deux autres blocs :

- **Mémoire Programme**
- **Mémoire Données**

Voyons maintenant pourquoi il faut utiliser deux mémoires.

* Par le terme display, ou écran (ou visuel), on désigne le dispositif de visualisation des données. Nous reviendrons sur ce sujet par la suite.

La fonction de la Mémoire Programme est évidente. Cette dernière contient une série d'informations qui, correctement interprétées, signalent au CPU la séquence (A-B-C-D) de fonctions à mettre en action, et qui constituent donc le programme. La Mémoire Données permet d'emmagasiner les données au moment de leur entrée et de les restituer quand le CPU le demande.

La séquence des opérations est la suivante :

1 / La balance signale qu'un objet est posé sur son plateau.

2 / Le CPU demande à la Mémoire Programme l'opération à accomplir.

3 / La Mémoire Programme signale qu'elle reçoit la donnée (poids) et qu'elle la transmet à la Mémoire Données.

4 / Une fois le transfert accompli, le CPU interroge la Mémoire Programme.

5 / La Mémoire Programme signale de mettre en œuvre le clavier et d'acquérir la donnée (prix).

6 / Le CPU actionne l'interface 2 et saisit la donnée du clavier.

7 / Le CPU interroge la Mémoire Programme.

8 / La Mémoire Programme répond qu'il faut prélever la donnée acquise au stade 3 et la multiplier par celle reçue au stade 5.

9 / Pour finir, après un nouveau cycle d'inter-

rogations du CPU à la Mémoire Programme, le résultat de l'étape 8 est envoyé à l'interface 3 qui le fait apparaître sur l'écran de la balance.

La suite d'opérations qui vient d'être décrite, une fois traduite dans un langage accessible à la machine, constitue un **programme.**

Où et comment utiliser un ordinateur

L'emploi d'un ordinateur est lié à une condition essentielle :

IL FAUT POUVOIR LUI FOURNIR

TOUTES LES DONNEES NECESSAIRES

Cela n'est pas aussi évident que l'on peut croire. On entend parfois des questions étranges de la part de ceux qui connaissent mal l'ordinateur et qui lui attribuent des capacités tout à fait exagérées.

Il faut se rappeler en effet qu'un ordinateur ne peut traiter des informations (données) que si ces dernières lui ont d'abord été fournies.

Le second point à examiner est :

QUAND L'ORDINATEUR SERT-IL ?

On ne peut pas dire, a priori, que l'acquisition d'un ordinateur soit toujours une bonne opération financière. En général, pour qu'une telle machine soit rentable, il faut qu'au moins une partie des conditions suivantes soit réunie.

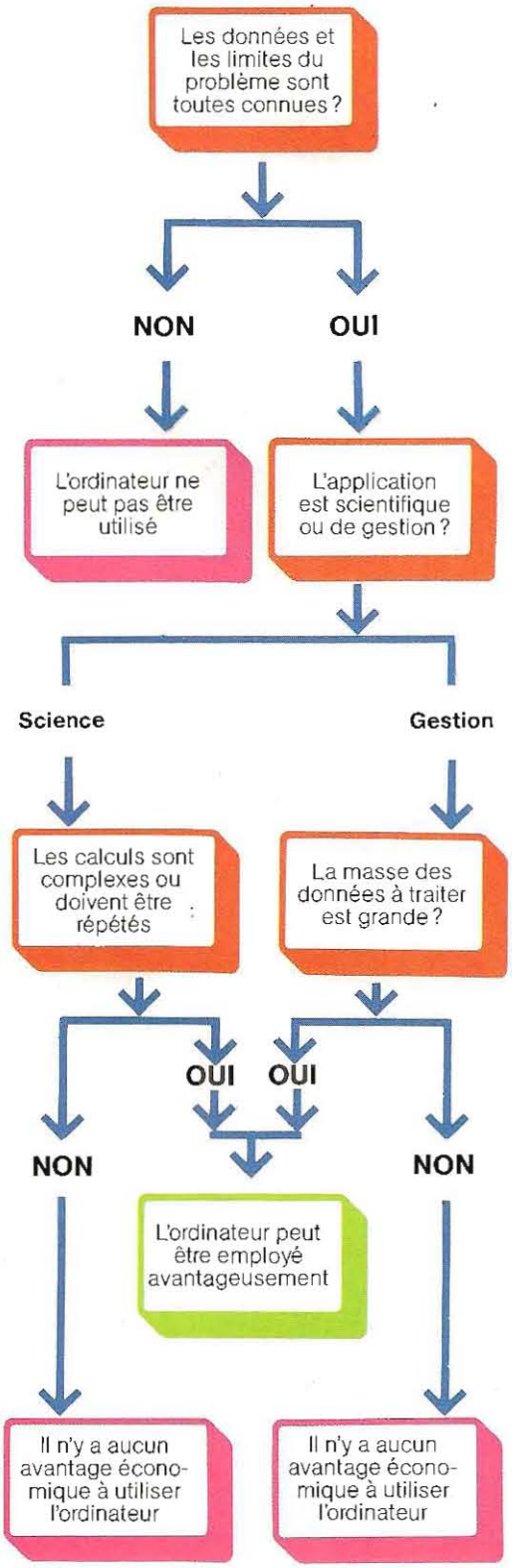
1 / **On connaît bien toutes les données.**
2 / **On aura à traiter une grande quantité de données de façon répétitive.**
3 / **On aura une grande quantité de calculs à faire.**
4 / **On aura à contrôler constamment la même chose.**

Le point 1 est la condition de base.

L'ordinateur ne peut pas donner plus que ce qui lui a été fourni. Il ne sortira pas de résultats si le programme ne sait pas de quoi il s'agit. Le point 2 indique dans quelles conditions un ordinateur est utile. Il faut se rappeler que si l'on installe ce type d'équipement, la dépense initiale sera suivie d'autres dépenses dues au développement des programmes (logiciel) qui imposent une mise de fonds importante. Ces dépenses ne seront amorties que si elles sont compensées par un fort allègement du travail humain, ou par l'acquisition d'un grand nombre d'informations permettant l'amélioration des décisions de gestion.

Le point 3 concerne les problèmes scientifi-

ANALYSE DU PROBLEME



ques en particulier. Très souvent, la mise au point d'un projet nécessite des calculs répétés pour obtenir le meilleur résultat possible. Le dernier point (4) concerne l'emploi de la machine pour contrôler un processus quelconque. Dans ce cas, l'acquisition d'un ordinateur est presque toujours un investissement rentable.
Le schéma de la page 37 résume bien l'ensemble des conditions qu'il faudra vérifier avant de décider l'achat d'un ordinateur.

Classement des ordinateurs

Jusqu'à maintenant nous avons utilisé le terme d'ordinateur pour désigner des machines qui peuvent, en réalité, être très différentes. Essayons donc de les classer selon les performances dont elles sont capables.
Elles peuvent avoir les caractéristiques essentielles suivantes :

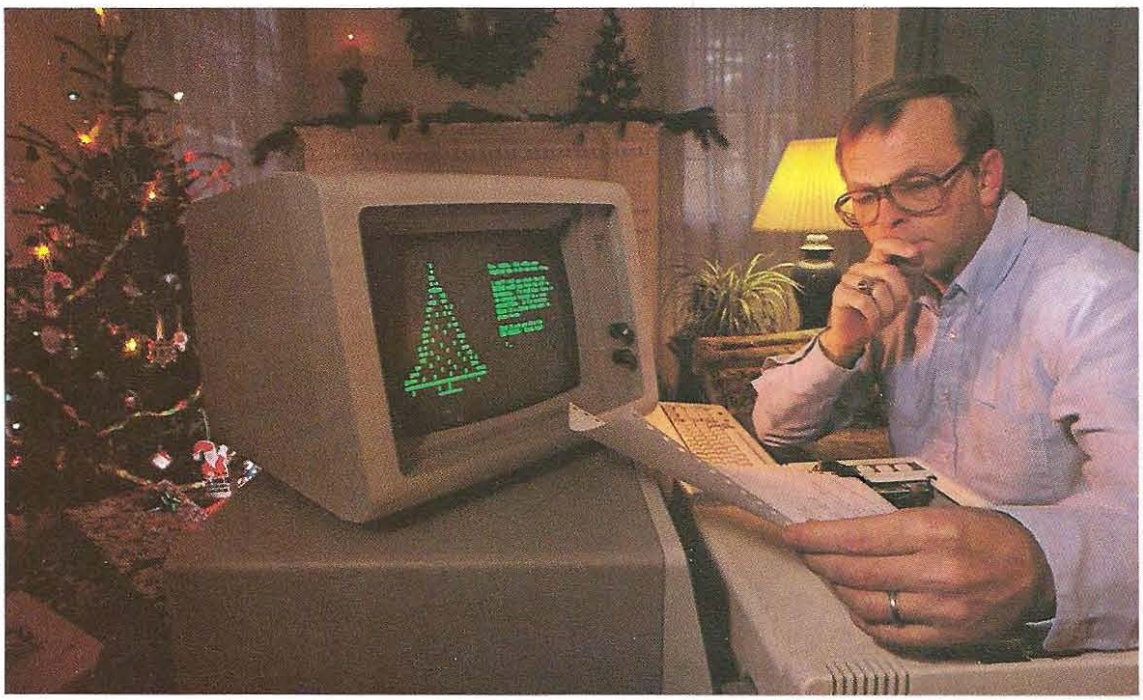
A / Rapidité de calcul.
B / Rapidité de transfert des données.
C / Capacité plus ou moins grande de mémorisation des données.
D / Nombre plus ou moins grand d'utilisateurs qui peuvent s'en servir en même temps.

La valeur de chacun de ces points permet de définir les grandes lignes du champ d'application de la machine, et donc la catégorie à laquelle elle appartient. On distingue :

- **GROS ORDINATEURS**
- **MINI-ORDINATEURS**
- **MICRO-ORDINATEURS et ORDINATEURS PERSONNELS**

Le GROS ORDINATEUR est, comme son nom l'indique, un ordinateur de grandes dimensions, et qui peut être utilisé par plusieurs personnes en même temps.
Il faut une équipe d'informaticiens pour le faire fonctionner. Pour être performant il doit avoir d'énormes capacités de mémoire et une grande rapidité d'exécution. Ce type de machine est réservé à la gestion et à l'administration des grandes industries ou à la recherche dans d'importantes unités scientifiques.
C'est sur les gros ordinateurs qu'on calcule les orbites des satellites ou les trajectoires des fusées ; qu'on effectue les calculs et les vérifications d'ingénierie civile pour les ensembles complexes, et, enfin, qu'on élabore les projets de circuits électroniques.
Dans le domaine de la gestion, ils sont utilisés pour des archives de grand volume, comme celles des tribunaux. Ils sont tout à fait adaptés, aussi, à la gestion des registres de l'état-civil et du fisc, des stocks de marchandises ou des cycles de production.

L'ordinateur personnel à la maison.



Grazia Neri/Contact



Ericsson

Reliés à un gros ordinateur, les terminaux permettent à tous les utilisateurs d'accéder à la banque de données de l'entreprise.

Il faut également mentionner d'autres emplois très spécifiques, comme la simulation d'une guerre. Ces ordinateurs sont en effet capables d'étudier toutes les phases d'une guerre hypothétique, ce qui permet de mettre au point un certain nombre de stratégies.

Le MINI-ORDINATEUR est au contraire un appareil de prix relativement limité, plus facile à utiliser et qui, en général, sert à plusieurs personnes en même temps. Les structures du mini-ordinateur sont assez extensibles dans la mesure où on peut leur ajouter des unités additionnelles qui en augmentent les capacités.

La tendance des fabricants est de fournir, pour commencer, des machines dont les possibilités sont limitées aux besoins immédiats de leurs clients, et d'y apporter ensuite les extensions qui leur sont devenues nécessaires. La puissance d'un système en termes de capacité de mémoire, de dispositifs d'entrée et de sortie s'appelle **configuration.**

Le champ d'utilisation de ces machines est très varié et dépend de leur configuration. L'utilisateur du mini-ordinateur n'a pas des besoins assez grands pour un gros ordinateur mais ne serait pas satisfait par les services d'un micro-ordinateur, insuffisant pour lui.

Le MICRO-ORDINATEUR et l'ORDINATEUR PERSONNEL sont en principe des machines destinées à un seul utilisateur, et dont l'emploi est très simple. Comme nous le verrons mieux par la suite, la frontière entre micro et ordinateur personnel est parfois assez vague, d'autant plus que de nouvelles technologies permettent d'augmenter la puissance des micro-ordinateurs.

Si l'on a quelquefois tendance à faire des micro-ordinateurs et des ordinateurs personnels deux types distincts, c'est plutôt en raison de la différence de leurs capacités de mémorisation des données et des langages utilisés (nombre plus ou moins grand et variété des instructions disponibles), alors que toutes leurs autres fonctions sont semblables. En revanche, leur utilisation simplifiée et leur prix nettement plus bas permettent à beaucoup de personnes de les acquérir pour des usages très divers.

Voici la liste des principales utilisations de ces petits ordinateurs :

- Préparation des enveloppes de paie dans les petites entreprises
- Comptabilité du petit commerce ou du grand magasin
- Facturation
- Gestion d'immeubles
- Calculs d'ingénierie civile
- Gestion des petites bibliothèques
- Gestion des fichiers d'adresses
- Analyses financières
- Gestion de stocks.

L'ordinateur à la maison

L'entrée dans les foyers de ce que l'on nomme terminal domestique ou micro-ordinateur familial, caractérise l'une des applications les plus révolutionnaires de la cybernétique moderne.

Les perfectionnements de la technologie et l'abaissement du prix des machines permettent désormais aux particuliers d'acquérir des micro-ordinateurs et de bénéficier d'un type tout nouveau d'informations et de services à domicile.

On peut déjà entrevoir les opérations bancaires faites avec le terminal domestique; de nombreux projets de ce type sont déjà en phase d'expérimentation aux Etats-Unis. Il est facile également d'envisager le «shop at home», c'est-à-dire «faites vos courses à domicile». Les services d'information apparaissent très diversifiés et certains, déjà opérationnels aux Etats-Unis, puisent leurs données dans plus d'un millier de bases de données. Parmi ces services nommons ceux qui fournissent des informations professionnelles de caractère médical ou agricole, pour ne citer que ceux-là.

Toutefois, les secteurs d'application qui intéresseront le plus le lecteur européen sont ceux de type interactif. Selon ce mode, l'utilisateur peut obtenir chez lui des informations qui le concernent particulièrement: sur la sécurité par exemple, aussi bien dans le cas d'une urgence médicale que pour la protection contre intrus et voleurs, ou encore contre les incendies et les inondations. L'utilisateur peut également participer en «temps réel» à des sondages d'opinion: on diffuse le résultat du sondage au fur et à mesure qu'on recueille les réponses des personnes interrogées.

Parmi les chaînes américaines privées, celles qui ont été créées par les groupes ethniques, les petites communautés ou les associations religieuses jouissent d'un grand succès dans ce type de transmission. Le terminal domestique est très utile en mode interactif surtout pour les associations de bienfaisance qui cherchent à collecter des fonds en direct chez le particulier.

D'autres applications, sans doute moins originales, mais assurées de réussir, sont les études à domicile et la version moderne de l'autodidacte ou du rat de bibliothèque, qui pourra désormais rester chez lui.

L'ordinateur familial vous aide à tenir votre compte bancaire, à établir la liste des achats de la semaine, et peut même vous donner des conseils de santé.



Dominique Fradin



Thermorégulation des pièces. Organisation et gestion des services de la maison



Liaison avec des banques de données



Transmission et réception de documents

Contrôle de la fermeture des portes et des fenêtres. Alarme incendie et fuite de gaz.





Agenda. Archives de la famille. Comptabilité et correspondance





Paiement et transfert de fonds



Logiciel de formation



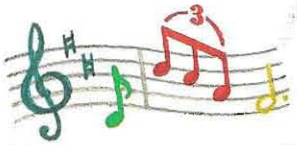
Achats au supermarché



Jeux



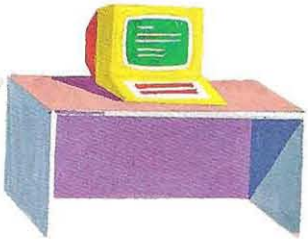
Annuaires téléphoniques



Faire de la musique et l'apprendre



Gérer la bibliothèque et les informations culturelles



Poste de travail à domicile en liaison avec le bureau



Dominique Fradin

L'ordinateur à la maison permet de mieux suivre la vie de tous les membres de votre famille.

Citons enfin les programmes de loisirs comme les émissions sportives, que l'utilisateur sélectionnera à l'avance.

Tous ces services, que l'on peut compter désormais au nombre de nos futures habitudes de vie, nous laissent imaginer à quel point cette agression cybernétique sera plus contraignante encore que celle que nous a imposé la télévision. Cette dernière implique en effet un programme rigide égal pour tous, parfois abêtissant, et concentré en des horaires souvent paralysants pour la vie sociale. Le micro-ordinateur, au contraire, est, à première vue, l'instrument du choix, de la participation, de la personnalisation et de la liberté.

Pourtant, cela n'empêche pas d'entrevoir les querelles de famille qui risquent de naître autour de cet instrument. En effet, c'est un peu comme si on concentrait dans un seul appareil un téléphone et une encyclopédie, un téléviseur et un projecteur de cinéma, un agenda et la liste des courses, le livre de comptes et un échiquier... Il ne faudra donc pas s'étonner qu'un seul clavier et qu'un seul écran soient vite insuffisants.

Mais la réponse naturelle sera là encore dans les progrès de la technologie : à très court terme il sera possible à chaque foyer d'acquérir plusieurs claviers et plusieurs écrans, permettant ainsi à tous les membres de la famille de bénéficier de multiprogrammes.

Ce sera l'apparition de petits réseaux d'ordinateurs familiaux.

Ces constatations, ces réflexions, partent des données observées par quelqu'un qui a vécu la réalité américaine. Cependant, il n'est pas inutile de présenter le problème au lecteur européen tant il est vrai que l'élévation de son niveau de vie aussi bien que les perfectionnements de la technologie mettront très prochainement à sa portée une situation semblable. L'informatique pénètre dans tous les pays industrialisés de la même manière : proportionnellement aux revenus des habitants. Si cette situation nouvelle ne se matérialise pas immédiatement chez nous par l'acquisition massive de ces micro-ordinateurs, cela ne veut pas dire que l'évolution ne se fait pas dans le même sens. Elle est moins rapide en Europe qu'aux Etats-Unis, mais, ce qui sera un phénomène de masse aux Etats-Unis vers 1985, le sera en Europe en 1990. Néanmoins, nous pourrons bientôt constater l'impact de ces expériences sur une partie limitée de la population. Ce résultat sera-t-il à la gloire des marchands d'ordinateurs ou à la gloire de l'évolution culturelle ? C'est là un des aspects du défi que comporte aussi la révolution cybernétique.

(D'après L'INFORMATICA A DOMICILIO de Mauro Langfelder. Feltrinelli 1983.)

Les plus petits de ces modèles entrent de plus en plus dans les foyers. Ils peuvent y servir de répertoire d'adresses, ou aider à étudier et à gérer le budget familial. Certains modèles coûtent déjà moins cher qu'un téléviseur et ils font désormais partie de nos appareils électroménagers.

Réseaux d'ordinateurs

Tous les ordinateurs doivent communiquer avec leur utilisateur. Ils disposent donc de circuits capables de recevoir et de transmettre des données. Le plus souvent, la transmission aboutit à une imprimante ou à un écran et provient d'un clavier. Mais rien n'empêche d'y ajouter un autre ordinateur.

C'est ce qu'on appelle un réseau d'ordinateurs. Il permet essentiellement la transmission et l'utilisation d'un plus grand nombre d'informations. Or, comme nous l'avons déjà vu, l'intérêt principal d'un ordinateur est la gestion de grandes quantités de données. Mais plus leur nombre augmente, plus il devient difficile de les gérer ensemble. Le réseau d'ordinateurs permet justement de centraliser ces données tout en en rendant l'accès possible à plusieurs utilisateurs.

La structure type d'un réseau d'ordinateurs est représentée dans le schéma ci-dessous.

Ce schéma met en évidence ses fonctions principales :

- Le satellite (*) 1 est directement relié à l'ordinateur principal, c'est ce que l'on appelle un réseau étoilé.
- Les satellites 2 et 3, reliés de la même façon étoilée, ont un concentrateur.

Ce système est adopté pour faciliter la transmission des données sur de longues distances.

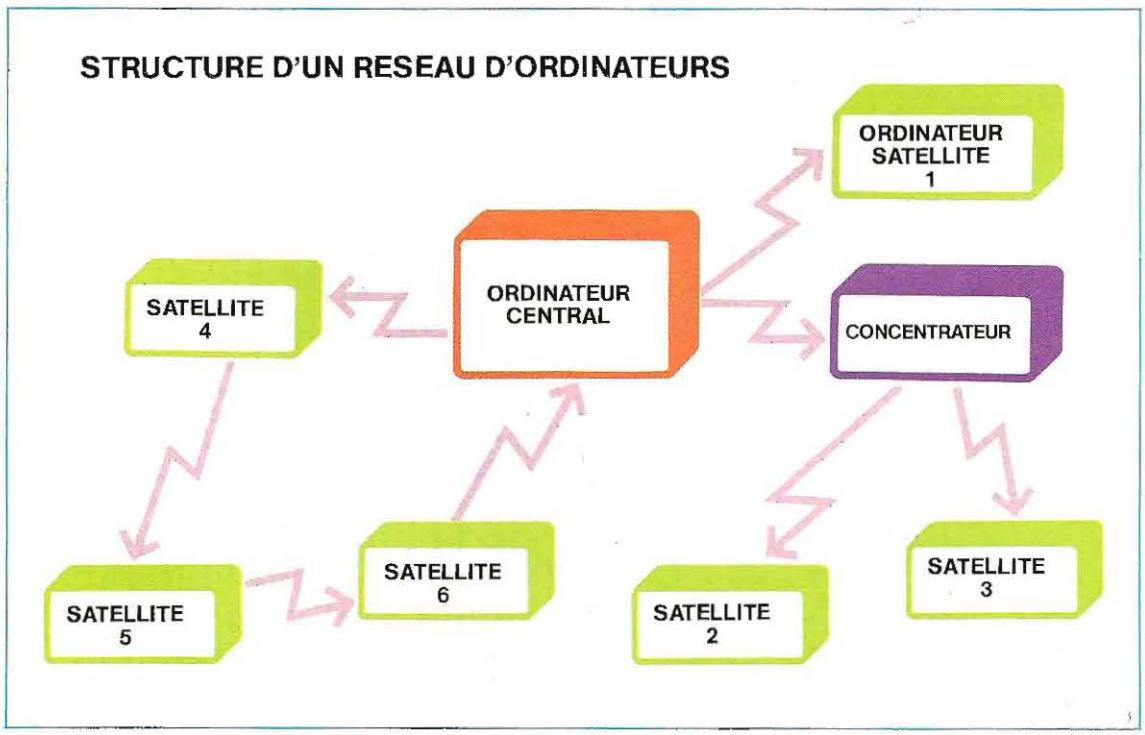
Dans notre exemple, les deux satellites (2 et 3) et le concentrateur sont dans un site très distant de l'ordinateur principal.

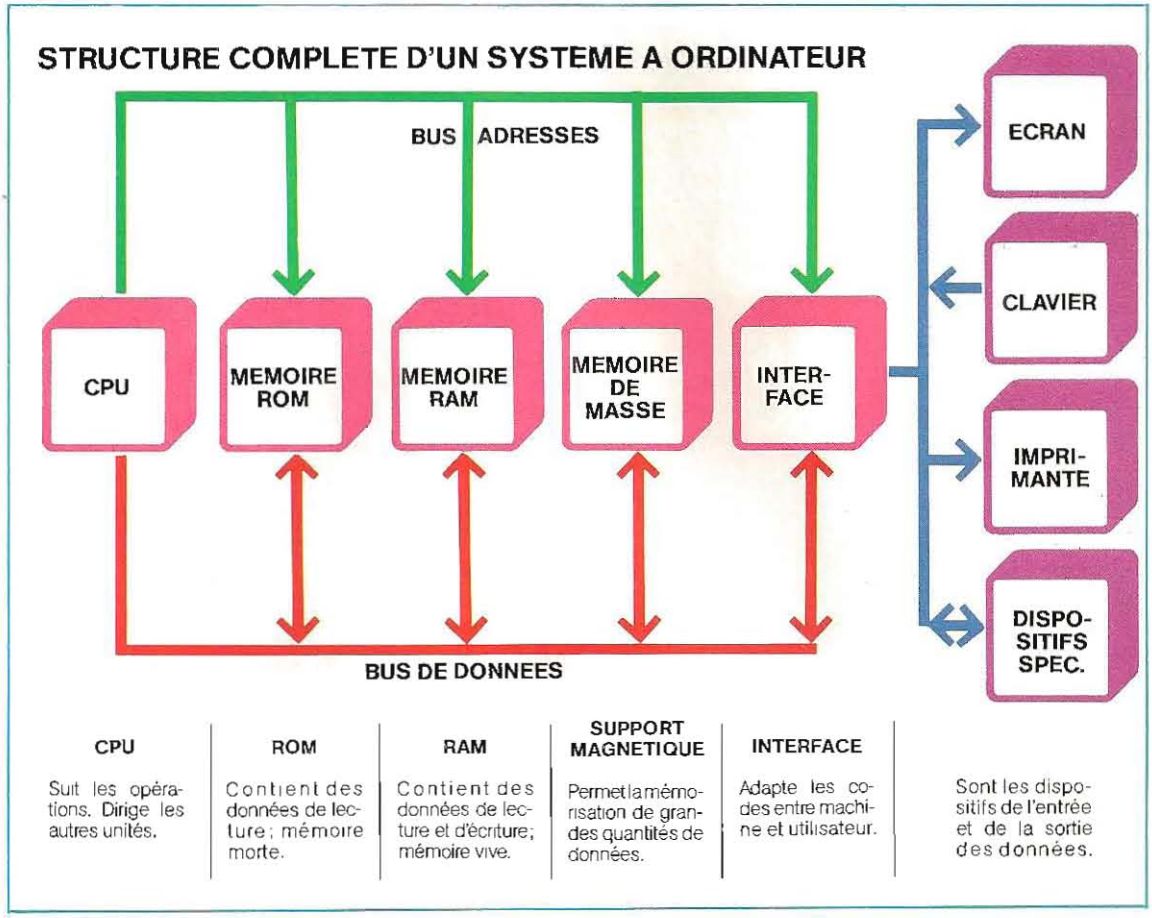
Grâce au concentrateur on peut réunir un certain nombre de lignes locales en une seule ligne étoilée. Le concentrateur a précisément la tâche de diriger l'échange des données sur la ligne éloignée et de les répartir entre les satellites. De cette manière, la ligne de liaison à longue distance est unique. Ensuite elle se divise en un certain nombre de lignes locales (une par satellite).

Toujours dans le même schéma, les satellites 4, 5 et 6 constituent un réseau maillé.

Dans ce système on se passe du concentrateur, mais ce sera la panne si l'un des satellites ou l'une des lignes tombe en panne.

* Par satellite on entend un ordinateur ou n'importe quel autre dispositif qui exécute des fonctions, en partie ou totalement dirigées par une autre machine.





Le schéma de la page 43 donne un aperçu des liaisons en réseaux. On distingue différents *modes de liaisons* : les liaisons télégraphiques (télex), les lignes téléphoniques (réseau commuté), les lignes spécialisées (deux ou quatre fils), les réseaux à valeur ajoutée (Transpac, Euronet) et les liaisons par satellite (Télécom 1).

Il faut distinguer également les *modes de transmissions*, qui peuvent être parallèles ou en série, synchrones ou asynchrones ; ou encore ce que l'on nomme half ou full duplex.

Citons enfin les *équipements* disponibles : modems et convertisseurs, multiplexeurs et concentrateurs, ainsi que commutateurs de circuits, de paquets, de messages, etc.

La télématique française est très avancée grâce, notamment, à la conversion progressive du réseau de télécommunications en technologique numérique et à son expérience des réseaux publics de commutation par paquets (Transpac), sujets sur lesquels nous aurons l'occasion de revenir plus tard dans cet ouvrage.

Un dispositif numérique reconnaît seulement deux états :
ON et OFF. (Allumé et Eteint).
L'ordinateur numérique peut exécuter quatre fonctions principales :
OPERATIONS ARITHMETIQUES
COMPARAISONS
TRANSFERT DES DONNEES
MEMORISATION DES DONNEES.
Un programme est une série logique de fonctions élémentaires.
L'emploi de l'ordinateur est possible seulement si les termes du problème sont cernés avec précision.

Le schéma en haut de la page résume la structure complète d'un système à ordinateur. Les différentes fonctions de chacun des composants seront analysées par la suite.

Les systèmes de numération

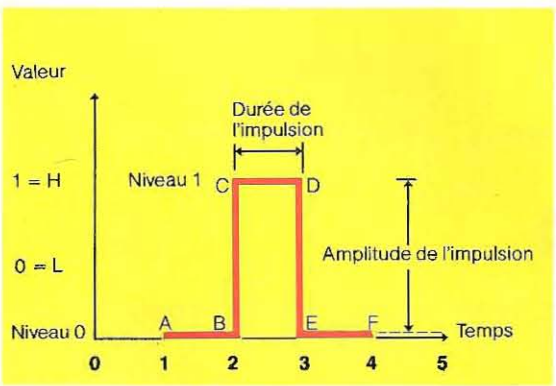
Dans ce chapitre nous exposerons les différentes manières de représenter nos symboles (chiffres et lettres) sous une forme accessible à l'ordinateur.
Nous donnerons à la fin du chapitre les méthodes de calculs à utiliser avec ces symboliques.
Le sujet peut paraître un peu rebutant parce qu'il oblige le lecteur à oublier sa logique personnelle faite de nuances, de réflexions et d'intuitions, pour entrer dans la logique de la machine qui exclut toute forme d'interprétation.
L'aspect conceptuel de cette question devra retenir toute votre attention, alors que l'aspect technique du développement d'un calcul dans sa version adaptée à l'ordinateur est beaucoup plus facile à assimiler, même si les exemples donnés dans ces pages paraissent compliqués.

Les impulsions

N'importe quel dispositif numérique reconnaît seulement deux états :

ON = Présence de signal
OFF = Absence de signal

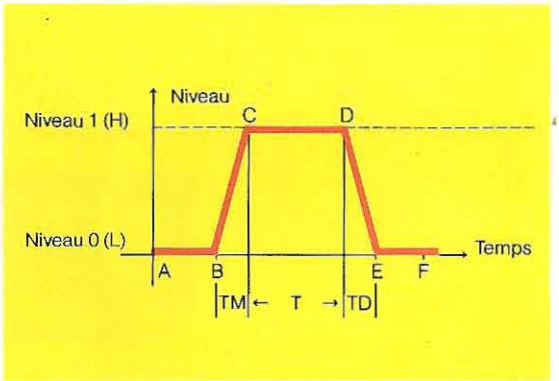
La condition OFF = 0 est le repère et prend le nom de NIVEAU 0.
La condition ON = 1 est représentée proportionnellement à la valeur du signal et prend le nom de NIVEAU 1.
Ces deux conditions peuvent être représentées graphiquement de la façon suivante :



Nous commençons à observer le phénomène à l'instant A. Entre A et B (disons : 1 seconde, soit de la seconde 1 à la seconde 2), le niveau reste bas = 0 (le niveau 0 peut aussi être indiqué par la lettre L).
Entre B et C (très court) on passe du niveau 0 au niveau 1 (le niveau 1 est en général indiqué par la lettre H).
Le niveau H se maintient pour le temps C-D.
Dans l'intervalle (très court) D-E on passe du niveau H au niveau L.
L'état de niveau 0 peut durer indéfiniment. Le cycle :

- Niveau L
- Niveau H
- Niveau L

prend le nom d'**impulsion.**
En fait, la forme d'une impulsion ne peut pas être celle que nous venons de lui donner.
Sur la figure, on ne tient pas compte du temps nécessaire au passage du niveau 0 (L) au niveau 1 (H). De même, pour le temps nécessaire au passage inverse H-L (points D et E).
Car aucun circuit électronique ne peut passer d'un état à l'autre en un temps nul. Donc, la course réelle d'une impulsion est celle-ci :



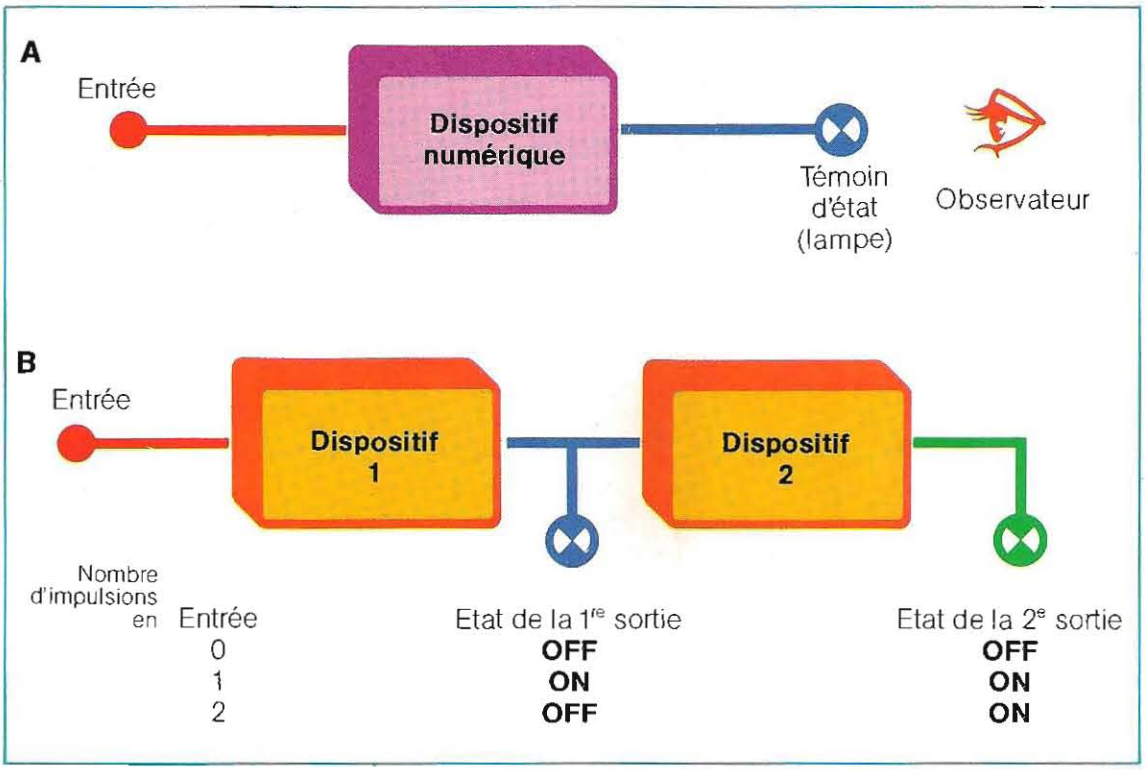
Pour faciliter la représentation, les temps ont été augmentés et on a fait coïncider le point A avec le temps 0.
Le temps nécessaire au passage du niveau 0 au niveau 1 (points B et C) prend le nom de **temps de montée** (il est indiqué sur la figure par TM).
La durée de l'impulsion, au niveau 1, se réduit au segment C-D (indiqué par T).
Le temps de passage du niveau 1 au niveau 0 (points D et E) prend le nom de **temps de descente** (indiqué par TD sur la figure).
La ligne B-C, qui matérialise le temps de montée (TM), est le **front de montée.**
De même, le segment D-E, qui matérialise le temps de descente, est le **front de descente.**

En résumé, les principaux paramètres d'une impulsion sont :

- **Durée = Temps T (segment C-D)**
- **Temps de montée = Temps TM (seg. B-C = Front de montée)**
- **Temps de descente = Temps TD (seg. D-E = Front de descente)**

Commutation

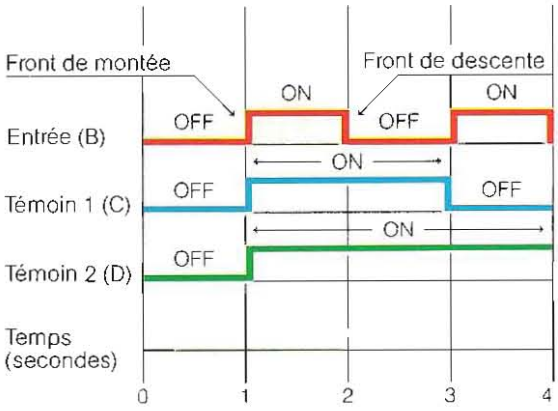
Pour ses calculs, un ordinateur n'utilise que les deux symboles ON et OFF.
On indique l'état ON par 1 (ou avec la lettre H), et l'état OFF par 0 (ou la lettre L).
Supposons maintenant un dispositif numérique en mesure de nous montrer son état (par exemple avec un témoin lumineux) et doté d'une entrée, comme dans le schéma A. Imaginons qu'au début l'état de sortie soit 0 (lampe éteinte), avec l'entrée en 0 également. En mettant l'entrée en 1, on changera l'état du dispositif numérique et la sortie sera portée à 1 (lampe allumée).
L'observateur dira qu'une impulsion a été produite. Ce changement d'état porte le nom de **commutation.**
Le dispositif a commuté une fois.
Si l'on remet l'entrée à 0, puis si on la porte de nouveau à 1, le dispositif aura une nouvelle commutation et la sortie se portera de 1 à 0.
Puisque l'observateur se souvient d'avoir vu d'abord l'état 1, en voyant l'état 0 il dira qu'une nouvelle commutation s'est produite. Le dispositif a donc commuté deux fois, c'est-à-dire qu'il a compté 2.
Supposons maintenant que l'observateur commence son expérience seulement au moment de la seconde impulsion d'entrée.
Dans ce cas il verra la sortie passer de 1 à 0 et il ne pourra pas savoir s'il s'agit d'une seule commutation ou de deux. Cette incertitude disparaît si l'on dispose à la sortie du système, en plus du témoin lumineux, un second dispositif identique, comme on peut le voir sur le schéma B.
La sortie du dispositif 1 provoque la commutation du dispositif 2, il devient donc possible de déterminer quel a été le nombre d'impulsions en entrée (0, 1, 2), puisque le second dispositif mémorise et met en évidence les changements qui affectent le premier.



Les états des Entrée, Témoin 1, Témoin 2, peuvent être résumés en une table.

A	B	C	D
Nombre d'impulsions	Entrée	Témoin 1	Témoin 2
0	0	0	0
1	1	1	1
2	1	0	1

Ce tableau illustre le fonctionnement du système du point de vue logique et se nomme **table de vérité**. Chaque circuit numérique a une table plus ou moins compliquée selon les fonctions qu'il doit exécuter.
Supposons qu'à l'entrée les impulsions arrivent au rythme d'une toutes les deux secondes, et construisons le diagramme suivant:



- Le témoin 2 (D) s'éteint au second allumage du témoin 1 (C); nous avons donc réalisé un compteur par 2.
- Le dispositif qui produit les impulsions équidistantes dans le temps, porte le nom d'**horloge** (clock).

Notons que, dans un circuit numérique électronique, **LE CHANGEMENT D'ETAT INTERVIENT QUAND IL SE PRODUIT UN PASSAGE OFF-ON A L'ENTREE.**

Les bases de numération

Il y a différents systèmes de numération, mais le plus adapté à notre logique est le système positionnel.
Dans ce système, chaque chiffre a un poids ou valeur, dépendant de sa position.
En écrivant le numéro 5, nous entendons 5 unités; si nous changeons la position du symbole 5 nous pouvons entendre 5 dizaines (50), 5 centaines (500), etc. Le symbole 5 prend donc une valeur différente selon sa position. Chaque position a une valeur correspondant à la valeur de base du symbole (0, 1, 2, ... 5, ... 9) multiplié par 10 un nombre de fois correspondant à sa position, comme nous le voyons sur cette table:

	Centaines	Dizaines	Unités
Poids	100	10	1
Exposant	5×10^2	5×10^1	5×10^0
Valeur position.	5×100	5×10	5×1
Valeur	500	50	5

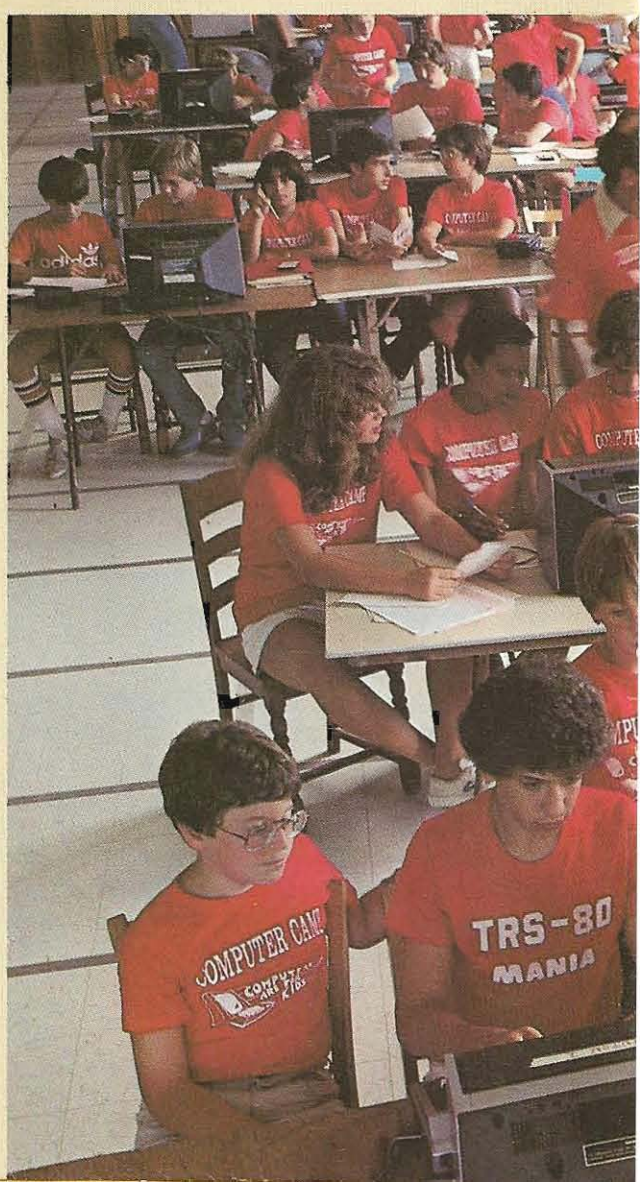
A chaque changement de position le nombre est multiplié par 10. La valeur 10 est la base de notre système de numération, qui à cause de cela se nomme **système positionnel en base 10** (numération arabe).
La numération romaine est un exemple de système non positionnel.
Dans ce système, la valeur 5 est représentée par le symbole V, tandis que la valeur 50 a le symbole L. Pour passer de 5 à 50, il ne suffit pas de changer la position du symbole 5 (V), mais il faut en introduire un nouveau (L).
Ce type de numération, peu pratique à interpréter et peu adapté à l'exécution des calculs, a été abandonné. A sa place, on a universellement adopté le système positionnel.
Le système positionnel le plus courant est le système décimal: les symboles sont dix et la base est 10.
Mais dans les ordinateurs on ne peut pas utiliser dix symboles différents. Comme nous l'avons déjà dit, la machine ne sait reconnaître que deux conditions: ON (niveau haut) et OFF (niveau bas). Elle ne dispose donc que de deux symboles 1 (ON) et 0 (OFF).
C'est pour cette raison qu'il faut utiliser un système de numération qui prévoit l'usage des seuls symboles 1 et 0, c'est pour cela que ce système s'appelle le **système binaire.** Ce type de numération, contrairement à ce que l'on pourrait penser, n'est pas né avec l'ordinateur. Il était déjà utilisé en Chine vers le XIIe siècle. Le système binaire est, lui aussi, un système positionnel car chacun des deux symboles possibles a une valeur dépendant de sa position.

A l'école avec l'ordinateur

On peut considérer l'ordinateur comme une machine extraordinaire pour enseigner, mais pour enseigner quoi? N'importe quoi: pas seulement l'informatique et toutes les matières scientifiques, comme les mathématiques, la chimie, la biologie, mais les matières qui, traditionnellement, n'appartiennent pas au groupe scientifique, comme les langues et la musique. Les enfants, d'ailleurs, semblent avoir une prédisposition naturelle à l'usage d'un ordinateur personnel. Prenons l'exemple d'un programme très simple pour enfants de l'école élémentaire, le «programme pour l'enseignement des soustractions». Dans le dialogue qui suit, les élèves obtiennent une réaction immédiate à leurs réponses.

Ordinateur: 3 — 2 = ?
Enfant: 1
Ordinateur: Très bien. 7 — 2 = ?
Enfant: 5
Ordinateur: Formidable. Et 23 — 16?
Enfant: 6
Ordinateur: Oh, oh! Tu veux essayer de nouveau?
Enfant: 7
Ordinateur: Cette fois c'est bien. Et 44 — 38?

A chaque réponse exacte, l'élève est félicité, et il est arrêté si la réponse est fausse.
Les expressions «Très bien», «Formidable», etc., utilisées dans l'exemple de la soustraction, ne sont que des propositions schématiques. Une réaction plus pittoresque, comme un visage souriant ou courroucé, ou des dessins animés accompagnés d'effets sonores, tels qu'une sonnerie de trompette pour les bonnes réponses, et un crissement de freins, ou le bruit d'un choc violent pour les réponses fausses, serait évidemment plus efficace et plus amusante, à condition cependant que les réactions à une réponse inexacte ne soient pas trop bouleversantes, sinon l'élève se découragera rapidement.
Le type de présentation choisie pour le programme peut varier à l'infini. Dans notre exemple, le programme propose des problèmes de plus en plus difficiles, dès qu'il est sûr que l'élève est capable de résoudre les plus faciles. C'est pourquoi le programme modèle passe rapidement au-delà des problèmes de soustractions à un chiffre, comme 3 — 2, 7 — 2, qui sont faciles à résoudre. Les réactions de l'ordinateur ne sont pas exclusivement déterminées par deux bonnes réponses successives de l'élève. L'horloge contenue dans l'ordinateur peut, par exemple, avoir signalé aussi le temps pris pour répondre aux deux questions. En outre, les données de la leçon précédente peuvent avoir été conservées pour que la machine puisse rappeler à l'élève (qui l'aurait alors oublié) qu'il a déjà fait ce genre de problèmes. En tout cas, le calcul à deux chiffres plus difficile, (23 — 16) lui a été présenté ensuite, suivi de 44 — 38, et ainsi de suite. A part des calculs de difficulté croissante, le programme peut aussi offrir à l'élève la possibilité de changer de sujet (et de passer par exemple à la lecture de poésies, à la composition de musique rock, ou à une partie de football), ou même de mettre fin à la leçon, si le temps d'une réponse à l'autre et la qualité de cette dernière indiquent que l'élève commence à se fatiguer ou à s'ennuyer.



Grazia Neri/E. Ferorelli

Bien qu'il soit élémentaire, cet exemple applique à l'enseignement de la soustraction une méthode très supérieure à celle des manuels classiques. L'élève disposant d'un livre ne peut résoudre les problèmes que dans l'ordre où ils lui sont présentés, qu'ils soient trop faciles ou trop difficiles. En outre, l'étudiant ne reçoit aucune indication sur l'exactitude de ses réponses. Si le corrigé du problème lui est fourni au bout de plusieurs jours, il risque d'avoir tout oublié de la question.

L'intérêt et le niveau d'apprentissage sont beaucoup plus élevés si la réponse est fournie au fur et à mesure. Cela est vrai aussi dans les lycées où, trop souvent, le professeur fait son cours sans tenir compte des questions posées par ses étudiants. Ceux-ci notent scrupuleusement ses paroles sans oublier la liste des devoirs à faire. Les étudiants attendent parfois plusieurs jours après la remise de leur travail et, au moment des corrections, ne sont plus toujours en mesure de comprendre pourquoi ils ont commis des erreurs. On se rend compte à quel point il est important que l'élève puisse dialoguer davantage et c'est pourquoi l'enseignement assisté par l'ordinateur pourrait être très utilement adopté pour d'autres cours que des cours de mathématiques. C'est toute une pédagogie nouvelle et constructive qui voit le jour avec l'emploi des ordinateurs individuels dans les écoles. L'expérience se développe déjà, aux Etats-Unis et dans de nombreux pays d'Europe comme la France ou la Grande-Bretagne.

Dans ce domaine, l'ordinateur est le nouvel outil didactique qui s'intègre dans un autre rapport entre le professeur et l'élève.

(D'après THE ELECTRONIC COTTAGE. © Joseph Deken, 1981.)

L'engouement des moins de 20 ans pour l'ordinateur favorisera autant l'implantation dans les écoles de l'enseignement assisté par ordinateur que la création de clubs d'amateurs.



Nous allons à présent examiner en détail les mécanismes du système positionnel. Nous avons vu que dans la notation décimale habituelle, chaque nombre est représenté par une suite ordonnée des chiffres 0, 1, 2, 3... 9. La valeur d'un chiffre, le 5 par exemple, est différente selon la position qu'il occupe dans le nombre ou, en d'autres termes, selon la quantité de chiffres qui le suivent : c'est ainsi que 50 diffère de 500.
Ecrire un nombre en base 10 est une opération abstraite qui nous est dictée par l'habitude. En réalité nous utilisons les chiffres disponibles comme multiplicateurs des puissances successives de la base 10. Rappelons que les puissances d'une base sont les valeurs obtenues en calculant le produit d'autant de facteurs égaux à la base qu'indiqué par l'exposant placé à droite et un peu au-dessus. Par exemple :

$10^0 = 1$*	puissance zéro (unité)
$10^1 = 10$	première puissance (dizaine)
$10^2 = 10 \times 10 = 100$	deuxième puissance (centaine)
$10^3 = 10 \times 10 \times 10 = 1000$	troisième puissance (millier)

Dans la numération décimale, chaque chiffre, en partant de la droite et en progressant vers la gauche, est un multiplicateur des puissances croissantes de la base 10 (on parle couramment d'unités, de dizaines, de centaines, de milliers, etc., mais ces termes perdent leur signification quand la base n'est plus 10). La procédure rigoureuse qui permet d'interpréter un nombre du système décimal et d'en obtenir la valeur est décrite par le tableau ci-dessous, qui se rapporte au nombre entier 1024 :

Colonne	d	c	b	a
Nombre (chiffres)	1	0	2	4
Puissance de la base	10^3	10^2	10^1	10^0
Poids des positions	1000	100	10	1
Valeur des chiffres (chiffre x poids)	1000	0	20	4
Valeur du nombre	$1000 + 0 + 20 + 4 = 1024$			

Si nous utilisons couramment la notation décimale c'est une pure coincidence, attribuée par certains au fait que notre capacité de visualiser immédiatement un comptage est limitée à 10, c'est-à-dire au nombre de nos doigts. Une quantité peut être exprimée par n'importe quelle autre notation, en choisissant une valeur quelconque pour la base et en répétant la procédure de construction du nombre. Aristote avait remarqué que la plupart des peuples comptaient par dizaines. On note cependant des restes de base 5 dans plusieurs idiomes (le grec par exemple) et des vestiges d'une base 20, en français notamment. C'est le choix de la base qui détermine le nombre des symboles utilisables.

BASE 10, 10 symboles :
0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

BASE 2, 2 symboles :
0, 1 (ou encore L, H).

BASE 8, 8 symboles :
0, 1, 2, 3, 4, 5, 6, 7.

BASE 16, 16 symboles :
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Nous reviendrons en détail sur les bases 8 et 16. Voyons pour l'instant le système de numération en base 2.

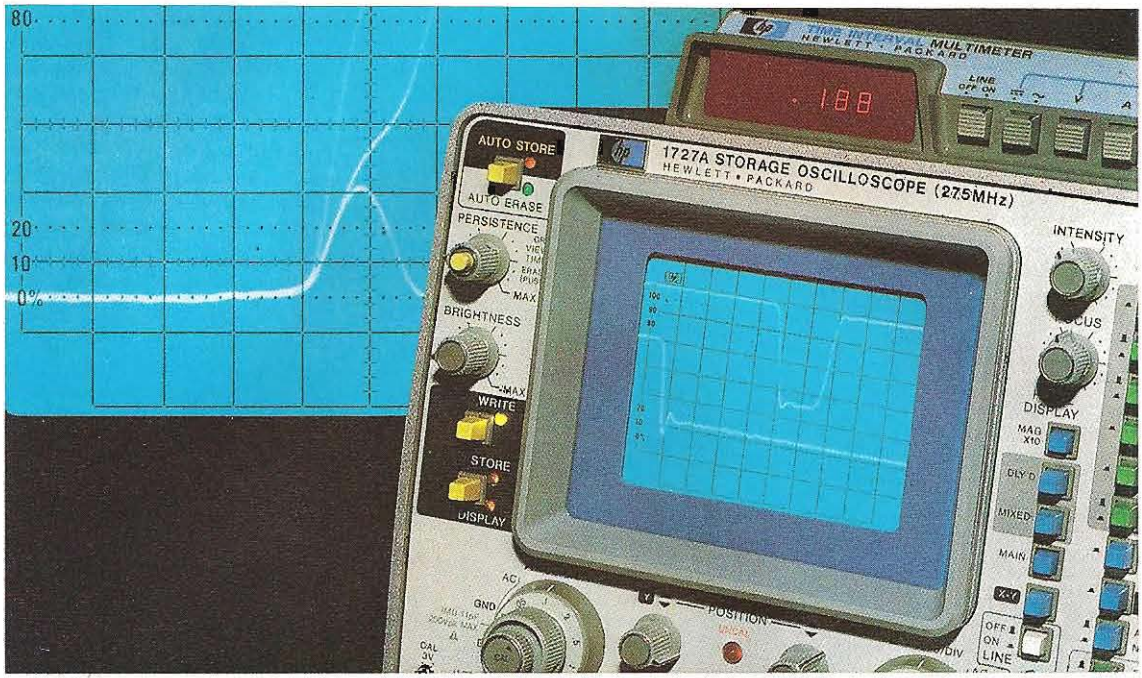
Notation binaire

La notation binaire (à base 2) est positionnelle, comme la notation décimale : si nous accomplissons machinalement les opérations nécessaires à l'écriture et à l'interprétation d'un nombre décimal, pour le système binaire, nous sommes obligés de les développer toutes, mais, par ailleurs, les règles restent les mêmes.
Cette question sera par la suite, approfondie bien au-delà des connaissances de base nécessaires à la programmation, et présente même un intérêt purement informatif : le lecteur acquerra la pratique indispensable grâce aux nombreux exemples qui lui sont présentés dans cet ouvrage.
La notation binaire n'utilise que deux symboles : 0 et 1. Les chiffres occupant les différentes positions vont multiplier les puissances successives de la base 2. La procédure d'interprétation d'un nombre binaire (ici le nombre 1011) est décrite par le tableau suivant, analogue à celui que nous venons d'utiliser pour la base 10 :

Colonne	d	c	b	a
Nombre (chiffres)	1	0	1	1
Puissances de la base	2^3	2^2	2^1	2^0
Poids des positions	8	4	2	1
Valeur des chiffres (chiffre x poids)	8	0	2	1

Valeur du nombre $8 + 0 + 2 + 1 = 11$ **(décimal)**

*Un nombre quel qu'il soit élevé à la puissance 0 donne pour résultat 1. Par conséquent, $2^0 = 1$



Hewlett Packard

Oscilloscope Hewlett Packard montrant la forme des ondes de signaux numériques.

Prenons un autre exemple, avec interprétation du nombre binaire de 8 chiffres 11111111 :

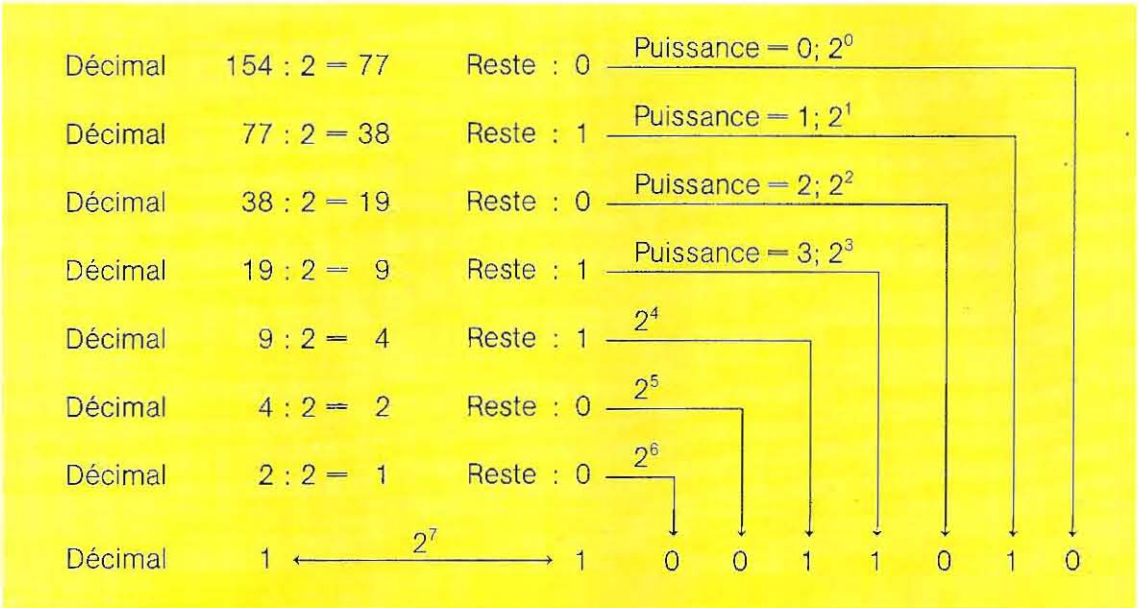
Colonne	h	g	f	e	d	c	b	a
Nombre (chiffres)	1	1	1	1	1	1	1	1
Puissances de la base	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Poids des positions	128	64	32	16	8	4	2	1
Valeur des chiffres (chiffre x poids)	128	64	32	16	8	4	2	1

Valeur du nombre 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255 **(décimal)**

Comme nous l'expliquerons plus loin, le groupe de 8 chiffres binaires (dont la valeur décimale maximum est 255) est l'unité de base (octet) de certains ordinateurs. Tout nombre est ici traduit par une série de 1 et de 0. Le système binaire exige toutefois une quantité importante de symboles pour indiquer une valeur numérique ; il est donc peu pratique pour l'utilisateur de la machine. Aussi a-t-on introduit deux autres systèmes positionnels : l'**octal** et l'**hexadécimal.**
Ces systèmes permettent d'utiliser une représentation adaptée au mode de fonctionnement de l'ordinateur tout en étant compacte, et donc plus commode que le binaire. Ils donnent la possibilité d'écrire un nombre en évitant de recourir à une longue série de 1 et de 0. Pour montrer les différences qui existent entre les systèmes binaire, octal et hexadécimal, on peut dire que le rapport est le même qu'entre l'expression du temps en minutes et en heures. 8 heures ou 480 minutes sont la même chose ; la différence est une expression plus ou moins compacte et plus ou moins pratique. De même, la valeur d'un nombre ne change pas, qu'on l'exprime en binaire, en octal ou en hexadécimal.
L'ordinateur, quant à lui, utilise toujours et uniquement le système binaire, appelé aussi langage machine. N'oublions pas que :
UN DISPOSITIF NUMERIQUE EXECUTE SES CALCULS EN BASE 2.

Nous avons établi comment calculer la valeur décimale d'un nombre binaire. Voyons à présent l'opération inverse : comment obtenir un nombre binaire en partant du décimal. On divise le nombre décimal par 2 jusqu'à obtenir 1. Pour chaque division, on inscrit 0 s'il n'y a pas de reste, 1 s'il y a un reste.

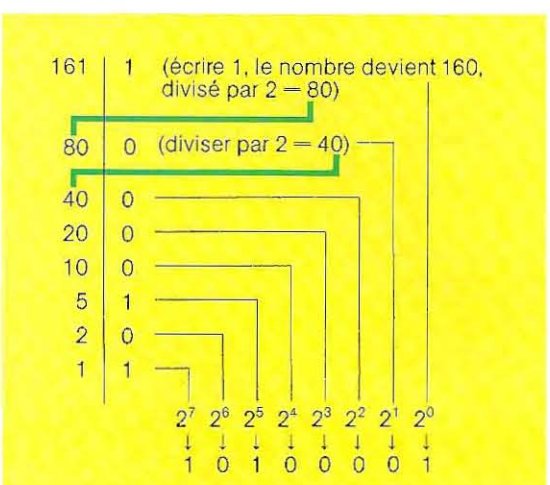
Considérons par exemple le nombre décimal 154 :



En reportant les restes, nous obtenons 10011010, soit l'équivalent binaire de 154 en décimal.
On peut accélérer l'opération en procédant comme suit :

A / si le nombre est pair, écrire 0
B / Si le nombre est impair, écrire 1 et déduire 1 du nombre
C / Diviser par 2
D / Si le résultat n'est pas 1, repartir de A avec la valeur obtenue.

Par exemple, en prenant le nombre décimal 161, nous avons :



Pour vérifier le résultat, transformons le nombre binaire 10100001 en décimal :

1 0 1 0 0 0 0 1
2^7 + 2^5 + 2^0

128 + 32 + 1
Le total nous donne 161.

Notation hexadécimale

Nous avons dit un peu plus haut que les ordinateurs utilisent couramment des nombres binaires composés de 8 chiffres ou multiples de 8 (16, 32). Pour indiquer ces nombres, il faut 8 symboles (ou respectivement 16, 32) pouvant valoir chacun 0 ou 1 ; la notation devient trop longue.
Pour plus de commodité, on a donc introduit une forme de numération plus compacte : la numération **hexadécimale**, c'est-à-dire à base 16. A l'intérieur de la machine, la représentation reste binaire. Le système hexadécimal n'est qu'un système de numération plus commode pour nous.
Avec ce système, les unités valent de 0 à 15 (comme les unités décimales valent de 0 à 9) ; le groupe supérieur commence à 16, c'est-à-dire à la valeur de la base (comme les dizaines du système décimal commencent à 10).
Considérons par exemple le nombre binaire 10100001 (qui correspond, nous l'avons vu, à 161 décimal) ; nous savons que chaque symbole (0 ou 1) représente une puissance de 2.

Reportons le nombre binaire sur un tableau indiquant les puissances de 2 correspondantes :

	7	6	5	4	3	2	1	0
Exposant								
Valeur de la puissance	128	64	32	16	8	4	2	1
Nombre binaire	1	0	1	0	0	0	0	1
		X				Y		

Considérons la moitié droite du nombre binaire 10100001, c'est-à-dire 0001 (il comporte les exposants 3, 2, 1, 0 et les valeurs 8, 4, 2, 1).
Avec ce nombre binaire, on peut écrire au maximum (en donnant la valeur de tous les chiffres qui le composent) :

- 1111 binaire = 8 + 4 + 2 + 1 = 15 décimal.

Pour écrire le nombre décimal 16, il faut passer au groupe suivant. En résumé :

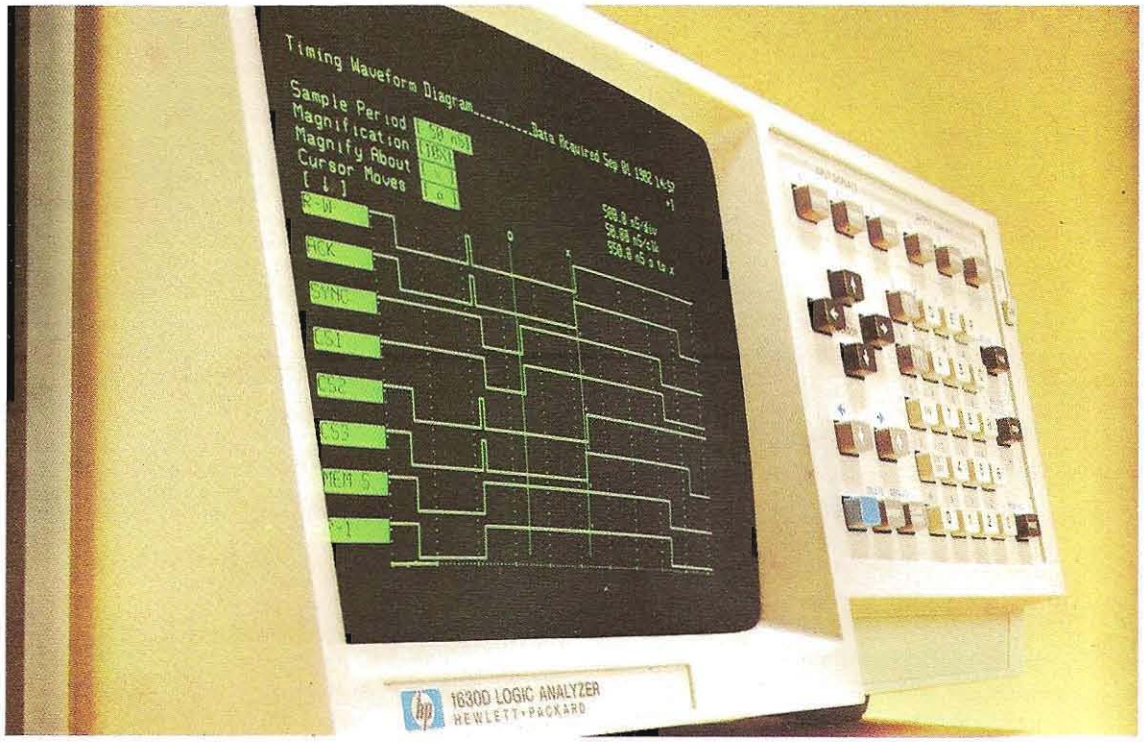
Un nombre binaire de 4 chiffres permet d'écrire au maximum 15. Pour écrire la valeur supérieure 16, il faut passer au groupe de 4 immédiatement supérieur (groupe X du tableau).
On peut donc en conclure qu'en regroupant les nombres binaires par 4, on obtient un système de numération à base 16, appelé hexadécimal. En fait :

- **Le groupe (Y) des unités hexadécimales permet d'écrire jusqu'à 15 (comme les unités décimales permettent d'écrire jusqu'à 9, c'est-à-dire une unité de moins que la base 10).**
- **Le groupe immédiatement supérieur (X) commence par la valeur de la base 16.**

En fait, la valeur minimale (en dehors de 0) que l'on peut écrire dans le groupe X est 0001, qui vaut justement 16, de même que la valeur minimale du groupe des dizaines est 10.

En notation hexadécimale, chaque groupe de 4 chiffres binaires a des valeurs comprises entre 0 et 15.
Les exposants de 2 sont 0, 1, 2, 3 et les puissances sont 1, 2, 4, 8, qui au total donnent 15. Ecrivons à présent le nombre binaire 10100001 sous cette nouvelle forme :

	second chiffre hexadécimal				premier chiffre hexadécimal			
Exposant	3	2	1	0	3	2	1	0
Valeur de la puissance	8	4	2	1	8	4	2	1
Nombre binaire	1	0	1	0	0	0	0	1
	X				Y			



Hewlett Packard

L'analyseur d'état logique HP 1630D visualise les niveaux logiques des circuits numériques.

Le calcul de Y et de X en hexadécimal est immédiat :

Y = 1 puisqu'il ne comporte que la puissance $2^0 = 1$

X = 10 puisqu'il comporte les puissances 2^3 et $2^1 = 8 + 2 = 10$.

Le nombre total doit donc s'écrire :

	X	Y
XY	10	1

Cette formulation n'est pas tout à fait claire, car on ne peut pas savoir a priori si le nombre 10 appartient entièrement au groupe **X** ou si le groupe **X** vaut 0 et s'il existe un autre groupe **Z** valant 1.

Pour pallier cet inconvénient, on a décidé de symboliser les chiffres hexadécimaux de valeur supérieure à 9 par des lettres, selon le tableau suivant :

Hexadécimal	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Symbole	0 1 2 3 4 5 6 7 8 9 A B C D E F

La valeur 10 est donc indiquée par le symbole A. Le nombre binaire 10100001 devient donc :

1 0 1 0	0 0 0 1
A	1

En résumé :

Valeur décimale : 161
Correspondant binaire : 10100001
Correspondant hexadécimal : A1

Comme on le voit, la notation hexadécimale est beaucoup moins encombrante.

Prenons pour second exemple le nombre décimal 215. Transformons-le en binaire :

215	1
107	1
53	1
26	0
13	1
6	0
3	1
1	1

215 = 1 1 0 1 0 1 1 1

Préparons à présent un tableau de traduction en hexadécimal.
Sachant qu'un chiffre hexadécimal regroupe 4 chiffres binaires, pour pouvoir écrire le nombre 11010111 il nous faut deux positions hexadécimales :

Valeur de la puissance	2e chiffre hexadécimal				1er chiffre hexadécimal			
	8	4	2	1	8	4	2	1
Nombre binaire	1	1	0	1	0	1	1	1
	8 + 4 + 1 = 13, c.-à-d. = **D**				4 + 2 + 1 = 7, c.-à-d. = **7**			

Nous avons donc :
- décimal : 215
- hexadécimal : **D7**
- binaire : 11010111

En guise de vérification, procédons en sens inverse. Nous allons tirer de la valeur hexadécimale **D7** son correspondant décimal :
- La valeur **7** représente les unités, le chiffre reste donc inchangé.
- La valeur **D** = 13 est le premier multiple de la base, qu'il faut donc multiplier par la base 16.

Soit (13 x 16) + 7 = 208 + 7 = 215 décimal.

Notation octale

C'est un procédé très comparable à l'hexadécimal. On divise le nombre binaire en groupes de 3 chiffres ayant les valeurs 4, 2, 1 (exposants 2, 1, 0). Pour le nombre binaire 10100001 = 161 décimal, on obtient :

	3e chiffre octal			2e chiffre octal			1er chiffre octal		
Exposant	2	1	0	2	1	0	2	1	0
Valeur de la puissance	4	2	1	4	2	1	4	2	1
Nombre binaire		1	0	1	0	0	0	0	1
Valeur octale		**2**			**4**			**1**	

On obtient les correspondances suivantes :
Décimal : 161
Binaire : 10100001
Hexadécimal : **A1**
Octal : **241**
Comme nous l'avons vu dans le cas de la notation hexadécimale, pour traduire le nombre octal **241** en décimal il faut appliquer les règles suivantes :
L'unité reste inchangée.
Le chiffre suivant doit être multiplié par 8^1.
Le suivant par 8^2, et ainsi de suite.
On obtient :

Octal	Transformation	Décimal
1	—	1
4	x 8	32
2	x64	128
	Résultat	161

En résumé :

- **Les circuits numériques fonctionnent en arithmétique à base 2.**
- **Un nombre binaire est représenté par une série de 1 et de 0.**
- **Chaque symbole représente le multiplicateur de la puissance correspondante de 2.**
- **La valeur décimale d'un chiffre hexadécimal s'obtient en le multipliant par la puissance correspondante de 16.**
- **En notation octale, la base est 8.**
- **La valeur décimale d'un chiffre octal s'obtient en le multipliant par la puissance correspondante de 8.**

Opérations en notation binaire

Nous avons dit que pour plus de commodité, on a introduit les notations octale (base 8) et hexadécimale (base 16). Mais l'ordinateur n'utilise pas directement ces notations : tous ses calculs se font en binaire.

Lorsqu'on écrit un programme, il faut parfois contrôler « à la main » les résultats de certaines instructions données à la machine, il est donc essentiel de connaître la méthode qui permet d'effectuer des additions et des soustractions en base 2.

Nous verrons que ces contrôles n'ont pas pour objet de vérifier l'exactitude des calculs mathématiques (la machine les exécute beaucoup plus rapidement et plus précisément que nous), mais bien d'obtenir des résultats particuliers dans les opérations logiques. Nous traiterons aussi de la signification et des fonctions des opérations logiques. Pour l'instant, nous allons simplement illustrer ce concept.

Pour pouvoir dialoguer avec nous, un ordinateur doit posséder un tableau de conversion entre notre alphabet et les symboles qu'il utilise à la place de nos lettres et de nos chiffres. Il est facile d'en déduire que n'importe quel nombre est représenté à l'intérieur de l'ordinateur par une série de symboles 1 et 0.

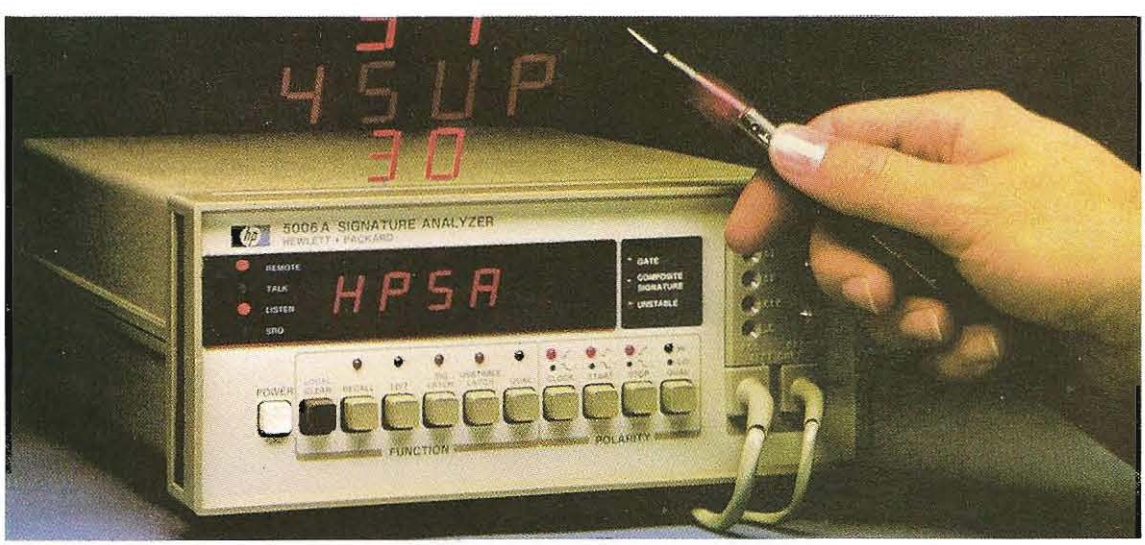
La machine ignorant tout autre symbole, il est également indispensable de traduire les lettres de l'alphabet.

Autrement dit, tous les symboles graphiques que nous utilisons sont représentés dans l'ordinateur en binaire. Si l'utilisateur veut entrer dans la mémoire de la machine la lettre A, cette lettre sera convertie en un nombre binaire.

Pour obtenir le code de la lettre B il suffira d'ajouter 1 au code de la lettre A, et ainsi de suite pour tout l'alphabet.

Cette méthode, nous le comprendrons mieux par la suite, permet de représenter la totalité de nos symboles.

Analyseur de signature HP 5006 A : il teste le fonctionnement d'un circuit.



Hewlett Packard

TEST 1

Chacun des sujets traités sera suivi d'un test de contrôle. Le lecteur pourra vérifier l'exactitude de ses réponses par comparaison avec les solutions fournies plus loin. Si la moitié au moins de vos réponses est exacte, vous pouvez poursuivre sans difficulté la suite de votre lecture. Si le résultat est inférieur, nous vous conseillons de relire le chapitre entier, sauf s'il s'agit de questions particulières, qui ne sont pas indispensables à la compréhension de la suite (cette indication sera donnée avec la solution).

1 / Lesquelles de ces affirmations sont vraies :

- ☐ a - L'ordinateur utilise la notation binaire qui permet une interprétation facile par l'opérateur.
- ☐ b - La notation octale a été introduite pour donner une représentation plus compacte des nombres binaires.
- ☐ c - Un chiffre hexadécimal peut exprimer au maximum la valeur 15.
- ☐ d - Le chiffre 8 n'existe pas en notation octale (comment l'écrit-on en octal ?)

2 / Traduire les nombres binaires suivants en valeurs décimales correspondantes :

1001 - 11 - 1 - 10101 - 11001 - 111

3 / Traduire les nombres binaires suivants en valeurs octales correspondantes :

110111 - 11 - 101010 - 111111

4 / Traduire les nombres binaires suivants en val. hexadécimales correspondantes :

11011111 - 11011011 - 11111111 - 111111

5 / Traduire en binaire les nombres décimaux suivants :

10 - 100 - 1000

6 / Les quatre nombres binaires :

a - 0001 b - 0010 c - 0100 d - 1000

s'obtiennent en déplaçant le symbole 1 d'un cran vers la gauche.

- ☐ Quelle est, pour chacun d'eux, la valeur décimale correspondante ?
- ☐ En observant les traductions en valeur décimale, à quelle opération arithmétique peut-on assimiler le déplacement du symbole 1 vers la gauche ?
- ☐ Quelle opération arithmétique fait-on en déplaçant le symbole 1 vers la droite ?

7 / Lesquelles de ces affirmations sont vraies :

- ☐ a - En notation hexadécimale, les nombres supérieurs à 9 sont indiqués par des lettres de l'alphabet.
- ☐ b - En notation hexadécimale, on ne peut indiquer de nombre supérieur à 9.
- ☐ c - Un circuit numérique ne peut reconnaître d'autre état que la présence ou l'absence de signal.
- ☐ d - Le front de montée d'une impulsion correspond à la zone dans laquelle le signal passe du niveau bas au niveau haut.

8 / Traduire en décimal les nombres octaux suivants :

7 - 70 - 700 - 111

9 / Traduire en décimal les nombres hexadécimaux suivants :

7 - 70 - A - AA - AAA - 3FB

10 / Quelles sont les valeurs décimales les plus fortes pouvant être représentées par un nombre binaire de :

2 chiffres (11) - 4 chiffres (1111) - 8 chiffres (11111111)

Les solutions se trouvent pages 76 et 77.

La révolution informatique

L'information, base de la plus grande partie de toute l'activité intellectuelle de l'homme, et l'information traitée, c'est-à-dire l'informatique, sont destinées à jouer un rôle de plus en plus grand dans notre société.
Du moins, pour autant que nous puissions en juger aujourd'hui.
En effet, la certitude en ce domaine n'est pas absolue, mais elle est pourtant déjà assez bien étayée pour servir de base à notre raisonnement. C'est un peu comme face à un iceberg: malgré les dimensions relativement réduites de la partie visible par rapport à celles de la partie cachée, l'iceberg nous paraît suffisamment impressionnant pour que nous puissions imaginer l'importance de ce que nous ne voyons pas. De même, nous pouvons affirmer que l'ordinateur et la micro-électronique seront les protagonistes et les éléments moteurs de la transformation des années à venir. Les plus importantes de ces nouvelles sciences ont été expérimentées, étudiées et portent déjà des noms. On parle en effet de télématique, de robotique et de bureautique pour désigner les principaux domaines d'application de l'informatique.
L'explosion de l'électronique concernera tous les secteurs de la vie quotidienne à la maison, au bureau et à l'usine. Le changement ne sera ni brutal, ni automatique; il s'effectuera à travers une évolution culturelle portant à la connaissance généralisée, à l'informatique de masse.

Réactualisées en permanence, les informations sont à la disposition de tous les particuliers possédant un poste de télévision et un téléphone, ainsi qu'un décodeur s'il n'est pas déjà intégré dans le téléviseur. (Télématique, Vélizy.)



Carlos Santos / Rapho

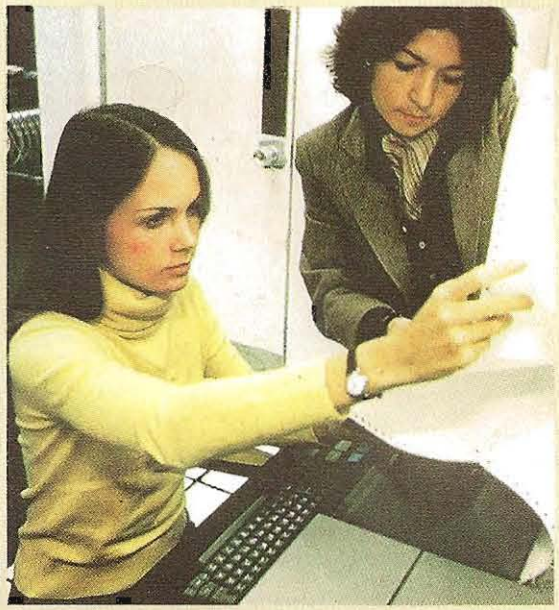
* * *

Le terme ***télématique*** *est issu de la fusion de deux autres mots: télécommunication et informatique. Il s'agit donc de la technologie de la communication reliée à l'informatique, ce qui représente le premier grand stade de la transformation qui nous intéresse.*
Le téléphone, c'est-à-dire le terminal le plus répandu – on comptait 500 millions de postes installés dans le monde à la fin de 1980, et 20 millions d'abonnés en France à la fin de 1983 –, le téléviseur et l'ordinateur individuel qui peut les relier l'un à l'autre, tels sont les appareils par lesquels se fera le changement. Ce dernier sera possible grâce à l'amélioration des communications: par câbles, avec l'introduction de matériaux nouveaux comme les fibres optiques; par voie hertzienne, avec l'utilisation des satellites à l'échelle planétaire. Encore une fois c'est l'information numérisée, c'est-à-dire traduite en code binaire, qu'elle soit graphique ou alphanumérique, qui sera l'instrument de la révolution à laquelle nous pouvons nous préparer. Le progrès se mesurera par la vitesse de transmission de l'information, la capacité de stockage de cette information dans les banques de données, ainsi que la possibilité de la récupérer et de la transmettre à grande distance.
Le terme télématique désigne toute une série de services supportés par les réseaux de télécommunication classiques ou nouveaux, qui mettront à la disposition des utilisateurs les informations les plus diverses contenues dans les banques de données (ensembles d'informations organisées). Cela laisse entrevoir la puissance dynamique et de développement de la télématique à l'égard des divers secteurs et des activités industrielles. Bon nombre de systèmes traditionnels sont appe-

lés à changer, tandis que d'autres naîtront sous la poussée des technologies nouvelles. La télématique agit d'ores et déjà comme facteur de développement pour les organisations de télévision, de téléphone et de communication qui préparent les structures nécessaires pour accueillir et transporter, par câbles ou par voie hertzienne, les informations vers les utilisateurs.
Par ailleurs, le développement né de la télématique intéressera toutes les structures de création, d'organisation et de fourniture d'informations. Les maisons d'édition et de presse, qui assument traditionnellement ce rôle, verront s'ouvrir un autre débouché pour leurs informations ; bien d'autres organisations économiques (instituts de crédits ou bourses, par exemple) bénéficieront de la télématique qui leur permettra de mieux distribuer l'information qu'elles possèdent. La publicité ne restera pas étrangère à cette nouvelle méthode de communication, et la télématique lui offrira sans aucun doute un nouveau moyen d'atteindre le destinataire du message publicitaire. Tout cela nous conduit enfin à une vaste gamme de machines nouvelles : téléviseurs améliorés, ordinateurs individuels, téléphones, instruments et appareils de télécommunications, etc., avec encore d'autres possibilités de développement pour la production industrielle.
Les premières applications de la télématique commencent déjà à être diffusées dans le monde.

* * *

La **robotique** n'appartient plus à la science-fiction. Les robots sont parmi nous. Dans les secteurs industriels de pointe, la recherche, l'espace et le fond des océans, ils remplacent l'homme pour toutes les tâches répétitives, fatigantes ou dangereuses. Leur emploi permet de réduire les coûts de production, rend accessibles des activités inimaginables jusque-là et augmente la compétitivité des procédés. Dans l'avenir, nous verrons les robots mécaniques, déjà dotés aujourd'hui d'organes sensoriels et dirigés par des systèmes électroniques et des microprocesseurs, se multiplier et remplir les fonctions les plus diverses.
Ces machines comportent en général un bras mécanique accomplissant des mouvements



Marka

Le Télex utilise le réseau téléphonique ; il comprend un clavier et une imprimante.

dans toutes les positions et une main servant à prendre et à porter des objets. Souvent, les robots sont également dotés d'organes sensoriels qui leur confèrent la vue et le toucher, voire la parole. Une autre caractéristique des robots industriels est qu'ils s'adaptent à de nombreux emplois : ils mémorisent les tâches à accomplir et en apprennent de nouvelles, ce qui présuppose, aux côtés d'une structure mécanique, une structure électronique, c'est-à-dire un véritable mini-ordinateur. Les champs d'applications de la robotique industrielle sont innombrables. Le grand public connaît surtout ceux de l'industrie automobile mais ils sont présents dans de multiples autres secteurs où ils prennent la place de l'homme.

* * *

Une fois de plus, l'histoire de l'information voit naître un nouveau terme par fusion de deux mots et de deux concepts, pour désigner un groupement empirique d'origine récente. Ce terme d'origine française désigne des techniques et des technologies relatives à l'automatisation des travaux de bureau : bureau plus informatique ont donné **bureautique**, dont l'usage se répand partout aujourd'hui.

(D'après A SCUOLA CON IL COMPUTER, Egidio Pentiraro. Laterza, 1983).

Addition binaire

Pour réaliser une addition en arithmétique binaire, il suffit de ne pas oublier que la base de cette numération est 2 : quand le résultat de l'addition donne 2, il faut écrire 0 et retenir 1, exactement comme on le fait dans une addition en décimal quand le résultat est égal ou supérieur à 10.

Additionnons par exemple le chiffre binaire 1 à lui-même :

$$\begin{array}{r} 1+ \\ 1= \\ \hline 10 \end{array}$$

Le résultat de 1 + 1 est 2 ; on écrit donc 0 et on retient 1.

Pour vérifier le résultat, transformons chacun de ces chiffres en décimal :

$$\begin{array}{rcl} 1 & \text{binaire} = 1 & \text{décimal} \\ + & + & \\ 1 & \text{binaire} = 1 & \text{décimal} \\ \hline \text{Résultat :} & 10 \text{ binaire} = 2 & \text{décimal} \end{array}$$

Additionnons à présent les deux nombres binaires :

$$\begin{array}{r} 1\,0\,1\,1\,0\,1 \\ 1\,0\,1\,1\,0 \end{array}$$

Avant d'effectuer cette addition, transformons les deux nombres binaires par leurs équivalents décimaux (nous utiliserons ces valeurs pour vérifier le résultat) :

$1\,0\,1\,1\,0\,1 = 2^0 + 2^2 + 2^3 + 2^5 = 1 + 4 + 8 + 32 = 45$ décimal

$1\,0\,1\,1\,0 = 2^1 + 2^2 + 2^4 = 2 + 4 + 16 = 22$ décimal

Le résultat de l'addition binaire doit donc être l'équivalent de 45 + 22 = 67 décimal.

Pour examiner cette addition binaire, mettons les deux nombres en colonnes à partir de la droite :

Colonne	g	f	e	d	c	b	a
1er terme		1	0	1	1	0	1
2d terme			1	0	1	1	0
	1	0	0	0	0	1	1
Retenues		1	1	1	1		

A partir de la **colonne A** : les deux termes sont 1 et 0, le résultat est 1, il n'y a pas de retenue.

Colonne B : les deux termes sont 0 et 1, résultat = 1, retenue = 0.

Colonne C : les deux termes sont 1 et 1, résultat = 0, retenue = 1.

1 + 1 = 0 avec retenue de 1.

De même, en décimal on obtient :

8 + 2 = 10 c'est-à-dire 0 avec retenue de 1.

La retenue s'ajoute au résultat de la colonne D.

Colonne D : les termes sont 1 et 0, résultat = 1. Ajoutons la retenue précédente (1) : 1 + 1 = nouv. résultat = 0. Nouvelle retenue = 1.

Colonne E : les termes sont 0 et 1, résultat = 1. Ajoutons la retenue précédente (1) : 1 + 1 = nouv. résultat : 0. Nouvelle retenue = 1.

Colonne F : les termes sont 1 et 0, résultat = 1. Ajoutons la retenue précédente (1) : 1 + 1 = nouv. résultat = 0. Retenue = 1.

Colonne G : elle ne comporte aucun terme, on y inscrit donc la dernière retenue (1). Le résultat est :

$1000011 = 2^0 + 2^1 + 2^6 = 1 + 2 + 64 = 67$ décimal.

On peut effectuer de la même façon les additions à base 8 et à base 16.

Dans ces notations, il faut se souvenir que la retenue existe quand la somme est respectivement égale (ou supérieure) à 8 et à 16.

Prenons pour exemple la somme des deux mêmes nombres en notation octale.

Transformons d'abord les deux termes en notation octale :

Valeur	4	2	1		4	2	1
Nombre binaire à transformer	1	0	1		1	0	1
Correspondant octal	5				5		
	(4 + 1)				(4 + 1)		

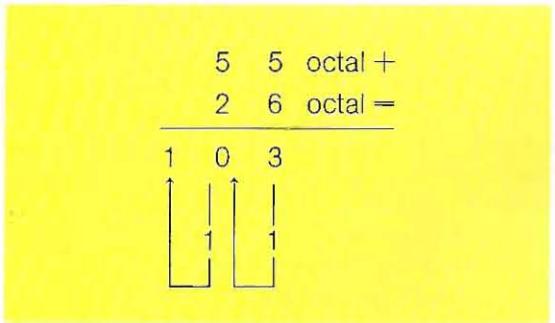
Valeur	4	2	1		4	2	1
Nombre binaire à transformer	*	1	0		1	1	0
Correspondant octal	2				6		

$1\,0\,1\,1\,0\,1 = 55$ octal

$1\,0\,1\,1\,0 = 26$ octal

*L'absence de symbole en première position équivaut à un 0. Le nombre binaire 10110 équivaut donc à 010110.

La somme en octal devient :



Le résultat (103 octal) s'obtient comme suit :

- En partant de la droite, 5 + 6 = 11, de ce fait (en retirant la base on obtient 3 et une retenue = 1) dans la seconde colonne nous avons : 5 + 2 + retenue (1) = 8 en retirant la base, on obtient 0 et retenue = 1.

Dans la dernière colonne, il suffit d'écrire la retenue puisqu'il n'existe aucun terme.
En guise de vérification, transformons le résultat en décimal :

Le nombre
103 octal signifie :



ou de manière plus ramassée :
$103 \text{ octal} = 3 \times 8^0 + 1 \times 8^2 = 3 + 64 = 67$

Soustraction binaire

La différence de deux nombres en notation binaire s'effectue selon les mêmes règles que la soustraction décimale mais, pour des raisons que nous verrons mieux par la suite, nous allons utiliser la « méthode du complément ».
Le complément d'un nombre binaire s'obtient en remplaçant le symbole 1 par le symbole 0 et vice versa. Le complément de 100101 est donc 011010.
La somme du nombre de départ et du complément est un nombre binaire composé uniquement de symboles 1 :

nombre	**1 0 0 1 0 1 +**
complément	0 1 1 0 1 0
somme	1 1 1 1 1 1

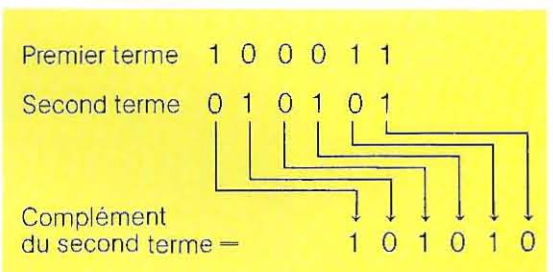
En ajoutant 1 à la somme on obtient une série continue de 0 : 111111 + 1 = 1000000.
En appliquant la méthode du complément, la différence entre deux nombres binaires s'obtient comme suit :

A / Compléter le plus petit nombre pour parvenir au même nombre de chiffres que le premier terme.
B / Additionner le premier terme et le complément du plus petit, en excluant du résultat l'éventuel chiffre 1 qui dépasse à gauche la longueur du premier terme.
C / Ajouter 1*.

Effectuons l'opération suivante :
Premier terme Second terme
1 0 0 0 1 1 − 1 0 1 0 1.
En décimal, cette différence est : 35 − 21 = 14.

Etape A

Complétons le second terme, plus petit que le premier, à l'aide de symboles 0 :



Etape B

On effectue la somme du premier terme (100011) et du complément du second terme (101010) :

Premier terme		**1 0 0 0 1 1**
Complément du second terme		**1 0 1 0 1 0**
Somme	**1**	**0 0 1 1 0 1**

Nous exclurons de la somme le symbole 1 qui dépasse la longueur du premier terme.

Etape C

On ajoute 1 à la somme :

somme	**0 0 1 1 0 1 +**
	1
Résultat	**0 0 1 1 1 0** = 14 décimal

La procédure ainsi décrite peut paraître beaucoup plus lourde qu'elle ne l'est en réalité. En

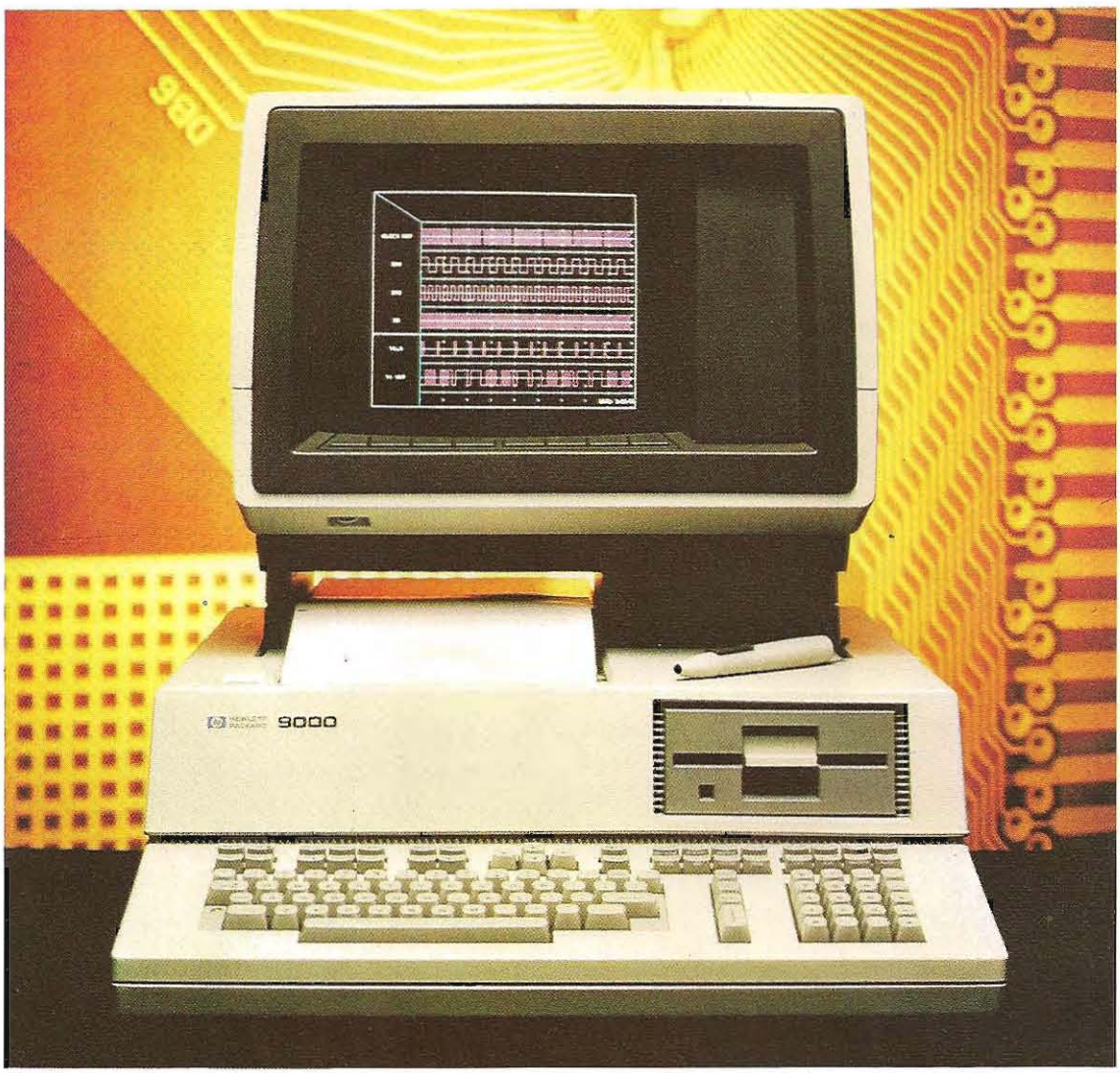
*Exclure les chiffres qui dépassent comme B.

fait, avec un minimum de pratique, la plupart des étapes que nous venons d'expliquer se font de tête.
Voici un autre exemple, exposé de manière abrégée.
Soit l'opération :
226 − 174 = 152 décimal =
= **1 1 1 0 0 0 1 0 − 1 0 1 0 1 1 1 0.**
En faisant le complément du second terme **(1 0 1 0 1 1 1 0)** puis la somme, on a :

Premier terme	**1 1 1 0 0 0 1 0**
Complément du 2[d] terme	**0 1 0 1 0 0 0 1**
Résultat de l'addition **1**	**0 0 1 1 0 0 1 1** +
	1
	0 0 1 1 0 1 0 0 =
	= 52 décimal

Pour compléter cet exposé, le listing (ou liste) 1 de la page 63 à la page 66 donne en partie les équivalences binaires, octales (OCT), et hexadécimales (HEX) des nombres décimaux de 1 à 255.
Le listing 2, pages 66 et 67, donne les mêmes équivalences pour les nombres compris entre 31100 et 31150.
Les données des colonnes OCT (octal) et HEX (hexadécimal) ont été tournies par l'ordinateur avec une instruction unique qui permet d'obtenir le résultat en quelques millièmes de seconde. Essayez donc de traduire le nombre 31142 : même avec beaucoup de rapidité et d'attention, il vous faudra plusieurs minutes.

Ordinateur HP 9000 montrant le comportement des signaux numériques d'un circuit logique.



Hewlett Packard

LISTING 1

DECIMAL	BINAIRE	OCT	HEX
1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	1	1
2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	2	2
3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	3	3
4	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	4	4
5	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	5	5
6	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	6	6
7	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1	7	7
8	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	10	8
9	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	11	9
10	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0	12	A
11	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1	13	B
12	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0	14	C
13	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1	15	D
14	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0	16	E
15	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	17	F
16	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	20	10
17	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1	21	11
18	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0	22	12
19	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1	23	13
20	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0	24	14
21	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1	25	15
22	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0	26	16
23	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1	27	17
24	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0	30	18
25	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1	31	19
26	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0	32	1A
27	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1	33	1B
28	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0	34	1C
29	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1	35	1D
30	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0	36	1E
31	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1	37	1F
32	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	40	20
33	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	41	21
34	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0	42	22

DECIMAL	BINAIRE	OCT	HEX
35	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1	43	23
36	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0	44	24
37	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1	45	25
38	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0	46	26
39	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1	47	27
40	0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0	50	28
41	0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1	51	29
42	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0	52	2A
43	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1	53	2B
44	0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0	54	2C
45	0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1	55	2D
46	0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0	56	2E
47	0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1	57	2F
48	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	60	30
49	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1	61	31
50	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0	62	32
51	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1	63	33
52	0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0	64	34
53	0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1	65	35
54	0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0	66	36
55	0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1	67	37
56	0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0	70	38
57	0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1	71	39
58	0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0	72	3A
59	0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1	73	3B
60	0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0	74	3C
61	0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1	75	3D
62	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0	76	3E
63	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1	77	3F
64	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	100	40
65	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1	101	41
66	0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0	102	42
67	0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1	103	43
68	0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0	104	44
69	0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1	105	45

70	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	106	46
215	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	327	D7
216	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	330	D8
217	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	331	D9
218	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	332	DA
219	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	333	DB
220	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	334	DC
221	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	1	335	DD
222	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	336	DE
223	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	337	DF
224	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	340	E0
225	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	341	E1
226	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	342	E2
227	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	343	E3
228	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	344	E4
229	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	345	E5
230	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	346	E6
231	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	347	E7
232	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	350	E8
233	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1	351	E9
234	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	352	EA
235	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	353	EB
236	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	354	EC
237	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1	355	ED
238	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	356	EE
239	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	357	EF
240	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	360	F0
241	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	361	F1
242	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	362	F2
243	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	363	F3
244	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	364	F4
245	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	1	365	F5
246	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	366	F6
247	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	367	F7

DECIMAL	BINAIRE	OCT	HEX
248	0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0	370	F8
249	0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1	371	F9
250	0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0	372	FA
251	0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1	373	FB
252	0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0	374	FC
253	0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1	375	FD
254	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0	376	FE
255	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	377	FF

LISTING 2

DECIMAL	BINAIRE	OCT	HEX
31100	0 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0	74574	797C
31101	0 1 1 1 1 0 0 1 0 1 1 1 1 1 0 1	74575	797D
31102	0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0	74576	797E
31103	0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1	74577	797F
31104	0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0	74600	7980
31105	0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1	74601	7981
31106	0 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0	74602	7982
31107	0 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1	74603	7983
31108	0 1 1 1 1 0 0 1 1 0 0 0 0 1 0 0	74604	7984
31109	0 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1	74605	7985
31110	0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0	74606	7986
31111	0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1	74607	7987
31112	0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 0	74610	7988
31113	0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1	74611	7989
31114	0 1 1 1 1 0 0 1 1 0 0 0 1 0 1 0	74612	798A
31115	0 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1	74613	798B

31116	0	1	1	1	1	0	0	1	1	0	0	0	1	1	0	0	74614	798C
31117	0	1	1	1	1	0	0	1	1	0	0	0	1	1	0	1	74615	798D
31118	0	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	74616	798E
31119	0	1	1	1	1	0	0	1	1	0	0	0	1	1	1	1	74617	798F
31120	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	74620	7990
31121	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0	1	74621	7991
31122	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	74622	7992
31123	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	1	74623	7993
31124	0	1	1	1	1	0	0	1	1	0	0	1	0	1	0	0	74624	7994
31125	0	1	1	1	1	0	0	1	1	0	0	1	0	1	0	1	74625	7995
31126	0	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	74626	7996
31127	0	1	1	1	1	0	0	1	1	0	0	1	0	1	1	1	74627	7997
31128	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	0	74630	7998
31129	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	74631	7999
31130	0	1	1	1	1	0	0	1	1	0	0	1	1	0	1	0	74632	799A
31131	0	1	1	1	1	0	0	1	1	0	0	1	1	0	1	1	74633	799B
31132	0	1	1	1	1	0	0	1	1	0	0	1	1	1	0	0	74634	799C
31133	0	1	1	1	1	0	0	1	1	0	0	1	1	1	0	1	74635	799D
31134	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	0	74636	799E
31135	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1	74637	799F
31136	0	1	1	1	1	0	0	1	1	0	1	0	0	0	0	0	74640	79A0
31137	0	1	1	1	1	0	0	1	1	0	1	0	0	0	0	1	74641	79A1
31138	0	1	1	1	1	0	0	1	1	0	1	0	0	0	1	0	74642	79A2
31139	0	1	1	1	1	0	0	1	1	0	1	0	0	0	1	1	74643	79A3
31140	0	1	1	1	1	0	0	1	1	0	1	0	0	1	0	0	74644	79A4
31141	0	1	1	1	1	0	0	1	1	0	1	0	0	1	0	1	74645	79A5
31142	0	1	1	1	1	0	0	1	1	0	1	0	0	1	1	0	74646	79A6
31143	0	1	1	1	1	0	0	1	1	0	1	0	0	1	1	1	74647	79A7
31144	0	1	1	1	1	0	0	1	1	0	1	0	1	0	0	0	74650	79A8
31145	0	1	1	1	1	0	0	1	1	0	1	0	1	0	0	1	74651	79A9
31146	0	1	1	1	1	0	0	1	1	0	1	0	1	0	1	0	74652	79AA
31147	0	1	1	1	1	0	0	1	1	0	1	0	1	0	1	1	74653	79AB
31148	0	1	1	1	1	0	0	1	1	0	1	0	1	1	0	0	74654	79AC
31149	0	1	1	1	1	0	0	1	1	0	1	0	1	1	0	1	74655	79AD
31150	0	1	1	1	1	0	0	1	1	0	1	0	1	1	1	0	74656	79AE

Un terminal dans le téléviseur

Un simple écran de télévision peut aujourd'hui servir de terminal vidéo à un micro-ordinateur et permettre à tout utilisateur d'accéder à un réseau de transmission de données.

Depuis déjà quelques années nous avons accès à une série limitée d'informations de caractère général, par exemple les conditions et les prévisions météorologiques, l'heure exacte, les conditions de circulation, en composant tout simplement un numéro de téléphone. Mais c'est seulement depuis peu que la qualité du service a fait un bond en avant avec la mise en place des systèmes combinés de traitement et de diffusion de l'information. Aujourd'hui, beaucoup d'utilisateurs de la télévision, en appuyant sur certaines touches de leur appareil de télécommande, peuvent faire apparaître sur l'écran (en surimpression ou en écran complet) une vaste gamme de fiches d'informations touchant les domaines les plus variés. Deux systèmes offrant ce type de services ont été réalisés : le Vidéotex diffusé ou Télétexte, et le Vidéotex interactif ou Télétel. Le premier, réalisé vers 1970, était diffusé par le réseau normal de télévision, le second est transmis par le réseau téléphonique ; dans les deux cas, il faut un décodeur spécial pour visualiser les informations sur l'écran.

L'utilisateur, en même temps qu'il regarde un programme télévisé quelconque, peut, en appuyant sur l'une des touches de sa télécommande, faire apparaître la première page des informations Télétexte disponible à cet instant. Cette première page contient l'index général de la série et il suffit de composer sur le clavier de télécommande le numéro de la page intéressante pour qu'elle apparaisse à l'écran et y demeure jusqu'à ce qu'on la remplace en faisant un nouveau numéro.

Le système Prestel diffuse des informations variées par l'intermédiaire du téléphone.



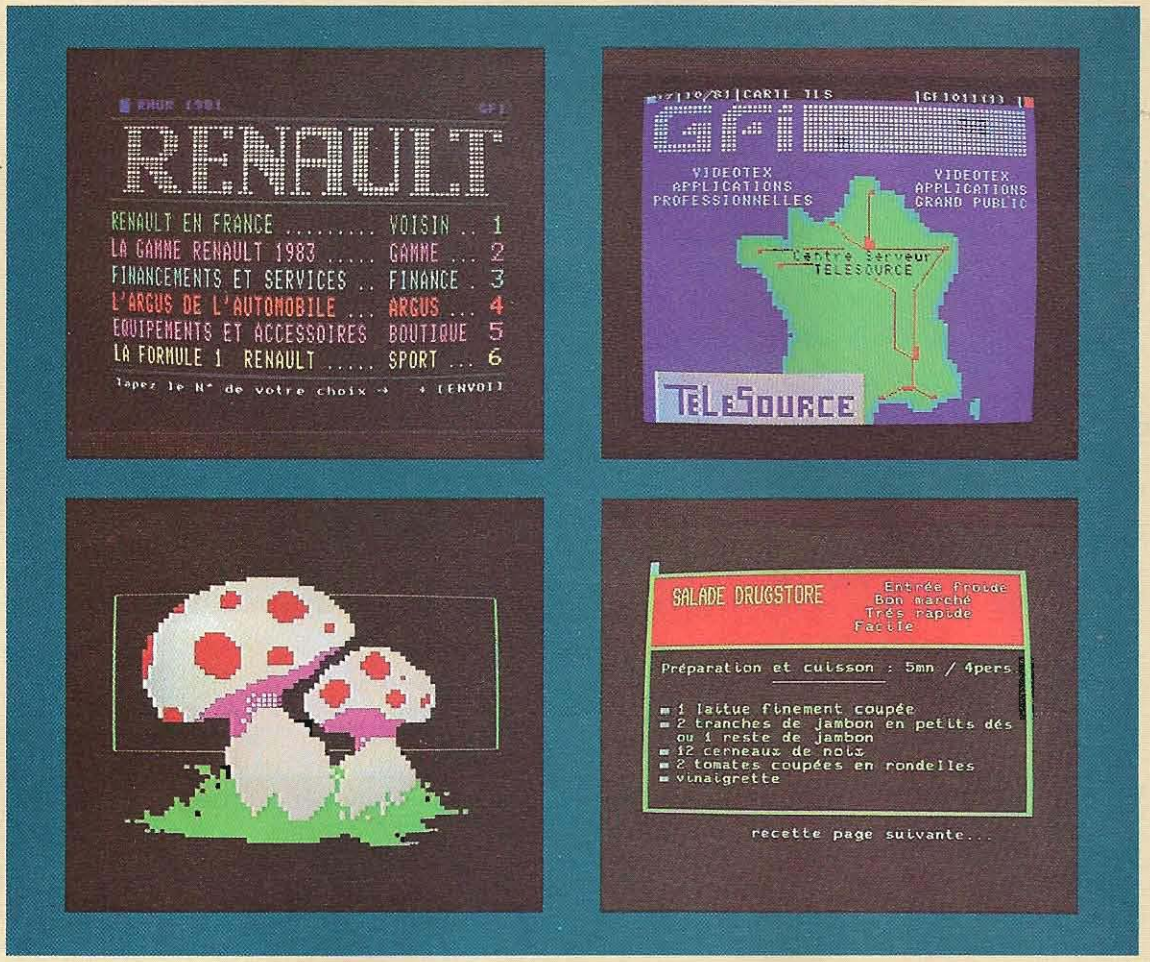
Standard Telephones & Cables Ltd.

Les images de Télétexte sont préparées par des groupes de rédacteurs spécialisés qui entrent régulièrement dans l'ordinateur central (appelé «serveur») les informations nécessaires. Chaque page de transmission se compose de vingt-quatre lignes de texte, de quarante caractères chacune. La composition du texte peut faire appel à sept couleurs (blanc, rouge, bleu, vert, magenta, bleu clair et jaune). Chaque caractère est transmis à l'aide d'un code numérique qui indique aux circuits électroniques chargés de la formation de l'image dans l'appareil récepteur lesquels des soixante-dix points élémentaires constituant sur l'écran l'ensemble destiné à représenter un caractère doivent être éclairés. Une seconde information codée précise la couleur à donner à chaque groupe de points. Un technicien décrirait plus exactement ce mécanisme en remplaçant avec indignation le terme de groupe par celui de «zone» destinée sur l'écran à chacun des caractères mis en œuvre sur une matrice de 7 x 10 pts. Certaines possibilités graphiques permettent en outre d'unifier les zones disponibles pour plusieurs caractères afin de former des lettres et des titres plus grands, et de colorer sélectivement certaines zones pour créer des images simples en deux dimensions.

Pendant l'horaire de transmission qui coïncide en général avec celui des émissions normales des programmes de télévision, toute la série des pages de textes est transmise sans interruption. La transmission d'une page de vingt-quatre lignes imprimées demande un peu moins d'un quart de seconde: intervalle négligeable. Si l'on pense qu'un service Télétexte normal ne comprend jamais moins d'une centaine de pages et que le service tout entier doit être transmis cycliquement, ligne par ligne et page par page, plusieurs minutes peuvent s'écouler entre l'instant où l'utilisateur demande une page déterminée et celui où elle apparaît à l'écran. Pour éviter la disparition de l'image, un circuit de mémoire maintient sur l'écran la page sélectionnée précédemment jusqu'à l'apparition de la nouvelle page. Ce temps mort constitue l'un des principaux inconvénients du système Télétexte, et l'on y a cherché remède. Les perfectionnements qui sont en phase expérimentale permettent déjà de prévoir un large développement des applications du système. Le premier projet est connu sous le nom de Telesoftware ou Télélogiciel. D'après les techniciens, l'addition d'un microprocesseur aux circuits électroniques récepteurs permettra à l'avenir d'accéder directement à l'ordinateur central ou à un réseau d'ordinateurs interconnectés. On pourra de la sorte utiliser la télévision comme un «terminal intelligent» capable d'exécuter des programmes de calcul répondant à nos exigences particulières, par exemple pour la comptabilité et la gestion domestiques. Le second projet concerne un autre service réalisable dans le cadre du système Télétexte, c'est-à-dire l'addition, sur demande expresse du téléspectateur, de sous-titres au programme télévisé normal. D'une part, cela permettrait d'éliminer les difficultés dont souffrent les malentendants, et, de l'autre, cela donnerait à tous la possibilité de suivre les émissions en langue étrangère dans leur version originale.

En France, le Télétexte A2-Antiope (acquisition numérique et télévisualisation d'images organisées en pages d'écriture) est exploité depuis 1978, et fonctionne depuis novembre 1983 avec décodeur.

Viewdata est un système comparable à Télétexte, mais il offre de plus grandes possibilités à long terme. Diversement baptisé selon les pays (Vidéotex en France, et ailleurs Videotel, Prestel, Televideo), il rattache la multitude de ce que l'on peut à présent dénommer «terminaux télévisuels» à l'ordinateur central par l'intermédiaire des lignes téléphoniques normales. Prestel par exemple, le service Viewdata offert par l'administration britannique de télécommunications, fut le premier introduit en Europe. Pour accéder à ce service, l'utilisateur appuie sur l'une des touches de son boîtier de télécommande de télévision, comme pour le Télétexte; le téléviseur se branche automatiquement sur l'ordinateur central par la ligne téléphonique locale. Dans d'autres versions plus simples (l'annuaire électronique), il faut effectuer manuellement l'opération de mise en relation avec la centrale de transmission, ce qui se fait tout simplement en soulevant le combiné du téléphone et en composant le numéro de la centrale. On peut alors appeler instantanément l'index général des informations diffusées par le service, choisir le sujet qui vous intéresse et composer sur le boîtier de télécommande le



A. Haltoppé/GFI

Ces écrans montrent l'extrême diversité des informations transmises par Vidéotex.

mot-clé permettant l'accès à cette page. Un second index analytique et plus détaillé apparaît alors sur l'écran, et le choix se poursuit par une succession de sélections en arbre logique. Le système Vidéotex offre d'énormes possibilités. A la différence de ce qui se produit pour le Télétexte, les pages ne sont transmises à l'utilisateur qu'à sa demande sur le mode du dialogue. Les limites d'un système de ce type sont celles de la capacité de mémoire de l'ordinateur central.

Le système Prestel reçoit les données et les mises à jour d'environ cent cinquante organismes spécialisés dans différents secteurs «producteurs d'informations». Il peut aujourd'hui renseigner l'utilisateur sur les questions les plus diverses : l'emploi de son temps libre, les voyages et les moyens de transport, les prix de détail, les statistiques, etc. L'utilisateur paie les pages de transmission qu'il demande, le prix de chaque page apparaît sur l'écran en même temps que celle-ci et l'ordinateur débite automatiquement les frais sur le compte de l'utilisateur. Ce dernier peut à tout moment demander un état de son compte, qui apparaît directement à l'écran.

Le système Vidéotex permet déjà d'accéder directement aux banques de données. Certaines grosses sociétés comme Lockheed et des universités comme le Massachusetts Institute of Technology (MIT) aux Etats-Unis, gèrent déjà des banques de données considérables et ouvertes au secteur privé. En France, La Redoute, entre autres, offre un service opérationnel de prise de commandes à domicile par Vidéotex (expérience Télétel de Vélizy et annuaire électronique).

Un tel service est cependant destiné en priorité au monde professionnel et commercial. Son but est d'assurer la communication rapide

et sélective de nouvelles à caractère professionnel. Un certain degré de discrétion peut être obtenu en attribuant à chaque utilisateur une clé d'accès personnalisée fonctionnant comme un mot d'ordre pour l'ordinateur central, lequel peut alors accorder ou refuser à l'utilisateur l'accès à une section déterminée du fichier contenant les informations à diffusion limitée.

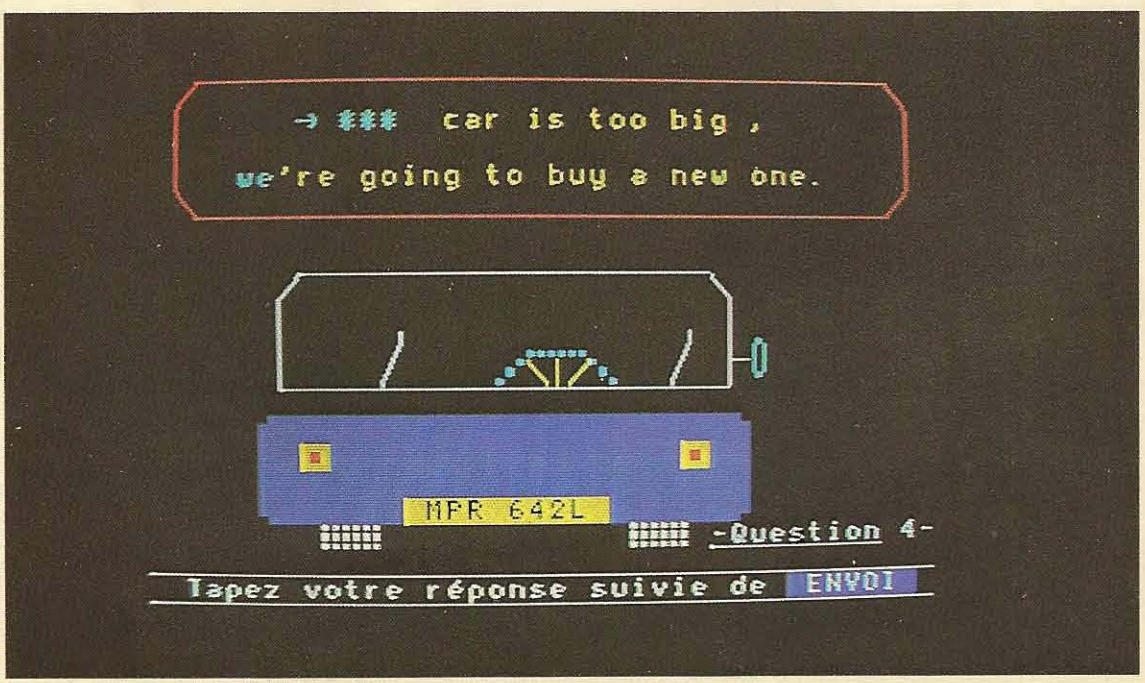
On peut échapper au risque d'interception de ces informations par des tiers au cours de leur passage sur la ligne de l'utilisateur habilité en adoptant des codes de transmission qui les rendent incompréhensibles à des personnes étrangères, mais cela exige l'intégration dans le système récepteur d'un microprocesseur programmé pour décoder le message à l'arrivée.

Autre caractéristique particulière au système Vidéotex : l'utilisateur peut communiquer de manière interactive avec l'ordinateur central. Les champs d'applications sont multiples. Le diagnostic médical assisté par ordinateur, par exemple, est un service qui exige un contact interactif entre l'utilisateur et l'ordinateur. Les programmes de diagnostic déjà utilisés par les plus grands hôpitaux permettent aux médecins d'économiser un temps précieux dans l'analyse de l'état du patient à l'arrivée, et la mise en place d'un service analogue à domicile pourrait faire gagner beaucoup de temps aux médecins généralistes.

Pour améliorer encore l'interactivité (échange), il faudrait doter le système récepteur d'un microprocesseur et d'un clavier alphanumérique, ce qui ouvrirait à l'utilisateur l'accès non seulement à la mémoire de masse de l'ordinateur mais aussi à l'unité centrale de traitement, par exemple, pour faire exécuter des programmes personnels et en obtenir les résultats sur son écran.

Le coût d'un service de ce type, surtout s'il est utilisé rarement et sur des programmes simples, pourrait bien se révéler inférieur à celui d'un ordinateur individuel. L'adoption du microprocesseur offre aussi la possibilité de se relier à sa banque, en utilisant comme « passerelle » l'ordinateur qui gère le système de diffusion, pour accomplir toutes les opérations bancaires, y compris le transfert de fonds sur son compte ou à partir de celui-ci. Le téléviseur individuel fonctionne dans ce cas comme l'un des terminaux vidéo de la succursale bancaire, et permet d'accomplir les opérations ordinaires en toute commodité, chez soi et en temps réel.

Grâce au système Vidéotex, les programmes de formation (ici cours d'anglais) donnent lieu à un « dialogue » entre ordinateur et utilisateur.



A. Halloppé/GFI

Eléments de logique

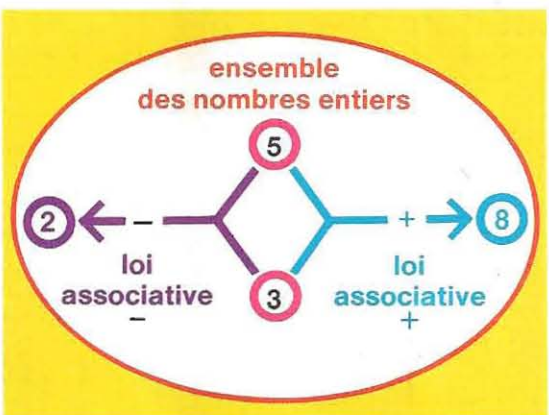
Certaines des fonctions accomplies par l'ordinateur peuvent être obtenues avec d'autres types de circuits. Nous traiterons dans ce chapitre des aspects fondamentaux de cette technique appelée **logique câblée.**
La question n'est pas directement rattachée aux ordinateurs mais constitue un premier pas indispensable pour comprendre la structure et le fonctionnement de ces machines. Nous allons commencer par donner quelques principes de logique qui seront repris et approfondis dans la partie consacrée aux langages de programmation.

Les opérateurs

Le micro-ordinateur, ou plus exactement l'un de ses composants, l'unité centrale (UC), peut accomplir aussi bien des opérations arithmétiques que des **opérations logiques.**
Les opérations sont surtout utilisées pour comparer des données entre elles et déclencher, à partir du résultat, une transaction particulière.
Par exemple, on peut insérer dans un programme calculant la feuille de paie un système de contrôle du nom qui se présente ; si ce nom est FIN, la machine doit en aviser l'opérateur et mettre fin au programme.
Dans ce cas, l'opération logique est la comparaison entre le nom et le mot FIN. Si le résultat est négatif (c'est-à-dire si le nom présenté n'est pas FIN) la machine poursuit son travail ; si le résultat est positif (le nom est FIN), la machine émet un signal et s'arrête.
Par certains aspects, les opérateurs logiques peuvent être considérés comme des opérateurs arithmétiques normaux. Nous commencerons par exploiter cette analogie.
Dans l'exécution d'un calcul arithmétique normal nous pouvons distinguer trois entités fondamentales :

- **Les opérandes**
- **L'opérateur**
- **Le résultat**

Les opérandes sont les nombres sur lesquels nous effectuons l'opération ; l'opérateur est le symbole indiquant l'opération à accomplir ; le résultat est le nombre associé par l'opérateur aux opérandes.
Considérons l'expression arithmétique 5 + 3 = 8. Le résultat (8) est associé aux opérandes 5 et 3 par l'opérateur + (addition).
Si nous changeons l'opérateur (par ex. −, soustraction), les symboles 5 et 3 seront associés (par ce nouvel opérateur) au résultat (2).
Les diverses opérations (addition, soustraction, multiplication, etc.) sont des **lois associatives**, c'est-à-dire des lois qui associent des groupes de nombres (opérandes) à d'autres nombres (résultats). Les modalités de ces associations sont exprimées par le symbole (+, −, etc.) de l'opérateur. D'où la règle fondamentale suivante : une loi associative ne peut être appliquée qu'à des éléments homogènes c'est-à-dire appartenant au même ensemble (voir schéma). Cherchons à rendre plus clair le concept d'**ensemble.**



Au niveau de l'intuition, on peut considérer comme ensemble un groupe d'éléments (nombre, lettres, objets) répondant à des lois déterminées. La définition de ces lois et les propriétés des éléments incombent à l'algèbre abstraite. Ici, nous appellerons ensemble un groupement d'éléments homogènes.
Ce concept d'ensemble, inexact du point de vue mathématique, est cependant d'emploi courant dans les applications de l'ordinateur. Bon nombre d'erreurs dans les calculs ont pour origine l'utilisation d'opérandes non homogènes (nous reprendrons un peu plus loin cette question en détails).
Nous venons d'évoquer des opérateurs correspondant aux opérations arithmétiques ordinaires, mais on peut inventer d'autres lois associatives, leur affecter un nom et les utiliser comme de nouveaux opérateurs.

En réalité, le processus n'est pas si simple. Il s'agit essentiellement d'introduire une nouvelle série d'opérateurs : les **opérateurs logiques**. Ces opérateurs obéissent à des règles spécifiques, parfois très complexes. Nous ne nous occuperons ici que de celles qui ont trait à la programmation des ordinateurs.

Les opérateurs logiques

Reprenons l'exemple des feux de circulation. Il y a quatre situations possibles :
1 - feux éteints
2 - feu rouge
3 - feu orange
4 - feu vert.
A la question : « Quel est l'état des feux de circulation ? » la réponse est nécessairement l'une des quatre situations énumérées plus haut. Soit : éteint, OU rouge, OU orange, OU vert.
Le mot OU est un opérateur logique (en anglais OR). Le symbole est OU. On peut donc exprimer les quatre situations possibles de la façon suivante (en conservant feux éteints = 1, feu rouge = 2, etc.) : état des feux de circulation = 1 OU 2 OU 3 OU 4.
Question : « Quand une voiture a-t-elle le droit de passer ? »
Réponse : « Feux éteints ou feu vert ».
On peut traduire symboliquement la réponse par :

PASSEZ = 1 OU 4 (1 = éteint, 4 = vert).

En réalité, l'expression 1 OU 4 n'est pas suffisante puisque si les feux sont éteints, il faut encore s'assurer que la voie est libre. La réponse complète sera donc :

PASSEZ = feu vert ou feux éteints ET voie libre.

Considérons cette dernière condition. Le mot ET signifie qu'on doit avoir simultanément les deux situations : feux éteints et voie libre. C'est un nouvel opérateur logique que l'on symbolisera par ET (en anglais AND).
En attribuant à la condition « voie libre » le chiffre 5, on aura :

PASSEZ = 4 OU 1 ET 5.

Mais cette expression est susceptible de deux interprétations :
1 - PASSEZ = (4 OU 1) ET 5.
2 - PASSEZ = 4 OU (1 ET 5).
L'expression n° 1 signifie que le passage est autorisé si l'on a 4 ou 1 (4 OU 1) et en même temps 5 (ET 5). C'est une interprétation erronée.
L'expression n° 2 signifie que le passage est autorisé si l'on a la condition 4 qui se suffit à elle-même ou (OU) à la fois 1 et 5 (1 ET 5). Interprétation correcte.
Cet exemple fait apparaître l'intérêt qu'il y a d'utiliser correctement les parenthèses, exactement comme dans les formules algébriques.
Les opérateurs ET et OU s'appliquent à deux éléments d'un ensemble quelconque. Considérons un exemple à deux chiffres binaires A et B.
Les deux éléments étant des chiffres binaires, ils peuvent prendre la valeur 0 ou 1. Il s'ensuit que les diverses combinaisons entre les valeurs A et B sont les suivantes :

A	B
1	1
1	0
0	1
0	0

En faisant appel aux opérateurs ET et OU, nous aurons :

I	II	III	IV
A	B	A ET B	A OU B
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Les colonnes I et II (A, B) sont les opérandes. La colonne III est le résultat de l'opération A ET B. Ce résultat est égal à 1 quand A et B valent 1.
La colonne IV est le résultat de l'opération A OU B. Ce résultat est égal à 1 quand au moins l'un des deux éléments A ou B vaut 1.
Introduisons maintenant un nouvel opérateur : OU eXclusif (XOR en anglais), que l'on représente par OUX (ou X OU).
Cet opérateur est équivalent à OU, sauf dans le cas où A = 1 et B = 1. Dans ce cas, il donne pour résultat 0, alors que OU donne 1.
En d'autres termes, OUX (OU exclusif) n'équivaut à 1 que si l'un seulement, de A ou de B, a pour valeur 1.

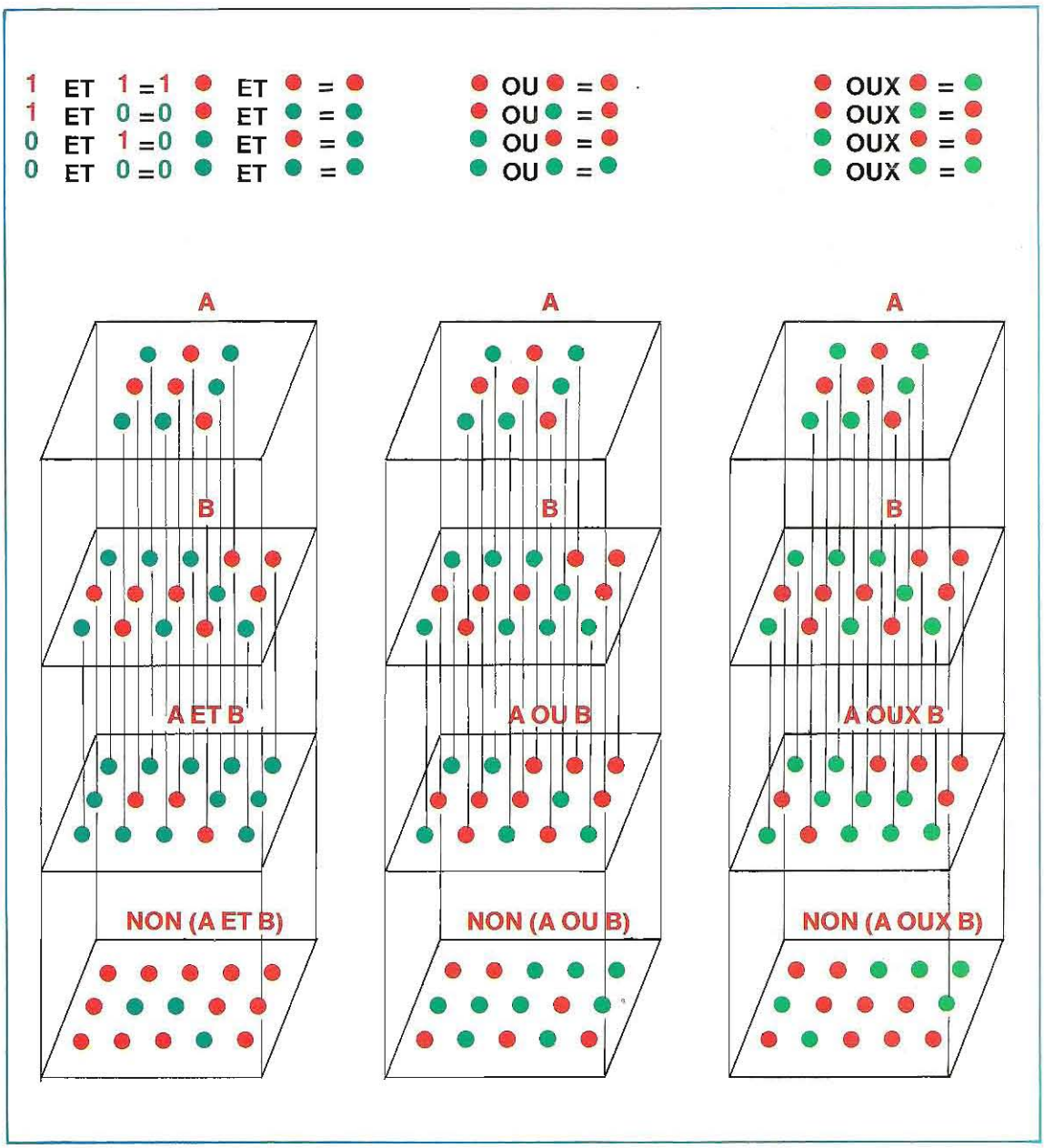
La table de vérité de cet opérateur est par conséquent:

A	B	A OUX B
1	1	0
1	0	1
0	1	1
0	0	0

Le dernier de nos opérateurs est NON (en anglais NOT). Il s'applique à un seul opérande, avec un sens complémentaire, c'est-à-dire qu'il remplace le symbole 1 par le symbole 0, et inversement:

A	NON A
1	0
0	1

Dans le graphique ci-dessous (point rouge = symbole 1; point vert = symbole 0), on verra l'application des opérateurs ET, OU, OUX, NON, à deux ensembles binaires.



Pour nous résumer, les principaux opérateurs logiques sont: ET, OU, OUX, NON. Leur table de vérité est celle-ci:

A	B	ET	OU	OUX	NON A	NON B
1	1	1	1	0	0	0
1	0	0	1	1	0	1
0	1	0	1	1	1	0
0	0	0	0	0	1	1

Le traitement des opérateurs logiques suit le même principe que celui des opérateurs arithmétiques.
En outre, les opérations logiques ne s'appliquent qu'à des éléments homogènes.

Jusqu'à maintenant, nous avons considéré les opérateurs logiques comme des nombres binaires à un seul chiffre (A = **1/0**, B = **1/0**). Mais on peut également les appliquer à des nombres binaires à plusieurs chiffres.
Si nous prenons:

A = **0 1 1 0 1**
B = **1 0 0 1 1**

Nous aurons la représentation suivante:

NON A = **1 0 0 1 0**
NON B = **0 1 1 0 0**

A ET B

A	0	1	1	0	1
B	1	0	0	1	1
A ET B	0	0	0	0	1

Les deux nombres binaires A et B, écrits en système décimal, ont respectivement pour valeur:
A = **0 1 1 0 1** = **13** décimal ($2^3 + 2^2 + 2^0$)
B = **1 0 0 1 1** = **19** décimal ($2^4 + 2^1 + 2^0$)
et puisque A ET B = **1** (exemple précédent), le résultat doit rester valable si l'on considère, pour A et B, les valeurs décimales:
01101 ET **10011** – **13** ET **19** = **1**

A OU B

A	0	1	1	0	1
B	1	0	0	1	1
A OU B	1	1	1	1	1

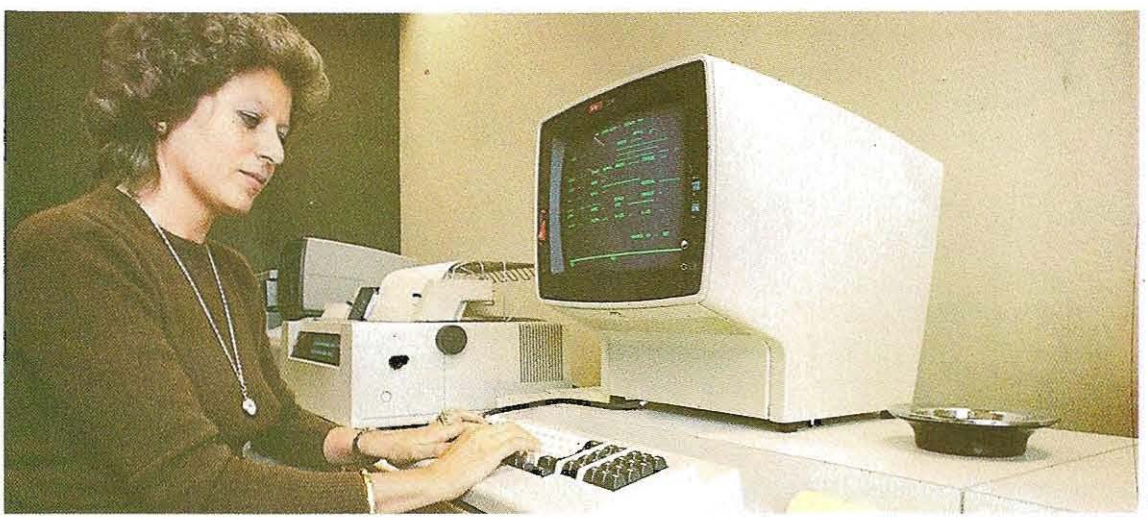
En décimal **13** OU **19** = **31** (**11111** = $2^4 + 2^3 + 2^2 + 2^1 + 2^0$ = **31**)

A OUX B

A	0	1	1	0	1
B	1	0	0	1	1
A OUX B	1	1	1	1	0

En décimal **13** OUX **19** = **30** (**11110** = **30**).

Opératrice travaillant sur un ordinateur à clavier séparé.



Marka

Solution du test 1

1 / a - La notation binaire découle du principe de fonctionnement des circuits numériques. Ces circuits ne reconnaissent que deux situations : ON et OFF. La seule base de numération possible est donc une base binaire.

b - La numération octale permet d'exprimer 3 symboles binaires avec un seul symbole octal. Par exemple, pour 111 binaire, on a 7 en octal ; 101111 binaire est 57 en octal. La forme est compacte et d'un maniement plus facile.

c - En numération hexadécimale, un seul chiffre suffit pour représenter les nombres décimaux jusqu'à 15 (15 décimal = F hexadécimal). En effet, un symbole hexadécimal a pour base 16, et symbolise une information binaire de 4 chiffres : 1111 binaire = F hexadécimal = 15 décimal. Pour transcrire des nombres supérieurs à 15, plusieurs symboles hexadécimaux sont nécessaires. Par exemple : 27 décimal = 11011 binaire = 1B hexadécimal.

d - La notation octale a pour base le nombre décimal 8, donc une telle valeur est représentée par le nombre octal 10.

---

2 / La traduction de binaire en décimal se fait en utilisant la table des poids. Lorsqu'un nombre binaire est affecté du symbole 1, on sait qu'il faut considérer la puissance 2 correspondante. Lorsqu'on établit une table des puissances, la conversion binaire-décimal consiste à additionner les puissances représentées par un symbole 1.

Puiss.	2^5	2^4	2^3	2^2	2^1	2^0	
Valeur	32	16	8	4	2	1	
			1	0	0	1	1 + 8 = 9 1001 binaire = 9 déc
					1	1	1 + 2 = 3 11 binaire = 3 déc.
						1	1 = 1 1 binaire = 1 déc.

On trouvera les autres valeurs grâce aux tables de conversion décimal-binaire.

---

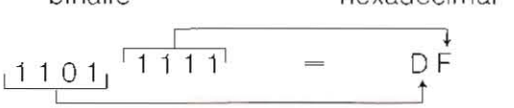
3 / La conversion binaire-octal s'obtient en divisant le nombre binaire en groupe de trois chiffres :

binaire	1 1 0	1 1 1
octal	6	7

On aura ainsi :

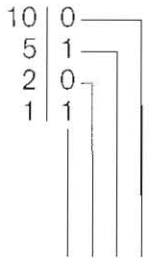
binaire		octal
11	→	3
101010	→	52
111111	→	77

4 / Pour opérer la conversion binaire-hexadécimal, il faut diviser le nombre binaire en groupes de quatre chiffres :

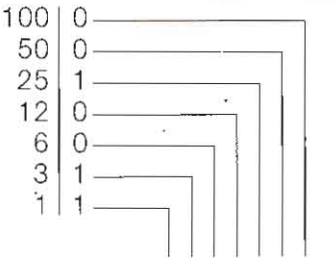


On obtiendra les autres valeurs en utilisant les listings 1 et 2 p. 63 à 67.

5 / La conversion s'opère en reportant les restes de la division par 2 :



10 décimal = 1 0 1 0



100 décimal = 1 1 0 0 1 0 0

de la même manière 1000 décimal = 1111101000 binaire = 3E8 hexadécimal.

---

6 /

Binaire	Décimal
0 0 0 1	1
0 0 1 0	2
0 1 0 0	4
1 0 0 0	8

Comme on le voit, dans un nombre binaire, le déplacement d'un chiffre vers la gauche équivaut à une multiplication par deux. De même, un déplacement vers la droite équivaut à une division par deux. On arrive à la même conclusion en observant que, quel que soit le système de numération, un déplacement vers la gauche équivaut à une multiplication par le nombre qui sert de base. Dans le langage de la programmation, un déplacement prend le nom de ROTATION.

---

7 / a - vrai b - vrai c - faux d - vrai

---

8 / La conversion d'un nombre octal en décimal consiste à multiplier chaque chiffre octal par la puissance de 8 correspondante. Prenons pour exemple la conversion du nombre octal 700 :

$$700 \text{ octal} = 7 \times 8^2 + 0 \times 8^1 + 0 \times 8^0 = 7 \times 64 + 0 + 0 = 448$$

---

9 / La conversion d'un nombre hexadécimal est plus simple si l'on passe par la notation binaire.

Par exemple, le nombre hexadécimal 3FB correspond à

3 = 0 1 1
F = 1 1 1 1
B = 1 0 1 1

par conséquent : 3FB = 1 1 1 1 1 1 1 0 1 1 = 1019 décimal

On obtiendra le même résultat en multipliant chaque nombre hexadécimal par la puissance de 16 correspondante :

B = 11

$3\ F\ B = 3 \times 16^2 + 15 \times 16^1 + 11 \times 16^0 = 3 \times 256 + 15 \times 16 + 11 = 1019$

F = 15

---

10 /

2 chiffres = 1 1 = décimal 3
4 chiffres = 1 1 1 1 = décimal 15
8 chiffres = 1 1 1 1 1 1 1 1 = décimal 255

LISTING 3

I	J	ET	OU	OUX	J + LNON I) + 1
11	65	1	75	74	54
11	66	2	75	73	55
11	67	3	75	72	56
11	68	0	79	79	57
11	69	1	79	78	58
11	70	2	79	77	59
11	71	3	79	76	60
11	72	8	75	67	61
11	73	9	75	66	62
11	74	10	75	65	63
11	75	11	75	64	64
11	76	8	79	71	65
11	77	9	79	70	66
11	78	10	79	69	67
11	79	11	79	68	68
11	80	0	91	91	69
11	81	1	91	90	70
11	82	2	91	89	71
11	83	3	91	88	72
11	84	0	95	95	73
11	85	1	95	94	74
11	86	2	95	93	75
11	87	3	95	92	76
11	88	8	91	83	77
11	89	9	91	82	78
11	90	10	91	81	79

LISTING 4

I	J	ET	OU	OUX	J + LNON I) + 1
32	65	0	97	97	33
32	66	0	98	98	34
32	67	0	99	99	35
32	68	0	100	100	36
32	69	0	101	101	37
32	70	0	102	102	38
32	71	0	103	103	39
32	72	0	104	104	40
32	73	0	105	105	41
32	74	0	106	106	42
32	75	0	107	107	43
32	76	0	108	108	44
32	77	0	109	109	45
32	78	0	110	110	46
32	79	0	111	111	47
32	80	0	112	112	48
32	81	0	113	113	49
32	82	0	114	114	50
32	83	0	115	115	51
32	84	0	116	116	52
32	85	0	117	117	53
32	86	0	118	118	54
32	87	0	119	119	55
32	88	0	120	120	56
32	89	0	121	121	57
32	90	0	122	122	58

Dans les listings 3 et 4 (p. 78), on retrouve les applications des opérateurs ET, OU, OUX et de l'opérateur composé J + (NON I) + 1, aux deux nombres I = 11 et J de 65 à 90 (listing 3), et I = 32, J de 65 à 90 (listing 4).
La série des nombres situés entre 65 et 90 peut, dans un ordinateur, prendre une signification très particulière. Dans les listings 3 et 4, la colonne J + (NON I) + 1 représente la différence entre les deux nombres J et I.
L'expression J + (NON I) + 1 constitue en fait la différence entre J et I, calculée selon la méthode du complément (NON I est le complément de I).

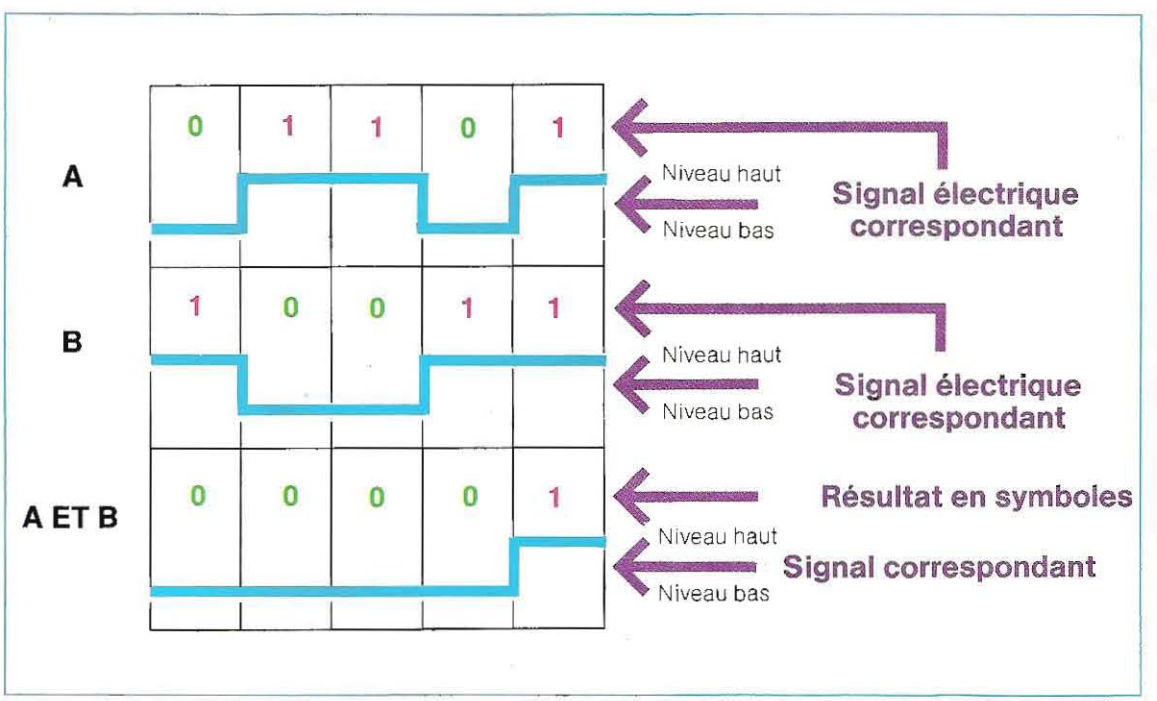
Logique câblée

Les opérateurs logiques peuvent s'appliquer aux signaux électriques de la même manière qu'ils s'appliquent aux symboles **1** et **0**.
Les signaux électriques numériques (voir le paragraphe consacré aux impulsions, page 45 et suivantes) sont la représentation physique des symboles **1** et **0**. En utilisant la même analogie :

signal électrique haut = 1

signal électrique bas = 0

les valeurs binaires A = **01101**, B = **10011** sont représentées par des signaux de niveau **haut** pour les symboles **1** et de niveau **bas** pour les symboles **0** (voir le schéma en bas de cette page).
L'opération qui permet d'obtenir le signal A ET B (par traitement des deux signaux A et B) est effectuée par des circuits numériques spécialisés que l'on appelle les **circuits logiques**.
Chacun de ces circuits a une fonction qui lui est propre. A chaque opérateur logique correspond un circuit spécifique. Donc, en théorie, il nous faut construire un appareil particulier, renfermant un circuit pour chacune des fonctions logiques dont nous avons besoin.
Concevoir un circuit, c'est relier électriquement entre eux un certain nombre de composants. On obtient ainsi un circuit intégré. Cette opération de connexion s'appelle un câblage. Et le circuit prend alors le nom de **logique câblée.**
On voit immédiatement l'inconvénient de ce type de logique : il faut autant de circuits que de fonctions. A l'inverse, l'ordinateur électronique se présente comme une machine à **logique programmable.** Il dispose en effet de circuits capables d'accomplir diverses opérations selon des ordres (**instructions**) dictés par le programmeur. Toutefois, de nombreuses fonctions annexes sont confiées à des logiques câblées.



Circuits logiques

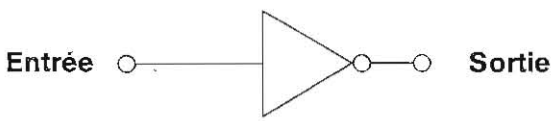
Les principaux opérateurs logiques sont:

- **NON**
- **ET**
- **OU**
- **OUX**

Ces fonctions sont réalisées par des circuits spécifiques. A chaque opérateur correspond un circuit électronique capable de réaliser la fonction définie.

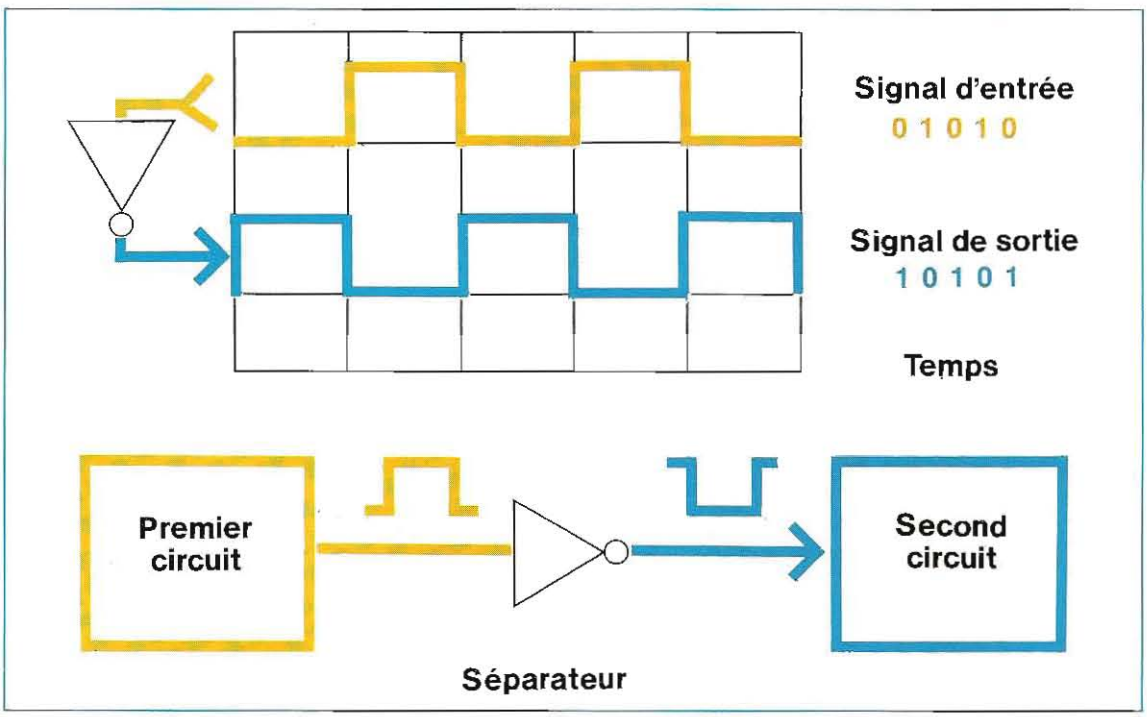
Opérateur NON

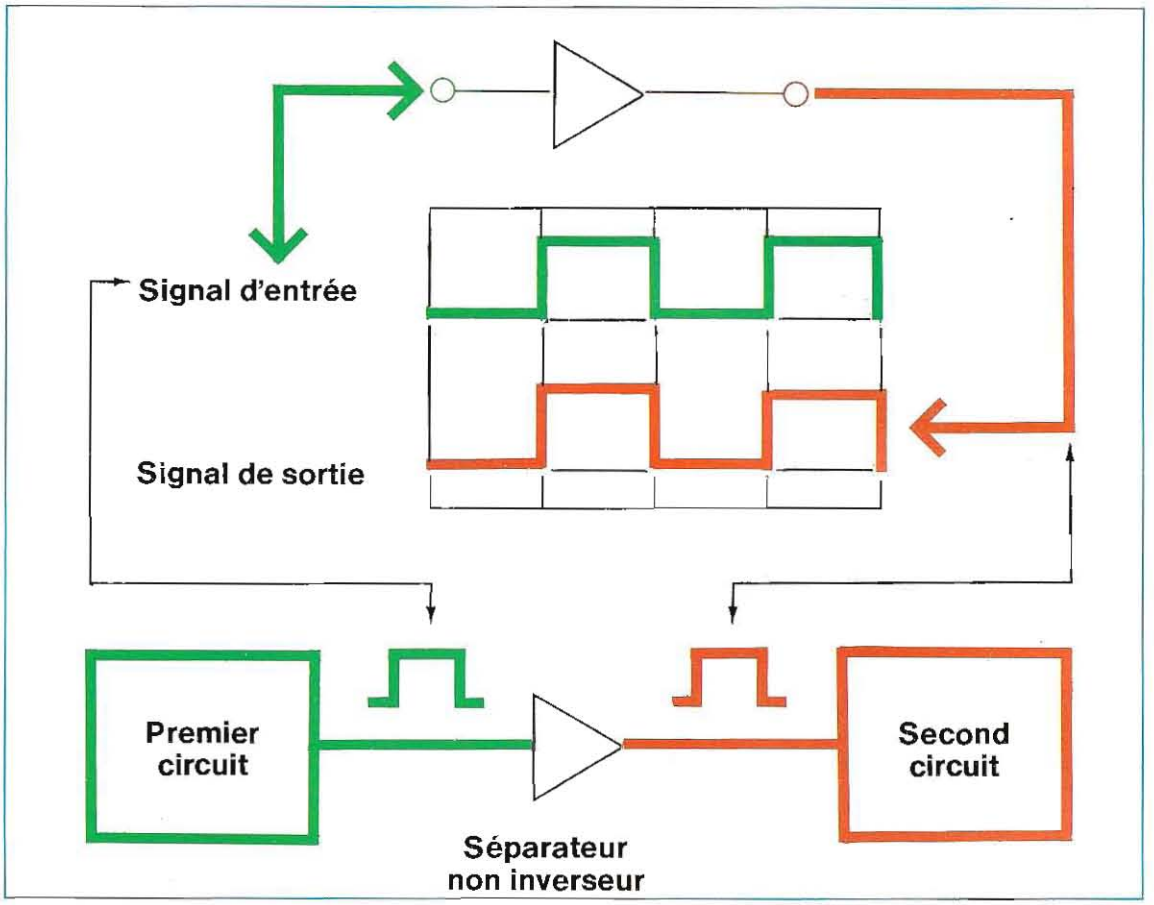
L'opérateur NON est celui qui inverse une donnée: la valeur **1** devient **0**, et vice versa. Le circuit capable de réaliser la fonction NON comporte donc une sortie pouvant prendre la valeur **0** (ou **L**) quand l'entrée est **1** (ou **H**).
Si l'entrée varie, dans un temps donné, de l'état **1** (**H**) à l'état **0** (**L**), la sortie varie en sens opposé, de l'état **0** (**L**) à l'état **1** (**H**). Ce circuit est symbolisé de la façon suivante:



Introduisons en entrée une horloge (voir p. 47): la sortie fournira l'opposé du signal d'entrée.

On trouvera dans le schéma au bas de la page le parcours des signaux d'entrée et de sortie. Ce circuit a pour fonction, non seulement d'inverser le signal, mais aussi de séparer des circuits d'un autre type. Il faut expliquer cette fonction: dans les ordinateurs électroniques, chaque élément a des fonctions complexes et comporte des circuits extrêmement sophistiqués qui renferment des dizaines de milliers de composants sur quelques millimètres carrés. On parvient à cette forte densité de composants (intégration) grâce à une technologie de pointe.
Les circuits sont conçus en vue d'une certaine fonction, mais ils sont incapables de générer de la puissance. Aussi, lorsqu'il s'agit de faire fonctionner un circuit qui requiert de la puissance, il faut interposer un troisième composant, appelé amplificateur, qui a pour but d'isoler l'un de l'autre les deux premiers et de donner de la puissance à un signal de sortie.
Le schéma logique est représenté dans le graphique en bas de la page. Dans ce graphique, le séparateur est à la fois un inverseur remplissant la fonction NON (il représente à la sortie un signal inversé par rapport à celui d'entrée) et un amplificateur.
Dans certains cas, l'inversion est gênante, c'est pourquoi on a construit des circuits qui séparent, sans inverser le signal.





Dans le graphique ci-dessus, on voit le symbole, le parcours des signaux et le schéma logique d'application du circuit en question. Quant à l'amplificateur, c'est un circuit qui délivre un signal de sortie proportionné à celui d'entrée, mais d'intensité plus élevée.

On parle d'amplificateurs

- de signal : l'amplification s'applique à la valeur du signal ;
- de puissance : l'amplification s'applique à la puissance.

Prenons un exemple.

Imaginons un ordinateur qui commande une imprimante placée dans une pièce éloignée. Pour transmettre les informations de l'ordinateur à l'imprimante, on a recours à une liaison par câble.

En raison de la résistance du câble et des perturbations électriques provenant de l'extérieur, les signaux parviendront à l'imprimante déformés, et à un niveau trop bas. Un amplificateur est alors nécessaire. On le place immédiatement avant l'imprimante, et il a pour tâche de "régénérer" le niveau et la forme des signaux.

Par ailleurs, pour des raisons de sécurité, comme les circuits de l'ordinateur ne peuvent pas être reliés directement à une ligne de transmission (risques de court-circuit), il faut prévoir un deuxième amplificateur placé entre l'ordinateur et la ligne. Le schéma au bas de la page 82 illustre ce système et les formes d'onde utilisées.

Pour une application de ce type (transmission d'information par ligne), il existe des amplificateurs (line drivers) appropriés.

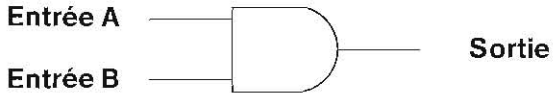
En réalité, les techniques de reconstitution des signaux sont beaucoup plus complexes qu'il n'apparaît sur le schéma.

Pour en terminer avec ce sujet, notons que ces circuits (buffer) s'emploient beaucoup, tant à l'intérieur des ordinateurs (pour isoler les uns des autres les différents composants) qu'à l'extérieur (pour faire la liaison entre l'ordinateur et l'extérieur).

Opérateur ET

Ici, nous avons affaire à un circuit à double entrée, puisque l'opérateur ET exige deux

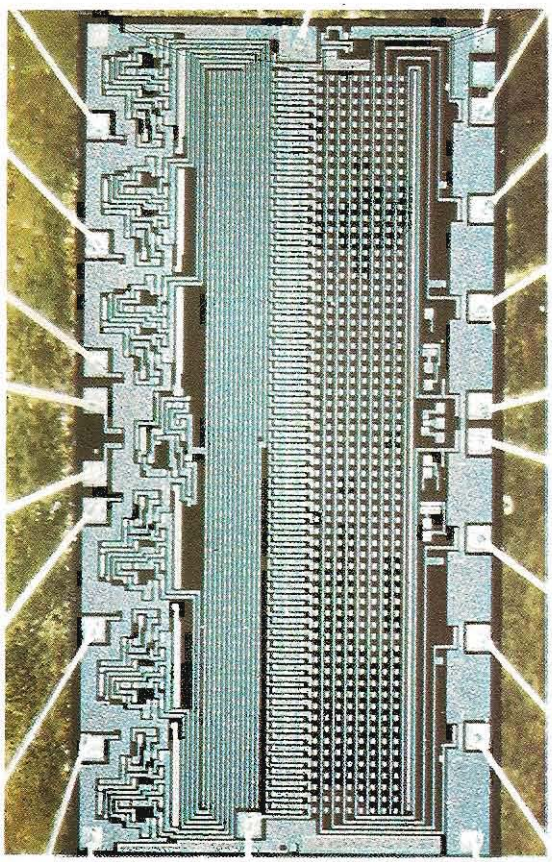
opérandes (A ET B). Le symbole graphique de ce circuit est le suivant:



Il fonctionne exactement comme l'opérateur ET. Il faut se souvenir que le symbole **1** représente le signal à haut niveau = **H** et que le symbole **0** représente le signal à bas niveau = **L**. La table de vérité immédiatement déduite de la table de l'opérateur ET (en remplaçant **1** par **H** et **0** par **L**) est par conséquent:

A	B	Sortie = A ET B
H	H	H
H	L	L
L	H	L
L	L	L

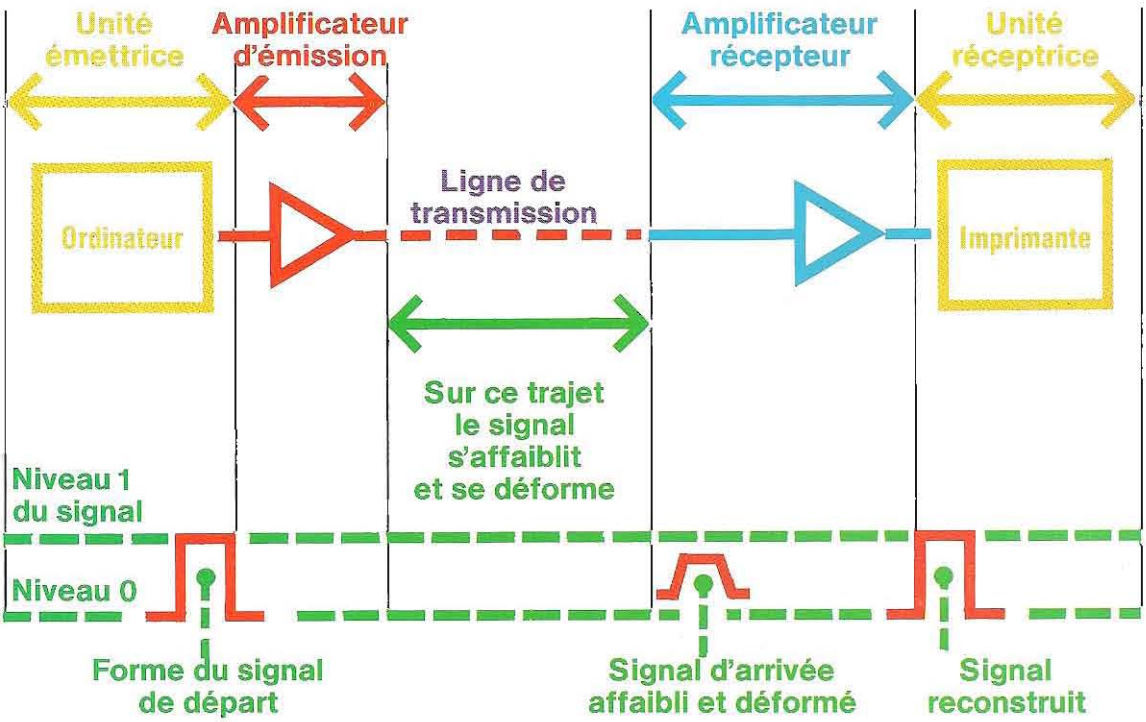
Le schéma reste inchangé si on inverse les symboles (on remplace **1** par **L** et **0** par **H**). Dans ce cas, on a un circuit dont la logique est inversée, mais le fonctionnement reste le même.



Marka

Intérieur d'un circuit intégré.

SCHEMA DE LA TRANSMISSION DE SIGNAUX PAR LIGNE

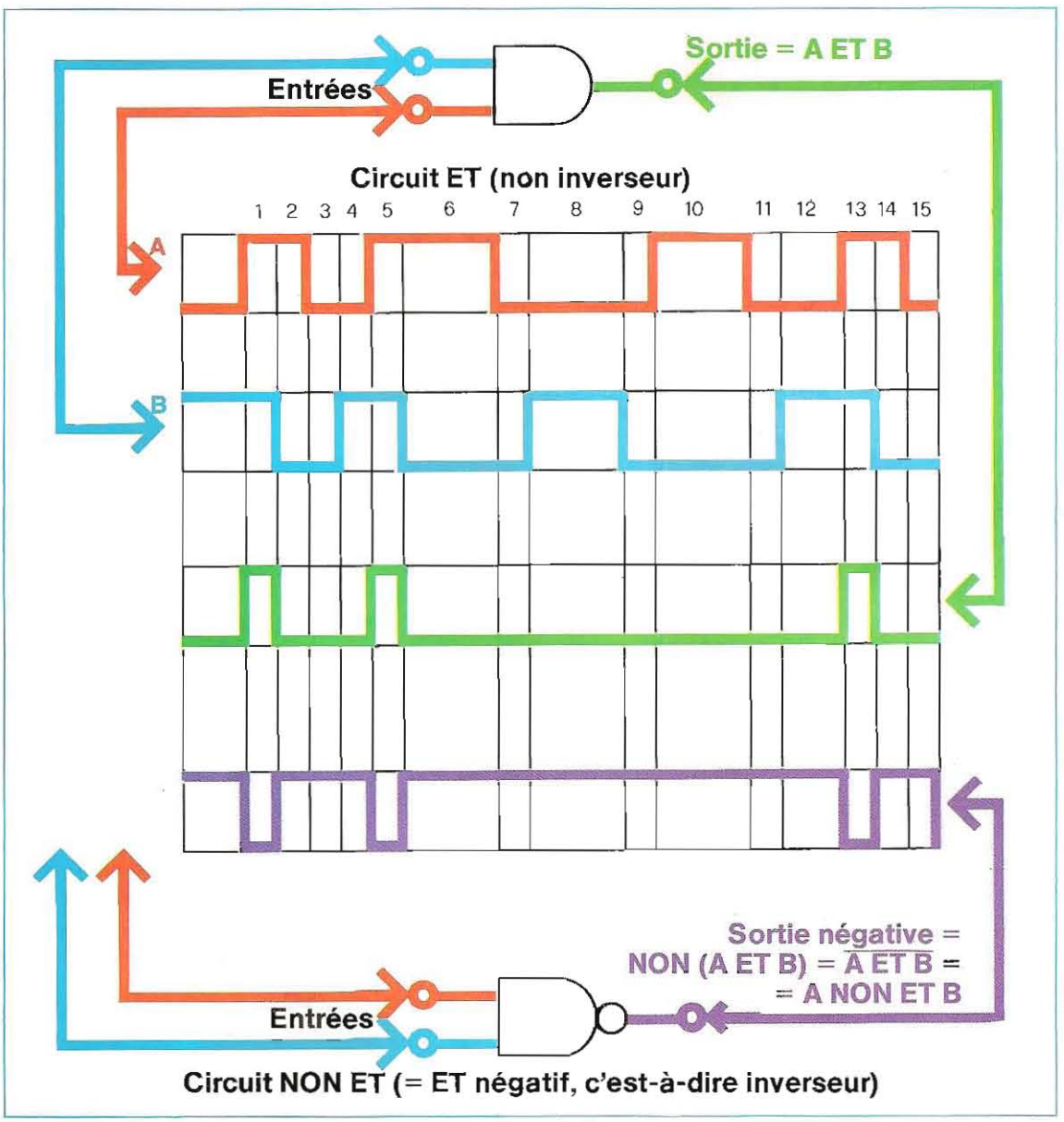


Dans le schéma au bas de cette page, on a figuré les formes des signaux.
Dans le même schéma, on trouve également la version « inverseur », que l'on appellera NON ET (en anglais NAND).
Pour le circuit NON ET, la table de vérité devient :

A	B	NON ET
H	H	L
H	L	H
L	H	H
L	L	H

C'est l'opposé de la table du circuit ET.
Enfin, si l'une des deux entrées (A par exemple), se trouve bloquée à l'état **H**, le circuit NON ET se transforme en inverseur. Si l'on retranche de la table de vérité les situations correspondant à l'état A = **L** (qui n'existe pas, puisque A est bloqué en **H**), on a :

A	B	NON ET
H	H	L
H	L	H
—	—	—
—	—	—



Comment consulter un ordinateur

Nous avons à plusieurs reprises fait allusion aux applications possibles des ordinateurs à des tâches de gestion. Nous allons approfondir cette question et étudier, en particulier, comment procéder pour obtenir l'accès à un ensemble de données stockées en mémoire.

On a souvent besoin de consulter de grands fichiers informatisés selon une organisation très complexe. Comme nous l'avons vu, c'est justement l'intérêt de l'informatisation.
Mais dans la vie courante aussi, des ordinateurs plus simples et plus personnalisés se révèlent également utiles. Dans la vie moderne en général, ce sont des fichiers informatisés qui épargnent de longues et fastidieuses recherches. Il peut s'agir aussi bien d'une bibliographie portant sur un sujet spécifique, que l'utilisateur met en mémoire dans un ordinateur, que d'un répertoire téléphonique à usage domestique.

Il faut dans tous les cas un programme permettant d'insérer au fur et à mesure des données nouvelles et permettant de les consulter simplement et rapidement. Il serait trop long et encore un peu difficile à comprendre pour le lecteur d'expliquer ici le fonctionnement de ces programmes, mais nous pouvons néanmoins exposer les principes logiques.

Nous avons vu par exemple (p. 16) comment obtenir un numéro de téléphone figurant dans une liste mise en mémoire. On peut dire que c'est une recherche qui procède par sélections successives à plusieurs niveaux. Ces sélections sont de plus en plus « affinées » et portent sur des champs de plus en plus restreints, qui aboutissent à une réponse unique. Prenons un exemple et suivons cette recherche.
Nous voulons trouver le numéro de téléphone de Michel Martin, qui habite au n° 6 de la rue M.

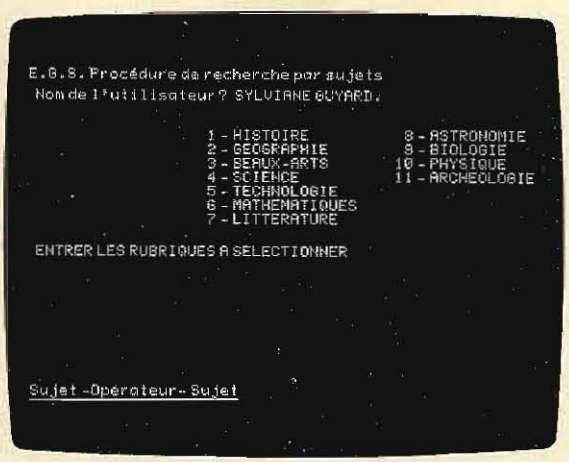
Nom = Martin

Notre ordinateur va comparer les noms figurant dans sa mémoire, et sélectionner ceux qui satisfont à la condition demandée : nom = Martin. Ce premier niveau est très peu sélectif, et nous allons obtenir un grand nombre de réponses. Il faut donc procéder à une nouvelle sélection, dite de deuxième niveau, et introduire une donnée supplémentaire : le prénom.

Prénom = Michel

Si nous posions les deux questions de façon indépendante, nous obtiendrions deux listes : celle des abonnés au téléphone de nom patronymique Martin, et celle des abonnés de prénom Michel. Ce n'est pas ce que nous voulons. Il faut donc lier les deux opérations au moyen de l'opérateur logique ET, afin que la





réponse (ou les réponses) satisfasse à la condition :

Nom = Martin (premier niveau)
ET
Prénom = Michel (deuxième niveau)

Mais nous avons besoin d'introduire un troisième critère de sélection, l'adresse, qui sera, elle aussi, vérifiée par rapport à d'autres adresses. La condition globale à fournir à l'ordinateur sera :

Nom = Martin
(premier niveau)
ET
Prénom = Michel
(deuxième niveau)
ET
Adresse = n° 6 rue M.
(troisième niveau)

L'ordinateur doit alors nous donner le numéro de téléphone désiré, sauf s'il existe deux Michel Martin à l'adresse indiquée, mais on peut aussi prévoir ce cas.

Supposons maintenant que nous connaissions le nom et l'adresse de la personne, mais que nous ayons un doute quant au prénom. Monsieur Martin se prénomme peut-être Michel, ou Frédéric.
Nous allons cependant pouvoir faire notre recherche. Notre question sera, par conséquent, exprimée en ces termes :

1) Nom = Martin
2) Prénom = Michel Ou Frédéric
3) Adresse = n° 6 rue M.

Ces conditions doivent être simultanément remplies. La condition 2 se transcrit dans les termes que nous connaissons déjà :

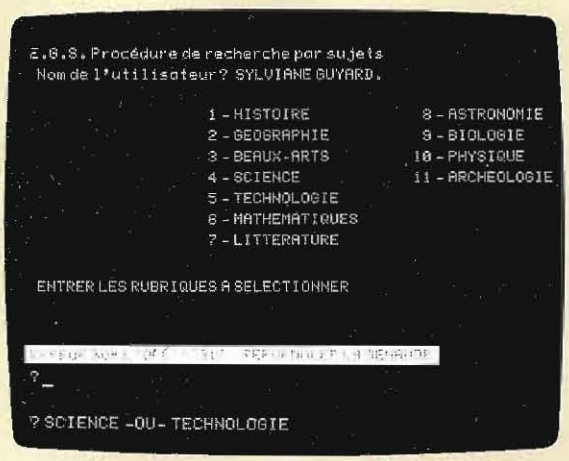
Nom = Michel OU Frédéric

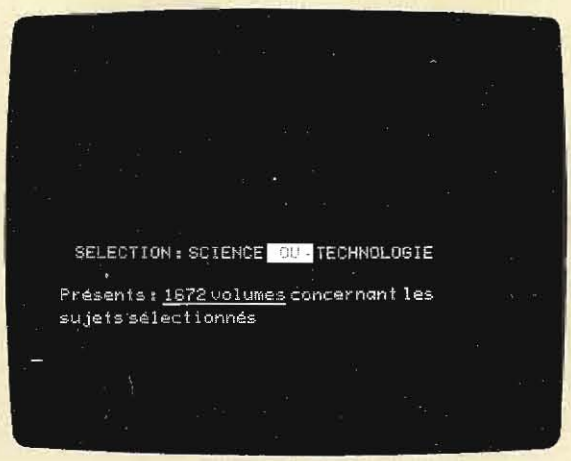
La question globale est donc celle-ci :

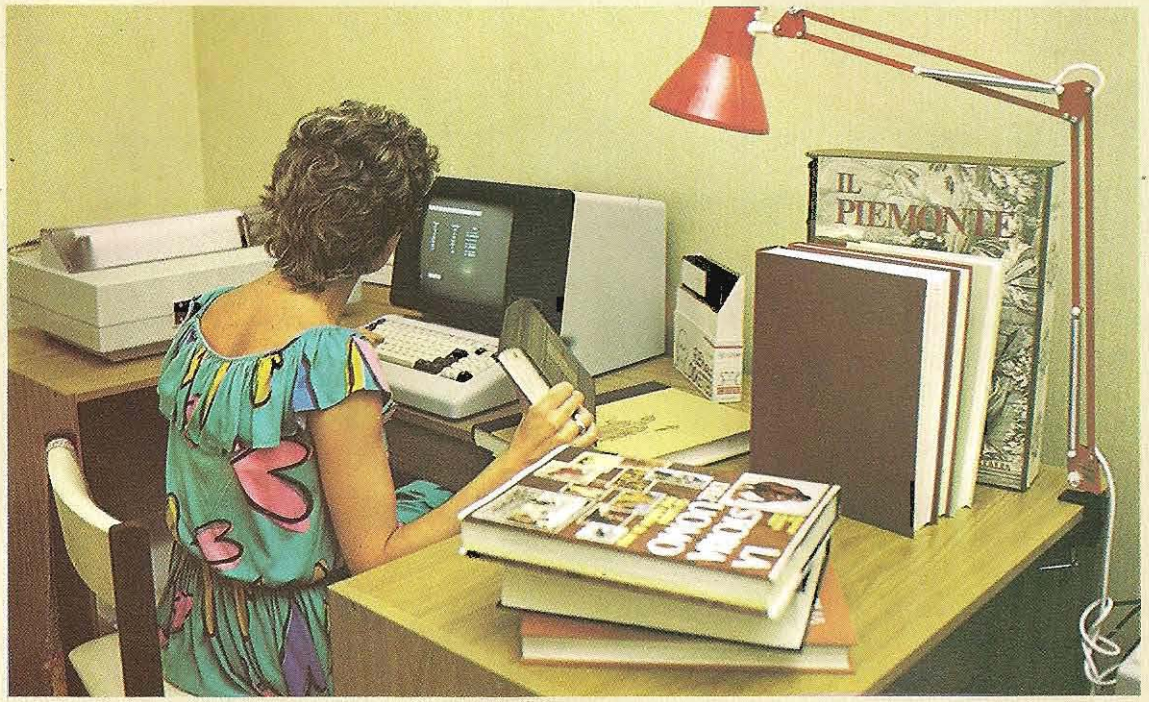
Nom = Martin-ET-Prénom = Michel-OU-Frédéric-ET-adresse = n° 6 rue M.

Le doute que nous avons quant au prénom ne nous empêche pas d'obtenir un renseignement fiable, puisque la condition 3 viendra compléter la condition 2. En effet, l'ordinateur va explorer la zone du fichier qui contient les Martin. Il s'arrêtera dès qu'il rencontrera un Michel ou un Frédéric, et vérifiera (ou sélectionnera) en fonction de l'adresse fournie dans la question.

Comme dans le cas précédent, nous avons recherché une information caractérisée de manière univoque par la condition contenue dans la question.







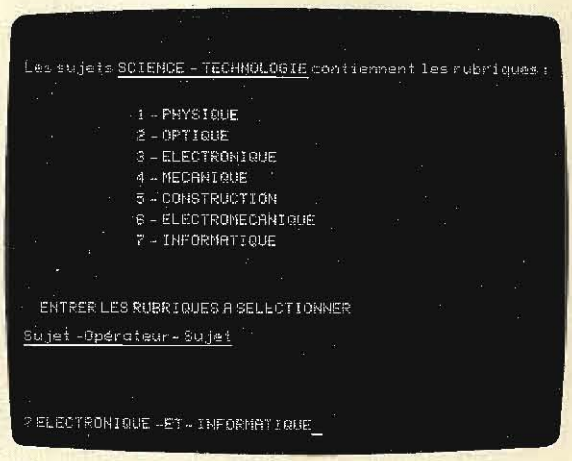
Dominique Fradin

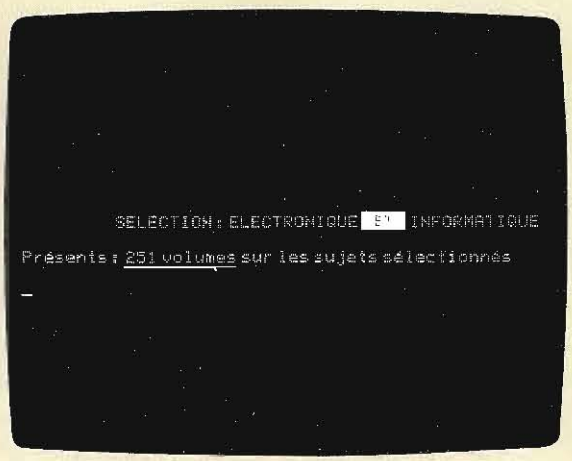
En stockant dans l'ordinateur les données bibliographiques, on facilite les recherches de la bibliothécaire.

La situation serait différente si nous n'avions que le nom, sans prénom ni adresse. L'ordinateur nous fournirait en sortie les numéros de téléphone de tous les Martin figurant sur la liste. Liste très longue sans doute, puisque nous aurions demandé une information sans fournir les paramètres permettant de la cerner de manière univoque.

Ces programmes ont des applications multiples. Prenons pour exemple les grandes bibliothèques dans lesquelles chaque texte du fond documentaire est enregistré sous le nom de l'auteur, et selon le titre, le thème, la langue dans laquelle il est écrit, la localisation sur les rayonnages de la bibliothèque, la disponibilité à la date de la demande, et bien d'autres renseignements encore. Cette gestion informatisée évite de longues et fastidieuses recherches, souvent génératrices d'erreurs et d'oublis.

Grâce aux fichiers informatisés, ce travail est aujourd'hui simplifié. Les opérateurs permettent d'organiser les informations archivées en mémoire, de façon à fournir aisément le ren-





seignement demandé. On verra par exemple dans nos illustrations comment utiliser un ordinateur pour établir une bibliographie sur un sujet donné – ici le thème concerne les « mémoires informatiques ». Le bibliothécaire moderne demande au clavier la mise en mémoire du programme de recherche à utiliser. En réponse, l'ordinateur demande le nom de l'utilisateur. Cette demande peut avoir des motifs variés. Il peut s'agir d'interdire la consultation à des personnes non agréées ; ou bien de limiter la recherche de certains textes réservés ; ou tout simplement d'inscrire le nom du chercheur en tête de la liste qui constituera le produit final, et d'éviter ainsi toute confusion entre plusieurs recherches consacrées à des sujets proches. La machine fait alors apparaître sur l'écran un menu : les différentes rubriques où se retrouve le sujet de la recherche. Les sections qui nous intéressent peuvent être Science et Technologie. Nous aurons donc à fournir les conditions :

Science - OU - Technologie

L'ordinateur nous indique les commandes à donner, et fait apparaître sur l'écran :

Sujet - Opérateur - Sujet

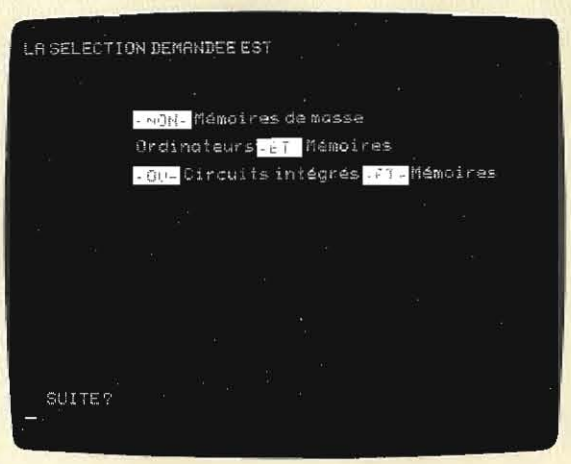
Il faut respecter la syntaxe correcte pour que l'ordinateur reconnaisse l'instruction. Une erreur de formulation entraînerait l'annulation de l'instruction et ferait apparaître un message d'erreur. Il faudrait reformuler l'instruction. Une fois reçue la première condition de sélection, la machine répond en donnant le nombre des textes relatifs au sujet choisi. Si ces textes ne sont pas trop nombreux, l'utilisateur peut demander la liste des ouvrages et leurs caractéristiques. Ici, notre liste est trop longue. Il faut donc affiner notre investigation. Nous allons en informer la machine en lui donnant une nouvelle instruction.
L'ordinateur va nous présenter sur l'écran les sous-sections qui composent la section générale Science - Technologie. Si le sujet qui nous intéresse peut rentrer dans la sous-section Electronique et Informatique, nous allons donner l'instruction :

Electronique - ET - Informatique

Ce nouveau tri va permettre une réduction du nombre des titres. Nous allons opérer la sélection définitive en donnant l'instruction :

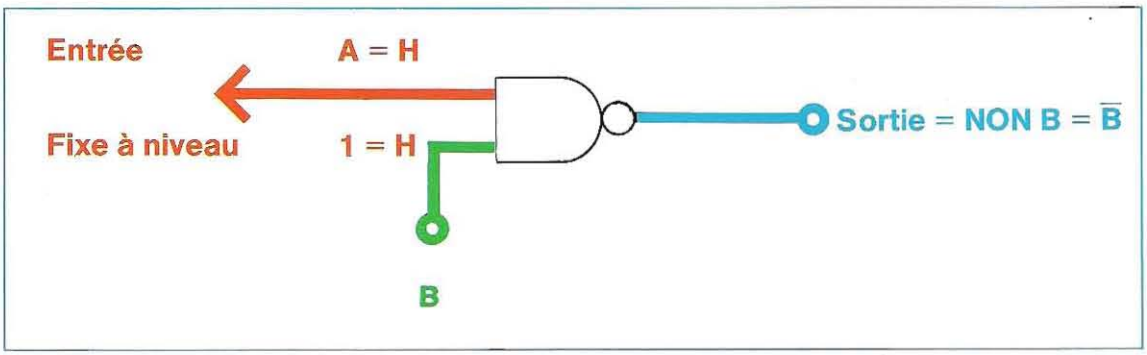
Circuits intégrés - ET - Mémoires

Si nous avions donné pour seule instruction Mémoires, nous aurions trouvé en sortie tous les textes qui ont trait aux mémoires, parmi lesquelles les bandes magnétiques et les disques. La caractéristique supplémentaire de « Circuits intégrés » nous permet de sortir seulement les textes relatifs aux mémoires électroniques et aux circuits intégrés. L'ordinateur a maintenant tous les éléments nécessaires pour mener à bien la recherche.





La sortie est l'opposé de l'entrée B, ou, en d'autres termes, la négation de B (NON B). En langage symbolique, si l'entrée est B, la sortie fournie par le circuit, c'est-à-dire l'opposé de B (négation de B), se transcrira par $\overline{B}$. C'est ce qu'indique le schéma suivant:



On voit dans ce shéma deux entrées générales (A et B) en forme d'ondes non périodiques. Pour en déduire la sortie, il faut observer que le signal de haut niveau est équivalent à **1**, et le signal de bas niveau est équivalent à **0**. De cette façon, comme le montrait le schéma page 83, nous avons:

Segment 1:

les niveaux de A et B sont hauts: **A = 1, B = 1**. Donc A ET B = **1**, c'est-à-dire haut: **H**.

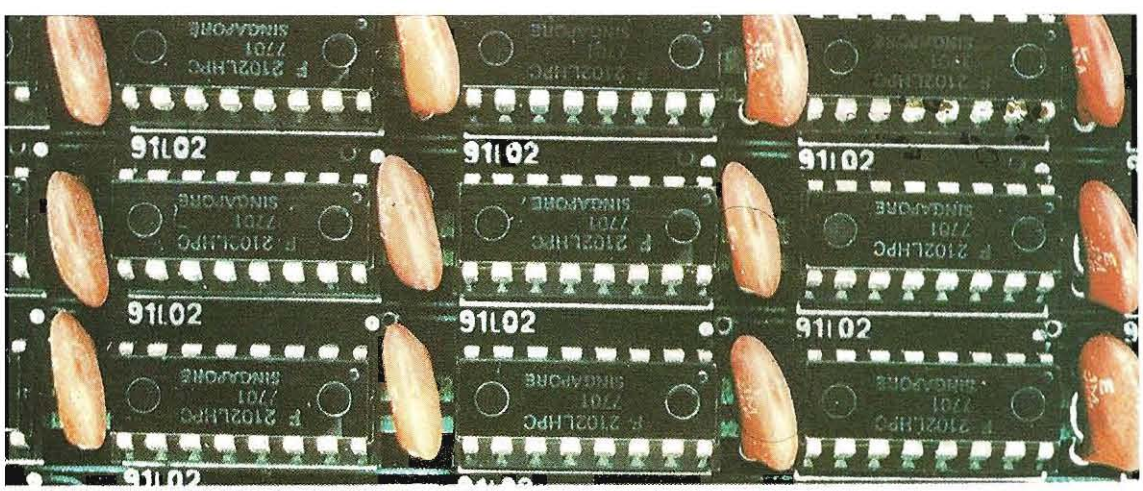
Segment 2:

A est haut, A = **1**; B est bas, B = **0**. Nous avons donc: A ET B = **0**, c'est-à-dire bas: **L**.

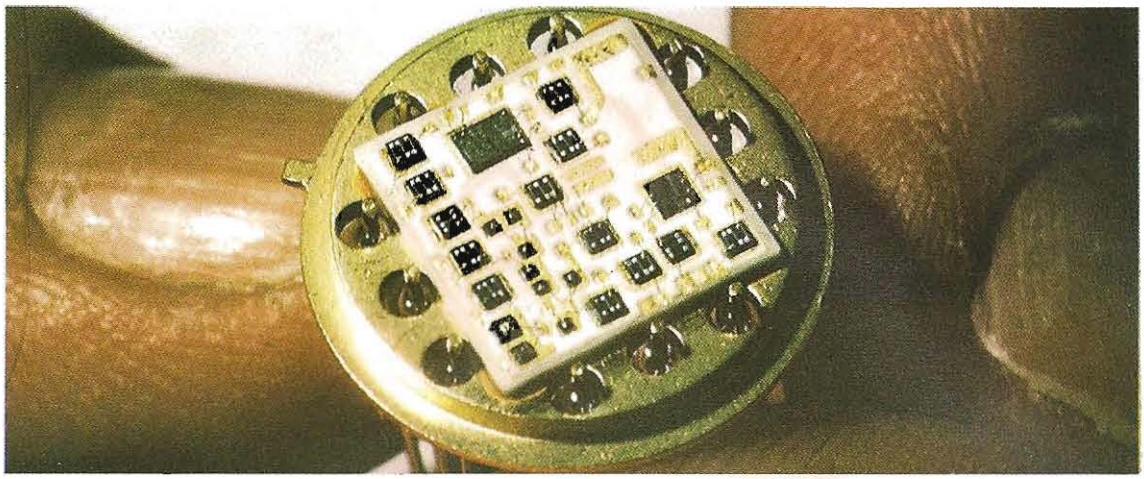
Dans le tableau à droite, nous avons réuni les 15 segments du schéma de la page 83.

Segment	A	B	A ET B	$\overline{\text{A ET B}}$
1	1	1	1	0
2	1	0	0	1
3	0	0	0	1
4	0	1	0	1
5	1	1	1	0
6	1	0	0	1
7	0	0	0	1
8	0	1	0	1
9	0	0	0	1
10	1	0	0	1
11	0	0	0	1
12	0	1	0	1
13	1	1	1	0
14	1	0	0	1
15	0	0	0	1

Circuits intégrés regroupés sur une carte.



Marka



Marka

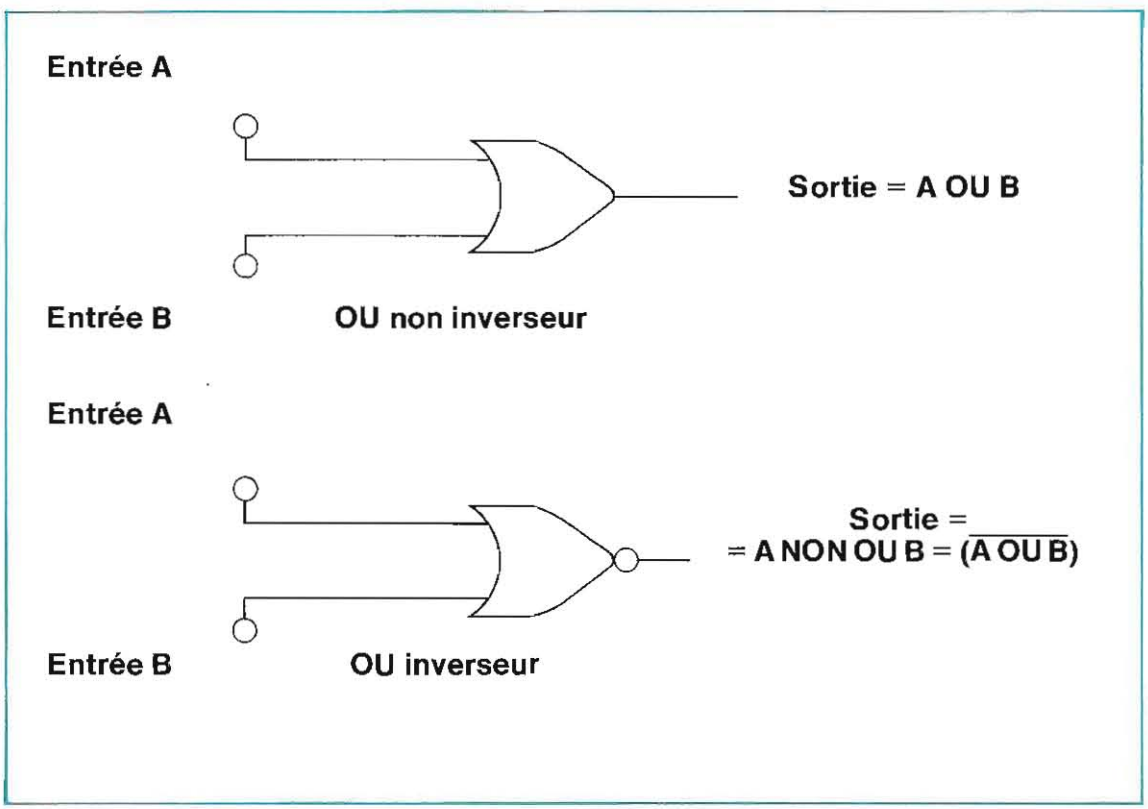
Une puce de silicium-saphir, très agrandie.

Opérateur OU

En utilisant la notation habituelle : 1 = **H** et 0 = **L**, le circuit qui réalise la fonction OU a la même table de vérité que l'opérateur.
Ce type de circuit existe également en non inverseur et inverseur. Leurs symboles respectifs figurent sur le schéma ci-dessous.
La table du OU inverseur s'obtient en inversant la table du OU non inverseur :

A	B	OU	$\overline{\text{OU}}$*
H	H	H	L
H	L	H	L
L	H	H	L
L	L	L	H

* Le trait horizontal au-dessus d'un symbole d'opérateur logique indique qu'il s'agit du même opérateur, mais inversé, c'est-à-dire négatif. $\overline{\text{OU}}$ signifie OU négatif (inversé).



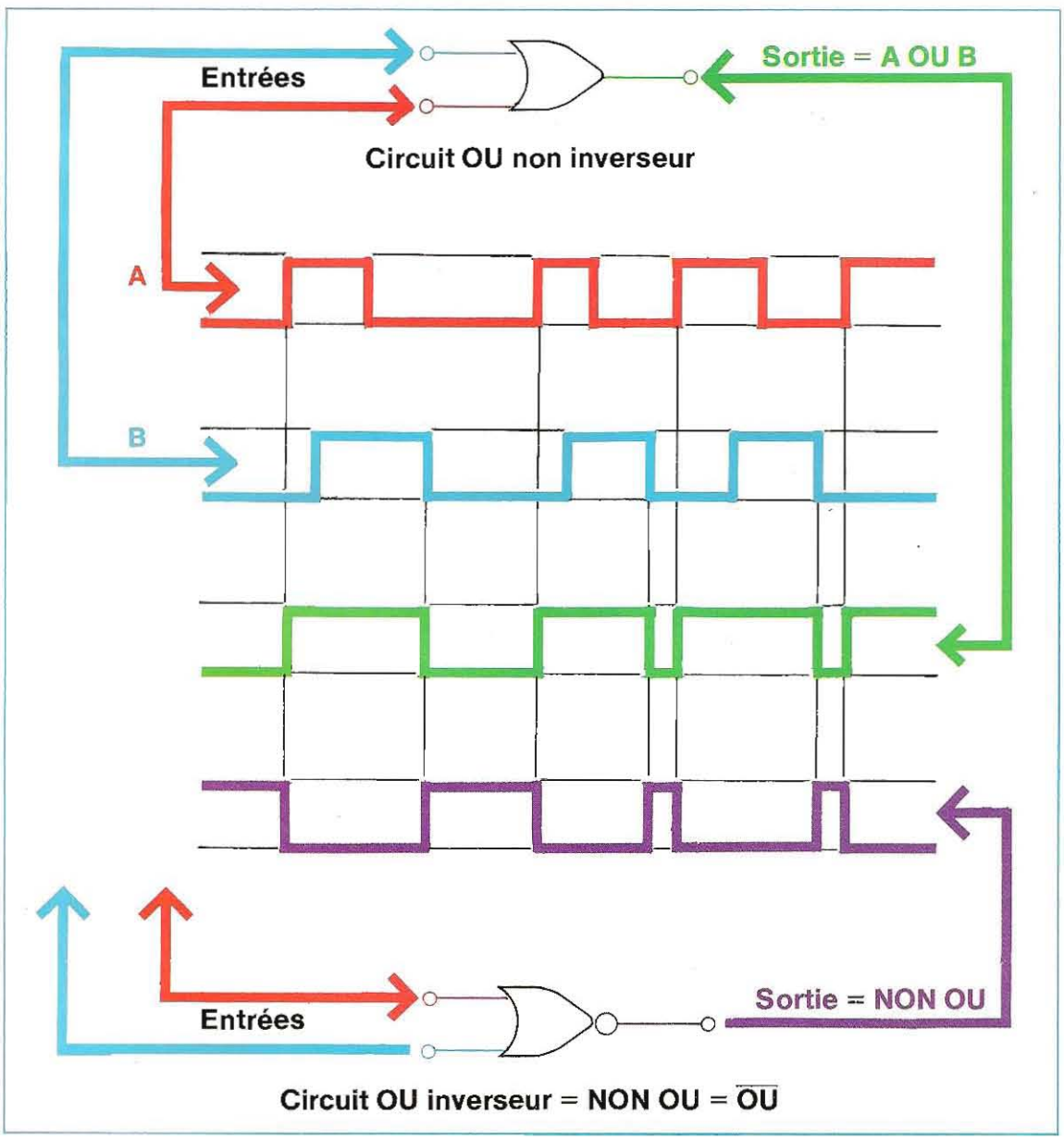
Le schéma de cette page figure un exemple de forme d'onde (de signal).
Ici aussi, pour les deux entrées, nous avons supposé la présence d'un signal non répétitif (non périodique).

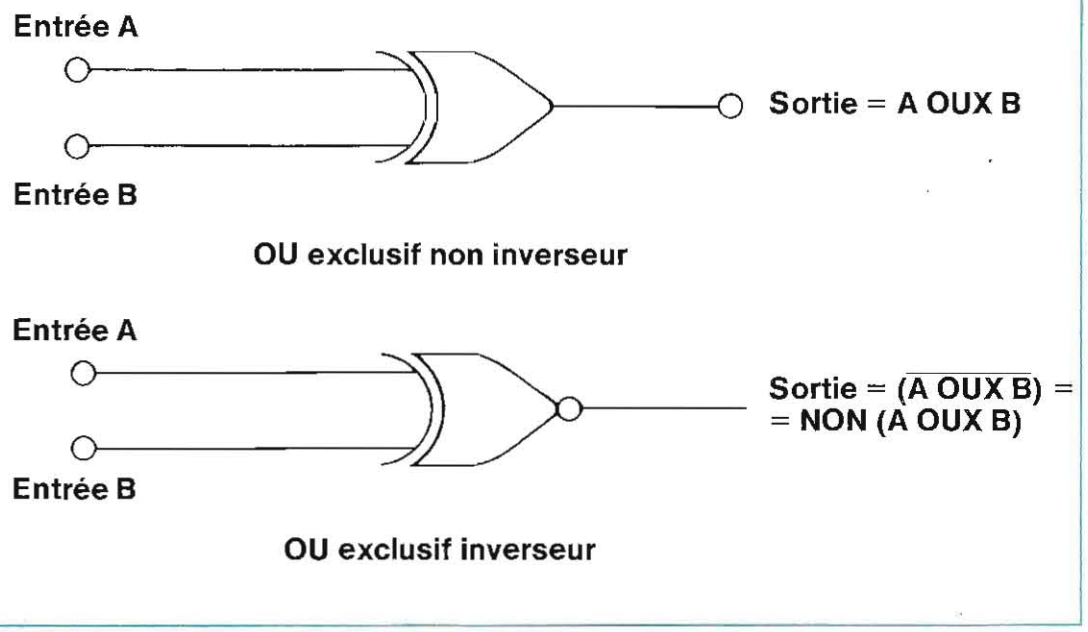
Opérateur OUX

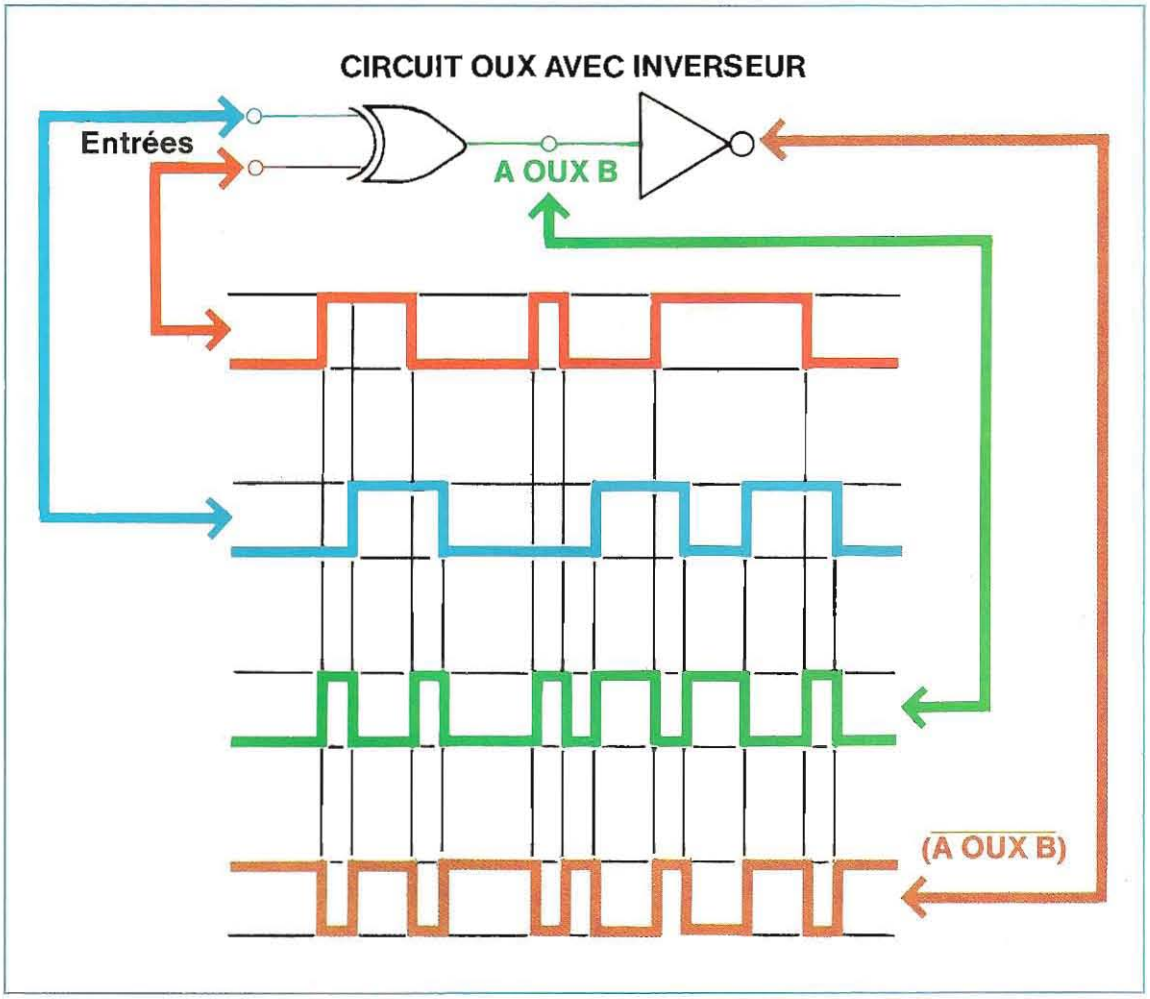
Pour ce circuit, on applique la table de vérité de l'opérateur qui lui donne son nom. Les symboles sont représentés dans le schéma en haut de la page 91.
Pour ce circuit, il est plus difficile de trouver des constructeurs qui fabriquent le type inverseur ($\overline{A \text{ OUX } B}$). On peut réaliser cette fonction en adjoignant un inverseur au circuit OU non inverseur.
Le schéma en bas de la page 91 figure les connexions et les formes de signaux.

Autres types d'inverseurs

Dans le circuit amplificateur (inverseur ou non), il y a une entrée et une seule.
Tous les autres circuits en ont deux (ou plus de deux). On peut donc les ramener au premier circuit (sauf le type OUX) en reliant symboliquement les deux entrées entre elles.




Entrée A
Entrée B
Sortie = A OUX B
OU exclusif non inverseur
Entrée A
Entrée B
Sortie = (A OUX B) =
= NON (A OUX B)
OU exclusif inverseur


CIRCUIT OUX AVEC INVERSEUR
Entrées
A OUX B
(A OUX B)

La banque de l'avenir

Dans le monde entier, les banques s'efforcent de mettre au point les services qu'elles vont offrir à leur clientèle grâce à l'introduction de techniques nouvelles. Il est clair que les transactions bancaires vont en être radicalement transformées.

Il est encore un peu difficile de préciser ce que seront ces nouveaux services dans le détail, mais nous pouvons d'ores et déjà affirmer qu'ils rendront vite obsolètes les actuelles « calculettes ».

Les confédérations industrielles et commerciales s'inquiètent d'ailleurs de ces projets encore secrets. A juste titre : l'esprit inventif et l'ambition des jeunes générations de techniciens semblent sans limites. En revanche, ce qui est certain, c'est que les banques vont fonctionner dans un proche avenir selon des formules de self-service. Les clients pourront opérer leurs transactions pratiquement sans sortir de chez eux.

Cela ne signifie pas nécessairement la disparition des banques comme établissements ouverts au public. Mais, avec l'informatisation des opérations de routine, certains changements interviendront ; le personnel bancaire sera plus disponible pour des services plus sophistiqués et plus personnalisés.

* * *

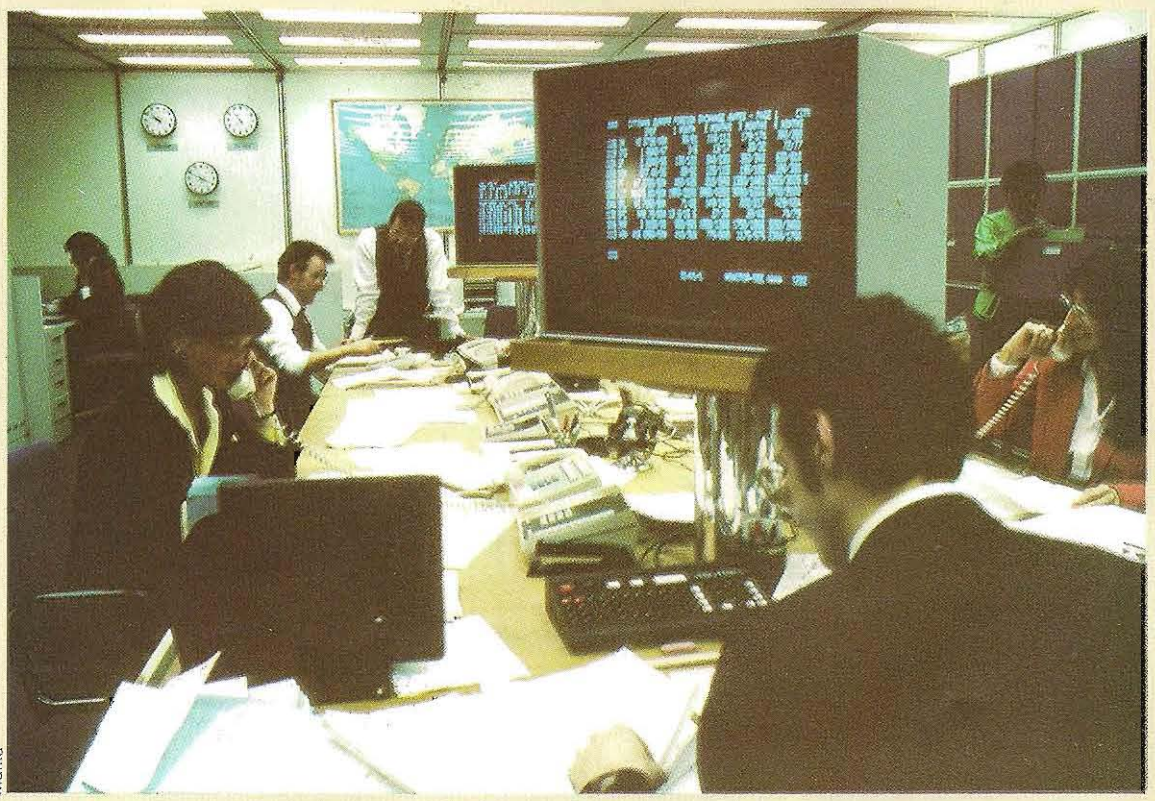
Dans de nombreux pays, certaines banques ont décidé de tenter ce type d'expérience. La Barclays Bank est l'une des banques les plus à l'avant-garde pour les services « à domicile ». Nombre de ses clients disposent déjà de terminaux personnels chez eux ou à leur bureau.

Aux Etats-Unis, la City Bank et la Chemical Bank procèdent à des expériences analogues en collaboration avec deux cents de leurs propres clients.

Ces deux banques fournissent à leurs usagers des terminaux personnels qui permettent des transactions et répondent à certaines questions.

En Allemagne fédérale, un petit établissement, la Verbraucher Bank, a également tenté une expérience avec quelques clients. Ceux-

Terminaux de consultation de l'ordinateur de la Chase Manhattan Bank.

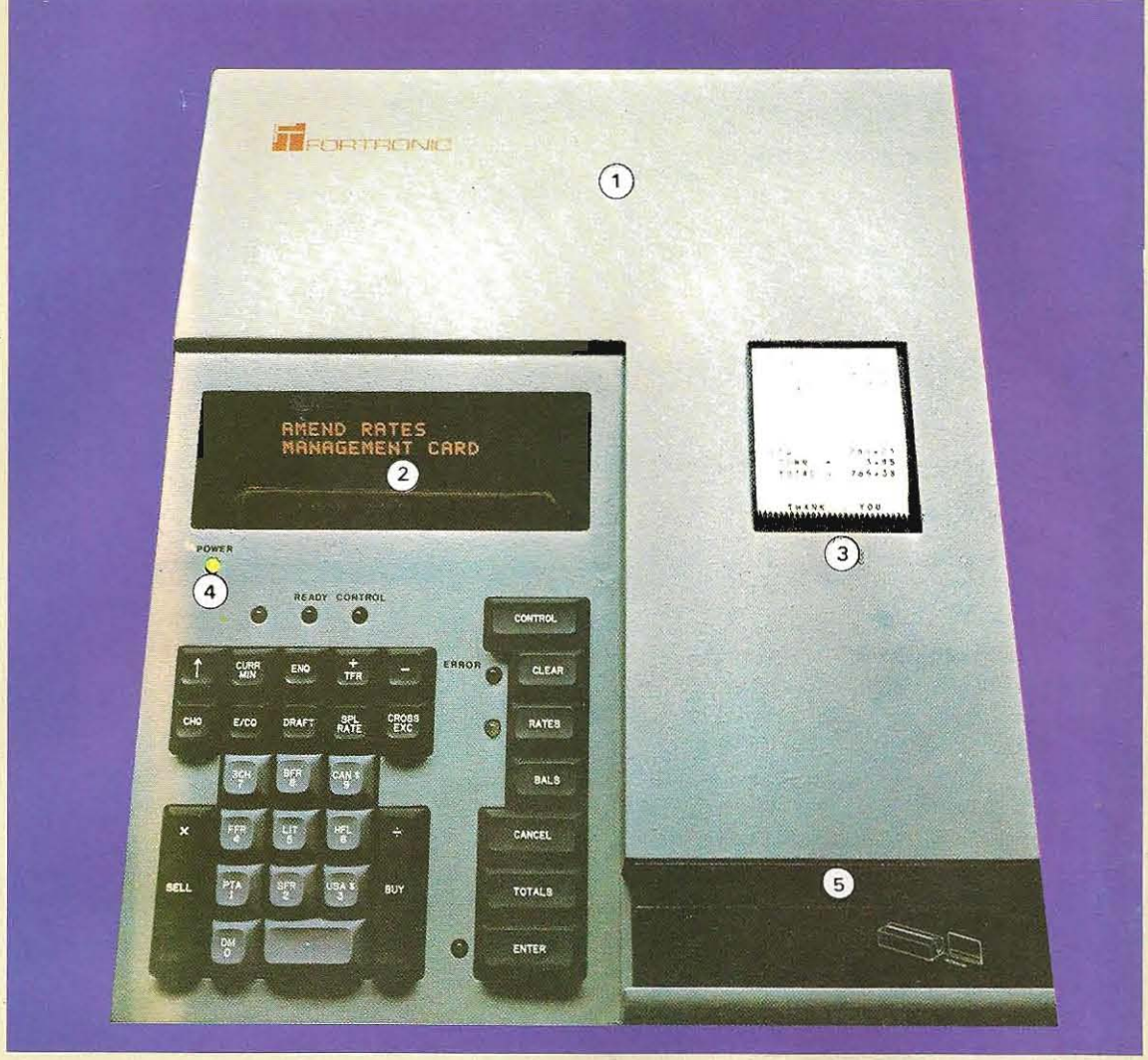


Marka



ci ont reçu un petit terminal relié à un ordinateur central au moyen d'un système de type Vidéotex. Ils peuvent ainsi connaître le solde de leur compte, en obtenir un relevé, effectuer des virements sur d'autres banques, entre autres. Des banques plus grandes, comme la Deutsche Bank, se voient obligées de suivre cet exemple.

En France, les banques se livrent à une expérience encore plus futuriste depuis le mois de janvier 1982, dans plusieurs villes. Tous les habitants possédant un compte courant y ont reçu un module électronique et un terminal domestique. Chaque carte à mémoire, de la dimension d'une carte de crédit, contient un microprocesseur (ordinateur miniature), une mémoire ROM avec les programmes et les algorithmes de contrôle, et une mémoire capable de transcrire, mais non d'effacer, un grand nombre de transactions. Cette carte tient lieu de carnet de chèques ; on peut l'utiliser aux guichets des banques mais aussi dans les supermarchés, les grands magasins, les stations-service, les cabines télépho-



Fortronic

Terminal bancaire (1) à affichage (2), imprimante (3), clavier (4) et lecteur de fiches magnétiques (5).

niques, entre autres. Naturellement, de leur côté, les points de vente ont été équipés de terminaux spéciaux, dits POS (Point Of Sale terminal) capables d'effectuer des opérations avec les cartes de crédit normales aussi bien qu'avec les cartes à mémoire.
Dans ce cas des cartes à mémoire, l'« intelligence » nécessaire pour exécuter les contrôles n'est pas celle du terminal mais bien celle de la carte.

* * *

En Belgique, la Caisse générale d'épargne et de retraite a mis en route un programme de réorganisation de ses agences dans cet esprit. Dans les nouvelles agences, les clients trouvent d'abord des consoles, sur lesquelles ils formulent d'abord leur demande. Ils sont alors dirigés vers un guichet s'il s'agit d'une opération de routine. S'ils ont à résoudre un problème plus spécifique ou plus personnel, ils sont orientés vers un bureau où ils vont trouver la personne compétente. L'ordinateur évite les situations désagréables comme les attentes inutiles et les erreurs d'orientation du client.
Dans les banques self-service, plus d'employés derrière les guichets. Le client utilise un dispositif composé d'un écran où s'affichent les informations souhaitées, et d'un clavier pour communiquer les instructions à l'ordinateur de la banque. Clavier, écran, et éventuellement une petite imprimante, peuvent très bien trouver leur place au domicile de l'utilisateur (on emploie alors le téléviseur comme écran d'ordinateur), ou encore au bureau, à l'usine, dans les agences bancaires. Au Japon, on trouve même ces terminaux à l'extérieur des agences bancaires, à la disposition du public, un peu comme les cabines téléphoniques.
Comment s'opère la liaison entre le terminal domestique et l'ordinateur central de la banque ? Très simplement, à travers une ligne

téléphonique normale. Sur le clavier du terminal, le client appelle l'ordinateur, puis il compose son numéro de code (un mot de passe), puis il dialogue avec l'ordinateur, guidé par une série de programmes (menu) et de questions spécifiques. Dans certains cas, le client peut même se dispenser de composer son numéro d'accès : le terminal le lira lui-même directement sur la bande magnétique de la carte de crédit.

De nombreuses banques américaines fabriquent leur propres terminaux. C'est le cas de la Chemical Bank et de la City Bank. Pour donner un ordre de grandeur, le prix de chaque terminal individuel varie actuellement entre 300 et 1 000 dollars. Une production de masse de ces terminaux en abaisserait le prix à environ 100 dollars.

* * *

En Europe, il semble que les banques s'intéressent davantage à des systèmes de type Vidéotex (Minitel), avec branchement sur le réseau commuté, et utilisent donc un service déjà disponible aux usagers du téléphone. Les services nationaux seront rapidement reliés entre eux. Les clients auront ainsi directement accès à leur compte à partir de n'importe quel terminal Vidéotex situé en Europe. Les banques sont en train d'élaborer un logiciel qui, au moyen d'instructions spécifiques et de contrôles appropriés, permet d'ouvrir l'accès à leur fichier à partir de terminaux de ce type.

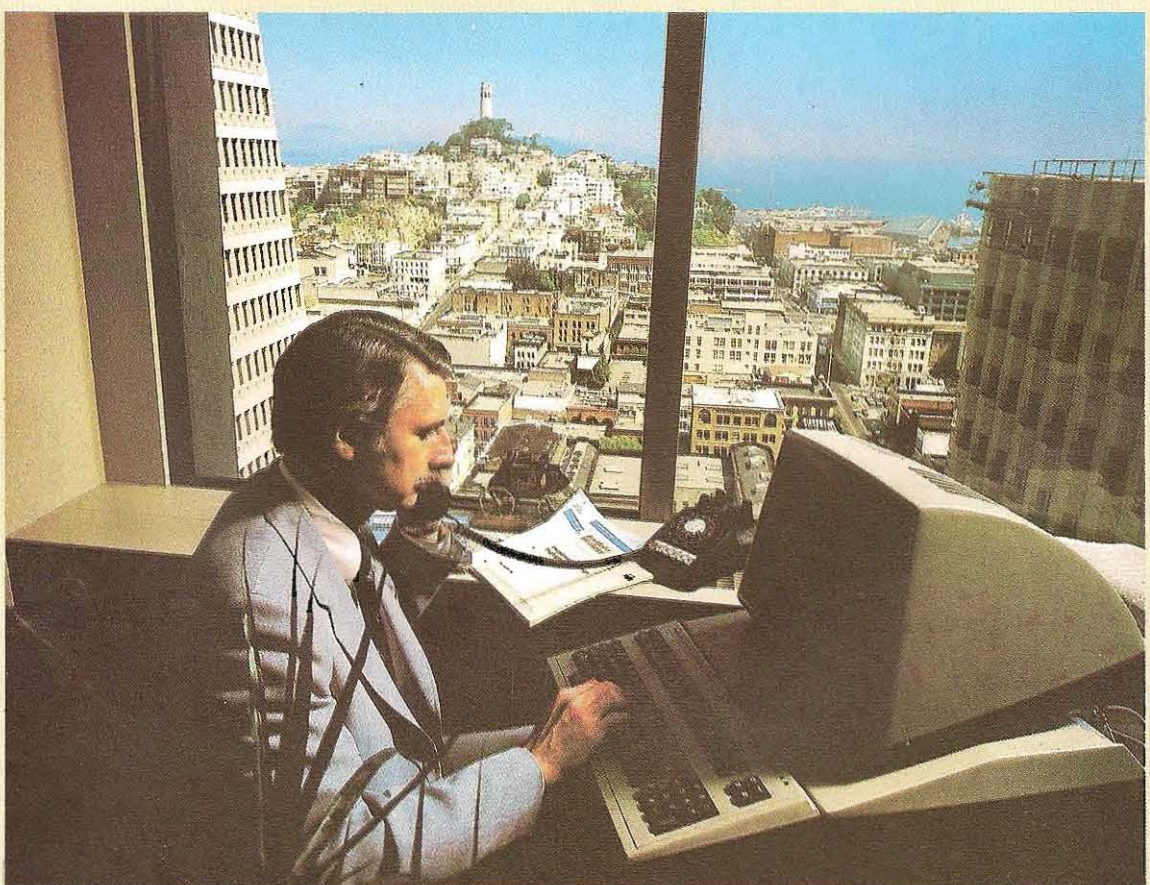
Il est également possible d'adapter un terminal domestique au système Vidéotex. En effet, un terminal domestique est plus puissant qu'un téléviseur aménagé ; on pourrait même l'utiliser chez soi comme un vrai ordinateur personnel.

La Chemical Bank, au stade expérimental dont nous avons parlé, ne permet pas aux clients d'accéder directement à son fichier, mais seulement à une copie stockée en mémoire dans un ordinateur programmé à cet effet. La copie est mise à jour et enregistrée quotidiennement, en fin de journée.

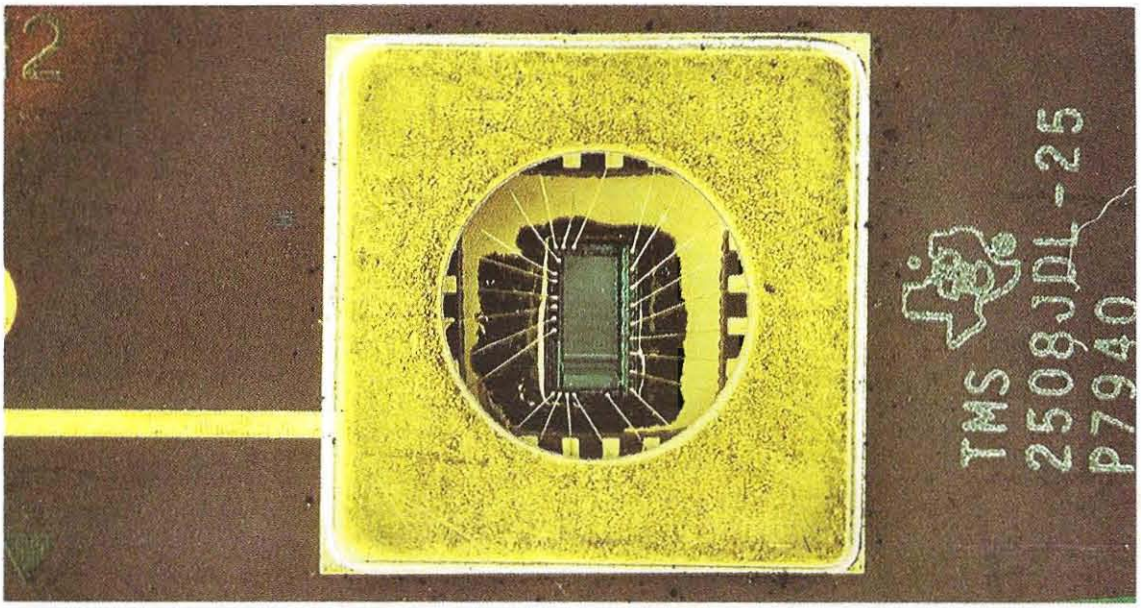
La Verbraucher Bank, quant à elle, donne directement accès à son fichier, mais avec un contrôle très strict qui exige non seulement le numéro de compte mais aussi trois codes susceptibles de modifications, d'opération en opération.

(D'après un article de C. Merli et L. Gibin, INFORMATICA 70, n° 95, mars 1982.)

Les ordinateurs modernes sont d'un emploi relativement facile ; ils sont utilisables même par des personnes peu familiarisées avec l'électronique.



Prime



Marka

Sur cette puce de silicium, véritable joyau de l'électronique, on distingue clairement les liaisons entre module et pattes (ou broches).

Dans un dispositif à deux entrées, chacune d'elles pouvant être dans l'état **H** ou **L**, les différentes combinaisons sont les suivantes :

Première entrée	Seconde entrée
H	L
H	H
L	H
L	L

Lorsque deux entrées sont liées entre elles, elles se trouveront dans le même état : ou toutes deux **H** ou toutes deux **L**.
Les états possibles sont donc :

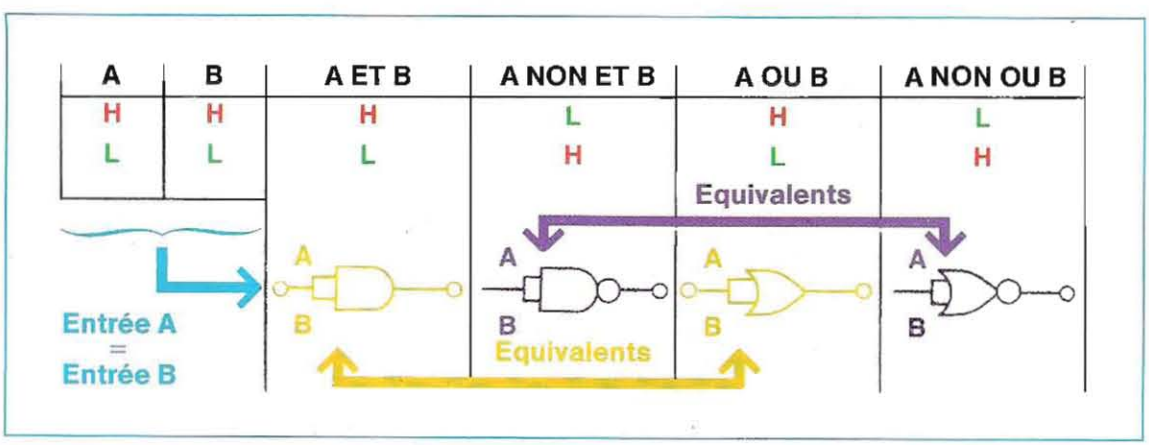
A	B
H	H
L	L

Si l'on applique à ces états les opérateurs ET, NON ET, OU, NON OU, nous obtenons le schéma du bas de la page.
On remarquera immédiatement qu'avec ce type de relation, on obtient :

ET = OU = amplificateur non inverseur
NON ET = NON OU = amplificateur inverseur

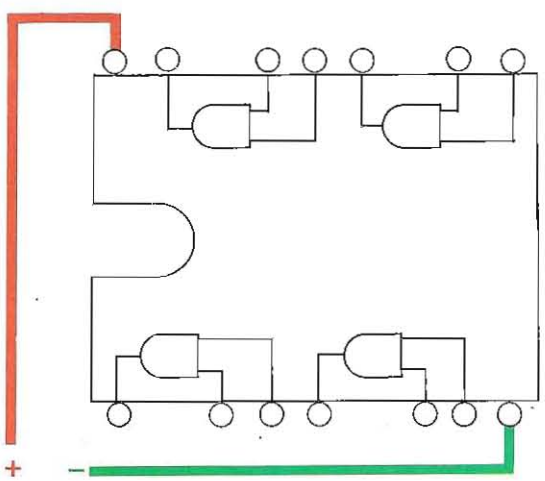
Le circuit OUX ne peut pas être relié de cette manière puisque sa sortie serait dans tous les cas :

L (**H** OUX **H** = **L** ; **L** OUX **L** = **L**).



Les circuits intégrés

Nous venons de parler des circuits capables d'effectuer certaines fonctions, et nous avons étudié leur comportement sous l'angle logique, mais sans envisager la façon de les fabriquer.
Dans les débuts de l'électronique numérique, les circuits étaient constitués de composants discrets, c'est-à-dire réalisés séparément et qu'il fallait relier entre eux pour constituer un circuit complet. Par la suite, on a pu rassembler tous les constituants d'un circuit sur une seule plaquette et obtenir avec un seul composant des fonctions dont l'exécution exigeait jusque-là des dizaines d'éléments séparés. Ce nouveau type de composant a reçu le nom de **circuit intégré ;** il est d'une utilisation extrêmement simple. Il suffit de brancher les entrées et les sorties et de prévoir une alimentation électrique.
Les circuits intégrés les plus courants (que l'on appelle circuits TTL d'après leur structure) ne demandent que 5 volts et peuvent donc fonctionner avec une pile sèche.
Un même boîtier rassemble en général plusieurs circuits complets. L'un des plus répandus est celui qui accomplit la fonction ET, et où l'on trouve quatre de ces circuits, ou même davantage.
Ce boîtier comporte quatorze branches, trois par circuit (chaque circuit ET possède en fait deux entrées + une sortie) et deux broches pour l'alimentation générale.
Les constructeurs fournissent pour chaque modèle le schéma du branchement.
On trouvera, en bas de cette page à gauche, le schéma du circuit intégré ET.

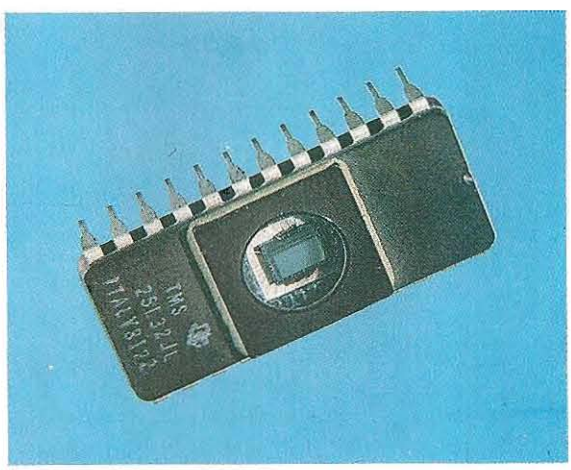


En dépit de la dimension relativement réduite des circuits intégrés, leur utilisation pour la construction d'un ordinateur conduirait à réaliser des machines volumineuses.
La miniaturisation permet heureusement de rassembler dans un seul boîtier la totalité des circuits nécessaires à l'exécution des fonctions principales d'un ordinateur : c'est le **microprocesseur** (**μ P**) ; il constitue le cœur de l'ordinateur.

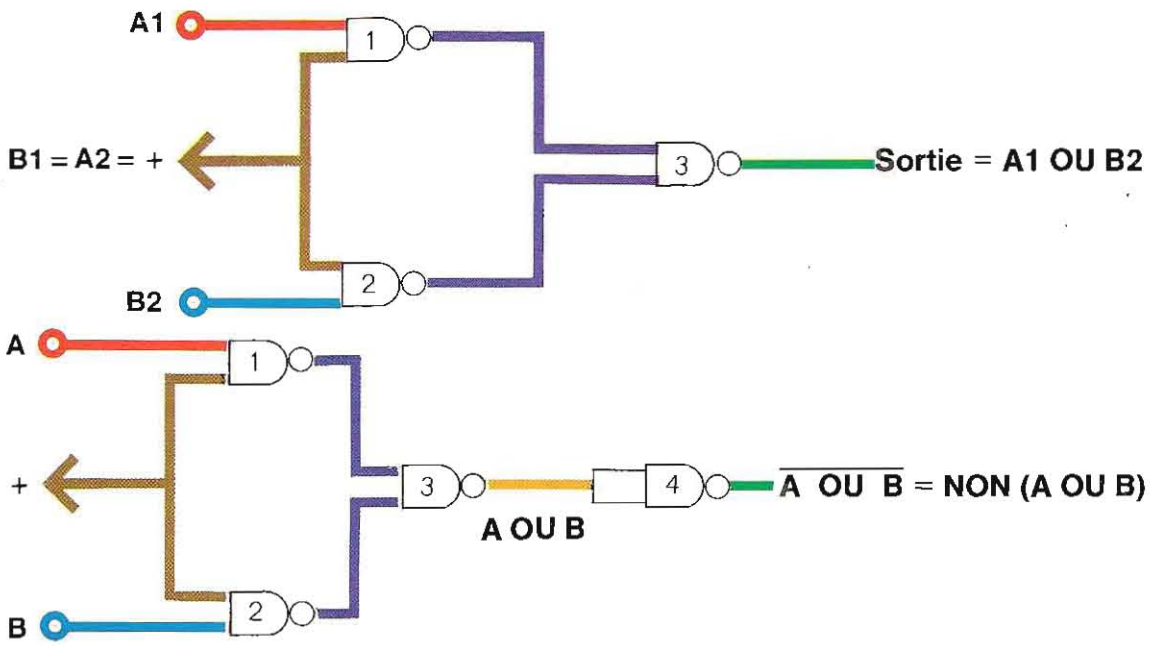
Les circuits composites

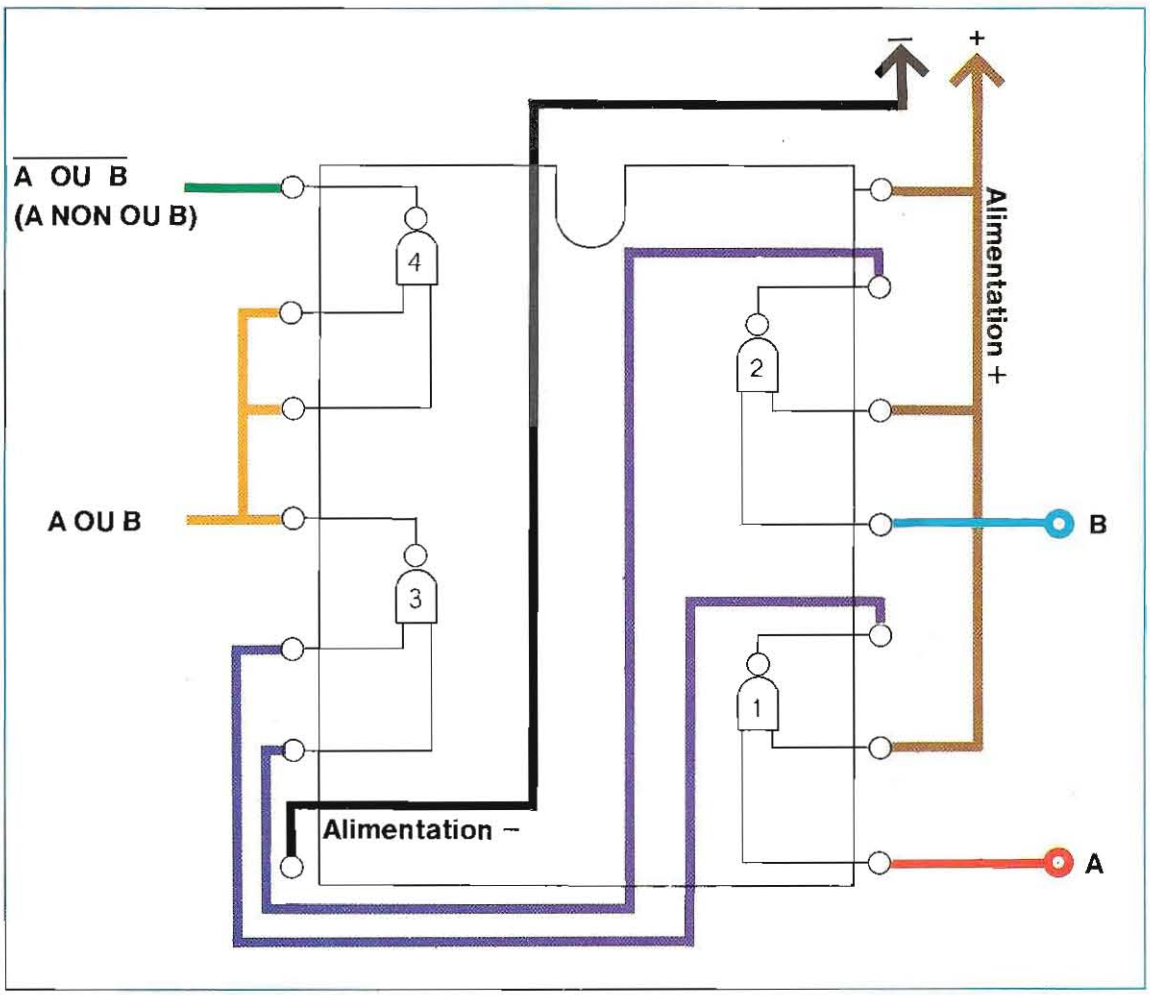
Les circuits que nous venons de voir peuvent être reliés entre eux pour obtenir une fonction logique ; la connexion des différents circuits n'exige aucun soin particulier à condition toutefois qu'ils appartiennent à la même famille.
La page 98 (en haut) montre un montage réalisé avec des circuits NON ET (NAND) pour accomplir la fonction OU.
On peut immédiatement vérifier son fonctionnement par la table de vérité et en observant que les circuits 1 et 2 ont l'une de leurs entrées (respectivement B_1 et A_2) au niveau H.
Rappelons que le niveau H peut être le pôle positif de l'alimentation, le niveau L étant le pôle négatif. Sur le schéma, les entrées B_1 et A_2 sont donc reliées au pôle plus (+). En connectant en sortie un autre NON ET on obtient la fonction NON OU (NOR) ; c'est ce qu'illustre le second schéma en haut de la page 98. Les branchements à effectuer pour obtenir les fonctions OU et NON OU figurent sur le graphique du bas de la page 98.

Une Eprom de Texas Instrument. La fenêtre transparente protège la pastille de silicium et permet aux rayons ultraviolets d'effacer le contenu de la mémoire.



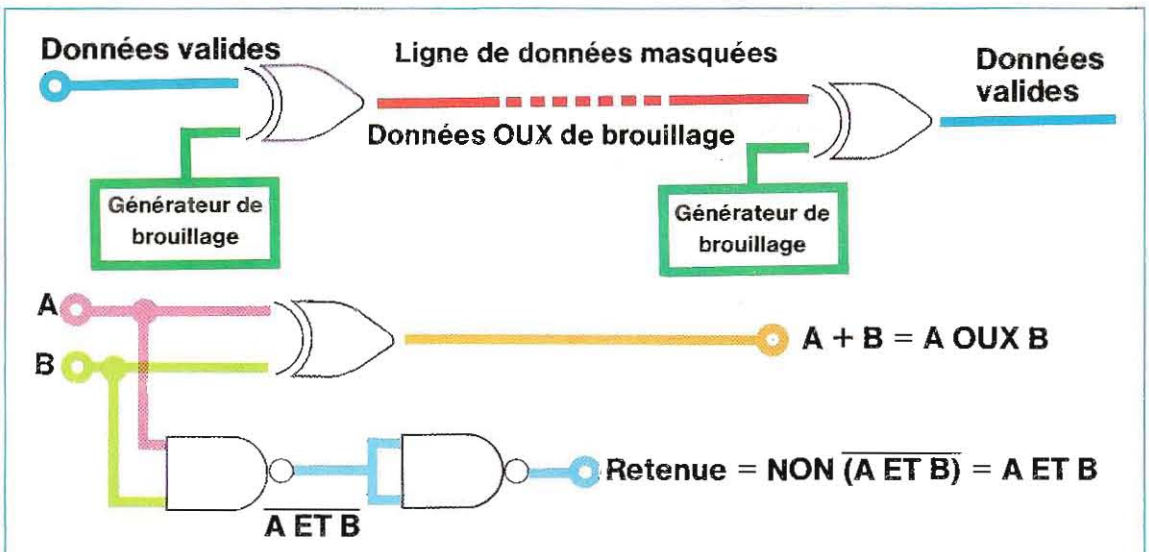
Archives Curcio/Buffetti Data


A1
B1 = A2 = +
B2
Sortie = A1 OU B2
A
+
B
A OU B
A OU B = NON (A OU B)


A OU B
(A NON OU B)
A OU B
Alimentation +
Alimentation −
B
A

Les applications

Nous avons expliqué que ces circuits ont une fonction principale de séparateur. Nous allons en étudier deux applications particulières.
La première concerne la sécurité du transport des données : les réseaux d'ordinateurs permettent en effet de transmettre à grande distance, et les données transmises risquent d'être interceptées et modifiées ; d'où la nécessité d'adopter des méthodes de protection.
Ces méthodes sont nombreuses et plus ou moins sûres. L'une des plus simples (quoique suffisamment sûre) consiste à envoyer sur la ligne et en le superposant aux données, un brouillage constitué d'une série d'impulsions aléatoires qui les rend indéchiffrables. A l'arrivée, il faut injecter un brouillage semblable en utilisant le même générateur de bruit pour obtenir un signal dépouillé, composé uniquement des données utiles. Le système qui effectue le mixage des signaux au départ et leur séparation à l'arrivée est un simple circuit OUX (XOR), ce qui signifie l'un ou l'autre, mais pas les deux. On en trouvera le schéma sur cette page. Supposons qu'au départ les données soient de niveau H : si l'on injecte comme brouillage un bruit de niveau H, le résultat est un niveau L (H OUX H = L) ; à l'arrivée, il n'est possible de définir si ce niveau L est un véritable niveau L, ou bien un niveau H masqué, qu'en injectant le même brouillage qu'au départ (H) : on obtient ainsi un résultat (brouillage) OUX L (données d'arrivée) = H (données vraies).
La seconde application est le circuit qui permet de réaliser un additionneur binaire. Au bas du schéma de cette page, le lecteur trouvera la table de vérité correspondante. On notera qu'en utilisant les NON ET, il faut faire appel à deux circuits, le second ne fonctionnant que comme inverseur. Si l'on remplace le circuit 1 par un circuit ET, le second devient inutile, la sortie étant directement A ET B.



A	B	Somme	Retenue
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Autres types de circuits intégrés

Les circuits intégrés utilisés dans la construction d'un système de micro-ordinateur ne se limitent pas aux types que nous avons présentés.

Les circuits accomplissant les fonctions d'opérateur logique (ET, OU, OUX, etc.) sont les plus simples et les plus utilisés ; lorsqu'on augmente la spécialisation d'un circuit, on augmente aussi sa complexité, jusqu'au microprocesseur, qui renferme des milliers de composants sur un même circuit intégré.

Nous allons décrire quelques circuits plus complexes courants. Les circuits à microprocesseur seront abordés plus loin.

Temporisateur (timer en anglais) : dispositif fournissant des signaux périodiques sous forme d'ondes, ou oscillations ; on l'emploie en général pour la création d'impulsions. Il en existe deux types fondamentaux : à réseau RC (circuit résistance et capacité, connecté à un oscillateur) et à quartz. Dans un oscillateur à circuit RC, la durée de l'impulsion est réglée par un circuit composé d'une résistance et d'un condensateur : ce sont les plus économiques et les moins précis. Le circuit à quartz utilise un cristal de quartz dont la vibration mécanique commande la durée de l'impulsion : ils sont plus coûteux, mais beaucoup plus précis. Ce dernier type est utilisé pour générer des signaux dans les systèmes à micro-ordinateur. La fréquence est alors de quelques mégahertz (MHz), ou millions d'oscillations par seconde.

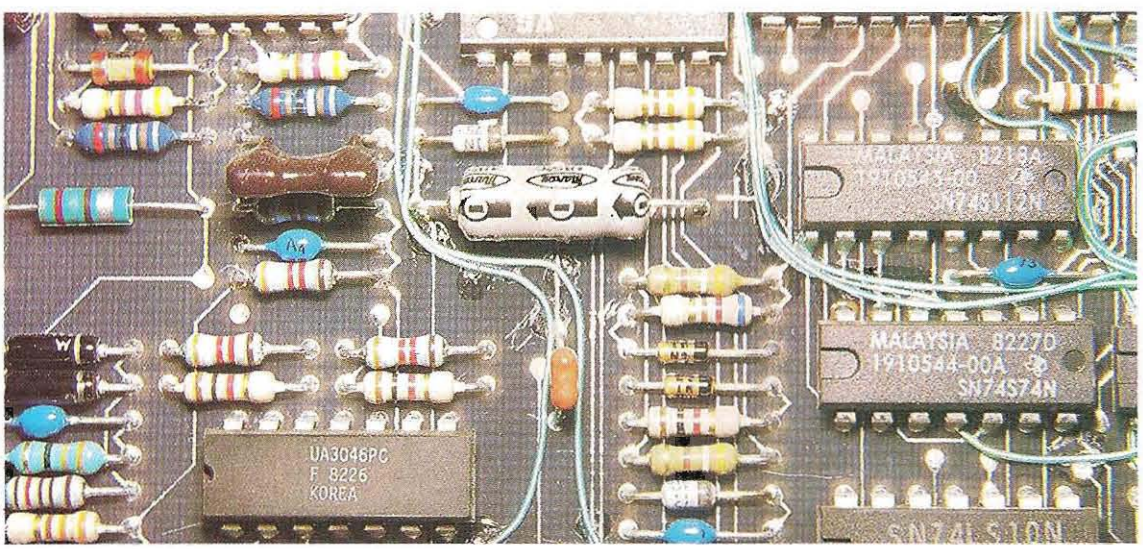
Bascule (trigger en anglais) : circuit à déclenchement, c'est-à-dire sensible au niveau du signal d'entrée. Quand ce niveau dépasse une valeur déterminée, le circuit « bascule » et la sortie change d'état, passant du niveau bas au niveau haut, ou vice versa. Ces circuits servent comme régénérateurs de signaux.

Diviseur (divider en anglais) : circuit capable de fournir en sortie une quantité d'impulsions égale à celle d'entrée divisée par un nombre quelconque. Il en existe deux types, à division fixe ou programmable. Les diviseurs fixes fournissent en sortie le nombre d'impulsions d'entrées divisé par une quantité fixe, qui dépend du branchement du circuit. Un diviseur programmable permet au contraire un réglage du diviseur.

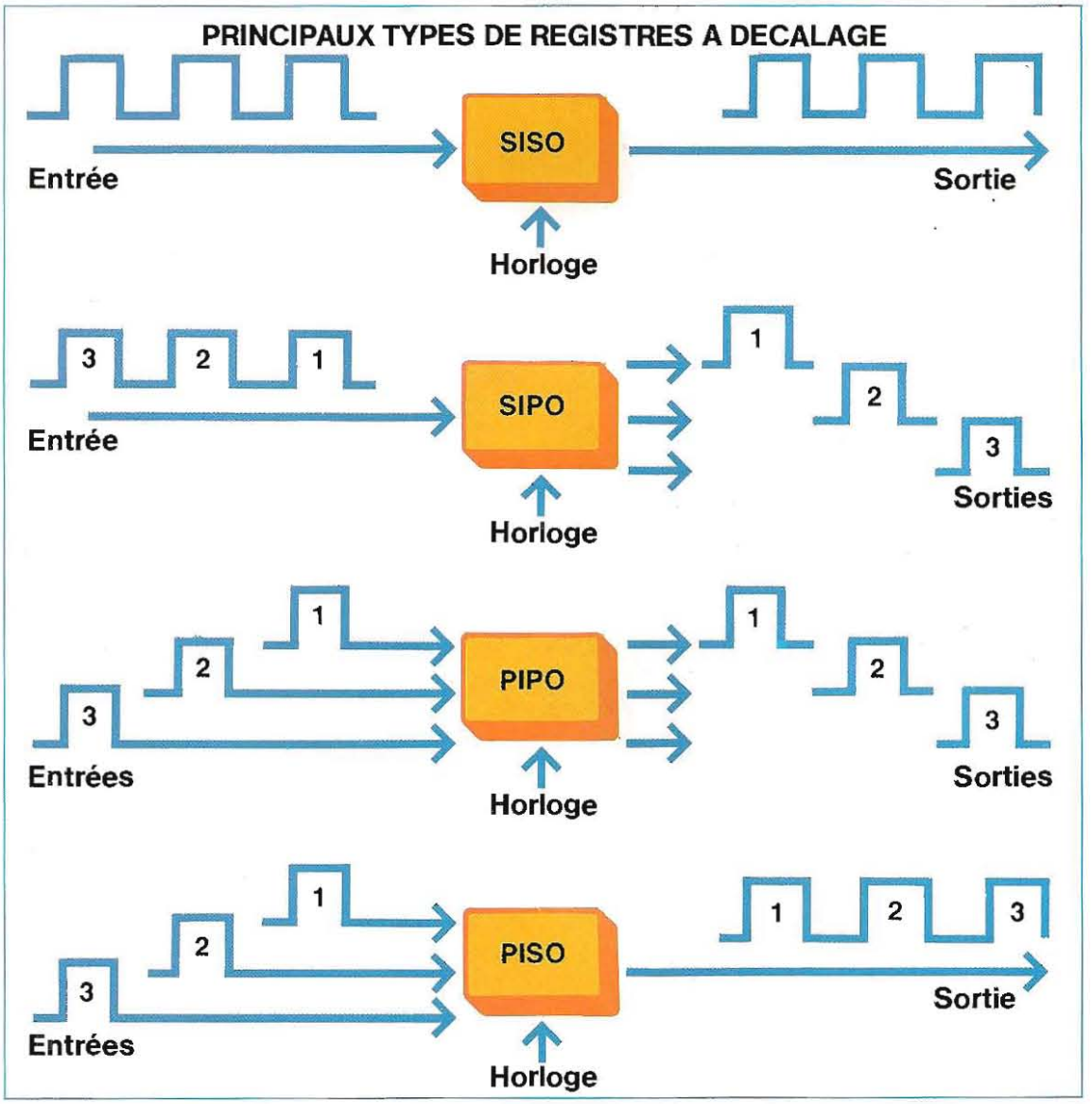
Bascule bistable (flip-flop en anglais) : circuit à deux états (ouvert-fermé, ON-OFF) dans lequel la sortie ne se modifie pas immédiatement à réception du signal d'entrée. Le circuit mémorise le changement d'état de l'entrée et le suit, à réception d'un autre signal (ou impulsion) dénommé signal d'horloge (clock).

Il s'agit d'un dispositif bistable, c'est-à-dire possédant deux états stables qui suivent l'impulsion d'entrée avec retard, à réception du signal d'horloge. Ce circuit sert pour la division par deux ou comme mémoire. Il est de type synchrone, la communication se faisant en coïncidence avec le signal d'horloge.

Détail de l'intérieur d'un ordinateur. Sur cette plaquette on voit plusieurs circuits intégrés.



Centre THC



Registre à décalage (shift-register) : dispositif constitué de bascules bistables correctement assemblées. La donnée (impulsion) se déplace d'une cellule à l'autre à l'arrivée de chaque signal d'horloge.
La donnée est chargée dans la première cellule ; au premier signal d'horloge, elle passe à la seconde cellule, et ainsi de suite. Il existe différents types de circuits de ce genre, selon leur mode de fonctionnement.
Les principaux sont :
Entrée série, sortie série (SISO, de l'anglais Serial Input Serial Output). Les données sont chargées en série (une à la fois) et sortent en série, en coïncidence avec les signaux d'horloge.
Entrée série, sortie parallèle (SIPO, de l'anglais Serial Input Parallel Output). Les impulsions d'entrée sont chargées, puis présentées sur un nombre équivalent de sorties diverses. Par exemple, quatre impulsions d'entrée sont mémorisées et présentées en même temps sur quatre sorties différentes.
Entrée parallèle, sortie parallèle (PIPO, Parallel Input Parallel Output). L'entrée et la sortie acceptent en même temps plusieurs impulsions.
Entrée parallèle, sortie série (PISO, Parallel Input Serial Output). L'entrée accepte des impulsions simultanées qui sont mémorisées et présentées en série sur une sortie unique (schéma de cette page).

Codes secrets contre les pirates de logiciels

Les énormes progrès réalisés dans le domaine des techniques télématiques de transmission des données permettent de réduire considérablement le délai technique indispensable à l'échange des informations entre personnes ou bureaux que séparent des milliers de kilomètres. Des données confidentielles stockées dans les archives électroniques, concernant, par exemple, les comptes bancaires ou les secrets industriels, circulent quotidiennement sur les réseaux de télécommunications. Elles sont plus nombreuses chaque jour, et leur interception, ainsi que leur manipulation, seraient chose facile si on n'assurait pas leur protection.

Les cas les plus célèbres, ceux qui ont eu les honneurs de la presse mondiale, sont ceux dans lesquels sont intervenus les « pirates de logiciels ». Des personnes ingénieuses, et dépourvues de tout scrupule, ont trouvé le moyen d'accéder aux réseaux de transmission de données de plusieurs grosses banques internationales, s'informant ainsi sur d'importantes transactions financières.

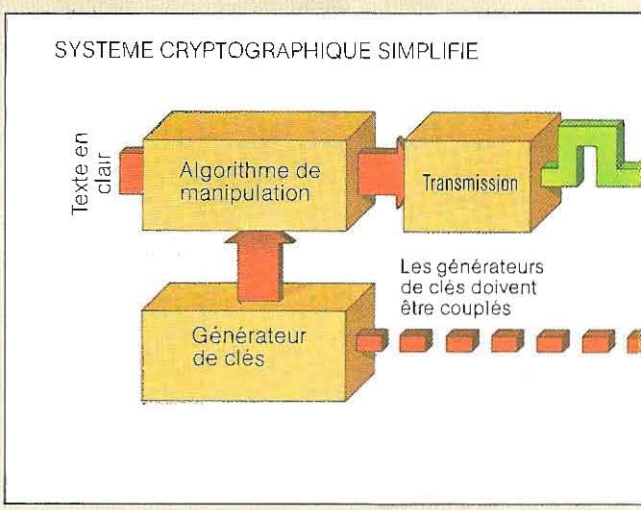
Les précautions adoptées pour les procédures de transmission sont aussi variées que complexes, mais ne suffisent pourtant pas et n'ont pas empêché quelques spécialistes malhonnêtes d'organiser pour leur propre compte le transfert de sommes importantes. Des nouvelles de ce genre sont soigneusement étouffées par les établissements bancaires, qui ne tireraient aucun profit de cette publicité, mais elles ont plusieurs fois fait leur chemin jusqu'aux premières pages des journaux, et ont provoqué l'inquiétude du public. La piraterie au niveau du logiciel n'est pas toujours de type financier. Parmi les communications confidentielles qui circulent chaque jour sur les lignes téléphoniques du monde entier, on trouve aussi des informations sur les inventions et les procédés industriels couverts par des brevets, ou encore des messages échangés par des organes politiques internationaux. Il est plus que jamais indispensable de mettre au point de nouveaux systèmes de **cryptographie.**

Le cryptage, ou codage, ou chiffrement, est le procédé qui consiste à traduire un texte original écrit en clair, lettre par lettre, en un message codé contenant toutes les lettres du texte original, mais dans une autre position **(codage par transposition),** *ou avec remplacement de certaines lettres par d'autres* **(codage par substitution).** *Dans la pratique, on peut combiner les deux méthodes en les appliquant une ou plusieurs fois dans le même texte. Un système ingénieux de codage par transposition où toutes les lettres sont mélangées plusieurs fois* **(codage par transposition double ou triple),** *appliqué à des textes relativement longs, peut constituer un système de cryptographie valable.*

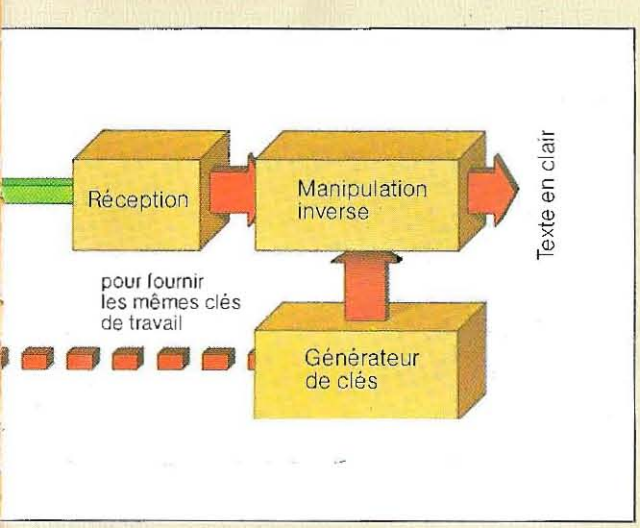
Voici un exemple très simple de codage par transposition:

V I E N S I
M M E D I A
T E M E N T
A V E C M A
T E R I E L

Transcrit horizontalement, le texte se lit: VIENS IMMEDIATEMENT AVEC MATERIEL, mais le message pourrait être codé par lecture verticale: VMTAT IMEVE EEMER NDECI SINME IATAL. Un spécialiste se rendrait immédiatement compte qu'en assemblant successivement les premières lettres de chaque groupe, puis les secondes et ainsi de suite, le texte prend une signification. Il faut donc adopter une méthode plus compliquée. Un autre codage simple par substitution consiste à déplacer d'un cran chacune des lettres de l'alphabet: A devient B, B devient C, etc. Le message « VIENS IMMEDIATEMENT » se lit alors: « WJFOT JNNFEJBUFNFOU ». Une transformation de ce genre peut paraître



satisfaisante à première vue, mais chaque langue possédant une structure propre, la solution du problème n'est qu'un jeu.
Chaque lettre du texte original étant toujours représentée par la même lettre du code, ce dernier appartient à la catégorie mono-alphabétique.
Au lieu de déplacer d'un seul cran les lettres de la phrase VIENS IMMEDIATEMENT, on pourrait déplacer les lettres successives en augmentant chaque fois d'une unité le nombre de crans, ce qui donnerait pour VIENS le mot WKHRX. De cette manière, n'importe quelle lettre du texte original peut en théorie être représentée par n'importe quelle lettre du texte chiffré. Le décryptage du message exige alors une analyse mathématique beaucoup plus poussée.
Les sytèmes de cryptage plus élaborés exigent l'application de deux procédures séparées. La première consiste à déplacer chaque lettre suivant un schéma prédéterminé, que l'on appelle **algorithme**, qui sera ensuite répété en sens inverse pour retrouver le texte original. La seconde procédure consiste à se munir d'une **clé** indiquant comment appliquer l'algorithme. De la longueur de cette clé dépend le degré de sécurité des codes de substitution. Une clé de longueur illimitée et comprenant des lettres et des nombres décimaux ou binaires synthétisés et disposés de manière aléatoire constituerait un code de chiffrement imbattable.
La plus grande garantie de sécurité que puisse offrir un code polyalphabétique (ou son équivalent sous forme numérique) réside dans le nombre des alphabets de substitution. En d'autres termes, la lettre A qui apparaît dans le texte dans des positions diverses peut être traduite selon les cas par n'importe quelle autre lettre de B à Z; c'est ce qu'il faudra déchiffrer correctement pour retrouver A. On peut aussi par mesure de précaution prévoir que les 25 alphabets de substitution (autant que les lettres de l'alphabet, moins la lettre codée) seront utilisés dans un ordre aléatoire, ce que l'on obtient facilement avec une clé de travail aléatoire de longueur illimitée.
Ce principe est celui de la chaîne à usage unique, visualisation d'une succession aléatoire de lettres ou de nombres décimaux ou binaires, dans laquelle chaque lettre clé indique de combien de crans chacune des lettres du texte doit être déplacée pour le déchiffrement.
Une cryptographie de ce type assure toute sécurité et défie les efforts du cryptographe pour découvrir le code; le système présente cependant des difficultés pour celui qui reçoit le message, et il est d'emploi compliqué et coûteux. Il exige aussi la production et la distribution de copies des clés d'une longueur suffisante pour couvrir le nombre total des lettres transmises sur une période de quelques semaines ou de quelques mois.
Dans la pratique, beaucoup de codes polyalphabétiques utilisent des clés de longueur limitée, fondées sur des dispositions comme les carrés de Vigenère ou les carrés inverses de Beaufort. Dans ce cas, la clé peut être une phrase brève, avec des clés additionnelles fournies, par exemple, par le message lui-même. Les techniques de résolution des codes polyalphabétiques ont d'ailleurs été renforcées par l'existence d'ordinateurs à grand débit. Cependant, on n'a pas toujours besoin de systèmes définitivement indéchiffrables: il suffit le plus souvent qu'ils imposent un retard à la découverte du texte original ou qu'ils impliquent un coût de déchiffrement démesuré et décourageant pour un éventuel utilisateur.
Au cours des dix dernières années, la cryptographie a remarquablement progressé grâce aux ordinateurs numériques eux-mêmes. En électronique, nous le savons, on dit qu'un système est numérique lorsque les signaux et les formes d'ondes qu'il élabore sont en nombre restreint, par opposition à un système analogique classique où les signaux traités peuvent adopter un nombre infini de formes et de niveaux. Le système binaire est



un système numérique qui ne prévoit que deux états différents, le signal étant représenté par l'une des deux conditions ON-OFF, ou signal présent-signal absent, ou encore 1-0. En un sens, toutes les transmissions chiffrées, par voie hertzienne ou par câble, exigent le recours à un code numérique. Le Morse et le langage des téléscripteurs par exemple sont des codes numériques, même si pendant longtemps les messages étaient d'abord transcrits sous forme de lettres avant d'être transmis en alphabet Morse. On tend actuellement à codifier le texte en clair directement sous une forme numérique, en utilisant la modulation à impulsion codifiée. L'étape suivante consiste à transformer en chiffres le flux de chiffres binaires (bits) en les combinant avec une clé de travail numérique. Grâce aux appareils électroniques à grand débit, tout le processus peut être exécuté en temps réel quelle que soit la vitesse requise, qu'il s'agisse d'un texte écrit ou d'un discours parlé. L'utilisation d'une clé numérique constitue un chiffrement par substitution polyalphabétique. Si la clé est véritablement aléatoire et de longueur illimitée, le codage est très sûr.

Cela présente cependant des difficultés opératoires semblables à celles que l'on rencontre dans le déchiffrement analogique avec les clés à usage unique.

En pratique, on extrait une clé pseudo-aléatoire d'une autre clé relativement brève (en général inférieure à 100 bits), en l'agençant de telle manière que l'on rencontre de longues séries de 1 ou de 0 avant répétition de la séquence. La méthode d'établissement de la séquence pseudo-aléatoire, nous l'avons vu, a un effet très important sur la sécurité du cryptage. En effet, si des systèmes numériques à base de « logique linéaire », offrent de nombreuses possibilités de codifications, celles-ci sont cependant aisément déchiffrables. Un petit ordinateur suffirait, à condition qu'une partie du message (par exemple l'adresse d'ouverture) soit connue de l'analyste.

C'est un risque qui a conduit des firmes comme IBM à confectionner des systèmes cryptographiques faisant appel à un algorithme non linéaire tout en utilisant un nombre relativement réduit de bits clés. Chacun des blocs du texte est soumis, avec sa clé, à une série complexe de transformations, telles que la transposition de l'ordre et les substitutions fondées sur de longues séquences-clés dérivées de quelques clés d'origine. Ces systèmes sont considérés comme imbattables.

Exemple 1

CLE	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
DEPLACEMENT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

TEXTE EN CLAIR	V	I	E	N	S	I	M	M	E	D	I	A	T	E	M	E	N	T	A	V	E	C	M	A	T
CLE DE TRAVAIL	C	F	Z	A	L	H	T	R	P	A	S	H	R	Y	F	N	Q	O	B	E	G	R	A	M	N
DEPLACEMENT	3	6	26	1	12	8	20	18	16	1	19	8	18	25	6	14	17	15	2	5	7	18	1	13	14
TEXTE CHIFFRE	Y	O	E	O	E	Q	G	E	U	E	B	I	L	D	S	S	E	I	C	A	L	U	N	N	H

Exemple :
La lettre H de la clé transforme le I en Q, ou le A en I.
La lettre R de la clé transforme le M en E, ou le T en L, ou le C en U.

En 1977, l'organisme américain de normalisation, ANSI (American National Standard Institute), a donné son approbation à l'algorithme DES (Data Encryption Standard) fondé sur une clé de 56 bits. Chaque bloc de données subit dix-huit étapes de manipulation, pour lesquelles seize clés cryptographiques différentes sont dérivées d'une clé principale à 56 bits de manière à former une clé dont la longueur totale est de 10^{17} bits. Les utilisateurs du système font appel à un algorithme publié, mais utilisent des générateurs de clés couplés pour lesquels la clé principale à 56 bits

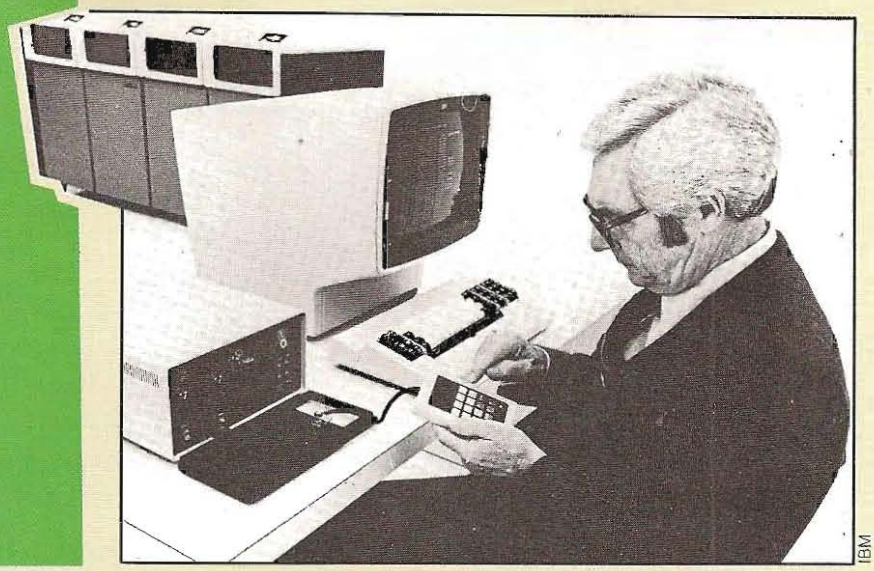
(ou plutôt 64, avec la codification interne) est tenue secrète. Le générateur de clés peut être un petit module scellé, mais l'essentiel est de le garder en sécurité, ce qui entraîne des problèmes pour la gestion et la distribution des clés. On estime aujourd'hui que ce système présente une sécurité suffisante pour tous les usages commerciaux courants. Les systèmes numériques comme DES peuvent être appliqués directement aux ordinateurs pour protéger des archives confidentielles ou des transmissions par téléphone et par radio de données et de messages. Certains spécialistes pensent toutefois que la clé principale de 56 bits est trop courte pour que le système garantisse une sécurité totale. Bien que les systèmes cryptographiques DES soient progressivement adoptés par les organisations bancaires et commerciales, et même si personne n'a encore réussi à démontrer publiquement la possibilité de les violer, plusieurs groupes de spécialistes américains ont récemment proposé un système plus puissant. Il s'agit de la cryptographie à clé publique, dont l'intérêt est d'éliminer tous les problèmes de gestion et de distribution des clés. Dans un tel système, les fonctions de codage et de décodage sont distinctes. Ainsi, l'émetteur ne possède pas la clé de déchiffrement; mais il assure le codage du message pour un destinataire spécifique par l'emploi d'une liste connue de clés de codage. Ces systèmes cryptographiques ont aussi l'avantage de supprimer les difficultés liées à la distribution des clés, et à la protection des générateurs de clés ou des chaînes à usage unique.

Finalement, l'attention apportée par les spécialistes à l'établissement des systèmes de cryptographie permet d'espérer que l'on pourra profiter de la technologie télématique sans courir le risque de retrouver son compte en banque à découvert au profit d'un inconnu trop bien familiarisé avec l'ordinateur.

Exemple 2

A	00001	N	01110
B	00010	O	01111
C	00011	P	10000
D	00100	Q	10001
E	00101	R	10010
F	00110	S	10011
G	00111	T	10100
H	01000	U	10101
I	01001	V	10110
J	01010	W	10111
K	01011	X	11000
L	01100	Y	11001
M	01101	Z	11010

TEXTE EN CLAIR	V	I	E	N	S
TEXTE TRADUIT EN SYSTEME BINAIRE	10110	01001	00101	01110	10011
CLE NUMERIQUE	01100	11001	00110	10101	01011
TRANSMISSION CODEE	11010	10000	00011	11011	11000
FORME D'ONDE					



IBM

TEST 2

1 / Compléter la table de vérité suivante :

A	B	$\overline{A \text{ OUX } B}$	A ET $\overline{(A \text{ OUX } B)}$
1	0		
1	1		
0	0		
0	1		

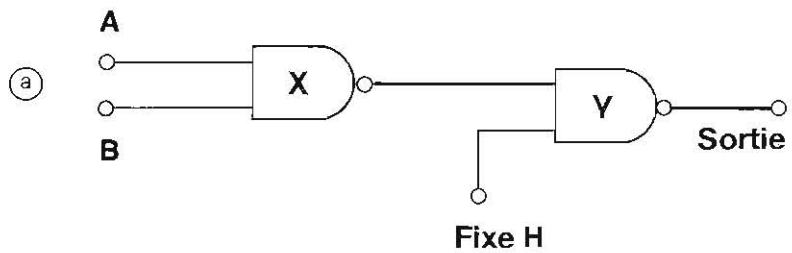
2 / Résoudre les opérations suivantes :

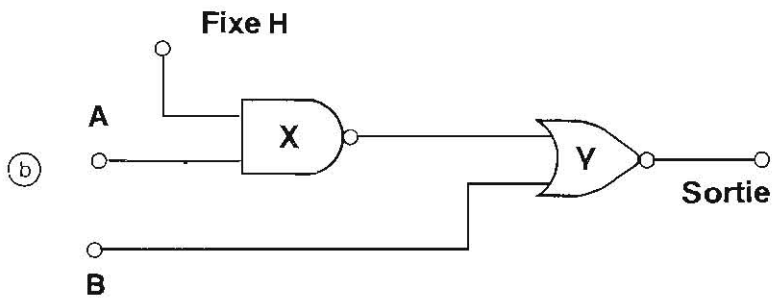
a) 11011 ET 011
b) 12 ET 7 (12 et 7 comme nombres décimaux)
c) 12 ET 7 (12 et 7 comme nombres octaux)
d) 1101 OU 0110
e) 1011 OUX 1101

3 / Dessiner les symboles graphiques et la table de vérité des circuits :

a) NON b) ET c) OU d) OUX

4 / Ecrire la table de vérité des circuits :





Les exercices 3 et 4 ne sont pas déterminants pour la poursuite de la lecture. On peut considérer le sujet comme assimilé si le lecteur sait répondre à l'exercice 1 et à 3 parties au moins de l'exercice 2. Les solutions du test se trouvent page 122.

Les codes de transmission des données

Pour qu'un ordinateur fonctionne correctement, il faut lui permettre d'interpréter les données et les informations qui lui sont fournies. Si la transposition d'une donnée numérique en forme binaire est immédiate, les données non numériques doivent être traduites sous une forme compréhensible par la machine. On utilise donc un code de transmission des données.

Bit et byte

La représentation binaire d'un nombre est une série de symboles **0** et **1** : chacun de ces symboles prend le nom d'élément binaire (binary digit: **bit**). Par exemple, la notation binaire du nombre 35 est 00100011 (voir schéma ci-dessous).
La valeur binaire de chaque bit est déterminée par sa position: le bit placé en position 1 vaut **1**, le bit placé en position 3 vaut **4**, etc. Dans l'ordinateur, chaque bit est représenté par un signal électrique qui est de niveau haut (H) si le bit correspondant vaut **1** ou bas (L) si le bit vaut **0**.

Position	8	7	6	5	4	3	2	1
Puissance de 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur	128	64	32	16	8	4	2	1
35 décimal =	0	0	1	0	0	0	1	1

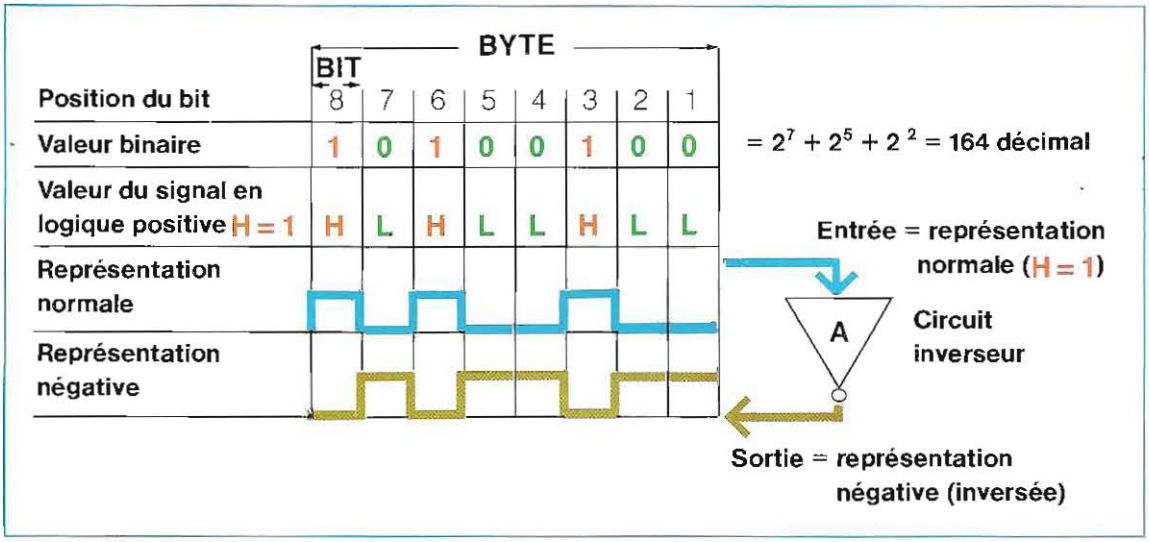
Le schéma ci-dessous représente une situation pour les signaux correspondant à 35 décimal.

Position = numéro du bit	8	7	6	5	4	3	2	1
Puissance de 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur binaire correspondant à 35 décimal	0	0	1	0	0	0	1	1
Niveau du signal	L	L	H	L	L	L	H	H
Représentation graphique								

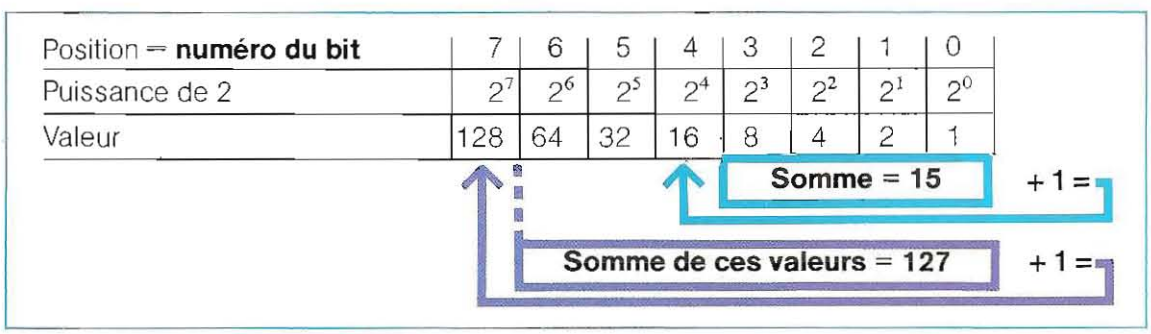
Niveau haut = H
Niveau bas = L

On suppose dans ce cas que la valeur binaire **1** correspond à un signal électrique de niveau haut; rien n'empêche d'adopter la logique inverse (valeur binaire **1** = niveau bas), c'est d'ailleurs le mode de fonctionnement (logique négative) de nombreuses machines.

Voici la représentation du nombre 164 décimal, en logiques positive et négative :

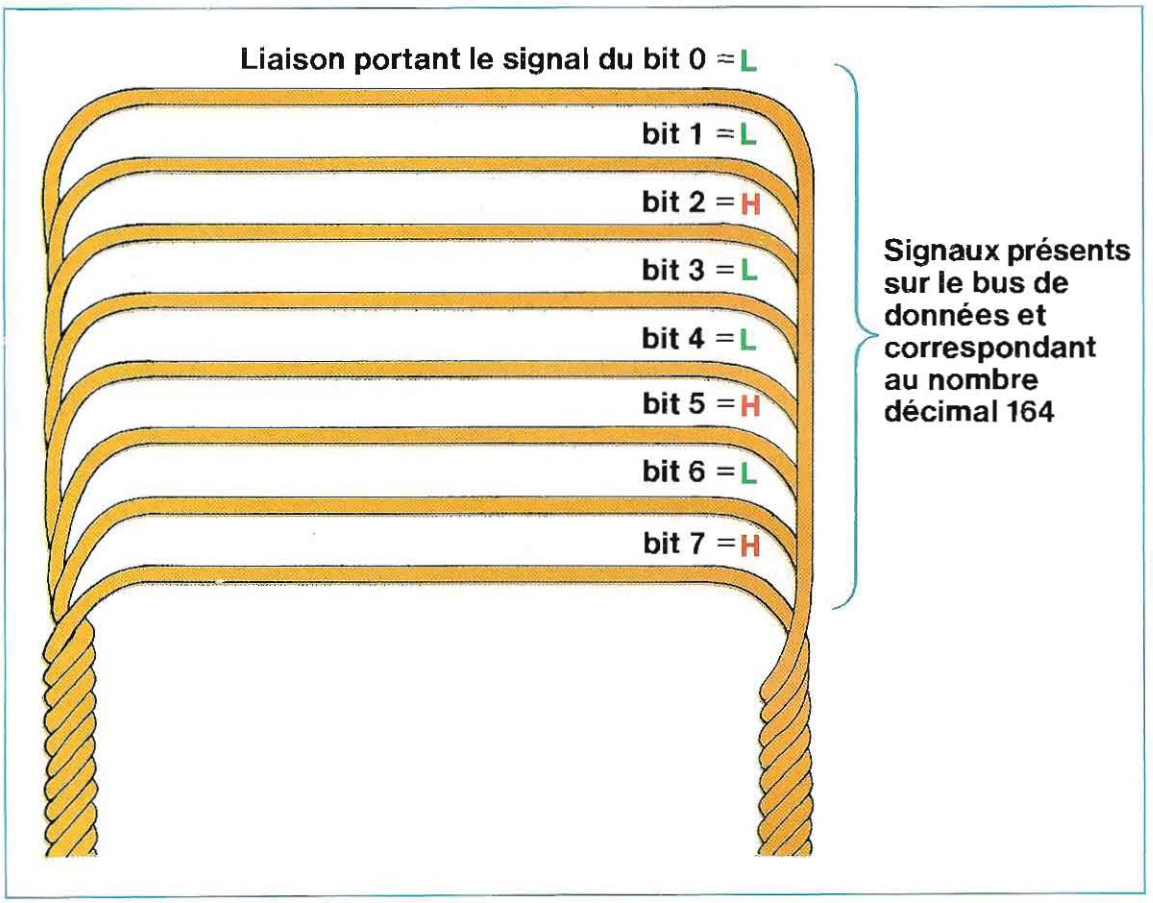


Ces deux modes de fonctionnement ne posent aucun problème dans la pratique, et pour passer de l'un à l'autre il suffit de disposer d'un circuit inverseur (A sur la figure).
Comme l'illustre le schéma, pour représenter en binaire le nombre 164 décimal, il faut un « multiplet » de 8 éléments binaires, soit 8 bits, (la valeur binaire est 10100100) ; c'est-à-dire un **octet** (**byte** en anglais).
La valeur numérique maximale que l'on peut écrire avec un octet s'obtient en prenant les 8 bits qui le composent égaux à 1 :
Valeur maximale de 1 octet : 11111111 = $= 2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = 255$.
Avec un bus de données à 8 bits (de 0 à 7), le nombre maximal que la machine peut utiliser est 255 : il ne sutfit donc pas pour les applications pratiques. Pour éliminer cette entrave, il existe des circuits particuliers qui permettent de regrouper plusieurs octets afin de former de très grands nombres.
En regroupant 2 octets, on obtient par exemple 16 bits et donc une valeur numérique d'environ 32000. Les représentations les plus utilisées comportent 2, ou 4 octets (32 bits). On notera qu'ajouter 1 seul bit équivaut à multiplier par 2 la valeur numérique maximale : en fait, la valeur numérique d'un bit quelconque est égale à la somme des valeurs des bits qui le précèdent, plus 1. Par exemple, le bit 4 qui apparaît dans le schéma ci-dessous (somme des bits 0,1,2,3 = $1+2+4+8=15$) vaut 16. Le bit 7 vaut 128 ; soit la somme des valeurs $64+32+16+8+4+2+1$, plus 1.



Pour représenter un nombre en langage machine, l'ordinateur dispose d'autant de liaisons électriques que de bits composant le nombre. La représentation d'un mot de 8 bits exige 8 liaisons, consacrées chacune à indi-

quer l'état du bit « transporté ». Ce groupe de liaisons prend le nom générique de **bus**. Les ordinateurs comportent deux bus pour assurer l'échange des données : l'un pour la circulation des données, l'autre pour les adresses de départ ou d'arrivée des données.
Dans un microprocesseur, le bus de données peut être de 8 bits (1 octet) et le bus d'adresse de 16 bits (2 octets). Le schéma en haut de la page 108 donne une numération des bits de 1 à 8 ; en utilisation courante, on préfère généralement la numération de 0 à 7 (voir le deuxième schéma de la page 108), car à partir de 0 le numéro successif de chaque bit en indique aussi la puissance de 2 (bit 0 : 2^0, bit 1 = 2^1, etc.).
Le schéma ci-dessous donne la situation des signaux pour un bus de données correspondant au nombre 164 décimal.



- **En notation binaire, la plus petite quantité est un état, ou un niveau. Sa valeur est 0 ou 1, et elle prend le nom de bit, ou élément binaire.**
- **Un byte de 8 bits est un octet. La valeur maximale (décimale) que l'on peut écrire avec un octet est 255.**
- **Le canal électrique sur lequel circulent les données est le bus de données : dans la majorité des micro-ordinateurs, il comporte 8 ou 16 liaisons et peut donc transférer, simultanément 8 ou 16 bits, c'est-à-dire 1 ou 2 octets.**
- **Pour prélever ou déposer une donnée, il faut fournir aux circuits électroniques l'adresse de cette donnée, c'est-à-dire sa position dans la mémoire.**
- **Les adresses des données circulent sur un connecteur électrique, le bus d'adresses. Ce bus est normalement de 16 bits, ou 2 octets pour une mémoire de capacité égale ou inférieure à 64 K octets.**

Le code ASCII

L'ordinateur, comme toutes les machines numériques, ne reconnaît que les symboles 0 et 1 (ou L et H). Pour pouvoir introduire des données (mots ou nombres) dans la machine, il faut donc avant tout les transformer en une série de 0 et de 1 ; de même, pour lire les données contenues dans l'ordinateur, il faut effectuer la transformation inverse, c'est-à-dire tirer d'une série de 0 et de 1 les lettres de l'alphabet ou les chiffres décimaux équivalents. La transformation en une série de 0 et de 1 s'appelle **codage** ; l'opération inverse est le **décodage**.

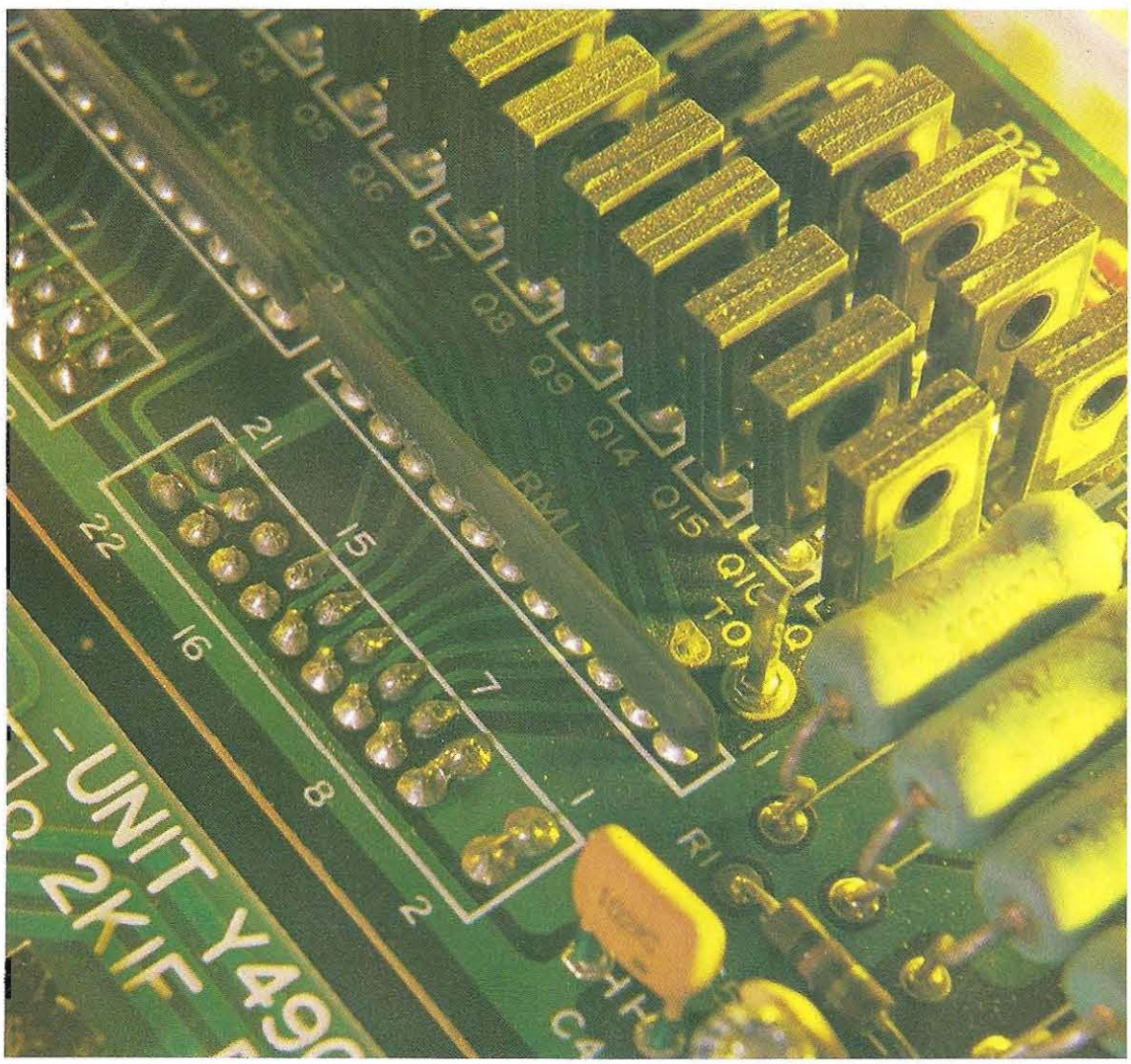
La forme de codage la plus simple est la transformation d'un nombre décimal en son équivalent binaire, c'est-à-dire l'opération qui fournit la représentation d'un nombre sous la forme de symboles 0 et 1.

Pour les lettres de l'alphabet, il n'existe pas de codage mathématique ; la méthode consiste à établir une table de traduction, arbitraire dans sa conception, et qui fournit pour chaque lettre de l'alphabet (ou autre symbole) une représentation sous la forme d'une série de 0 et de 1.

Cette représentation s'appelle **code**. Chaque lettre de l'alphabet a son propre code, sous lequel l'ordinateur la reconnaît.

A l'inverse, quand l'ordinateur fournit des données, il faut disposer de circuits capables de

Détail des circuits de commande d'une imprimante. En haut et à droite, on aperçoit les transistors de puissance qui pilotent la tête d'impression.



Centre THC

traduire les informations binaires en symboles graphiques compréhensibles par l'utilisateur.
L'hypothèse d'un codage arbitraire n'est pas admissible car si les tables étaient différentes, les ordinateurs ne pourraient échanger des données que par l'intermédiaire d'opérations compliquées de conversion des codes.
En outre, un code arbitraire n'offre aucune possibilité de vérifier si la donnée a été transmise correctement ; elle ne permet donc pas de détecter d'éventuelles erreurs.
Telles sont précisément les raisons qui ont présidé à la création d'un code standard adopté par une majorité de constructeurs, le code **ASCII** (American Standard Computer Information Interchange).
Ce code utilise 7 bits pour traduire un symbole, d'où la possibilité d'utiliser 128 codes différents (de 0 à 1111111 = 127 décimal).
Le code ASCII est divisé en trois groupes, ayant chacun des caractéristiques et des fonctions bien déterminées :

- **codes transparents**
- **symboles et nombres**
- **lettres**

Codes transparents

Ce terme recouvre tous les codes qui n'ont pas de correspondant alphabétique et qui sont utilisés pour transmettre des commandes.

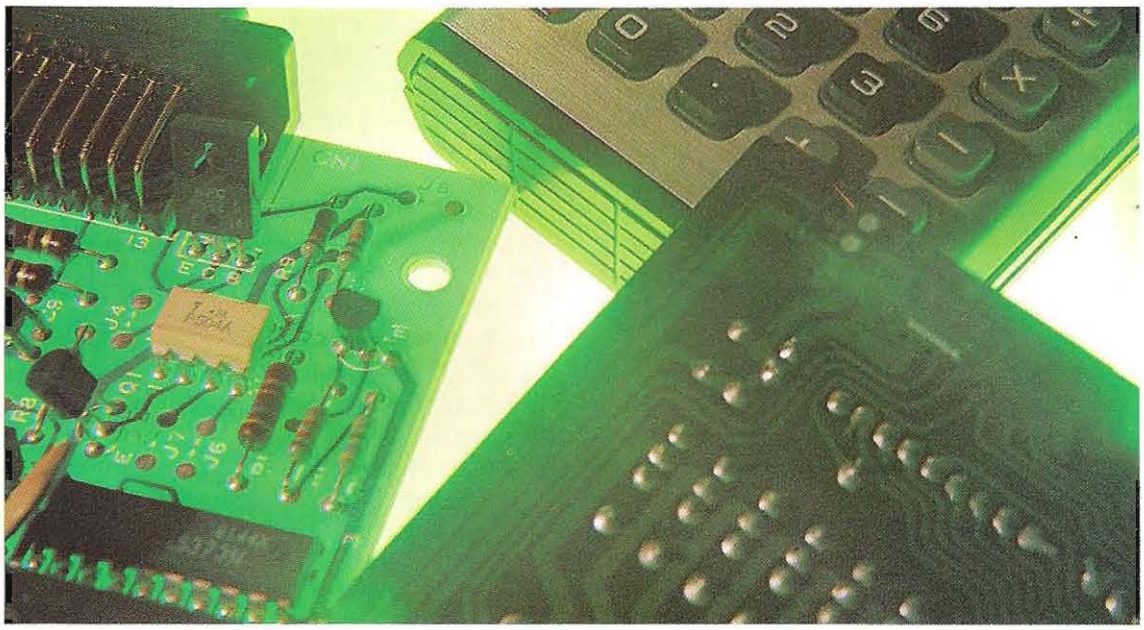
Au cours de l'échange de données entre l'ordinateur et les périphériques, il est indispensable de transmettre, en dehors des données proprement dites, une série d'instructions assurant le fonctionnement de ces périphériques.
Par exemple, lorsqu'il pilote une imprimante, l'ordinateur doit envoyer les commandes de changement de lignes et d'avancement du papier ; la suite des instructions d'impression est donc la suivante :

1 - Positionnement de la tête d'impression au début de la première feuille.
2 - Envoi et impression de la première ligne de caractères.
3 - Ordre donné à la tête d'impression de retourner à la marge gauche de la feuille.
4 - Ordre d'avancement du papier.
5 - Envoi et impression de la deuxième ligne de caractères.
6 - Reprise du cycle à partir du point 3, et répétition pour toutes les lignes à imprimer.

Les codes transmis aux points 1, 3 et 4 sont des codes transparents car ils ont pour fonction la commande de l'imprimante ; ils sont envoyés aux circuits électroniques qui assurent la commande. Les codes transparents sont au nombre de 31 et ils occupent les premières positions, c'est-à-dire que leurs valeurs sont comprises entre 1 et 31 décimal (1 et 1F hexadécimal).
Nombre de ces codes transparents sont utili-

PRINCIPAUX CODES TRANSPARENTS

	Sigle	DEC code décimal	HEX code hexadécimal
1. Sonnerie (Bell) Caractère d'appel dans le terminal	BEL	7	7
2. Retour arrière (Backspace) Caractère d'espacement arrière	BS	8	8
3. Retour de chariot (Carriage Return) Caractère de passage à la ligne	CR	13	D
4. Echappement (Escape) Caractère de changement de code	ESC	27	1B
5. Présentation de formule (Form Feed) Caractère d'alimentation en papier	FF	12	C
6. Tabulation horizontale (Horiz. Tab.)	HT	9	9
7. Interligne (Line feed) Caractère de changement de ligne	LF	10	A
8. Tabulation verticale (Vert. Tab.)	VT	11	B



Centre THC

Une calculatrice démontée et ses circuits internes.

sés pour des applications particulières, comme l'échange de données entre ordinateurs, et non dans les programmes d'application.
On trouvera page 111 la liste des principaux codes utilisés dans les applications.
Pour plus de clarté, le tableau donne la traduction française des sigles qui proviennent de l'anglais.

1 - Bell (Décimal = 7, HEX = 7). Beaucoup de terminaux disposent d'un signal d'avertissement acoustique pour appeler l'attention de l'opérateur. C'est le code 7 qui active la sonnerie.

2 - Backspace (Décimal = 8, HEX = 8). C'est la commande de recul d'un caractère. On l'utilise pour corriger des erreurs : la commande se fait par le code 8, et l'on tape la valeur correcte à la place de l'erreur. Cela ne peut évidemment fonctionner qu'avec l'écran, puisque le nouveau caractère vient effacer le précédent.

3 - Carriage Return (Décimal 13, HEX = D). Commande de retour du chariot d'impression en début de ligne. Avec l'imprimante, ce code actionne véritablement le chariot, alors que sur l'écran, c'est simplement le curseur électronique qui se déplace.

4 - Escape (Décimal 27, HEX = 1B). Ce code n'a pas d'effet visible immédiat. Il sert à informer le périphérique (imprimante ou écran) que les caractères venant à la suite sont encore des codes de commande. On en trouve une application courante sur les écrans de visualisation graphique. Le terminal, s'il n'est pas informé que les nombres envoyés indiquent les positions du curseur, les interprète comme des caractères à inscrire, avec un tout autre résultat. C'est le code ESCAPE qui signale la nature particulière de ces nombres. L'instruction prend la forme suivante : code ESCAPE + nombre.
Si l'on entre le nombre sans le faire précéder du code ESCAPE, on obtient simplement l'écriture de symboles.

5 - Form Feed (Décimal 12, HEX = C). Il commande le passage d'une page à la suivante dans l'imprimante. Il est bon de l'utiliser avant et après chaque phase d'impression, pour que les listings soient séparés par du blanc.

6 - Horizontal Tabulation (Décimal 9, HEX = 9). Qu'il s'agisse d'une imprimante ou d'un écran, le nombre maximum de caractères pouvant être inscrit sur une ligne est généralement constant, chaque caractère ou espace occupant une position bien définie (toutefois la densité de l'impression peut être modifiée également par un code). La commande HT permet de sauter un certain nombre de ces positions par

une seule instruction. Ce code est utilisé par exemple lorsque l'on ne veut pas écrire trop près de la marge gauche : le code 9 déplace le chariot d'impression (ou le curseur) jusqu'à la première tabulation horizontale ; deux codes 9 successifs permettent de sauter à la deuxième tabulation, etc.

7 - Line Feed (Décimal 10, HEX = A). Permet de sauter d'une ligne à la suivante.

8 - Vertical Tabulation (Décimal 11, HEX = B). Le fonctionnement est le même que pour le code HT, si ce n'est que le passage d'une tabulation à l'autre se fait dans le sens vertical. Les codes HT (tabulation horizontale) et VT (tabulation verticale) peuvent être programmés sur certains périphériques, c'est-à-dire que le nombre de caractères d'une HT et le nombre de lignes d'une VT peuvent être modifiés par l'entrée de codes ultérieurs (voir tableau).

TABLEAU DES CODES ASCII TRANSPARENTS

Décimal	Hexadécimal	Sigle	Fonction dans les programmes d'application
0	0	NUL	N'actionne aucune commande
1	1	SOH	
2	2	STX	
3	3	ETX	
4	4	EOT	
5	5	ENQ	
6	6	ACK	
7	7	BEL	Active le signal acoustique
8	8	BS	Fait reculer le curseur d'un espace
9	9	HT	Tabulation horizontale
10	A	LF	Avance d'une ligne
11	B	VT	Tabulation verticale
12	C	FF	Change de page
13	D	CR	Renvoie le chariot
14	E	SO	
15	F	SI	
16	10	DLE	
17	11	DC1	
18	12	DC2	
19	13	DC3	
20	14	DC4	
21	15	NAK	
22	16	SYN	
23	17	ETB	
24	18	CAN	
25	19	EM	
26	1A	SUB	
27	1B	ESC	Annonce une séquence de contrôle
28	1C	FS	
29	1D	GS	
30	1E	RS	
31	1F	US	

Symboles et nombres

Ce groupe de codes va de 32 décimal (20 HEX), à 64 décimal (40 HEX) et comprend des symboles et des nombres. Le tableau (ci-dessous) donne également les nombres décimaux, considérant les nombres comme des symboles graphiques. On peut avoir deux formes de représentation:

- **valeur numérique**
- **symbole ASCII**

La représentation numérique est utilisée pour effectuer les calculs, la représentation ASCII pour les opérations d'impression. Par exemple, le nombre décimal 5 peut être représenté selon les deux modes: numérique (binaire) = 1 0 1, ASCII: 35 HEX = 1 1 0 1 0 1. En utilisant le symbole 5 comme un nombre (pour les calculs), son code est 101, alors que dans les opérations d'impression (en ASCII) il est 110101.

Lettres

Le dernier groupe de codes comprend les lettres majuscules et minuscules, et un nombre limité de symboles, comme on peut le voir dans les tableaux ci-dessous et à droite.

TABLEAU ASCII DES SYMBOLES ET DES NOMBRES

Décimal	Hexa-décimal	Symbole ou nombre représenté	
32	20	Espace	Symboles 1re partie
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	
45	2D	—	
46	2E	.	
47	2F	/	
48	30	0	Nombres
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	Symboles 2e partie
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	

TABLEAU ASCII POUR LES LETTRES

Décimal	Hexadécimal	Symbole
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	—
96	60	`
97	61	a
98	62	b
99	63	c

Décimal	Hexadécimal	Symbole
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Applications

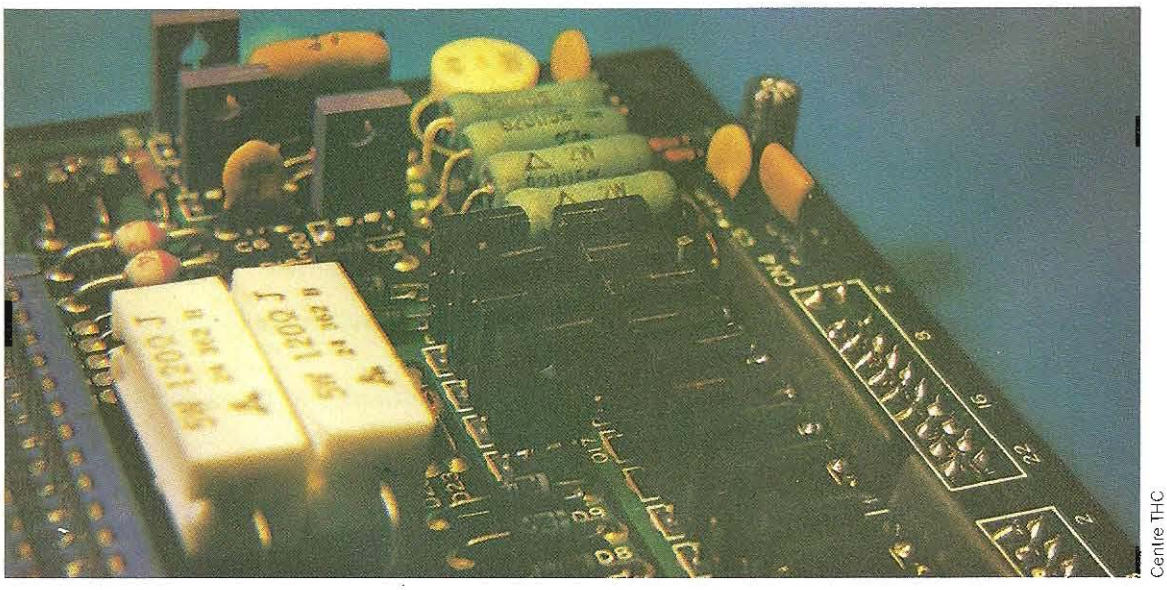
Voici la suite de codes qu'il faut envoyer à l'imprimante pour écrire : « Ceci est un exemple ». La procédure se compose des opérations :

1 - Envoi de la commande de mise en place du chariot en début de page (FF).
2 - Envoi des codes correspondant à la phrase à écrire.
3 - Envoi d'une nouvelle commande de changement de page (FF).

Le code correspondant aux étapes 1 et 3 est FF = 12 décimal = C (HEX). Pour le codage de la phrase, il faut procéder lettre par lettre, en se référant aux tables de conversion présentées page 114 et ci-contre.

Lettre	Code HEX
C	43
E	45
C	43
I	49
espace =	20
E	45
S	53
T	54
espace =	20
U	55
N	4E
espace =	20
E	45
X	58
E	45
M	4D
P	50
L	4C
E	45

Circuits de commande d'une imprimante. Au premier plan : deux résistances de puissance, plus grandes que des composants habituels.



Centre THC

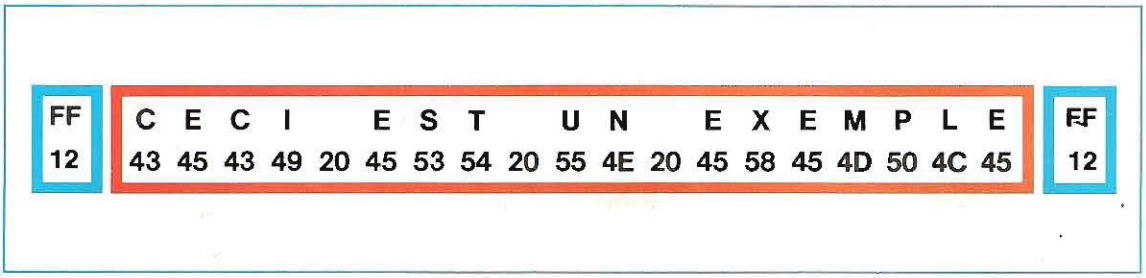


Table ASCII en format compact

On trouvera ci-dessous la table des codes ASCII sous une forme plus pratique et plus compacte, c'est-à-dire organisée de manière à fournir les valeurs binaires de chaque symbole.

TABLE DES CODES ASCII

7 6 5 4 3 2 1	COLONNE / LIGNE	000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC 1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC 2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC 3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC 4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10 (A)	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11 (B)	VT	ESC	+	;	K	[	k	{
1 1 0 0	12 (C)	FF	FS	,	<	L	\	l	|
1 1 0 1	13 (D)	CR	GS	-	=	M	]	m	}
1 1 1 0	14 (E)	SO	RS	.	>	N	^	n	~
1 1 1 1	15 (F)	SI	US	/	?	O	_	o	DEL

Par exemple, le symbole F occupe la ligne 6 et la colonne 4.
Le codage binaire de ce symbole s'obtient en réunissant les valeurs des bits de la ligne avec ceux de la colonne.

La ligne 6 a les valeurs 0 1 1 0 **correspondant aux positions 4, 3, 2, 1**

La colonne 4 a les valeurs 1 0 0 **correspondant aux positions 7, 6, 5**

Réunion des deux nombres : F = 1 0 0 0 1 1 0 **= 70 décimal = 46 hexadécimal**

4 6

Ce résultat peut être obtenu très simplement en écrivant à la suite la colonne et la ligne.

F = colonne **4**, ligne **6** = **46** hexadécimal.

De même pour le symbole **m** qui se trouve dans la colonne 6 et la ligne 13 (= D) on a :

m = colonne **6**, ligne **13** (= D) = **6D** hexadécimal.

Transmission des codes ASCII

La transmission des données entre l'ordinateur et les périphériques peut se faire de deux manières :

- **par transmission en série**
- **par transmission en parallèle**

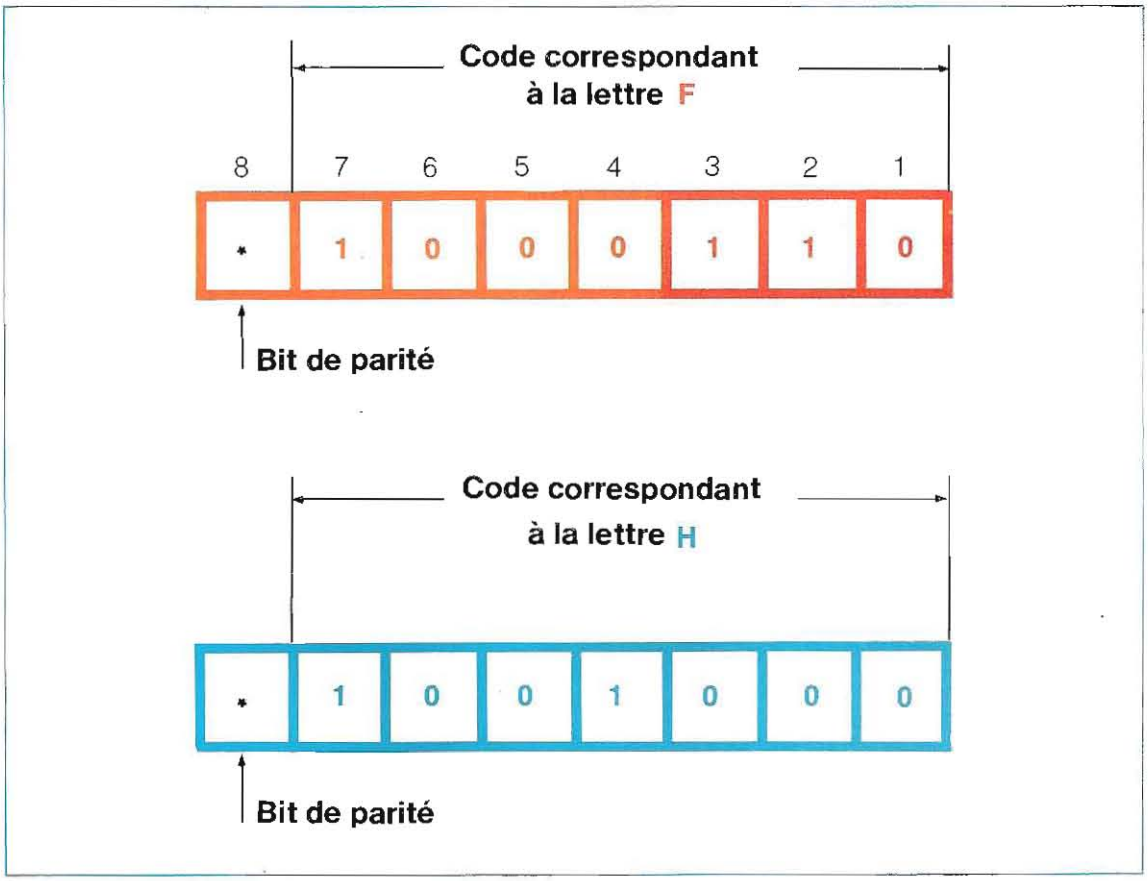
La transmission en série consiste à envoyer un seul bit à la fois, l'un derrière l'autre. Au contraire, en mode parallèle, un certain nombre de bits sont transmis simultanément. La transmission en série n'exige qu'une seule liaison électrique puisque les signaux (bits) voyagent l'un derrière l'autre, alors que la transmission en parallèle nécessite autant de liaisons électriques que de signaux à envoyer simultanément. Les schémas de la page 118 illustrent ces deux modes de transmission.

La transmission en parallèle permet une plus grande vitesse mais elle est plus coûteuse et presque impossible à réaliser sur de longues distances en raison du nombre très élevé des liaisons électriques ; la transmission en série autorise au contraire des vitesses plus limitées mais elle est plus économique et donc plus largement utilisée.

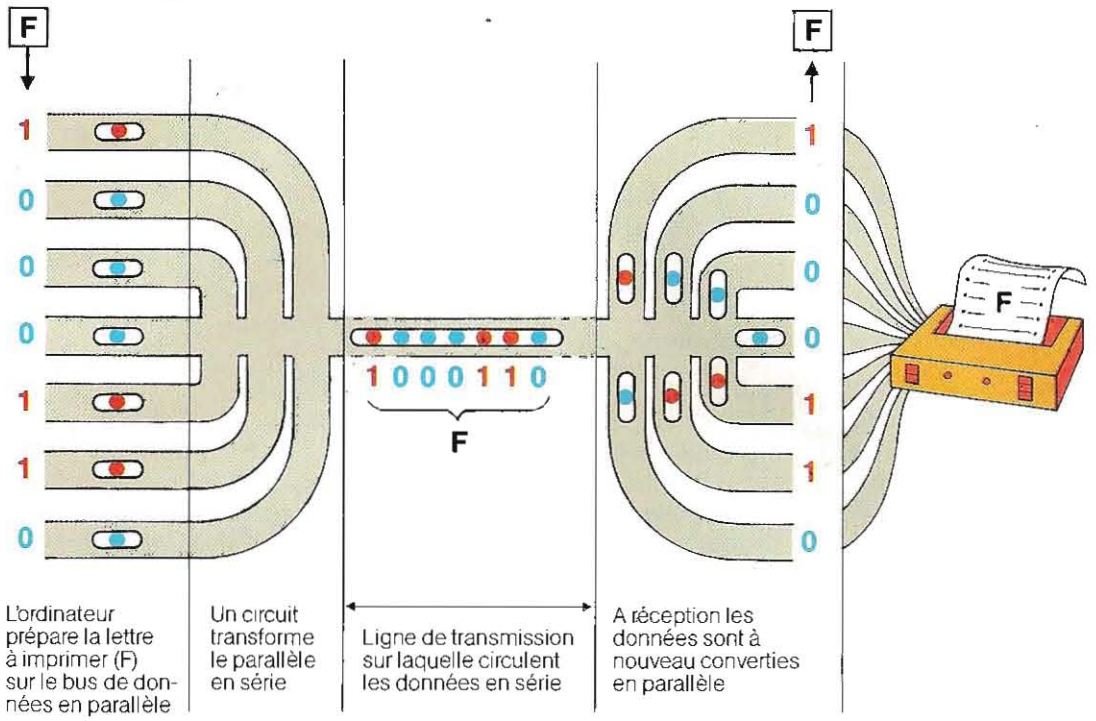
A partir de maintenant, nous travaillerons toujours dans l'optique de la transmission en série. Cette liaison prévoit pour le message envoyé deux formes différentes :

- **mode asynchrone**
- **mode synchrone**

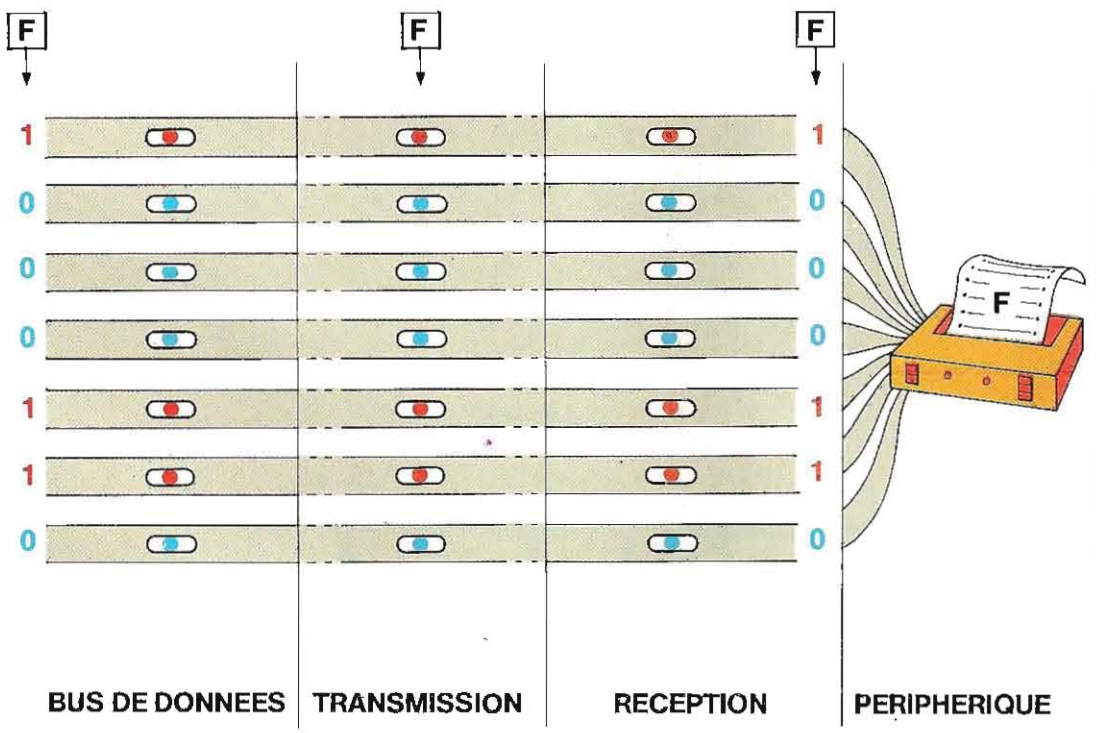
En mode asynchrone, l'espace (en termes de temps) entre un caractère à envoyer et le suivant est arbitraire, alors que la distance entre les bits de chaque caractère est fixe. Le schéma ci-dessous montre la transmission des caractères **F** et **H** en mode asynchrone.

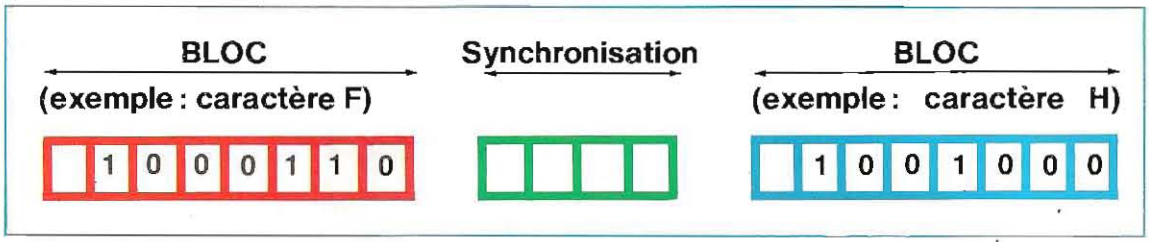


SCHEMA LOGIQUE DE LA TRANSMISSION EN SERIE

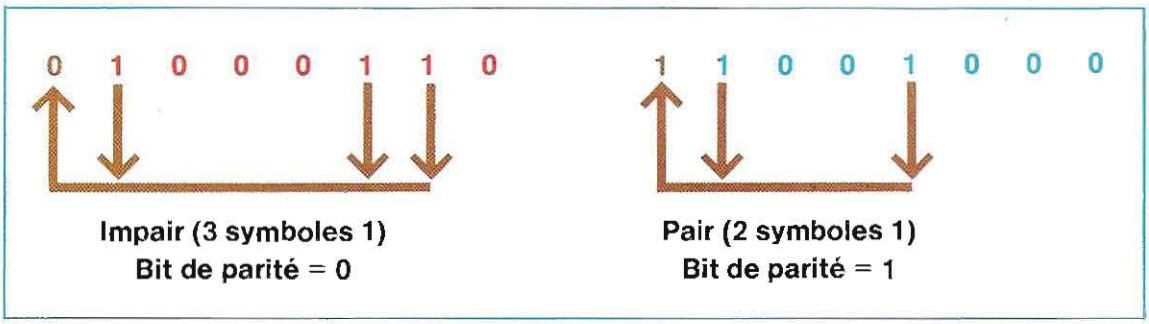


SCHEMA LOGIQUE DE LA TRANSMISSION EN PARALLELE





En mode synchrone, les données doivent être transmises par groupes que sépare un intervalle de temps déterminé et qui prennent le nom de **bloc**. Chaque bloc est séparé du suivant par un code particulier assurant la synchronisation du système.
Le schéma ci-dessus montre la structure en mode synchrone pour la transmission des deux mêmes lettres **F** et **H**.
Le schéma de la page 117 donne, en plus des 7 bits normaux du code ASCII, un bit signalé par le symbole *. Ce bit de parité est un élément de contrôle de la fiabilité du message reçu. Ce contrôle se fait en prenant le bit de parité (bit 8) égal à 1 si les symboles 1 sont présents en nombre pair dans les autres bits. Si la quantité de symboles 1 contenus dans le caractère à envoyer est un nombre impair, le bit de parité est égal à 0. Dans le schéma de la page 117, le caractère **F** (1000110) contient 3 symboles 1, son bit de parité est donc égal à 0; en revanche le caractère **H** (1001000) contient 2 symboles 1, son bit de parité prend la valeur 1. La forme du message devient donc:



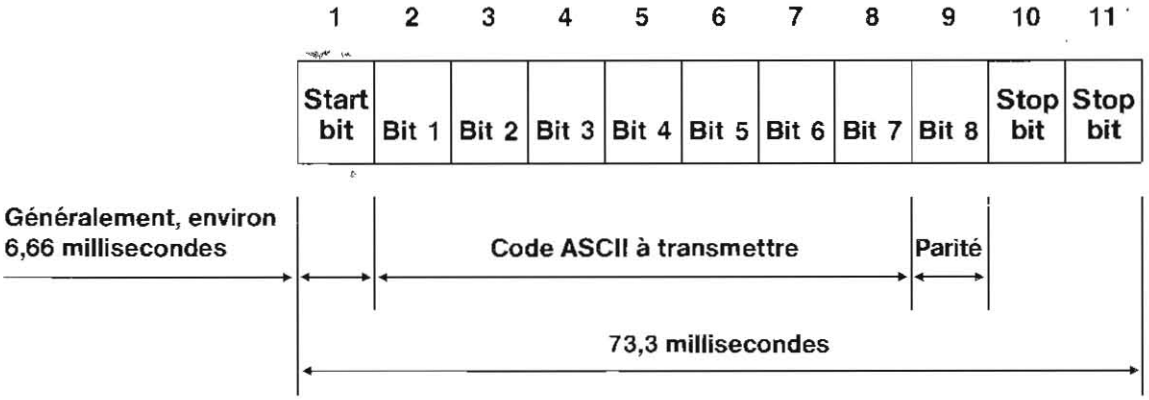
Il arrive aussi que dans les applications on rencontre la logique inverse, c'est-à-dire :
bit de parité = 1 dans le cas impair,
bit de parité = 0 dans le cas pair.
En dehors du bit de parité, en mode de transmission asynchrone, il faut envoyer un signal de reconnaissance de début de caractère et un signal de fin de caractère.
Le signal de début de caractère prend le nom de bit de départ (START). Le signal de fin de caractère s'appelle bit d'arrêt (STOP). Un message se conclut normalement par deux bits d'arrêt.
Le format complet est donc le suivant :

- bit de START (départ)
- 7 bits de la donnée
- bit de parité
- premier bit de STOP (arrêt)
- deuxième bit de STOP (arrêt).

La vitesse de transmission s'exprime en nombre de modulations par seconde, ou d'unités d'information par seconde. On parle plus volontiers

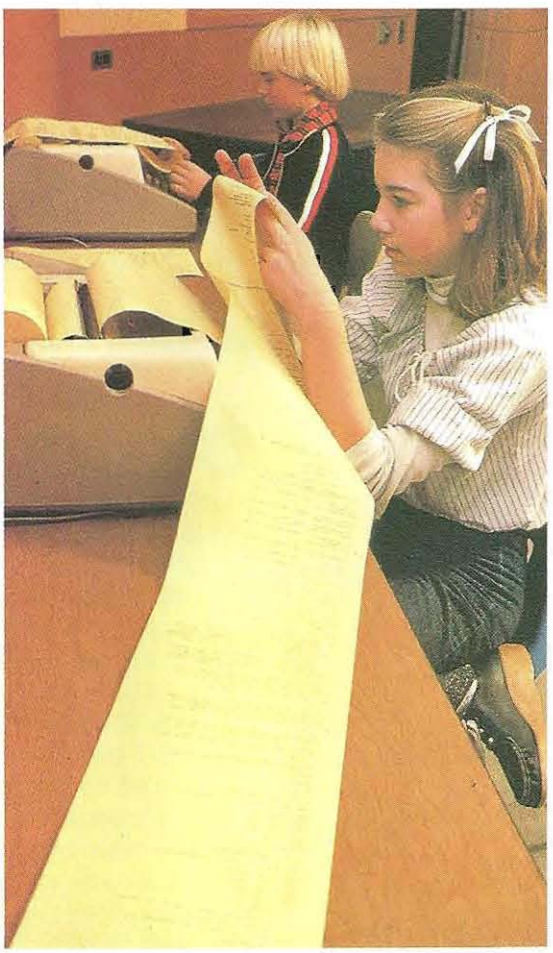
de bits par seconde (bps) : l'unité en est le **baud**. Le schéma ci-dessous représente un message dont la vitesse est 150 bauds. Dans ces transmissions, le bit de départ vaut 0 et les bits d'arrêt valent 1. L'état 1 prend le nom de MARQUE (MARK), l'état 0 celui de ESPACE (SPACE).

STRUCTURE DES SIGNAUX EN TRANSMISSION A 150 BAUDS



- Les données (symboles, lettres et nombres) sont codées selon le code ASCII.
- Le code ASCII est utilisé dans les transmissions vers les périphériques (écrans, imprimantes) ou entre ordinateurs.
- Pour la gestion des périphériques, il existe un certain nombre de codes particuliers que l'on appelle CODES TRANSPARENTS.
- La transmission des données peut se faire selon deux modes : transmission EN SERIE et transmission EN PARALLELE.
- La transmission EN SERIE est la plus utilisée dans les échanges entre ordinateurs ; elle est ASYNCHRONE ou SYNCHRONE.
- En transmission ASYNCHRONE, en dehors des 7 bits constituant la donnée (le caractère), il faut transmettre :
  Un premier bit de START (DEPART), qui vaut normalement 0 ;
  le bit de PARITE, après les 7 bits de la donnée ;
  deux bits de STOP (d'ARRET), qui valent normalement 1.

L'ordinateur à l'école : jeunes élèves aux prises avec des imprimantes en fonctionnement.



Grazia Neri

Contrôle de réception

Dans la transmission d'un caractère en mode asynchrone, le bit de parité permet de contrôler la validité de la donnée. Il faut donc pouvoir vérifier à la réception la cohérence entre le nombre de symboles 1 présents dans la donnée et la valeur du bit de parité.

A l'arrivée du message, le circuit récepteur extrait le premier bit (Départ, Start) et les deux derniers (Arrêt, Stop ou Fin), et présente donc au périphérique (écran ou imprimante) un paquet de 8 bits.

Les 7 premiers bits constituent la donnée proprement dite, et le dernier le bit de parité.

Pour contrôler la validité de la donnée reçue :

1 / on isole le 8e bit (bit de parité) ;

2 / on compte le nombre de symboles 1 présents parmi les autres bits ;

3 / si le total « donnée » est pair et si le bit 8 vaut 1, ou si le total « donnée » est impair et si le bit 8 vaut 0, la donnée est valide.

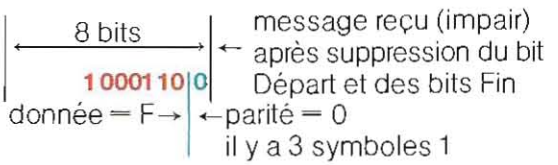
Pour notre exemple, nous avons choisi arbitrairement un message impair : l'inverse est parfaitement concevable.

Pour isoler le 8e bit des 7 autres (qui représentent la donnée), on peut adopter l'opérateur ET :

pour isoler la parité :
message à l'arrivée ET 00000001 ;
pour isoler la donnée :
message à l'arrivée ET 11111110.

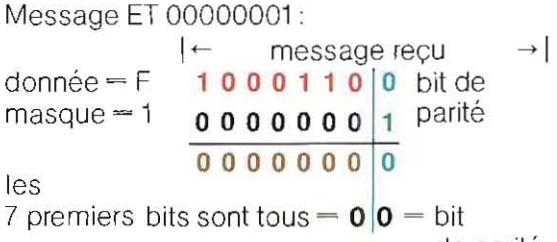
En fait, en utilisant l'opérateur ET entre un nombre binaire quelconque (message reçu) et la valeur 1, on obtient le dernier bit du message reçu, c'est-à-dire le bit de parité.

Par exemple, dans le cas de la lettre **F** = 10001100 :

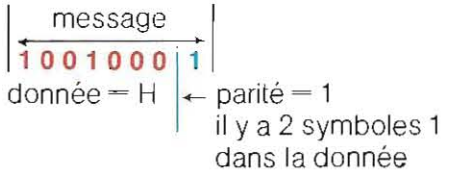


La vérification du bit de parité se fait avec l'opérateur ET.

Message ET 00000001 :

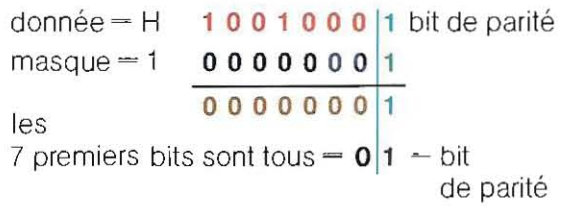


Mais dans le cas de la lettre **H** = 10010001 :



Dans ce cas également, pour extraire le bit de parité, il suffit d'appliquer l'opérateur ET et le masque 00000001.

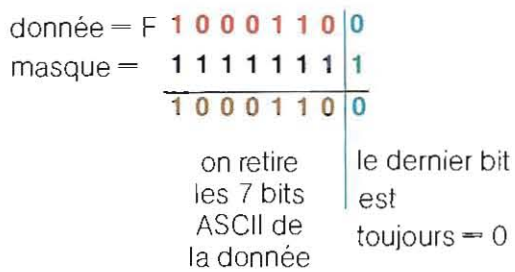
Message ET 00000001 :



La valeur constante avec laquelle on effectue l'opération ET (1 dans les exemples ci-dessus) pour isoler une partie d'une donnée prend le nom de **masque**, car elle permet d'extraire la partie intéressante en la lisant comme s'il y avait un filtre sur la donnée.

Pour extraire du message reçu (8 bits) les 7 bits qui constituent la lettre transmise, il faut appliquer le masque = 11111110.

Message ET 11111110



Les exemples ci-dessus montrent comment on isole les données et les bits de parité à l'aide de programmes conçus à cet effet. Les circuits récepteurs modernes possèdent cette fonction au niveau du matériel, c'est-à-dire sans apport d'aucun programme.

Codes de sécurité

Les listings de la page 78 donnent les résultats obtenus en appliquant les différents opérateurs logiques entre le nombre fixe 11 (masque) et le nombre J variable de 65 à 90. Ces valeurs sont les codes ASCII correspondant aux lettres en majuscule (A = 65 ; Z = 90).

La colonne OUX fournit un excellent instrument de codification car elle associe à chaque lettre une autre valeur différente et qu'on ne peut déduire sans connaître la valeur du masque.
Sur la première ligne du listing 3 page 78, on lit :

11 OUX 65 = 74

La lettre A (65) devient J (74).
En changeant de masque (listing 4 page 78)

32 OUX 65 = 97

La lettre A devient a (97).

Cette méthode permet une certaine sécurité dans la transmission des données. Il existe cependant des méthodes pour forcer les systèmes de sécurité à base de codification ; elles consistent à exploiter la vitesse de calcul considérable de la machine pour effectuer un grand nombre de tentatives, jusqu'à découvrir par approximations successives la clé de code utilisée pour le chiffrement.

Solutions du test 2

1 / Le test propose le calcul du résultat à l'aide de l'opérateur $\overline{\text{A OUX B}}$, c'est-à-dire NON (A OUX B).
Pour faciliter la recherche, il faut d'abord calculer A OUX B, puis sa négation.

A	B	A OUX B	$\overline{\text{A OUX B}}$	A ET ($\overline{\text{A OUX B}}$)
1	**0**	**1**	**0**	**0**
1	**1**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**

On notera que A ET ($\overline{\text{A OUX B}}$) est identique à A ET B.

2 / a) 11011 ET 011 = 00011

b) Avant de résoudre l'opération logique il faut convertir les deux nombres décimaux 12 et 7 en binaire :

12 décimal = 1100 7 décimal = 0111

12 ET 7 = 1100 ET 0111 = 0100

c'est-à-dire, en reconvertissant tout en décimal (0100 = 4)

12 ET 7 = 4 (en décimal)

c) Les nombres 12 et 7 octaux possèdent les représentations binaires suivantes :

12 octal = 001010 7 octal = 000111

donc :

12 ET 7 = 0001010 ET 000111 = 000010

d'où :

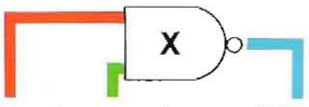
12 ET 7 = 2 (en octal)

d) 1101 OU 0110 = 1111

e) 1011 OUX 1101 = 0110

3 / Pour trouver la solution, se reporter au texte relatif aux opérateurs logiques : NON, ET, OU, OUX, pages 80 à 83 et pages 88 à 91.

4 / a) Dans les circuits composites, il faut toujours partir du premier, résoudre sa table de vérité puis l'appliquer au suivant. Cette méthode donne pour le circuit X :



A	B	$\overline{\text{A ET B}}$
1	1	0
1	0	1
0	1	1
0	0	1

La colonnne $\overline{\text{A ET B}}$ (0111) devient l'une des deux entrées du circuit suivant (Y). La seconde entrée de ce circuit est fixe, et vaut 1. De ce fait :



A	B	$\overline{\text{A ET B}}$
0	1	1
1	1	0
1	1	0
1	1	0

b)



A	F	A NON ET F	B	$\overline{\text{B OU (A NON ET F)}}$
1	1	0	0	1
0	1	1	1	0
		0	1	0
		1	0	0

On a répété dans la seconde partie du tableau les états 0, 1 de A NON ET F pour obtenir toutes les combinaisons possibles avec la colonne B.

TEST 3

1 / Ecrire les chiffres 7, 9, 0, 3 en binaire, en nombres et en code ASCII.

2 / Enumérer les principales différences entre la transmission en série et la transmission en parallèle.

3 / Construire le message complet qui doit être envoyé pour transmettre le caractère K en mode asynchrone.

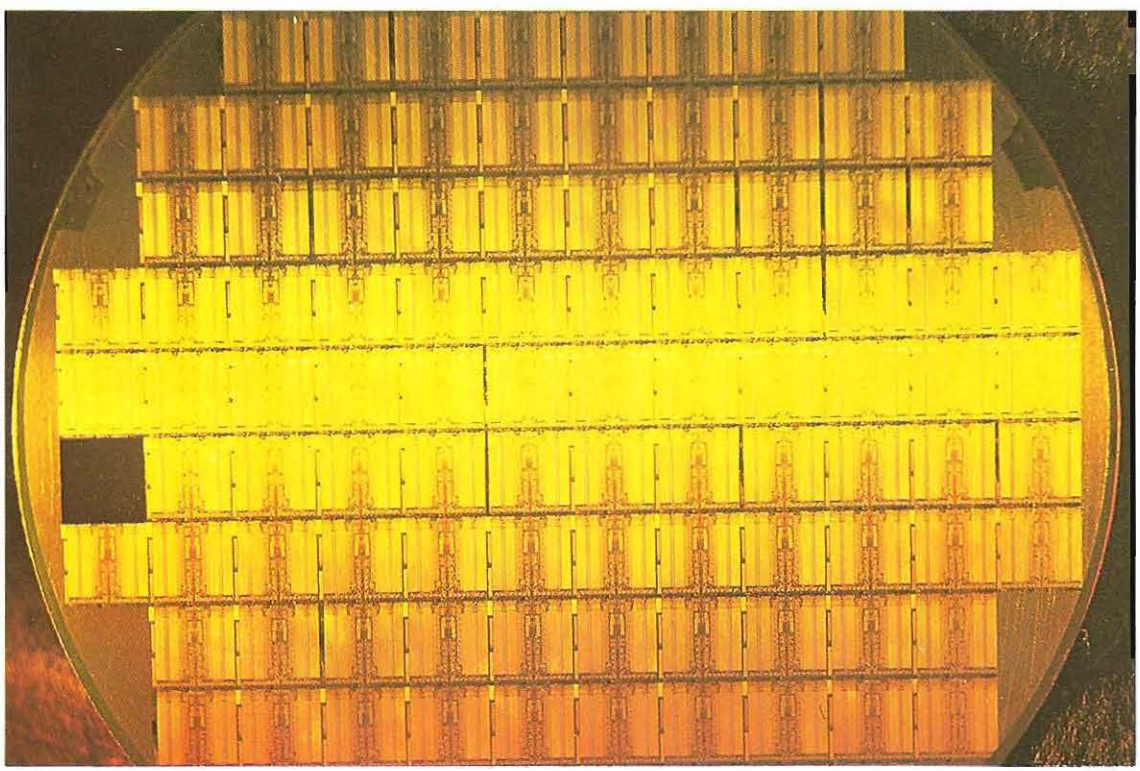
4 / Traduire en code décimal les opérations suivantes réalisées sur imprimante :

a) présentation de formule (saut de page)
b) écriture des caractères X et Y
c) interligne
d) écriture des mêmes caractères en minuscules
e) saut de page

5 / En étudiant la table des codes ASCII, dire quelle est l'opération qu'il faut accomplir pour transformer une lettre majuscule en minuscule.

6 / Ecrire le masque qui extrait d'une donnée quelconque les bits 3, 7, 1 (les bits étant numérotés à partir de 0). Contrôler l'exactitude de la réponse en appliquant le masque sur la représentation binaire des nombres décimaux 12, 21, 6, 15.

Les solutions du test se trouvent pages 150 et 151.



Archives Curcio/IBM

Le dispositif numérique à jonction Josephson permet des vitesses de commutation très élevées.

Microsystèmes

Pour bien utiliser un système à microprocesseur (ordinateur personnel ou micro-ordinateur professionnel) il faut connaître son architecture interne.
On peut distinguer trois niveaux d'utilisation du micro-ordinateur.

- **Utilisation de programmes figés.**
- **Utilisation de certains programmes évolutifs et adaptables.**
- **Développement de programmes spécifiques (« à la demande »).**

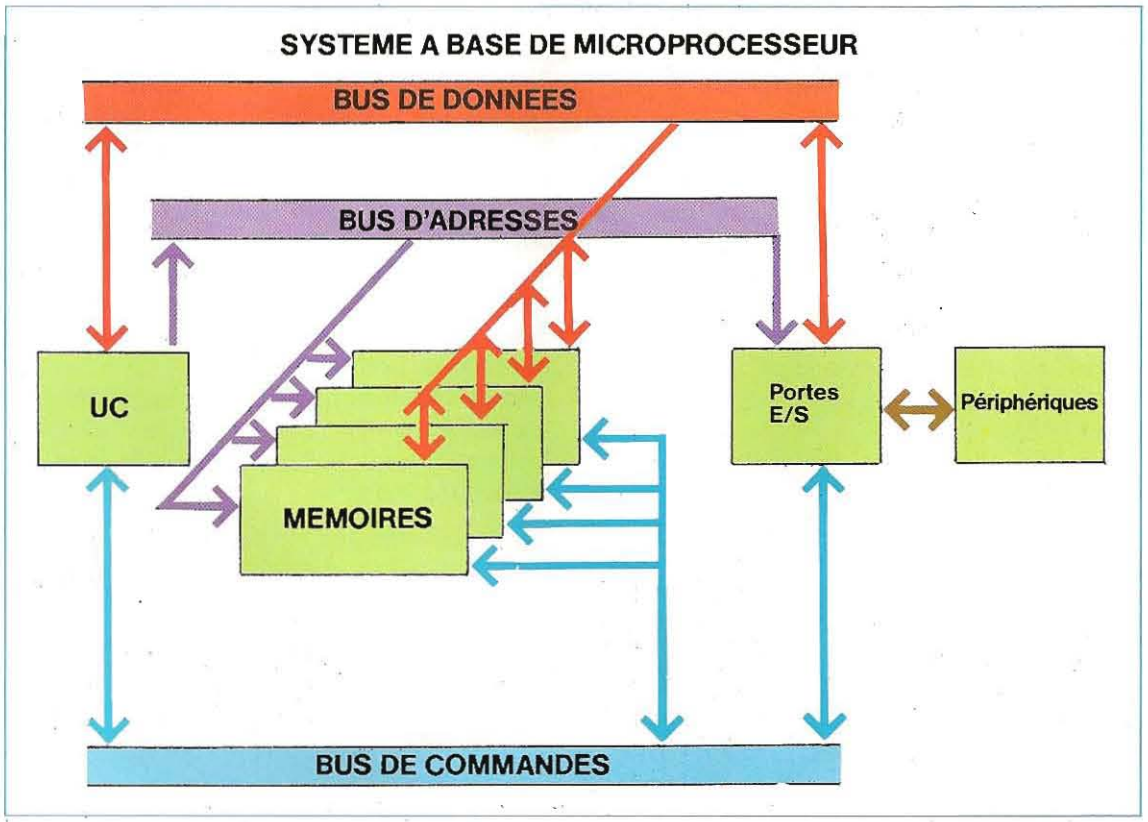
Le degré de connaissance est naturellement fonction du type d'application. Dans tous les cas, un examen, même sommaire, de la structure du système est indispensable.
La structure physique de l'ordinateur est du **matériel** (hardware), tandis que ce qui concerne la programmation prend le nom de **logiciel** (software).

En micro-informatique, c'est la technologie du matériel qui est à l'origine des progrès les plus décisifs.
Cela se traduit par la production de systèmes à la fois plus performants et moins coûteux. C'est même à tel point qu'aujourd'hui certains programmes coûtent plus cher que la machine elle-même.
Les possibilités considérables de tels équipements permettent en outre de vulgariser l'emploi des micro-ordinateurs dans un nombre croissant d'applications nouvelles. Cette prolifération ne peut que se poursuivre.

Les principaux organes

Le schéma ci-dessous montre l'organisation interne d'un système à base de microprocesseur. Les éléments principaux en sont :

- **UC = unité centrale (ou CPU = Central Processing Unit)**
- **MEMOIRES**
- **PORTES ENTREE/SORTIE (E/S)**
- **PERIPHERIQUES**
- **BUS DE DONNEES**
- **BUS D'ADRESSES**
- **BUS DE COMMANDES**



UNITE CENTRALE (UC)
Elle exécute deux fonctions principales: commande de tous les autres composants et exécution des calculs.

MEMOIRES
Dispositif renfermant sous une forme binaire les instructions qui constituent le programme et les données qui seront utilisées dans l'exécution d'un programme.

PORTES E/S
Circuits permettant la communication du système avec le monde extérieur. Le sigle E/S (entrée/sortie) correspond à l'anglais I/O (Input/Output).

PERIPHERIQUES
Tous les dispositifs extérieurs à l'ordinateur et lui servant d'E/S.
Les principaux périphériques sont:
le **visuel** (ou **moniteur**): écran d'affichage permettant à l'opérateur de dialoguer avec la machine;
le **clavier**: très comparable au clavier d'une machine à écrire, c'est l'élément qui permet d'envoyer des caractères et des ordres à la machine (saisie);
l'**imprimante**: organe de sortie sur papier;
l'**unité à disque, disquette,** ou **bande magnétique**: ils constituent la mémoire de masse du système et permettent de stocker d'une manière permanente une grande quantité de données. Ces dispositifs ne sont pas de véritables périphériques mais des extensions de la mémoire centrale de la machine.
des dispositifs spéciaux: dans des applications spécifiques (contrôle de procédés dans l'industrie), ils sont indispensables pour piloter des machines ou en surveiller le fonctionnement.

BUS DE DONNEES
Ensemble de lignes physiques assurant le transfert des données. Les systèmes utilisent un bus de données constitué de circuits parallèles pouvant transporter 8 ou 16 bits (1 ou 2 octets).
Dans l'usage courant, l'octet est souvent appelé « mot » (ou caractère), mais la réciproque n'est pas vraie: dans certains systèmes, le mot comprend 16 bits, c'est-à-dire deux octets, et parfois plus.

BUS D'ADRESSES
Liaison sur laquelle circulent les signaux constituant l'adresse c'est-à-dire la localisation en mémoire des données.

BUS DE COMMANDE
Troisième liaison, inséparable d'un système, par laquelle l'UC envoie ses instructions aux autres composants et en reçoit les signaux en réponse.
Le schéma en haut de la page ci-contre représente de façon détaillée les composants d'un système.

Structure de l'unité centrale

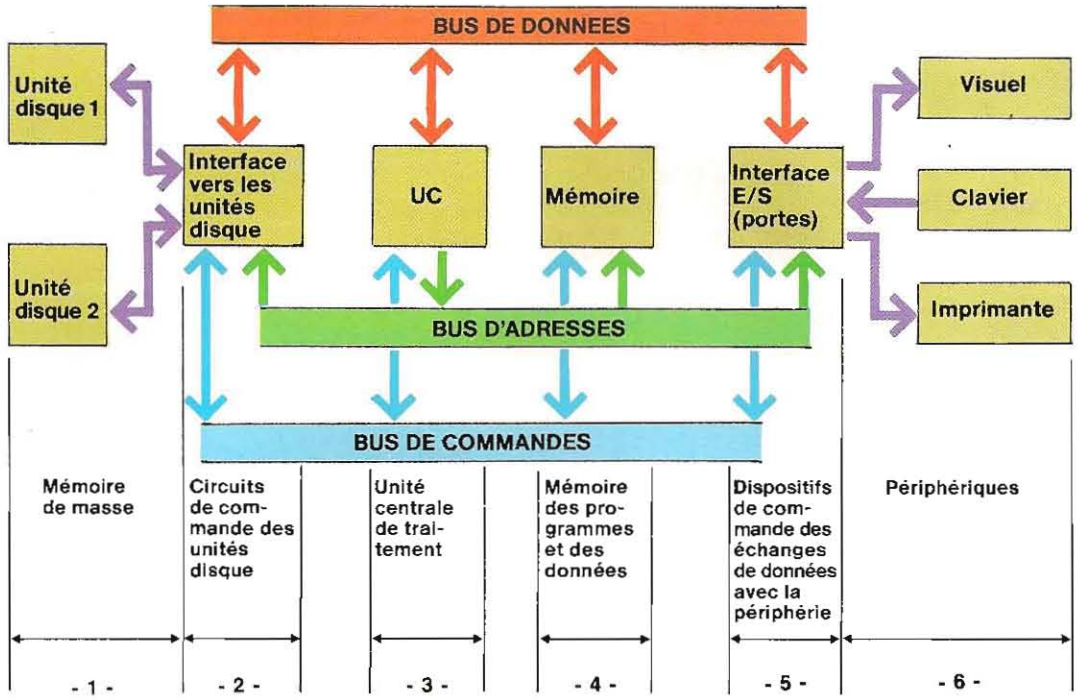
L'unité centrale (UC) est le composant le plus complexe. C'est elle qui génère tous les signaux nécessaires au fonctionnement des autres circuits et qui effectue le traitement des données. Le schéma en bas de la page ci-contre montre que sa structure est comparable à celle du système tout entier. Ses composants principaux sont les suivants.
Le **bus interne**: ses fonctions sont analogues à celles des bus externes (voir schéma p. 127 en haut), et consistent à assurer le transfert de l'information.
Les **accumulateurs** et les **registres**: dispositifs de mémorisation des données avant et après le traitement.
Le **décodeur d'instructions**: circuit traduisant les instructions binaires d'un programme en une série d'impulsions électriques.
Les **circuits de commandes**: pilotés par le circuit précédent, ils servent à générer divers types de signaux pour assurer le fonctionnement des autres composants, à l'intérieur comme à l'extérieur de l'unité centrale (mémoires, portes, etc.).
L'**unité arithmétique et logique** ou **UAL** (en anglais ALU, Arithmetic and Logic Unit). C'est l'organe chargé d'exécuter les calculs et les opérations logiques.

Fonctionnement de l'unité centrale

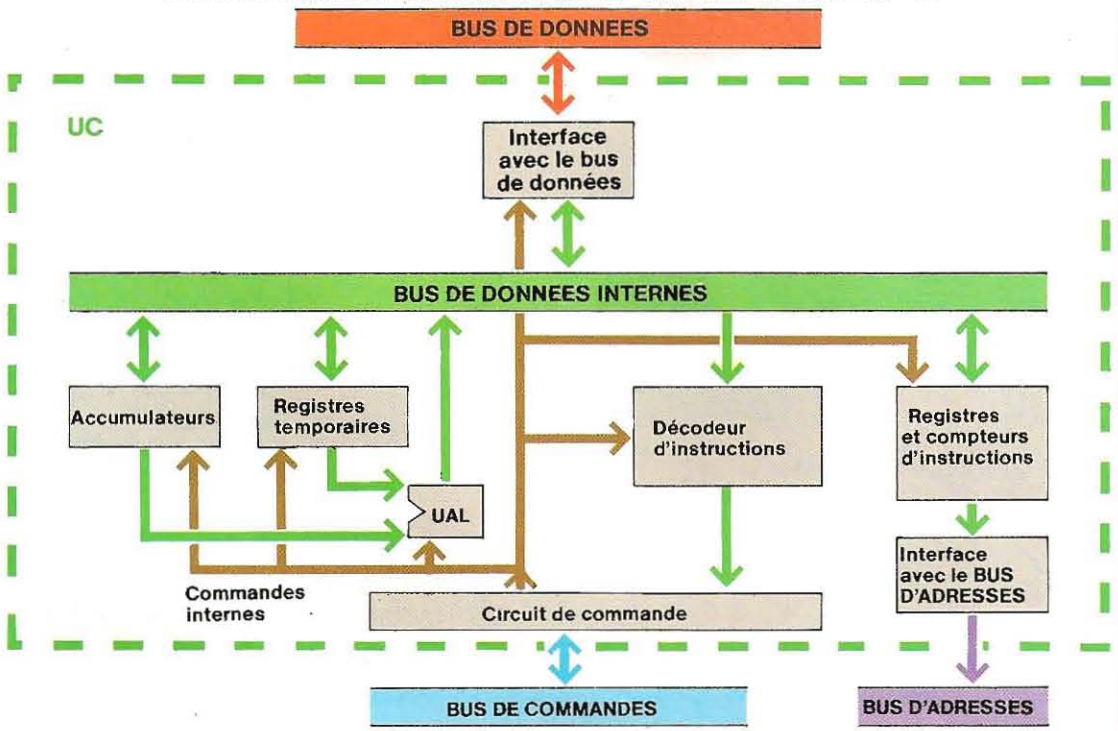
Un programme se compose d'une série d'instructions, divisées au moins en deux parties:

- **le code opération (opcode): indique la nature de l'opération à effectuer**
- **la donnée (ou opérande): quantité sur laquelle l'opération doit être effectuée.**

SYSTEME COMPLET A MICROPROCESSEUR



STRUCTURE CARACTERISTIQUE D'UNE UNITE CENTRALE

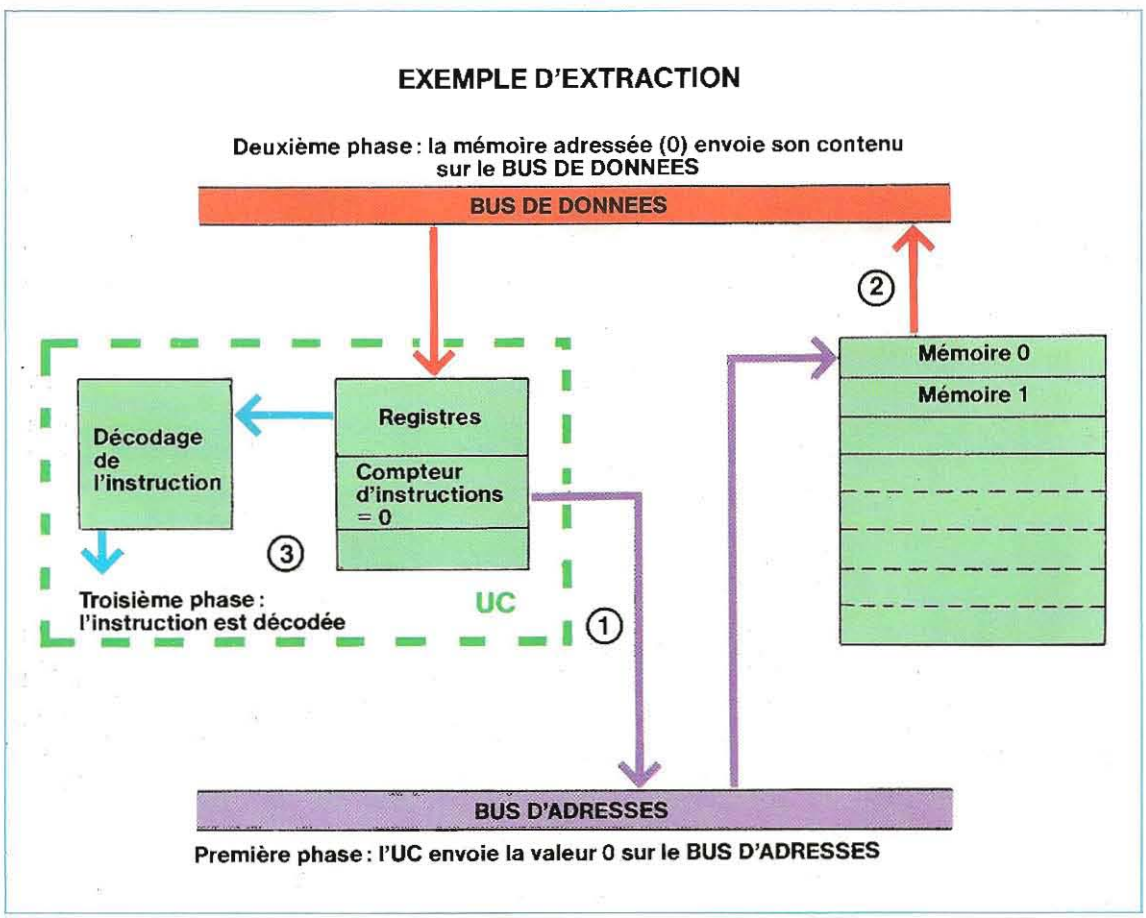


Les codes et les données (ou leurs adresses) sont inscrits dans les mémoires du système. L'ensemble de ces mémoires peut être considéré comme un livre composé de pages portant chacune un certain nombre de caractères. Considérons par exemple une mémoire divisée en pages de 1024 octets (voir schémas p. 125 et 128) ; pour connaître le contenu de la mémoire 3 de la page 1, il faut donner l'adresse 1026 (la première page porte le numéro zéro et contient les mémoires de 0 à 1023 ; la page 1 commence à 1024).
Chaque zone de mémoire (composée d'une ou plusieurs pages) est réservée à un usage précis. Le tableau donnant la topographie des différentes zones et leur emploi s'appelle **image mémoire** ; il varie selon la machine et les programmes qu'elle accepte.
Le « déroulement » d'une instruction de programme se fait selon une séquence d'opérations appelée **cycle** :

- **Extraction de l'instruction (en anglais FETCH)**

- **Décodage et exécution (en anglais EXECUTE)**

C'est la forme la plus simple ; beaucoup d'instructions, plus complexes, exigent plusieurs opérations d'exécution.
Au branchement de la machine, ou lorsqu'on donne l'ordre de commencer un travail, l'unité centrale extrait la première instruction, l'exécute, passe à la seconde, l'exécute, et ainsi de suite pour toutes les instructions.
Il existe à l'intérieur de l'unité centrale une mémoire que l'on appelle un **compteur d'instructions,** ou **compteur ordinal** (en anglais : Program Counter, PC). C'est elle qui stocke l'adresse des instructions successives à exécuter. Si toutes les instructions étaient séquentielles (c'est-à-dire si l'exécution du programme ne comportait pas de saut d'un point à un autre), il suffirait, pour connaître l'adresse des différentes instructions, d'ajouter 1 au contenu du compteur ordinal.
Le schéma ci-dessous illustre un exemple d'extraction (Fetch) d'instruction.



① L'UC envoie le contenu du compteur d'instructions (ici = 0) sur le bus d'adresses.
② La mémoire correspondante (adresse 0) envoie son contenu (instruction à exécuter par l'unité centrale) sur le bus de données.
③ Le contenu du bus de données (dans ce cas, une instruction à exécuter) est chargé dans une mémoire interne de l'unité centrale (registre) ; puis elle passe dans l'unité de décodage où elle est transformée en signaux électriques, par l'intermédiaire de **microprogrammes** qui sont inscrits par le constructeur dans l'unité centrale.

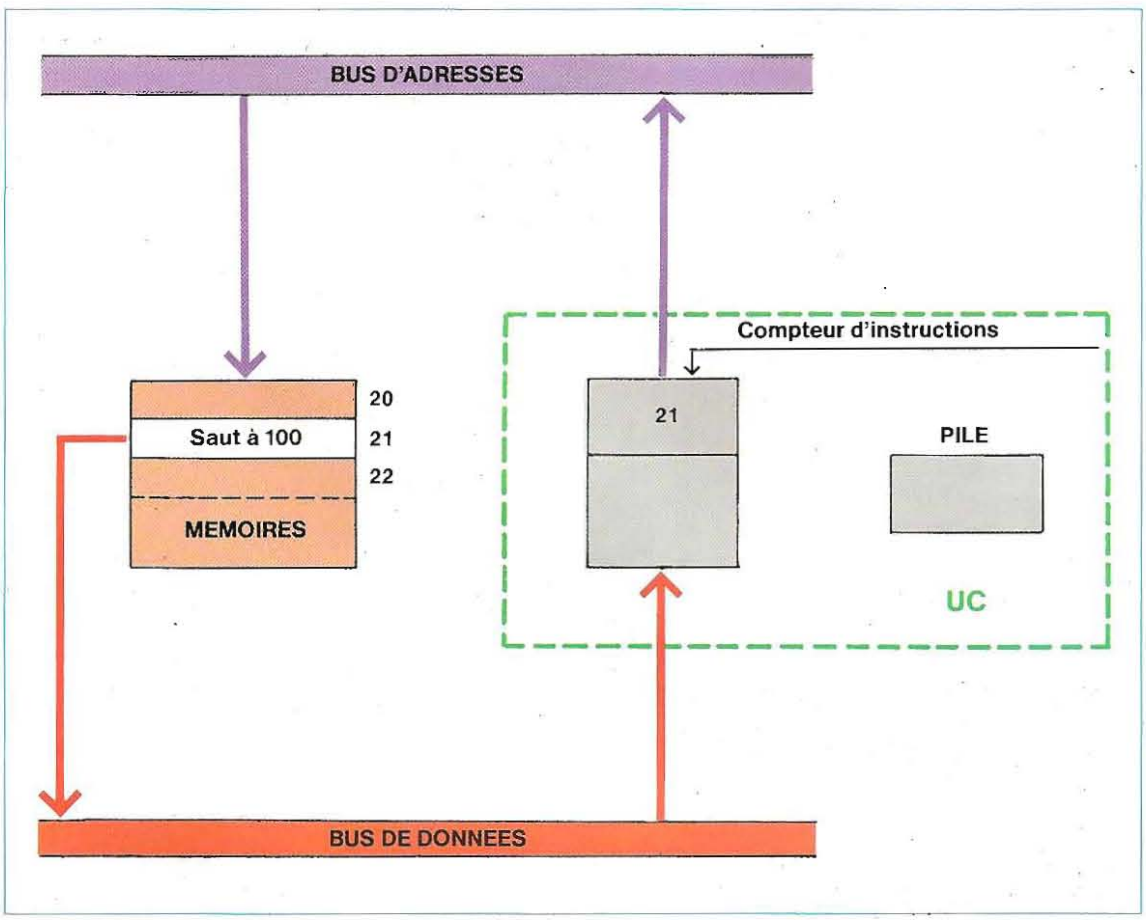
A la fin de ce cycle, on ajoute 1 au contenu du compteur d'instructions ; dans l'instruction suivante, la mémoire portera l'adresse 1 et ainsi de suite. Il arrive fréquemment que le déroulement d'un programme n'entraîne pas l'exécution obligatoire de toutes les instructions de manière séquentielle. Un choix logique peut conduire à sauter une partie du programme. Pour stocker en mémoire le contenu du compteur d'instructions et pouvoir le remettre à zéro, on prévoit à la fin du saut une zone de mémoire dénommée **pile** (stack) qui est une file d'attente.

A l'arrivée d'une instruction de rupture de séquence, l'unité centrale place le contenu du compteur d'instructions dans la pile ; le compteur contient alors la valeur nouvelle (adresse), dont on extrait et exécute l'instruction. L'unité centrale reprend ensuite la valeur primitive du compteur d'instructions, ajoute l'**incrément**, et procède de manière séquentielle jusqu'à la prochaine instruction de saut.

Phases d'une instruction de saut

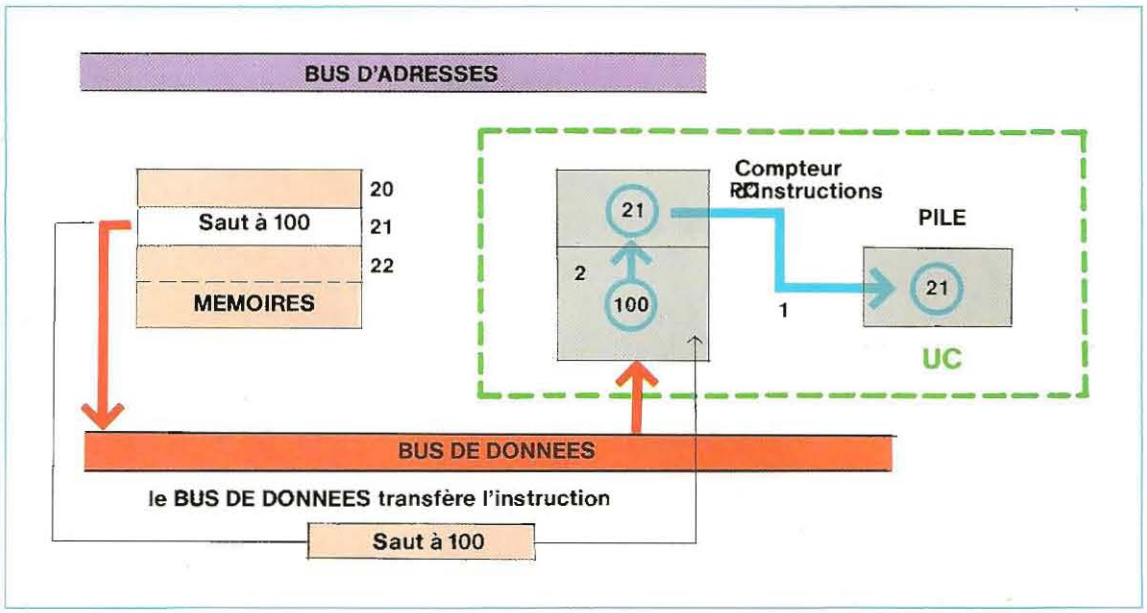
Première étape

Dans l'exécution du programme, le compteur d'instructions est parvenu (par exemple) à la valeur 21. Cette valeur est envoyée sur le bus d'adresses. La mémoire n° 21 répond en émettant sur le bus de données son contenu, par exemple le code correspondant à l'instruction « saut à 100 ». La file d'attente (pile) n'est pas encore concernée.

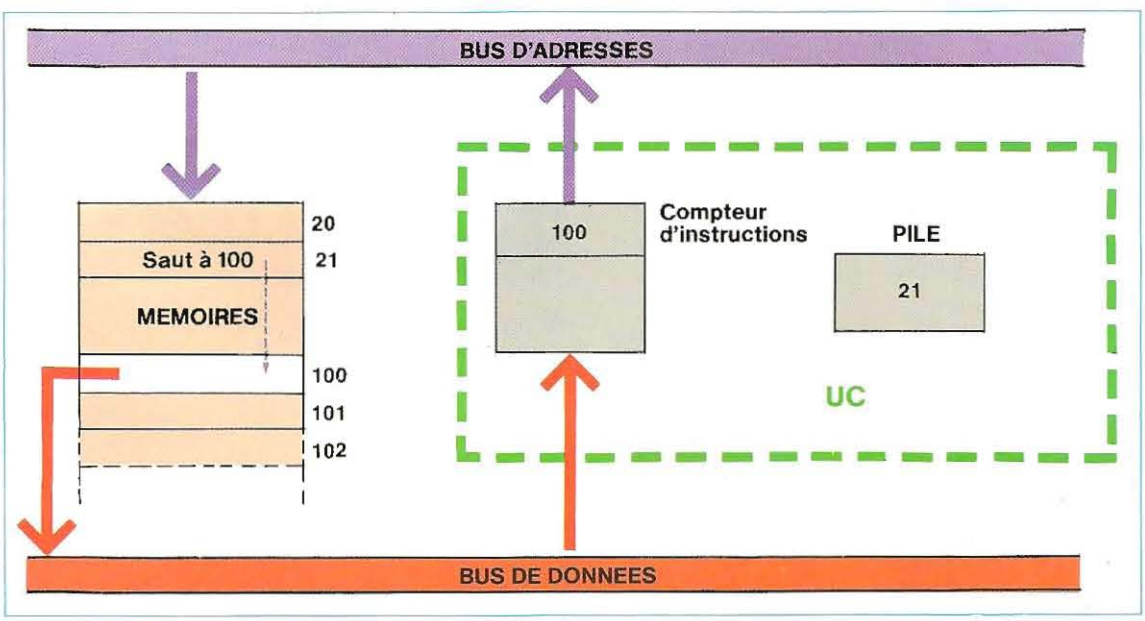


Seconde étape

L'unité centrale reconnaît l'instruction de saut ; elle archive le contenu du compteur d'instructions dans la pile 1 et transfère la nouvelle adresse (100) dans le compteur d'instructions 2. Cette valeur 100 représente l'instruction ultérieure à exécuter.



Le fonctionnement reprend alors son allure normale ; l'unité centrale envoie sur le bus d'adresses le contenu du compteur d'instructions (100) ; la mémoire correspondante répond en envoyant son contenu sur le bus de données.



Cette nouvelle instruction est exécutée, et l'incrément 1 ajouté au compteur d'instructions ; l'unité centrale exécute à présent l'instruction portant l'adresse de mémoire 101, et ainsi de suite jusqu'à ce qu'elle rencontre une instruction particulière qui commande le retour (Return).

Quand l'unité centrale rencontre cette instruction, elle se contente de prélever le contenu de la pile et de le déposer avec incrément dans le compteur d'instructions.

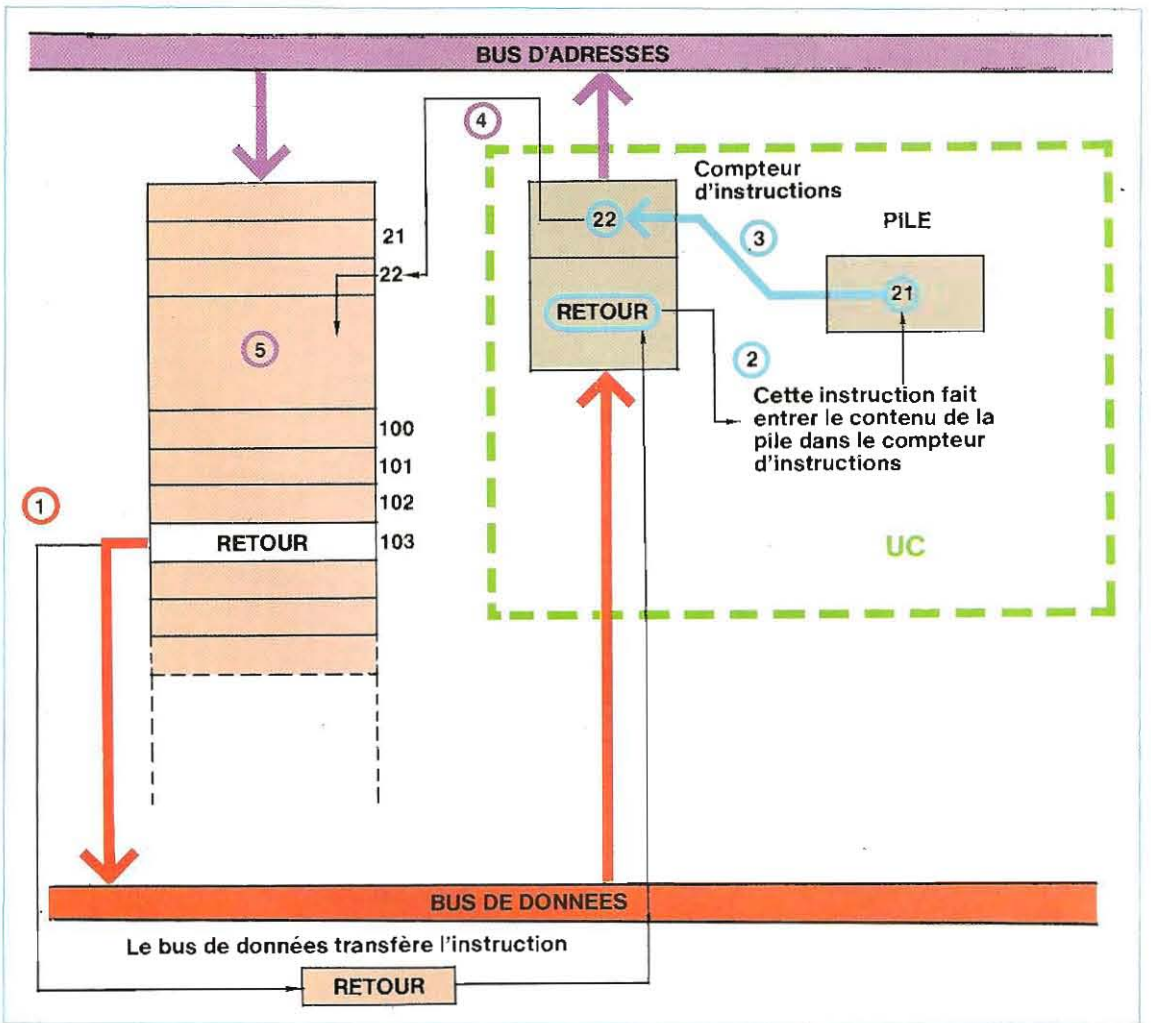
Le programme repart alors sur l'instruction qui suit le saut. Après le saut à 100 (voir schéma sur cette page), l'unité centrale exécute de manière séquentielle les instructions contenues en 101, 102 et 103. La mémoire 103 contient l'instruction Retour. Dès qu'elle identifie ce code, l'UC prélève le contenu de la file d'attente (21 dans cet exemple), l'incrémente et l'inscrit en compteur d'instructions (voir schéma ci-dessous : ①, ②, ③).

Dès ce moment, le programme reprend son cheminement initial (instructions 22 et suivantes : ④, ⑤).

Cette technique de sauvegarde de l'adresse de retour peut être utilisée pour des sauts successifs. Par exemple, si la mémoire 101 contenait à son tour une instruction de saut, le fonctionnement se ferait suivant le même mécanisme, mais avec deux sauvegardes de l'adresse de retour, la première pour le premier saut (21) et la seconde pour le second saut (101). Le premier retour en arrière se ferait à 102 (101 + 1) et le second à 22 (21 + 1).

Dans ce cas, la pile occupe deux cases de mémoire. On appelle **pointeur de pile** l'adresse de l'emplacement de mémoire contenant le dernier article rangé dans une pile.

Dans les systèmes les plus généralement utilisés, la pile peut être réalisée sur la mémoire



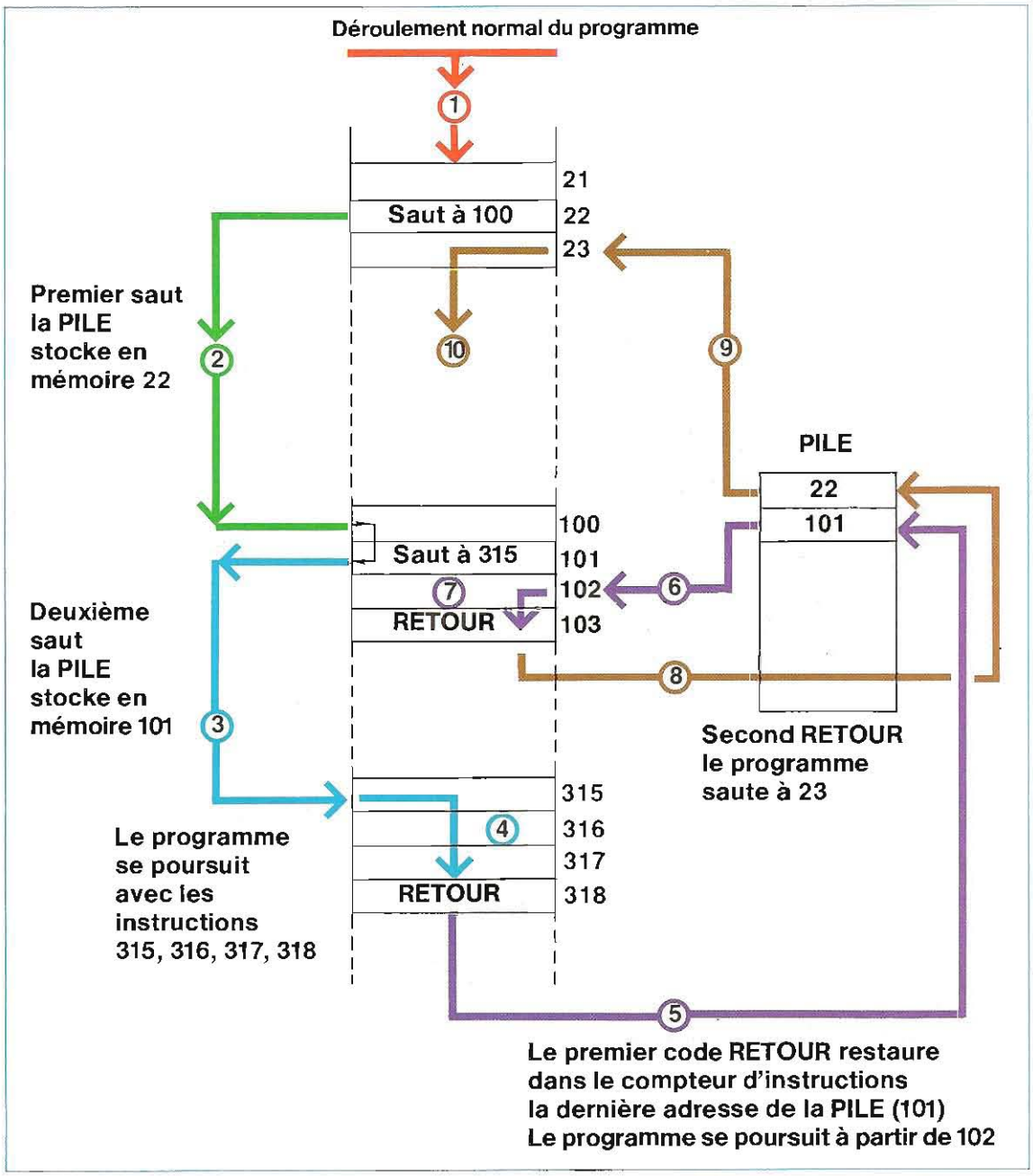
toute entière et ne connaît donc pas d'autres limites que celles de la mémoire elle-même. Le schéma de cette page illustre la logique de fonctionnement avec deux sauts successifs. La séquence des opérations est numérotée de **1** à **10**.

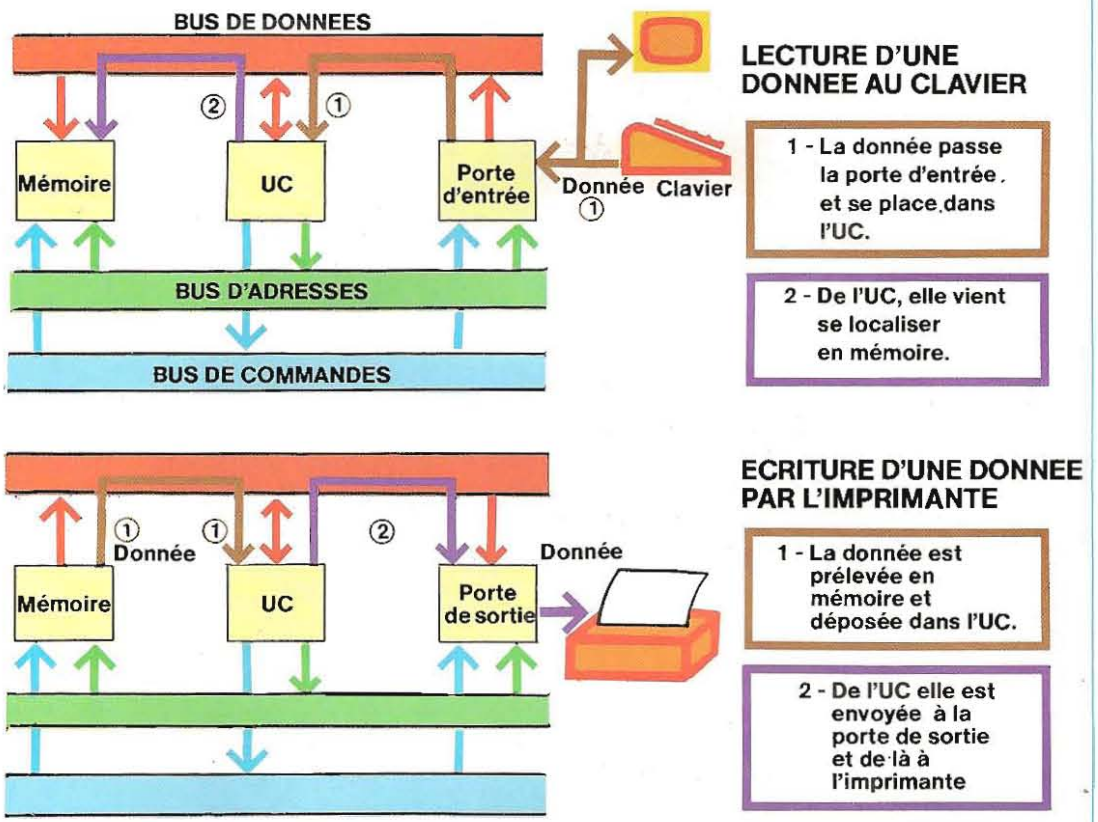
Gestion des interruptions

Le déroulement normal d'un programme peut être momentanément arrêté par l'apparition d'un signal d'interruption (Interrupt).

Ce signal arrête momentanément les opérations en cours pour permettre le déroulement d'une autre fonction, et à la fin de celle-ci, le retour au point où le programme a été interrompu.

Dans tous les systèmes, les périphériques ont des niveaux de priorité différents : normalement, l'élément prioritaire est le clavier, moyen d'entrée des commandes ; les autres (écran, imprimante) peuvent fonctionner si le clavier n'exerce pas sa priorité.





Cette logique de fonctionnement ne peut être gérée que par les signaux d'interruption.

Les interruptions de programme sont nécessaires pour laisser place aux demandes de service provenant de l'extérieur et non prévues dans le programme d'application. Ainsi en est-il de l'opération qui consiste à interrompre une impression (par exemple si les données ou la mise en colonnes sont erronées). Le programme d'impression lui consacre toutes les ressources du système ; l'unité centrale prélève les données en mémoire, leur attribue un code et les envoie à l'imprimante. Dans ces conditions, le clavier, organe de commande, est totalement inactif, le programme d'application ne prévoyant pas d'entrée de données. L'introduction de nouvelles instructions ne peut se faire qu'à la fin du programme qui se déroule, une fois l'unité centrale libérée de la fonction d'impression.

En réalité, il suffit d'entrer au clavier un code d'interruption : l'UC arrête le déroulement du programme d'application et identifie le périphérique « appelant » avant de lui donner la main. L'opérateur peut alors communiquer avec le système et entrer l'ordre d'interruption de l'imprimante.

Lecture des données

Les périphériques d'un système se classent en deux catégories principales :

- **périphériques d'entrée** : ce sont les éléments qui permettent d'introduire les données dans la machine. Pour les ordinateurs individuels et les micro-ordinateurs, cette catégorie ne comprend généralement qu'un clavier.
- **périphériques de sortie** : ce sont les éléments permettant à la machine de communiquer avec l'opérateur. Il s'agit essentiellement de l'écran de visualisation et de l'imprimante.

Les opérations d'E/S (entrée/sortie) sont illustrées par le schéma de cette page.

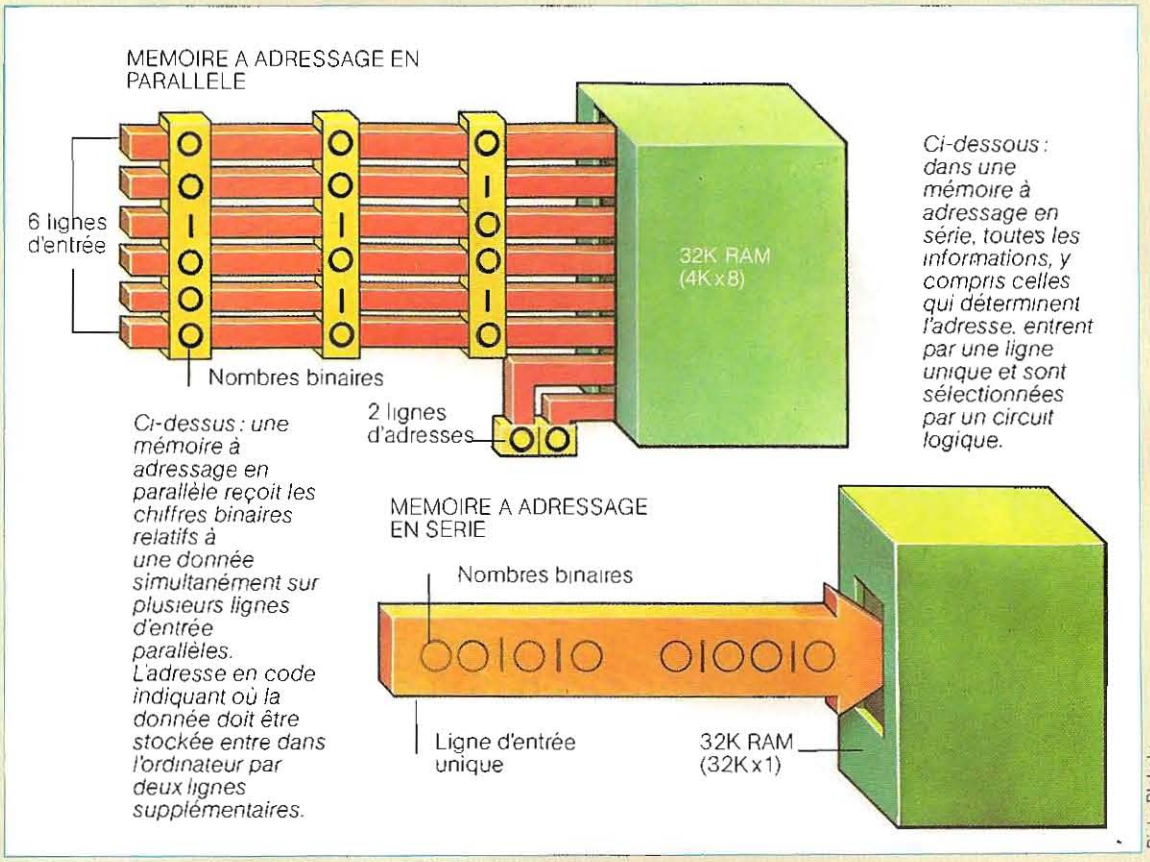
Des mémoires qui n'oublient rien

Même dans les ordinateurs les plus puissants, l'unité centrale de traitement ne peut effectuer qu'une opération à la fois. Après avoir exécuté un calcul, elle stocke le résultat en mémoire, puis passe au suivant, qu'elle stocke ailleurs, etc. L'ordinateur traite ainsi des milliers d'opérations dans une fraction de seconde. Quand il a besoin d'un résultat, il faut qu'il puisse y avoir accès immédiatement. La structure de la mémoire doit donc autoriser l'introduction et l'extraction à volonté de toutes les données, sans avoir à parcourir tout le circuit comme on le ferait avec les pages d'un carnet.
Ce type de mémoire à accès sélectif, ou direct, s'appelle mémoire vive, ou RAM (de l'anglais Random Access Memory). Elle permet d'adresser et de classer toute information reçue ou émise.

Nous avons vu que l'ordinateur travaille sur les données après les avoir transformées en binaire. Une fois codée, la donnée se présente sous la forme d'une chaîne de chiffres binaires adressés, un par un, en mémoire (adressage en série).
On peut aussi présenter à la mémoire tous les chiffres binaires en même temps, sur des conducteurs parallèles, correspondant chacun à l'une des puissances de la base 2 (adressage en parallèle).
La mémoire est constituée de rangées et de colonnes de transistors, alignés ou disposés sur une unique pastille, ou puce (chip), de silicium. Cette pastille doit également être pourvue des circuits nécessaires pour commander, sélectionner, envoyer, rechercher et identifier les informations qui entrent dans la mémoire ou en sortent.

Il existe deux types de mémoires vives à semi-conducteurs : les mémoires dynamiques et les mémoires statiques.
Une mémoire dynamique ne conserve son contenu qu'un certain temps, il faut donc la « rafraîchir » régulièrement à l'aide d'un signal d'horloge, ce qui exige des circuits complémentaires. Chaque cellule de mémoire (ou unité de stockage sur la micropastille) est cependant plus petite que dans le cas d'une mémoire statique où une cellule exige au moins deux transistors pour stocker un bit. Si la mémoire dynamique tend à perdre son contenu, c'est que les chiffres 0 ou 1 y sont emmagasinés sous forme d'une charge électrique (égale à 0, ou différente de 0). Le transistor se comporte donc comme un simple condensateur et perd graduellement sa charge sous l'effet du phénomène de dispersion.
Dans la mémoire statique, au contraire, le signal est enregistré par un dispositif à bascule (flip-flop, ou plus exactement bascule bistable). Dans la pratique, c'est un interrupteur électronique ne pouvant être que ouvert (ON) ou fermé (OFF), ce qui donne la possibilité de représenter les deux chiffres binaires 0 et 1 sans avoir à emmagasiner de charges électriques.

Le transistor le plus couramment utilisé pour les mémoires est de type MOSFET (sigle anglais correspondant à transistor en métal oxyde semi-conducteur à effet de champ). Des milliers de transistors de ce type peuvent être imprimés par photogravure sur un centimètre carré de silicium semi-conducteur. Chaque transistor est composé d'un microscopique cristal de matériau semi-conducteur, traversé par un courant électrique circulant entre deux électrodes, que l'on appelle émetteur et collecteur. Dans la masse du semi-conducteur, une troisième électrode, assurant la régulation du courant, s'appelle porte en raison de sa fonction.
Le transistor à effet de champ faisant partie d'une mémoire est employé comme un interrupteur qu'on peut fermer en envoyant un signal à sa porte.
En position de fermeture du circuit (ouverture de la porte), le courant circule de l'émetteur au collecteur en traversant le dispositif et va charger un condensateur relié au transistor. Ce condensateur prend ainsi un état de charge correspondant au chiffre 1 du code binaire, ou un état de non-charge (chiffre 0). Cette technique permet de stocker les informations en binaire.
L'ensemble du MOSFET et du condensateur constitue la cellule de mémoire. La porte de chaque cellule est connectée à une ligne de sélection que l'on active par une instruction appropriée lorsqu'on veut qu'un chiffre soit chargé dans le condensateur de la cellule.



Dans une mémoire à adressage en série, chaque chiffre binaire s'accompagne d'une adresse à sélection de colonne à deux bits qui définit dans quelle colonne l'information doit entrer. Un circuit logique assure l'envoi des chiffres à leur adresse exacte.
Dans une mémoire à adressage en parallèle, un mot se présente sur plusieurs lignes d'entrée. Les signaux circulant sur les deux lignes d'adresses stockent le mot dans la zone voulue. Cette méthode permet de stocker un grand nombre d'informations sur une seule pastille de petite dimension.

Le recours aux résistances proprement dites est restreint dans les mémoires RAM statiques à MOSFET. On les remplace en effet par un transistor dont la porte est connectée au collecteur et qui constitue une résistance non linéaire de surface réduite.
Pourtant, la société Mostek a réhabilité l'usage des résistances, en choisissant un silicium polymorphe à implantation ionique qui, pense-t-on, n'occuperait guère plus d'espace que le système actuel mais également avec une moindre consommation d'énergie.

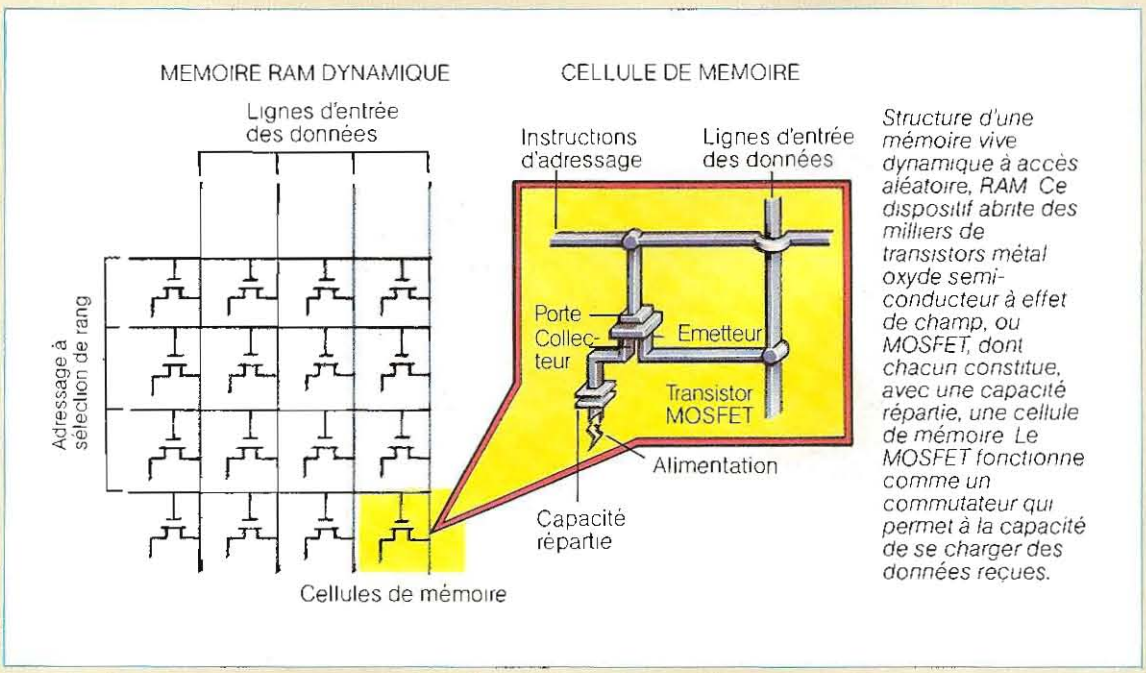
Les mémoires vives ne sont pas toutes construites sur la technologie MOS (métal oxyde semi-conducteur). On utilise aussi des mémoires dites bipolaires, que l'on tend toutefois à réaliser à l'aide de couples de bascules, tout comme les mémoires statiques MOS, d'où une augmentation de la surface nécessaire pour chaque cellule. Les mémoires bipolaires peuvent être beaucoup plus rapides que les mémoires dynamiques, même si celles-ci restent plus économiques.

Il existe une variante de la mémoire vive statique MOS : la mémoire MOS complémentaire, ou CMOS, dans laquelle les résistances de charge du couple croisé de transistors constituant un flip-flop sont remplacées par des transistors polarisés en direction opposée à celle des véritables transistors de commutation. La résistance présentée par les deux transistors de charge est infinie en théorie, le courant ne passe donc pas et il n'y a pas dissipation d'énergie. Il en résulte que les

mémoires vives statiques CMOS consomment encore moins d'énergie, ce qui explique leur utilisation dans le domaine spatial, malgré un fonctionnement plus lent. Si, en théorie, aucun courant ne doit passer à travers les transistors des mémoires CMOS, on constate dans la pratique un flux de courant au cours de la brève période de commutation de la bascule passant d'un état à l'autre. Cet inconvénient, dû aux effets des capacités parasites, a été récemment éliminé par la société RCA, qui a mis au point une méthode de réalisation de mémoire CMOS sur substrat de saphir. Cette technique multiplie par quatre la vitesse de fonctionnement. Mais, le saphir étant beaucoup plus coûteux que le silicium, les recherches se poursuivent, sans que l'avenir de ce procédé soit assuré.

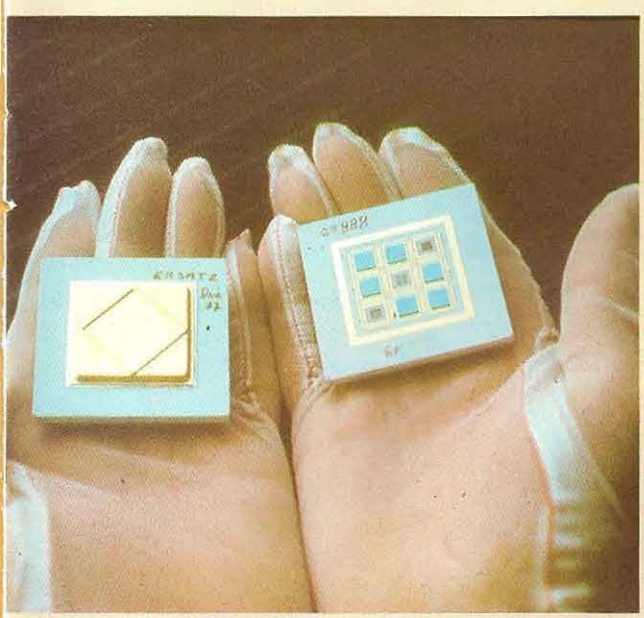
Toutes les mémoires vives à semi-conducteurs sont volatiles. On a étudié de nombreuses méthodes pour remédier à ce défaut. En cas de coupure de l'alimentation électrique de la mémoire, on peut effectuer le transfert immédiat des informations les plus importantes dans une mémoire magnétique à disque, ou à ruban.
On emploie aussi des mémoires à semi-conducteur non volatiles, pour stocker des programmes à durée indéfinie, par exemple celui d'un jeu vidéo, ceux qui commandent les opérations internes d'un ordinateur, ou qui indiquent à la machine les phases d'une procédure mathématique.
Ces mémoires ne peuvent fournir qu'un ensemble figé d'informations, lues et utilisées continuellement par l'ordinateur. Ce sont les mémoires mortes ROM (Read Only Memory). Elles sont formées d'une simple matrice à diodes et leur programmation est effectuée une fois pour toutes en faisant fondre certaines diodes par application d'un courant de haute tension. La présence ou l'absence de connexion entre les lignes peut servir à indiquer un état logique 1, ou 0. La simplicité d'une mémoire morte explique son coût très faible et la haute densité d'informations que l'on peut y stocker.

Une fois programmées, les mémoires mortes ne peuvent plus être modifiées. Cependant, il existe aujourd'hui d'autres familles de mémoires mortes, des composants que l'on peut effacer en exposant la matrice de diodes à un faisceau ultraviolet à travers une petite fenêtre pratiquée au sommet du dispositif, puis reprogrammer. Ces mémoires ont été dénommées EPROM (Electrically Programmable Read Only Memory), par opposition aux mémoires mortes programmables (PROM).
Les mémoires EPROM ont à présent atteint



Structure d'une mémoire vive dynamique à accès aléatoire, RAM. Ce dispositif abrite des milliers de transistors métal oxyde semi-conducteur à effet de champ, ou MOSFET, dont chacun constitue, avec une capacité répartie, une cellule de mémoire. Le MOSFET fonctionne comme un commutateur qui permet à la capacité de se charger des données reçues.

Ricky Blakely



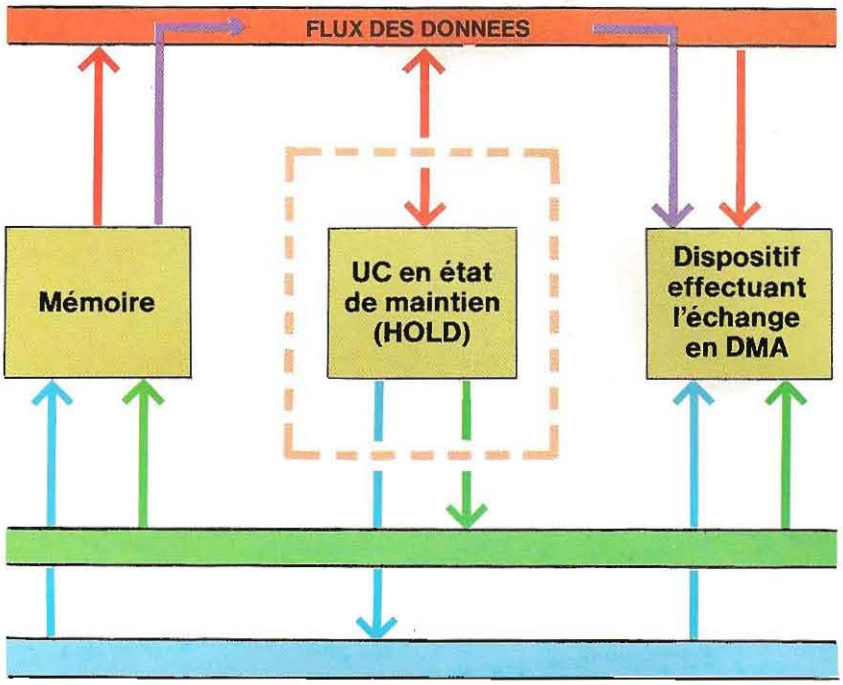
Archives Curcio/IBM

Deux phases du montage des pastilles (chips) de mémoire sur support céramique. A droite : les plaquettes de support. A gauche : les plaquettes isolées et recouvertes. A ce stade, le module est prêt à être mis en place.

une capacité de 64 Kbits (kilobits), les travaux de réalisation des versions à 128 et 256 Kbits sont déjà très avancés.
Dans une autre variante de la mémoire morte, dénommée EAROM (Electrically Alterable Read Only Memory), il n'est pas nécessaire d'effacer toute la mémoire pour en reprogrammer une partie puisque la simple application de tensions plus élevées que la normale provoque la reprogrammation des cellules individuelles.

Il existe aussi d'autres systèmes prometteurs pour stocker des données numériques, mais ils ne sont pas encore apparus en grande quantité sur le marché. Les plus intéressants sont les mémoires à bulles magnétiques et ceux qui utilisent les dispositifs à couplage de charge (CCD, Charge Coupled Device).
Dans le cas des bulles magnétiques, les informations sont stockées en utilisant la propriété de certains cristaux qui, sous l'action d'un champ magnétique, emmagasinent ce que l'on pourrait considérer comme des bulles dotées d'une polarité magnétique. Le sens de cette polarité détermine si c'est un 1 ou un 0 qui se trouve en mémoire. La signification 1 ou 0 peut aussi être donnée par la présence ou l'absence de bulle. Ces dispositifs permettent de stocker un très grand nombre de données (20 000 bits sur 5 mm²).
Contrairement aux disques magnétiques, le lecteur, support des mémoires à bulles, étant fixe, ne peut déchiffrer une information que lorsque celle-ci défile devant lui sous l'effet du mouvement des bulles à l'intérieur du support. C'est le déplacement du champ magnétique qui provoque le mouvement des bulles. Tout cela rallonge le temps d'accès, mais, comme dans tous les dispositifs magnétiques, les bulles ont l'avantage de ne pas être volatiles ; en outre, la densité de stockage autorisée par ce procédé est à peu près dix fois supérieure à celle d'une mémoire à semiconducteurs.
De telles mémoires sont désormais largement utilisées par des entreprises comme IBM, Texas Instrument et des constructeurs japonais, entre autres.
Les dispositifs à couplage de charge appliquent une technique fondée sur les semiconducteurs qui les rend comparables à d'immenses fichiers tournants. Les informations sont stockées en une longue suite et l'on ne peut y accéder, comme dans le cas de mémoires à bulles, que lorsqu'elles parviennent devant des têtes de lecture spéciales. Physiquement, une mémoire CCD est un transistor à effet de champ de grande longueur, doté d'un certain nombre de sorties (ou électrodes) disséminées entre les deux extrémités, exactement comme un tuyau d'arrosage pour pelouse percé de nombreux trous.
On peut évidemment faire varier la quantité de charge qui s'écoule le long du transistor avec plus de précision que pour un tuyau d'arrosage, et l'on présente à chaque électrode un tableau complet de la série de signaux qui parviennent à l'entrée du dispositif.
Pour que les informations progressent le long du dispositif à couplage de charge, il faut fournir régulièrement un signal d'horloge, et le temps d'accès à chaque bit est plus long qu'avec d'autres dispositifs comparables.
On pense que ce système devrait être peu coûteux, mais il n'est pas sûr qu'il convienne à un emploi généralisé.

FLUX DES DONNEES DANS L'ACCES DIRECT A LA MEMOIRE



En lecture, par le clavier par exemple, la donnée est transférée d'abord à la porte d'entrée puis à l'unité centrale et enfin à la mémoire.
Au contraire, en sortie, la donnée est prélevée en mémoire, transférée à l'unité centrale, puis envoyée à la porte de sortie.
Cette gestion des opérations d'E/S, qui implique un passage par l'unité centrale, ne peut se faire à très grande vitesse et s'applique donc au transfert des données vers les périphériques lents (le clavier et la plupart des imprimantes).
Pour obtenir des échanges de données plus rapides on utilise plutôt la technique de l'accès direct à la mémoire (DMA, Direct Memory Access) qui consiste en un échange direct entre la mémoire et le dispositif qui reçoit ou transmet les données. L'unité centrale n'a pas à intervenir dans cet échange, son activité est donc suspendue jusqu'à la fin de celui-ci.
Cet état d'activité suspendue s'appelle maintien (HOLD).
Le schéma de cette page illustre un échange en accès direct à la mémoire.
Ce type d'échange est généralement adopté pour des unités à disque, dispositifs ayant à traiter de grandes quantités de données.
Une disquette peut contenir 80000 à 1200000 caractères ; les disques rigides pour micro-ordinateurs peuvent dépasser 10 millions de caractères ; une économie de temps minime sur chaque caractère se traduit donc au total par un gain appréciable.
En outre, dans les programmes d'application, les données sont continuellement extraites de l'unité à disque, soumises à des calculs puis à nouveau stockées sur ces unités : cet accès continu au disque rend indispensable l'utilisation de la procédure d'accès direct.
Le choix de la procédure est automatique, les programmes se chargeant de la sélection et de l'exécution sans que l'utilisateur ou le programmeur aient à s'en préoccuper.
Cette catégorie de programmes fait partie du système opératoire fourni avec la machine.

Les mémoires

Les mémoires sont les dispositifs dans lesquels on stocke les programmes et les données utilisés ou produits. Il s'agit donc d'une mise en mémoire provisoire, puisqu'en écrivant le nouveau programme on détruit le précédent; de même, lorsqu'on coupe l'alimentation électrique de la machine, le contenu de la mémoire disparaît (sauf avec certains types particuliers). Le stockage permanent se fait par transfert sur disques (ou sur ruban magnétique).

La zone de mémoire est divisée en quatre blocs fondamentaux:

1 / zone des signaux d'interruption
2 / zone de système
3 / zone des programmes de service
4 / zone utilisateur

La zone 1 stocke les instructions qui permettent de gérer les signaux d'interruption de programme. Les zones 2 et 3 sont consacrées au système d'exploitation, ensemble de programmes effectuant les fonctions fondamentales d'impression, lecture au clavier, gestion des disques, etc. La zone 4 est disponible pour les programmes d'application (et les données qui s'y rattachent).

Les ordinateurs individuels et micro-ordinateurs disposent d'un minimum de 16 000 à 64 000 caractères de mémoire, dont en général 28 000 ou 58 000 sont réservés à l'utilisateur, selon la complexité du système d'exploitation. Dans les micro-ordinateurs les plus perfectionnés et les mini-ordinateurs, on dépasse 260 000 caractères jusqu'à atteindre 2 000 000 (2 Mo) ou 4 000 000 (4 Mo) pour les machines plus importantes.

Le schéma de cette page énumère les divers types de mémoires.

Mémoires à lecture et écriture : ce sont les mémoires vives ou RAM (Random Access Memory). Elles permettent d'effectuer les opérations de lecture et d'écriture et se divisent en:

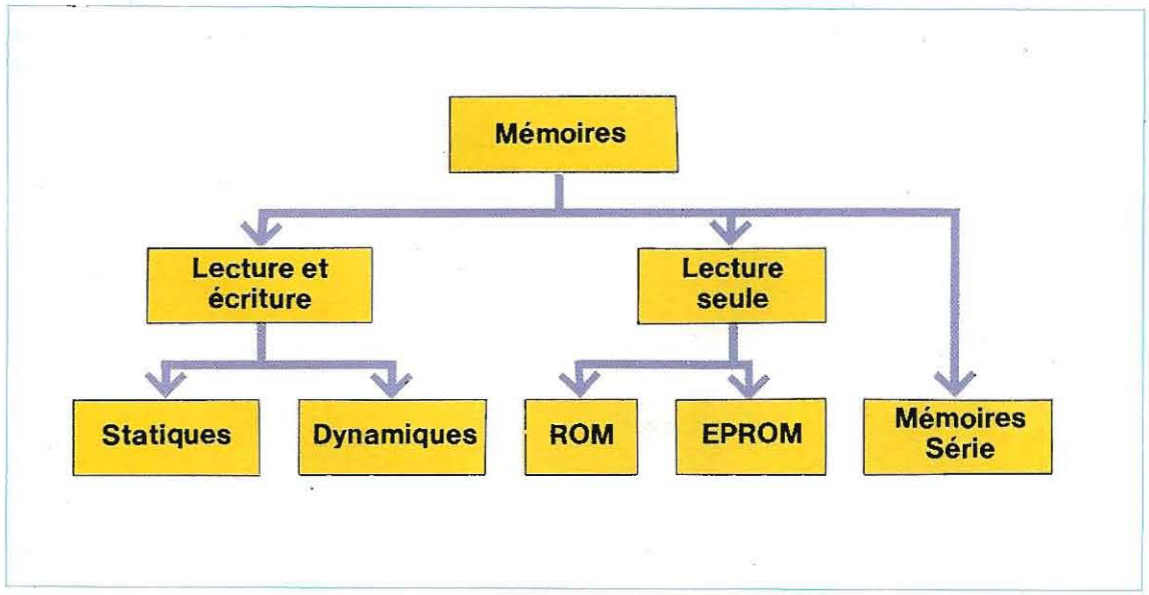
- **mémoires dynamiques**
- **mémoires statiques**

Le type dynamique exige un entretien, un rafraîchissement (Refreshing en angl.) régulier des données (quelques centaines de fois par seconde) pour en éviter l'effacement. Le type statique ne nécessite aucune opération d'entretien de la mémoire, une fois la donnée écrite: celle-ci peut être extraite même au bout d'un certain temps.

Mémoires à lecture seule : ce sont des mémoires sur lesquelles on ne peut intervenir. Leur emploi est réservé aux programmes qui n'ont pas à être modifiés. Elles se divisent en deux types:

- **ROM (mémoire morte)**
- **EPROM (mémoire morte reprogrammable)**

Le type ROM (Read Only Memory) comporte toutes les données nécessaires dès sa construction; il n'existe donc aucun moyen de le modifier. Le type EPROM (Electrically Programmable ROM) peut être effacé par exposition aux rayons ultraviolets puis reprogrammé.





Archives Curcio/Bufletti Data

Appareil portatif de programmation de mémoire EPROM.

Mémoire série : ce sont des mémoires d'emploi très particulier qui donnent des possibilités très grandes mais exigent des circuits complexes. De ce fait, elles sont peu utilisées.

Les périphériques

Ecran de visualisation (moniteur) : il ressemble beaucoup à un poste de télévision normal, au point que certains micro-ordinateurs économiques utilisent simplement un appareil de télévision classique (avec un modulateur). Toutefois, l'image sera moins bonne.

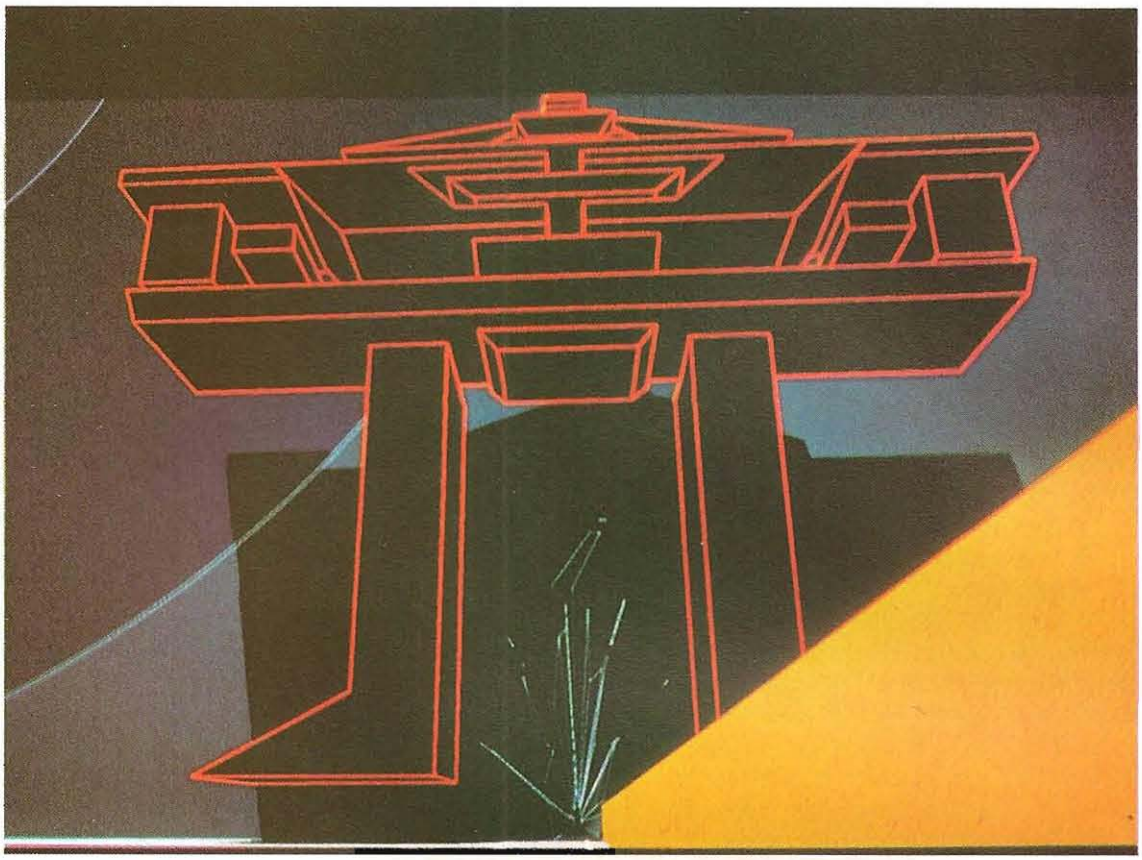
Les moniteurs peuvent être monochromes (gris, vert, ou ambré), ou polychromes, et se différencient par leur taille. Dans le cas d'un moniteur monochrome, la qualité de l'affichage est déterminée par sa bande passante et par son prix. Il permet d'afficher avec plus ou moins de stabilité et de netteté un nombre variable de caractères alphanumériques (lettres et chiffres) et graphiques (dessins).

La finesse d'une image graphique dépend du nombre de points élémentaires (picture element, ou pixel) qui la constituent. Le nombre de ces points varie selon le micro-ordinateur utilisé et la qualité de la restitution est fonction du moniteur. 256 x 192 (pixels) constituent une bonne résolution, mais il y a mieux... et moins bien !

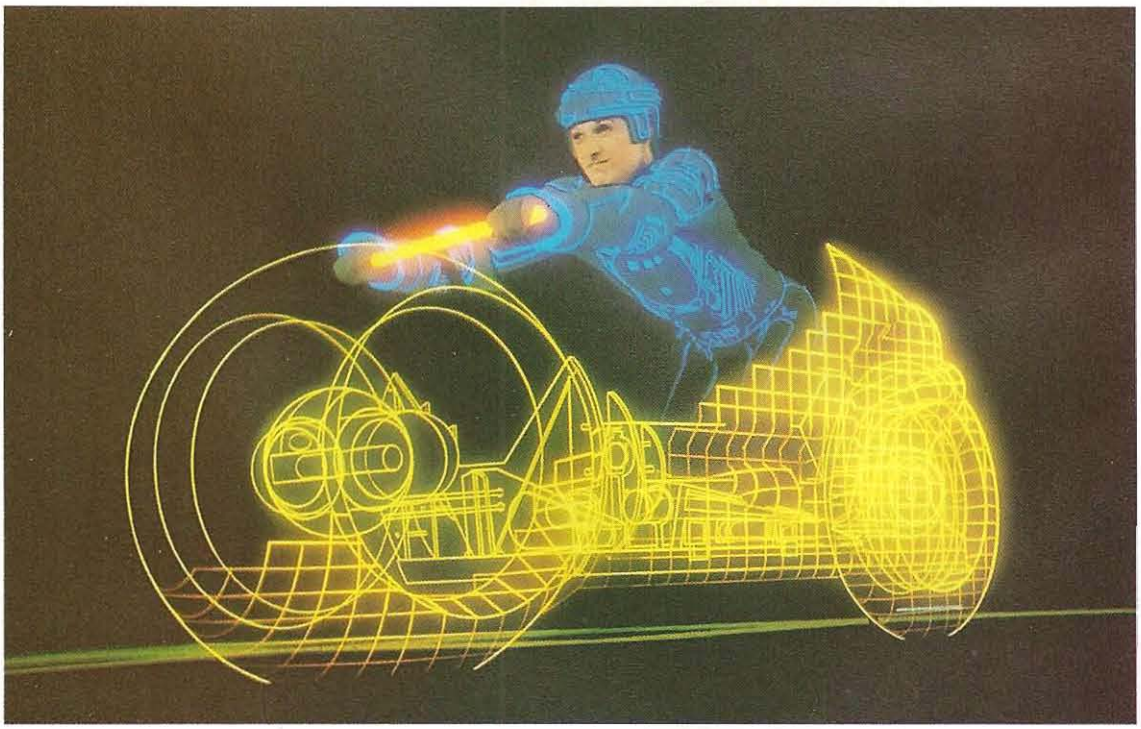
Les écrans couleurs peuvent être classés en deux catégories :

- téléviseur couleurs récent et disposant d'une entrée vidéo dite Péritel. C'est le cas le plus courant et le moins cher.
- vrai moniteur couleurs (ne pouvant pas recevoir les chaînes de télévision) ayant une bande passante beaucoup plus élevée qu'un téléviseur standard et permettant un affichage plus fin. Cependant, son coût le met hors de portée de l'amateur moyen, le réservant à des applications professionnelles.

Le nombre de caractères par ligne et le nombre de lignes par écran varient avec le type de micro-ordinateur utilisé.



1982 © Walt Disney Productions



1982 © Walt Disney Productions

Deux images du film « Tron » illustrant l'emploi du graphisme informatisé dans le cinéma d'animation.

Le clavier. Il est généralement divisé en deux sections : la partie alphanumérique qui contient (comme le clavier d'une machine à écrire) les caractères alphanumériques et les chiffres, et la partie numérique ne contenant que des chiffres. Sur certaines machines, chaque touche a deux significations : d'abord le caractère alphanumérique normal et ensuite une fonction déjà programmée (et que l'on obtient grâce à un code particulier). L'utilisation de ce type de clavier permet de gagner beaucoup de temps car il évite, dans l'écriture d'un programme, de rédiger en clair certaines instructions. Il suffit d'appuyer sur une touche pour en écrire une.

Les imprimantes. Dans les systèmes à microprocesseur, on utilise surtout les imprimantes :

- **à aiguilles**
- **à marguerite**
- **par ligne**

L'imprimante à aiguilles, ou matricielle : son nom lui est donné par le procédé de formation des caractères. La tête d'impression est composée d'un certain nombre d'aiguilles (sélectionnées par un électro-aimant) qui avancent en fonction de la lettre ou du symbole à imprimer. Le caractère n'est pas tracé d'un trait continu mais formé d'un ensemble de points très rapprochés.

On trouvera dans le tableau ci-dessous quelques-uns des caractères que l'on peut obtenir avec une imprimante à aiguilles.

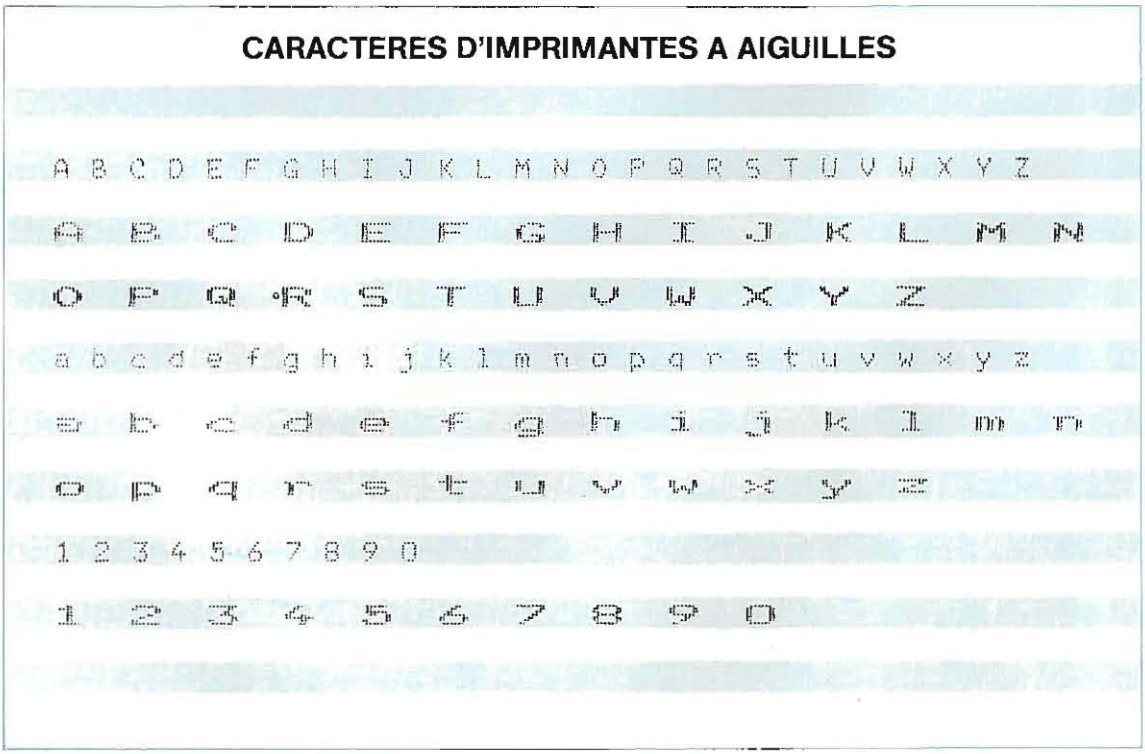
Les avantages de ces machines sont leur coût modeste, leur vitesse moyenne, leur possibilité d'impression de caractères différents.

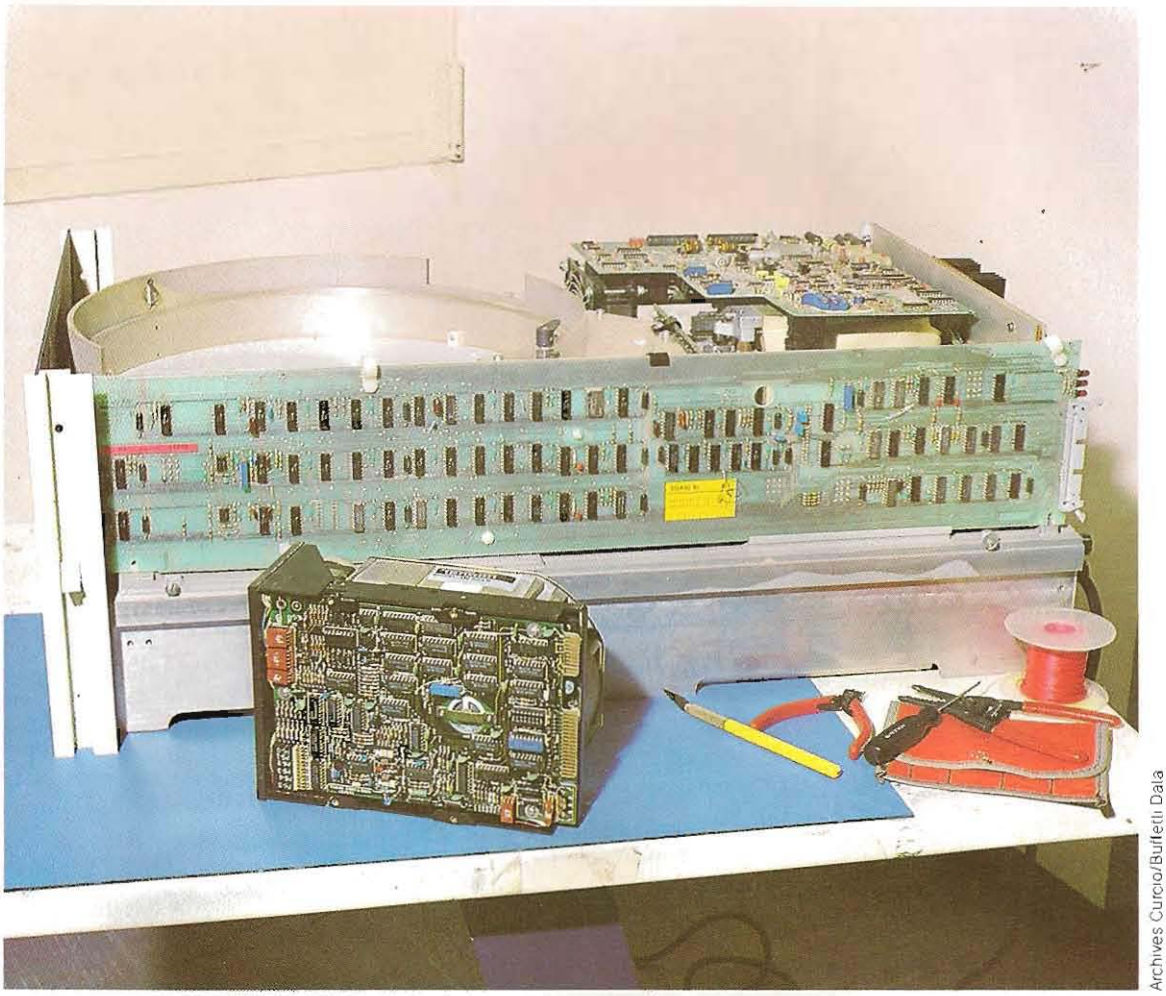
L'imprimante à marguerite, ou caractère par caractère : dans ce modèle chaque caractère tout préparé n'a plus qu'à être imprimé sur le papier. Les roues d'impression (marguerites) sont identiques à celles des machines à écrire. Ces imprimantes sont plus coûteuses que les précédentes, beaucoup moins rapides, et ne permettent de changer de caractère qu'en changeant de marguerite. En revanche, la qualité de l'impression est très soignée (qualité courrier).

L'imprimante par ligne : beaucoup plus rapide, elle imprime plus de 600 lignes par minute. Le coût en est très élevé et son emploi ne se justifie que dans les systèmes à mini-ordinateur de plus grande taille (listings). Citons encore les imprimantes à jet d'encre, les imprimantes thermiques, les imprimantes à laser connectables à des micro-ordinateurs.

Unités à disques. Les ordinateurs individuels et les micro-ordinateurs en utilisent deux types, à disque dur ou souple.

CARACTERES D'IMPRIMANTES A AIGUILLES





Archives Curcio/Buffetti Data

Comparaison de deux types d'unité à disque. En dépit de la grande différence de taille, la capacité de stockage des deux dispositifs est la même.

Le disque dur, qui est généralement fixe, a une capacité minimum de 5 ou 10 millions de caractères.
Le disque souple, ou disquette, est amovible, et sa capacité de mémoire varie entre 80 000 et 1 200 000 caractères (selon ses dimensions et son type).

Les disquettes peuvent être :

- **à simple ou double face**
- **à simple ou double densité.**

La première caractéristique indique si l'on peut utiliser un côté du disque ou les deux ; la seconde donne une idée de la densité de stockage des données.
C'est évidemment la disquette simple face-simple densité (SF-SD) qui contient le moins de données ; la capacité est nettement plus grande sur les disquettes double face-double densité (DF-DD).
Les systèmes de plus grandes dimensions (mini-ordinateurs) emploient aussi des disques durs fixes ou amovibles.
Le type fixe a une grande capacité de mémoire ; il est utilisé dans les applications d'une certaine complexité. Le second type (dur amovible) a une capacité de l'ordre de 10 millions d'octets (caractères) et un coût plus limité.
La photo de cette page compare deux unités à disque, ayant toutes deux une capacité de 10 Moctets (10 Méga-octets, 10 Mo, ou 10 millions de caractères) ; la plus grande est à disque dur amovible, l'autre est une unité à disquettes multiples. La photo montre clairement la différence d'encombrement.

Fonctionnement des mémoires mortes

Les mémoires mortes, ou à lecture seule (ROM), nous le savons, servent à stocker des algorithmes permanents pour commander l'exécution de certaines opérations. Ce logiciel permanent, ou microprogramme, est utilisé pour accomplir toute une gamme de fonctions répétitives dans de nombreuses machines. L'exemple ci-dessous décrit l'emploi d'une mémoire morte pour la commande d'un écran à sept segments, destiné à la présentation d'un nombre sur le visuel d'une calculatrice de poche.

Nous avons déjà appris que les signaux provenant du microprocesseur de la mémoire sont codés en binaire.

Un affichage numérique à sept segments est un dispositif numérique à base 10, et doit avoir comme tel la possibilité de recevoir en entrée dix signaux différents.

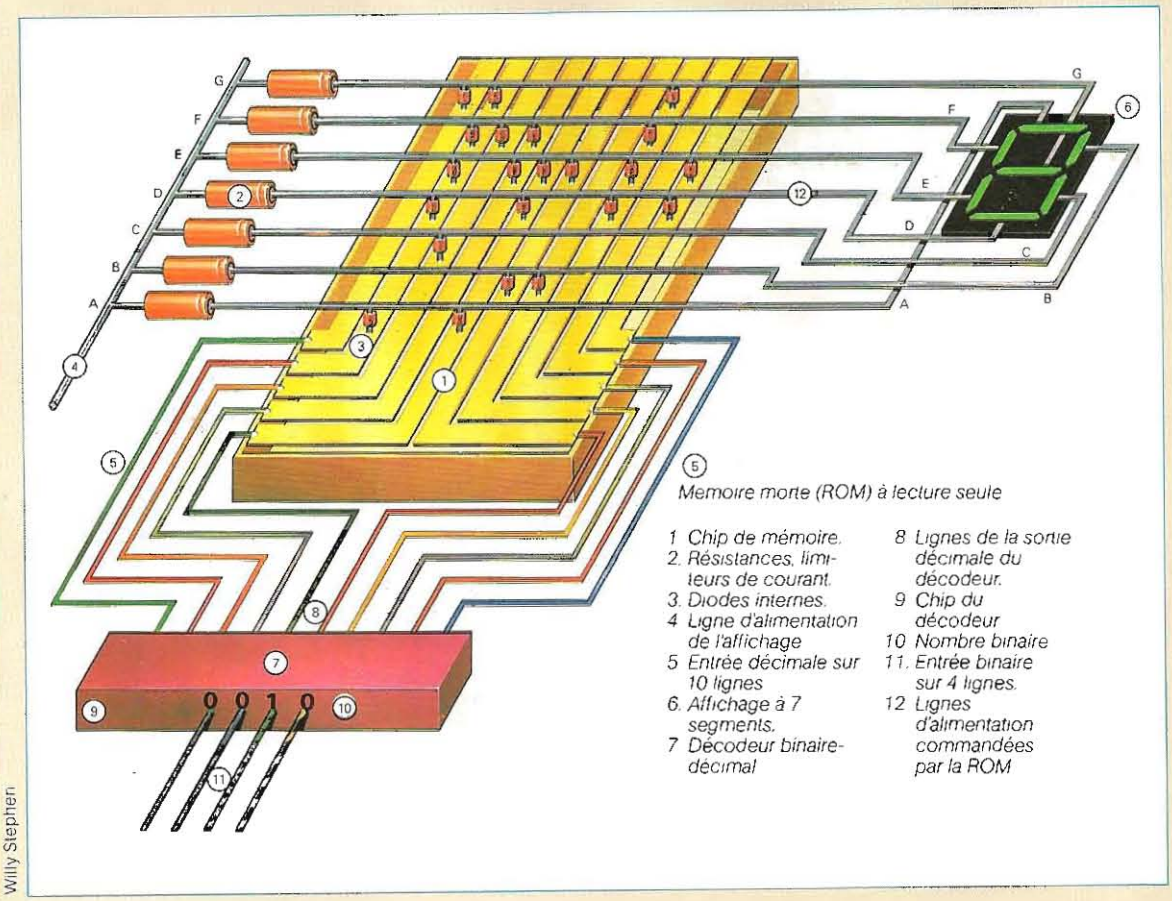
Le problème de la conversion est résolu par le recours à un décodeur binaire-décimal, composé d'un ensemble de portes logiques.

Les quatre lignes d'entrée étudiées dans cet exemple peuvent fournir au décodeur les chiffres binaires qui représentent les nombres de 0 à 9; pour convertir ces chiffres il faut, en revanche, dix lignes de sortie.

En l'absence de signaux binaires en entrée, les dix lignes de sortie sont parcourues en permanence par une tension continue, celle du fonctionnement de l'affichage (en général 5 volts).

Quand, par exemple, le décodeur reçoit le nombre binaire 0000 (0 dans le système décimal), il fournit en sortie un 0, ce qui s'obtient en produisant une absence de tension sur la première ligne de la sortie décimale (celle du 0) et un 1 (présence de tension) sur toutes les autres lignes.

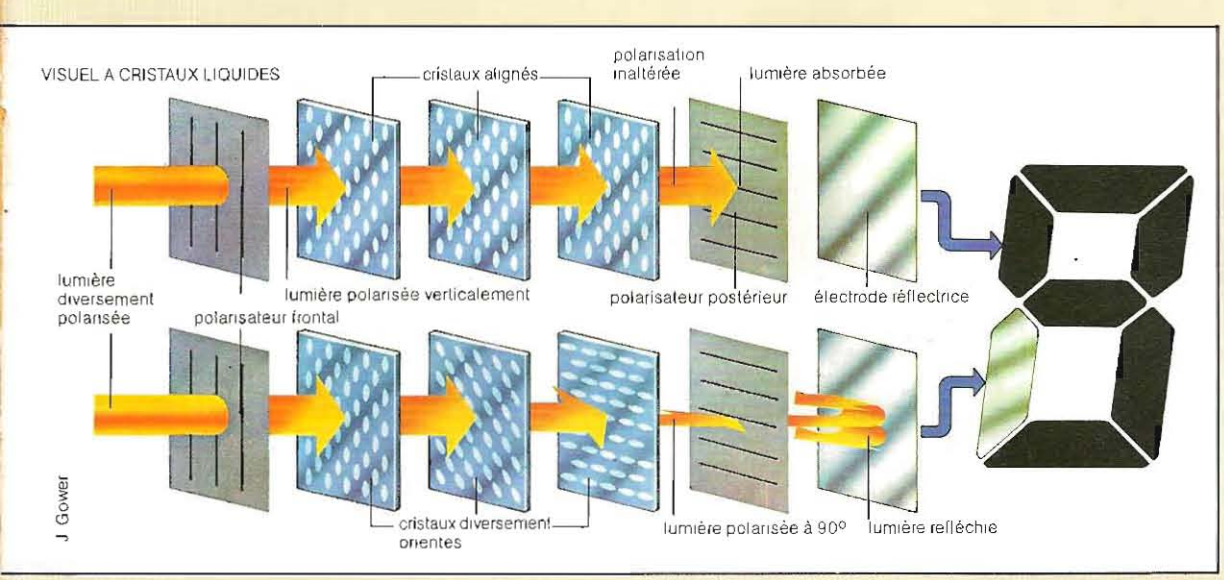
Si le décodeur reçoit le nombre binaire 0001 (1 dans le système décimal) il produit un 0 (absence de tension) sur la deuxième ligne (qui représente le chiffre décimal 1) alors que toutes les autres lignes transportent un 1 (présence de tension).



5 *Mémoire morte (ROM) à lecture seule*

1. *Chip de mémoire.*
2. *Résistances, limiteurs de courant.*
3. *Diodes internes.*
4. *Ligne d'alimentation de l'affichage*
5. *Entrée décimale sur 10 lignes*
6. *Affichage à 7 segments.*
7. *Décodeur binaire-décimal*
8. *Lignes de la sortie décimale du décodeur.*
9. *Chip du décodeur*
10. *Nombre binaire*
11. *Entrée binaire sur 4 lignes.*
12. *Lignes d'alimentation commandées par la ROM*

Willy Stephen



On peut répéter le même procédé pour tous les autres nombres décimaux.

On utilise cette tension nulle comme indicateur, comme on le fait avec les logiques inversées parce que l'absence de tension dans une ligne signifie qu'elle est reliée à la masse, et que, par conséquent, elle possède la propriété d'annuler la tension sur toute ligne qui entre en contact avec elle. Elle peut donc couper l'alimentation de l'un quelconque des segments du visuel, et l'empêcher de s'allumer. L'exclusion sélective des segments permet d'obtenir sur un visuel à sept segments les chiffres compris entre 0 et 9.

C'est la mémoire ROM qui élimine du visuel les segments appropriés, pour chacune des opérations du décodeur. Par exemple, si on a un 0 sur la première ligne de sortie du décodeur (chiffre décimal 0), il faut déconnecter le segment horizontal au centre du visuel, ce qui fera apparaître le 0.

Une mémoire ROM très simple, capable d'accomplir certaines fonctions nécessaires, se construit par étapes successives. Il faut d'abord composer une table qui indique quels sont les segments du visuel qu'il convient d'allumer, pour chacune des combinaisons réalisées avec les quatre bits susceptibles de se présenter en entrée.

Ensuite, on met en place les connexions sur une matrice à diodes, comme on le voit sur la figure. Si on insère une diode entre la première ligne de sortie du décodeur et la ligne d'alimentation G, chaque fois qu'il se produit une absence de tension sur la première ligne, l'alimentation du segment central du visuel est interrompue. Les diodes qui relient la ligne d'alimentation de A à F à la ligne 0 du décodeur deviennent inutilisables. Elles sont délibérément fondues, éliminées de la matrice, au cours de l'élaboration du programme de la ROM, grâce à un courant intense sous une haute tension. On peut donc, par cette méthode simple de la matrice à diodes, programmer une mémoire ROM, qui remplira une fonction donnée.

Un visuel numérique est donc commandé par une unité de sortie (interface) qui reçoit de l'ordinateur central la donnée à faire apparaître. Elle la décode et fournit en sortie les impulsions nécessaires pour activer le visuel. L'unité d'affichage à diodes électroluminescentes (LED) peut faire apparaître les chiffres de 0 à 9 en déclenchant les combinaisons appropriées des sept segments qui la composent. Si l'on utilise des cristaux liquides, il faut également sept segments pour former le chiffre 8 ; en activant les segments appropriés, on peut former les chiffres de 0 à 9.

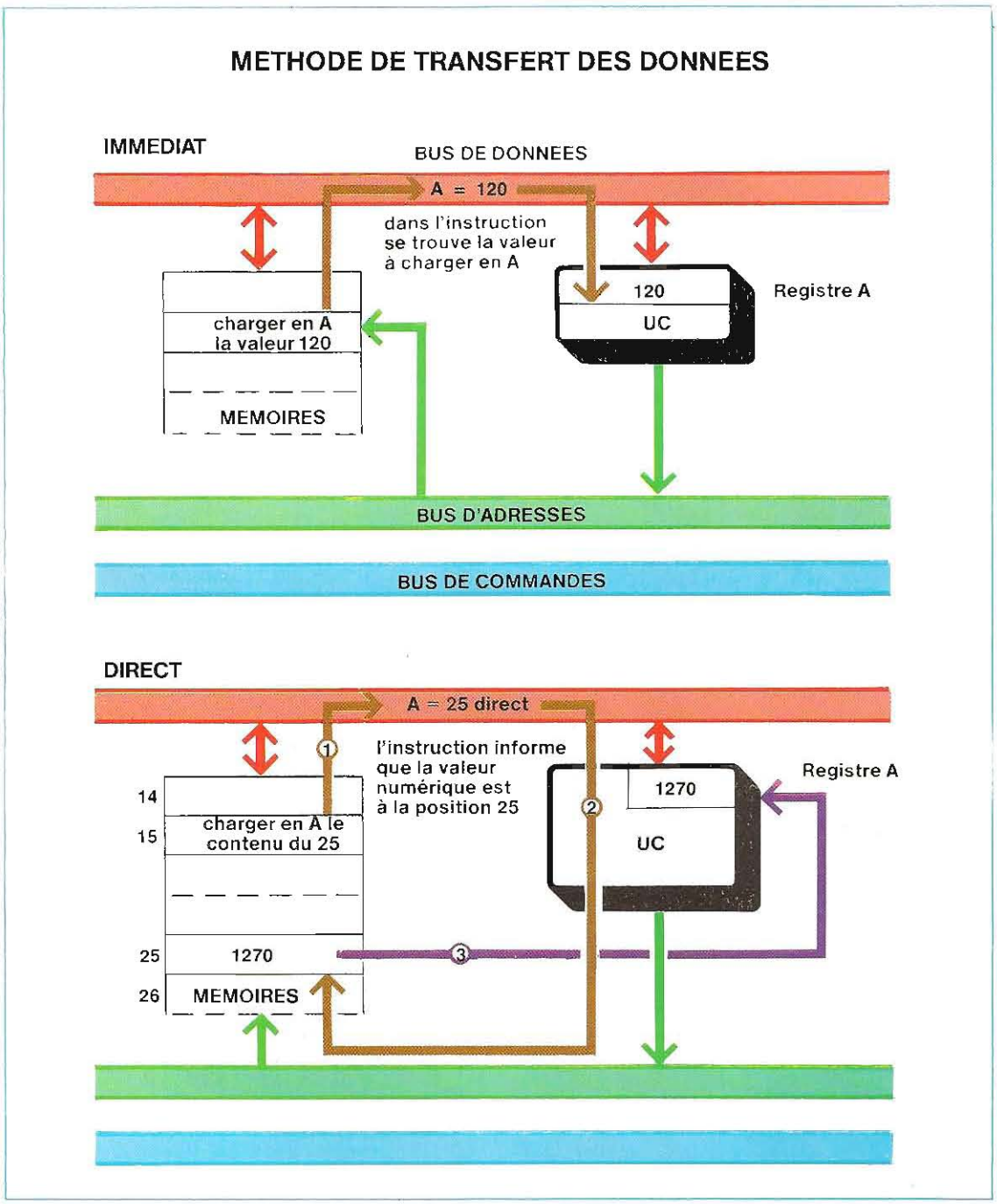
Cette visualisation se fonde sur les propriétés de cristaux nématiques (du grec nêmatos : fil) en suspension dans un liquide emprisonné dans une plaque transparente placée à l'avant, et une plaque réfléchissante placée à l'arrière. Si l'on applique une différence de potentiel entre les segments de la plaque antérieure et ceux de la plaque postérieure, le liquide devient opaque et, grâce à une altération des indices de réflection, produit un segment visible.

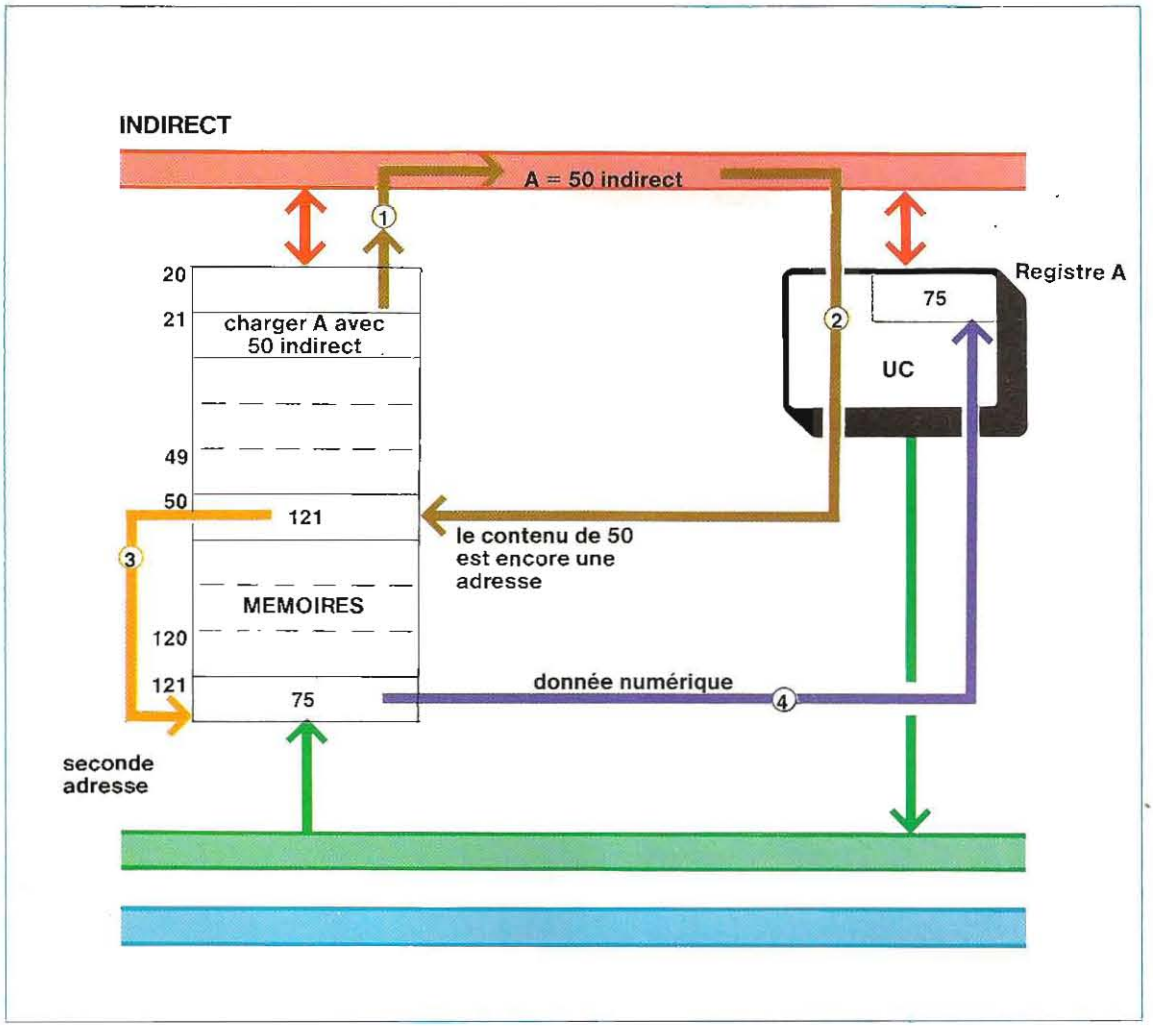
Les instructions dans l'UC

Le fonctionnement d'un système à microprocesseur consiste à exécuter la série d'instructions constituant le programme. Dans l'unité centrale, une instruction est interprétée et convertie en signaux électriques, par l'intermédiaire de microcodes.
Une unité centrale interprète les cinq types d'instructions suivants :

1 / transfert de données
2 / opérations arithmétiques et logiques
3 / instructions de saut
4 / fonctions d'E/S (Entrée/Sortie)
5 / instructions spéciales

Selon le type de l'unité centrale, chacun des





cinq groupes se compose d'une quantité plus ou moins grande d'instructions plus ou moins complexes.

Instructions de transfert des données

Ces instructions de mouvement des données sont celles qui permettent d'extraire une donnée et de l'adresser en mémoire.
Dans les microprocesseurs (μP), la transmission se fait toujours entre la mémoire du système et une mémoire particulière, contenue dans l'unité centrale, et appelée registre (voir schémas). Ces instructions sont :
immédiates : dans le registre désigné, on introduit une valeur numérique donnée. Par exemple, la valeur 120 dans le registre A ;
directes : on charge le registre du contenu d'une mémoire spécifique. Le registre A reçoit, par exemple, la valeur numérique contenue dans la mémoire 25. A l'inverse de ce qui se passe avec les instructions immédiates, le registre n'est pas chargé d'une donnée spécifique (immédiate) mais de tout le contenu de la mémoire 25 ;
indirectes : c'est une extension du type précédent. Le registre est chargé de deux adresses successives : la première produit la seconde, la seconde produit la donnée. Par exemple : chargement du registre A de 50 indirect. L'instruction se déroule de la façon suivante :
① - ② extraction du contenu de la mémoire 50 ;
③ ce contenu, par exemple 121, est interprété comme une nouvelle adresse ;
④ extraction de la donnée contenue dans la mémoire 121.
Dans ces exemples, nous n'avons considéré que des transferts de la mémoire au registre, mais l'inverse est possible.

Opérations arithmétiques et logiques

Ces opérations sont réunies dans un même groupe. Pour mener à bien une opération, il faut en général fournir les adresses où rechercher les données et l'adresse où déposer le résultat. La séquence logique complète des instructions à fournir comprend trois étapes :

① introduire dans le registre A la première donnée (par exemple le contenu de la mémoire 26) ;

② exécuter l'opération arithmétique (par ex. la somme) entre le contenu de A (première donnée) et celui de la mémoire concernée (par ex. 41) ; le résultat est en A ;

③ transférer le contenu de A (résultat) dans la mémoire désignée (par ex. 53).

Instructions de saut

Ce groupe d'instructions permet de sauter d'un point à l'autre du programme au moment de son exécution par l'UC. On distingue :

- **les instructions de saut proprement dites (Jump en anglais),**
- **les instructions de saut avec retour (Return en anglais).**

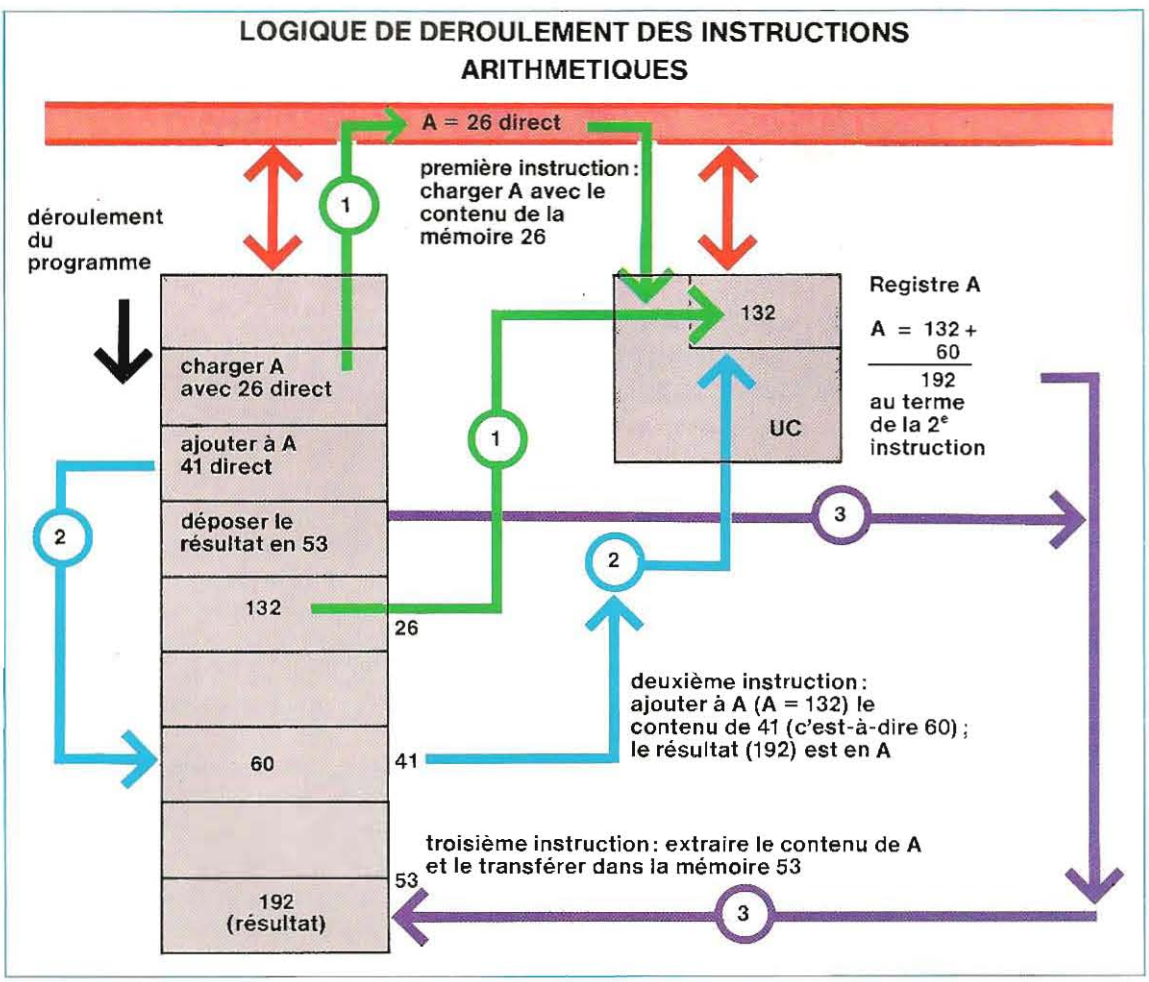
Dans le premier cas (schéma en haut de la p. 149), l'opération se poursuit du point d'arrivée du saut jusqu'à la fin du programme. Dans le second, lorsqu'on arrive à l'instruction Retour, l'opération reprend à l'instruction qui suit celle de saut (schéma en bas de la p. 149).

Fonctions E/S (Entrée/Sortie)

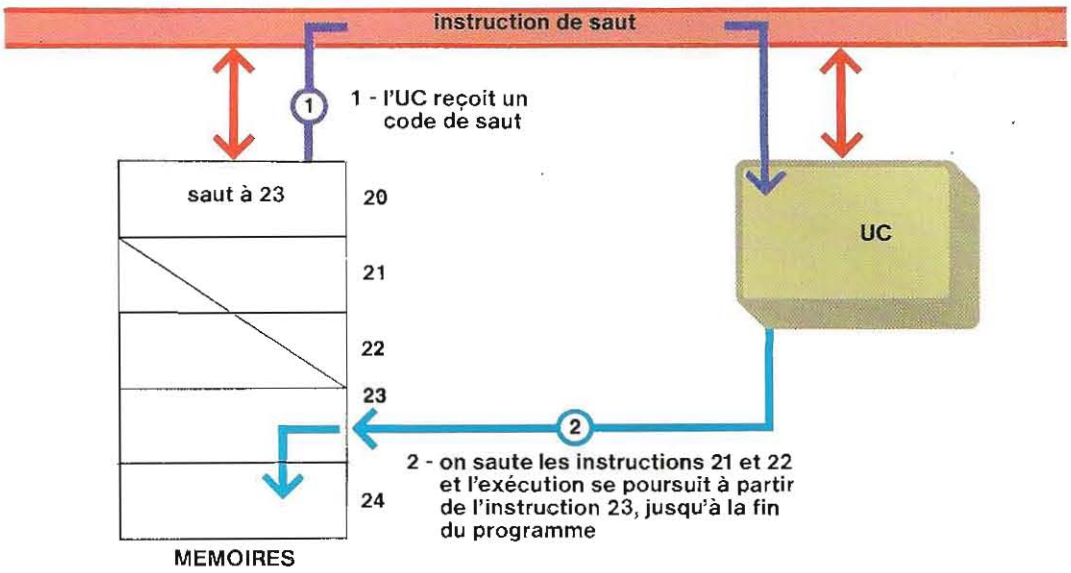
C'est dans ce groupe qu'on trouve toutes les instructions qui permettent de lire ou d'écrire une donnée au moyen d'un périphérique.

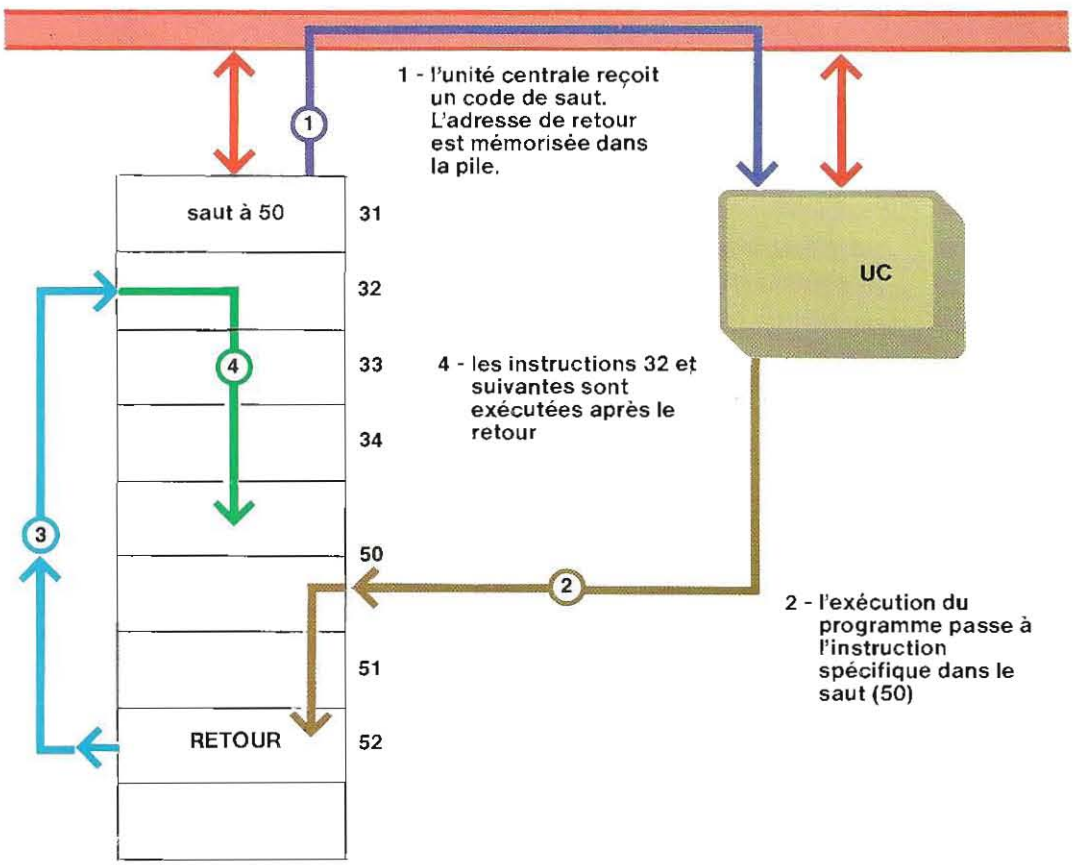
Instructions spéciales

C'est un groupe d'instructions particulières, souvent propres à des unités centrales déterminées. Les principales concernent la fonction d'interruption (anglais Interrupt).



LOGIQUE DE DEROULEMENT DES INSTRUCTIONS DE SAUT





Solutions du test 3

1 / Conversion en binaire sous forme chiffrée :

Décimal	Binaire	
7	**0111**	**$(2^2+2^1+2^0=4+2+1)$**
9	**1001**	**$(2^3+2^0=8+1)$**
0	**0000**	**0**
3	**0011**	**$(2^1+2^0=2+1)$**

Conversion en binaire sous forme de caractères (symboles ASCII) :

Décimal	ASCII	
7	37 hexadécimal	= 55 décimal
9	39 hexadécimal	= 57 décimal
0	30 hexadécimal	= 48 décimal
3	33 hexadécimal	= 51 décimal

La première forme (numérique) est utilisée dans les calculs, la seconde dans les opérations d'E/S et de mémorisation.
Le programmeur doit préciser, au moyen d'instructions appropriées, quelle forme il a adoptée pour éviter toute erreur.

2 / La transmission en série consiste à envoyer les données, une à la suite de l'autre (bit par bit). Dans la transmission parallèle, on lance un bloc entier d'informations à la fois (par exemple un caractère complet). Ce dernier mode est certes plus rapide, mais exige des lignes à conducteurs multiples.

3 / La codification du caractère K en ASCII est 4B (hexadécimal) = 75 (décimal) = 1001011 binaire. Les symboles 1 sont au nombre de quatre, le bit de parité est donc égal à 1. Pour compléter le message, il faut encore un bit de départ (Start = 0) et 2 bits d'arrêt (ou Fin, Stop = 1). La forme complète est donc :

0	1	0	0	1	0	1	1	1	1	1
START	←			Lettre K			→	Parité	STOP	

4 / a) Saut de page = FF = 12 décimal
b) Caractères X, Y = 88, 89 décimaux
c) Saut de ligne = LF = 10 décimal
d) x, y (minuscules) = 120, 121 décimaux
e) Saut de page = FF = 12 décimal

Le message complet (en décimal) est le suivant :

12	**88**	**89**	**10**	**120**	**121**	**12**
FF	**X**	**Y**	**LF**	**x**	**y**	**FF**
commande	caracteres		commande	caractères		commande

Si on ne précisait pas qu'il s'agit de caractères ASCII, le message serait erroné puisque ces données, interprétées comme nombres et non comme symboles ASCII, n'ont aucune signification pour l'imprimante.

---

5 / Les lettres majuscules ont pour codes des nombres allant de 65 à 90 (décimaux), les minuscules, des nombres allant de 97 à 122. Il suffit d'ajouter 32 (décimal).

65 +	(valeur décimale de A)
32	transformation
97	(valeur décimale de a)

De la même façon :

z (=122) = Z (90) + 32

---

6 / Pour extraire (au moyen de ET) un bit quelconque, il faut avoir recours à un masque contenant le symbole 1 dans la position à extraire.
Le masque requis est :

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Masque	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**

En décimal, ce masque vaut : $2^7 + 2^3 + 2^1 = 128 + 8 + 2 = 138$.
Pour la seconde partie, il faut tout d'abord convertir les nombres 12, 21, 6, 15 en binaire (voir listing 1 p. 63).

12 =	0	0	0	0	1	1	0	0
21 =	0	0	0	1	0	1	0	1
6 =	0	0	0	0	0	1	1	0
15 =	0	0	0	0	1	1	1	1

Si l'on applique l'opérateur ET entre les nombres et le masque, on obtient pour le nombre 12 :

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Nombre 12 =	0	0	0	0	1	1	0	0
Masque =	1	0	0	0	1	0	1	0
12 ET masque	0	0	0	0	1	0	0	0

De façon analogue :

21 ET masque = 00000000
6 ET masque = 00000010
15 ET masque = 00001010

Test 4

1 / Quelle fonction le compteur d'instructions remplit-il ?
a) Il contient le nombre des instructions exécutées.
b) Il contient l'adresse de la mémoire d'où sera extraite la prochaine instruction à exécuter.
c) C'est une mémoire auxiliaire qui sert à effectuer une somme.

---

2 / Dans les tableaux de mémoires ci-dessous, quelle est l'instruction qui sera exécutée (dans les deux cas) après l'instruction 106 ?

	18
sauter à 103	19
	20
	102
	103
	104
	105
RETOUR	106
	107

a

	18
sauter à 103	19
	20
	102
	103
	104
	105
	106
	107

b

---

3 / Dans quel élément de l'Unité Centrale les calculs arithmétiques sont-ils exécutés ?
a) Dans la pile.
b) Dans l'U A L.
c) Dans le compteur d'instructions.

---

4 / Qu'est-ce qu'un système d'exploitation ?
a) Un ensemble de programmes remplissant des fonctions de base telles que la gestion des périphériques.
b) Les programmes qui servent à effectuer des calculs.
c) Les éléments spéciaux qui commandent l'échange des données dans le DMA.

---

5 / Qu'est-ce que le DMA ?
a) Pour l'utilisateur, l'accès direct à la mémoire.
b) La modification des contenus de la mémoire, opérée par l'Unité Centrale.
c) Un échange de données sans passer par l'Unité Centrale.

---

6 / Dans la figure suivante, on voit un tableau de mémoire. Le programme commence à la position 31. Dites s'il y a une erreur, et, si c'est le cas, laquelle :

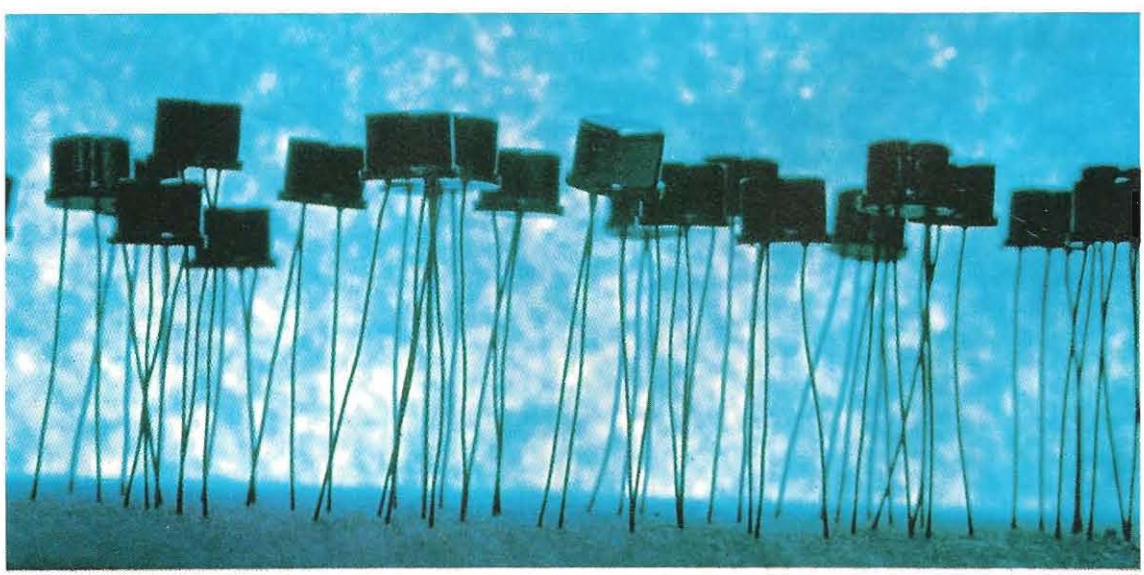
Ajouter 12 au contenu de A	31
Sauter à 55	32
RETOUR	33
	55
	56
RETOUR	57
	58
	59

7 / Laquelle de ces définitions de « Interruption » (Interrupt) est exacte ?
a) Interruption du fonctionnement de l'Unité Centrale.
b) Interruption du programme à la suite d'une demande extérieure.
c) Interruption des échanges avec la mémoire.

8 / Parmi les affirmations suivantes, lesquelles sont exactes ?
a) Une mémoire EPROM ne peut pas contenir de programme.
b) L'unité de disque contient environ 64 000 caractères.
c) Une mémoire ROM ne peut pas être écrite à partir du programme.
d) A l'arrivée d'une Interruption, l'unité centrale se place en position Attente (Hold).
e) L'afficheur et le clavier ne peuvent pas déclencher une Interruption.
f) L'unité disque, normalement, échange des données dans le DMA.

Les solutions du test se touvent à la page 183.

Dans les systèmes numériques, les transistors sont le plus souvent remplacés par des circuits intégrés.



Marka

La Programmation

Ecrire un programme consiste à traduire toutes les opérations à effectuer dans un langage compréhensible pour la machine.
Lorsque nous manipulons une calculatrice courante, nous donnons nos instructions en appuyant sur les touches. Les circuits internes ainsi activés répondent aux ordres transmis par les touches. Les signaux-résultats, émis par ces circuits, sont traduits et viennent s'afficher sur l'écran de la calculatrice.
Dans les machines programmables (ordinateurs), on ne commande pas les opérations une par une. Les fonctions sont chargées en mémoire, et sont exécutées à la mise en route du programme sans autre intervention de l'opérateur. Une machine programmable est ainsi capable d'effectuer des tâches beaucoup plus complexes qu'un simple calcul arithmétique. Sa principale caractéristique est sa capacité de traiter une grande quantité de données. L'ordinateur doit être capable d'acquérir (de lire) ces données, de les mémoriser, de les traiter, d'effectuer sur ces données les recherches souhaitées par l'utilisateur. Pour déclencher ce traitement informatique, il faut un programme, ou série d'instructions, indiquant à la machine quelles fonctions exécuter, et dans quel ordre.
Lorsqu'on recherche, par exemple, le nom de Martin dans un répertoire, on doit prévoir les trois fonctions suivantes :

1 / lire un nom figurant dans la liste ;
2 / si ce nom est Martin, inscrire O.K., afficher Martin et les données associées, puis terminer le programme ;
3 / si le nom n'est pas Martin, revenir à l'étape 1, et lire le nom suivant.

Mais la machine n'est pas en mesure de comprendre des phrases formulées en langage naturel. Pour mettre en mémoire de telles instructions, il faut d'abord les traduire, leur donner une forme susceptible d'être interprétée et convertie en signaux électriques.
La forme d'expression la plus accessible à la machine est le langage binaire. De nombreuses solutions existent pour mettre en corrélation la démarche de l'esprit humain et le fonctionnement de l'ordinateur. Ce sont les outils de la programmation. Ils sont écrits en langages artificiels mais évolués.

Les langages de programmation

Le choix de tel ou tel langage est déterminé par le type de problème à résoudre : gestion d'une entreprise, gestion de production, calculs scientifiques, etc. Il peut arriver, cependant, que le choix du langage de programmation n'ait qu'une incidence marginale : il s'agit de gagner du temps, d'économiser sur le nombre d'instructions à donner à la machine. Des circonstances extérieures, ou des préférences personnelles, guideront alors la décision du programmeur.
Parfois, au contraire, un problème donné ne peut se résoudre qu'en employant un langage donné (c'est le cas du Fortran pour les applications mathématiques). Tous les langages de programmation sont symboliques. Ils expriment, dans une formulation compréhensible par l'homme et par la machine, les tâches que la machine doit accomplir. La traduction à l'usage de la machine est assurée par des programmes particuliers intermédiaires : les interpréteurs et les compilateurs.
Outre les langages symboliques, il y a des langages « absolus », où les instructions prennent directement la forme binaire. En général, on n'emploie pas ces langages machine pour rédiger des programmes d'application (ce serait trop fastidieux), mais uniquement pour certaines applications.

Langages compilés et interprétés

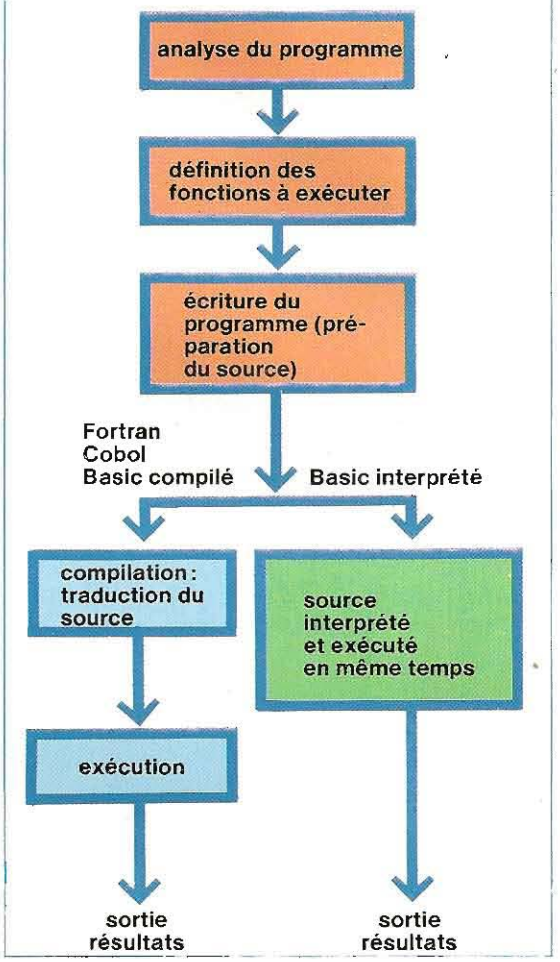
Ces langages (dits de haut niveau) s'adressent au programmeur. Toutes les instructions y sont codées et décrivent l'action à accomplir : par exemple Print (écrire) lorsqu'on veut faire apparaître un mot, ou une formule, sur l'écran, etc. L'ordinateur ne peut pas exécuter directement ces ordres, puisque l'unité centrale ne reconnaît qu'un nombre limité d'instructions fondamentales (voir « Les instructions dans l'UC », p. 146). Il est donc nécessaire de traduire la commande écrite en langage de haut niveau en une série d'instructions élémentaires. Cette conversion se fait au moyen de deux catégories de programmes : les compilateurs et les interpréteurs.

Les compilateurs travaillent sur le programme d'application global, et en fournissent la traduction complète. Les interpréteurs procèdent instruction par instruction, ce qui permet de détecter immédiatement une erreur éventuelle.
Les trois langages de haut niveau les plus utilisés dans les micro-ordinateurs sont :

■ **le Basic** ■ **le Fortran** ■ **le Cobol**

D'autres langages, également employés, seront présentés dans le chapitre consacré aux systèmes d'exploitation.
Le Fortran est conçu pour un usage scientifique, le Cobol pour des utilisations dans le domaine de la gestion et de l'administration.
Le Basic est le langage le plus souple : il se prête – bien que d'une manière moins puissante – aux deux emplois.
Le Fortran et le Cobol sont toujours compilés ; le Basic existe sous les deux formes, Basic compilé et Basic interprété.
Un programme d'application, quel que soit le langage dans lequel il est écrit, prend le nom de **programme source** (on dit aussi le source).
Le schéma ci-contre illustre les différentes phases du processus de préparation et d'exécution d'un programme d'application, avec un langage compilé ou interprété.
Dans les langages interprétés (Basic interprété), chaque instruction du programme source est traduite (interprétée) au moment même de son exécution. La traduction des instructions se fait en même temps que s'effectue le programme d'application. L'exécution est donc plus longue. Mais l'avantage est que, dès l'introduction du programme source, on obtient un diagnostic. Le programme source est introduit, de façon codée, sur le clavier, et c'est le système d'exploitation qui le charge en mémoire. Au cours de cette phase, si des erreurs se révèlent, l'interpréteur les signale, ce qui permet de les corriger immédiatement. Des erreurs banales telles que les fautes de frappe sont immédiatement repérées et corrigées. En revanche, dans les langages compilés, les erreurs n'apparaissent qu'au moment de l'exécution du programme et il faut attendre la fin du processus de compilation.
Résumons les caractéristiques de ces deux types de langage.

SCHEMA LOGIQUE DU PROCESSUS D'ECRITURE ET D'EXECUTION D'UN PROGRAMME



Langage interprété : possibilité de correction immédiate. Exécution plus lente.
Langage compilé : chaque correction implique une nouvelle compilation, donc une attente. En revanche, l'exécution est rapide.
L'idéal est de disposer d'un langage qui puisse être aussi bien interprété que compilé. L'interpréteur est donc très utile lors de la rédaction du programme d'application.
La phase d'essai avec compilation intervient seulement après vérification que le programme d'application est correct.
Grâce à sa souplesse, le Basic se prête bien à cette méthode : on rédige en interprété et on teste en compilé : c'est pourquoi c'est le langage le plus utilisé dans les micro-ordinateurs.
A l'origine, le Basic avait été conçu pour les

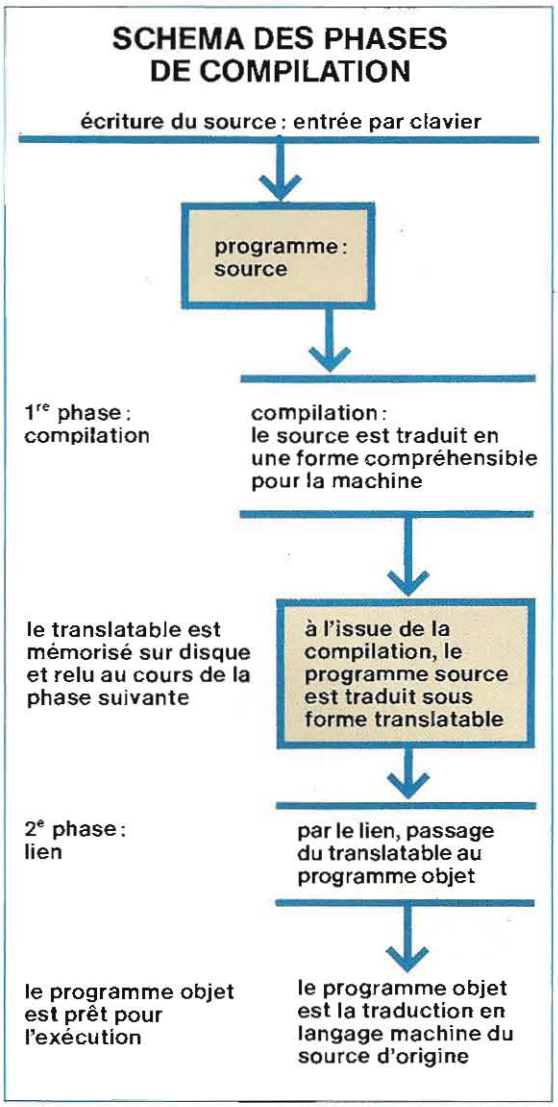
débutants et il reste le plus adapté pour une initiation à la programmation.

Compilation

La compilation d'un programme source se fait en deux étapes successives.

- La première étape aboutit à la production d'une forme intermédiaire qui prend le nom de **translatable**.
- La seconde étape consiste à relier entre elles les différentes parties du translatable. Cette opération s'appelle le **lien** (en anglais link). Au terme de cette opération, on obtient le **programme objet**, qui sera chargé en mémoire avant d'être exécuté.

Habituellement, dans les systèmes comportant un compilateur, il faut prévoir des unités disque, chargées de mémoriser le translatable. Dans les langages interprètes, les unités disque ne sont pas indispensables, puisque l'interpréteur peut résider en mémoire de façon permanente.



Basic

Ce langage est adapté à la fois aux usages scientifiques et de gestion puisque, outre les caractéristiques du Fortran, il a aussi la capacité de traiter des textes et différentes instructions destinées à l'impression de rapports économiques. Le traitement des données sur disque (fichiers) est de moindre puissance que celui du Cobol, mais il convient tout à fait à la structure des micro-ordinateurs.

Le Basic est plus récent que les autres langages, et les formes sous lesquelles il a été adopté ne sont pas encore unifiées. Mais les différences (mineures) qui existent d'un micro-ordinateur à l'autre sont destinées à disparaître.

Fortran

C'est un langage à l'usage des scientifiques. Voici quelles en sont les caractéristiques principales :

- nombre restreint d'instructions fondamentales ;
- peu de texte à traiter ;
- des données normalement sous forme de nombres exponentiels ;
- relativement peu de données à traiter, mais une exploitation très poussée de ces données ;
- nécessité d'instruments de calcul particuliers (algorithmes mathématiques).

L'emploi de ce langage en gestion n'est pas exclu, mais il se révèle lourd et moins maniable, surtout par manque d'instructions permettant d'élaborer des caractères alphanumériques.

Cobol

C'est un langage destiné exclusivement à des applications de gestion. Ses caractéristiques principales sont les suivantes :

- pas d'instructions de calcul (sauf les opérations principales) ;
- possibilité de bonne gestion des données sur disque (archives) ;
- instructions pour l'édition de rapports de caractère économique.

Analyse et organigrammes

La rédaction d'un programme n'est que la phase ultime, l'aboutissement d'un travail complexe d'analyse d'un problème donné, qu'il faut ensuite traduire sous forme de structures synthétiques compréhensibles pour la machine.

L'analyse du problème consiste à décomposer les opérations à réaliser en une série de fonctions élémentaires, auxquelles correspondent des symboles graphiques. A partir de cette représentation graphique, **l'organigramme**, ou **ordinogramme** (en anglais flowchart), on passe à l'écriture proprement dite des instructions. Nous allons tenter d'illustrer les principales méthodes d'analyse et les techniques de préparation des organigrammes.

Etablissement et écriture d'un programme

Pour chaque application spécifique, il faut fournir à l'ordinateur le programme adéquat. Lorsque l'application est complexe, on peut recourir à des sous-programmes, dont l'ensemble constitue une **procédure.**

Par exemple, dans la gestion d'un commerce, on distinguera trois étapes fondamentales :

- **entrée des marchandises**
- **sortie des marchandises**
- **inventaire des stocks.**

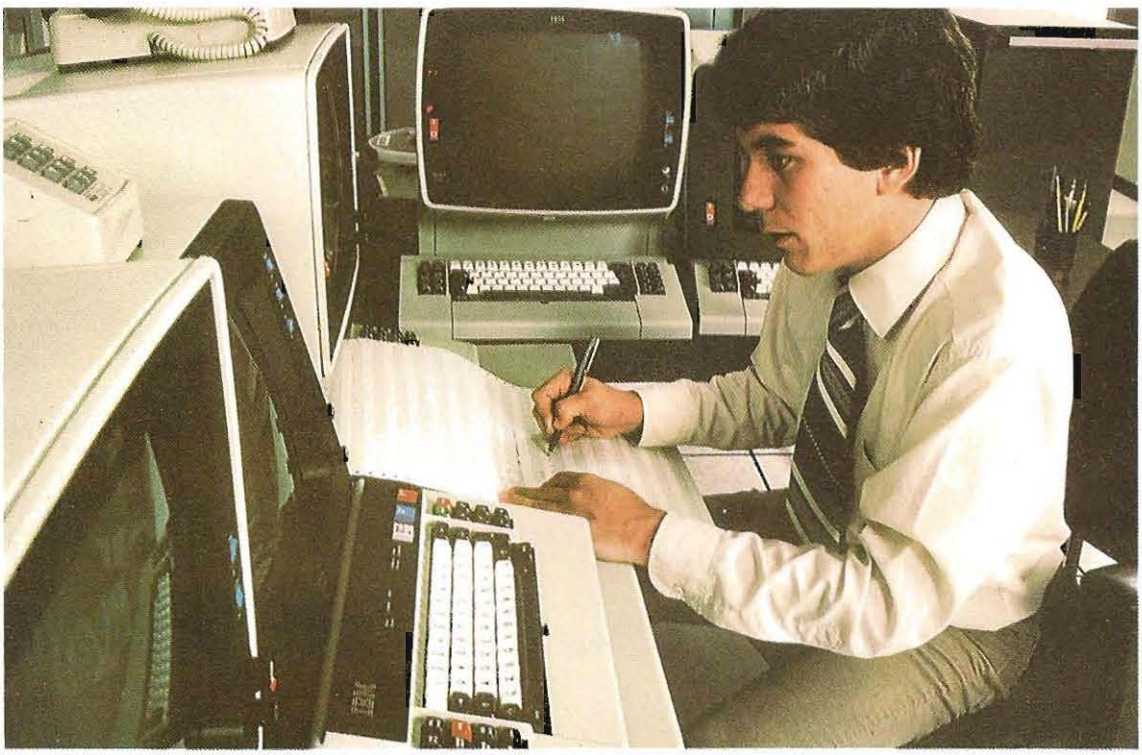
A chacune de ces parties correspondra un programme, l'ensemble constituant la procédure de gestion du magasin.

Avant de commencer l'écriture d'un programme, il faut connaître les différents aspects du problème et définir une méthode. Cette phase du travail est la plus délicate, car c'est d'elle que dépend toute la qualité du programme.

L'établissement d'un programme peut se diviser en trois phases :

- **analyse du problème**
- **synthèse**
- **mise au point des organigrammes.**

Dans les compagnies d'assurances, la capacité de mémorisation et la rapidité de recherche de l'ordinateur permettent un traitement rapide des dossiers.



Marka

Analyse

Un programme, quel qu'il soit, ne peut traiter que des données préalablement fournies à la machine. Dans l'exemple précédent (inventaire des stocks), le programme ne donnera de résultats corrects que si l'on a introduit tous les mouvements (entrée et sortie) des marchandises. Ces mouvements doivent en outre porter sur des articles connus de la machine.
La première phase de l'analyse consiste à définir les résultats recherchés et à vérifier que l'on dispose bien de toutes les données nécessaires.
Supposons par exemple que nous voulions obtenir l'inventaire des stocks sous forme d'une liste contenant les informations suivantes :

1 / code de l'article
2 / description
3 / quantité en stock
4 / prix unitaire
5 / valeur globale du stock.

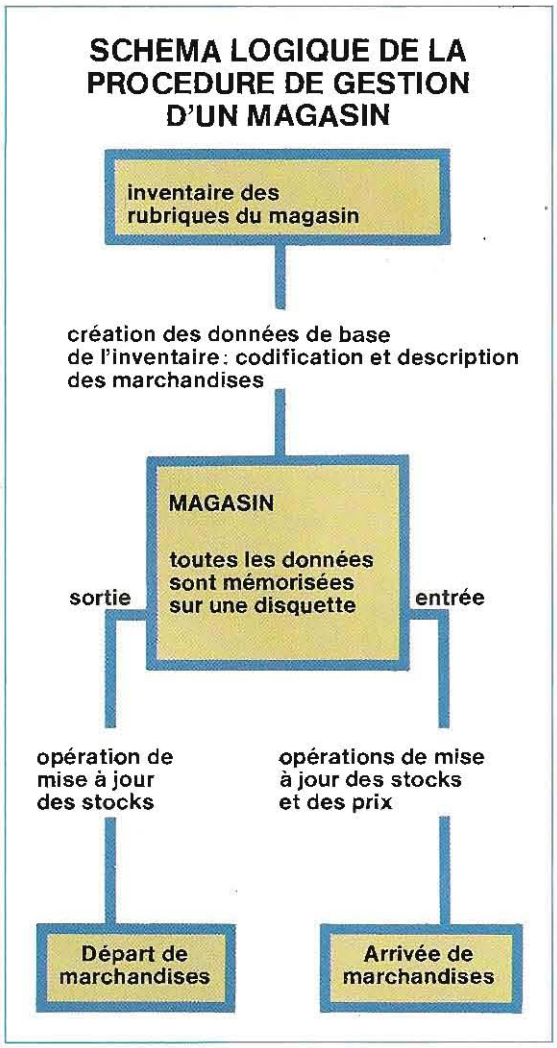
En analysant les résultats recherchés, on définit les différents programmes qui constitueront la procédure (schéma à droite). Dans notre exemple, les premières données à écrire sont :

1 / code de l'article
2 / description.

Il faut d'abord fournir ces données à la machine. Il faut donc, en premier lieu, un programme pour créer les rubriques correspondant aux articles du magasin.
La demande suivante (quantité en stock) est celle du chiffre représentant la quantité existant en magasin. Ce chiffre variera en fonction de deux opérations :

- départ d'un article
- arrivée d'un article.

Le programme doit, à chaque sortie et à chaque entrée de marchandise, effectuer la soustraction et l'addition correspondantes, c'est-à-dire mettre à jour l'état des stocks.
Le quatrième point (prix unitaire) est une donnée à fournir au moment de l'entrée de la marchandise. Ainsi, pour les points 3 et 4, il faut



deux programmes :

- sortie de marchandises
- entrée de marchandises.

Enfin, le point 5 requiert un programme qui, utilisant les résultats des précédentes opérations, effectue un calcul final. On peut résumer les résultats de l'analyse en un tableau des principales opérations à exécuter :

1 / Constitution des rubriques (codification et description des articles)
2 / Mouvements d'entrée (état des stocks) et nouveaux prix
3 / Mouvements de sortie (état des stocks)
4 / Impression des listings des stocks (calcul de la valeur totale).

Tableau des programmes et des données nécessaires :

PROGRAMMES	DONNEES
Création des rubriques	Codification des articles et description
Entrées	Volume du stock et prix unitaire
Sorties	Volume du stock
Impression des listings	Valeur totale

Les fonctions principales de chacun des programmes peuvent se résumer ainsi :

Création des rubriques

L'utilisateur introduit, pour chaque article existant dans le magasin, un code permettant d'identifier l'objet, et sa description.

Entrées

Il s'agit d'ajouter au stock chaque nouvelle arrivée de marchandise, et de mettre à jour les prix. Les opérations à effectuer sont :
quantité actuelle = quantité précédente + quantité arrivée ;
prix = prix de l'article arrivé.

Sorties

Il suffit ici de mettre à jour l'état du stock après livraison au client, soit : quantité actuelle = quantité précédente − quantité prélevée.

Impression du listing

Avant d'imprimer le listing d'état des stocks, le programme doit calculer, pour chaque article, la valeur totale de ce qui est en magasin, soit :
valeur totale = stock x prix unitaire,
soit : Rubrique 5 = Rubrique 3 x Rubrique 4.
Les opérations sont illustrées par les schémas des pages 160 et 161.

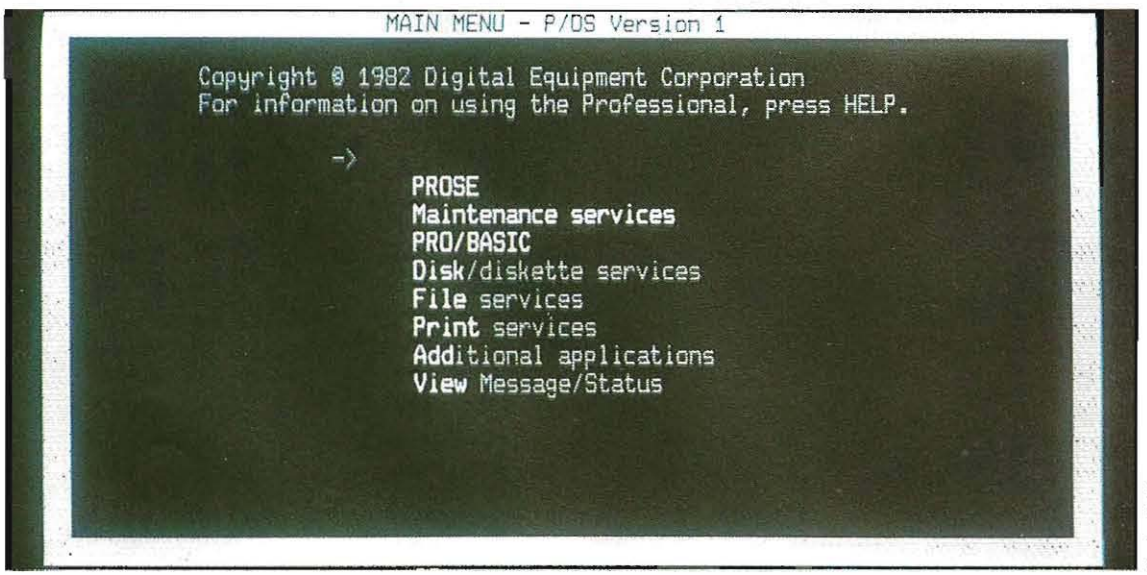
Les données relatives à chaque rubrique doivent être mémorisées par l'ordinateur. Pour cela on utilisera une mémoire de masse, par exemple une disquette. Et on choisira le type de machine en fonction de la capacité de mémoire nécessaire, de la vitesse requise, du nombre d'utilisateurs appelés à travailler simultanément sur la machine. Le choix définitif se fera après une analyse précise de l'utilisation qu'on veut en faire et se portera sur :

Ordinateur domestique. Pour un seul utilisateur à la fois. Peu de données à mémoriser. Pas d'exigence de rapidité. Coût minimal. Peu, ou pas de possibilité d'extension.

Micro-ordinateur. Un seul utilisateur, éventuellement plusieurs. Données assez importantes. Possibilité d'obtenir une bonne rapidité de traitement. Coût moyen. Assez bonnes possibilités d'extension.

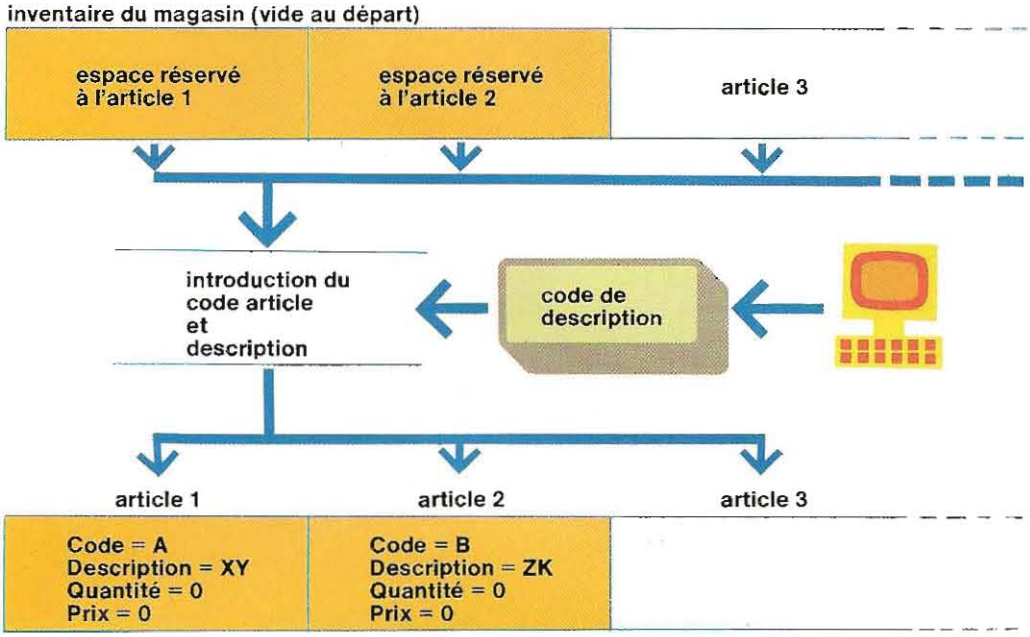
Mini-ordinateur. Possibilité d'utilisateurs multiples, simultanément et éventuellement avec des programmes différents. Quantité de données selon les besoins. Rapidité moyenne ou élevée. Coût élevé. Possibilités d'extension très grandes.

Informations visualisées sur l'écran d'un ordinateur.



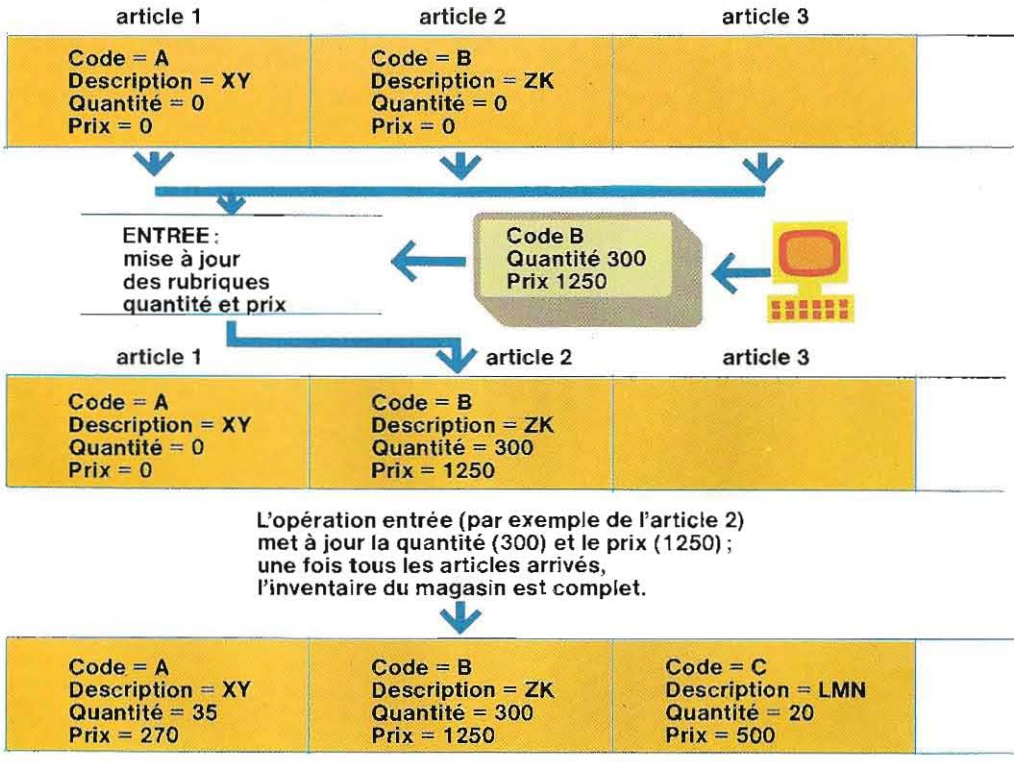
Centre THC

CREATION DES RUBRIQUES DU MAGASIN



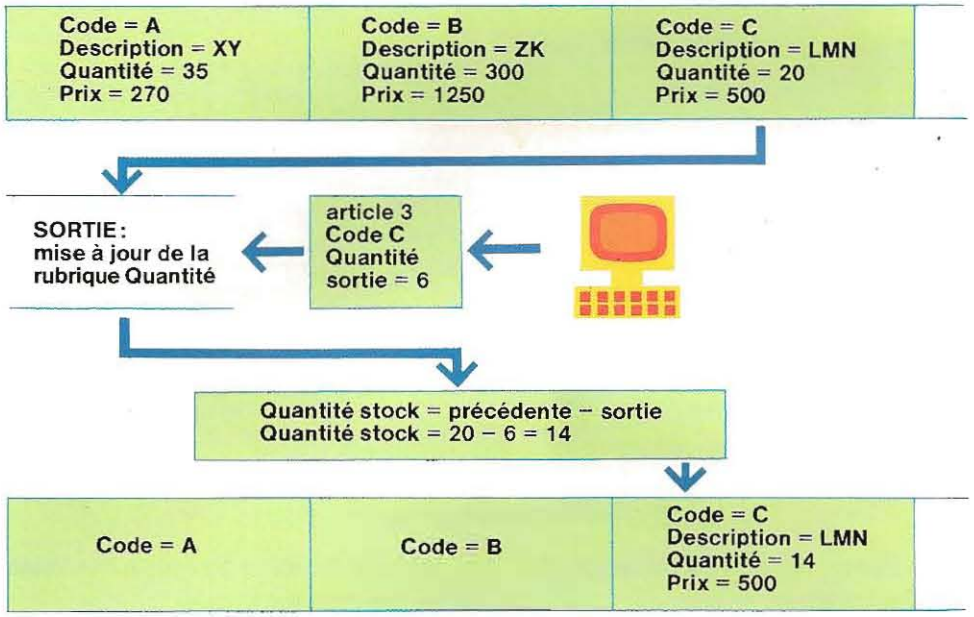
On a introduit dans l'inventaire du magasin les codes et les descriptions de chaque article.
Quantités et prix sont égaux à 0 puisque le chargement n'est pas encore effectué.

OPERATIONS ENTREE DE MARCHANDISES



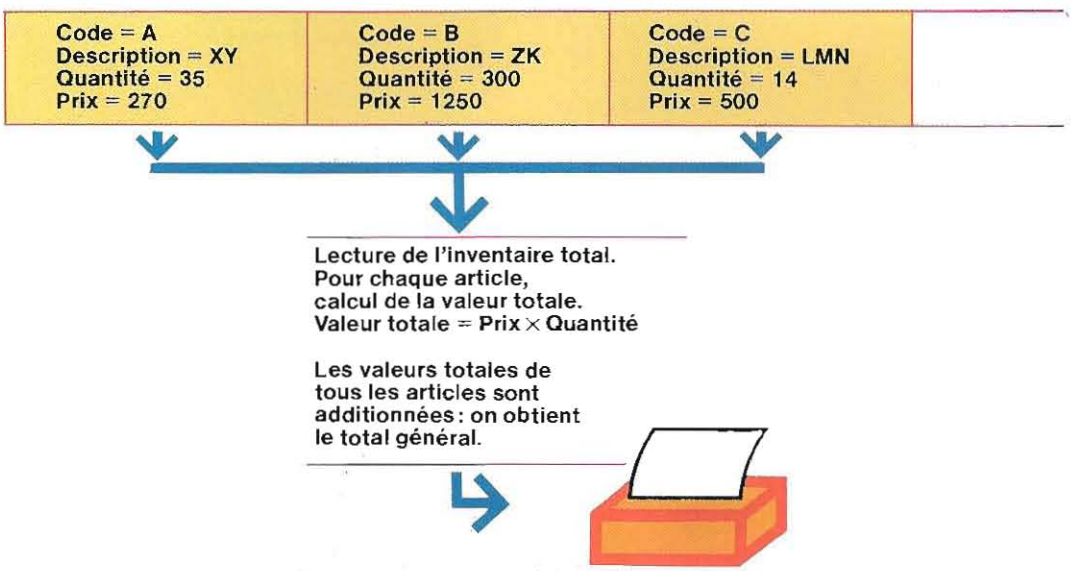
L'opération entrée (par exemple de l'article 2) met à jour la quantité (300) et le prix (1250) ; une fois tous les articles arrivés, l'inventaire du magasin est complet.

OPERATIONS SORTIE DE MARCHANDISES



L'opération de sortie (prélèvement sur stock) soustrait la quantité prélevée de la quantité stock précédente et met à jour les données dans l'inventaire.

EDITION LISTINGS DU STOCK



Code	Description	Quantité	Prix	Valeur totale
A	XY	35	270	9450
B	ZK	300	1250	375000
C	LMN	14	500	7000
			Total général	391450

Le choix de base se fera en fonction des besoins prévisibles.

Les objectifs de la phase d'analyse sont :

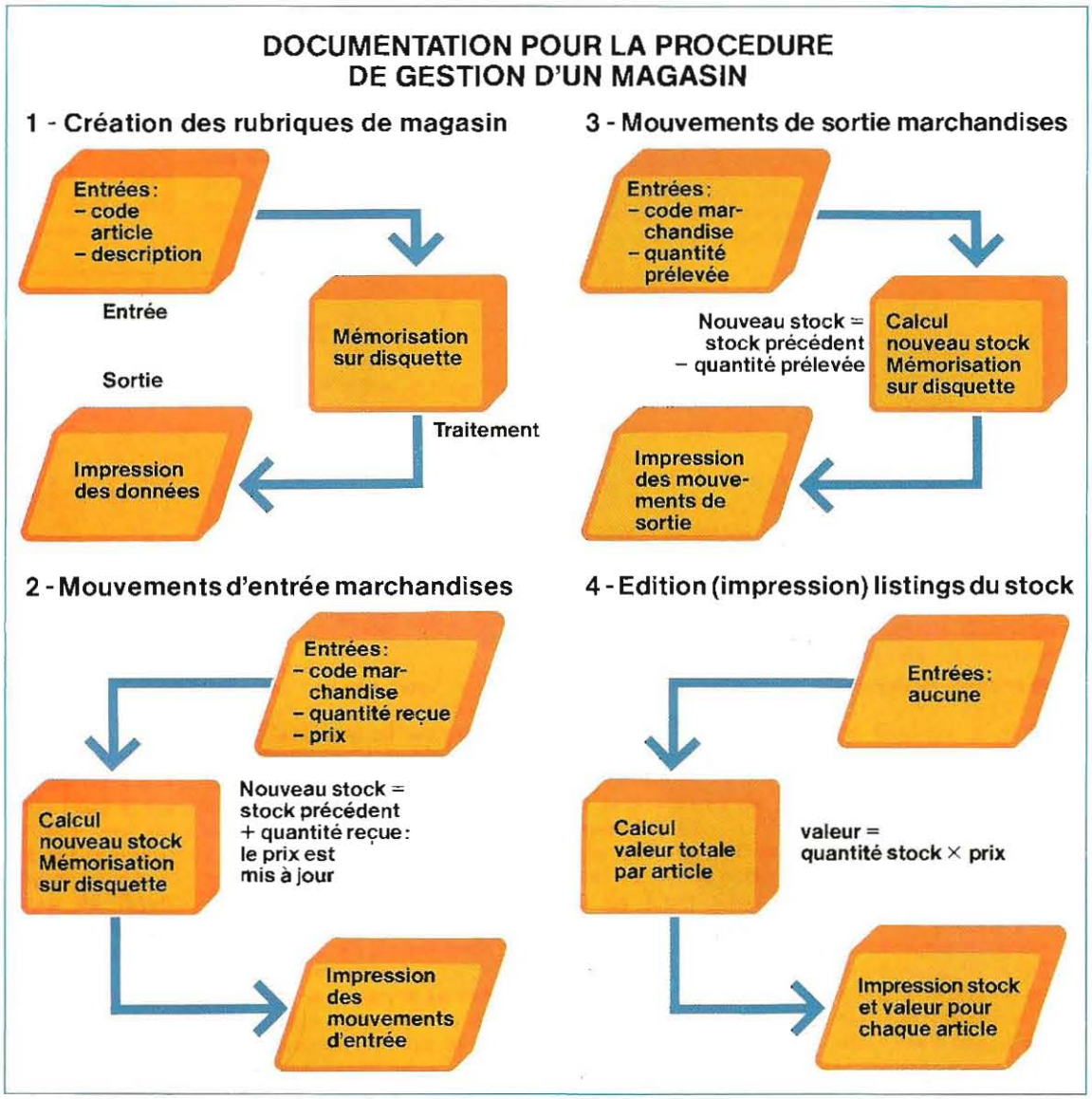
1 / contrôler l'existence de toutes les données d'entrée,
2 / définir les éventuels programmes annexes,
3 / choisir le type de machine.

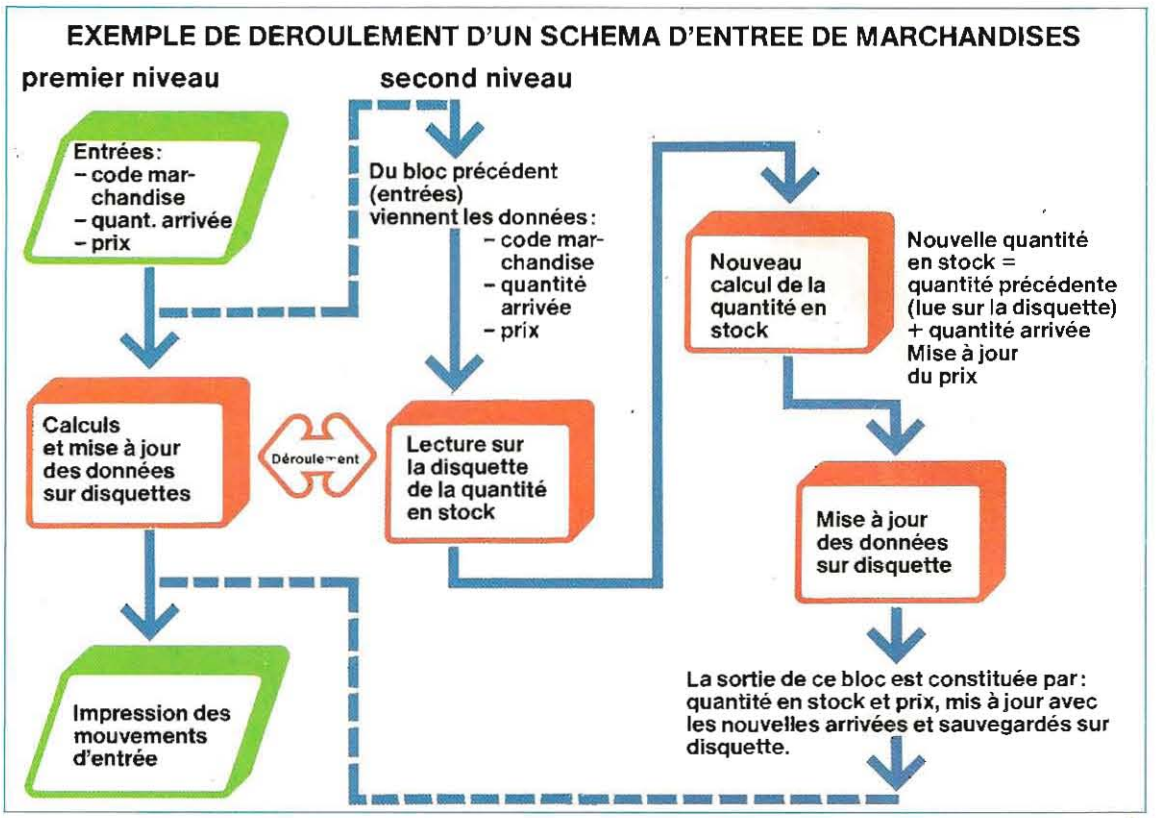
Synthèse

La synthèse consiste à préparer un ou plusieurs schémas qui montrent de façon synthétique les fonctions d'un programme, ou des programmes d'une procédure. Le plus simple est d'utiliser la technique HIPO (Hierarchy plus Input-Process Output).
Cette technique consiste à dresser pour chaque programme (ou partie de programme) un schéma en trois blocs.

Premier bloc : entrée. Contient la description de toutes les entrées nécessaires à l'établissement du programme.

Deuxième bloc : traitement. Contient la des-





cription de tous les calculs et, en général, de toutes les opérations à effectuer sur les données.

Troisième bloc: sortie. Contient la description de toutes les sorties prévues.

Dans les cas les plus simples, un seul organigramme HIPO peut suffire. Pour des programmes plus complexes, il en faudra plusieurs, plus ou moins détaillés. La technique HIPO est non seulement un auxiliaire précieux dans le déroulement du travail, mais fournit également une bonne documentation sur l'application (dossier de programmation).
Le schéma de la page 162 présente des organigrammes HIPO appliqués à une procédure de gestion de magasin.
Ensuite on pourra développer les blocs les plus complexes. Par exemple, dans la partie Mouvements d'entrée, on peut développer le bloc traitement de la façon suivante:

a / lecture sur disque, pour chaque code marchandise, du stock existant;
b / addition au stock ancien de la nouvelle quantité arrivée;
c / remplacement de l'ancien prix unitaire par le nouveau;
d / remplacement sur le disque des données anciennes par les nouvelles.

Chacun des traitements énumérés a trait à un article déterminé; la donnée d'entrée est le code de l'article.
Au terme du traitement, on a sur disque les données mises à jour: c'est la sortie, le résultat du traitement.
Le schéma ci-dessus représente l'état initial et le développement (deuxième état). Avec ce deuxième niveau d'information, nous n'avons pas encore suffisamment de détails pour procéder à l'écriture d'un programme. Il faut passer à un niveau supérieur de description et préparer un schéma qui illustre, à l'aide de symboles, toutes les opérations à accomplir.
Dans le schéma en haut de cette page, il y a lieu de prévoir une vérification des données (voir par exemple si on n'a pas essayé d'introduire un code marchandise qui n'existe pas).
A la sortie sur imprimante, il faut prévoir quelques lignes pour en-tête, etc.
Le schéma des opérations à effectuer, à ce niveau de détail, prend le nom d'organigramme. C'est l'ultime étape préparatoire avant l'écriture du programme proprement dite.

Des machines qui savent lire

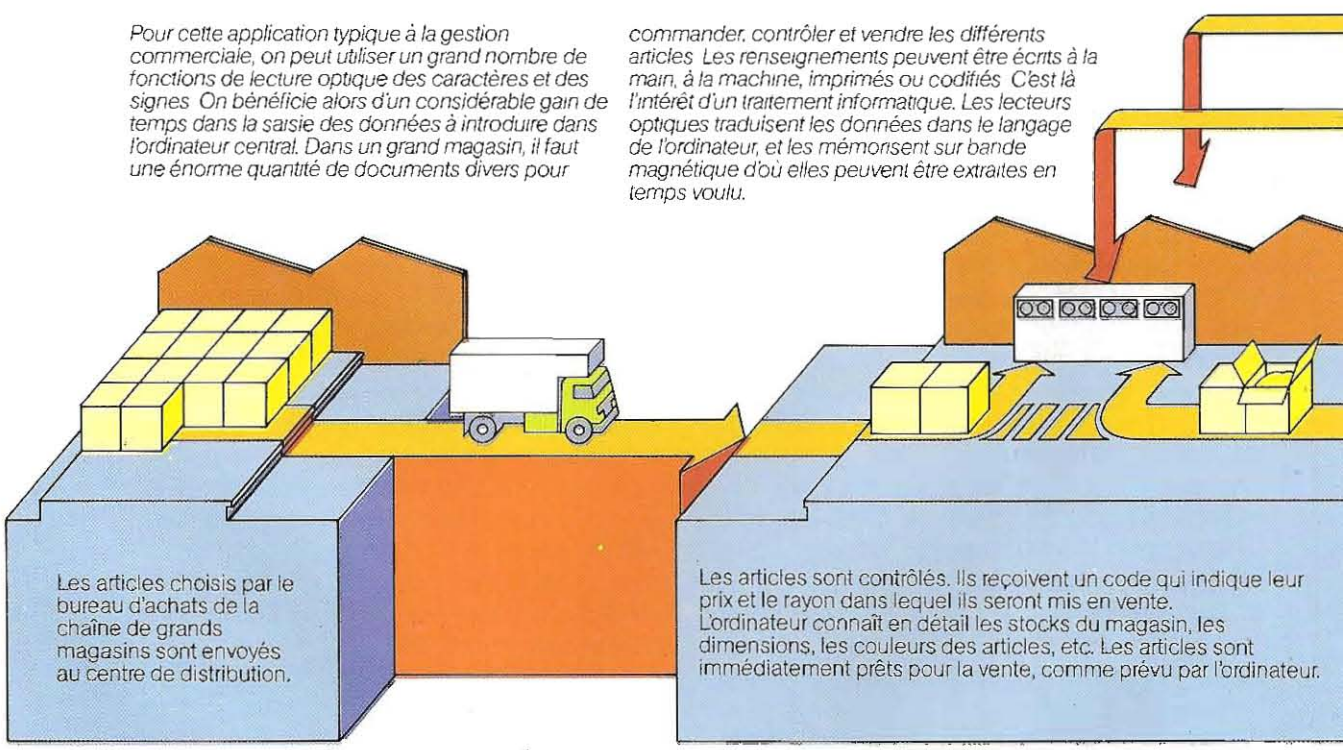
Si l'on considère le rôle joué par les ordinateurs dans notre vie quotidienne, on comprend que pour d'innombrables applications on a besoin de machines capables de lire automatiquement.

Chaque année, plusieurs milliards de chèques bancaires sont rédigés. Il faut les classer de telle manière que chaque chèque puisse être échangé avec l'agence émettrice ou avec une chambre compensatrice.
Seules les machines peuvent effectuer cet énorme travail. Elles lisent en effet les caractères imprimés à l'encre magnétique au bord inférieur du chèque et opèrent une rapide sélection automatique.

Dans certains pays, le volume du courrier est encore assez modeste et le tri peut se faire manuellement ou avec des équipements semi-automatiques : c'est un employé qui lit l'adresse et compose sur un clavier le code postal ; une machine classe ensuite la lettre automatiquement.

Au Japon et aux Etats-Unis, le volume du courrier dépasse les capacités en main-d'œuvre disponible. On emploie des machines lectrices appelées des lecteurs. Ces machines rejettent environ 8 % des lettres parce qu'elles ne réussissent pas à lire l'adresse sur l'enveloppe.
Ces matériels ne se bornent pas à classer des documents, elles peuvent aussi introduire des données dans les ordinateurs. Par exemple, si l'on veut tenir à jour la liste des noms et des adresses d'un million de personnes ou d'établissements, il faut enregistrer environ 25 000 modifications par semaine. On peut fournir les adresses à un ordinateur, par l'intermédiaire de la console. Mais on peut aussi écrire les adresses sur une simple feuille de papier, à la machine à écrire, et communiquer cette feuille à l'ordinateur.
Dans plusieurs pays d'Europe, les Postes utilisent déjà des machines capables de lire les caractères imprimés. Le rythme moyen de ces machines est de 1 000 caractères par seconde.
La Bank of America en possède une capable de lire 40 000 caractères par

Pour cette application typique à la gestion commerciale, on peut utiliser un grand nombre de fonctions de lecture optique des caractères et des signes. On bénéficie alors d'un considérable gain de temps dans la saisie des données à introduire dans l'ordinateur central. Dans un grand magasin, il faut une énorme quantité de documents divers pour commander, contrôler et vendre les différents articles. Les renseignements peuvent être écrits à la main, à la machine, imprimés ou codifiés. C'est là l'intérêt d'un traitement informatique. Les lecteurs optiques traduisent les données dans le langage de l'ordinateur, et les mémorisent sur bande magnétique d'où elles peuvent être extraites en temps voulu.



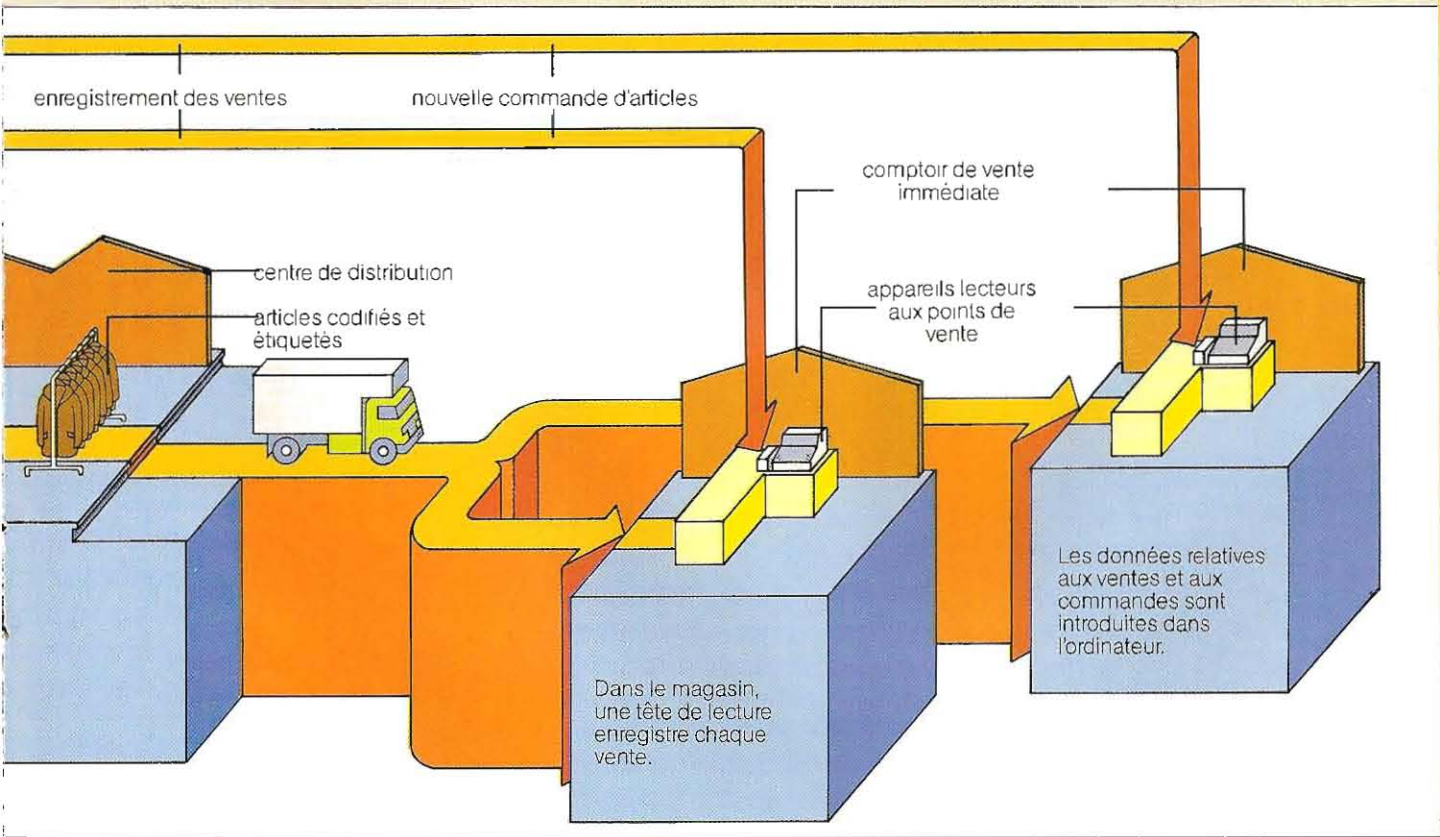
seconde. Mais ce sont là des équipements compliqués et très coûteux.

On utilise les lecteurs électroniques également pour introduire des textes dans des machines de traitement de texte. Un dactylographe compose le texte sur une machine appropriée, l'auteur le revoit et le corrige s'il y a lieu. Ensuite, une machine lectrice explore le texte et le transmet à la machine de traitement de texte, qui permet à l'opérateur d'apporter très rapidement les modifications nécessaires.
Malheureusement, ces lecteurs peu coûteux utilisés avec les processeurs de texte sont peu fiables lorsqu'il est nécessaire de lire un texte qui n'est pas écrit en caractères spéciaux, comme par exemple une correspondance portant des corrections. Il faut une écriture spéciale et des intervalles nets entre les caractères.

Les compagnies européennes du gaz et de l'électricité utilisent des lecteurs de factures qui font immédiatement apparaître les arriérés de paiement.

Pour acquérir une machine ultra-rapide, capable de lire lettres, chiffres et ponctuation, il faut investir des sommes importantes. Cependant, le marché propose des machines à des prix très différents et on peut, par exemple, faire des économies considérables si l'on demande à une machine de lire, non pas des caractères, mais des séquences codifiées de barres parallèles plus ou moins épaisses et espacées.
Ces codes à barres, déchiffrables même par des appareils portatifs, sont d'un emploi désormais courant dans les bibliothèques et les magasins. Les appareils de lecture optique pour ce type de code sont beaucoup plus simples et économiques que ceux auxquels on demande de déchiffrer des caractères d'écriture normale.
Par rapport aux hommes, les machines tolèrent moins les différences de forme et de qualité des caractères, mais elles sont infiniment plus rapides.

Le prix d'un lecteur dépend de la gamme de caractères qu'il aura à lire, et de sa rapidité. Les progrès de la micro-électronique



ont permis de réduire davantage le coût de la partie électronique de l'appareil que celui de sa partie mécanique (chargée par exemple de la transmission des cartes). Beaucoup de machines pourtant récentes, mais peu perfectionnées sur le plan mécanique, ne peuvent lire qu'une dizaine de caractères par seconde. Une solution particulièrement économique consiste à faire passer une tête lectrice sur la ligne de texte à lire.
Si l'encre, le papier et le type de caractères correspondent exactement aux normes établies par le fabricant, une bonne machine est capable de lire chiffres, lettres (majuscules et minuscules) et ponctuation avec une très faible marge d'erreur. Une erreur typique est la confusion entre deux caractères (par exemple 5 et S), mais le taux normal d'erreur est de moins de 1 pour 100 000. Autre inconvénient possible : la machine signale (environ 1 fois sur 10 000) qu'elle n'a pas pu reconnaître un certain signe.

Par ailleurs, si une machine est destinée à lire seulement des chiffres, il n'est pas nécessaire d'avoir une aussi bonne qualité d'écriture. Avant de reconnaître la forme d'un caractère, un lecteur optique explore le texte au moyen d'une série de cellules photo-électriques, ou d'un dispositif laser ou, plus rarement, d'une télécaméra. Après analyse, le caractère est représenté par la machine comme une matrice de bits.
Dans le diagramme (reproduit ci-dessous) qui représente le nombre 222, le chiffre 1 correspond au noir et le 0 au blanc. L'homme comprend immédiatement que le code lu correspond au chiffre 2, mais, pour arriver à la même conclusion, la machine a besoin de faire des calculs.

```
0000000000  0000000000  0000000000
0011111000  0011111000  0011111000
0111111100  0111111100  0111111100
0110001110  0110001110  0110001110
0000000110  0000000110  0000000110
0000000110  0000000110  0000000110
0000001100  0000001100  0000001100
0000011000  0000011000  0000011000
0000110000  0000110000  0000110000
0011100000  0011100000  0011100000
0111111110  0111111110  0111111110
0111111110  0111111110  0111111110
0000000000  0000000000  0000000000
0000000000  0000000000  0000000000
```

Une méthode consiste à lui faire comparer cette matrice de 130 bits à d'autres matrices de 130 bits, précédemment mémorisées par la machine et pourvues chacune de son propre code. Lorsqu'on lui présente les deux modèles, l'appareil compte le nombre de bits qui diffèrent. La reconnaissance consiste à trouver le code du modèle qui possède le moins de bits discordants. Ce schéma est bien sûr simplifié ici ; en réalité, il est beaucoup plus complexe. Et cependant, il n'est pas encore assez efficace. Il peut en effet facilement fournir un code inexact à cause de caractères déformés, mal imprimés, empâtés etc., ou d'une panne dans le mécanisme d'exploitation électronique.

Pour remédier à ces inconvénients, on a mis au point une autre technique de reconnaissance, fondée sur l'emploi de relations logiques, qui permet de reconstituer un bit défectueux.
Si l'on considère par exemple la matrice de bits suivante, dans laquelle certains bits-clés ont été représentés par des lettres de l'alphabet,

```
0000000000  0000000000  0000000000
000aabb000  000aabb000  000aabb000
00aeeffb00  00aeeffb00  00aeeffb00
0ae0000fb0  0ae0000fb0  0ae0000fb0
0ae0ii0fb0  0ae0ii0fb0  0ae0ii0fb0
0ae0ii0fb0  0ae0ii0fb0  0ae0ii0fb0
0ae0000fb0  0ae0000fb0  0ae0000fb0
0cg0000hd0  0cg0000hd0  0cg0000hd0
0cg0jj0hd0  0cg0jj0hd0  0cg0jj0hd0
0cg0jj0hd0  0cg0jj0hd0  0cg0jj0hd0
0cg0000hd0  0cg0000hd0  0cg0000hd0
00cgghhd00  00cgghhd00  00cgghhd00
000cc0d000  000cc0d000  000cc0d000
0000000000  0000000000  0000000000
```

le caractère prédifini sera reconnu comme 0 si :
au moins cinq **a** sont **1**
ou
au moins quatre **e** sont **1** ;



Un micro-ordinateur est placé dans les caisses enregistreuses. Ces machines contiennent un système miniature avec unité centrale, mémoire et interface E/S.

et
au moins cinq **b** sont **1**
ou
au moins quatre **f** sont **1** ;
et
au moins cinq **c** sont **1**
ou
au moins quatre **g** sont **1** ;
et
au moins cinq **d** sont **1**
ou
au moins quatre **h** sont **1** ;
et
au moins trois **i** sont **0** ;
et
au moins trois **j** sont **0**.

Certaines des machines les plus coûteuses sont capables de reconnaître les majuscules écrites à la main en lettres d'imprimerie, à condition que l'espace entre les lettres soit respecté.
Avant de reconnaître un caractère écrit à la main, la machine (lecteur) doit déterminer sur le document une zone qui ne contient que ce caractère. Il faut donc qu'elle puisse l'isoler des caractères proches. D'où la nécessité de l'espacement.
La prolifération des ordinateurs à base de microprocesseur a provoqué une explosion de demandes de machines destinées à la conversion de données numériques en représentation codifiée à l'usage de l'ordinateur lui-même.
C'est ce marché qui pousse à la construction de machines lectrices.
Mais on constate que, vers la fin des années 1970, le nombre de brevets accordés pour ces appareils a fortement diminué.
Nombre de firmes américaines et japonaises ont parfois perdu beaucoup d'argent à construire des machines qui se sont finalement révélées invendables parce que trop chères.
Il semblerait pourtant qu'ampleur du marché et prix tendent progressivement à s'équilibrer, grâce aux progrès continuels de l'électronique.

Etablissement des organigrammes

L'élaboration d'un organigramme est la représentation d'un programme par une succession de symboles graphiques. L'ensemble des diagrammes qui en résulte est l'organigramme du programme. Chaque fonction accomplie par un ordinateur a en effet son symbole graphique propre. De ce fait, lorsque l'organigramme a été correctement établi, l'écriture du programme se réduit à un travail de traduction des symboles en instructions, telles qu'elles existent dans le langage de programmation choisi.
Les symboles graphiques utilisables peuvent être répartis en trois groupes fondamentaux. Ces groupes sont ceux des

- symboles liés aux opérations de l'ordinateur (calculs, décisions, etc.) ;
- symboles des opérations d'entrée et de sortie des données ;
- symboles liés à la mémorisation.

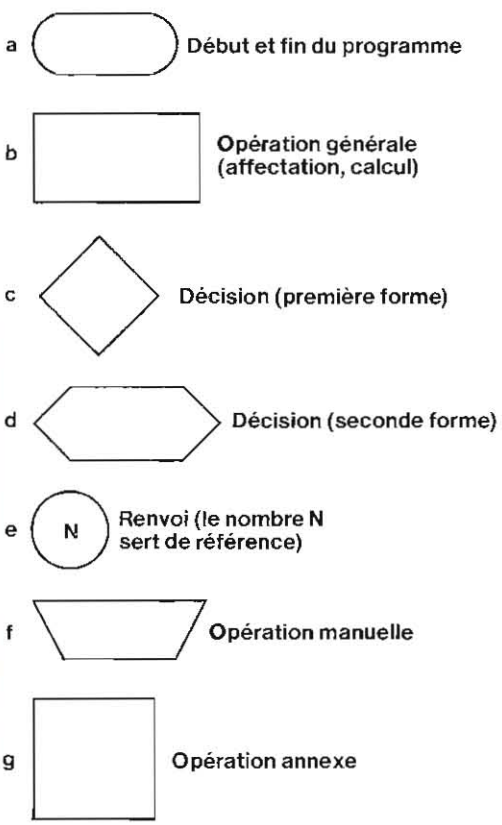
Dans le schéma de cette page, on trouvera les symboles les plus couramment utilisés. Nous les décrirons plus en détail par la suite. Nous décrirons plus sommairement ceux qui sont peu utilisés et qui ne nous serviront pas dans nos exemples d'applications. Mais il faut savoir que dans les applications de programmation sur micro-ordinateurs, la gamme des symboles utilisés est beaucoup plus restreinte.

Symboles d'usage courant

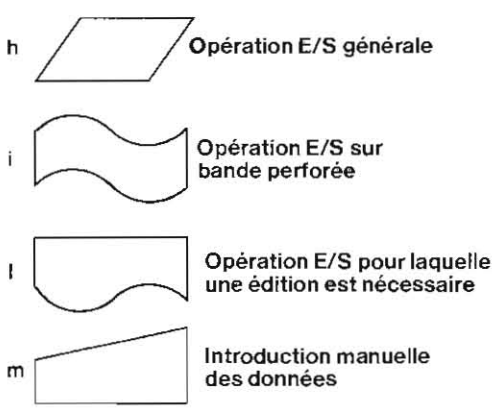
Ce sont les symboles de traitement puisque les opérations de traitement constituent l'essentiel d'un programme.
Le symbole **a** indique le début et la fin d'un programme. Tout programme, aussi complexe soit-il, revêt la forme générale qui apparaît dans le schéma de la page 169 (en haut). Les flèches indiquent le sens suivi par le traitement.

SYMBOLIQUE D'ORGANIGRAMME

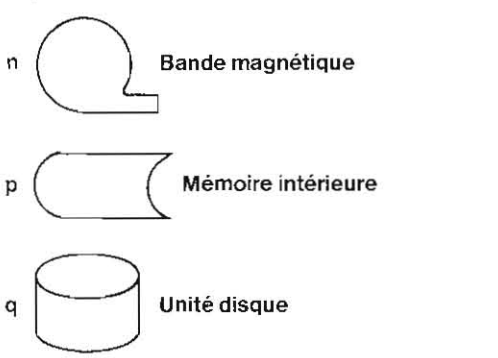
Symboles de traitement



Symboles pour les opérations



Symboles de mémorisation





Le symbole **b**, accompagné du commentaire nécessaire, indique un traitement général, par exemple un calcul. Dans la procédure de gestion d'un magasin que nous venons d'étudier, le coût total est obtenu en faisant le produit de la quantité (nombre d'articles) par le coût unitaire : il faudra indiquer cette opération par le symbole **b** de la façon suivante :

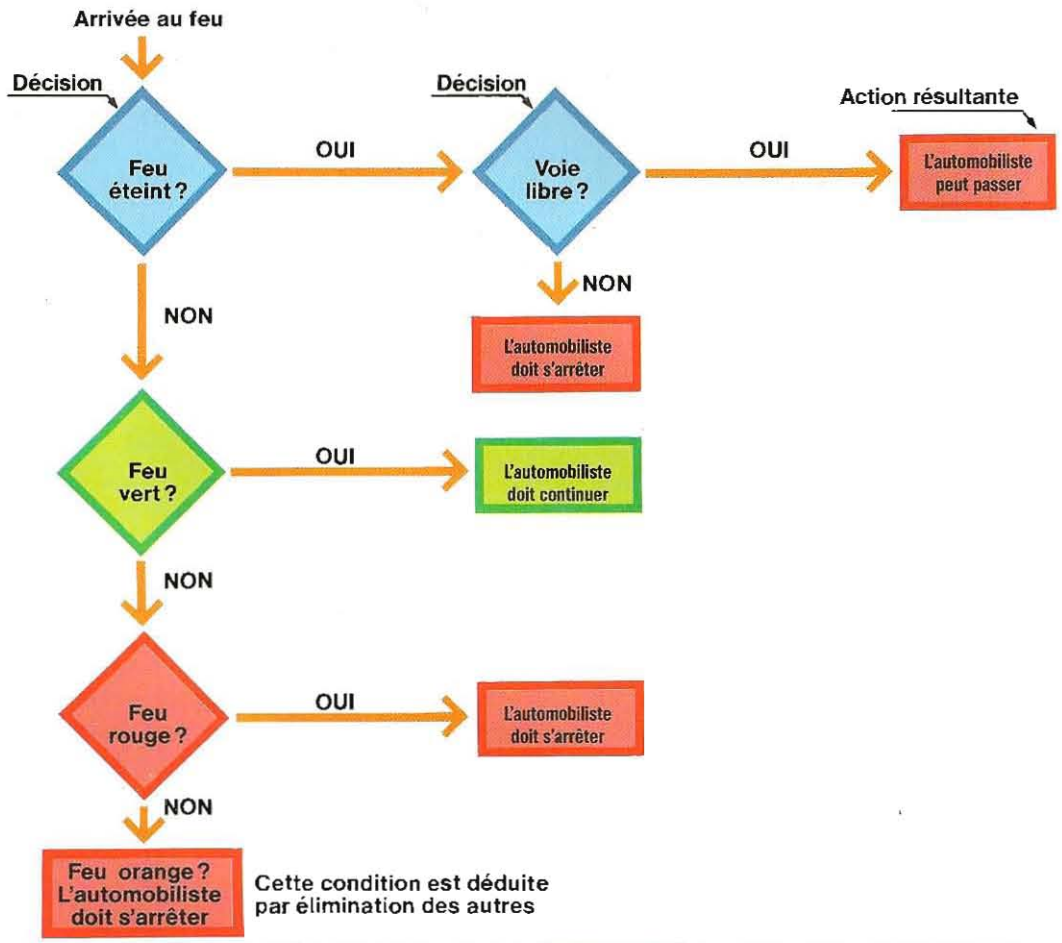
Coût total =
Quantité ×
Coût unitaire

Les symboles **c** et **d** sont équivalents et expriment une décision (appelée aussi test).
On choisit l'un ou l'autre de ces symboles selon la longueur du commentaire que l'on veut y introduire. La décision se décompose en deux phases.

- Première phase : comparaison entre deux (ou plusieurs) valeurs ;
- Seconde phase : choix de la marche à suivre en fonction du résultat de la comparaison.

Prenons l'exemple d'un automobiliste qui arrive à un feu de signalisation. On peut sché-

SCHEMA LOGIQUE DU COMPORTEMENT D'UN AUTOMOBILISTE A UN FEU DE CIRCULATION

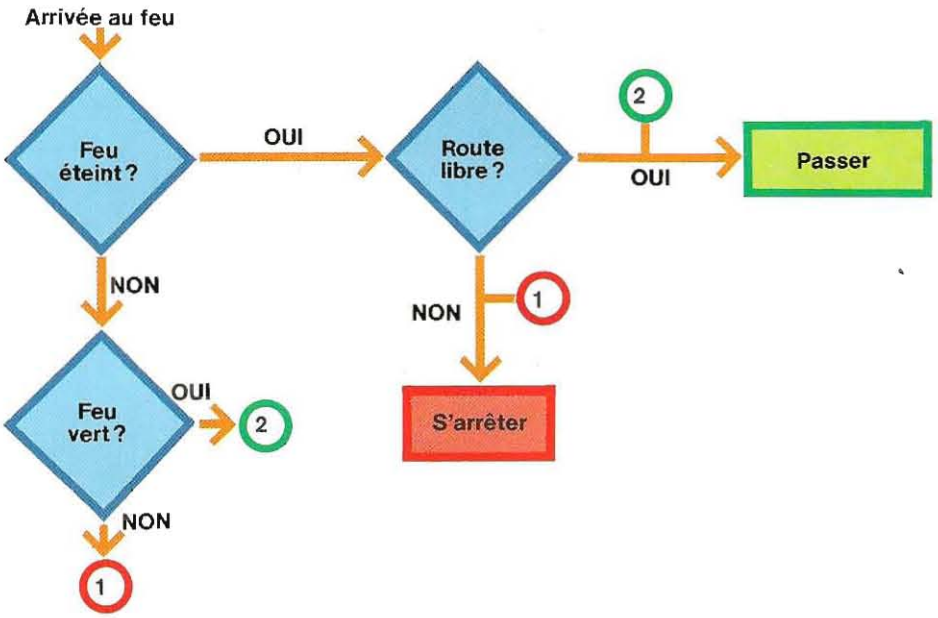


matiser ses différents comportements possibles par les symboles **b** et **c** (voir schéma p. 169).
On utilise le symbole **e** (renvoi) du schéma de la page 168 pour indiquer le point de reprise du programme après exécution du bloc d'instructions précédent. On évite ainsi de tracer la ligne de liaison, ce qui facilite la lecture des organigrammes et permet de les dessiner sur plusieurs pages lorsqu'ils sont longs ou compliqués. Le schéma de cette page illustre le déroulement de ce processus de renvoi.
On peut alors éviter la question "feu rouge ?" dans la mesure où, quelle que soit la réponse (oui = feu rouge, non = feu orange), l'action qui en découle est toujours l'attente.
Un programme peut s'accomplir si la machine a reçu toutes les données qu'elle doit traiter. L'introduction de ces données (par ex. à l'aide du clavier) est une opération d'entrée (Input), et leur restitution par la machine une fois traitées (sur une imprimante, par exemple), une opération de sortie (Output).
Le symbole graphique le plus couramment utilisé pour représenter une opération générale d'entrée-sortie (E/S) est le symbole **h** (voir p. 168). Le contenu du symbole indiquera s'il s'agit d'une entrée ou d'une sortie.

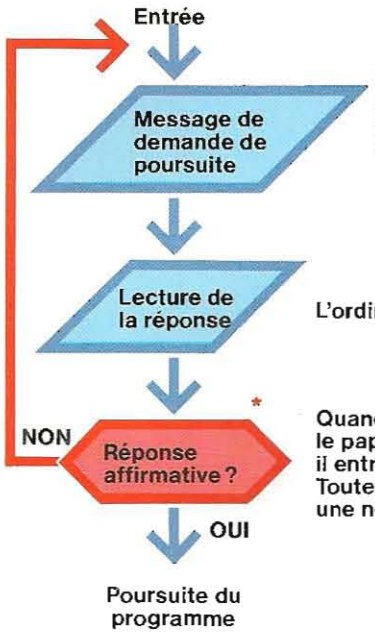
Symboles graphiques moins utilisés

Le symbole **f** représente une intervention manuelle, telle que le remplacement d'une disquette ou l'alimentation du bac de papier destiné à l'imprimante. Comme il s'agit d'une opération manuelle, le programme doit prévoir une pause qui laisse à l'opérateur le temps de l'effectuer. Le plus souvent, on définit cette pause en subordonnant la poursuite de l'exécution du programme à un accord explicite de l'opérateur.
Lorsqu'arrive le moment de l'opération manuelle, le programme exprime une demande d'accord et attend la réponse ; l'opérateur a alors le temps d'effectuer l'opération. Dès qu'il est prêt, il entre l'ordre de poursuivre. Le schéma en haut de la page 171 représente l'organigramme simplifié de cette procédure.
Le symbole **g** indique toute opération auxiliaire, sans lien étroit avec le programme, tandis que le symbole **i** désigne l'émission ou la lecture d'une bande perforée. Ce support d'information permet l'enregistrement des données par des perforations (l'existence d'un trou correspond au niveau 1, voir schéma p. 171). Ce ruban de papier a été supplanté par les supports magnétiques.

SIMPLIFICATION DE L'ORGANIGRAMME PRECEDENT PAR L'EMPLOI DES SYMBOLES DE RENVOI ① ET ②



METHODE D'ATTENTE POUR LES OPERATIONS MANUELLES



La demande est exprimée à ce moment, et peut être, par exemple : remplacer le papier de l'imprimante. Rentrer 1 dès que c'est fait.

L'ordinateur attend la réponse.

Quand l'opérateur a remplacé le papier, il entre la valeur 1. Toute autre valeur entraîne une nouvelle attente.

* *Cette décision équivaut à : la valeur entrée est-elle 1 ?*

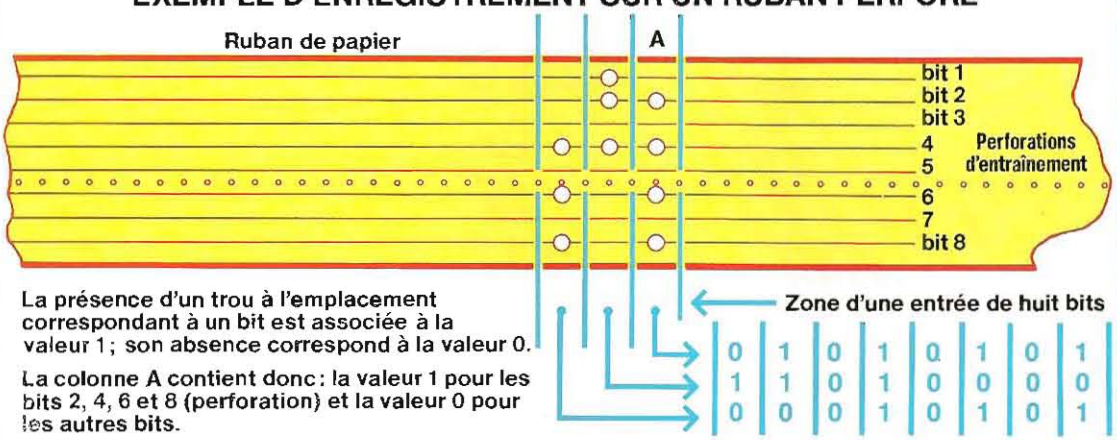
Décisions

Fonctions d'E/S

On peut remplacer l'organigramme détaillé par le symbole d'opération manuelle seul :



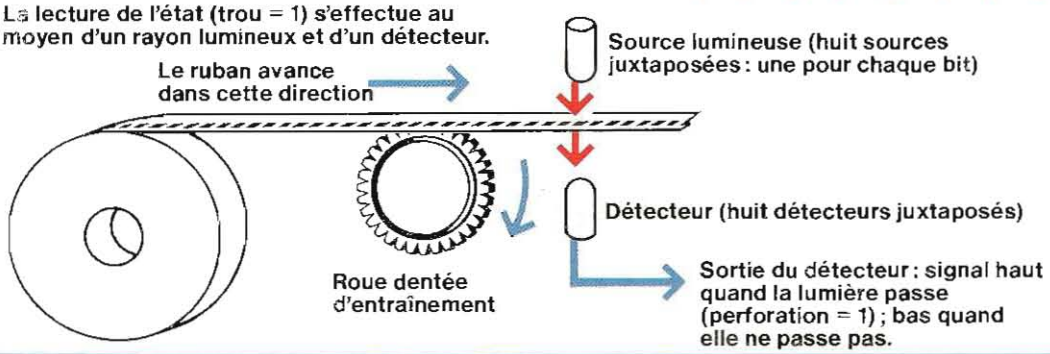
EXEMPLE D'ENREGISTREMENT SUR UN RUBAN PERFORE

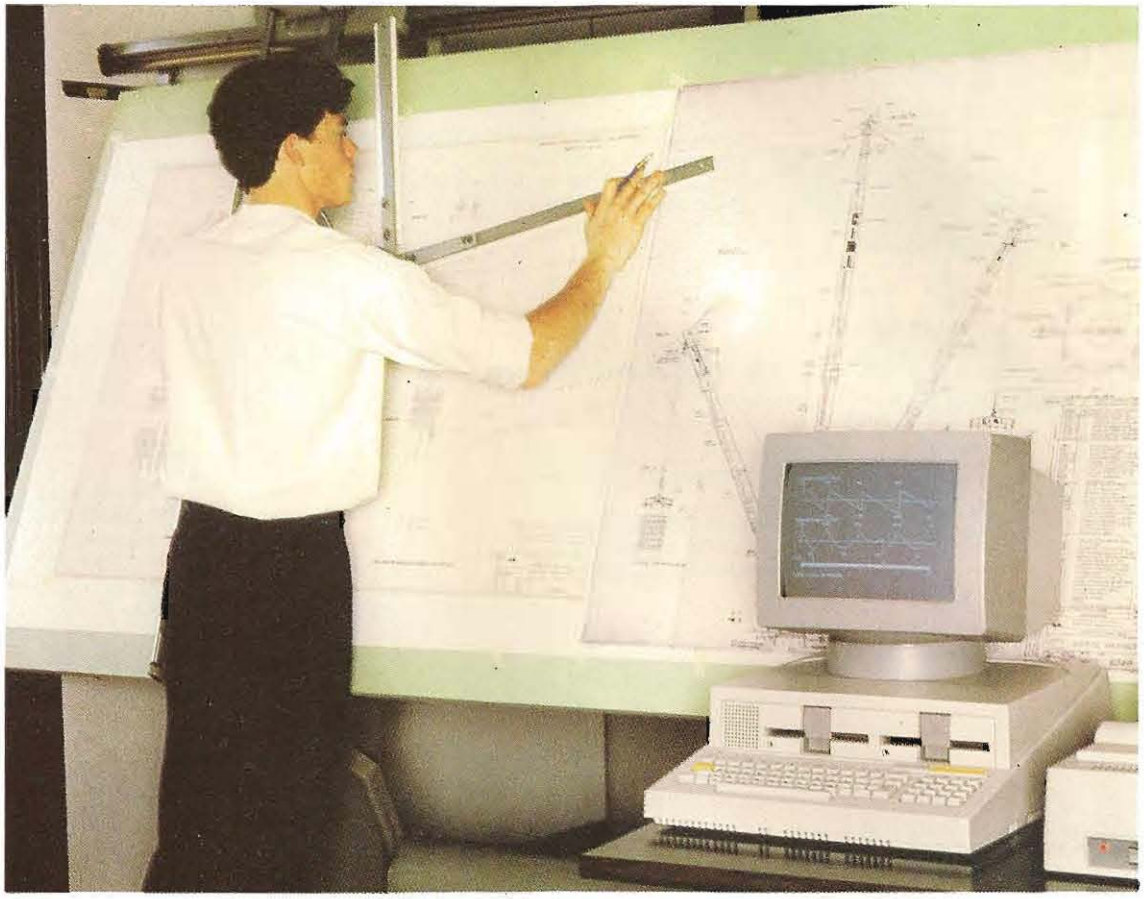


La présence d'un trou à l'emplacement correspondant à un bit est associée à la valeur 1 ; son absence correspond à la valeur 0.

La colonne A contient donc : la valeur 1 pour les bits 2, 4, 6 et 8 (perforation) et la valeur 0 pour les autres bits.

La lecture de l'état (trou = 1) s'effectue au moyen d'un rayon lumineux et d'un détecteur.





Olivetti

Utilisation de l'ordinateur pour l'établissement de plans : la machine fournit des données d'une extrême précision et des idées nouvelles en un temps record.

Le symbole **i** représente une opération d'entrée-sortie qui nécessite l'édition d'un document. On l'utilise surtout pour attirer l'attention sur les opérations particulières d'impression comme l'impression des bulletins de paie. Le symbole **m** indique une introduction manuelle des données. Il convient d'utiliser la même logique que pour le symbole **f** (opération manuelle). Le symbole **n** désigne une bande magnétique. Ce support a désormais presque totalement remplacé le ruban perforé et sert à l'enregistrement des données auxquelles on ne recourt pas très souvent, le disque ou la disquette étant préférables en cas d'accès fréquent aux données.
Quant au symbole **p**, il désigne la mémoire et les opérations qui s'y rattachent, tandis que le symbole **q** représente les disques, c'est-à-dire les mémoires de masse les plus employées.

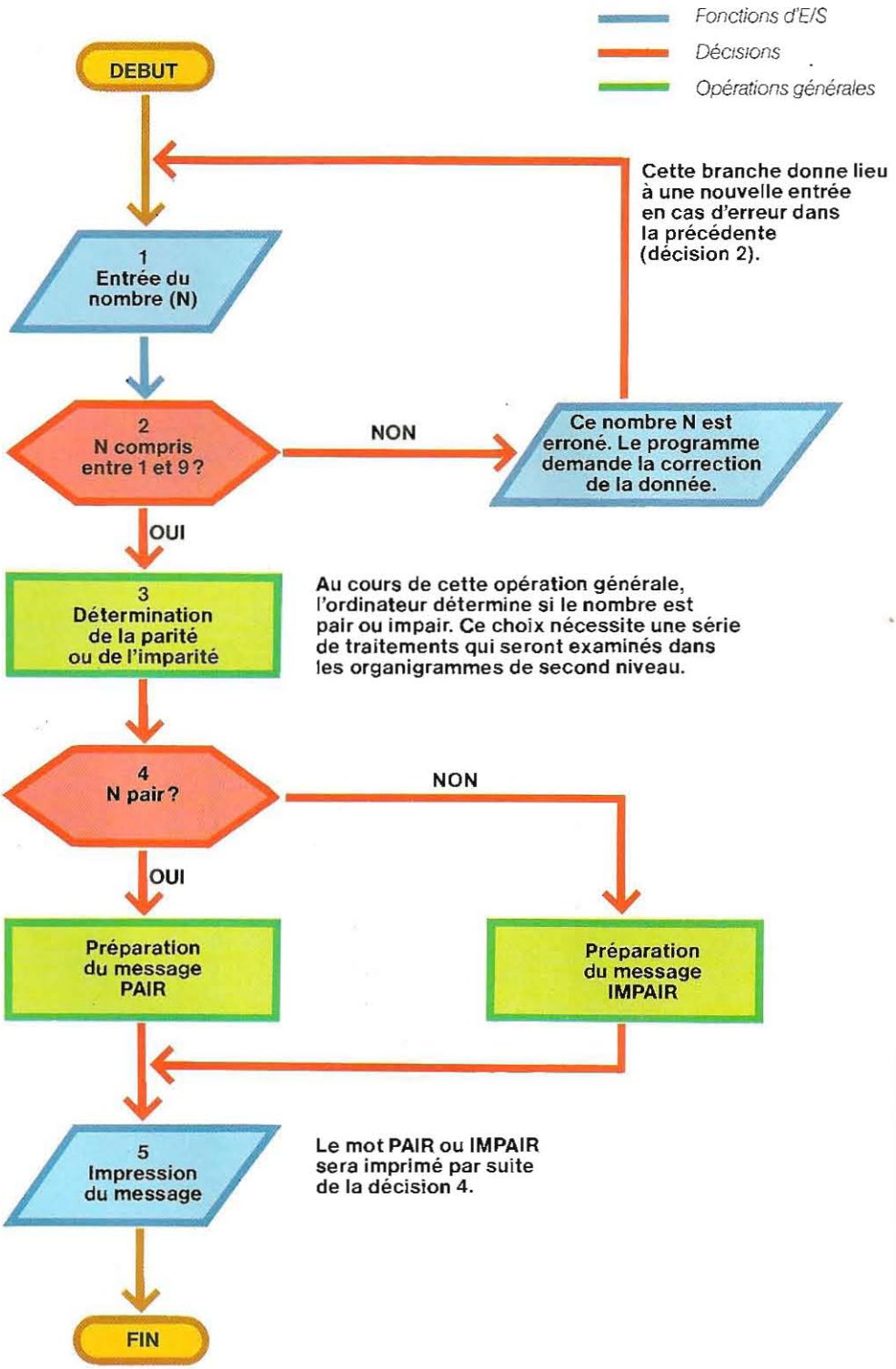
Exemple d'application

Nous voilà donc en mesure d'établir l'organigramme d'un problème donné. Nous compléterons par la suite l'illustration de cette méthode, mais nous pouvons, d'ores et déjà, étudier un exemple complet d'établissement d'un organigramme.
Prenons le problème suivant : lire un nombre N compris entre 1 et 9, et écrire s'il est pair ou impair.
Il faut commencer par décomposer le problème en un certain nombre d'actions plus simples. Ici on aura :

1 / lecture du nombre N (opération d'E/S) ;
2 / vérification : N est-il bien compris entre 1 et 9 (décision) ? ;
3 / détermination de la parité ou de l'imparité de N (action générale) ;
4 / choix de la réponse à fournir (décision) ;
5 / impression de la réponse (opération d'E/S).

L'ensemble des symboles correspondants

EXEMPLE D'APPLICATION D'UN ORGANIGRAMME DE PREMIER NIVEAU



constitue l'organigramme de premier niveau illustré par le schéma de la page 173.
L'application de la méthode HIPO (Hierarchy + Input-Process-Output ; ou en français HETS, Hiérarchie Entrée Traitement Sortie) au schéma du programme donne :

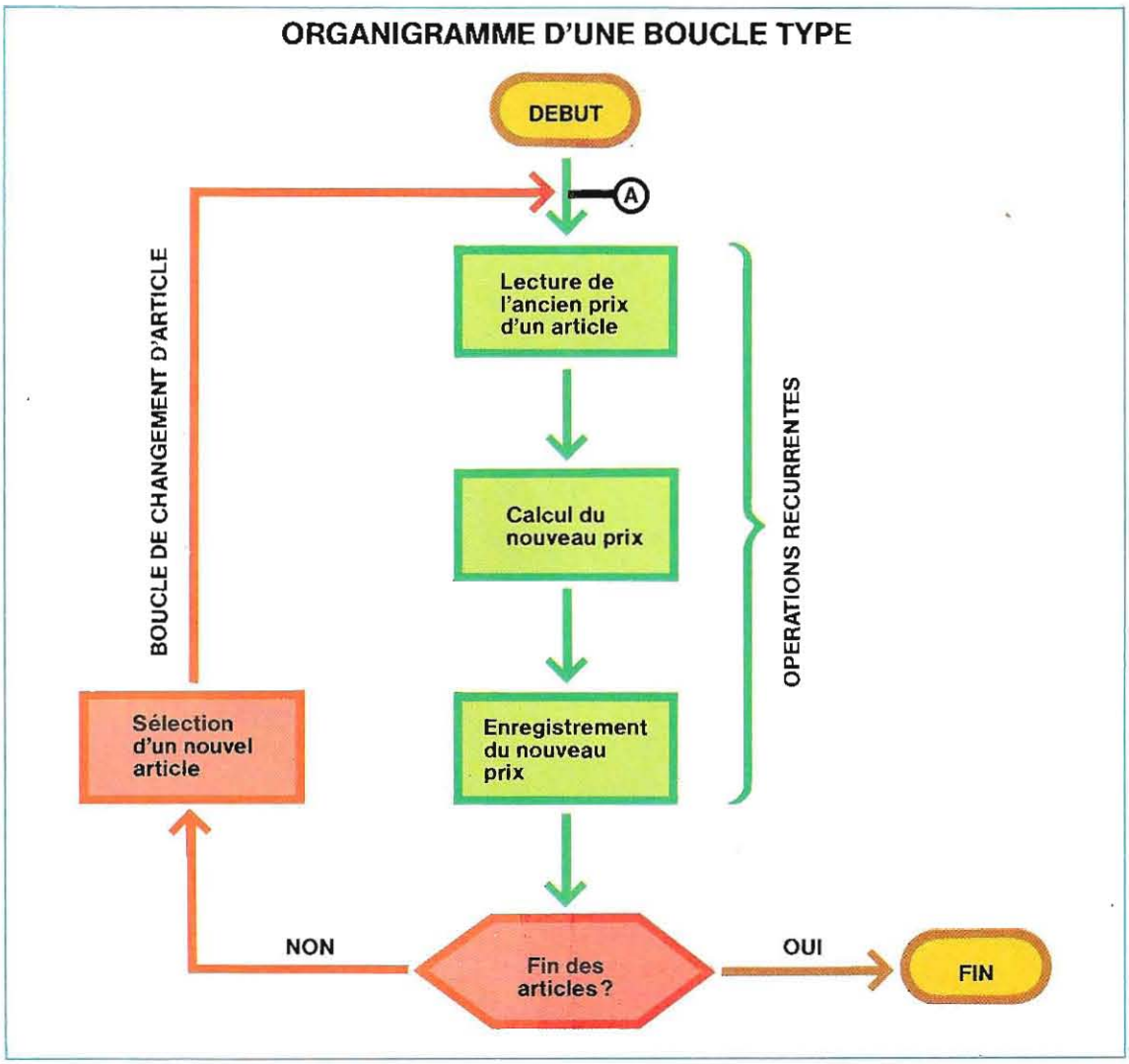
- entrée : N ;
- traitement : vérification de la validité de N (compris entre 1 et 9) et détermination de sa parité ou de son imparité ;
- sortie : message d'erreur si N n'est pas compris entre 1 et 9 ; sinon, message PAIR ou IMPAIR.

Les boucles

Il est tout particulièrement utile d'employer un système à microprocesseur pour effectuer des calculs récurrents portant sur différentes données. On résout ce problème par l'emploi d'une **boucle** (loop en anglais), c'est-à-dire d'un programme ou d'une partie de programme capable, à la fin d'un traitement, de revenir à son propre début, de prélever une nouvelle donnée et d'y effectuer les mêmes opérations. Ce cycle (boucle) doit être répété jusqu'à épuisement des données.
Par exemple, à la suite d'une augmentation de 12 %, on veut connaître les nouveaux prix des articles d'un magasin. Pour chaque article, il faut effectuer le calcul récurrent suivant :

$$\text{nouveau prix} = \text{ancien prix} + \left(\frac{\text{ancien prix}}{100} \times 12\right)$$

Le schéma ci-dessous, illustrant l'organigramme de premier niveau de ce calcul, fi-

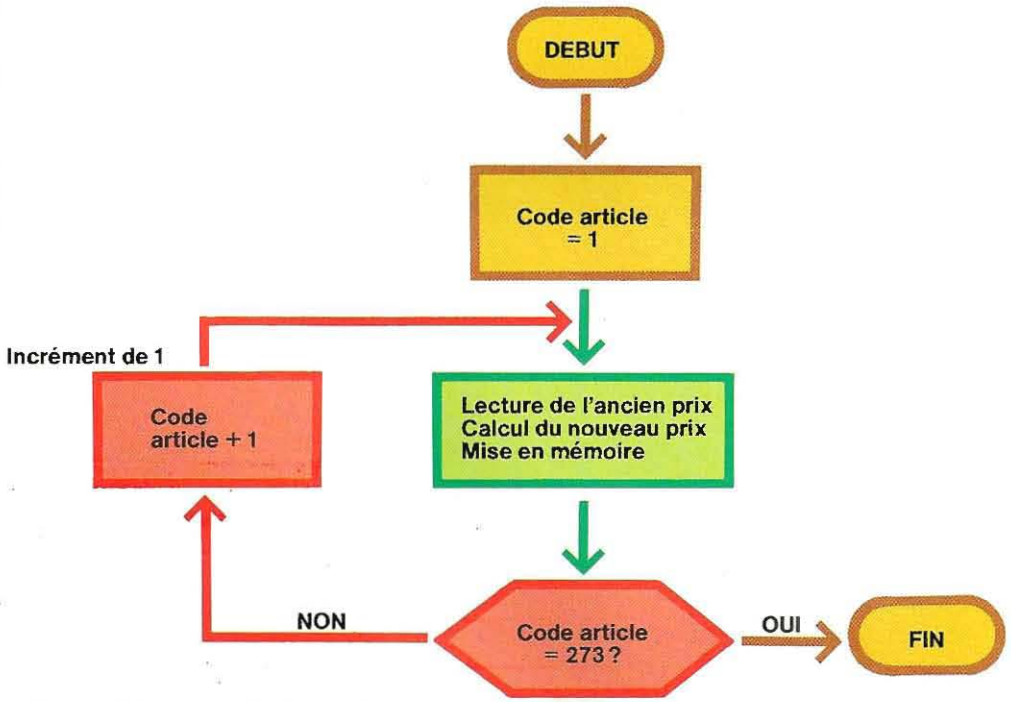


gure en rouge les parties constituant la boucle. Pour chaque article, le branchement, c'est-à-dire le retour au début du traitement, se fait au point A.

Boucle à limites explicites

Si un magasin contient 273 articles différents codifiés selon une numérotation allant de 1 à 273, il faudra répéter la boucle 273 fois (une fois par article) en commençant par l'article numéro 1. Pour savoir si tous les articles ont été traités, il suffira de demander si l'on en est au numéro 273. Si ce n'est pas le cas, on pourra sélectionner un nouvel article en ajoutant simplement 1 au code de l'article précédent. L'organigramme représenté ci-dessous tient compte de ces modifications. Dans ce programme, le code article est un nombre

BOUCLE MODIFIEE POUR LE CONTROLE DES ARTICLES JUSQU'AU CODE 273



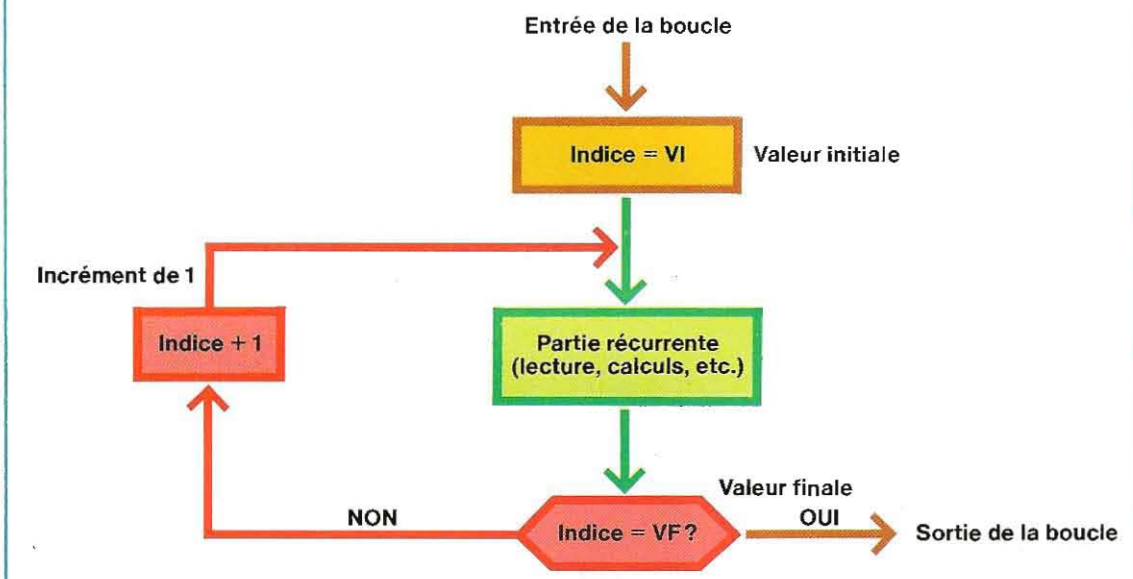
compris entre 1 et 273, et qui désigne l'article en cours de traitement. On l'appelle l'**indice de boucle**. La plupart des boucles sont indicées : elles se répètent de la valeur initiale à la valeur finale de leur indice (ici, 1 à 273).

Boucle à limites paramétrées

Sur le schéma de cette page, les limites de la boucle, c'est-à-dire les valeurs initiale et fi-

nale de l'indice (1 et 273) sont exprimées sous une forme numérique explicite. On peut également les exprimer de façon indirecte en leur affectant un symbole.
Si l'on appelle **VI** (valeur initiale) et **VF** (valeur finale) ces deux limites, on peut représenter l'organigramme d'une boucle type par le schéma ci-dessous. La désignation symbolique des limites d'une boucle permet de les paramétrer. On dit qu'une partie de programme est paramétrée lorsqu'au lieu de se répéter entre des valeurs numériques explicites, elle accepte en entrée des noms symboliques, les **paramètres**. Il faut alors, avant l'exécution des instructions paramétrées, déclarer la valeur numérique des paramètres utilisés. Cette méthode permet d'appliquer les mêmes instructions à des valeurs numériques différentes. Dans notre exemple, si l'on veut effectuer le calcul pour tous les articles, on donnera aux paramètres les valeurs :

BOUCLE MODIFIEE POUR SON FONCTIONNEMENT ENTRE LES DEUX LIMITES VI ET VF



- **Valeur initiale** = **VI** = 1
- **Valeur finale** = **VF** = 273.

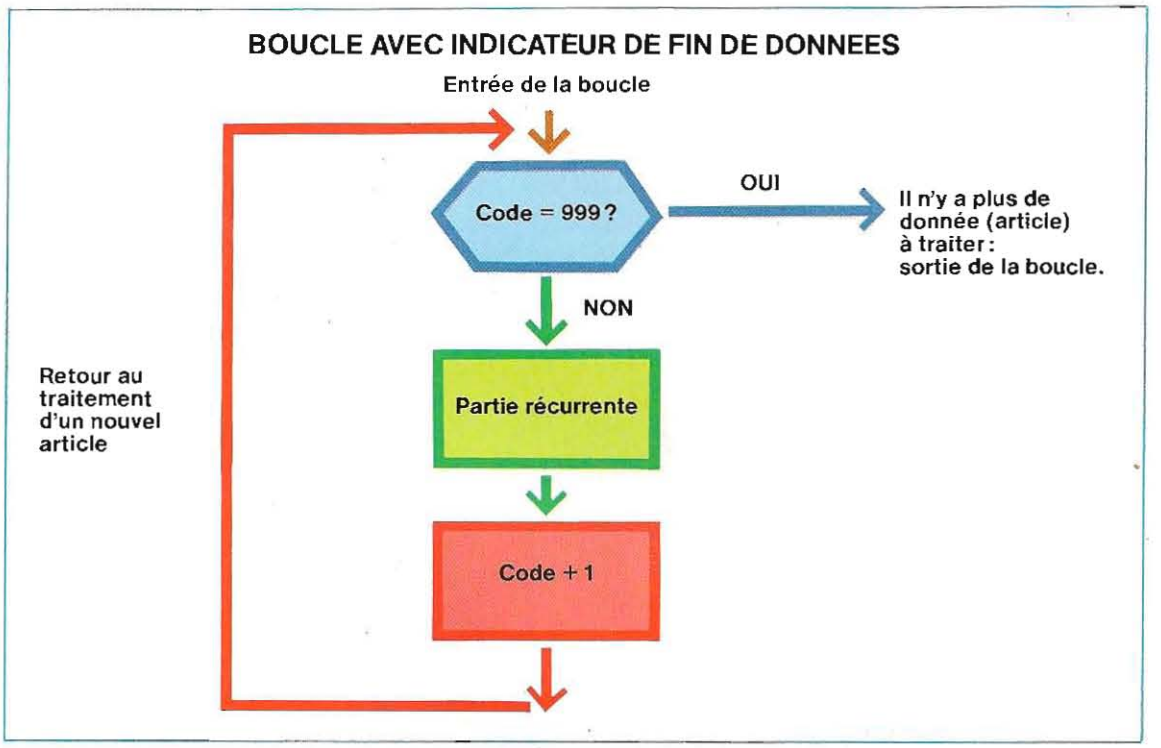
Si, au contraire, on ne voulait effectuer le calcul que pour les articles allant des numéros 20 à 56, il faudrait insérer, avant la boucle, deux instructions d'affectation :

- **VI** = 20 (commencement au code 20)
- **VF** = 56 (fin au code 56).

Boucle avec indicateur de fin de données

Les boucles que nous avons vues jusqu'ici, c'est-à-dire à limites explicites ou à limites paramétrées, ne s'achèvent qu'après le traitement de toutes les données. Elles présentent toutefois un inconvénient : on doit connaître à l'avance le nombre des données (273 dans l'exemple).
Or, on ne connaît pas ce nombre dans toutes les applications. Il faudra alors adopter un

type de boucle qui permette de déterminer la valeur finale. Pour cela, on écrit à la fin des données une valeur conventionnelle facile à reconnaître ; lorsque la boucle rencontre cette valeur, elle s'arrête. On écrit la valeur symbolique de **fin de données** au terme de la phase d'introduction des données. On peut aussi appeler cette valeur une **fin de fichier** (EOF, End of File) ou un **label de fin**.
Reprenons l'exemple du magasin. On peut si-



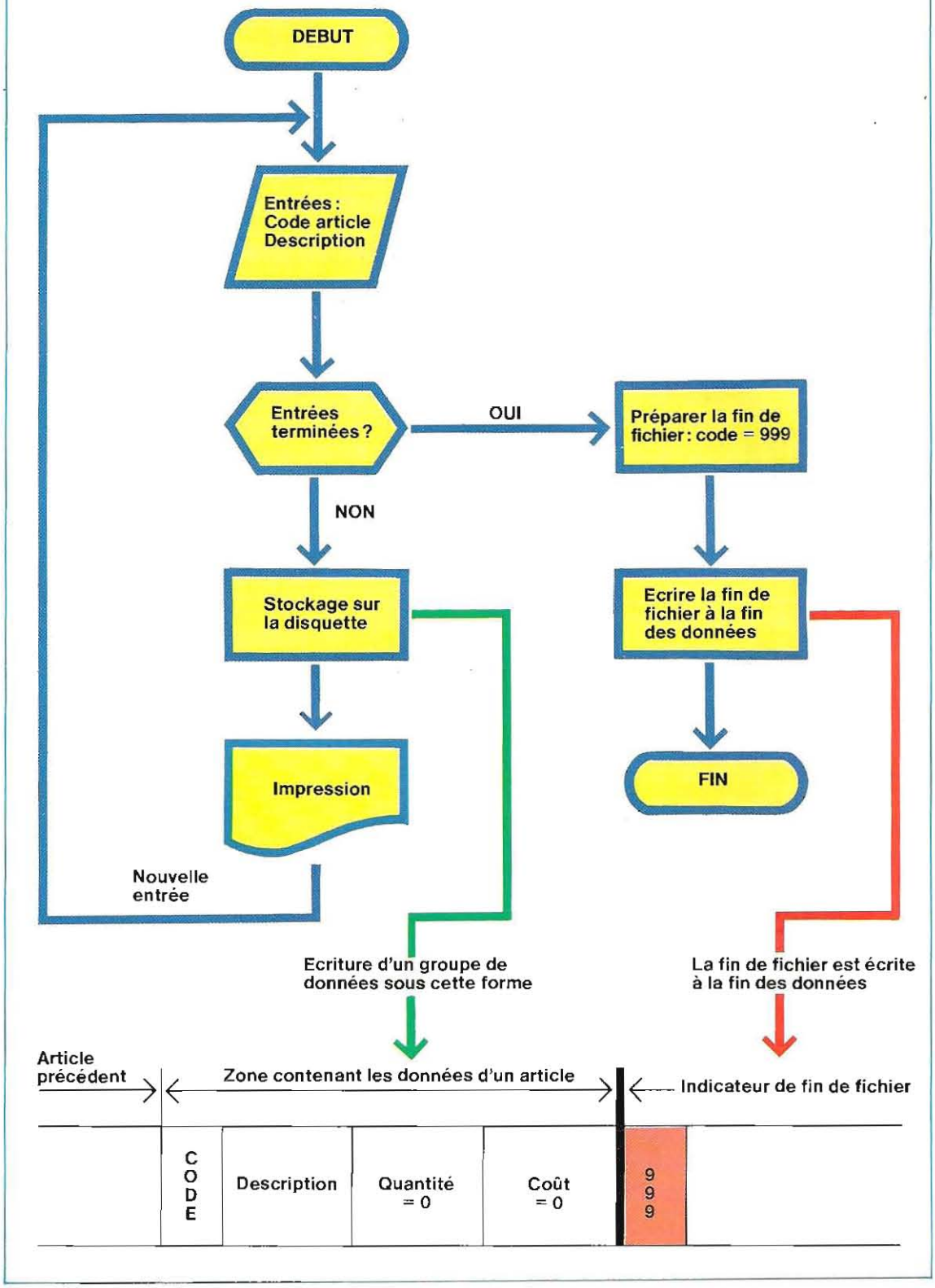
gnaler la fin des données en inscrivant un code article inexistant : la rencontre de ce code signifiera que les données sont terminées et que la boucle peut être abandonnée. Si ce code fictif est 999, on obtient l'organigramme ci-dessus. On prend alors la décision de sortir ou non de la boucle avant la partie récurrente afin de ne pas effectuer de calculs sur des valeurs erronées. Le code de fin de fichier s'écrit en phase d'introduction des données et sa valeur ne doit pas être celle d'une donnée. Dans l'exemple du magasin, on doit, avant de fixer cette valeur, prévoir la possibilité de création de nouveaux articles. On modifie alors l'organigramme (schéma p. 178).

Boucle à rupture de code

Ce type de boucle permet d'effectuer des opérations itératives tant qu'un paramètre donné conserve sa valeur initiale. Quand ce paramètre change de valeur, on dit qu'il y a **rupture de code** et la boucle est interrompue.
On utilise couramment cette définition dans les langages Cobol et RPG*, alors que le Basic ne dispose pas d'un nom particulier pour désigner les boucles de ce type.
La technique de la rupture de code est d'une grande utilité dans les traitements sélectifs (c'est-à-dire sur des données déterminées, celles dont le code a la même valeur). Le code désigne n'importe quel élément de la

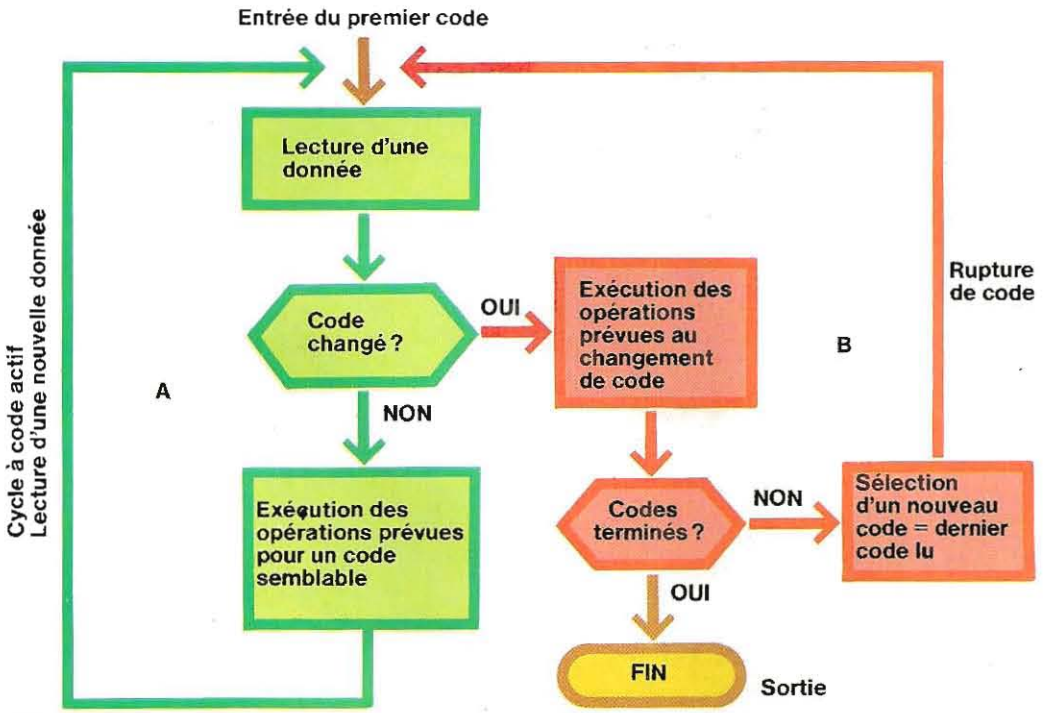
* Très proche du Cobol, le RPG (Report Program Generator), ou GAP (Générateur Automatique de Programmes), en français, est cependant moins complexe et n'est utilisé que sur certains ordinateurs (IBM).

CREATION DES RUBRIQUES DU MAGASIN



donnée. Ainsi, dans la gestion de notre magasin, nous pouvons prendre comme code la quantité en stock. Les instructions définies dans la boucle ne seront alors exécutées que pour les articles de stock égal. On peut voir sur cette page l'organigramme simplifié d'une boucle à rupture de code.
Les deux lignes de travail correspondent, la première aux opérations à accomplir sur les données de même code (bloc fonctionnel A), et la seconde aux opérations à effectuer en

ORGANIGRAMME GENERAL D'UNE BOUCLE A RUPTURE DE CODE



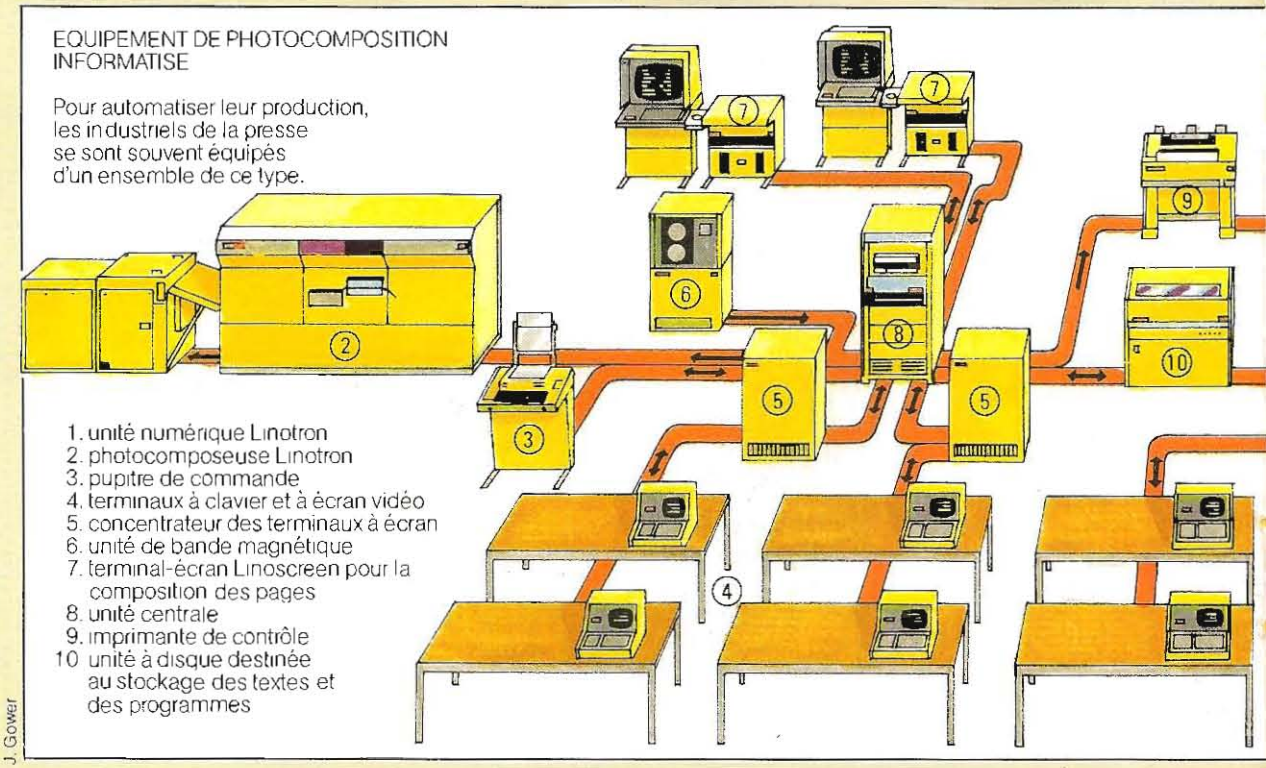
cas de rupture de code (bloc fonctionnel B). Lors de la rupture de code, on doit vérifier, après l'exécution du programme, si tous les codes ont été traités. Si c'est le cas, le programme s'arrête. Sinon, le nouveau code est sélectionné et on reprend la boucle.
L'analyse des ventes effectuées par un groupe de représentants est un bon exemple de l'utilisation de cette méthode. Il s'agit de déterminer, au terme d'une période donnée, le chiffre des ventes réalisées par chacun d'entre eux. On prend comme code le numéro du représentant. L'opération (traitement) à effectuer sur

Une révolution dans l'imprimerie

On a salué la photocomposition comme la plus grande révolution dans l'imprimerie depuis l'invention des caractères mobiles. Les procédés électroniques de reproduction permettent désormais, grâce aux transmissions par satellite, de publier simultanément journaux et revues dans le monde entier. Ainsi, on n'a plus besoin de déléguer les travaux d'impression à de grands ateliers qui doivent en sous-traiter le transport. Aujourd'hui, ce sont des petits établissements automatisés qui effectuent les différentes opérations de traitement des textes et leur transmission par ordinateur. Depuis un certain temps déjà, plusieurs hebdomadaires internationaux envoient, sur film, des pages entières prêtes à imprimer aux quatre coins du monde. L'un des plus grands quotidiens des Etats-Unis procède à la transmission par satellite d'exemplaires de son édition principale. Mais il ne s'agit là que de deux exemples d'une révolution qui a touché tous les secteurs de l'imprimerie, des journaux aux billets de banque.

Les journaux et les revues d'actualité, dont l'utilisation est immédiate, doivent être publiés rapidement. Ils ont donc été parmi les premiers à bénéficier de la technologie électronique moderne, recourant aux techniques les plus efficaces de manière à conserver leur compétitivité. C'est le cas, en particulier pour les grandes revues financières qui doivent parvenir rapidement à leurs lecteurs, aussi éloignés soient-ils. C'est pourquoi elles ont recours à la télétransmission pour s'assurer une diffusion à peu près simultanée vers tous les points du globe, tandis qu'elles se contentent des transports aériens pour la distribution régionale.

La publicité spécialisée, les informations boursières et les guides des spectacles, entre autres, sont autant de sujets qui se prêtent à une diffusion électronique par les systèmes de Télétexte (vidéographie diffusée). Les services rendus par les ordinateurs (et tout particulièrement la lecture des informations sur leurs écrans) constituent dans certains domaines (les agences de compagnies aériennes par exemple) une inestimable amélioration de la diffusion des renseignements. Malheureusement, le Télétexte n'a pas encore remporté un succès commercial suffisant pour qu'on puisse généraliser la diffusion télématique des journaux. Selon certains experts, cela n'est pas dû seulement à la résistance naturelle du public aux changements, mais aussi à la difficulté réelle qu'il y a



EQUIPEMENT DE PHOTOCOMPOSITION INFORMATISE

Pour automatiser leur production, les industriels de la presse se sont souvent équipés d'un ensemble de ce type.

1. unité numérique Linotron
2. photocomposeuse Linotron
3. pupitre de commande
4. terminaux à clavier et à écran vidéo
5. concentrateur des terminaux à écran
6. unité de bande magnétique
7. terminal-écran Linoscreen pour la composition des pages
8. unité centrale
9. imprimante de contrôle
10. unité à disque destinée au stockage des textes et des programmes

J. Gower

à s'adapter aux mutations qui se succèdent à un rythme extrêmement rapide.

Malgré sa mécanisation, l'imprimerie a fonctionné selon des principes à peu près inchangés pendant plus de quatre siècles. La composition se faisait lettre par lettre, avec des caractères en alliage de plomb, ou en bois pour les affiches de grandes dimensions. Le métal offrait cependant l'avantage de pouvoir être fondu et réutilisé.

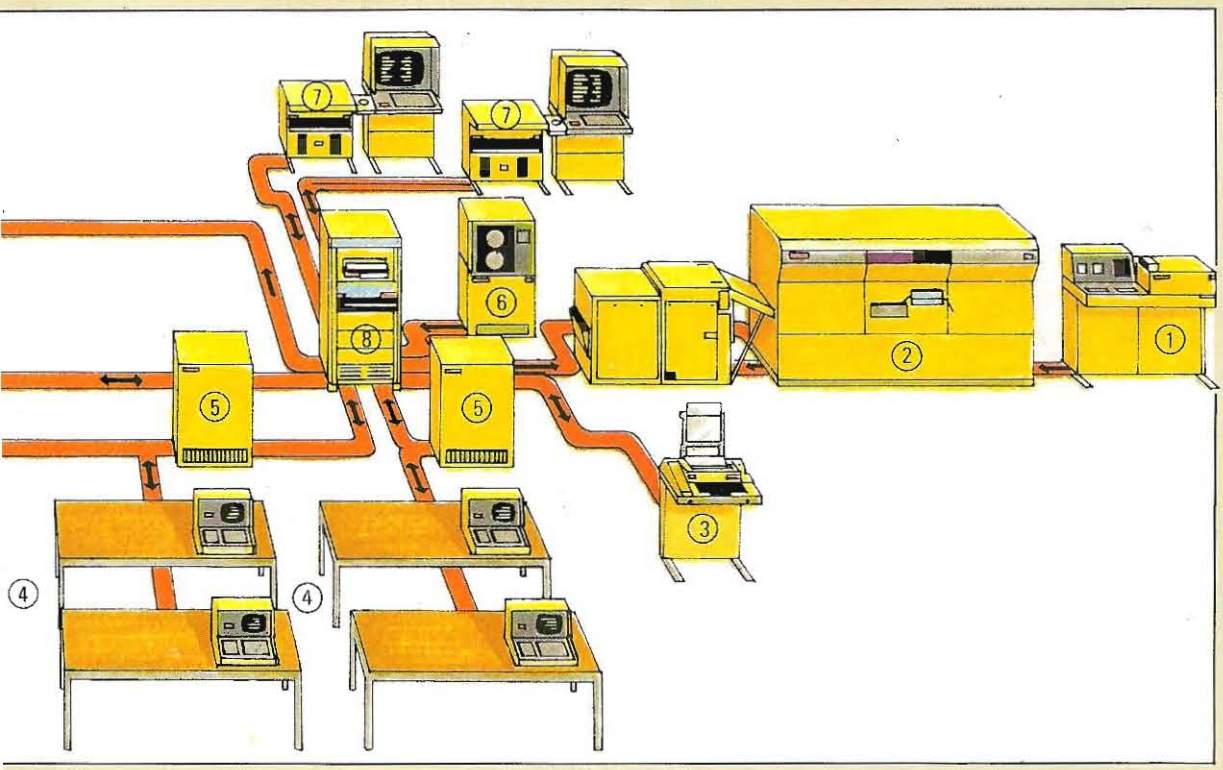
On employait alors des planches à imprimer planes, et c'est le quotidien londonien «Times» qui bouleversa la technique de transfert du texte sur le papier en introduisant la première rotative, machine à imprimer munie de plaques courbes enroulées à la surface d'un cylindre.

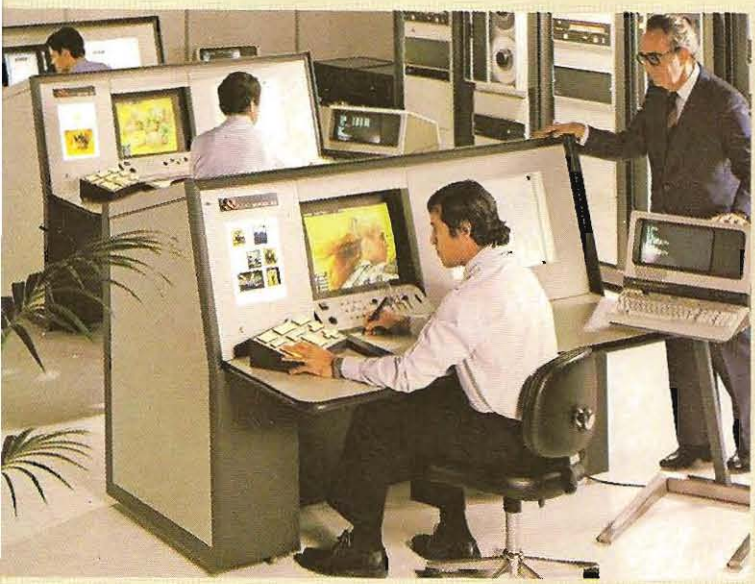
Aujourd'hui, tous les systèmes mécaniques de composition, de reproduction et d'impression sont peu à peu supplantés par les procédés électroniques. La composition se fait désormais essentiellement par photographie. L'opérateur dispose d'un clavier et d'un écran, et tape le texte qui, à sa demande, est automatiquement traité par l'ordinateur et enregistré sur une bande ou un disque magnétique. L'ordinateur effectue les espacements entre les lettres et les mots ainsi que la mise en place du texte dans la page, comme le veut l'opérateur. Les signaux enregistrés magnétiquement sont ensuite communiqués à une photocomposeuse.

Les premières photocomposeuses ont été de simples adaptations des machines munies d'un clavier et qui servaient à composer les caractères de plomb. Aujourd'hui, les photocomposeuses de la «quatrième génération» fonctionnent à des vitesses si extraordinaires qu'une seule suffit à traiter les textes provenant de centaines de clavistes. La reproduction des images bénéficie également des améliorations apportées au matériel électronique.

On estime qu'aujourd'hui, près de 50% des reproductions en couleurs sont faites par analyse électronique. Des analyseurs décomposent les couleurs de l'original en deux, trois ou quatre «sélections» d'une seule couleur reproduites séparément sur des films qui sont ensuite superposés et imprimés de façon à restituer une image en couleurs semblable à l'original.

La reproduction électronique des images et des textes en un seul processus, ou «prepress», doit son apparition à l'industrie de la presse quotidienne.





Sci-Tex Europe S.A.

Grâce au système Sci-Tex, la mise en page apparaît directement sur l'écran.

Souvent annoncée, la réunion de ces deux éléments essentiels dans un produit imprimé n'a abouti que récemment. Le texte et les images sont transformés en signaux électroniques, puis introduits sous cette forme dans un ordinateur. Les informations sont alors manipulées par l'intermédiaire d'un clavier et mémorisées. Le claviste peut facilement apporter modifications et corrections car il suit sur l'écran les corrections au fur et à mesure qu'il les tape sur le clavier. Puis il donne à l'ordinateur les instructions nécessaires pour qu'il enregistre la nouvelle version.

Au moment de l'impression, les signaux conservés dans la mémoire de masse de l'ordinateur sont transférés d'abord sur un film, puis sur des plaques sensibles et, enfin, sur du papier.

Les procédés purement électroniques coûtent cher. C'est pourquoi on a récemment mis au point une méthode simplifiée de reproduction électronique en couleurs qui convient surtout aux petites entreprises.

Les systèmes électroniques n'intéressent pas que les éditeurs de journaux et de revues, mais aussi tous ceux qui publient des catalogues de vente par correspondance et ont également besoin d'une méthode très rapide pour imprimer des mises à jour de dernière minute.

Si les ordinateurs travaillent à très grande vitesse, ils exigent néanmoins un temps assez important de programmation et de codification des informations. Dès que le travail de mise en page est compliqué, de nombreuses personnes, surtout si elles sont adroites et minutieuses, préfèrent travailler directement sur le film. En effet, facile à manipuler, le film offre en outre une image directe de ce que l'on fait.

L'héliogravure, procédé d'impression basé sur un cylindre gravé à l'acide, offre la meilleure qualité possible sur du papier léger et bon marché. On l'emploie dans les revues et les catalogues de grande diffusion. Depuis l'adoption des techniques électroniques les plus récentes, la phase de gravure des cylindres fait partie intégrante de la conversion directe d'une image en couleurs en une image à imprimer. Or, la lenteur du processus de gravure et son coût élevé constituent les deux principaux inconvénients de ce procédé.

Pour effectuer la gravure au laser, on recouvre les cylindres d'un revêtement de plastique sur lequel un rayon laser commandé par un analyseur de couleurs ou par d'autres unités grave l'image à imprimer. Cette méthode permet d'espérer une diminution des coûts. Cependant, et notamment dans les établissements les plus petits, la plus grande partie de la production sur cylindre se fait encore par gravure chimique à l'acide sous contrôle électronique.

Dans l'imprimerie américaine Harris à mouvement automatique, l'impression se fait véritablement en continu. Tandis que deux unités impriment des parties d'un livre en une ou deux couleurs, on prépare deux autres unités d'impression pour la partie suivante. Un microprocesseur augmente la vitesse des unités de préparation jusqu'au maximum tandis qu'il arrête automatiquement les autres unités d'impression dès que le nombre d'exemplaires prévu est imprimé.

On peut aujourd'hui panacher les techniques d'impression. C'est ce que prouve le succès du procédé Néosan selon lequel on peut utiliser des typons (films offset) respectant une angulation de trame compatible avec la préparation des cylindres pour l'impression en héliogravure.

Les mêmes films de sélection des couleurs peuvent ainsi servir à l'impression par l'un ou l'autre procédé.

C'est cette grande souplesse de fonctionnement qui peut donner à la presse toutes les chances de faire face aux exigences modernes de communication rapide des informations.

Solutions du test 4

1 / b

2 / a : l'instruction 106 contient le mot RETOUR qui déclenche le mécanisme de retour à l'instruction venant après l'instruction de saut. Après l'instruction 106, le programme reprend donc à l'instruction 20.

b : dans ce cas, l'instruction RETOUR est absente ; le programme continue donc en série à partir de l'instruction 106 : il exécute les instructions 107 et suivantes.

3 / b : UAL est l'abréviation d'unité arithmétique et logique, qui est, comme son nom l'indique, destinée à effectuer des calculs.

4 / a

5 / c

6 / L'erreur réside dans la présence du mot RETOUR dans l'instruction 33. Ce mot ne doit être utilisé que pour le retour au point de débranchement après un saut. Or, ce programme ne comporte aucun saut qui conduise à la ligne 33.
Inversement, le mot RETOUR est justifié dans l'instruction 57 puisqu'il y a un saut (instruction 32).

7 / b : l'Interruption permet de suspendre le traitement en cours afin d'utiliser les ressources du système pour un service demandé par l'extérieur.

8 / a : faux. Les EPROM sont destinées à stocker des programmes. On ne peut les effacer ou y écrire de nouveau qu'au moyen d'appareils spéciaux et elles ne peuvent donc pas servir à mettre des données en mémoire.

b : faux. L'unité de disque peut contenir de 250 000 à plusieurs millions de caractères (jusqu'à des centaines de millions pour les plus grandes). Le chiffre 64 000 est de l'ordre de grandeur de la mémoire d'un micro-ordinateur.

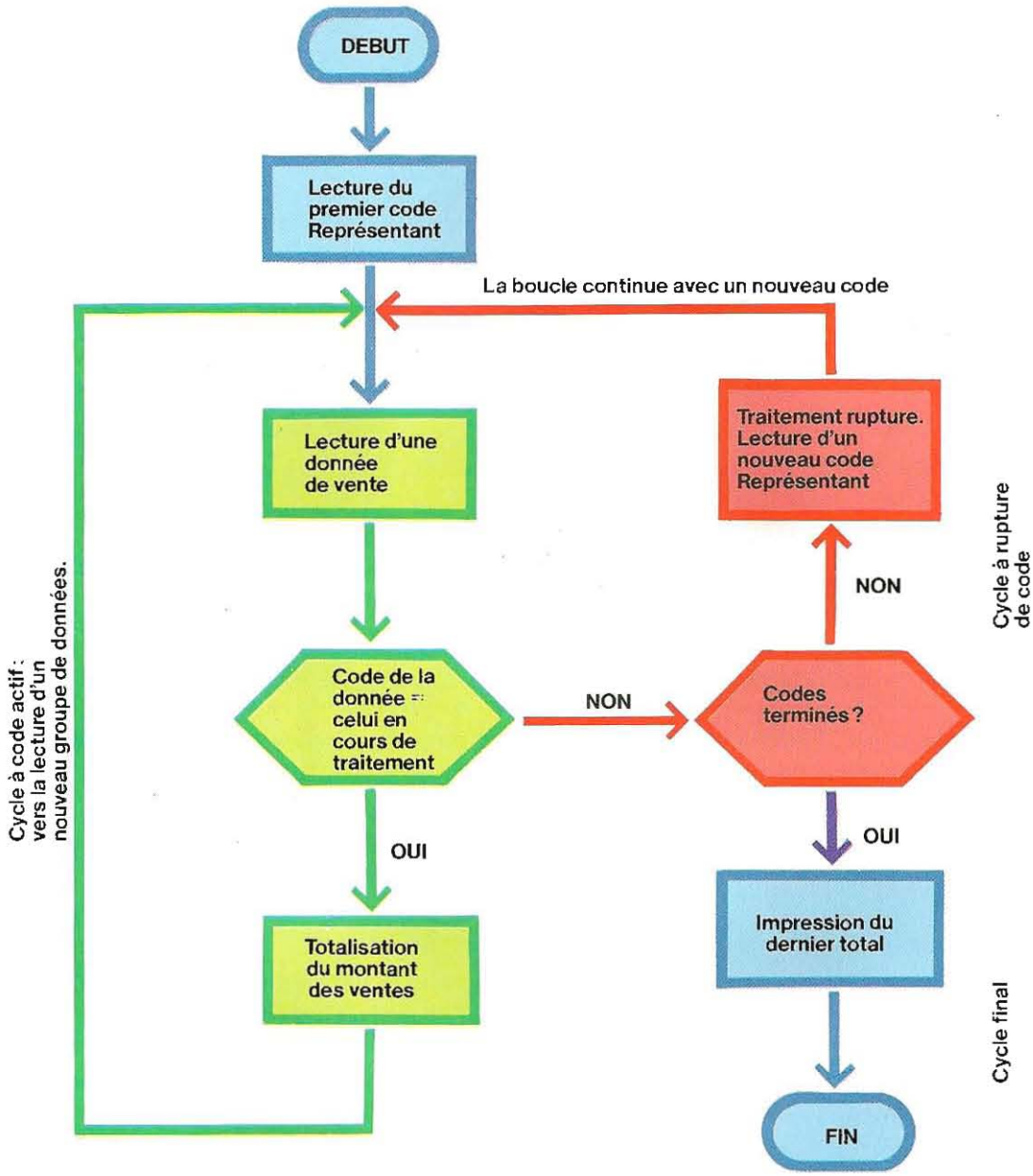
c : vrai. Il faut un équipement spécial pour programmer une ROM.

d : faux. L'UC (unité centrale de traitement) met en mémoire ce qui est en cours d'exécution, et se consacre à la nouvelle activité. Ensuite, elle reprend le déroulement normal du programme.

e : cette affirmation est partiellement vraie. Normalement, l'écran ne génère pas d'Interruption, alors que cela fait partie du fonctionnement normal du clavier.

f : vrai. Le DMA (accès direct mémoire) est le seul moyen d'accélérer la vitesse de transfert, étant donné le nombre élevé de données échangées avec ce périphérique.

ORGANIGRAMME D'UNE BOUCLE A RUPTURE DE CODE : EXEMPLE D'APPLICATION



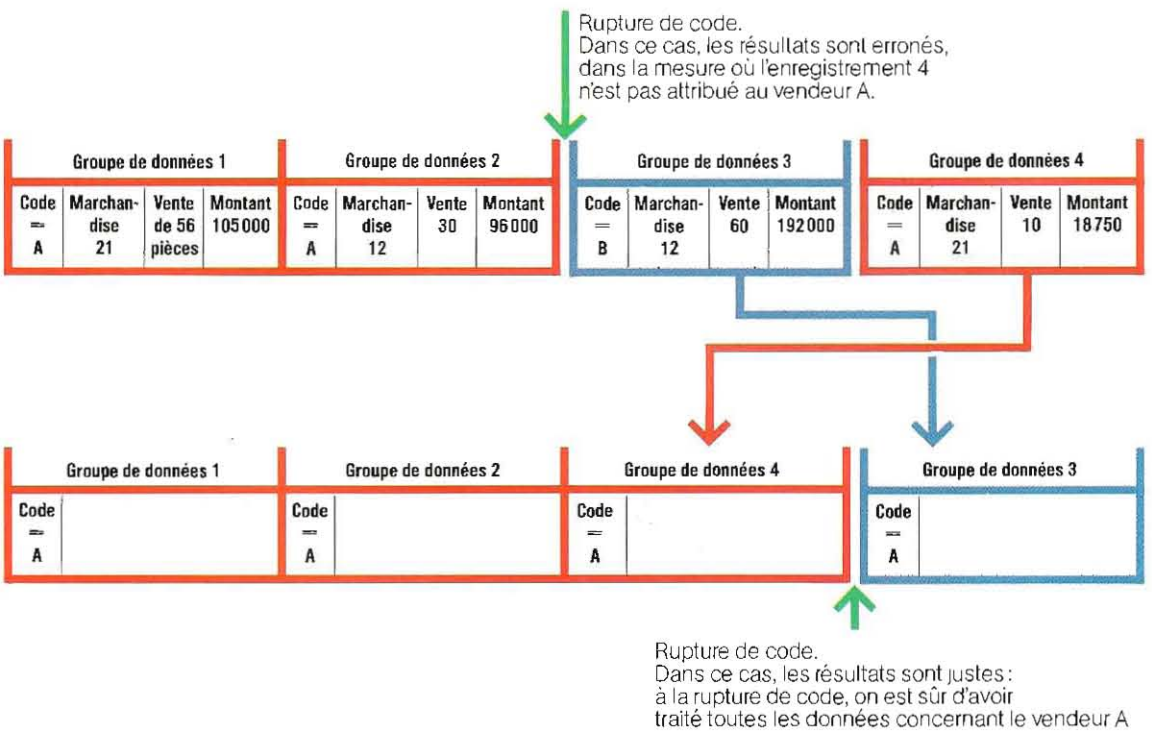
Format des ventes

	Code Repré-sentant	Code Marchandise	Quantité vendue	Montant	Code Repré-sentant	Code Marchandise	

Groupe des données concernant une vente

ce code actif consiste à fournir la somme des ventes réalisées. L'opération à effectuer à la rupture de code est l'impression du total.
On voit page 184 l'organigramme de ce problème, ainsi qu'un exemple de représentation des données concernant les ventes.
L'exécution d'une rupture de code nécessite un classement des données sur le code par ordre croissant ou décroissant sous peine de fausser les résultats. Supposons qu'il n'y ait que deux représentants (codés A et B), et que les informations sur les ventes aient été introduites dans le désordre comme il apparaît sur le schéma ci-dessous. La rupture de code (de A à B) n'indiquerait alors pas la fin des données sur le premier vendeur et le total des ventes serait donc faux puisqu'il ne tiendrait pas compte du contenu de l'enregistrement 4. En effet, à la deuxième entrée dans la boucle, après la rupture de code A, seules les données (informations) concernant B seront prises en compte et l'enregistrement 4 ne sera pas traité. Il convient donc, avant d'exécuter la boucle, de hiérarchiser les données, par exemple en ordre croissant. Les enregistrements 3 et 4 seront alors inversés et, à la rupture de code, on pourra totaliser toutes les ventes d'un même vendeur (A).

RUPTURE DE CODE





Marka

Opérateurs au travail sur des terminaux de gros ordinateurs.

En résumé, les principaux types de boucle sont les suivants :
– Boucle à limites explicites : les valeurs initiale et finale sont indiquées dans la boucle sous forme numérique.
– Boucle paramétrée : les valeurs initiale et finale sont désignées par des noms symboliques auxquels il faut attribuer une valeur numérique avant exécution. Cette méthode permet d'utiliser la boucle à plusieurs endroits du programme.
– Boucle à indicateur de fin de données : la valeur initiale est définie normalement, tandis que la valeur finale est déduite de la reconnaissance d'un code particulier.
– Boucle à rupture de code : elle traite les enregistrements du premier au dernier et comporte deux branches : l'une pour effectuer les opérations sur un même code, l'autre les traitements de fin de code (à la rupture). Il est donc indispensable que les données soient classées.

Les boucles à pas différent de 1

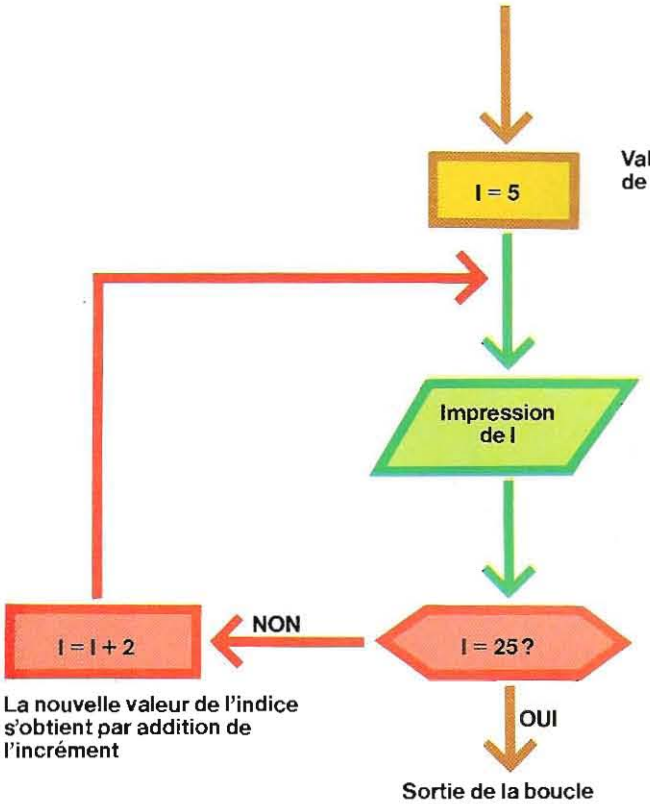
L'indice d'une boucle, c'est-à-dire la variable qui est incrémentée après chaque passage par la partie récurrente (voir schéma en haut de la page 187), n'a pas forcément un incrément de 1, ce qui permet d'opérer une sélection parmi les données.

Supposons qu'on veuille écrire tous les nombres impairs compris entre 5 et 25 : on choisira 5 comme valeur initiale, 25 comme valeur finale et 2 comme **pas** (step en anglais), c'est-à-dire comme incrément. Au premier passage, l'indice aura la valeur 5, celle du premier nombre à imprimer. Au deuxième passage, l'indice sera 5 + 2 = 7, et ainsi de suite jusqu'à 25 (voir l'organigramme de la page 187, en haut).

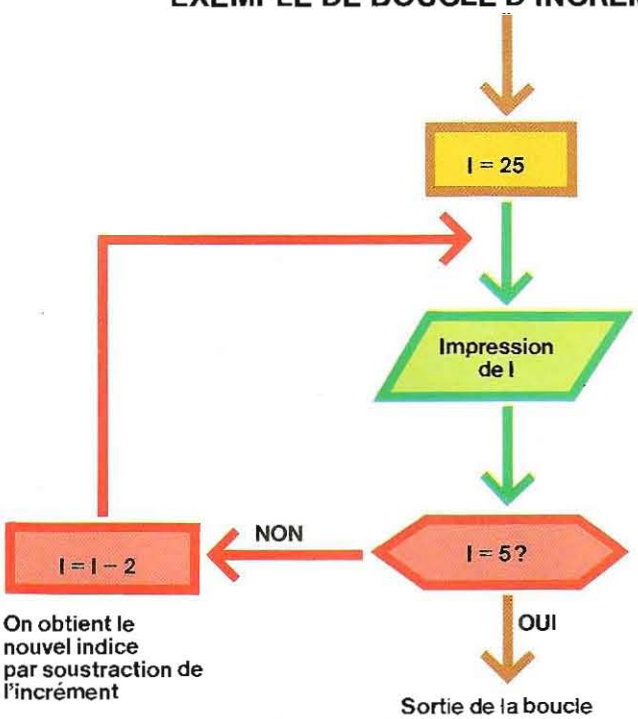
L'incrément peut aussi être négatif : la valeur initiale est alors supérieure à la valeur finale. Ainsi, dans notre exemple, en commençant à 25 pour terminer à 5, avec un pas de – 2, la liste aurait été composée des mêmes nombres mais en sens inverse (25, 23, 21..., 5). Le second schéma de la page 187 figure l'organigramme de cette boucle.

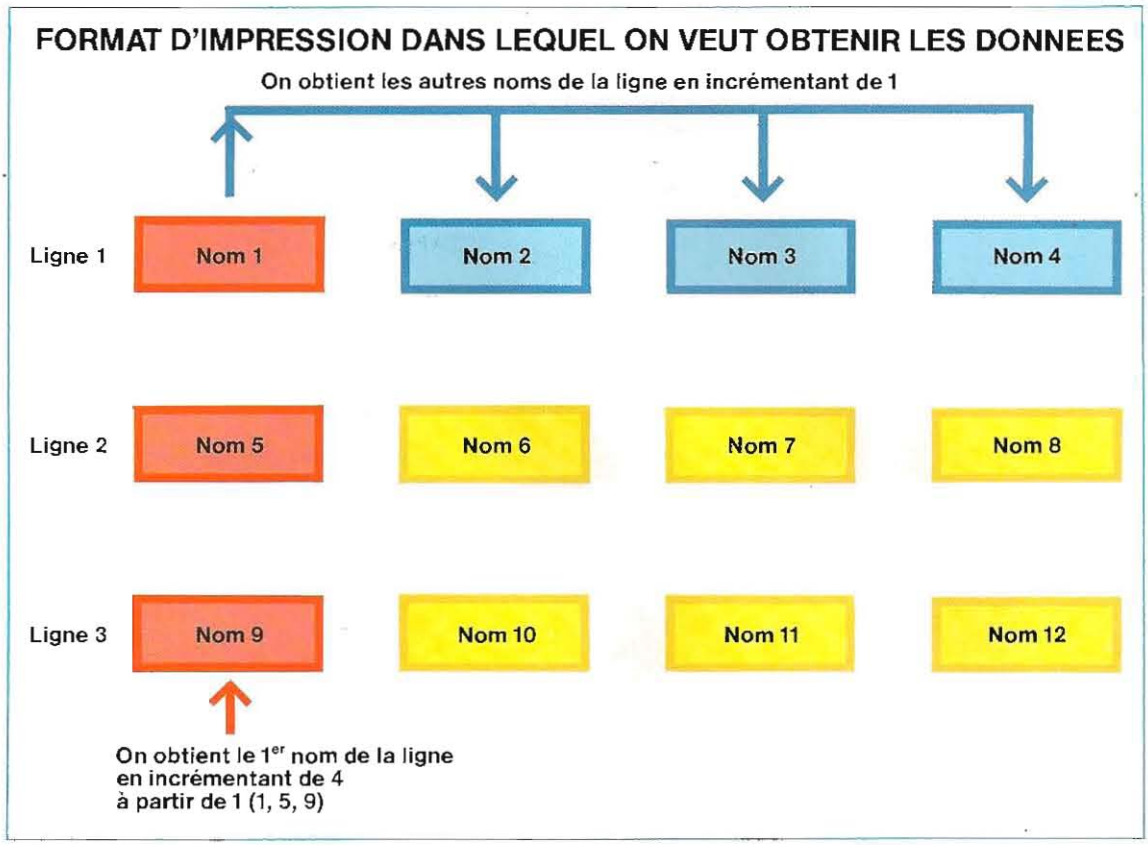
Imaginons, comme application, que nous avons douze noms à imprimer quatre par quatre, sur trois lignes. On voit, en haut de la page 188, le format d'impression désiré. Les données à imprimer (noms de 1 à 12) sont enregistrées séquentiellement dans une zone-mémoire

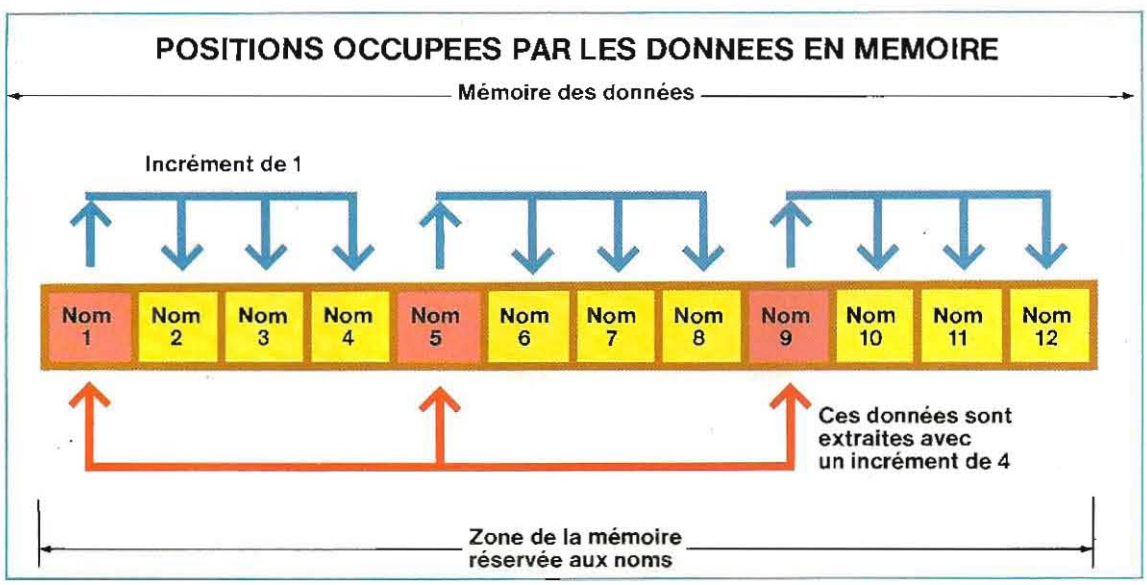
EXEMPLE DE BOUCLE D'INCREMENT DE 2



EXEMPLE DE BOUCLE D'INCREMENT DE – 2







(voir schéma ci-dessus).
Pour obtenir le format désiré, il faut procéder comme suit :

a / sélection de la donnée de début de ligne : respectivement nom 1, nom 5, nom 9 pour les lignes 1, 2 et 3 ;

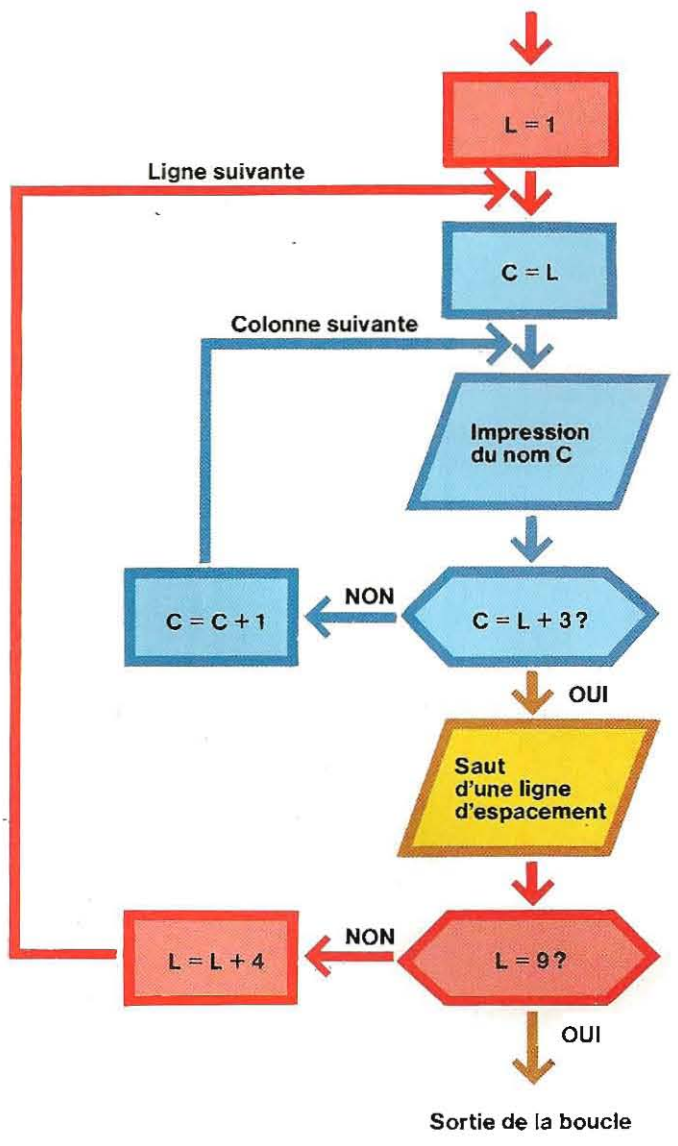
b / impression de la donnée sélectionnée (noms 1, 5 et 9), et des trois données suivantes sur la même ligne ;

c / saut d'une ligne pour aérer le listing.

On peut résoudre le point **a** par une boucle ayant une valeur initiale = 1, une valeur fi-

nale = 9 et un pas = 4. Le point **b** constitue une deuxième boucle dont la valeur initiale est l'indice de la première (1, 5, 9) et la valeur finale ce même indice + 3 (ainsi, pour la première ligne, VI est 1 et VF est 4). Cette boucle a un pas de 1.
Pour exécuter le point **c**, il faut envoyer la commande de saut de ligne (LF, voir code ASCII), entre deux impressions. Le schéma ci-dessous en représente l'organigramme.

ORGANIGRAMME DE L'IMPRESSION DE 12 DONNEES (NOMS) REPARTIES EN 3 LIGNES DE 4 NOMS CHACUNE



Sélection de la ligne.

Lors du premier tour (L = 1), C passe de 1 à 4. Les noms 1, 2, 3 et 4 sont donc imprimés. Lors du second tour (L = 5), C passe de 5 à 8 ; les noms 5, 6, 7 et 8 sont imprimés. Au troisième tour (L = 9), C commence à 9 et termine à 12 : les noms sont 9, 10, 11, 12.

Cette boucle a 1 pour incrément (pas = 1).

Cette boucle a 4 pour incrément (pas = 4).

Sortie forcée d'une boucle

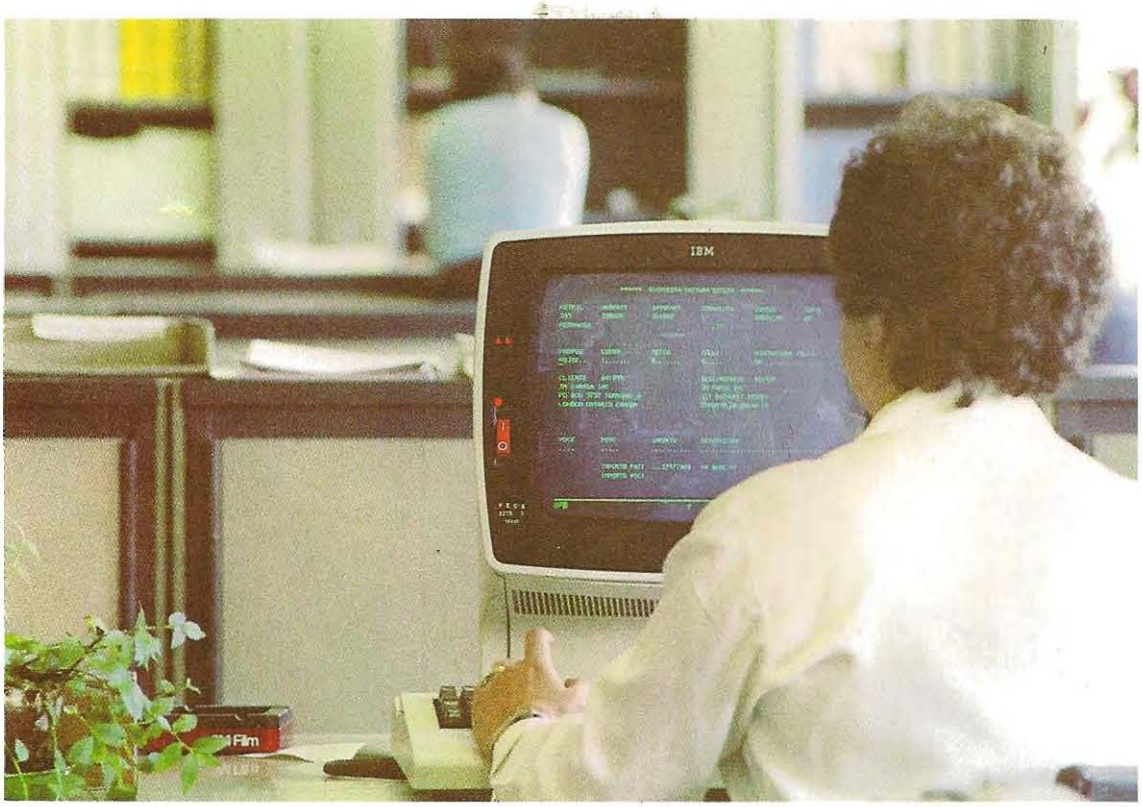
Les boucles sont très employées dans la recherche des données. Supposons que nous avons enregistré les noms et les numéros de téléphone de nos amis et que nous comptons 200 données (nom + numéro). Nous voulons savoir quel est le numéro de M. Martin. Il nous faut écrire une boucle qui accomplisse les opérations suivantes :

a / lecture d'une donnée en mémoire ;
b / comparaison entre le nom Martin et le nom contenu dans la donnée ;
c / en cas d'égalité : écriture du numéro de téléphone ;
d / en cas de non-égalité : retour au point **a** et lecture de la donnée suivante.

Ce programme s'écrit très simplement en utilisant une boucle 1 et 200 (première et dernière données contenues dans la mémoire) d'après l'organigramme de la page 191. Toutefois, cette solution présente un inconvénient : elle ne prévoit pas l'interruption de la recherche une fois obtenu le renseignement. Du temps sera donc perdu. En effet, si les données se trouvaient sur une disquette, le temps de lecture ne serait pas négligeable (bloc « lecture de donnée en position I » sur l'organigramme). Dans un programme structuré d'après ce schéma, la lecture se poursuit jusqu'à la fin, même s'il lui était possible de s'arrêter à 3. Pour réduire ce temps au strict nécessaire, on a recours à un **indicateur,** ou **aiguillage** (flag en anglais), qui signale l'aboutissement de la recherche. Les opérations de lecture et de comparaison sont alors interrompues et le programme se termine.
Voici comment les choses se passent.

- Au départ, l'aiguillage est mis à zéro (initialisation), ce qui indique que toutes les opérations prévues doivent être exécutées.
- Une fois obtenu le résultat que l'on recherchait, on donne à l'indicateur une valeur différente (1, par exemple), signifiant que toutes les opérations doivent être sautées.

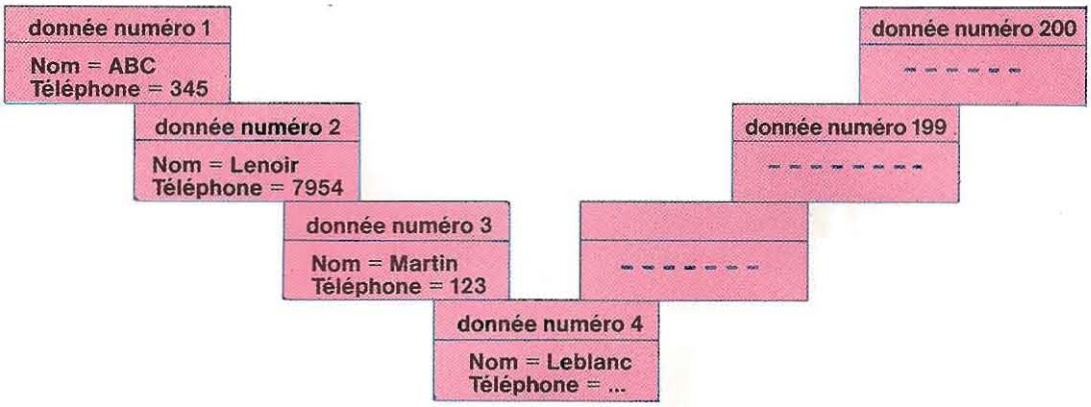
Opératrice occupée à suivre le déroulement d'une boucle de recherche.

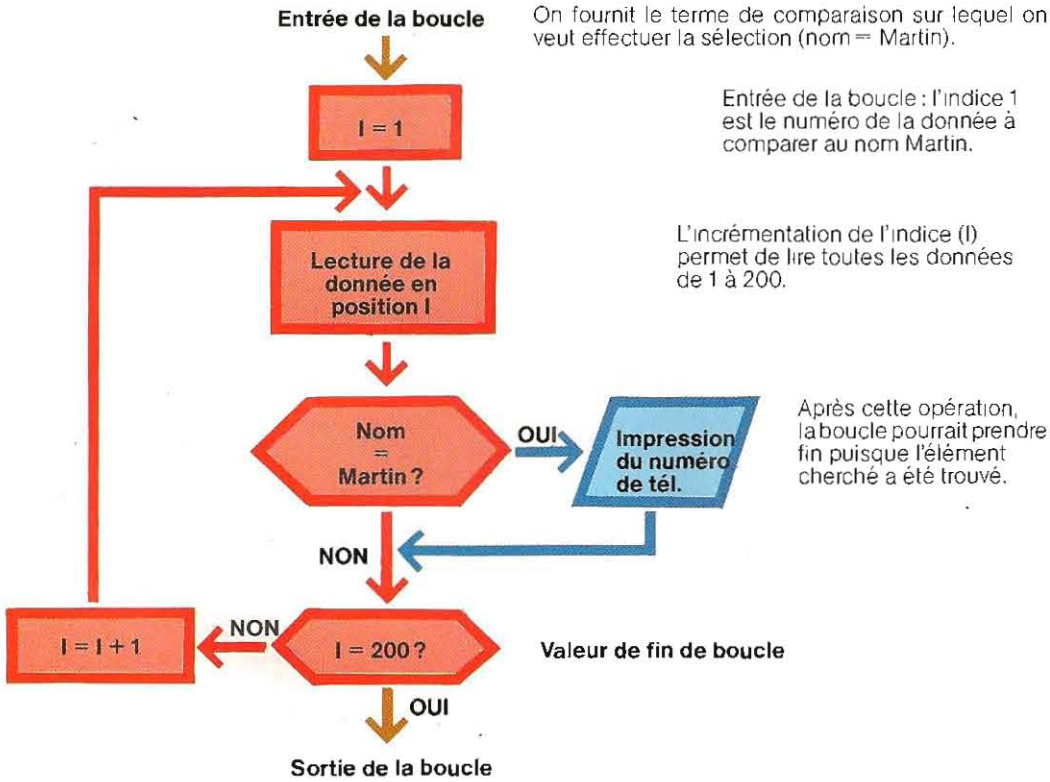


Marka

ORGANIGRAMME D'UNE BOUCLE DE RECHERCHE DE DONNEE

Structure des données utilisée dans l'organigramme. La donnée recherchée (Martin, tél. = 123) porte le numéro 3.





On fournit le terme de comparaison sur lequel on veut effectuer la sélection (nom = Martin).

Entrée de la boucle : l'indice 1 est le numéro de la donnée à comparer au nom Martin.

L'incrémentation de l'indice (I) permet de lire toutes les données de 1 à 200.

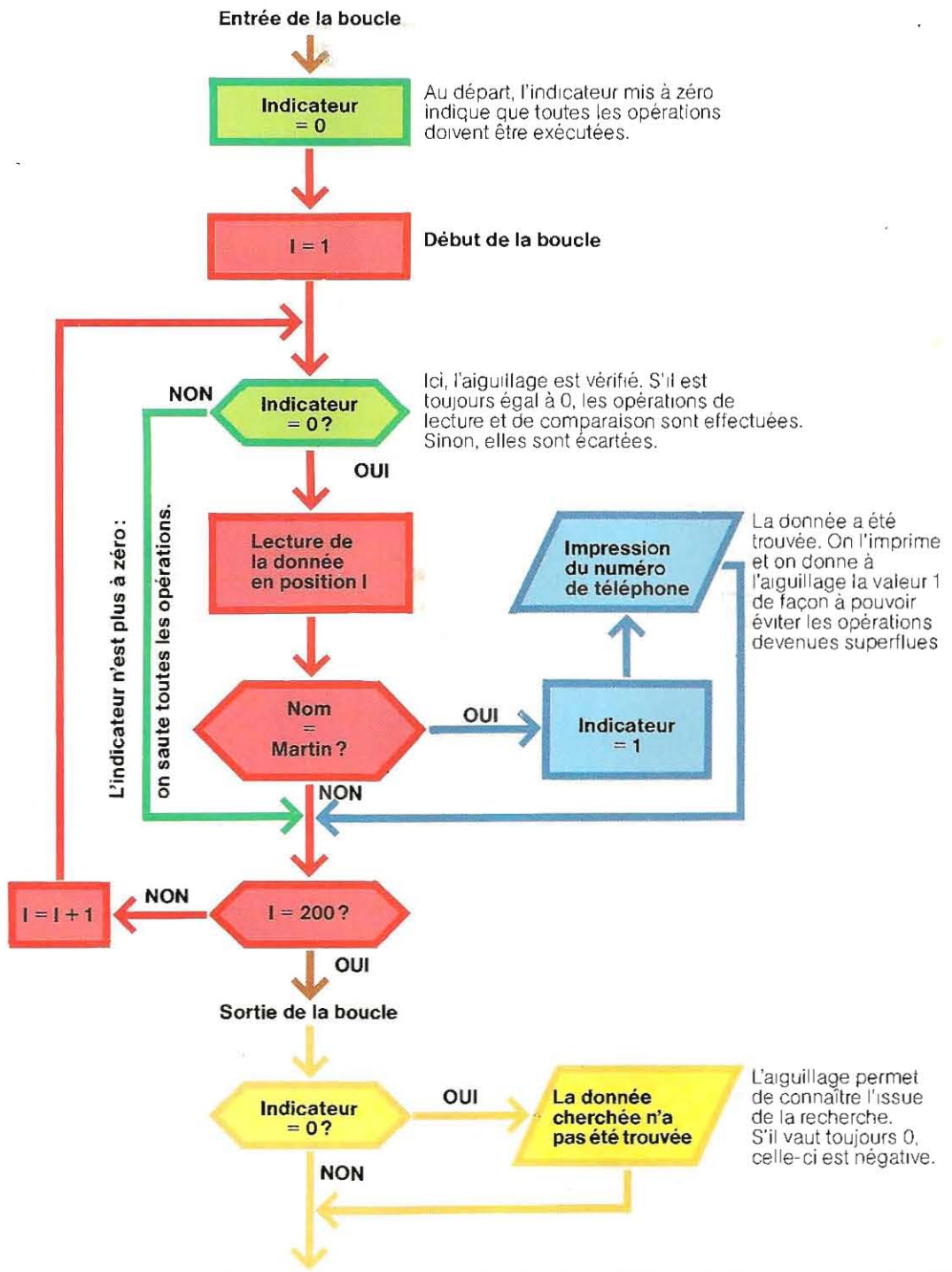
Après cette opération, la boucle pourrait prendre fin puisque l'élément cherché a été trouvé.

Valeur de fin de boucle

Dans cette organisation de boucle, on ne peut modifier l'enchaînement des opérations : la recherche se poursuit donc jusqu'à la fin, même si la donnée a déjà été trouvée.

L'organigramme de la page 192 illustre les modifications apportées par l'introduction d'un aiguillage. L'issue de la recherche est tout de suite connue grâce à l'impression (sur l'écran, ou sur papier à la sortie de l'imprimante) du numéro de téléphone. Si un programme ne prévoit pas cette impression, des erreurs peuvent se produire dans l'exécution des calculs suivants, en cas d'insuccès de la recherche. L'indicateur permet donc de subordonner les opérations suivantes au succès de la recherche.

ORGANIGRAMME D'UNE BOUCLE DE RECHERCHE DANS LAQUELLE UN INDICATEUR REDUIT LES TEMPS EN ELIMINANT LES OPERATIONS SUPERFLUES





E Ferorelli/Life-Grazia Neri

C'est souvent grâce aux jeux vidéo que l'ordinateur a fait son apparition dans les foyers.

En d'autres termes, l'aiguillage sert d'indicateur et permet de choisir le sens du déroulement du programme.

Utilisation des organigrammes

Les méthodes d'analyse et l'emploi de l'organigramme pour la résolution de problèmes spécifiques ne concernent pas uniquement les questions de gestion. On y a également recours pour des applications mineures, telles que l'emploi du micro-ordinateur comme adversaire de jeu.

Ces applications peuvent d'ailleurs poser des problèmes fort complexes et qui ne seront résolus qu'au prix d'une analyse très poussée. Avec l'organigramme qui clôt cette analyse, la rédaction des instructions constituant le programme n'est plus qu'un travail de traduction, sans grand risque d'échec.

A titre d'exemple, nous allons étudier la mise au point d'un programme permettant à la machine de jouer à Strike and Ball, jeu proche du célèbre Mastermind.

Cette étude nous fournira un intéressant modèle d'application.

Avant d'aborder la programmation proprement dite, énonçons les règles du jeu.

Le partenaire électronique (l'ordinateur) génère un nombre constitué de quatre chiffres aléatoires et que l'adversaire doit deviner en dix essais au maximum.

Après chaque essai, l'ordinateur donne l'une des trois réponses suivantes :

NUL, si le joueur n'a pas deviné un seul chiffre.

STRIKE, si le joueur a deviné des chiffres, mais pas leur position. Dans ce cas la machine doit également dire combien de chiffres ont été devinés.

BALL, si le joueur a deviné à la fois la valeur et la position de certains chiffres. Dans ce cas, la machine doit également donner le décompte de BALL.

La réponse peut être deux STRIKE et un BALL, ce qui veut dire que sur les quatre chiffres générés par l'ordinateur, le joueur a deviné la valeur de deux d'entre eux, mais non leur position, et la valeur et la position d'un troisième.

Les circuits intégrés

Aujourd'hui, les circuits intégrés sont presque exclusivement fabriqués avec du silicium, mais leur construction fait appel au moins à huit technologies fondamentales, présentant chacune, à côté de nombreux avantages, des inconvénients inévitables.

Ces huit technologies se répartissent en trois groupes principaux, selon qu'il s'agit de circuits intégrés bipolaires, MOS, ou au contraire de pastilles dans lesquelles le silicium est remplacé par l'arséniure de gallium.

Les semi-conducteurs les plus répandus aujourd'hui sont au silicium, et leur construction se fait selon deux principes essentiels. D'une part, nous trouvons les composants bipolaires, dans lesquels le courant est divisé en deux flux dont chacun circule vers un pôle électrique différent. D'autre part, nous avons les composants MOS, ou plus précisément MOSFET (Metal Oxide Semiconductor Field Effect Transistor), ce qui signifie transistor en métal oxyde semi-conducteur à effet de champ : ce sont des composants dans lesquels un seul flux de courant est exploité pour obtenir l'effet transistor.

Les puces, ou pastilles, bipolaires peuvent être construites selon trois techniques différentes, mais très comparables : TTL, ECL, I_2L.

- *Les semi-conducteurs de la famille TTL (Transistor-Transistor Logic) sont peut-être les plus anciens et les plus largement utilisés ; offrant une vitesse de commutation* élevée (comprise entre 1,5 et 3 nanosecondes**), ils présentent l'inconvénient d'une forte consommation d'énergie, ce qui provoque aussi un échauffement notable. Dans les circuits complexes, quand des centaines ou des milliers de puces fonctionnent en même temps, ce phénomène peut nécessiter des installations spéciales de refroidissement des machines.*

Dan Silversmith, responsable du développement des composants au silicium auprès du Lincoln Laboratory du Massachusetts Institute of Technology, a qualifié les composants TTL de « chevaux de trait » de la technologie bipolaire, tout en reconnaissant que la demande du marché est encore élevée pour des dispositifs qui remontent à une quinzaine d'années, bien que leurs constructeurs soient aujourd'hui peu nombreux.

Le principe TTL présente l'inconvénient d'une technique de fabrication compliquée. Pour mettre au point un circuit intégré TTL, il faut construire en moyenne dix à douze masques superposés (les masques sont la reproduction photographique des schémas de liaison entre les diverses parties du semi-conducteur). La microplaquette est construite par superposition de plusieurs masques. Plus le nombre de masques est grand, plus la construction est complexe, et plus la probabilité d'erreur, ou de défaut du produit fini, augmente, entraînant un rendement plus faible et une élévation des coûts.

- *La technologie ECL (Emitter Coupled Logic, logique à couplage par émetteurs) est voisine de la TTL, mais les puces ECL ont une vitesse de commutation sensiblement plus élevée, avec des valeurs comprises entre 0,4 et 1 nanoseconde. Elle convient donc aux composants, tels que les mémoires, qui exigent des vitesses de fonctionnement très élevées. Un autre avantage de la technique ECL est la possibilité de miniaturisation qu'elle offre. C'est en fait la seule technologie bipolaire qui puisse être utilisée pour les circuits à très forte densité d'intégration (VLSI, Very Large Scale Integration) de la toute dernière génération, où le nombre de composants par unité de surface est considérable. La famille TTL ne convient que pour les puces à densité d'intégration moyenne MSI (Medium Scale Integration), deux niveaux au-dessous du VLSI. Toutefois, la consommation des puces ECL reste plus élevée que celle des autres familles.*
- *I_2L. Cette technologie bipolaire très récente a permis de réduire en partie le problème de la consommation. Mais en gagnant sur le plan de la consommation et des dimensions, on a perdu sur celui de la vitesse : les puces I_2L sont les plus lentes, avec des vitesses de commutation comprises entre 2,5 et 10 nanosecondes. Cet inconvénient est en partie compensé par une construction plus facile. La technique I_2L exige de huit à dix masques, soit moins que TTL et ECL (9 à 11).*

La technologie MOS (ou MOSFET) est plus simple que la technologie bipolaire et se

---

*La vitesse de commutation détermine la vitesse maximale à laquelle le composant peut fonctionner (par exemple la vitesse maximale d'exécution d'une opération logique).
** 1 nanoseconde = 10^{-9} seconde = 0,000000001 seconde.

contente d'un nombre de masques toujours inférieur à dix. De plus, elle consomme moins d'énergie. Il en existe trois types fondamentaux : NMOS, PMOS et CMOS. Dans les deux premiers, les lettres N et P indiquent le signe des charges électriques (respectivement négatives et positives) qui circulent dans le canal.

La technologie CMOS, où C signifie complémentaire, utilise des circuits doubles, dont chaque composant est formé d'un transistor PMOS et d'un transistor NMOS. Les charges étant de signes différents, lorsqu'un transistor fonctionne, l'autre est inactif, de sorte qu'en condition statique, la consommation est pratiquement nulle.

Les NMOS présentent l'avantage des dimensions. Cet avantage leur donne un très bon rapport nombre de circuits intégrés - surface de la pastille (puce) et permet de réunir beaucoup plus de fonctions que dans les autres technologies. De ce fait, les circuits NMOS sont moins coûteux, et sont aussi les plus rapides de la famille MOS, ce qui les rend particulièrement attrayants sur le marché de la grande distribution. L'inconvénient est leur consommation, supérieure à celle des CMOS, car ils exigent une plus forte alimentation statique. Mais ces consommations restent très largement inférieures à celles des puces bipolaires.

Les PMOS ont été les premiers circuits MOS produits, et sont aujourd'hui moins chers que tous les autres, car le nombre de masques nécessaires à leur production est plus réduit : six à huit en moyenne. Leurs dimensions, supérieures à celles des NMOS, et leur vitesse sont leur principal inconvénient. Les PMOS sont sans aucun doute les puces les plus lentes, et leur consommation ne vient pas contrebalancer cette caractéristique.

Les circuits CMOS sont certainement les plus intéressants de la famille, car les plus rapides et les plus efficaces, mais ils coûtent cher. En raison de leur faible consommation, ils sont utilisés dans des applications spatiales.

Les CMOS présentent un inconvénient technique : dans certaines conditions, les éléments du circuit ont tendance à se bloquer, et on ne peut le faire changer d'état qu'en coupant l'alimentation.

Pour pallier cet inconvénient, il faut améliorer l'isolation entre les transistors PMOS et NMOS de chaque élément du circuit, d'où une augmentation du coût de production.

L'emploi du silicium comme base et comme élément porteur n'est pas la seule solution pour la construction de semi-conducteurs. Il en existe d'autres, et le remplacement du silicium par l'arséniure de gallium semble bien être une possibilité d'avenir.

Ce composé, de formule chimique GaAs, permet de produire des transistors très rapides, mais une plaquette de GaAs coûte cent fois plus cher qu'une plaquette de silicium ! De plus, on a du mal à obtenir la croissance des cristaux de GaAs sous forme cylindrique, forme naturelle des cristaux de silicium et qui se prête mieux que toute autre à l'utilisation industrielle. Ce produit pourrait d'ailleurs faire naître une autre difficulté. Le GaAs est obtenu par combinaison de deux éléments, le gallium et l'arsenic, dont le second est très toxique. Son utilisation en quantités importantes comporte donc des risques qui n'existent pas avec le silicium.

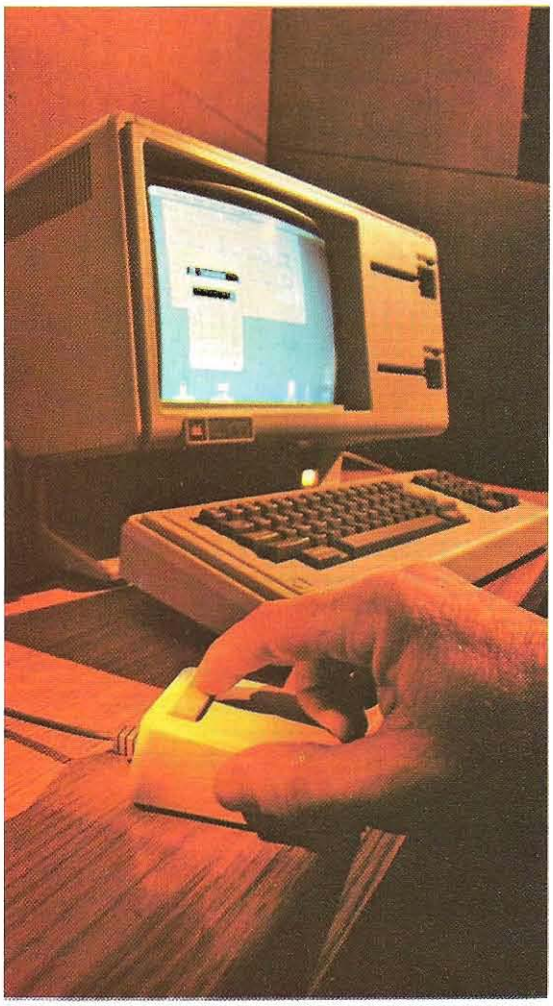
Une solution permettant d'obtenir les effets désirés (consommation réduite et vitesse élevée) nous est offerte, non par l'application d'une technologie nouvelle comme celle du GaAs, mais par la mise en œuvre dans des conditions de milieu différentes. L'immersion des circuits dans un bain d'hélium liquide permet de les porter à une température très proche du zéro absolu, à laquelle on voit apparaître le phénomène de la supraconductivité (découvert en 1911). Les divers éléments du circuit tendent à perdre à peu près totalement leur résistance passive, et peuvent alors devenir très rapides.

Les circuits capables de fonctionner dans ces conditions sont dits « à effet Josephson », du nom d'un chercheur qui étudiait la supraconductivité en 1962. Un circuit à effet Josephson peut commuter en environ 100 picosecondes (1 picoseconde = 10^{-12} seconde), donc à une vitesse dix à cent fois supérieure à celle des circuits normaux. IBM a déjà réalisé des prototypes de processeurs à circuit Josephson, mais d'après Silversmith, il faudra encore au moins dix ans avant que ces produits n'arrivent sur le marché. Et en dix ans, il peut se passer tant de choses.

(D'après « la Jungle des sigles » dans la revue COMPUTER, n° 53, octobre 1982.)

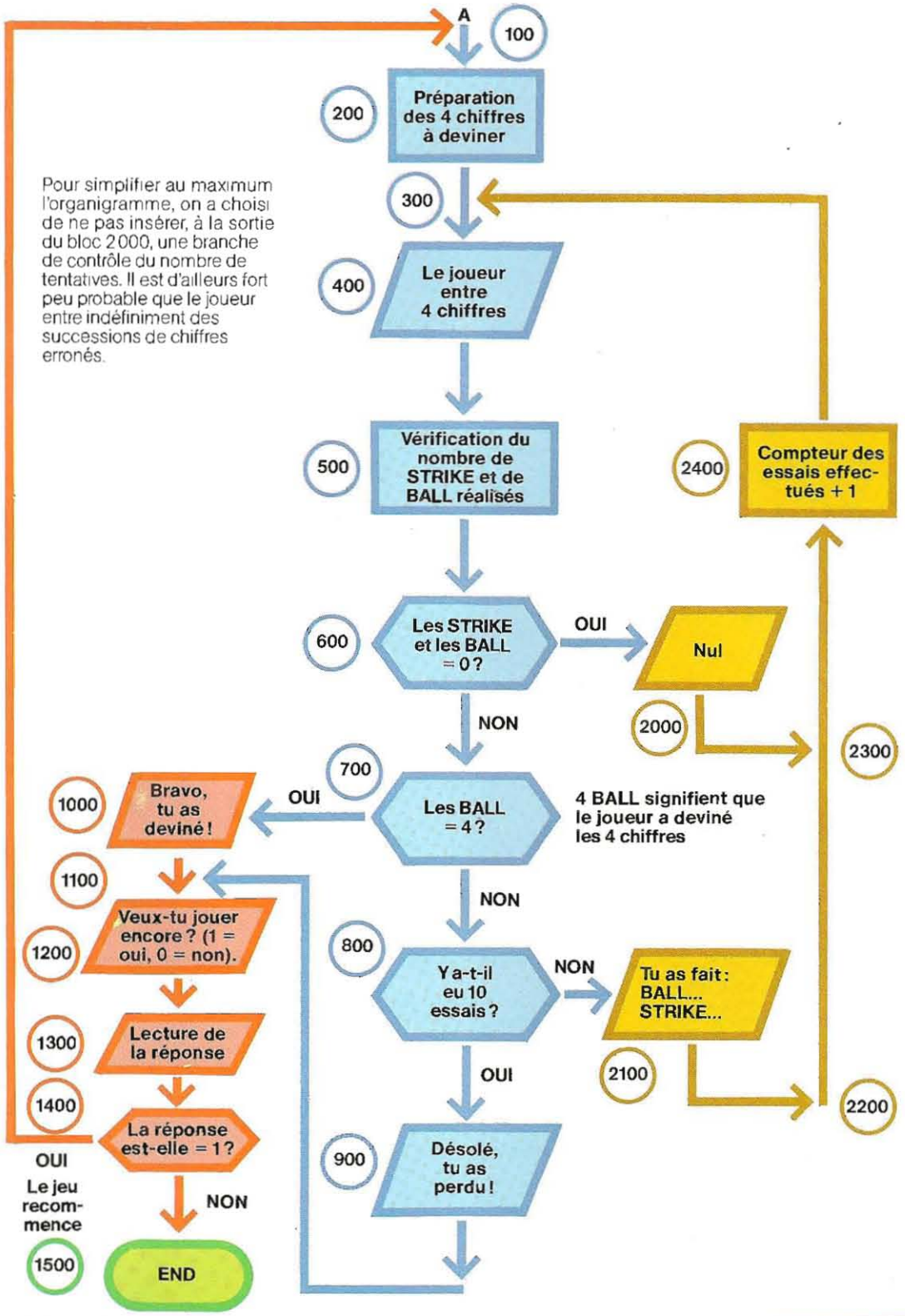
Exposer les règles du jeu revient à faire l'analyse du problème.
Cette méthode très simple, qui consiste à énumérer de manière claire, détaillée et rigoureusement ordonnée les fonctions à exécuter, permet l'analyse et la résolution de n'importe quel type de problème. C'est le point de départ de la préparation de tout organigramme.
La traduction de cette schématisation en symboles d'organigramme constitue le premier niveau d'analyse. Le schéma page 197 représente l'organigramme correspondant.
Il est très utile, du moins au début, de faire figurer à côté de chaque symbole des «étiquettes» numériques: cela facilite la lecture du schéma et permet d'écrire plus rapidement le programme.
Sur l'organigramme de la page 197 on distingue trois colonnes principales: la première, ponctuée de nombres inférieurs à 1000, est le flux principal; la deuxième, dont la numérotation s'échelonne de 1000 à 1500, est le flux qui termine la partie en cours et en entame une autre; la troisième, dont la numérotation va de 2000 à 2400, est le flux des réponses que la machine doit fournir.
C'est ainsi par exemple que le bloc 600 correspond à une décision. A ce stade, la machine vérifie l'existence de la situation: «nombre de STRIKE = 0 et nombre de BALL = 0». Si telle est la situation, le joueur n'a réalisé aucun point. On notera que la phrase «nombre de STRIKE = 0 et nombre de BALL = 0» équivaut à «STRIKE = 0 ET BALL = 0» où l'opérateur logique ET est adopté pour vérifier la simultanéité de deux situations. Ecrit de cette manière, l'organigramme rassemble bien toutes les fonctions du programme, mais ce programme est «squelettique», peu amusant et inutilisable si le joueur ne connaît pas les règles.
Avant d'entrer dans le corps principal de l'organigramme, qui commence en A et se termine avec FIN (END, bloc 1500), il faut une partie descriptive que l'ordinateur présentera au joueur et sur sa demande, pour l'informer sur les règles du jeu.
Cette partie d'information préalable sur les objectifs du programme et la façon de l'utiliser est indispensable, quelle que soit l'application envisagée.
Il est difficile que l'utilisateur d'un programme (ou le programmeur lui-même) puisse se souvenir exactement de tous les paramètres à fournir et de toutes les fonctions du programme. Cette partie descriptive initiale sert justement d'aide-mémoire et facilite la tâche de quiconque utilise le programme.
Le schéma de la page 197 comprend deux blocs (200 et 500) représentant des fonctions complexes, mais rien n'indique comment se déroulent ces fonctions. C'est ce qui sera explicité dans les organigrammes suivants. En renvoyant ainsi les explications détaillées à une phase ultérieure, on évite de compliquer l'organigramme principal, et surtout on se réserve la possibilité de mieux développer (à part) les fonctions les plus complexes. Cela permet de les écrire en termes généralisés et d'en faciliter l'utilisation pour d'autres applications. Cette technique constitue la règle d'or de la programmation.

Cet ordinateur personnel affiche sur son écran le menu d'une application.



C. O'Rear/West Light - Grazia Neri

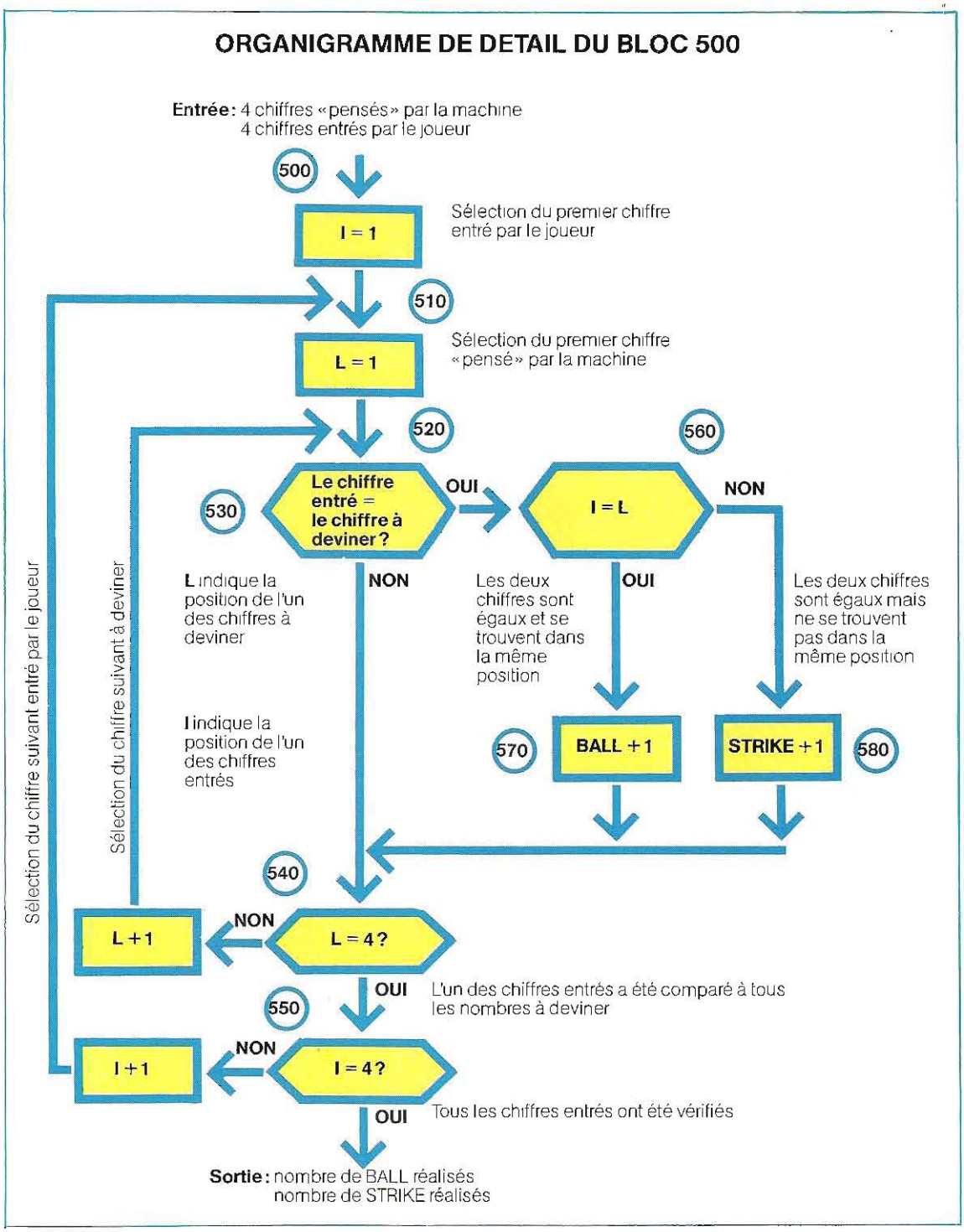
ORGANIGRAMME DE PREMIER NIVEAU POUR LE JEU STRIKE AND BALL



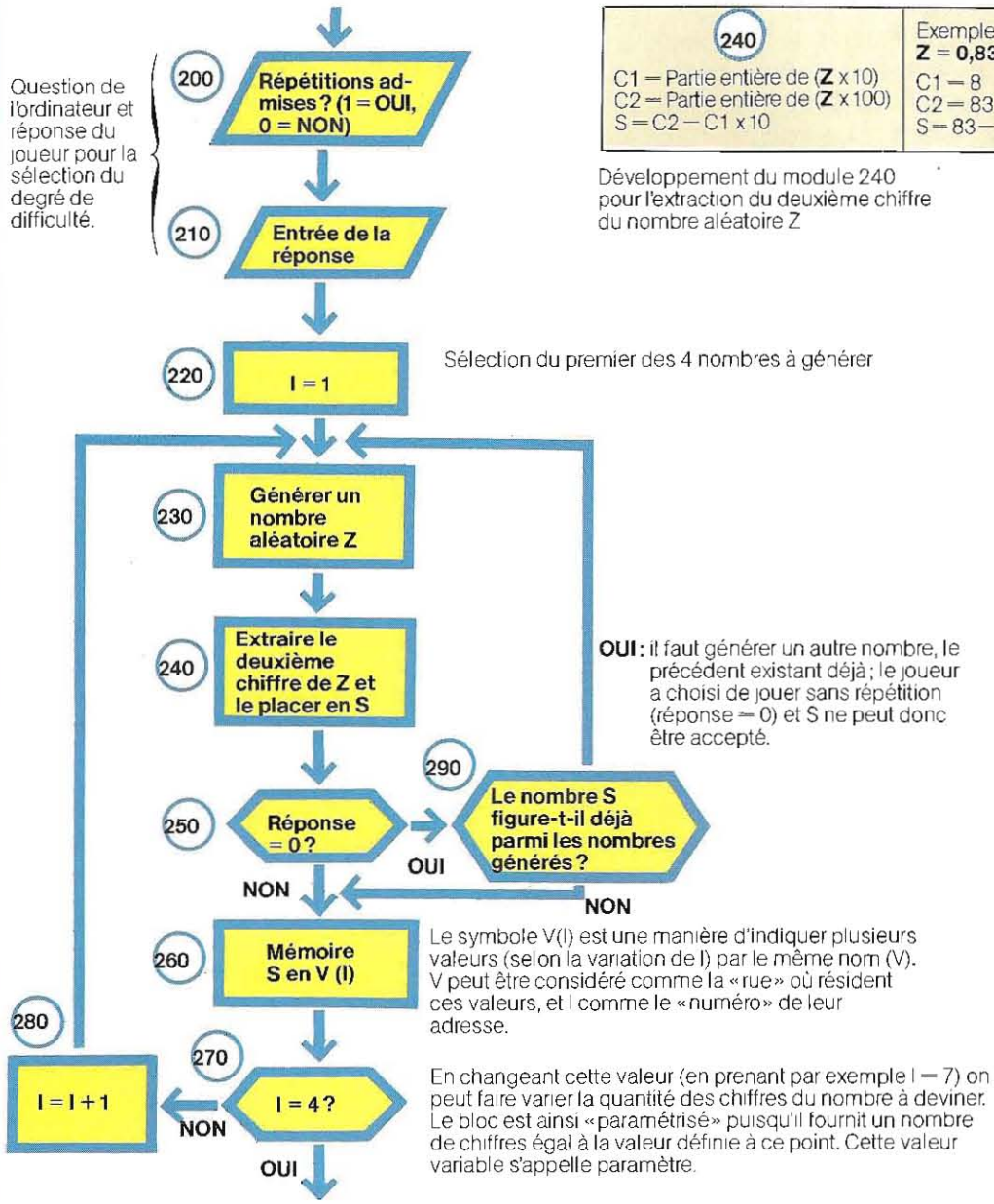
Pour simplifier au maximum l'organigramme, on a choisi de ne pas insérer, à la sortie du bloc 2000, une branche de contrôle du nombre de tentatives. Il est d'ailleurs fort peu probable que le joueur entre indéfiniment des successions de chiffres erronés.

Le schéma ci-dessous représente l'organigramme du module qui compte les points réalisés pour STRIKE et pour BALL (bloc 500 sur l'organigramme de la p. 197).
Toutes les indications numériques de ce bloc doivent être comprises entre 500 (valeur attribuée sur l'organigramme principal) et 590.
Pour compléter les organigrammes du programme, il ne reste qu'à développer le module désigné par le nombre 200 (voir schéma p. 199) et qui a pour objet de préparer les quatre chiffres à deviner.

ORGANIGRAMME DE DETAIL DU BLOC 500



ORGANIGRAMME DE DETAIL DU BLOC 200



La fonction principale de ce module est de générer des nombres de manière aléatoire, c'est-à-dire des nombres n'ayant aucune relation entre eux. Précisons ici que les ordinateurs sont dotés d'une instruction particulière permettant de générer un nombre aléatoire compris entre 0 et 1. Nous reviendrons ultérieurement sur la nature réelle du nombre ainsi obtenu et sa différence par rapport à un véritable nombre aléatoire.

Dans cette application, le résultat final consiste à obtenir un nombre de quatre chiffres,

c'est-à-dire le nombre à deviner. Ce résultat pourrait être obtenu simplement en isolant les quatre premiers chiffres du nombre aléatoire, mais nous avons choisi la méthode la plus longue qui consiste à générer quatre nombres, puis à isoler dans chacun d'eux l'un des chiffres à deviner.

Si le programme obtenu est plus long il a toutefois l'avantage de permettre de vérifier plus facilement s'il existe dans le nombre à deviner une duplication de chiffres.

Supposons que la machine soit capable de fournir un nombre aléatoire de six chiffres compris entre 0 et 1. Un seul de ces chiffres nous intéressant, il nous faudra multiplier le nombre entier par 10 et en extraire la partie entière.

Prenons par exemple, le nombre aléatoire 0,837915. En le multipliant par 10, on obtient 8,37915, et en en extrayant la partie entière, on obtient 8. De même, si nous voulons extraire le deuxième chiffre (3) au lieu du premier, il nous faut :

1 / extraire le premier chiffre comme dans l'exemple précédent (ce qui donne 8) ;

2 / multiplier le nombre aléatoire par 100 (0,837915 × 100 = 83,7915) et extraire la partie entière (ce qui nous donne 83) ;

3 / multiplier par 10 le nombre extrait à la première phase (8 × 10 = 80) ;

4 / calculer la différence entre le nombre extrait à la deuxième phase et le nombre calculé à la troisième phase (83 − 80 = 3) ; le résultat est le deuxième chiffre du nombre aléatoire initial.

Le chiffre ainsi extrait du nombre aléatoire est l'un des quatre qui composent le nombre à deviner ; en répétant quatre fois la procédure et en stockant les quatre résultats en mémoire, nous obtiendrons le nombre complet. Il faudra à ce moment décider du degré de difficulté que nous voulons donner au jeu : il s'agit dans ce cas d'admettre ou non les répétitions des valeurs entre les quatre chiffres à deviner. La question est posée au début du jeu et la décision incombe au joueur.

Le schéma de la page 199 représente l'organigramme du module 200, développé pour composer le nombre à deviner en extrayant le deuxième chiffre d'une série de nombres aléatoires, avec possibilité d'exclure les répétitions à l'intérieur du nombre à deviner.

Pour compléter ce travail, il suffit à présent de traduire les organigrammes en un langage symbolique. Nous verrons ultérieurement comment établir la liste des instructions de ce programme.

Applications scientifiques

Les organigrammes sont également utilisés, quoique moins souvent, pour la résolution de problèmes scientifiques. Ces applications comportent normalement des programmes de structure simple, c'est-à-dire composés d'un nombre de pas relativement réduit. En revanche les formules à adopter peuvent être compliquées.

La structuration du programme perd de sa minutie et c'est la finesse de l'analyse des procédures mathématiques et des séquences de calcul qui devient déterminante. Ce n'est plus un organigramme qui en résulte mais la méthode mathématique de calcul (algorithme) à adopter pour résoudre le problème donné. Dans les programmes de gestion, au contraire, l'algorithme de calcul est toujours déterminé à l'avance.

La principale difficulté des problèmes scientifiques est justement de déterminer la formule qu'il faut appliquer, et de quelle manière. Le déroulement des calculs peut aboutir à des erreurs selon l'ordre dans lequel ils ont été effectués. En effet, l'ordinateur doit arrondir le résultat de chaque opération car les circuits de calcul ne peuvent traiter des nombres dépassant une certaine longueur : la partie excédentaire est ignorée, d'où la possibilité d'erreurs.

Supposons que notre ordinateur puisse traiter des nombres comprenant un maximum de cinq chiffres et que l'on veuille calculer (100 × 0,0002) : 100. Le résultat devrait être 0,0002, puisqu'en multipliant puis en divisant par 100 la même quantité, elle reste invariable.

Développons ce calcul, en effectuant d'abord la multiplication :

100 × 0,0002 = 0,02.

Le résultat peut encore être traité par la machine, puisqu'il comporte un nombre de chiffres inférieur à cinq.

Effectuons la division par 100 :

0,02 : 100 = 0,0002.

On obtient encore un nombre à cinq chiffres qui peut être traité, et le résultat est exact.

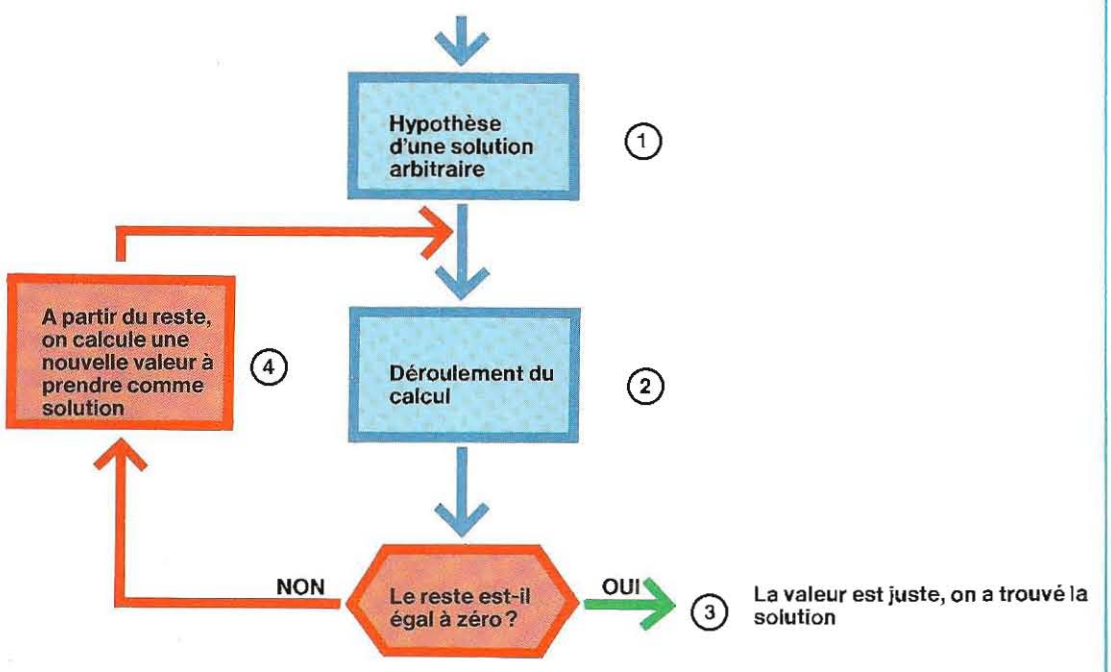
Mais la situation se modifie si nous commençons le calcul par la division :

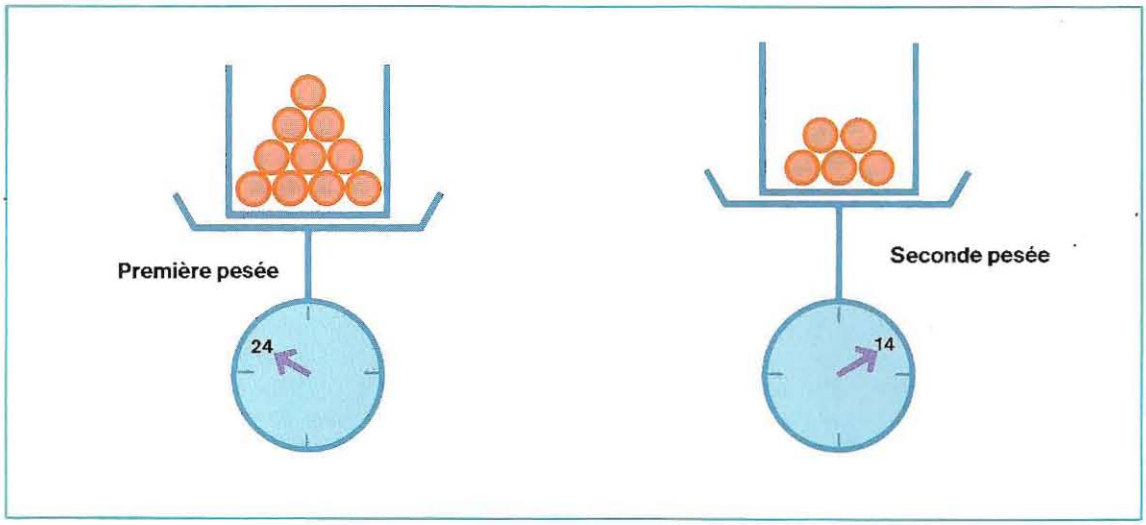
0,0002 : 100 = 0,000002.

Le nombre obtenu dépasse les cinq chiffres, et l'ordinateur ignore automatiquement tous les chiffres au-delà du cinquième.
Le résultat de la division devient 0,0000, et la totalité du calcul sera fausse.
Evidemment les machines ne sont pas aussi limitées ; leur précision courante est très supérieure. Toutefois, il ne faut pas négliger l'éventualité d'erreurs d'arrondi, en particulier, lorsque certains calculs sont répétés un grand nombre de fois : l'accumulation graduelle des erreurs peut conduire à des inexactitudes de résultat, même sur les gros calculateurs, qui sont pourtant des machines extrêmement précises.
C'est la situation qui se présente lors de la résolution des problèmes par la méthode itérative, méthode permettant de trouver la solution d'un problème en donnant initialement à l'inconnue une valeur aléatoire que l'on modifie en fonction du résultat du calcul. Ce principe est illustré ci-dessous.
La première étape consiste à choisir une valeur au hasard (par exemple zéro), la deuxième étape consiste à utiliser cette valeur dans la formule de résolution et à vérifier (troisième étape) si le déroulement du calcul donne bien un zéro pour résultat. Dans ce cas, le calcul est exact et l'hypothèse choisie pour l'inconnue est juste. Dans le cas contraire, il existe un reste (différence entre la valeur prise pour hypothèse et la valeur calculée), qui est utilisé pour formuler une nouvelle valeur d'essai, entrée à son tour dans le module 2 pour une nouvelle vérification. Le processus se poursuit ainsi par approximations successives jusqu'à obtention d'un reste égal à zéro (ou très proche de zéro).
A titre d'exemple, appliquons la méthode itérative à un problème de pesée.
Un récipient, dont on ignore le poids, contient dix boules, dont on ne connaît pas non plus le poids. En plaçant le récipient sur le plateau d'une balance, on constate le poids global (récipient + dix boules) = 24 kg. En retirant cinq boules, on obtient un nouveau poids de 14 kg (récipient + cinq boules). Déterminer quel est le poids du récipient et le poids unitaire des boules.
Appelons **x** le poids du récipient et **y** le poids d'une boule.
La première pesée nous donne :

Récipient + 10 boules = 24 kg
soit : $x + 10y = 24$

SCHEMA DE PRINCIPE DE LA METHODE ITERATIVE





La seconde pesée nous donne :

Récipient + 5 boules = 14 kg
soit : $\mathbf{x} + 5\mathbf{y} = 14$

On trouvera la solution du problème en résolvant le système algébrique suivant, constitué par les deux équations :

$$\mathbf{x} + 10\mathbf{y} = 24$$
$$\mathbf{x} + 5\mathbf{y} = 14$$

Dans ce cas, la résolution du problème relève de méthodes algébriques et l'on obtient (en faisant par exemple la différence entre les deux équations) : $\mathbf{x} = 4$ et $\mathbf{y} = 2$. Vérifions

En cas d'erreur dans un programme d'itération, il arrive que l'on soit littéralement submergé par un océan de papier.



M. Wolf/Black Star-Grazia Neri

cette solution en lui appliquant la méthode itérative.

1 / Attribuons à l'une des inconnues une valeur arbitraire, par exemple $\mathbf{x} = 0$.

2 / Cette valeur est introduite par substitution dans la première équation qui devient :

$$0 + 10\mathbf{y} = 24$$

d'où :

$$10\mathbf{y} = 24$$

$$\mathbf{y} = \frac{24}{10} = 2{,}4$$

3 / La valeur de **y** ainsi calculée ($\mathbf{y} = 2{,}4$) est introduite par substitution dans la seconde équation, d'où l'on peut extraire une nouvelle valeur de **x**. L'équation $\mathbf{x} + 5\mathbf{y} = 14$, avec $\mathbf{y} = 2{,}4$ devient :

$$\mathbf{x} + (5 \times 2{,}4) = 14$$

soit :

$$\mathbf{x} + 12 = 14$$

$$\mathbf{x} = 14 - 12 = 2$$

4 / Cette nouvelle valeur de **x** est introduite dans la première équation de la même manière qu'à l'étape 2 :

$$2 + 10\mathbf{y} = 24$$

$$10\mathbf{y} = 24 - 2$$

$$10\mathbf{y} = 22$$

$$\mathbf{y} = \frac{22}{10} = 2{,}2$$

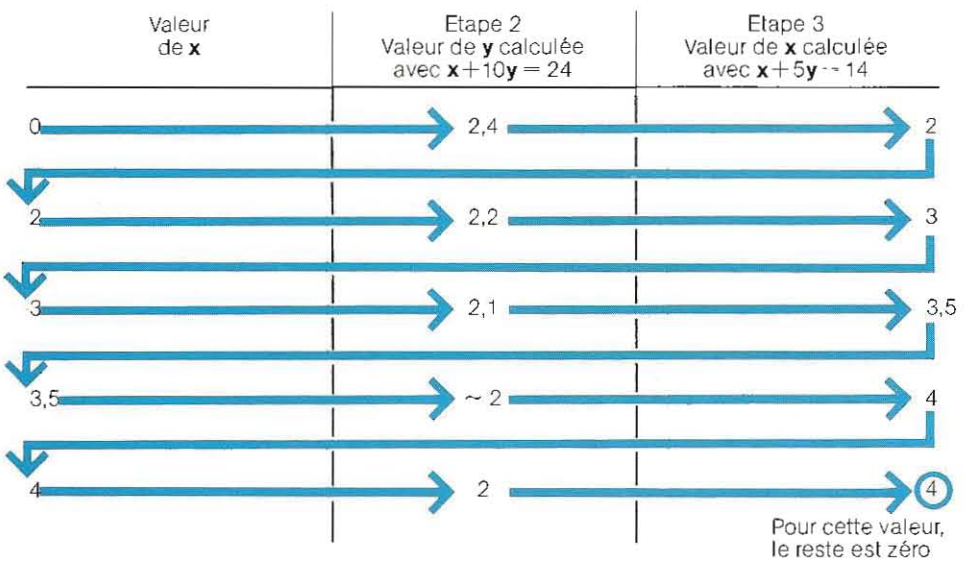
Une série de passages correspondant aux points 2, 3 et 4 fournissent des valeurs qui figurent sur le schéma ci-dessous.

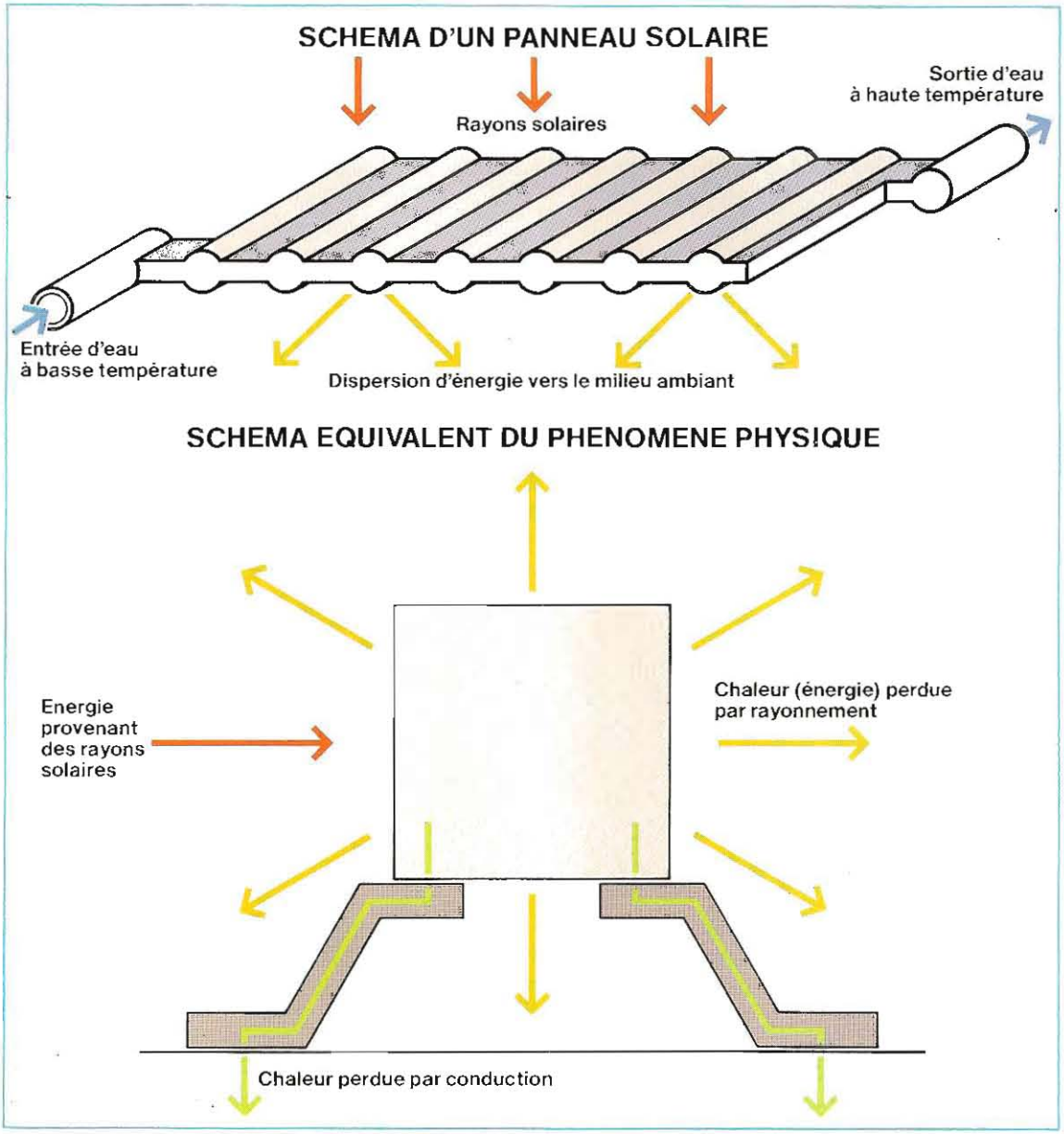
Les deux équations $\mathbf{x} + 10\mathbf{y} = 24$ et $\mathbf{x} + 5\mathbf{y} = 14$ donnent toutes deux zéro si l'on remplace **x** par 4 et **y** par 2. Ces valeurs représentent donc la solution cherchée. Si l'on intervertit les termes du problème, en faisant la première pesée avec cinq boules et la seconde avec dix, la méthode, au lieu de donner, par convergence, des solutions de plus en plus proches, donnera des valeurs négatives divergentes, s'écartant de plus en plus de la solution. Lorsqu'on veut adopter cette dernière méthode, il faut analyser avec beaucoup d'attention la mise en place du calcul, afin d'éviter d'aboutir à la divergence des solutions d'essai.

En fait, le système illustré ici possédant une solution algébrique, rien ne justifie l'utilisation de la méthode itérative pour en chercher la solution. Dans les applications scientifiques, en revanche, on rencontre souvent des problèmes impossibles à résoudre autrement que de cette manière. Le sujet étant vaste, de nombreuses méthodes théoriques, parfois complexes, ont été mises au point pour assurer la convergence. Le paragraphe suivant illustre l'application de la méthode itérative à la résolution de problèmes thermiques.

Applications de la méthode itérative

L'exemple précédent (solution d'un système de deux équations à deux inconnues) n'a été choisi qu'à titre explicatif. En réalité, quelle





que soit leur complexité, les systèmes du premier degré (dans lesquels les inconnues ne sont pas élevées à une puissance), comportant un nombre d'équations égal au nombre d'inconnues, peuvent être résolus par des méthodes algébriques plus rapides et plus sûres que la méthode itérative.

Cette méthode itérative sera réservée aux équations qui ne possèdent pas de solution algébrique. Voici une équation caractéristique de cette espèce :

$$A \times T^4 + B \times T = C$$

L'inconnue est T, tandis que A, B et C sont des valeurs numériques constantes. Avant d'examiner une méthode de résolution de ces équations, voyons à quel domaine elle peut être appliquée.

La civilisation industrielle doit faire face au problème fondamental des sources d'énergie. Les ressources traditionnelles, et surtout le pétrole, semblent en voie d'épuisement, leur coût ne cesse d'augmenter, et elles provoquent également des pollutions. Il est donc urgent d'envisager l'exploitation d'autres sources d'énergie. L'une d'entre elles, renouvelable par excellence, et non polluante,

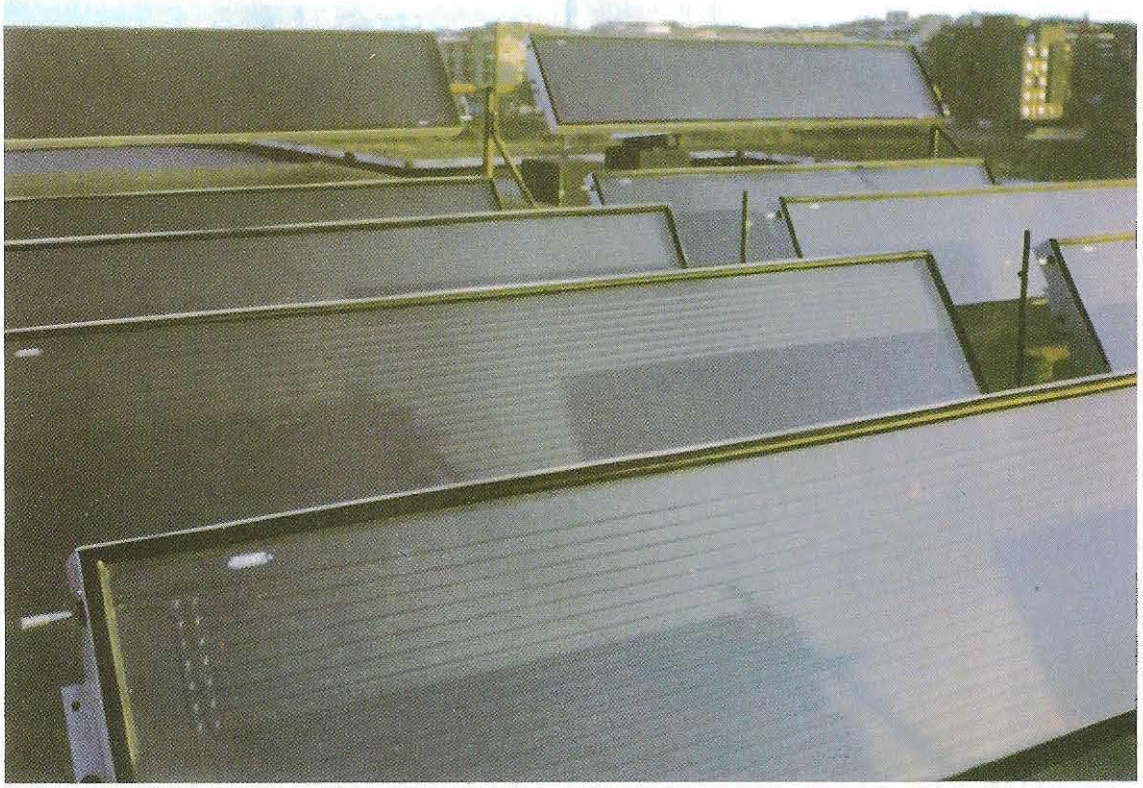
est l'énergie solaire. Nous savons qu'un récipient plein d'eau exposé aux rayons solaires se réchauffe par accumulation de l'énergie du soleil. Une fois l'eau réchauffée, nous pouvons exploiter cette énergie à des fins différentes, et, en particulier, pour les besoins domestiques. C'est déjà une réalité dans plusieurs pays, où sont commercialisés des panneaux solaires destinés à capter l'énergie solaire nécessaire au chauffage d'une maison (chauffage central, eau chaude). Pour que l'installation soit réellement efficace, il faut qu'elle soit bien étudiée et c'est à cette étude qu'on applique la formule exposée. Il est possible de schématiser le phénomène physique pour une première approximation, comme l'indique la figure de la page 204. En haut, on voit le schéma fonctionnel d'un panneau solaire ; en bas, une représentation schématique qui illustre un phénomène physique réel. Supposons que l'eau ne circule pas. Si nous voulons maintenir à une température donnée (T) l'eau contenue (immobile) dans le récipient, il faudra fournir autant d'énergie que le récipient en perd. Si nous en fournissons moins, l'eau refroidira, si nous en fournissons plus, elle s'échauffera plus que nécessaire pour obtenir la température requise. Dans les conditions « stationnaires » d'une eau immobile à température constante, le bilan énergétique est :

énergie fournie = énergie perdue.

L'énergie fournie est la quantité de chaleur apportée par les rayons solaires et dont la valeur est représentée par C. L'énergie perdue est celle que le récipient cède au milieu environnant (déperdition).

Les principales déperditions thermiques se font par conduction et par rayonnement. Les pertes par conduction sont dues à la quantité

Les panneaux solaires exploitent la chaleur du rayonnement solaire pour réchauffer de l'eau. Ci-dessous : une installation de chauffage solaire.



Archives Curcio/CTIP Solar

de chaleur qui migre d'un corps plus chaud vers un corps plus froid mis au contact de celui-ci (par exemple, entre le récipient et les poutrelles qui le soutiennent), et elles sont proportionnelles à la différence de température entre les deux corps.

Déperdition par conduction =
= B × (température du premier corps −
− température du second corps).

Pour plus de simplicité (ce que nous exposons ici n'a qu'un caractère indicatif), nous négligerons la température du second corps par rapport à celle du premier :

Déperdition par conduction = B × T

B étant un facteur lié au type de matériau et à la forme géométrique du système.

Le second type de perte, par rayonnement, est dû à la migration de chaleur d'un corps vers le milieu environnant (plus froid) sous la forme d'ondes électromagnétiques (radiations). Dans ce type d'échange de chaleur, l'énergie transmise est proportionnelle aux différences entre les puissances quatre des températures.

Déperdition par rayonnement =
= A × (température du premier corps)^4 −
− (température du deuxième corps)^4.

Dans ce cas aussi, en négligeant la température du second corps, nous obtenons :

Déperdition par rayonnement = A × T^4

Le facteur A dépend des caractéristiques physiques et géométriques du corps.

L'énergie totale perdue est la somme de ces deux pertes (conduction + rayonnement), soit :

Energie totale perdue = A × T^4 + B × T

Pour maintenir constante la température (T), il faut fournir au récipient une quantité égale d'énergie ; en d'autres termes, l'énergie apportée par les rayons solaires (C) doit être égale à l'énergie totale perdue :

C = A × T^4 + B × T

L'équation obtenue est de forme simplifiée, car elle ne tient pas compte des déperditions par échange de chaleur avec l'air qui entoure le récipient (échange de type convectif). En outre, dans bien des cas, si la valeur de T n'est pas grande, le terme A × T^4 peut être négligé. La formule que l'on en tire, connaissant les valeurs de A, B, C (constantes et dépendant des caractéristiques du récipient), fournit la température (T) d'équilibre à laquelle l'eau se maintient sous l'effet combiné de l'énergie d'entrée et de l'énergie de sortie. Dans la formule, la température (T) n'est pas exprimée en degrés Celsius, mais en degrés Kelvin (le 0° Kelvin se situe à −273,2° Celsius ; une température en degrés Kelvin se calcule donc en ajoutant 273,2 au nombre de degrés Celsius. Comme on le voit, l'utilisation des degrés Kelvin donne des nombres très élevés et, pour faciliter les calculs, on divise habituellement la valeur par 1 000.

TEST 5

1 / Le symbole  représente :
a : une décision ; b : le développement d'un calcul ; c : la fin du programme.

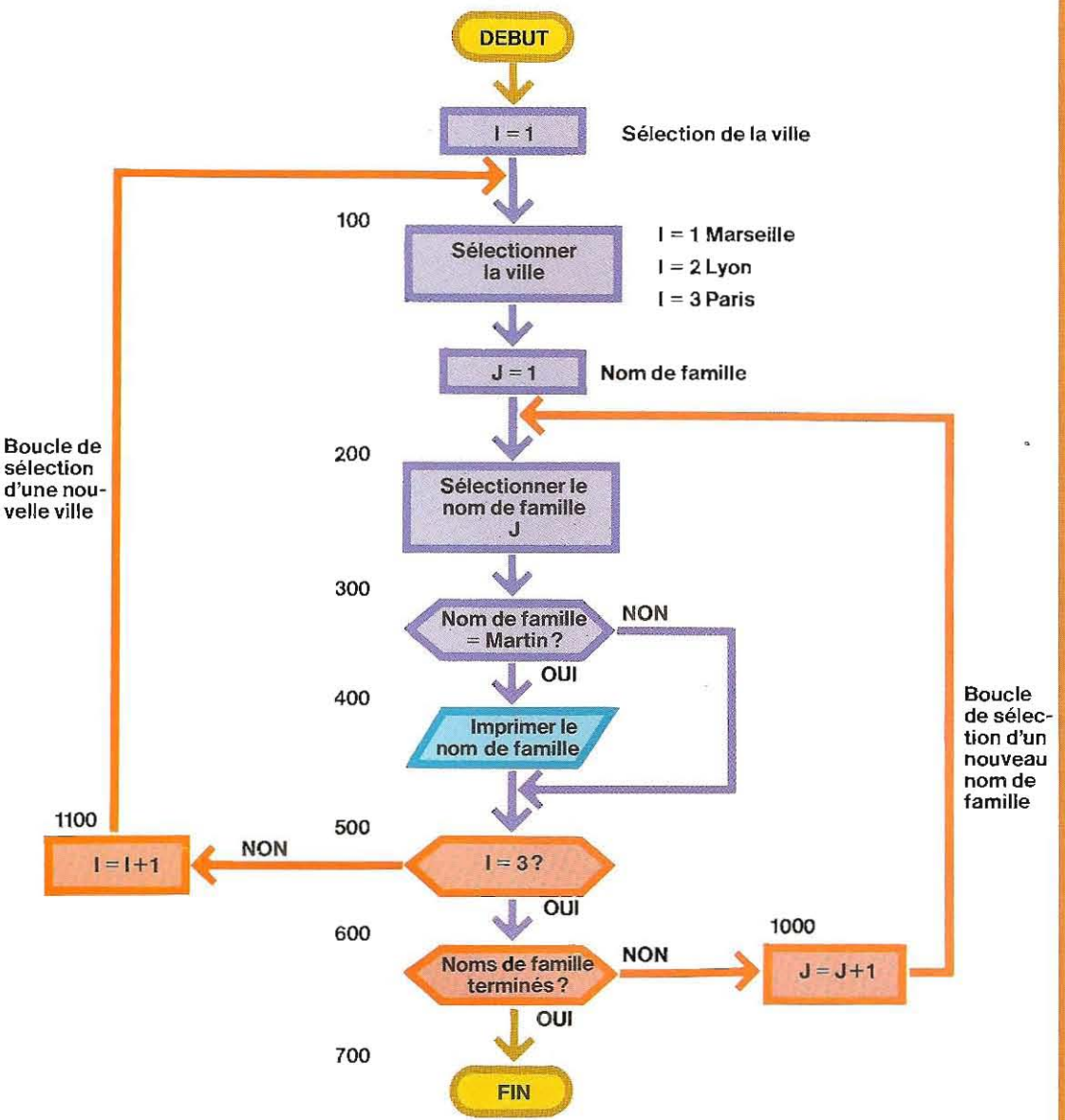
2 / Dessiner les symboles représentant les actions suivantes :
a : traitement général ; b : calcul ; c : impression.

3 / Quelle est l'utilisation du symbole : 

4 / Qu'est-ce qu'un drapeau (flag) :
a : un indicateur ; b : un compteur ; c : un registre de l'unité centrale.

5 / Dessiner l'organigramme d'un programme pour le calcul des pourcentages avec vérification à l'entrée que le pourcentage ne soit pas supérieur à 100.

6 / L'organigramme ci-dessous contient une erreur. Déterminer laquelle, et tracer la forme exacte du schéma. Le problème est le suivant : nous disposons de trois groupes d'adresses, le premier concerne la ville de Marseille, le deuxième la ville de Lyon et le troisième la ville de Paris. Chacun des groupes contient dix adresses. Nous voulons sélectionner toutes les adresses des personnes dénommées Martin (il faut deux boucles, une pour sélectionner la ville, et l'autre pour le nom de famille).



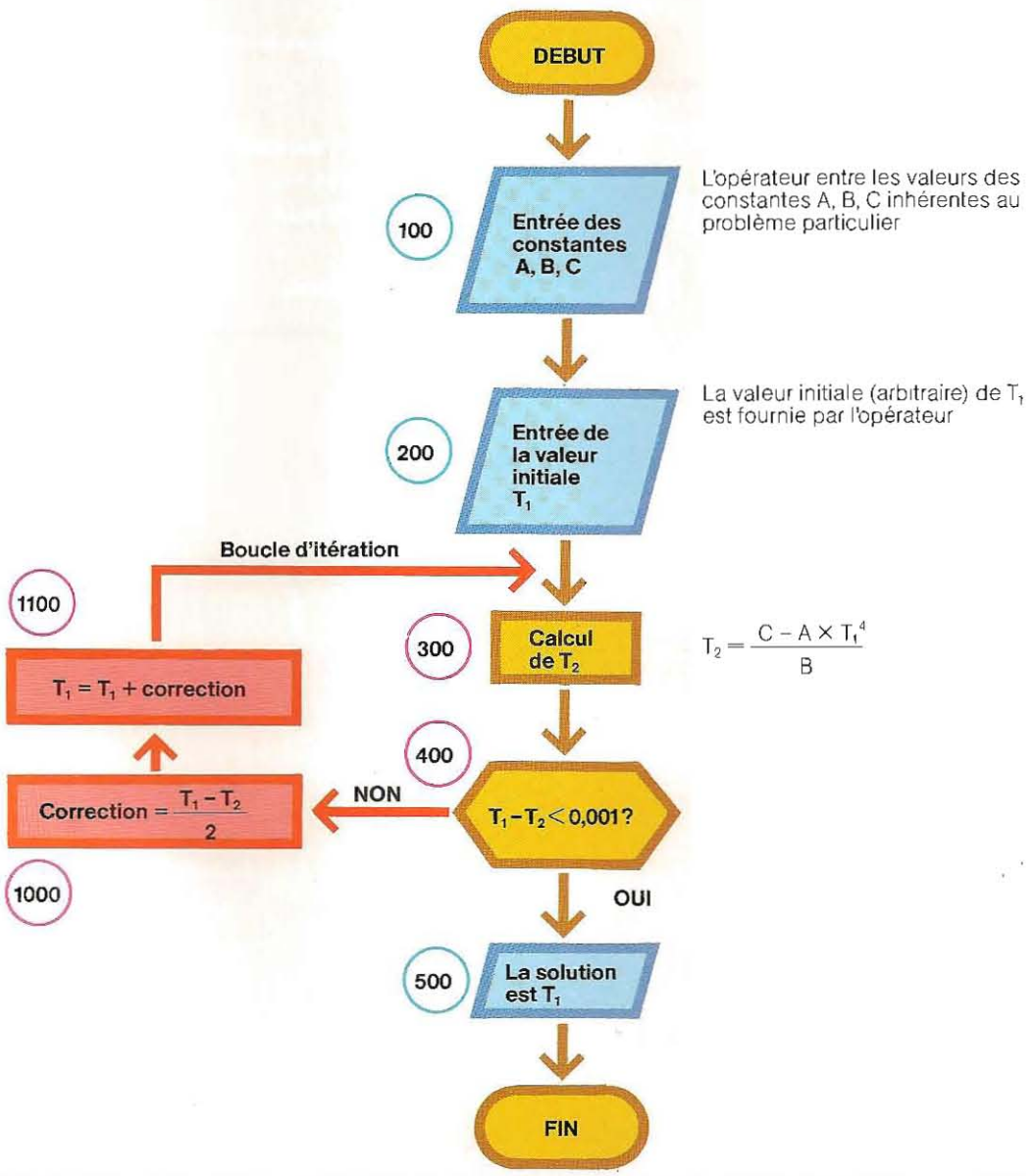
Les solutions du test se trouvent pages 214 et 215.

Tous les calculs se font alors sur de petits nombres. Par exemple, une température de 373,2° K élevée à la puissance quatre (T^4) donne 19 398 428 000, alors qu'en divisant la température par 1000 (c'est-à-dire 0,3732) on obtiendra 0,0193984, que l'on peut arrondir à 0,0194. A la fin du calcul, il faudra évidemment multiplier le résultat par 1000. Cette méthode est également pratique pour les valeurs particulières des constantes utilisées dans les calculs thermiques.

Inscrivons à nouveau la formule du bilan thermique:

C chaleur fournie par le soleil	=	$A \times T^4$ déperdition par rayonne- ment	+	$B \times T$ déperdition par rayonne- ment

ORGANIGRAMME POUR LA RESOLUTION DES EQUATIONS DE TYPE $C = A \times T^4 + B \times T$ PAR LA METHODE ITERATIVE



L'équation peut être réécrite de la manière suivante :

$$T = \frac{C - A \times T^4}{B}$$

La solution T (température à laquelle monte le récipient) s'obtient par la procédure itérative suivante :

1 / On introduit dans le terme $A \times T^4$ une valeur arbitraire de T (par exemple T_1) pour effectuer les calculs indiqués dans le second membre de l'équation.
Le calcul terminé, le premier membre de l'équation fournit une nouvelle valeur de T que nous appellerons T_2 :

$$T_2 = \frac{C - A \times T_1^4}{B}$$

2 / Si la différence entre les valeurs T_1 et T_2 est faible (par exemple 0,001) c'est que l'on a trouvé la solution.

3 / Dans le cas contraire, on calcule pour T une valeur de correction égale à la moyenne des deux valeurs T_1 et T_2 :

$$\text{correction} = \frac{T_1 - T_2}{2}$$

4 / Cette correction est appliquée à la valeur initiale (arbitraire) T_1 qui devient T_1 + correction.

5 / On repart de l'étape 1 avec cette nouvelle valeur de T.

L'organigramme de la page 208 illustre cette méthode.

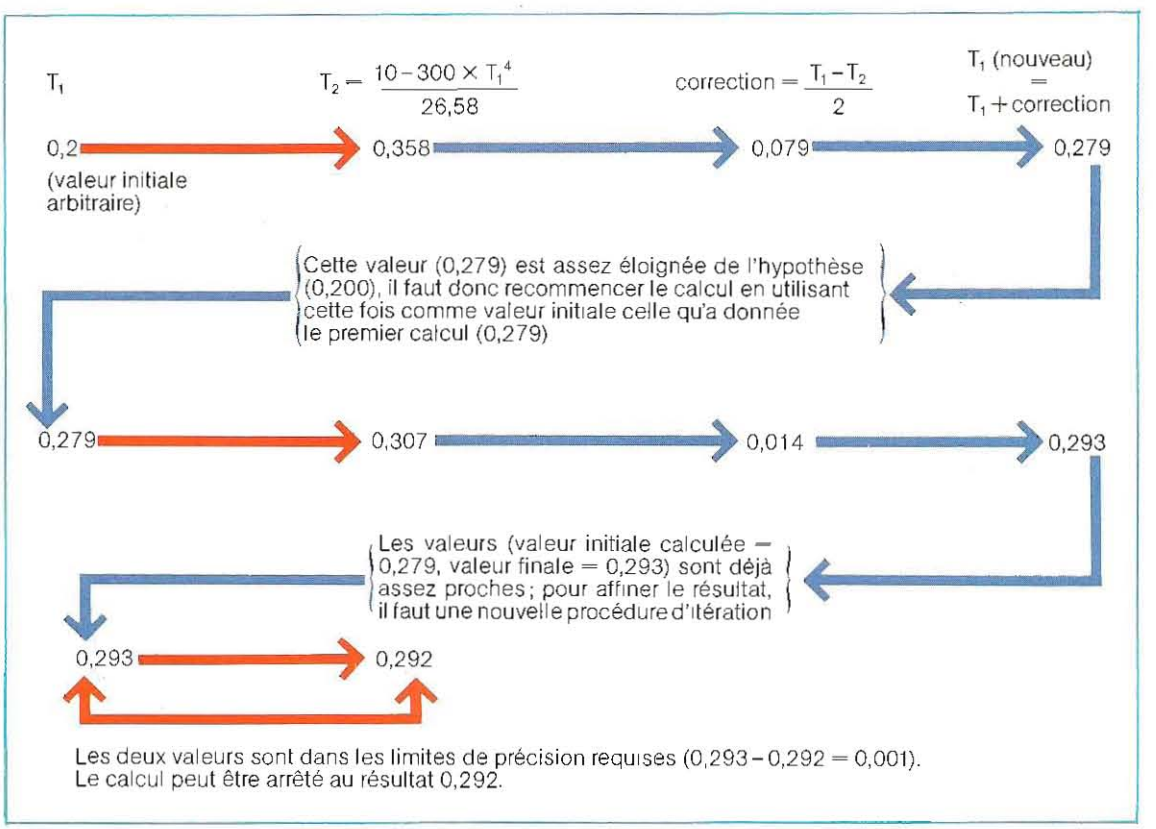
Le tableau ci-dessous illustre le développement d'un calcul avec adoption des valeurs A = 300 ; B = 26,58 ; C = 10 ; ce qui donne pour l'équation :

$$300 \times T^4 + 26{,}58 \times T = 10$$

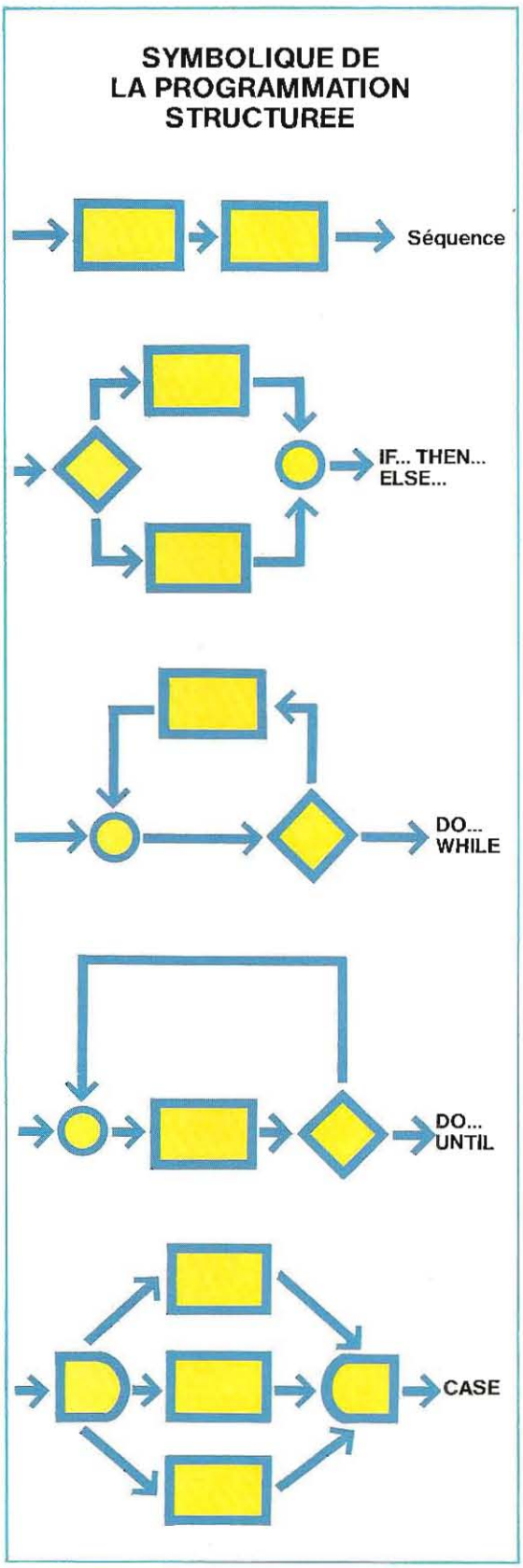
Programmation structurée

Les méthodes appliquées à l'analyse d'un problème, à sa décomposition en un certain nombre de problèmes plus simples, et finalement à l'établissement des organigrammes correspondants, permettent de définir la structure logique qui est valable dans la majorité des cas. Les problèmes plus complexes, toutefois, ne seront pas résolus de cette manière.

On appliquera avec profit la méthode HIPO et la préparation des organigrammes à la pro-



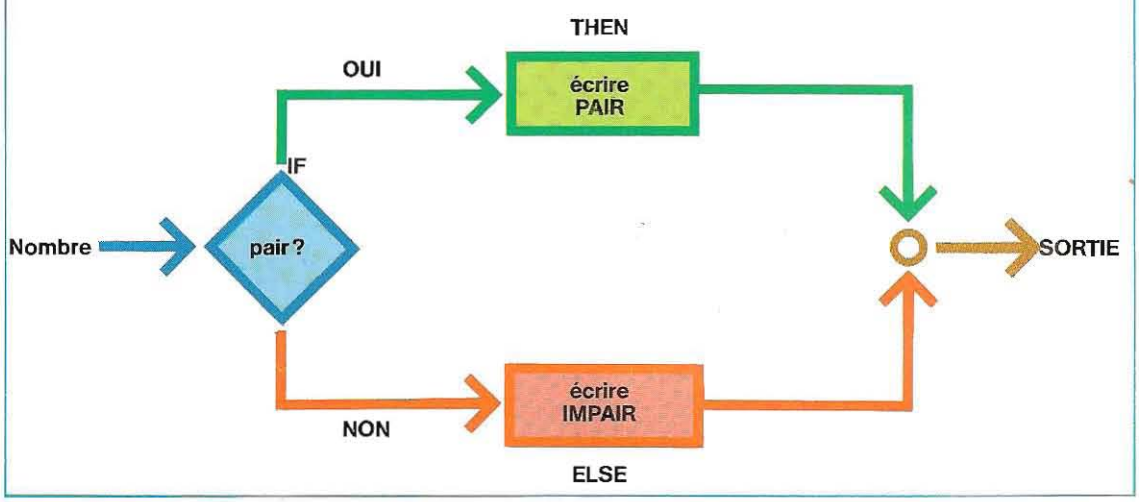
grammation des micro-ordinateurs et des mini-ordinateurs.
Pour les systèmes plus complexes, on aura recours à des techniques plus rigoureuses qui, bien que laissant moins de place à l'interprétation individuelle, sont régies par des règles suffisamment précises pour permettre un travail d'équipe.
La préparation d'un programme pour un micro ou un mini-ordinateur est généralement l'affaire d'une seule personne; le travail est alors assez souple. Mais dès que l'ordinateur devient plus performant et que la complexité des problèmes augmente, le nombre de personnes qui participent à la tâche augmente aussi, en même temps que la nécessité d'une documentation claire et de méthodes normalisées.
Il faut en outre adopter des techniques réduisant les risques d'erreur tout en permettant de gagner du temps.
La plus moderne et la plus sophistiquée de ces méthodes est la **programmation structurée**.
Il s'agit d'une méthode complexe, tout à fait précieuse pour les applications destinées aux gros ordinateurs ou aux mini-ordinateurs de haut de gamme.
Nous aurons l'occasion de revenir sur cette technique de programmation, fondée sur cinq fonctions principales, mais nous nous limiterons ici à en décrire certains des aspects principaux.
Puisque chacun des symboles d'un organigramme signale par lui-même une action élémentaire, un organigramme complet en contient généralement un nombre assez grand.
Au contraire, la programmation structurée ne comporte pas de symboles élémentaires, mais des symboles composés, qui expriment des actions complexes. Par exemple, un organigramme décrivant une boucle à rupture de code le fait à l'aide d'un seul symbole complexe.
Naturellement, cette approche n'a de sens que si l'on dispose, lors de la phase d'écriture du programme, d'un langage approprié. Il serait inutile de préparer un organigramme synthétique s'il fallait ensuite le développer en détails pour le traduire en instructions.
Les langages les plus courants (Basic, Fortran, Cobol) pour les micro-ordinateurs sont



en général peu adaptés à la programmation structurée. Mais il existe d'autres langages, tels le **PL/1**, le **PL/C**, le **PASCAL**, l'**ALGOL**. Relativement récents, ils permettent une notable économie de temps. Il n'est d'ailleurs pas impossible que leur emploi se généralise bientôt (PL/1 et PASCAL sont déjà utilisés pour certains micro-ordinateurs).
Les principales fonctions de la programmation structurée sont illustrées par le schéma de la page 210 et décrites ci-dessous.

Séquence
Le symbole graphique exprime clairement la fonction : une série d'opérations (de quelque type que ce soit) à exécuter à la suite l'une de l'autre.

ORGANIGRAMME D'UNE DECISION REECRITE AVEC LES SYMBOLES DE LA PROGRAMMATION STRUCTUREE



IF... THEN... ELSE...
C'est un symbole composé. Il commence par une décision (test de condition) ; le résultat débouche sur deux voies possibles, l'une à adopter si la condition est vraie, l'autre si la condition est fausse. La sortie est commune.
Le schéma ci-dessus en illustre une application. A l'entrée, le bloc reçoit un nombre et doit inscrire le message « pair », si le nombre est pair ; dans le cas contraire, le message doit être « impair ».
En interprétant à la lettre les phrases que contient ce bloc, on obtient :

IF « le nombre est pair » ; THEN « écrire pair » ;
ELSE « écrire impair ».

Il s'agit donc d'une forme très comparable au symbole de décision habituel.

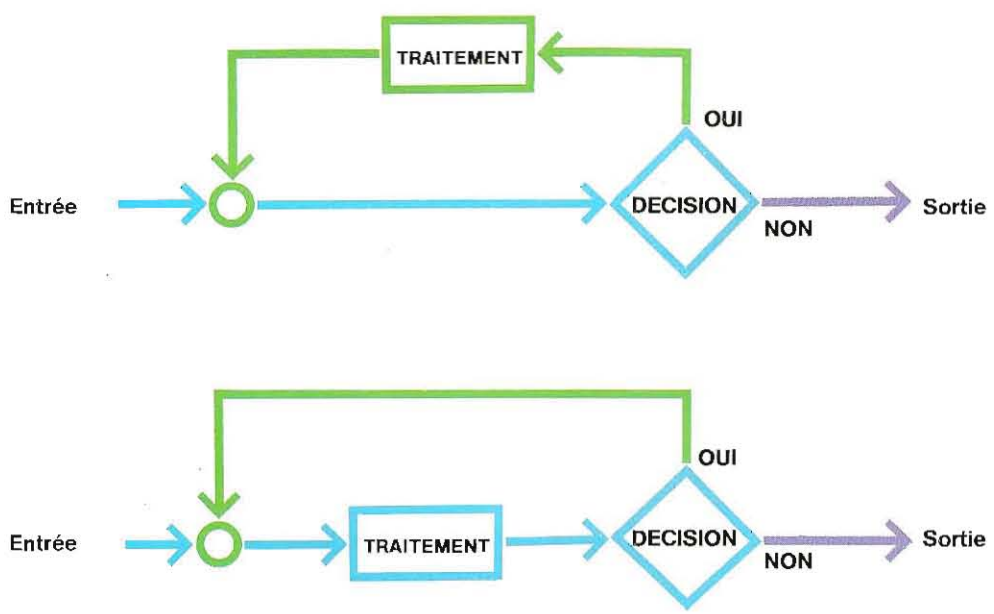
DO... WHILE et DO... UNTIL

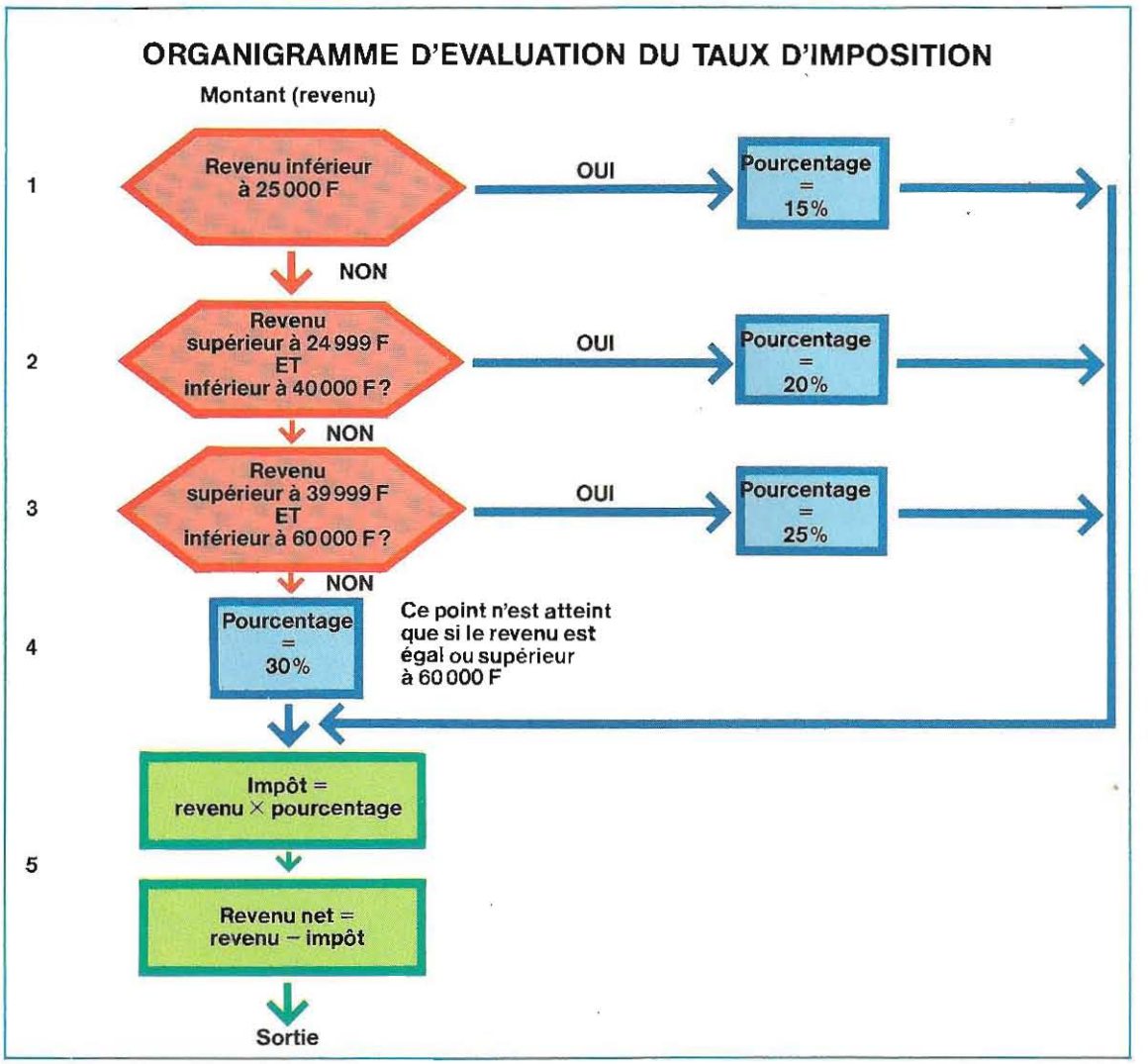
Ces deux blocs sont semblables et contiennent un test (symbole de décision) et un traitement. Dans le premier, la décision est exécutée après l'entrée ; si la réponse est positive, on exécute le traitement et on rentre dans la décision. La sortie est obtenue si la décision donne un résultat négatif. Dans le second bloc, le traitement précède la décision. Il s'agit de deux formes compactes de boucle.

CASE

Ce bloc n'a pas de lien immédiat avec la logique symbolique des organigrammes. Il signifie une comparaison multiple ; pour obtenir le même résultat avec des symboles de décision, il faudrait en utiliser autant que de voies possibles. Voici un exemple d'application. Les impôts sur le revenu dépendent du montant de ce revenu. Supposons qu'il n'y ait que quatre échelons :

BLOCS DO... WHILE / DO... UNTIL

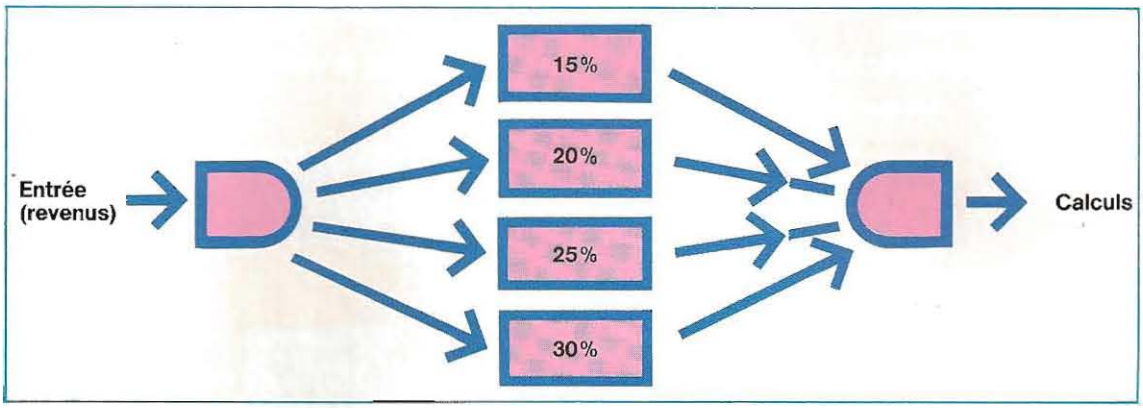




Revenu :

inférieur à 25 000 F	impôt = 15 %
de 25 000 à 39 999 F	impôt = 20 %
de 40 000 à 59 999 F	impôt = 25 %
égal ou supérieur à 60 000 F	impôt = 30 %

Le problème consiste à écrire un programme capable de déterminer le taux à appliquer et de calculer le montant de l'impôt et du revenu net.
Le schéma ci-dessus représente l'organigramme du problème avec les symboles normaux.
La décision 1 expose que le revenu n'est pas supérieur à 24 999 (inférieur à 25 000) ; dans ce cas le pourcentage est de 15 %. Ayant défini le pourcentage, le programme saute directement au calcul (étape 5).
Dans le cas où la réponse du bloc 1 est négative, on passe à la décision 2. Si le revenu est situé dans la deuxième fourchette (de 25 000 à 39 999 F), le programme sélectionne le taux de 20 % et passe au calcul.
On notera l'utilisation de l'opérateur ET pour exprimer deux conditions qui doivent se vérifier simultanément (revenu supérieur à 24 999 et inférieur à 40 000 F, c'est-à-dire de 25 000 à 39 999 F. L'étape 4 ne nécessite aucune vérification, car si l'étape 3 indique que le revenu n'est pas inférieur à 60 000 F, il faut automatiquement appliquer le pourcentage maximum.
La forme CASE permet d'exprimer cette logique d'une manière plus synthétique puisque le CASE peut être considéré comme les trois symboles de décision réunis. On en trouvera le schéma de principe à la page 214.
La programmation en Basic ou en Fortran ne permet pas de disposer d'instructions autorisant l'utilisation de logiques aussi compactes.



Solutions du test 5

1 / a : le symbole représente une décision. On indique le développement d'un calcul par un symbole de traitement général (rectangle) à l'intérieur duquel on inscrit le calcul à effectuer. La fin du programme est un ovale portant la mention FIN ou END.

---

2 / a et b:  le symbole du traitement général est utilisé aussi pour les calculs.

c :  ce symbole désigne l'impression d'un document.

---

3 / Ce symbole (renvoi) sert à indiquer les points de l'organigramme auxquels se réfèrent des parties éloignées (du schéma) et qu'il ne serait pas facile de relier entre elles par une ligne continue.

---

4 / a : le drapeau (flag) est un indicateur dont l'état signale si un événement déterminé a eu lieu ou non.

---

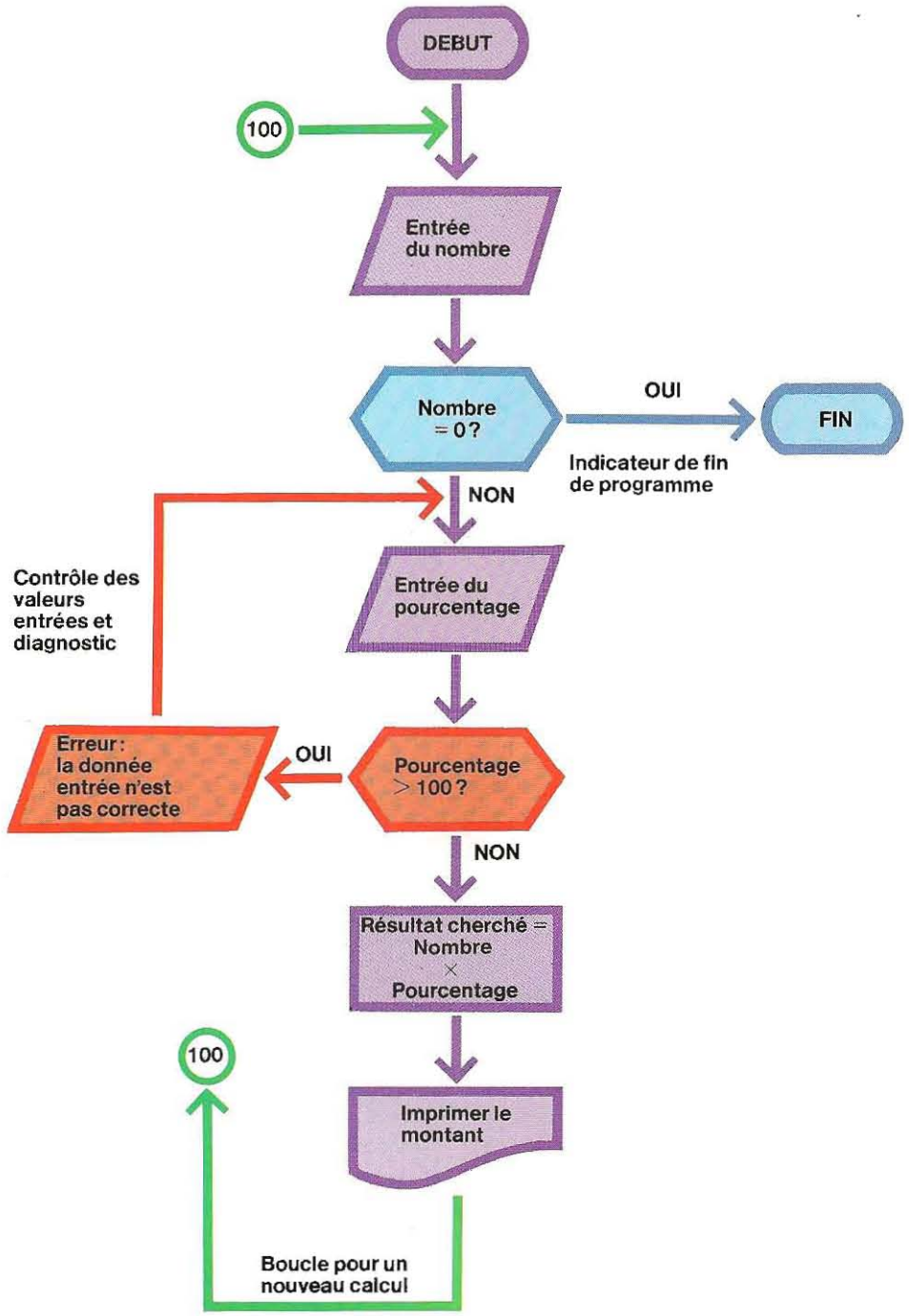
5 / L'analyse du problème peut être synthétisée comme suit :
- Entrée d'un nombre donné dont il faut calculer le pourcentage.
- Vérification que ce nombre ne soit pas égal à zéro (la valeur zéro peut être adoptée comme aiguillage, ce qui aboutirait à interrompre le programme).
- Entrée de la valeur du pourcentage et de sa vérification.
- Calcul : résultat cherché = nombre × pourcentage.
- Impression.

Cet organigramme est illustré page 215.

---

6 / L'erreur est double. Elle concerne d'abord la présence de deux boucles (I et J) qui se recoupent. La boucle I (qui commence au bloc 100) doit être fermée après la boucle J (qui commence à 200), pour éviter toute intersection. Pour bien déceler l'erreur, il suffit de tracer la boucle passant par le bloc 1 100 (à gauche sur le schéma p. 207) du même côté que la boucle 1 000 (à droite) : on voit clairement que les deux lignes de retour se coupent, ce qui n'est pas autorisé. La correction se fait en

échangeant la position des blocs 500 et 600 de fermeture des boucles.
L'erreur réside ensuite dans la forme du symbole utilisé pour désigner l'impression du nom : on aurait dû tracer le symbole I, spécifique de l'impression (voir p. 168).



Des machines qui parlent

Les considérables progrès techniques appliqués à l'informatique permettent, depuis quelques années déjà, des essais d'échanges verbaux entre l'homme et la machine.

Le problème est double : il faut que la machine soit capable d'entendre la voix humaine et de la comprendre, mais aussi qu'elle puisse parler de façon cohérente et formuler des phrases constituant une réponse à la question posée.

Les dispositifs électroniques dotés de parole sont fondés sur deux principes fondamentaux.

Le premier, la « synthèse verbale concaténée », utilise un ordinateur pour emmagasiner une sorte de dictionnaire des syllabes d'une langue donnée, présentées dans toutes les combinaisons possibles (voyelles-consonnes et consonnes-voyelles). Pour former un mot, l'ordinateur extrait de sa mémoire les syllabes requises et les relie ensemble dans le bon ordre (par exemple pour dire « or-di-na-teur »).

La méthode du parler concaténé est très rigide et produit un discours de mauvaise qualité. Elle a toutefois l'avantage d'être rapide et relativement économique.

L'autre méthode, dite de la « synthèse par règles », n'emmagasine pas dans la mémoire des sons ou des combinaisons de sons, mais plutôt un guide électronique des règles tirées d'une analyse détaillée du langage humain, et de la façon de générer ces sons. Le parler obtenu est de meilleure qualité ; toutefois, comme il faut mettre au point de très nombreuses règles, il faut aussi beaucoup de temps à la machine pour synthétiser le langage.

Un certain nombre de machines parlantes appliquant la « synthèse verbale concaténée » sont déjà utilisées, surtout aux Etats-Unis. Au lieu de les afficher, les calculatrices parlantes peuvent prononcer les chiffres correspondant aux touches du clavier que l'on enfonce, et donc lire le résultat à haute voix. Il existe également de petits dispositifs parlants, un peu plus grands qu'une calculatrice, et qui servent à apprendre à parler aux enfants. Une autre petite machine qui sait jouer aux échecs illustre verbalement chacun de ses mouvements.

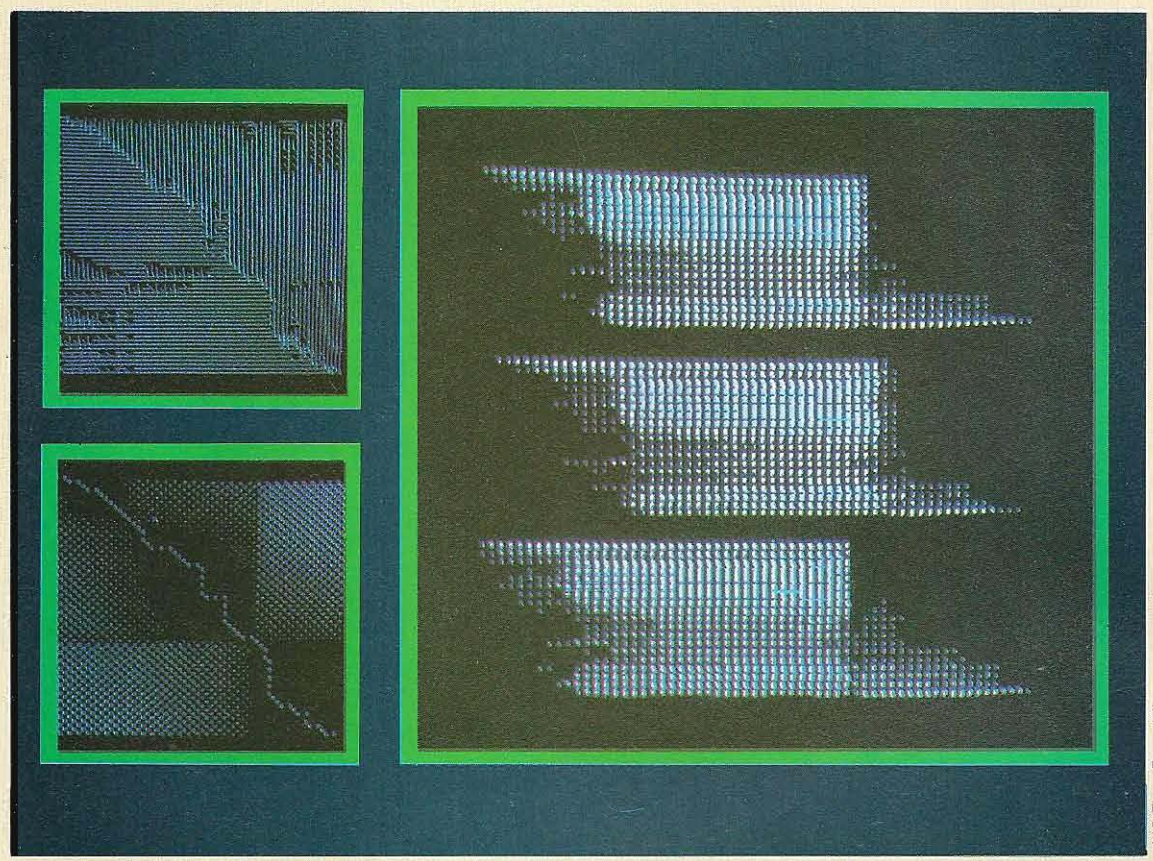
Une machine dotée de la parole est l'élément essentiel d'un système utilisé aux Etats-Unis par la Bell Telephone Company pour améliorer l'efficacité et réduire le coût de certains services normalement assurés par des opérateurs humains. Un petit clavier comparable à celui d'une machine à écrire est mis en place dans l'appareil téléphonique. Pour trouver le numéro d'un abonné, on se met en liaison avec le service des renseignements téléphoniques et l'on tape sur le clavier le nom et l'adresse de l'utilisateur recherché. L'ordinateur, à l'autre bout de la ligne, cherche dans sa mémoire, trouve le numéro requis et, avec la voix de la machine, répond par téléphone.

Le stade ultérieur prévoyant l'élimination du clavier et la dictée des questions à la machine, représente une difficulté supérieure.

Le spectrographe acoustique, inventé au début de la Seconde Guerre mondiale, est l'instrument fondamental des chercheurs qui tentent de mettre au point le système permettant à la machine de reconnaître le langage articulé. Il s'agit d'un dispositif électronique qui analyse les ondes acoustiques et en fournit le tableau graphique permanent. L'intérêt de ces tableaux est qu'ils peuvent être stockés (sous forme numérique) dans la mémoire d'un ordinateur.

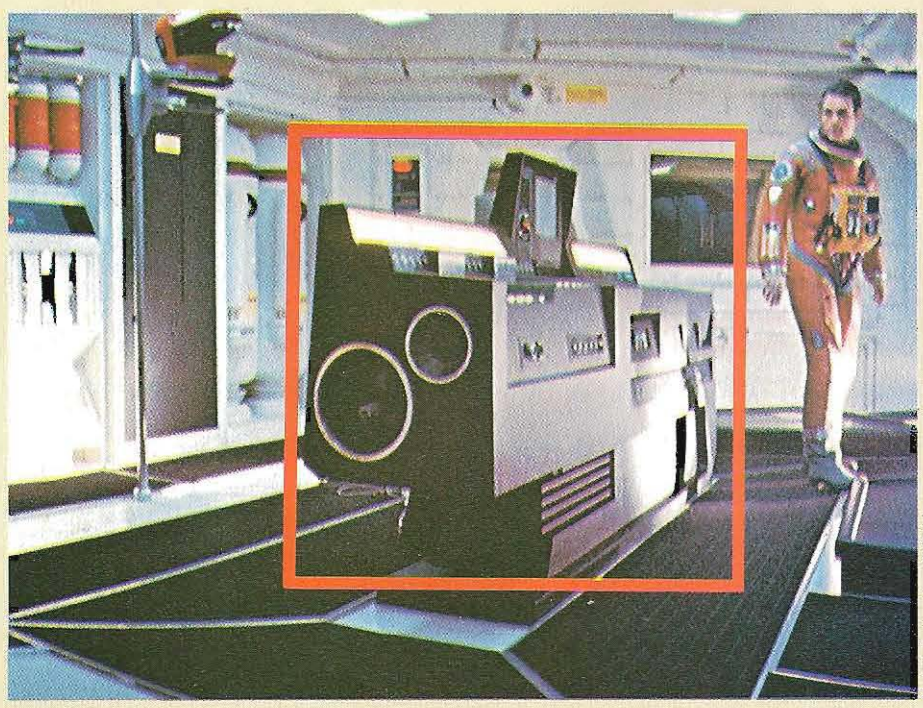
En théorie, il devrait être facile à ce dernier de reconnaître une parole donnée en comparant le spectrogramme de chacun des sons reçus avec les informations spectrographiques stockées dans sa mémoire. Le problème est qu'il n'existe pas deux personnes (même dotées d'un accent régional similaire) qui prononcent un même mot de manière exactement identique. Des paroles identiques prononcées par des sujets différents produisent des tableaux spectrographiques distincts, ce qui crée bien évidemment la difficulté, mais peut être fort utile lorsque l'ordinateur doit savoir identifier une voix (ouverture électronique des portières d'une voiture à la demande de son propriétaire).

Le moyen le plus efficace que l'on ait trouvé jusqu'ici pour surmonter cette difficulté consiste à « former » la machine, tout à fait comme on apprend à parler aux enfants. La formation commence avec un groupe d'environ cent personnes, ayant le même accent régional. Elles prononcent chacune à leur tour l'ensemble des mots que l'on veut faire reconnaître par la machine. Les différents spectrogrammes enregistrés sont ensuite introduits dans l'ordinateur qui analyse les tableaux graphiques d'un même mot et dresse un tableau intermédiaire, ou tableau type, afin de le mettre en mémoire pour les comparaisons futures. L'autre tâche de l'ordinateur consiste à évaluer la déviation par rapport au spectogramme type.



MŒ/Theo Bergström

A droite, de haut en bas : deux spectres du mot « sum » (somme), prononcé par une même personne, et la synthèse réalisée par l'ordinateur. A gauche : l'analyse des différences.



Kobal Collection MGM Inc.

Hal, l'ordinateur qui « joue » dans le film « 2001, l'Odyssée de l'espace », est l'exemple classique de la machine parlante de science-fiction.

Par la suite, quand d'autres personnes s'adresseront à la machine, cette dernière sera en mesure de comparer leur voix à celles dont les spectres ont déjà été enregistrés.
Les spectres stockés en mémoire jouent ainsi le rôle de portraits-robots, et ils peuvent se déformer, s'étirer, pour s'adapter aux différentes vitesses d'articulation des mots.
Lorsque l'un de ces spectres enregistrés dans la machine coïncide avec celui que l'ordinateur entend, l'ordinateur reconnaît le mot prononcé.

Ce type de procédé qui, à ce qu'on affirme, permet à la machine de reconnaître les mots émis même dans différentes constructions de phrase, peut être efficace dans 80,90, voire même 98 % des cas, si les mots lui sont adressés par la personne qui a préparé la machine, c'est-à-dire la personne qui a entré les mots dans l'ordinateur. Dès lors, il est possible de construire des machines capables de comprendre des ordres simples, et d'agir en conséquence. On peut facilement imaginer, par exemple, un distributeur de boissons auquel il suffirait de dire «thé», «lait», «sucre», pour obtenir effectivement une tasse de thé au lait sucré. Ce qui peut apparaître comme un prodige, n'est en fait qu'un modeste début, si l'on considère ce que serait l'étape suivante : un ordinateur capable de comprendre, non plus des mots isolés, mais un discours complet fait de phrases.

* * *

En 1970, le Département de la Défense américaine demanda à un groupe de chercheurs d'établir un rapport sur ce que l'on pouvait raisonnablement espérer d'un système électronique capable de reconnaître le langage parlé humain. Selon les conclusions du rapport, un système de ce type serait limité à un vocabulaire de 1 000 mots (le vocabulaire d'une personne ayant poursuivi des études jusqu'en classe de seconde peut comprendre entre 5 000 et 10 000 termes) et ne pourrait utiliser qu'une syntaxe grammaticale très simplifiée. Il faudrait en outre que son utilisation soit très spécifique.
L'entrée des mots dans l'ordinateur devrait aussi être réalisée de manière continue et égale, sans couper les mots en syllabes, de façon à ce que la machine ne prenne pas chaque syllabe et chaque son pour un mot distinct. La voix de la personne chargée de préparer ainsi la machine devrait donc être claire, le microphone utilisé de bonne qualité, et l'enregistrement devrait avoir lieu dans une pièce parfaitement silencieuse.

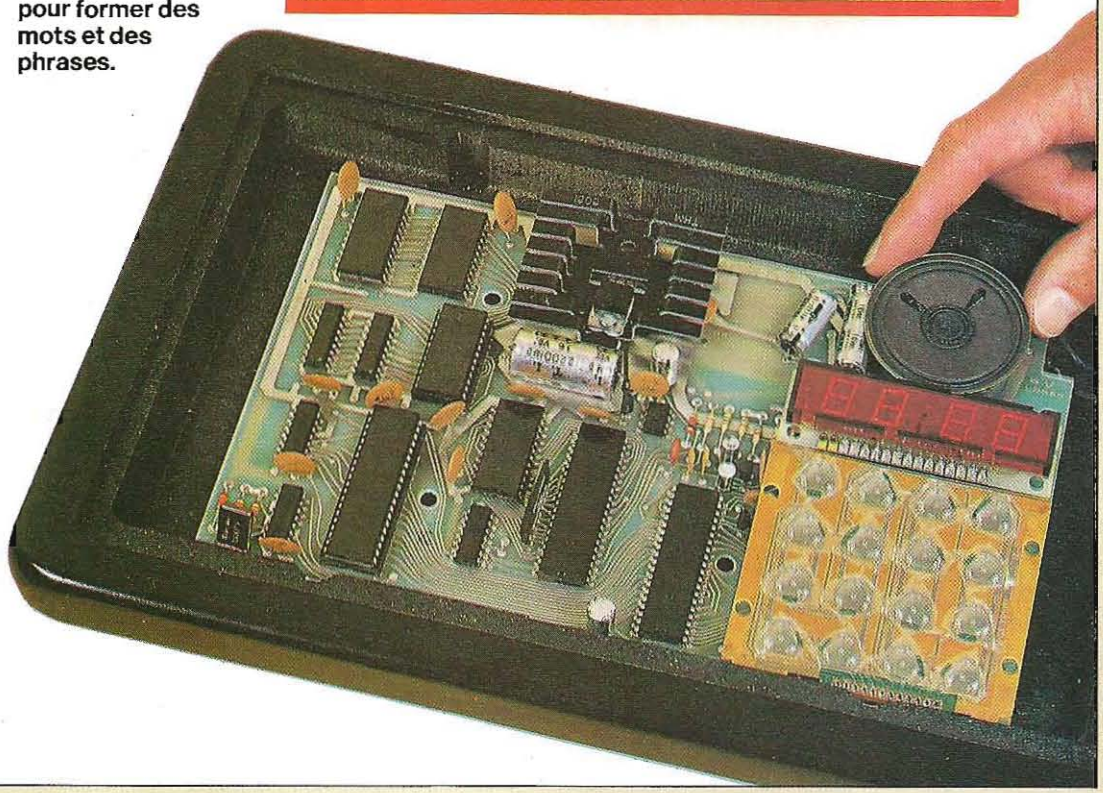
Vers la fin des années 1970, malgré certaines réalisations, les machines capables de reconnaître les mots étaient encore loin de remplir les conditions énoncées dans le rapport américain. Leur «vocabulaire» restait extrêmement pauvre. Deux des meilleurs systèmes mis au point étaient tout juste capables d'utiliser respectivement un répertoire de 39 mots (les 26 lettres de l'alphabet, les chiffres de 0 à 9 et trois ordres) et de 54 mots. On était loin des 1 000 termes annoncés comme une limite de la machine !
Pourtant c'est en 1980 qu'une équipe de chercheurs d'IBM annonça que le but était atteint. Passant deux heures devant un micro, l'opérateur devait prononcer neuf cents phrases constituant la base des connaissances de la machine en langage parlé. Le traitement par la machine d'une phrase prononcée en trente secondes par l'opérateur pouvait durer une centaine de minutes. Sans parler du risque d'erreurs...
En d'autres termes, l'entreprise n'était pas aisée et la commission américaine de 1970 n'avait sans doute pas prévu toutes les difficultés d'un tel projet.

* * *

Le langage humain est en effet extrêmement nuancé. Le discours le plus simple et le plus bref contient pourtant des centaines de petits fragments oraux, les phonèmes, que la machine doit d'abord identifier avec soin avant de les sélectionner. Elle doit aussi déterminer où finit un mot et où commence le suivant. Or, la plupart des gens parlent rapidement, sans nettement détacher les mots (ce qui est parfois justifié par les liaisons que l'on fait habituellement dans certaines langues).
En outre, et en admettant même qu'une machine soit assez perfectionnée pour isoler les mots prononcés et donc pour les reconnaître à coup sûr, il faudra aussi qu'elle soit capable d'accorder à deux mots identiques leur sens respectif.
Si les perfectionnements de la technique per-

Non seulement cette machine joue aux échecs à la perfection, mais elle parle. Elle «annonce» en effet les coups qu'elle est en train de jouer. La synthèse de la parole est obtenue par enchaînement de particules phonétiques enregistrées dans sa mémoire: les sons se trouvent assemblés électroniquement dans l'ordre voulu pour former des mots et des phrases.





Computer Games Ltd

mettent un jour de faire enregistrer à la machine assez d'éléments de base pour qu'elle puisse s'y retrouver dans toutes les subtilités d'une langue, ils devront également accélérer le traitement de ces données par l'ordinateur de telle manière qu'une vraie conversation puisse avoir lieu entre machine et opérateur, se déroulant selon le rythme habituel d'un dialogue entre deux personnes.

La mise au point de toutes ces améliorations techniques demandera sans doute un certain nombre d'années, et les recherches prendront peut-être un tour différent de celui qu'elles ont actuellement.

En tout cas, cette idée d'arriver à fabriquer des ordinateurs qui dialoguent avec l'opérateur est assez fascinante et l'avenir dira si elle relève de la pure science-fiction ou non.

Le stockage des données

Les informations traitées par l'ordinateur ne peuvent pas demeurer en permanence dans la mémoire centrale. Les données présentes dans cette mémoire n'y restent en effet que pour le temps de leur traitement par l'ordinateur et y sont remplacées par les autres informations qui doivent être traitées à leur tour. Il est donc nécessaire de préserver toutes les informations concernées par un traitement avant qu'une nouvelle utilisation de la machine ne vienne effacer le contenu de la mémoire. On le fera en les stockant sur les supports adéquats dont nous reparlerons. Un ordinateur peut saisir, traiter et fournir un très grand nombre d'informations, et cela à une vitesse extrêmement rapide. Aussi est-il nécessaire d'enregistrer ces informations à l'aide de supports de très grande capacité. Ces données devront être archivées et enregistrées à la sortie de l'ordinateur de façon à pouvoir être réutilisées en entrée lors de traitements ultérieurs.
Rappelons en outre que la moindre coupure de l'alimentation électrique de l'ordinateur entraîne la disparition des données contenues dans la mémoire centrale. Cela souligne encore la nécessité de stocker les données sur des supports totalement fiables, capables de conserver leur contenu pendant plusieurs années, et même sans alimentation électrique. Prenons un exemple : les salaires des employés d'une société sont calculés une fois par mois et les données de la paie doivent être disponibles chaque mois à une date déterminée (le 20 et le 21, par exemple). Il serait impensable d'encombrer la mémoire centrale de l'ordinateur, si grande soit-elle, avec ces millions de caractères pendant tout le reste du mois. De plus, il est bon qu'après le calcul du salaire de M. Leblanc, le résultat de ce calcul soit archivé ailleurs, libérant ainsi l'espace mémoire correspondant pour les données qui serviront à calculer la paie de M. Rousseau.
On voit bien là l'importance capitale du stockage des données sur des supports externes à l'ordinateur et que l'on appelle des mémoires auxiliaires ou mémoires de masse.

Les mémoires de masse

Laissons de côté le stockage des données sur cartes ou sur bandes perforées. Ce sont là des supports aujourd'hui dépassés ou, du moins, désormais réservés à des usages particuliers. Etudions plutôt les supports qui permettent l'enregistrement magnétique des données.. Ces supports ont en commun quelques caractéristiques générales. La lecture et l'écriture des données s'effectuent au moyen d'appareils spéciaux munis de dispositifs appelés têtes de lecture ou d'écriture. L'accès aux portions élémentaires du support magnétique se fait grâce à un mouvement mécanique de la mémoire de masse et de la tête située à proximité de la surface magnétisable (voir schéma page 223).
Les mémoires de masse se distinguent selon la manière dont elles permettent l'accès aux données qu'elles détiennent, c'est-à-dire selon la méthode qu'il faut utiliser pour repérer une information qu'on veut lire ou pour positionner une information qu'on veut écrire. On parlera ainsi d'accès **séquentiel** et d'accès **direct**.
L'accès est séquentiel lorsque, pour trouver une information, on doit nécessairement passer en revue tous les enregistrements existants jusqu'à la rencontre de la donnée recherchée.
Inversement, l'accès est direct lorsqu'il est possible de retrouver immédiatement l'information recherchée à l'aide de son adresse physique (numéro du disque, numéro de la piste et numéro du secteur).
Nous approfondirons ces notions dans la partie consacrée aux fichiers.

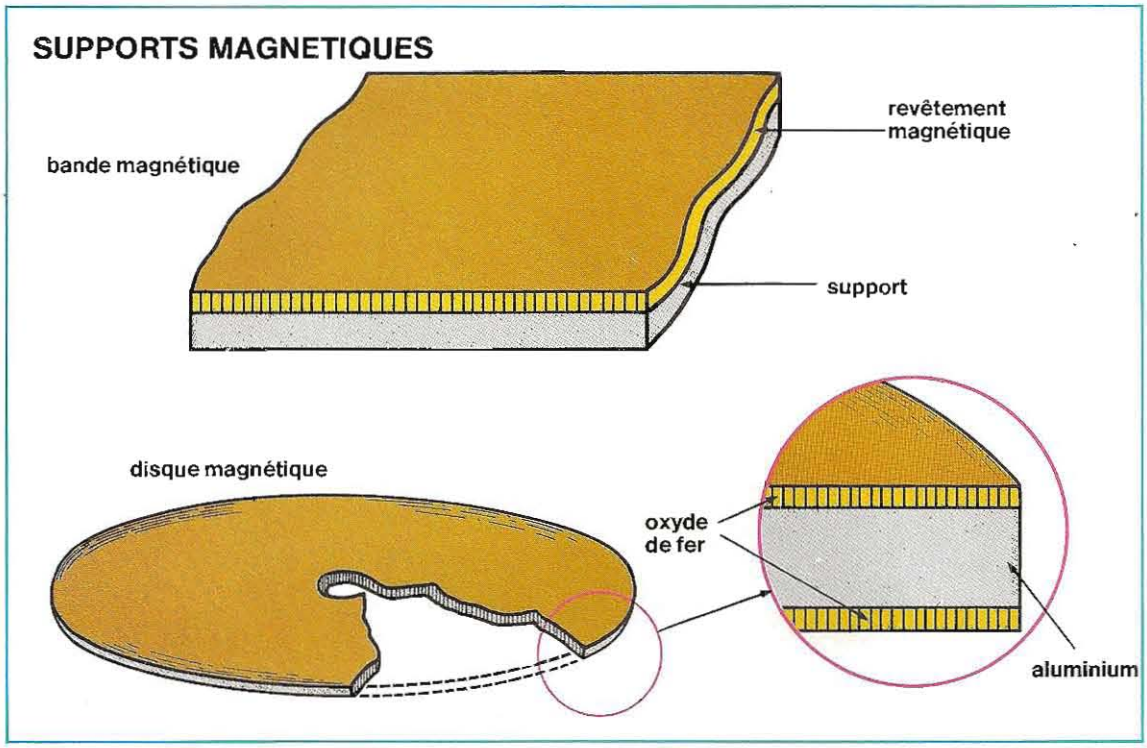
Les bandes magnétiques

On utilise beaucoup les bandes magnétiques pour l'enregistrement des très grandes masses de données, pour la copie des fichiers volumineux que l'on veut sauvegarder et pour faciliter le traitement de données organisées en fichiers séquentiels.
La bande est constituée d'un support plastique souple composé de nylon, de polyester, de mylar ou d'acétate de cellulose. Elle a, en général, une largeur d'un demi-pouce (12,7 mm) et sa longueur, variable, est couramment de 730 m. L'une de ses faces est recouverte d'un revêtement magnétique très mince. Elle ressemble donc beaucoup aux bandes que nous utilisons dans nos magnétophones.
Pour consulter ces bandes magnétiques



Italcable

Ces dérouleurs de bande utilisés dans un centre de calcul nécessitent une intervention manuelle.





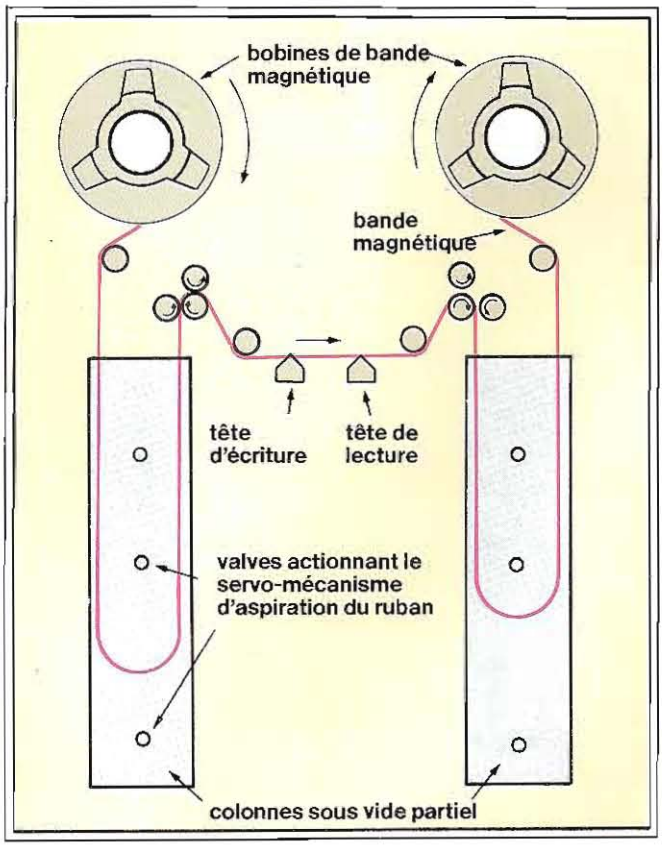
Sann

Exemple de mémoire de masse : dérouleurs à chargement automatique.

(lecture) ou y enregistrer des données (écriture) on a recours à des appareils spéciaux, les **unités de bande**, ou **dérouleurs**, considérés comme des unités périphériques de l'ordinateur. Le dispositif qui prend en charge la lecture et l'écriture ainsi que l'effacement de la bande est l'organe essentiel de l'unité de bande (voir schéma page 223). L'enregistrement se fait par la magnétisation de points qui peuvent prendre deux états associés aux valeurs 1 (point magnétisé) et 0 (point non magnétisé).

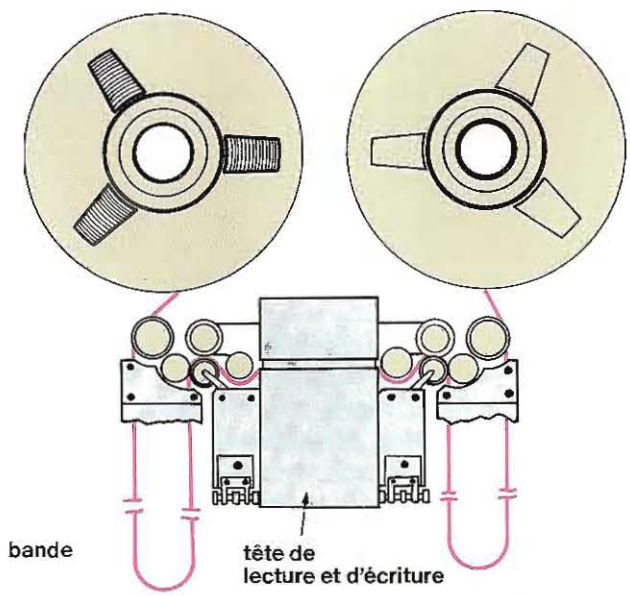
On peut considérer que la surface d'une bande magnétique est divisée en **pistes** (tracks en anglais) le long desquelles se succèdent les points magnétisables, de valeur 0 ou 1. Les bandes peuvent comporter sept ou neuf pistes, l'une d'elles étant utilisée comme piste de parité. Le bit (point) de parité prend la valeur 1 ou 0 selon que le nombre total de bits ayant la valeur 1 dans le caractère considéré est pair ou impair (parité transversale). Il existe le même type de contrôle* pour les bits d'un segment de piste, ou bloc, c'est la clé longitudinale (LRC, Longitudinal Redundancy Control).

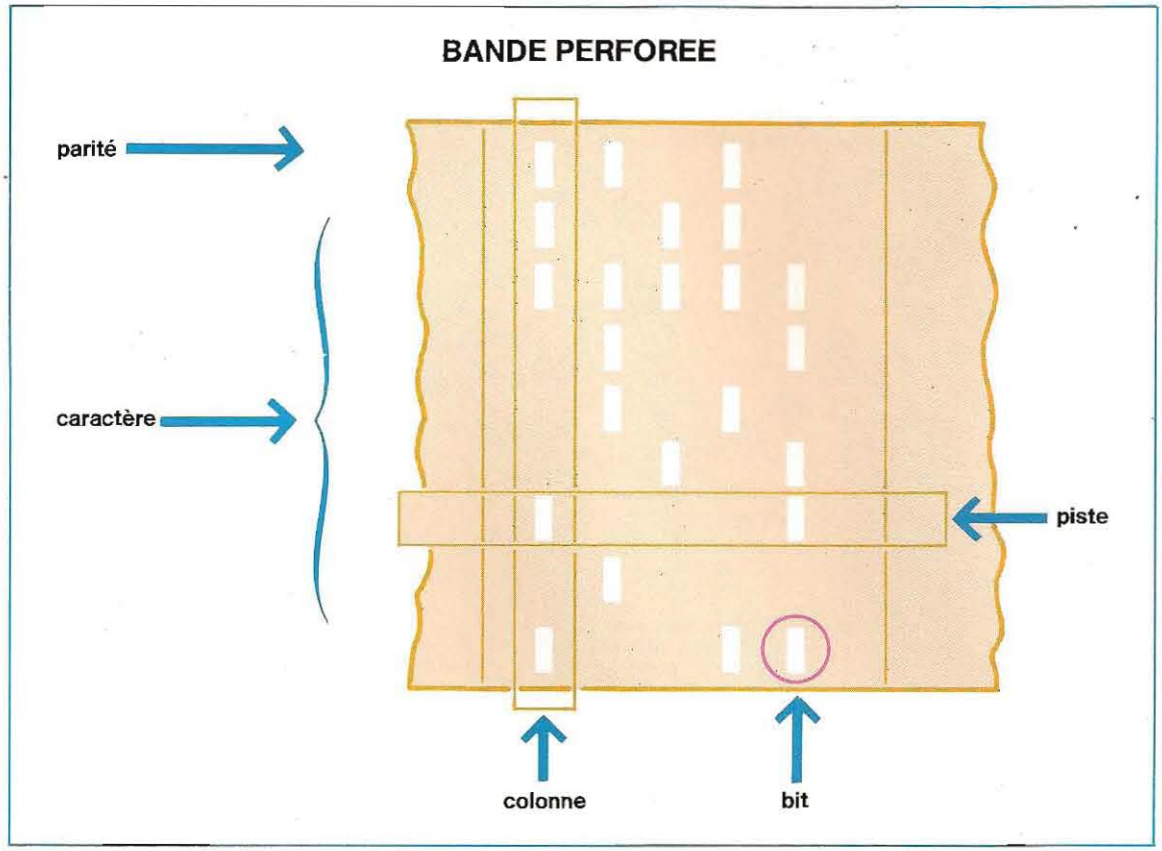
*Cette méthode est parfaitement comparable à la méthode de contrôle de transmission utilisée en code ASCII (bit de parité, voir de la page 117 à la page 119).



SCHEMA D'UNE UNITE DE BANDE

STRUCTURE D'UNE UNITE DE BANDE





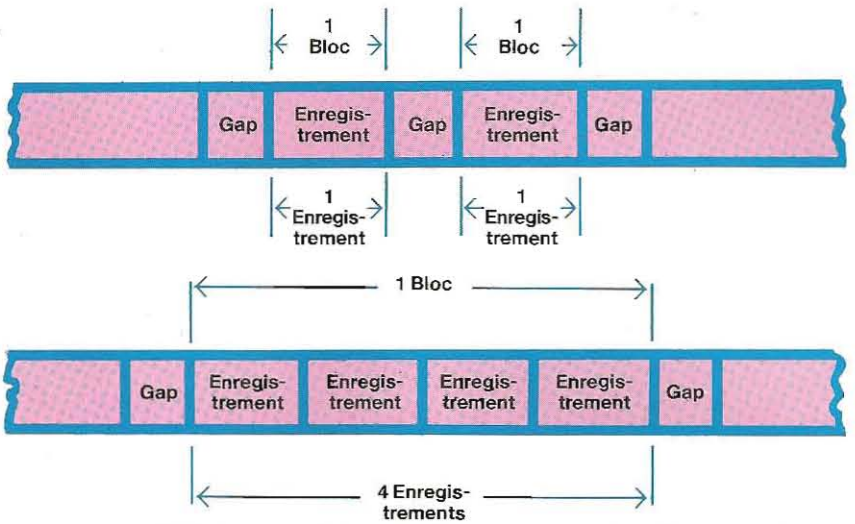
Le nombre de caractères que l'on peut enregistrer sur une bande magnétique dépend de la longueur de la bande (600, 1 200, 2 400 ou 3 600 pieds) et de la quantité de caractères que l'on peut enregistrer par unité de longueur (appelée **densité d'enregistrement**).

Les densités d'enregistrement les plus courantes sont 800, 1 600 et 6 250 caractères par pouce. Ces chiffres permettent d'évaluer la capacité théorique d'une bande. Ainsi, une bande de 2 400 pieds (730 mètres environ) enregistrée avec une densité de 800 caractères par pouce, a une capacité théorique de 24 000 000 de caractères.

En fait, l'enregistrement des caractères sur la bande n'est pas continu car les caractères sont regroupés en **blocs**, ou **enregistrements physiques**, séparés les uns des autres par une portion de bande non utilisée appelée **espace entre blocs**, ou **gap**. En général, la taille des blocs est fonction de l'organisation des données et de la capacité de la mémoire centrale de l'ordinateur. Quant à la longueur de l'espace entre blocs, elle dépend

du type d'unité de bande et est assez courte (par exemple, 0,6 ou 1,2 pouce).
La capacité réelle d'une bande magnétique dépend également de la longueur des blocs et de la fréquence des gaps. Ces espaces ont une influence importante sur la capacité et le

EXEMPLES D'ENREGISTREMENT DE DONNEES SUR BANDE MAGNETIQUE



débit réel des bandes magnétiques.
Comme nous l'avons vu plus haut, les unités de bande sont des périphériques reliés à l'ordinateur.
La transmission des caractères par câble (canal) à grande vitesse permet la commutation entre les deux appareils. La vitesse de transfert est très rapide : 40000, 160000, 320000, voire 1250000 octets par seconde.
Les bandes doivent, pour pouvoir être utilisées en toute sécurité, comporter une certaine longueur non enregistrable au début et à la fin (environ 3 et 4 mètres) qui leur permet, en début de bande, de garnir le mécanisme d'entraînement, et, en fin de bande, d'éviter un débobinage complet. Le début et la fin de la partie utilisable de la bande sont indiqués par des marques en aluminium de 25 mm environ, appelés **indicateur de début** et **indicateur de fin de bande**. Ce sont des plaquettes réfléchissantes, collées sur le côté non magnétique de la bande et qu'on appelle des **stickers.** Elles permettent aux cellules photoélectriques placées dans l'unité de bande de déterminer l'endroit où peut commencer **(point de chargement),**



Rhône-Poulenc Italie

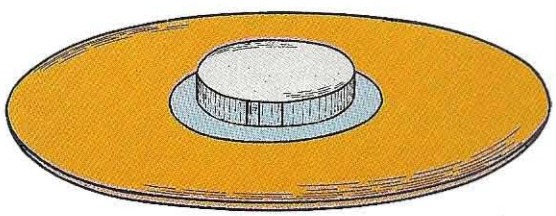
L'équilibrage sur deux plans confère une parfaite stabilité au disque pendant sa rotation dans le mécanisme d'entraînement (drive).

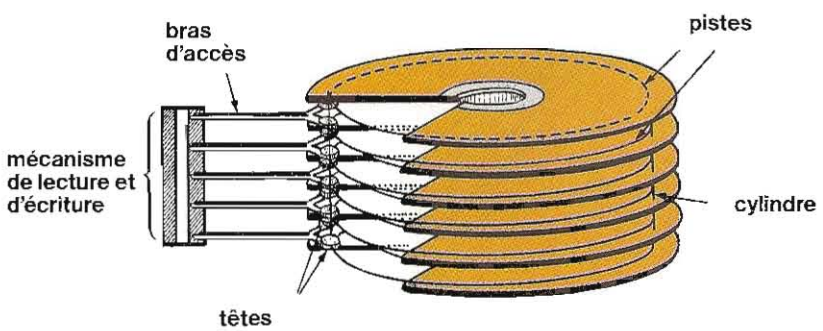


Rhône-Poulenc Italie

La certification numérique (contrôle de qualité à l'aide d'un oscilloscope) garantit la bonne adaptation du dispac à l'ordinateur.

DISPAC





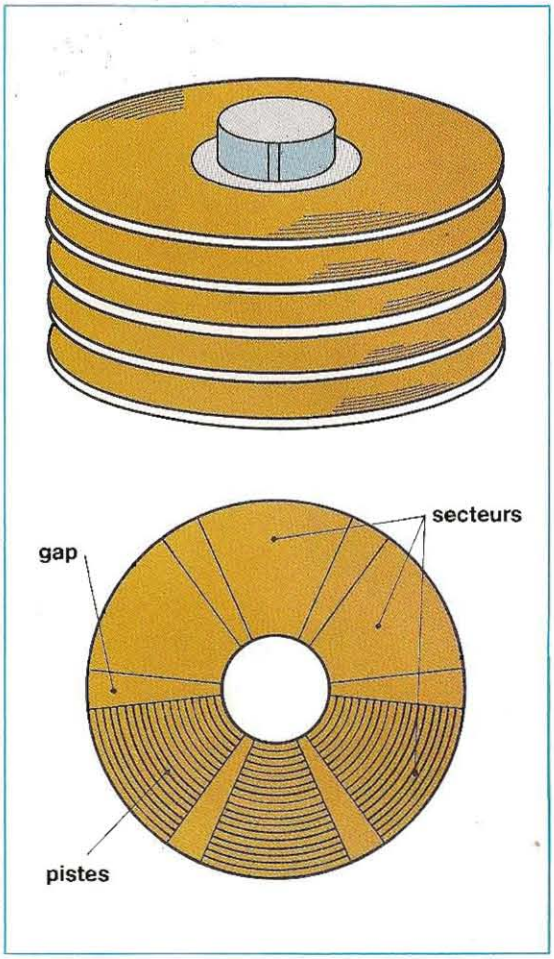
et l'endroit où doit prendre fin l'opération de lecture et d'écriture

Les disques magnétiques

Pour conserver les grandes quantités de données, on utilise aussi les disques magnétiques, qui sont des mémoires de masse à accès direct. Ils se différencient des bandes magnétiques par la rapidité avec laquelle ils permettent d'accéder à l'information recherchée. Un disque magnétique est constitué d'un mince disque, généralement en aluminium, dont les deux faces sont enduites d'un revêtement magnétique (schéma p. 226). Plusieurs disques peuvent être empilés sur un axe central et tourner à la même vitesse. Un ensemble de ce type est un dispac, ou disk-pack (voir schéma page 227). Si le montage est définitif, on dit que l'unité est à disques fixes ou à disques non amovibles. Si les mécanismes de lecture et d'écriture sont rétractables, on peut changer le dispac. On parle dans ce cas d'unité à disques amovibles.

Les disques sont séparés par des espaces dans lesquels se positionnent les têtes de lecture et d'écriture. Par un mouvement horizontal d'avant en arrière elles se placent en face des pistes.

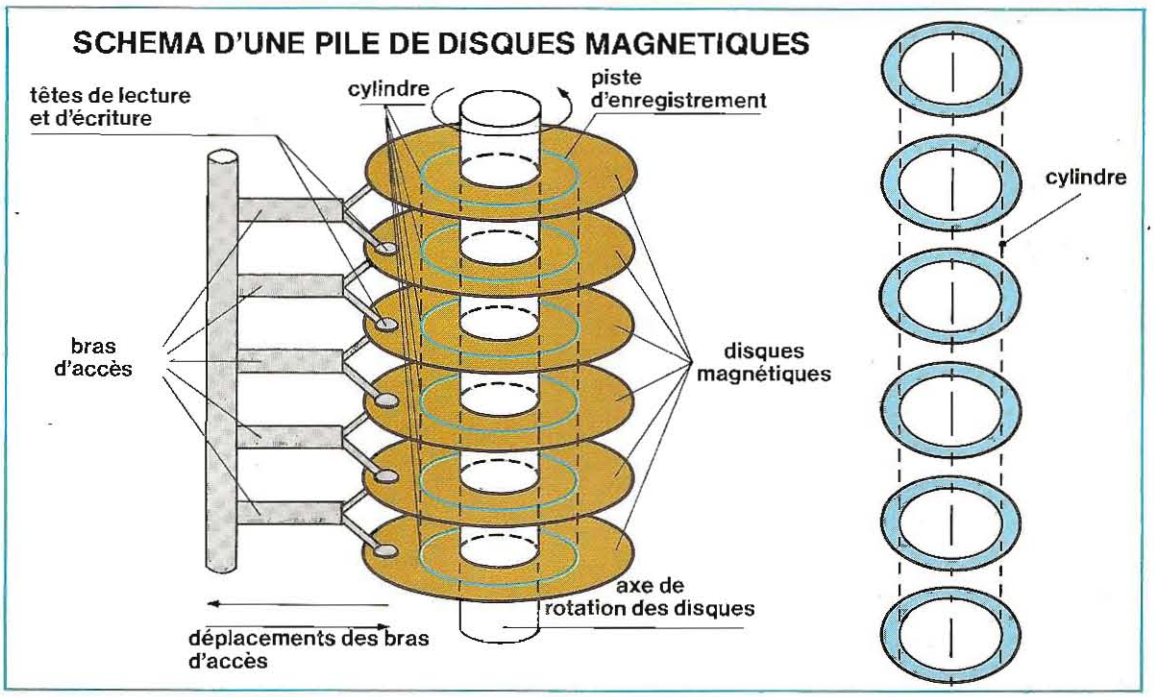
Les informations sont enregistrées sur les pistes concentriques suivant les mêmes principes physiques que sur les bandes. Chacune des faces d'un disque rassemble quelques centaines de pistes divisées en secteurs, eux-mêmes séparés par des intervalles (ou gaps) dans lesquels aucune donnée n'est enregistrée. Dans le cas



Le dispac constitue la mémoire de masse à accès direct des gros systèmes.

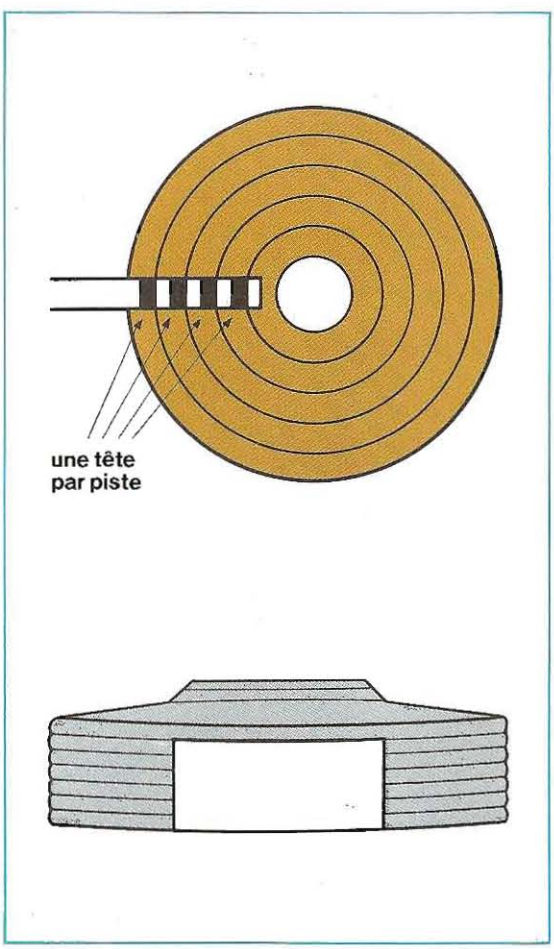


Intema



d'un dispac, c'est un seul bras mécanique qui porte la tête correspondant à la face inférieure d'un disque et la tête correspondant à la face supérieure du disque placé au-dessous. Le déplacement de tous les bras étant solidaire, les têtes se trouvent toutes placées sur une verticale, chacune étant positionnée devant la même piste de chaque disque. L'ensemble de ces pistes constitue un **cylindre** (voir schéma en haut de la page). Il est donc possible de lire ou d'écrire en même temps sur toutes les pistes qui appartiennent à un même cylindre, sans déplacer les bras, ce qui réduit considérablement le temps d'accès. On peut encore réduire ce temps en utilisant des têtes fixes, ce qui élimine le mouvement du bras d'accès (figure ci-contre). La rotation des disques fait que le contenu d'un secteur ne peut être lu ou écrit que lorsque ce secteur passe au-dessus (ou au-dessous) de la tête de lecture et d'écriture. Le temps d'accès aux informations varie donc selon l'ensemble de facteurs suivants :

- temps de positionnement de la tête sur la piste où se trouve le secteur recherché ;
- temps d'attente nécessaire à ce que le secteur se trouve au-dessus ou au-dessous de la tête ;
- vitesse de transfert des données (lecture ou écriture).



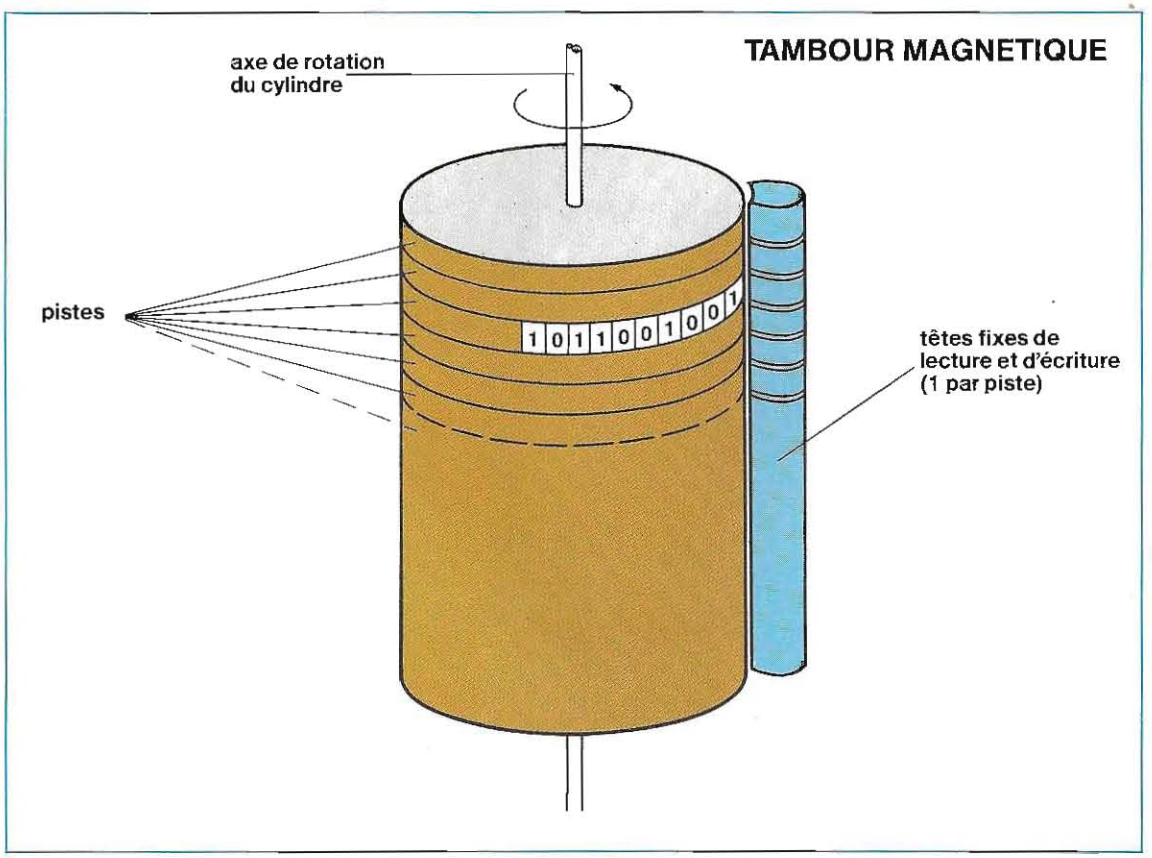
Ces temps sont d'ailleurs extrêmement brefs, puisqu'ils oscillent entre une dizaine et une centaine de millisecondes. La capacité d'un module varie entre quelques millions et 500 ou 900 000 000 de caractères.
L'échange des informations entre l'unité de disques et l'ordinateur s'effectue à des vitesses de transfert comprises entre quelques dizaines de milliers de caractères et 1 800 000, voire 2 000 000 caractères par seconde
Les disques fixes sont, en général, enfermés dans des conteneurs en plastique pourvus des ouvertures nécessaires aux bras d'accès (voir schéma en bas de la page 228).

Les tambours magnétiques

En raison de leur coût élevé, les tambours magnétiques servent plutôt sur les gros ordinateurs.
Il s'agit de mémoires à accès direct constituées d'un cylindre métallique, dont la surface latérale est recouverte d'une mince couche de substance ferromagnétique, et qui tourne à une vitesse élevée et constante.
Une tête de lecture et d'écriture est positionnée en face de chaque piste, à une distance de quelques centièmes de millimètre. On peut subdiviser les pistes en secteurs.
La plupart du temps les têtes sont fixes et il y en a autant que de pistes (voir schéma ci-dessous).
La capacité d'un tambour magnétique dépend de sa taille, du nombre de ses pistes et de la densité d'enregistrement des données sur les pistes.
Pour calculer le temps d'accès à une information, il faut additionner le temps de lecture proprement dit et le temps d'accès à la portion de piste concernée.

Les disquettes ou disques souples

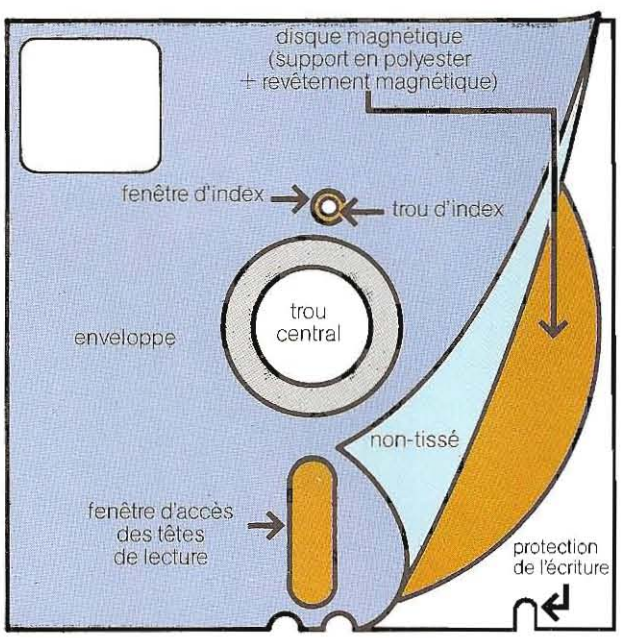
Les disquettes (floppy disk en anglais) reposent sur le même principe que les disques mais elles coûtent moins cher et elles sont plus faciles à manipuler.
Fabriquées dans une matière souple, les disquettes sont protégées par une enveloppe pourvue d'une petite fenêtre qui permet l'accès des têtes de lecture et d'écriture.





Centre THC

Une disquette (floppy) et un 45-tours ordinaire. Cette disquette 8"-DF-DD (8 pouces - double face - double densité) a une capacité d'environ 1 200 000 caractères.



STRUCTURE D'UNE DISQUETTE FLEXETTE FABRIQUEE PAR RHONE-POULENC

L'enveloppe
Celle-ci est fabriquée dans un PVC dont la résistance à la température est exceptionnelle.

Le non-tissé
La doublure en non-tissé de l'enveloppe est fixée de manière à assurer la rotation du disque à vitesse constante et son centrage parfait lors du chargement dans le lecteur.

Le disque
Il est composé d'une couche de polyester enduite d'une émulsion magnétique à haute densité.

Un petit trou sert à placer le disque dans son logement de manière à ce que les têtes soient alignées sur le point de départ des pistes.
Les disquettes sont également divisées en pistes subdivisées elles-mêmes en secteurs.
On distingue divers types de disquettes, selon qu'elles permettent :

1 / un enregistrement sur une seule face ou sur les deux ;
2 / un enregistrement à simple ou à double densité.

Selon les différentes combinaisons de ces éléments, la capacité d'un disque souple varie entre 80 000 et 1 200 000 octets (caractères). De plus, on peut loger plusieurs disquettes sur un même chargeur.
Ce type de mémoire est habituellement utilisé avec profit sur les micro-ordinateurs et les ordinateurs individuels. Leurs dimensions les plus courantes sont 5 pouces 1/4 et 8 pouces.

Les fichiers

L'ensemble des données enregistrées dans les mémoires de masse constituent un fichier (file en anglais). Un fichier se caractérise par l'homogénéité des informations qu'il contient, par exemple, les noms des employés d'une entreprise, des élèves d'une école ou les rubriques des articles d'un magasin.
Une donnée est un ensemble de un ou plusieurs caractères qui désigne, de façon précise et univoque, un élément quelconque. Ainsi, par exemple :

- **une personne :** Dupont Marc
- **un nombre :** 235
- **un sigle :** A2

Les données sont donc :

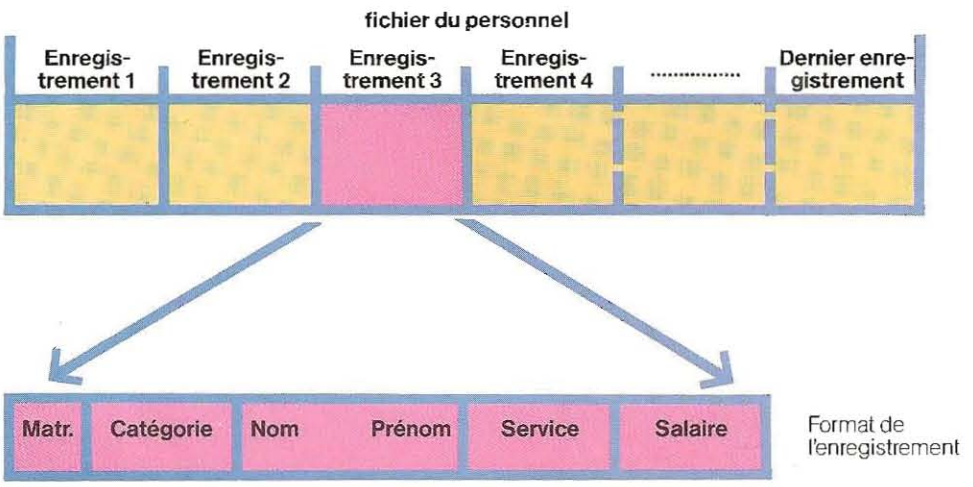
- **alphabétiques,** lorsqu'elles se composent de lettres et d'espaces ;
- **numériques,** si elles se composent uniquement de chiffres ;
- **alphanumériques,** lorsqu'elles se composent de lettres, de chiffres et d'espaces.

On appelle **zone** (field en anglais) l'emplacement de la mémoire qui est occupé par une donnée, et on lui donne, en général, un nom.

nom 40 caractères	
nom de famille 25 car.	**prénom 15 car.**

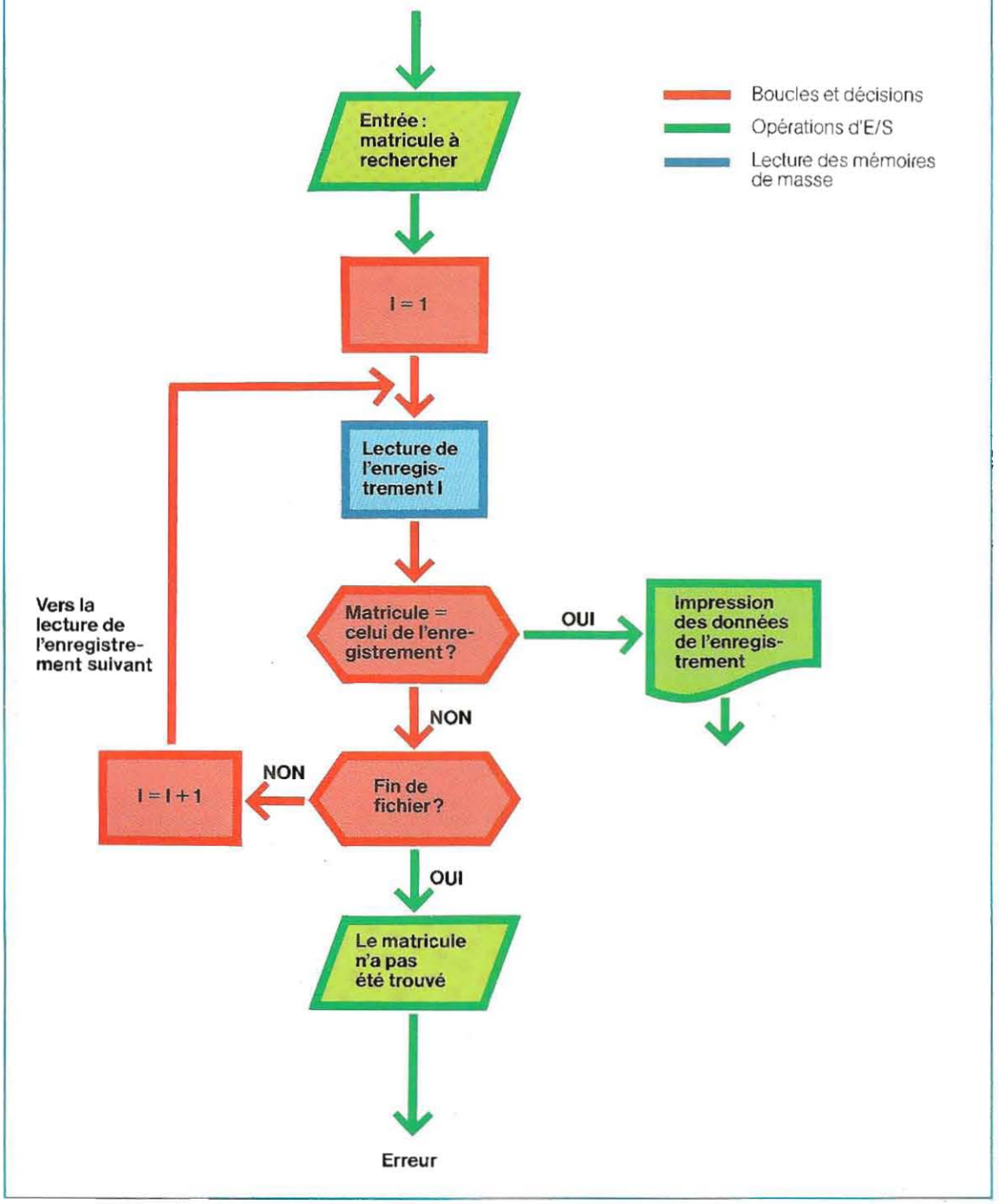
Dans cet exemple, la zone est divisée en deux éléments ou **sous-zones,** de longueur déterminée : vingt-cinq caractères pour la sous-zone nom de famille et quinze pour la sous-zone prénom. Si nous prenons le cas de M. Dupont Marc, nous constatons que le nom Dupont n'occupe que six des vingt-cinq caractères disponibles, et que le prénom Marc n'a besoin que de quatre des quinze caractères de la seconde sous-zone. Il reste donc de la place inutilisée dans les deux sous-zones. Leur taille a été déterminée pour qu'elles puissent accueillir les noms et les prénoms à traiter les plus longs. Lorsque, comme

EXEMPLE DE STRUCTURE DES DONNEES DANS UN FICHIER DU PERSONNEL



ici, la longueur d'une zone ne varie pas, on parle de zone de **longueur fixe.**
Lorsque, au contraire, l'espace mémoire utilisé est directement lié au nombre de caractères dont se compose la donnée, on dit qu'il s'agit d'une zone de **longueur variable.**
Dupont Marc, 11 caractères (y compris l'espace) ; Lhenry Marguerite, 17 caractères (y compris l'espace).
L'ensemble des zones qui qualifient le

ORGANIGRAMME D'UNE RECHERCHE A L'AIDE D'UNE CLE





Marka

Les enregistreurs magnétiques associés aux ordinateurs sont parmi les plus perfectionnés.

contenu de la donnée s'appelle un **enregistrement** (record en anglais). Imaginons que nous voulons construire un fichier contenant les principales informations administratives sur le personnel d'une société.
Les données à enregistrer sont les suivantes :

1 / numéro de matricule (3 caractères) ;
2 / catégorie (2 caractères) ;
3 / nom et prénom (40 caractères) ;
4 / service (2 caractères) ;
5 / salaire (7 caractères).

L'ensemble de ces informations (1,...5) constitue un enregistrement et l'ensemble des enregistrements est un fichier. Tout enregistrement du fichier comporte cinq zones, qui contiennent chacune l'un des cinq enregistrements définis ci-dessus. Le nombre total d'enregistrements est égal au nombre d'employés. La structure de ce fichier est illustrée par le schéma de la page 231.
En général, on utilise une ou plusieurs des zones comme **clé** afin de pouvoir reconnaître avec certitude les données d'un enregistrement. Dans notre exemple, les employés sont identifiés par leur numéro de matricule et celui-ci peut donc servir de clé d'accès aux données. Il faudra alors, pour retrouver les informations concernant un membre du personnel, entrer le numéro de son matricule. L'ordinateur commencera à lire le fichier à partir du premier enregistrement en comparant le matricule contenu dans celui-ci avec le matricule entré par l'opérateur ; et ainsi de suite, avec les autres enregistrements jusqu'à ce qu'il y ait égalité entre les deux numéros de matricule. L'ordinateur fera alors apparaître les données contenues dans l'enregistrement et terminera son travail. L'organigramme de ce programme simple est représenté page 232. On ne peut utiliser cette méthode de recherche (lecture séquentielle de tous les enregistrements) que lorsque les données ne sont pas très nombreuses. Pour les fichiers importants, il existe des méthodes plus rapides mais également plus complexes.
De même que les zones qui le composent, un enregistrement peut être soit de longueur fixe, s'il ne contient que des zones de longueur fixe ; soit de longueur variable, s'il contient au moins une zone de longueur variable ou un

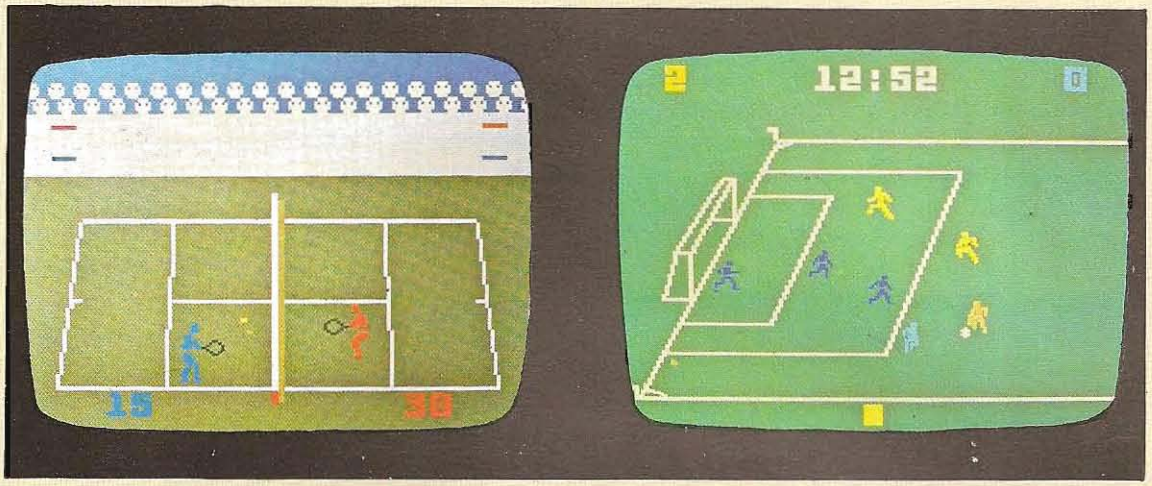
L'homme qui inventa le jeu vidéo

Bell, Edison, Fermi ; ces noms vous disent-ils quelque chose? Et celui de Higinbotham ? Non ? Pourtant, c'est cet homme qui, par une journée de 1958, en faisant apparaître une image sur son oscilloscope, inventa le jeu vidéo. Je dois avouer que, pour en arriver à cette conclusion, il m'a fallu consulter des montagnes de manuscrits et me faire une idée plus claire de l'histoire du jeu vidéo. Si vous entendiez parler d'un jeu vidéo antérieur à 1958, faites-le moi savoir ; sinon, acceptez que Willy Higinbotham occupe la place qui lui revient dans l'Histoire. Car c'est bien lui qui a inventé le jeu vidéo, n'en déplaise aux grandes sociétés qui s'en disputent la paternité.

* * *

Dans les années 1950, les visites guidées de laboratoires de recherche étaient plutôt ennuyeuses : on se bornait à vous montrer quelques photographies pour illustrer tel ou tel aspect des travaux.
C'est pourquoi Willy, qui avait déjà pris conscience de ses dons de physicien à la Cornell University et d'électronicien au MIT (Massachusetts Institute of Technology), décida de les rendre un peu plus vivantes. « Pourquoi ne pas faire participer les visiteurs à des petits jeux sur un écran, afin qu'ils puissent toucher des instruments et s'amuser à appuyer sur des boutons ?» se demandait-il. Sitôt dit, sitôt fait : Willy et ses collègues mirent au point le déroulement d'une partie de tennis destinée à l'écran de cinq pouces d'un oscilloscope.

A l'époque, les premiers ordinateurs numériques avaient déjà fait leur apparition et l'unité de recherche où travaillait Willy en avait justement construit un. Il utilisa, cependant, pour son petit jeu, un ordinateur analogique qui affichait les informations grâce à des tensions de valeur différente, et non à des impulsions ON/OFF (ouvert-fermé). L'ordinateur était connecté à un ensemble non programmable de relais électromécaniques, de potentiomètres, de résistances, de condensateurs et d'amplificateurs opérationnels. Willy reconnaît lui-même que ce rapide « bricolage » n'était pas des plus élégants mais, en tout cas, il fonctionnait très bien.
L'écran présentait un terrain de tennis vu de côté, un « T » renversé figurant le filet. Chacun des deux spectateurs désirant jouer tenait à la main un boîtier muni d'une manette et d'un bouton. La manette servait à commander l'angle de tir et le bouton à déclencher la frappe de la balle. En outre, le jeu tenait compte de certains éléments physiques inhérents au jeu réel : ainsi, quand la balle arrivait dans le filet, elle rebondissait moins fort et moins longtemps que lorsqu'elle frappait le sol.
En dépit de sa simplicité, ce jeu était passionnant. Les collègues d'Higinbotham se souviennent encore de l'enthousiasme des étudiants qu'il n'y avait plus moyen de faire partir. Je m'imagine sans aucune difficulté mon chef, Dave, les yeux rivés sur l'écran, totalement captivé par le déroulement du jeu. La balle et les limites du terrain étaient sans cesse redessinées sur l'écran, à un rythme tel qu'on ne percevait pas le scintillement de l'image. Le procédé est toujours utilisé aujourd'hui. En revanche, le système employé pour représenter le déplacement de la balle n'a pas été repris. Nous allons essayer d'en expliquer brièvement le principe.
On peut utiliser un oscilloscope pour visualiser des graphiques animés : un point lumineux dont les coordonnées sont proportionnelles à la valeur des tensions appliquées aux entrées x et y, se déplace sur l'écran. Partant de ce principe, Higinbotham utilisa des amplificateurs opérationnels pour réaliser un circuit qui simulait la trajectoire d'une balle et son impact sur le terrain. Au moment du choc, un relais était mis en action, et inversait la polarité d'un autre amplificateur. Ainsi, la balle repartait en arrière en donnant l'impression de rebondir. Elémentaire mais efficace, cette méthode permettait même de déterminer si la balle avait touché le filet car elle rebondissait alors à une vitesse différente (inférieure), ce qui conférait au jeu l'apparence de la réalité. La vitesse de la balle diminuait constamment à cause de la vitesse du vent et était simulée directement par une résistance de 10 megohms.
La hauteur du filet et la longueur du terrain étaient réglables, et une manette permettait aux joueurs de choisir le côté du service. Grâce à une pression sur le bouton du boîtier les joueurs



Marka

étaient assurés de frapper la balle, mais il fallait, toutefois, s'y prendre au bon moment et adopter le bon angle de frappe, sinon la balle n'allait pas dans le camp adverse.
Ces détails avaient été si bien étudiés que le jeu fut, deux années durant, le point fort des Laboratoires de Brookhaven et le « clou » de toutes les visites. Un jour, pourtant, il finit par être démonté.

J'ai demandé à Willy comment il était possible qu'il n'ait pas songé alors à faire breveter cette invention, lui qui était déjà détenteur de plus de vingt brevets dont le gouvernement était, depuis, devenu propriétaire. « Et bien, c'est peut-être mieux ainsi, sinon, aujourd'hui, on ne pourrait produire de jeu vidéo que sous licence gouvernementale ». Cette idée semble amuser Willy.
La société Magnavox, en revanche, ne trouve pas cela drôle du tout car les droits sur les jeux vidéo représentent des millions de dollars, et elle a déposé une demande de brevet pour des jeux vidéo basés sur le rebondissement d'une balle... Higinbotham n'a pas l'intention de tirer profit de cette affaire, mais il est tout de même décidé à défendre sa réputation et son prestige personnels.

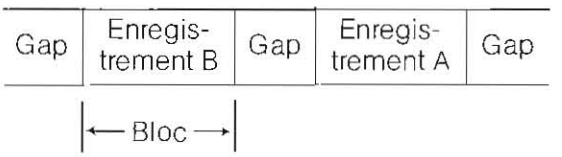
* * *

Voyons d'un peu plus près qui est cet homme. Higinbotham venait de décrocher son diplôme de physique à la Cornell University lorsque la Seconde Guerre mondiale éclata. Invité à participer aux recherches du MIT sur les radiations, il travailla à la mise au point d'une technique nouvelle, promise à un large succès, et alors connue sous le nom de Radio Detecting and Ranging. Cela ne vous dit peut-être rien, mais le sigle – RADAR – utilisé par la suite, vous est certainement plus familier ! Willy collabora ensuite au Manhattan District Project et consacra ses talents de physicien à une nouvelle technologie qui semblait fort prometteuse. En 1945, il était à la tête du Département Electronique et concevait les circuits temporisateurs qui allèrent, quelques années plus tard, égrener les dernières millisecondes précédant l'explosion de la première bombe atomique.
Quand la bombe explosa à Los Alamos, Willy se trouvait à une quarantaine de kilomètres de là. Aussi put-il observer les différentes phases de l'explosion. Il finit par me dire qu'il était remonté en voiture, ainsi que tous les autres observateurs, et que le trajet du retour s'était fait dans le plus grand silence. « Personne n'avait rien à dire ».
Durant les deux années qui suivirent, Willy occupa à Washington le poste de secrétaire de la fédération des Chercheurs américains et il milita activement en faveur de la non-prolifération des armes nucléaires, en qualité d'intermédiaire entre le Congrès et les chercheurs. Avec ses collègues, Willy qui est aujourd'hui doyen des Laboratoires de Brookhaven, a rassemblé la documentation la plus complète au monde sur les systèmes de protection nucléaire.
Et dire que c'est le même homme qui a inventé le jeu vidéo !

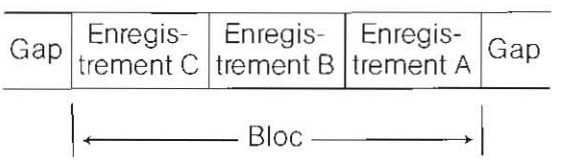
(D'après John Anderson, extrait de DATA MANAGER, avril 1983. © CREATIVE COMPUTING MAGAZINE.)

nombre variable de zones.
Les enregistrements que nous avons étudié sont des **enregistrements logiques*** ; ce sont les éléments d'un fichier, et ils contiennent une ou plusieurs informations relatives à un objet de ce fichier.
Les dispositifs de lecture et d'écriture utilisés sur les mémoires de masse permettent d'effectuer physiquement l'enregistrement d'une donnée sur des portions du support (bande, disque) limitées par deux entre-enregistrements. La partie de la mémoire concernée par cette opération est un **enregistrement physique.** Sa taille varie en fonction de celle de l'espace mémoire destiné à accueillir l'information dans l'unité centrale.
Lorsqu'on établit les enregistrements physiques, il faut à la fois réduire les temps d'accès aux informations et augmenter le plus possible le nombre des informations conservées dans la mémoire de masse, le tout, sans encombrer la mémoire centrale.
Pour en revenir aux enregistrements de notre fichier du personnel, ils peuvent se présenter de deux façons différentes sur un support magnétique (une bande, par exemple) :

Exemple 1



Exemple 2



Dans l'exemple 1, les entre-enregistrements (gaps) séparent tous les enregistrements, tandis que dans l'exemple 2, plusieurs enregistrements sont regroupés entre deux gaps pour former un bloc. Cette seconde méthode d'enregistrement permet de réduire la longueur des parties sans information et d'augmenter le nombre d'enregistrements logiques que l'on peut loger sur la bande.
En outre, le temps nécessaire aux opérations de lecture et d'écriture se trouve réduit dans la mesure où l'unité de bande livre trois enregistrements logiques en une seule fois. L'ensemble des enregistrements ainsi disposés s'appelle un bloc ou enregistrement physique, tandis que le nombre d'enregistrements contenus dans un bloc s'appelle le **facteur de blocage** (ici, trois).
On doit choisir le facteur de blocage de façon à réduire le plus possible le nombre des entre-enregistrements dans les limites consenties par l'espace mémoire disponible pour le bloc dans l'unité centrale.
Notons que les enregistrements de longueur fixe sont, en principe, regroupés suivant un même facteur de blocage et que les blocs sont donc également de longueur fixe. Lorsque les enregistrements logiques sont de longueur variable, on fixe une longueur maximale de bloc et on regroupe le plus d'enregistrements possibles par bloc, en évitant toutefois de morceler les enregistrements. Les blocs ont donc une longueur variable, inférieure ou égale à la longueur maximale prévue.
Lorsqu'il n'y a qu'un enregistrement par bloc, il y a coincidence entre enregistrements physique et logique. La longueur d'un enregistrement physique est, en règle générale, de 128 caractères sur les disquettes utilisées en micro-informatique. Dans le fichier du personnel pris comme exemple, la longueur de l'enregistrement logique est égale à la somme de la longueur des zones, soit $3 + 2 + 40 + 2 + 7 = 54$. On peut donc mettre deux enregistrements logiques par bloc ($128 : 54 = 2{,}37$).

Organisation des fichiers sur bande magnétique

Un fichier sur bande magnétique est une suite d'enregistrements logiques dont les informations peuvent se succéder sous les têtes de lecture et d'écriture. Un tel fichier se caractérise donc essentiellement par sa structure séquentielle, impliquant un mode particulier d'enregistrement. Les enregistrements peuvent être effectués dans le désordre, ou bien être rangés en ordre croissant ou en ordre décroissant. Des enregistrements écrits dans le désordre peuvent être classés à l'aide d'un programme de **tri** (en anglais, sort), généralement prévu dans le programme-système.

* Enregistrement logique a pour synonyme : article ; enregistrement physique a pour synonyme : bloc.

Les données du fichier du personnel peuvent être enregistrées selon les différentes façons suivantes :

1 / sans ordre de matricule

matr. 15 Vatin Jean
matr. 2 Arnoux Pierre
matr. 7 Durieux Robert
matr. 5 Blanchard Antoine
matr. 20 Violet François
.....

2 / en ordre croissant de matricule

matr. 2 Arnoux Pierre
.....
matr. 5 Blanchard Antoine
.....
matr. 7 Durieux Robert
.....
matr. 15 Vatin Jean
.....
matr. 20 Violet François
.....

3 / en ordre décroissant de matricule

matr. 20 Violet François
.....
matr. 15 Vatin Jean
.....
matr. 7 Durieux Robert
.....
matr. 5 Blanchard Antoine
.....
matr. 2 Arnoux Pierre

Dans cet exemple, l'ordre des matricules coïncide avec l'ordre alphabétique des noms des employés. Ce double classement n'a été adopté que pour permettre de retenir plus facilement l'exemple. En fait, il n'y a pas de classement alphabétique: seul importe l'ordre d'enregistrement sur le livre de paie.

Il va sans dire que la recherche est plus facile dans les fichiers triés en ordre croissant ou décroissant. L'organisation séquentielle des fichiers sur bande magnétique respecte, le plus souvent, l'ordre croissant.

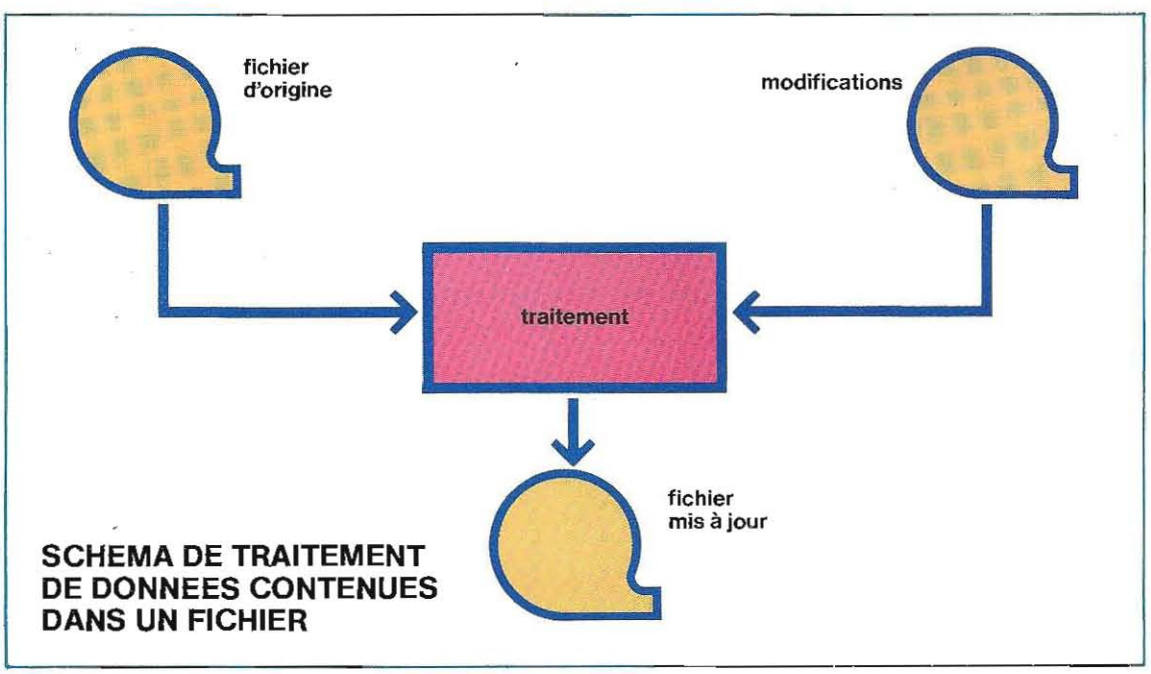
La recherche prend fin à la rencontre de l'enregistrement possédant le code demandé (enregistrement trouvé), à moins qu'un code supérieur au code recherché ne soit rencontré (enregistrement inexistant).

La mise à jour d'un fichier dépend de la méthode choisie pour l'organisation des données.

Supposons que nous devions mettre à jour le fichier du personnel trié selon l'ordre croissant de matricule, suite aux modifications survenues au cours du mois :

matr. 7 changement d'adresse
matr. 15 Vatin Jean démissionnaire
matr. 21 Violet Marie nouvelle employée

Le schéma ci-dessous illustre le traitement à effectuer.



SCHEMA DE TRAITEMENT DE DONNEES CONTENUES DANS UN FICHIER

Selon le principe de suppression physique, le fichier mis à jour contiendra les enregistrements suivants :

matr. 2 Arnoux Pierre
.....
matr. 5 Blanchard Antoine
.....
matr. 7 Durieux Robert (avec sa nouvelle adresse)
.....
matr. 20 Violet François
matr. 21 Violet Marie

Organisation des fichiers sur disque magnétique

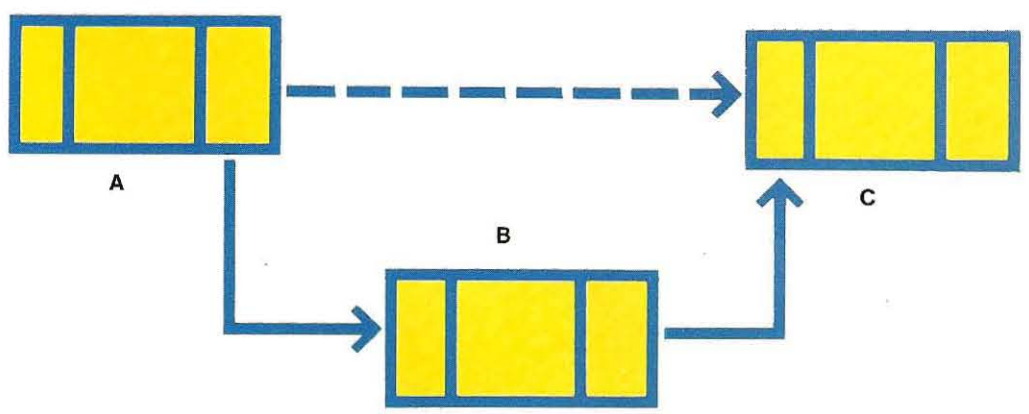
L'accès aux fichiers d'informations enregistrés sur bande est nécessairement séquentiel. En revanche, les caractéristiques physiques des disques magnétiques autorisent plusieurs modes d'accès aux informations.
On peut, en effet, structurer et relier les enregistrements afin de retrouver plus rapidement les données contenues dans les différentes zones.
Voici les quatre méthodes les plus fréquentes d'organisation d'un fichier sur disque :

- organisation séquentielle,
- organisation séquentielle avec chaînage,
- organisation séquentielle indexée,
- organisation directe.

Organisation séquentielle. Les enregistrements sont écrits l'un après l'autre suivant le critère de classement contenu dans la zone clé. Cette organisation est comparable à celle que l'on utilise sur bande magnétique : on doit, pour accéder à un enregistrement, lire tous ceux qui le précèdent.
On peut néanmoins effectuer un tri pour classer un fichier initialement écrit dans le désordre, ou, dans certaines limites, mettre à jour le fichier sans devoir le réécrire entièrement, ce qui est impossible avec une bande magnétique.
Ainsi, on peut supprimer un enregistrement en apposant une marque, ou drapeau, dans la zone appropriée. Il s'agit d'une suppression logique et non physique : l'enregistrement sera toujours présent mais il ne pourra plus être lu.
On peut également modifier un enregistrement à condition de réécrire intégralement le bloc auquel il appartient. Il est possible d'ajouter des compléments de données, grâce à des zones de dépassement situées en fin de fichier.
Cependant, en dépit de tous les avantages que nous venons de décrire et de la rapidité du support, la méthode séquentielle ne tire pas tout le parti possible des disques magnétiques.

Organisation séquentielle avec liens de chaînage. Chaque enregistrement contient, en plus des données proprement dites, des indications qui permettent de redescendre jusqu'à l'enregistrement suivant, et de former des chaînes logiques parfaitement ordonnées en fonction de la clé. Les indications de

INSERTION D'UNE DONNEE PAR LA MODIFICATION DES POINTEURS



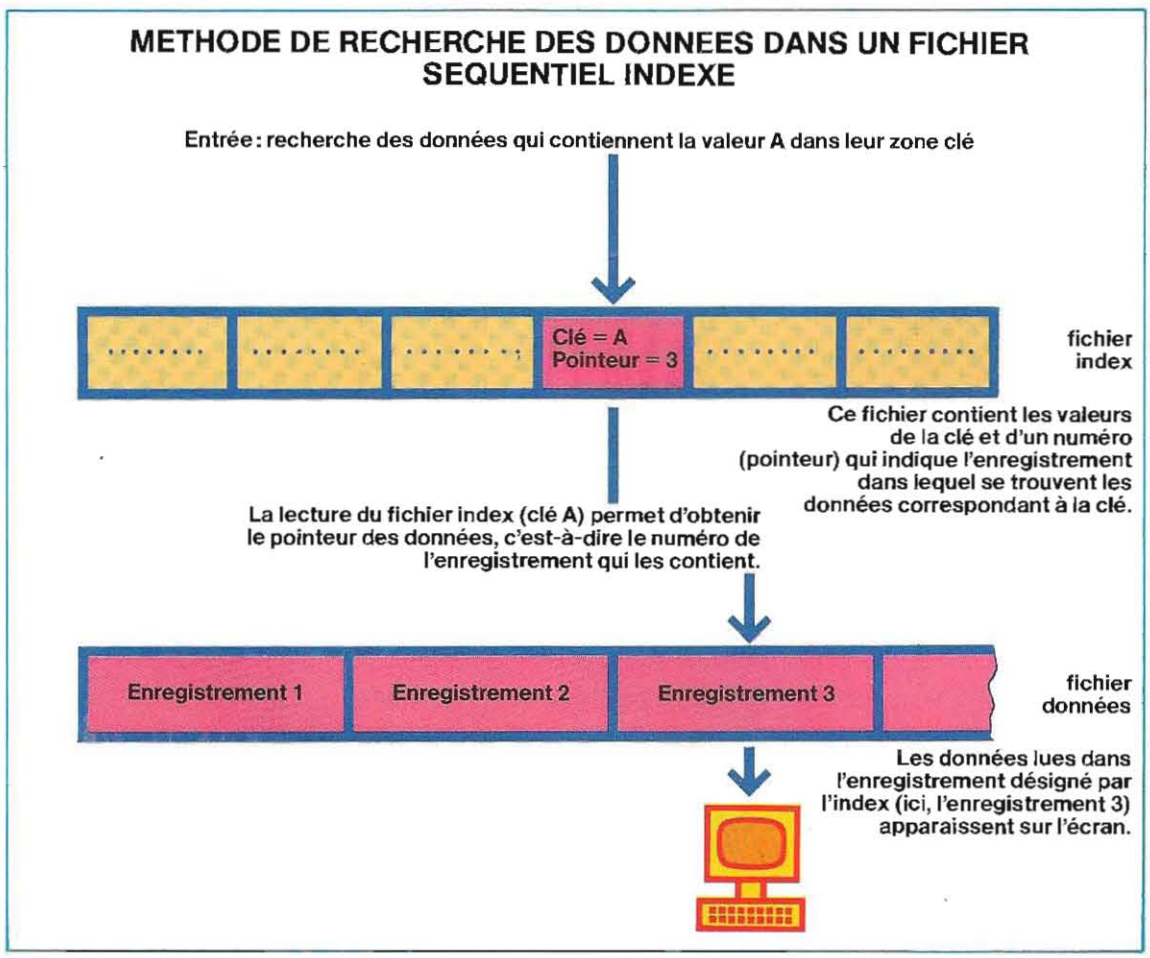
chaînage s'appellent des **pointeurs** (pointers en anglais). Les enregistrements sont alors composés de cette façon :

clé	données	pointeur

Cette méthode facilite insertions et suppressions : ces mises à jour se font par simple modification de la chaîne des pointeurs.

A / Exemple d'insertion
Pour ajouter un enregistrement B entre les enregistrements A et C, on modifie le pointeur de A pour qu'il désigne B, qu'on dote d'un pointeur vers C.

B / Exemple de suppression
Pour retirer l'enregistrement B de la chaîne A B C, il suffit de diriger le pointeur de A sur C.



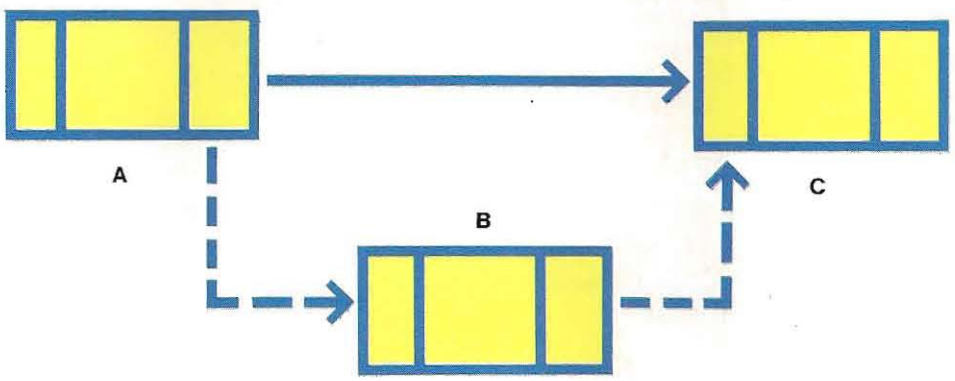
La succession physique des enregistrements reste inchangée et les nouveaux enregistrements sont insérés dans une **zone de dépassement de capacité,** périodiquement expurgée par la réécriture séquentielle du fichier.

Organisation séquentielle indexée. La zone clé est l'élément fondamental qui autorise l'introduction de ce type d'organisation dans un fichier : la valeur qu'elle prend dans chaque enregistrement permet d'identifier celui-ci sans équivoque.
La gestion du fichier se fait, là encore, de manière séquentielle. Il doit donc être classé en fonction de la clé d'accès. Mais il est cependant couplé à un autre fichier qui joue le rôle d'index.
Un fichier séquentiel indexé se compose donc de deux éléments :

- un fichier index,
- un fichier de données.

Le schéma de la page 239 illustre le principe de ce système.
C'est la table enregistrée sur le disque comme un fichier séquentiel qui constitue l'index du fichier.
Un même fichier peut être associé à plusieurs index, ce qui permet une nouvelle diminution du temps d'accès.

SUPPRESSION D'UNE DONNEE PAR LA MODIFICATION DES POINTEURS



Test 6

1 / Lesquelles de ces affirmations sur les mémoires de masse sont vraies ?
a) leur capacité est égale à celle de l'ordinateur ;
b) les bandes magnétiques sont les plus rapides ;
c) elles permettent de stocker d'énormes quantités de données.

2 / Combien un disque souple (disquette) peut-il contenir de caractères ? Donnez les ordres de grandeur habituelle minimale et maximale.

3 / Définissez : a) les fichiers ; b) les enregistrements ; c) les zones.

4 / A quoi servent habituellement les bandes magnétiques ?

5 / Remettez en ordre hiérarchique les mots suivants : octet, fichier, bit, enregistrement, ensemble de fichiers, zone.

Les solutions du test se trouvent pages 244 et 245.

Rappelons que le fichier qui contient les données doit impérativement être ordonné et que la valeur de la clé ne doit permettre d'identifier qu'un seul enregistrement. Nous aurons une meilleure idée de ce type d'organisation en l'illustrant par deux exemples classiques. On peut créer, à côté d'un fichier d'articles enregistré à partir de la piste 3 d'un disque, le fichier index suivant :

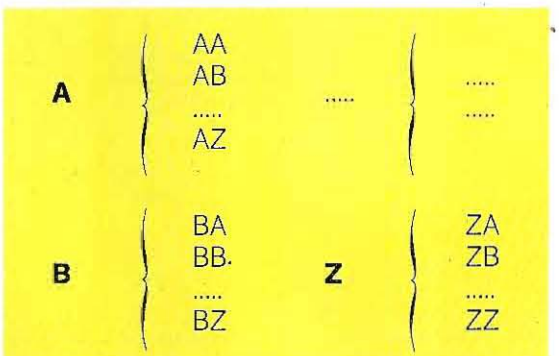
Fichier index

Pointeur de la donnée (numéro de la piste)	Clé d'accès à la donnée (dernière clé pour chaque piste)
3	1450
4	2972
5	4250
6	5587
7	6999

Pour accéder à une donnée de clé déterminée, on procède à une lecture séquentielle du fichier index pour retrouver la clé dans la table qu'il contient. On découvre ainsi le pointeur de la donnée, en l'occurence le numéro de la piste où se trouve l'enregistrement contenant la donnée cherchée. La lecture du fichier de données se limite aux enregistrements contenus dans la piste indiquée par le fichier index. Ainsi, la recherche de l'article dont la clé est 5350 ne nécessitera que la lecture des enregistrements du fichier de données présents sur la piste 6.

Considérons maintenant la gestion d'un fichier dont les éléments sont les mots d'un dictionnaire. Imaginons que ce fichier est divisé en 26 sous-fichiers, correspondant chacun à une des lettres de l'alphabet. Le premier niveau d'index alors créé est donc une table de 26 lettres. Si nous divisons ensuite chacun des groupes de mots ainsi formés en sous-fichiers, en fonction de la seconde lettre de chaque mot, nous créons 26 nouvelles tables, toujours composées de lettres de l'alphabet (second niveau d'index).

En somme, on fait correspondre à chaque élément du premier index l'adresse de la seconde table où sont répertoriées les



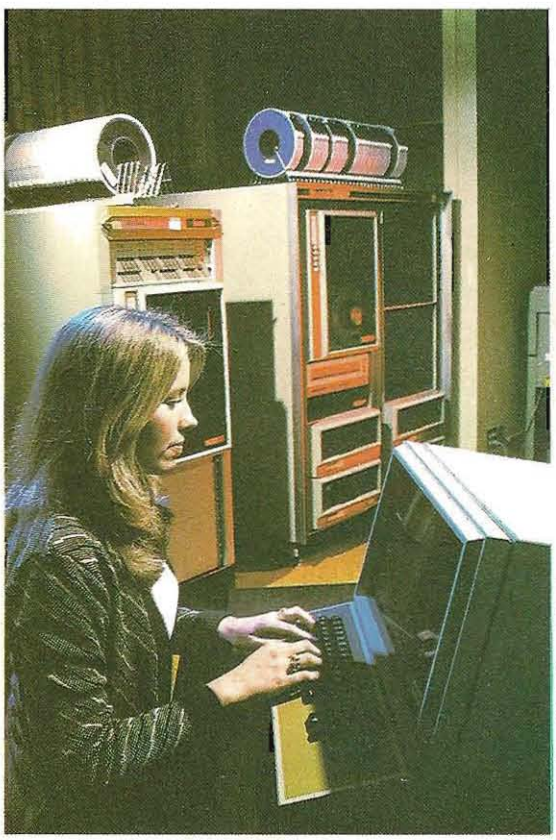
deuxièmes lettres des mots. Plus précisément, on obtient la relation ci-dessus :

L'adressage des sous-fichiers par l'index de second niveau permet de lire le fichier pour y trouver un mot déterminé. La recherche se compose des opérations suivantes :

a) accès à l'index de premier niveau et lecture séquentielle de ses éléments jusqu'à la rencontre de la première lettre du mot recherché : le pointeur renvoie à l'index de second niveau ;

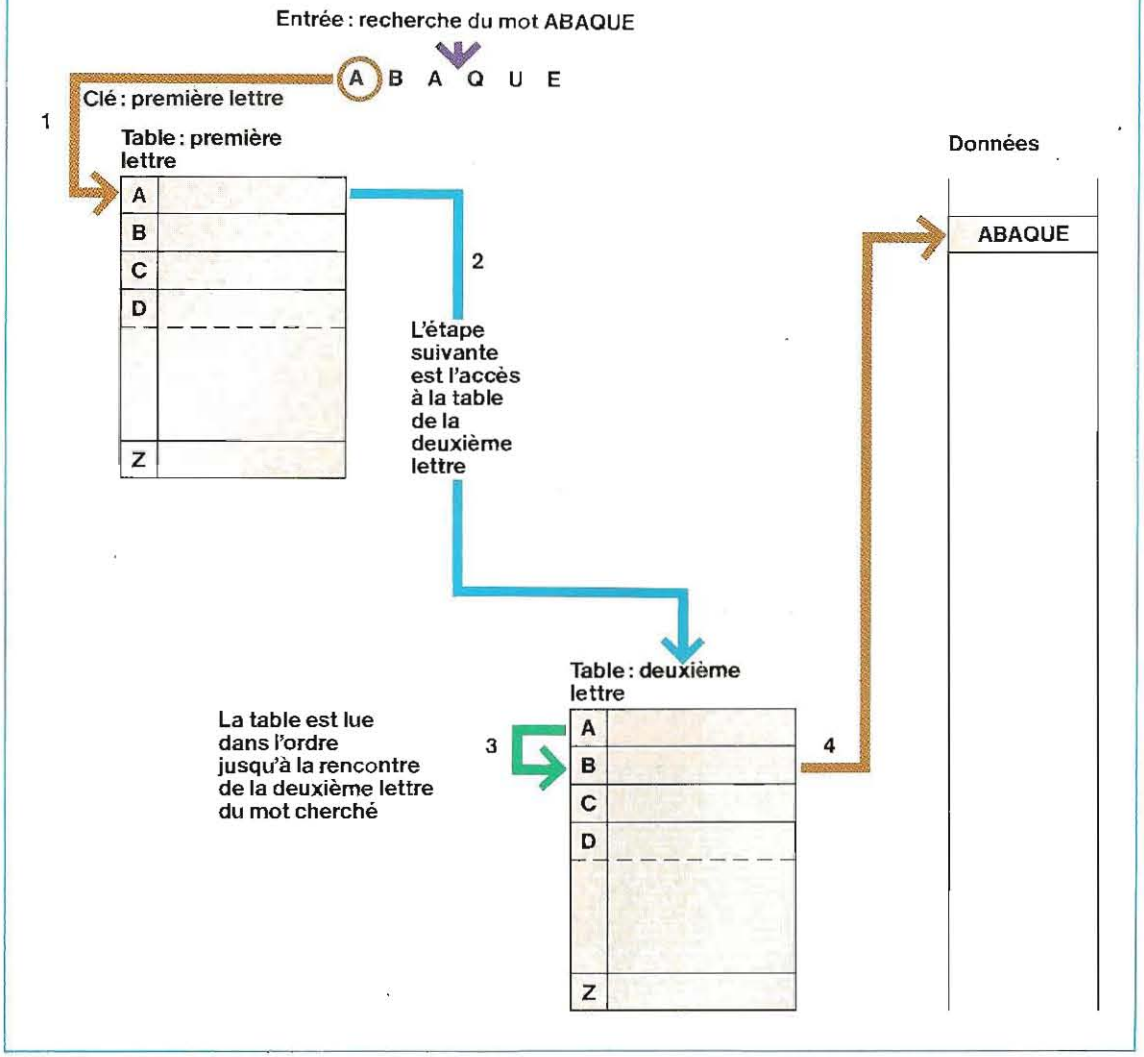
b) accès à l'index de second niveau, c'est-à-dire à la table correspondante aux deuxièmes lettres et lecture séquentielle jusqu'à la rencontre de la deuxième lettre du mot recherché : le pointeur renvoie, cette fois, au sous-fichier du fichier dictionnaire ;

Opératrice au travail.



C. O'Rear/West Light-Grazia Neri

SCHEMA LOGIQUE DE LA RECHERCHE DANS UN FICHIER DICTIONNAIRE A DEUX NIVEAUX DE CLES



c) accès au sous-fichier du dictionnaire et lecture séquentielle jusqu'à la rencontre du mot recherché.

Organisation directe. Une fois encore, le fichier apparaît comme une structure séquentielle organisée pour contenir les enregistrements à des emplacements prédéterminés. L'adresse physique d'un enregistrement à chercher ou à écrire sera déterminée à partir de sa clé logique, à l'aide d'un algorithme de calcul, appelé **formule de conversion** (ou de randomisation).
Le fichier doit pouvoir contenir tous les enregistrements dont l'adresse sera déterminée par ce calcul et doit donc offrir plus de place

que n'en occupent les enregistrements proprements dits.
Ce gaspillage de place sur le disque est le principal inconvénient de cette organisation.

En résumé :

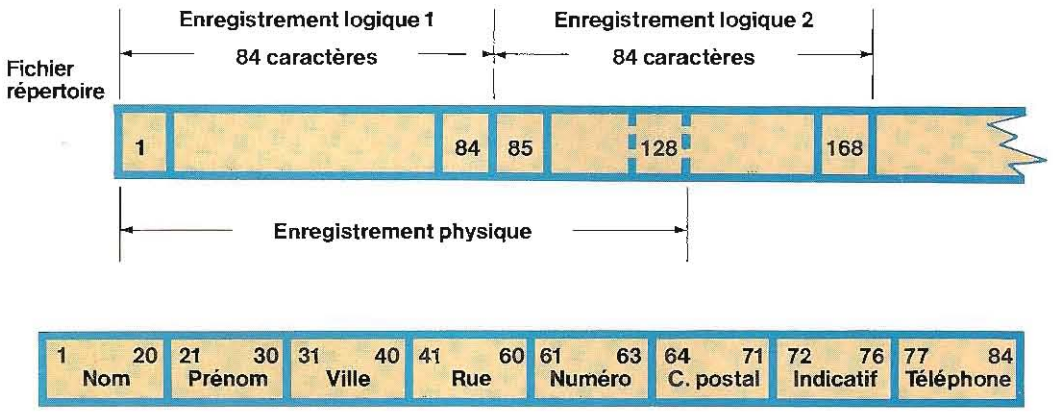
- **Les supports magnétiques de stockage des données sont principalement : les bandes, les tambours, les disques durs (fixes ou amovibles) et les disques souples ou disquettes.**
- **Un ensemble cohérent de données ayant un objet commun s'appelle un fichier.**
- **Les fichiers sont divisés en enregistrements physiques, pour les opérations de lecture et d'écriture, et en enregistrements logiques pour la programmation.**
- **Un enregistrement logique peut remplir plusieurs enregistrements physiques ou, inversement, un enregistrement physique peut contenir plusieurs enregistrements logiques, selon la longueur des zones de données.**
- **Les enregistrements logiques sont divisés en zones contenant chacune une information.**

Nous allons appliquer ces notions à l'exemple d'un répertoire téléphonique.
Les données à enregistrer sont les suivantes :

1 / Nom	20 caractères
2 / Prénom	10 caractères
3 / Ville	10 caractères
4 / Rue	20 caractères
5 / Numéro	3 caractères
6 / Code postal	8 caractères
7 / Indicatif régional	5 caractères
8 / Numéro de téléphone	8 caractères

Chaque donnée occupe une zone à laquelle a été attribuée une longueur maximale définie par un nombre déterminé de caractères. On n'aura à stocker que les données énoncées, et il suffira pour cela d'un seul fichier. Il existe des bases de données plus complexes composées de plusieurs fichiers dont chacun contient des données de structure différente. L'enregistrement logique regroupe la somme des caractères réservés pour les 8 données (zones) et a donc, ici, une longueur de 84 caractères.
Le schéma ci-dessous illustre la structure d'un tel fichier. La longueur de l'enregistrement physique dépend de la machine utilisée car elle est imposée par le matériel.
Sur la figure, on a adopté une longueur de 128 caractères car c'est la plus courante sur les micro-ordinateurs. Comme on le voit, un enregistrement physique peut contenir plus d'un enregistrement logique.

STRUCTURE DU FICHIER REPERTOIRE



Position des zones dans l'enregistrement logique

Solutions du test 6

1 / c) Les mémoires de masse sont permanentes et peuvent conserver de grandes quantités de données pendant une durée pratiquement illimitée.
La mémoire de l'ordinateur conserve les données pendant la durée de leur traitement. Les données de sortie (traitées) sont ensuite de nouveau transférées vers les mémoires de masse, soit pour y être conservées,* soit pour libérer la mémoire de traitement en vue de l'accomplissement d'une nouvelle tâche.**
Sur les ordinateurs qui travaillent en multiprogrammation, on met souvent les résultats intermédiaires « de côté » sur des disques, mémoires de masse à accès direct les plus rapides. Lors d'une pause dans l'exécution d'un programme, les données correspondantes sont transférées sur un disque pour laisser la place à celles d'un autre programme.
Le programme interrompu est repris dès que l'événement qui a provoqué son interruption disparaît, et que les ressources nécessaires à son exploitation sont disponibles. Ce mouvement des données et des programmes de la mémoire centrale aux mémoires de masse est géré automatiquement par le système d'exploitation ; il permet d'utiliser au mieux les ressources de l'unité de traitement.

---

2 / 80000 pour les plus petits, 1200000 pour les plus grands. Une page, entièrement écrite, de cette encyclopédie, contient en moyenne deux colonnes de 50 lignes chacune. Une ligne se compose de 45 caractères environ. Une page remplie contient donc, à peu près, $2 \times 50 \times 45 = 4500$ caractères. En fait, une page moyenne contient 3000 caractères environ car les photographies et les schémas occupent un tiers de la surface totale. Un tascicule comporte 24 pages, ce qui donne $24 \times 3000 = 72000$ caractères. Une disquette de 1200000 caractères pourrait donc abriter 16 fascicules et l'ouvrage entier pourrait tenir sur 5 disques de quelques grammes qui ne prendraient pas plus de place que quelques teuilles de papier.

---

3 / a) Les fichiers sont des parties des mémoires de masse qui regroupent sous un même nom (le nom du tichier) des données homogènes.
b) Les enregistrements peuvent être physiques ou logiques. Les enregistrements physiques constituent la structure physique du fichier, les enregistrements logiques définissent sa structure logique.
c) Les zones sont les emplacements qui contiennent les données dont sont constitués les enregistrements logiques.

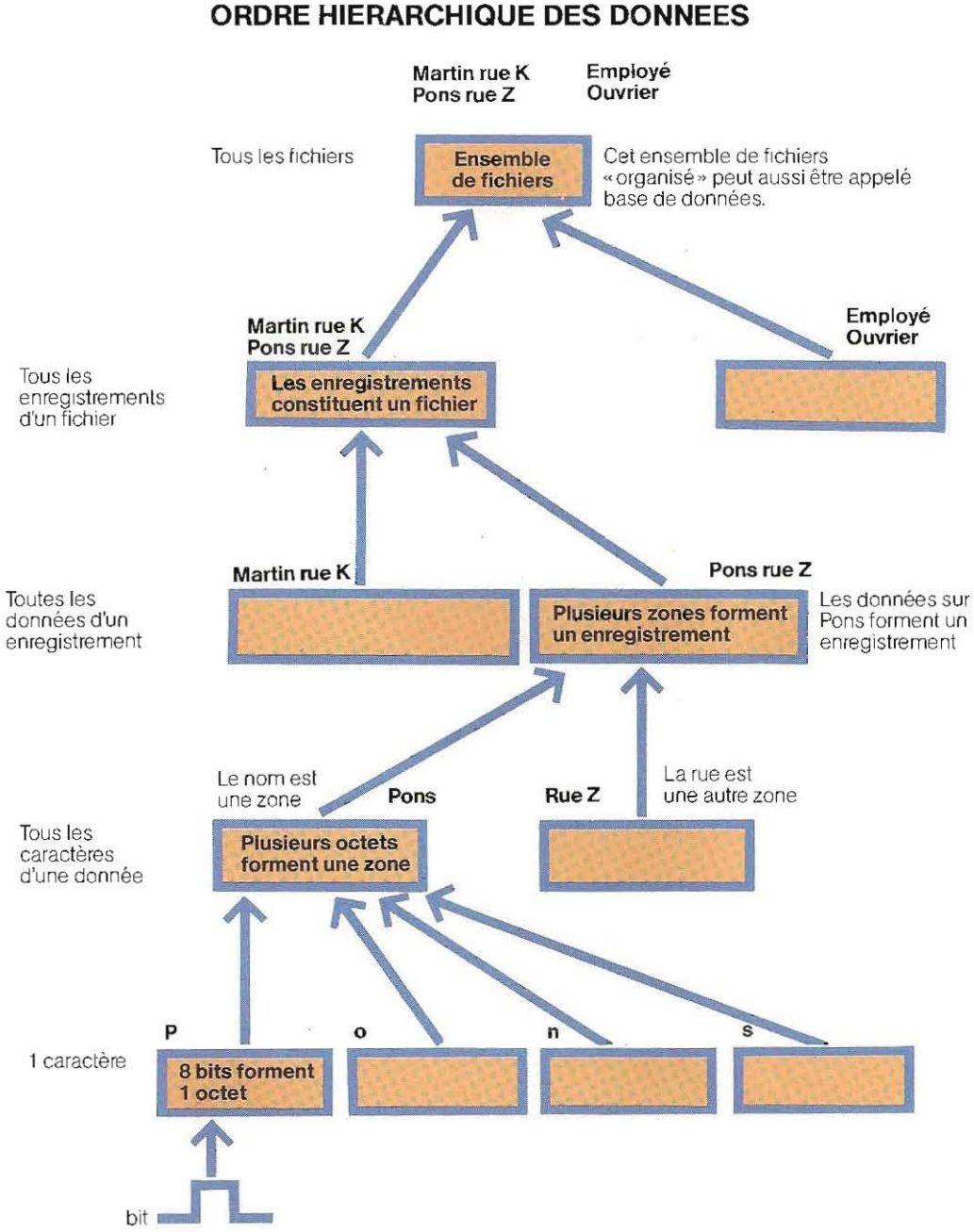
---

4 / On les utilise principalement pour sauvegarder les fichiers sur disque. En effet, leur accès est uniquement séquentiel, ce qui n'est pas pratique pour la manipulation fréquente des fichiers. Les disques, en revanche, sont à accès direct et permettent le transfert rapide des données.

---

* Fichiers permanents.
** Fichiers temporaires.

5 / Bit, octet, zone, enregistrement, fichier, ensemble de fichiers. Le schéma ci-dessous illustre la hiérarchie de ces éléments.



Lorsqu'on voudra lire, par exemple, l'enregistrement logique 1, une partie de l'enregistrement 2 sera donc également transférée en mémoire. C'est le **système d'exploitation** (c'est-à-dire l'ensemble des programmes qui régissent les fonctions de base du système) qui se chargera alors de séparer les données utiles des données inutiles.

Le système d'exploitation est, en général, fourni avec la machine. C'est un ensemble de programmes de base indispensables au fonctionnement de l'ordinateur, et notamment à la gestion des entrées et des sorties sur disque, imprimante, clavier ou écran. L'utilisateur peut ainsi programmer même s'il ne sait pas comment est construite sa machine. C'est le système d'exploitation qui traduit les instructions rédigées en langage évolué (comme le Basic) en instructions en langage machine (micro-instructions).

L'organisation des fichiers sur les micro-ordinateurs

Sur les gros systèmes et sur les mini-ordinateurs les plus perfectionnés, il existe des programmes de gestion de fichiers tout prêts qui épargnent au programmeur les détails de la recherche ou de la mise en ordre des données. Ces programmes font le lien entre le programme d'application écrit par le programmeur et la partie du système assurant la gestion des fichiers. Il suffit d'indiquer, par des instructions, l'opération (une recherche par exemple), que l'on désire effectuer. Les programmes de gestion interpréteront alors l'instruction pour mettre en œuvre le traitement correspondant.

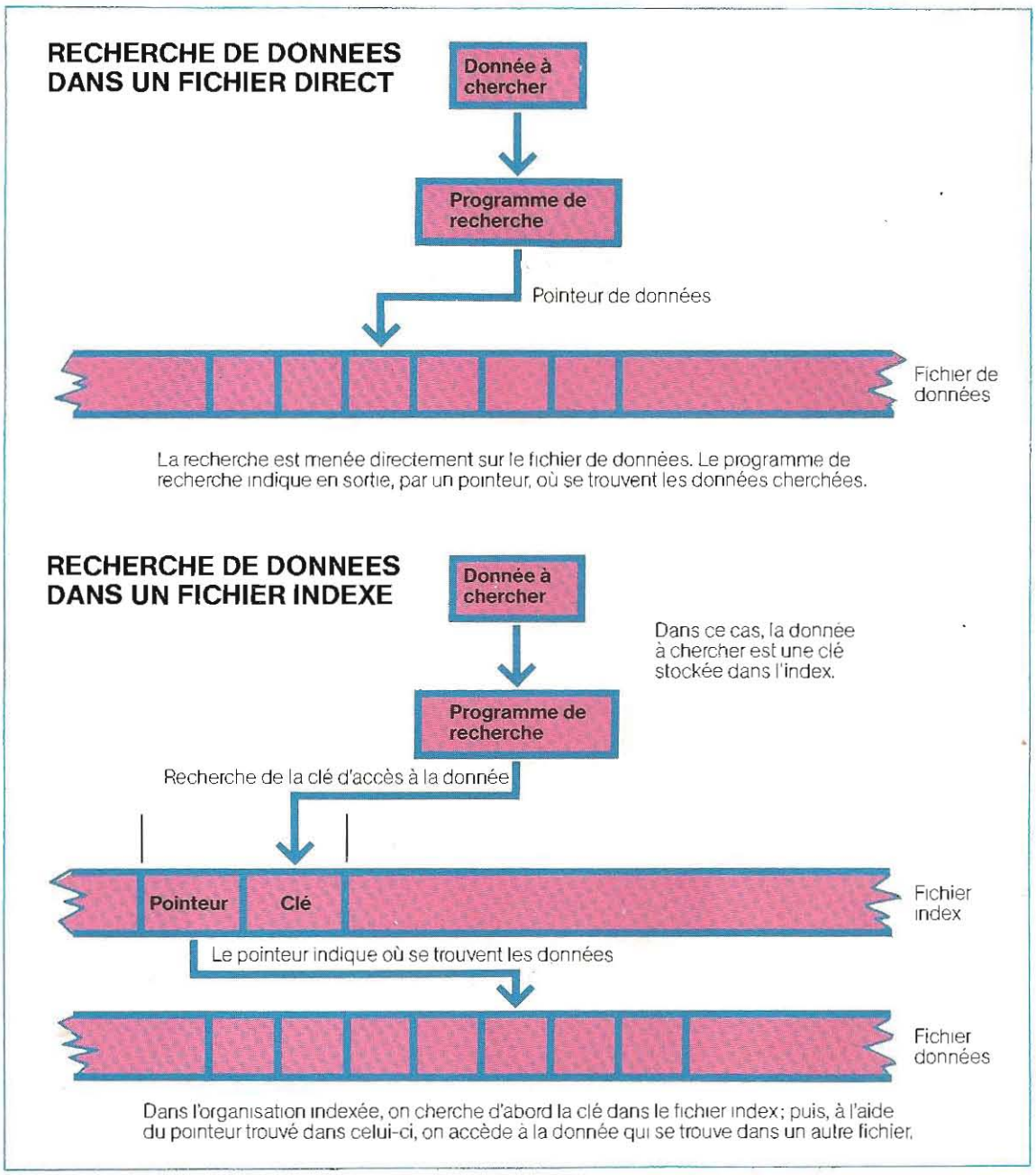
On dispose moins fréquemment de ce type de logiciel sur les petits systèmes (micro-ordinateurs et ordinateurs personnels). D'ailleurs, leur prix est élevé par rapport à celui de la machine. De plus, les programmes de gestion interne doivent être généralisés de façon à couvrir toutes les applications possibles. Leur extension entraîne l'occupation d'un plus grand espace sur les disques comme dans la mémoire et ceci est un gros inconvénient sur les petits systèmes où l'espace mémoire fait justement défaut.

Pour certaines applications, il faut écrire, en plus

Terminaux en phase de contrôle après fabrication.



C. O'Rear/West Light-Grazia Neri



du programme proprement dit, les instructions nécessaires à la recherche et à la mise en ordre des données, sans généraliser mais, au contraire, en trouvant une solution spécifique.
Notons l'existence de logiciels de gestion de fichiers qui, vendus séparément de l'ordinateur, permettent la création de fichiers, l'enregistrement, la consultation, la modification ou la suppression des données.
Les deux principales méthodes de recherche sont la **recherche logique** et la **recherche dichotomique**. On peut les appliquer soit aux fichiers index, soit directement aux fichiers données. Dans le premier cas, on doit, après avoir trouvé la clé dans le fichier index, se servir du pointeur pour localiser l'enregistrement dans le fichier données.
Le schéma ci-dessus illustre les méthodes de recherche sur fichier direct et sur fichier séquentiel indexé.
Précisons tout d'abord quels sont les principaux types de fichiers utilisés sur les micro-

ordinateurs. On distingue :

- **les fichiers directs (ou aléatoires),**
- **les fichiers séquentiels,**
- **les fichiers indexés.**

Les fichiers directs (ou aléatoires)

Dans ce type de fichier, l'adresse d'une donnée correspond au numéro de l'enregistrement qui la contient : les données du groupe 1 se trouvent dans l'enregistrement 1, celles du groupe 2 dans l'enregistrement 2, et ainsi de suite.

L'accès direct (DAM, sigle dérivé de l'anglais Direct Access Method) s'appelle aussi l'**accès aléatoire** ou **sélectif**. Il permet d'écrire ou de lire un enregistrement en indiquant sa position physique sur le disque, c'est-à-dire son adresse absolue, ou son adresse relative à l'intérieur du fichier.

On peut donc lire directement un enregistrement si l'on connaît son numéro (position physique) ou sa position relative par rapport à d'autres enregistrements.

Dans l'adressage relatif, le numéro de l'enregistrement lui-même peut être utilisé comme une clé. C'est au programme d'application de déterminer à quel endroit du fichier se trouve la donnée (numéro de l'enregistrement).

Dans l'organisation d'un répertoire téléphonique, on peut attribuer à chaque enregistrement le numéro correspondant à l'ordre dans lequel les données ont été saisies : la première adresse écrite se trouve dans l'enregistrement 1, et ainsi de suite. Pour retrouver une donnée dans le fichier, il suffira de trouver le numéro de son enregistrement.

Comme les données ont été enregistrées en ordre chronologique, il faudra, à chaque recherche, lire tout le fichier. Si, par exemple, nous voulons les numéros de téléphone de Paris, nous devrons lire tout le fichier car nous ignorons où ces numéros commencent et où ils finissent.

On peut pallier cet inconvénient, qui rallonge beaucoup la durée du traitement, en classant le fichier des données selon l'ordre alphabétique des noms de villes, avant de lancer la recherche sur Paris. Ainsi, dès que la donnée rencontrée (Pau, par exemple) aura une valeur supérieure à celle de Paris, on pourra abandonner la recherche, la ville Paris ne pouvant pas être contenue dans les enregistrements suivants. Cette procédure logique est illustrée schématiquement par l'organigramme de la page 249.

L'accès direct aux enregistrements est le principal avantage de l'organisation des fichiers directs. On a toute liberté pour lire ou écrire un enregistrement. On peut, par exemple, lire l'enregistrement 100 avant l'enregistrement 50. C'est pour cela qu'on parle de fichiers à accès aléatoire (random access, en anglais).

Les fichiers séquentiels

Ce sont des fichiers classés séquentiellement par numéro d'enregistrement et dans lesquels il n'est pas possible d'accéder à un enregistrement, en lecture ou en écriture, avant d'avoir lu ou écrit tous ceux qui précèdent.

On les utilise pour stocker des données en nombre déterminé, qu'on doit toutes lire ou écrire en même temps. C'est pourquoi les tables sont souvent contenues dans des fichiers séquentiels. Les enregistrements sont lus ou écrits dans l'ordre physique, et il n'est possible de faire une insertion dans ce type de fichier qu'en le récrivant tout entier, à moins, bien sûr, que l'ajout ne soit à placer à la fin du fichier.

En principe, les mises à jour sont donc impossibles ; cependant, certains systèmes permettent des modifications.

Les fichiers indexés

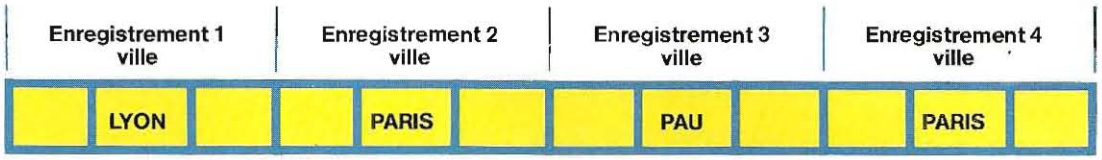
Ce sont des fichiers directs couplés à un index, lequel se trouve dans un autre fichier, ou bien dans le fichier de données lui-même. Ils servent pour de nombreuses applications et nous étudierons donc plus en détail leur fonctionnement.

Un fichier indexé comporte deux structures : le fichier index et le fichier données (voir schéma p. 250). Le fichier index est une suite d'enregistrements qui contiennent les clés des enregistrements de données et des pointeurs. Une clé correspond à un enregistrement et un seul. Parfois, des index secondaires permettent d'accéder aux enregistrements par des voies différentes.

Par ailleurs, la même clé peut apparaître plusieurs fois dans le fichier index. Si l'on a pris le nom de famille comme clé, on risque de rencontrer le cas de plusieurs personnes qui portent le même nom. Il faudra alors préciser le numéro

ORGANIGRAMME DU TRAITEMENT PERMETTANT D'INTERROMPRE UNE RECHERCHE DANS UN FICHIER ORDONNE

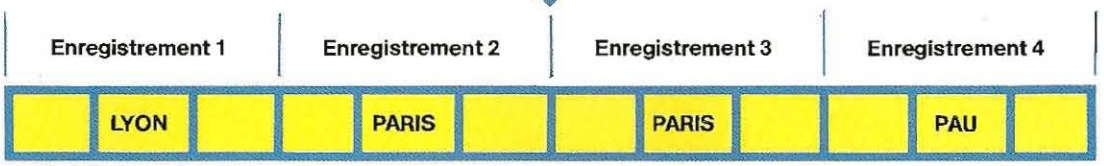
Fichier d'origine



Dans le fichier d'origine, les données sont placées suivant l'ordre de leur introduction. Pour faire une recherche par ville, il faut donc lire le fichier dans sa totalité.

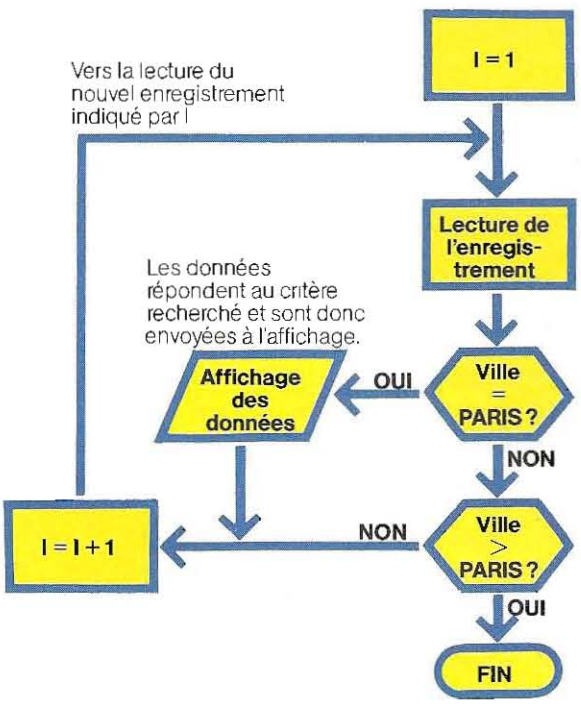


(c'est l'ordre alphabétique qui est retenu)



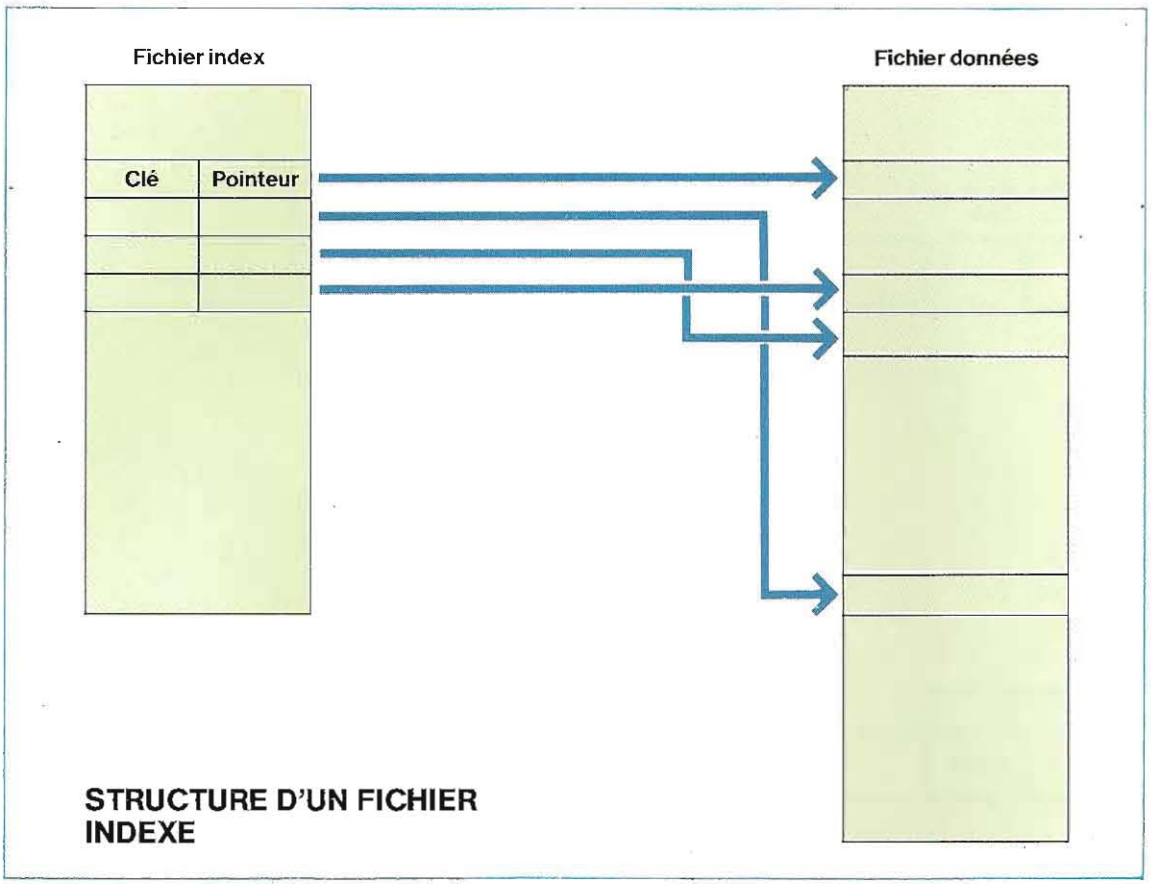
Le nouveau fichier est ordonné

On peut maintenant interrompre la recherche à l'enregistrement 4 puisque la donnée qu'il contient dans sa zone ville (Pau) est supérieure à la donnée choisie (Paris). En effet, le fichier étant classé dans l'ordre alphabétique des noms de villes, il ne peut pas y avoir d'enregistrements concernant Paris au-delà.



Pointeur de l'enregistrement duquel démarre la recherche

Il n'y a plus d'enregistrement concernant Paris. On peut donc interrompre le traitement.



STRUCTURE D'UN FICHIER INDEXE

d'ordre de chaque enregistrement par rapport aux enregistrements ayant une même clé.

Le fichier données contient les enregistrements où se trouvent les données. Il peut éventuellement être organisé en fonction de la valeur de la clé, selon la méthode que l'on désire utiliser.

Les index secondaires permettent d'accéder aux enregistrements par d'autres clés que la clé principale (primaire). Ainsi, on peut lire les enregistrements d'un fichier clients en indiquant soit le numéro de code, soit le nom du client. L'index primaire utilisera le numéro comme clé, tandis que l'index secondaire prendra le nom comme clé secondaire. On aura ainsi deux moyens indépendants d'accès au fichier (c'est ce qu'illustre le schéma page 251).

Il est parfois nécessaire d'utiliser une clé composite, regroupant plusieurs valeurs. Pour cela, on combine les données afin qu'elles forment un tout : c'est la **concaténation**.

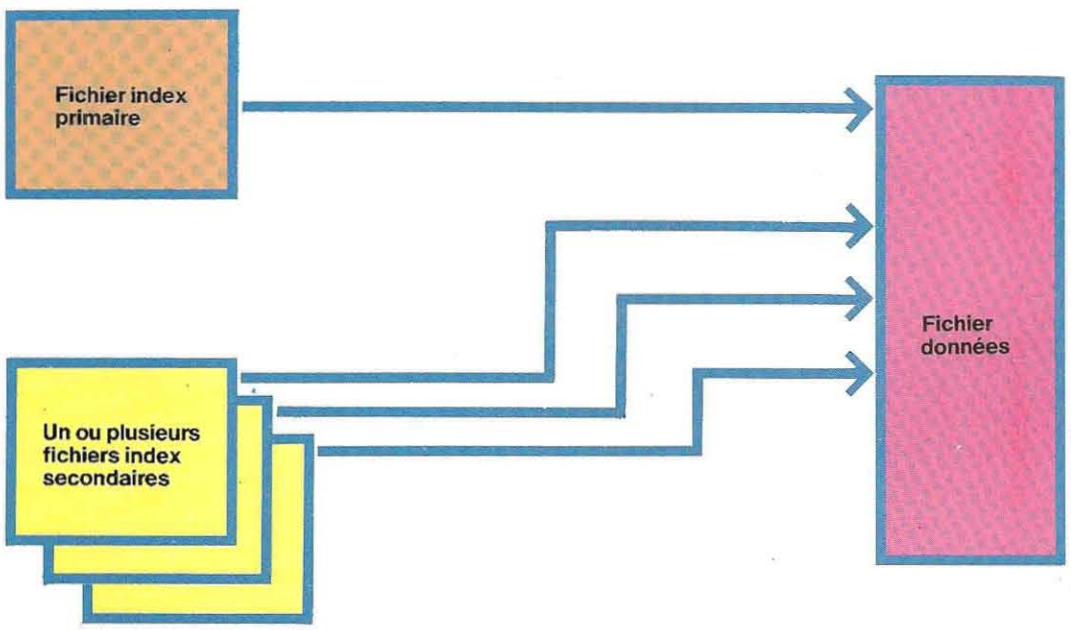
Une société peut, par exemple, souhaiter cer-

ner facilement l'activité de chacune de ses succursales. Si la clé des enregistrements se compose du code de la succursale et du numéro du client, les clients d'une même succursale seront regroupés et il sera possible de traiter les données les concernant indépendamment de celles des autres clients. Toutefois, un traitement global du fichier sera toujours possible.

Les fichiers indexés offrent la plus grande souplesse de manipulation, et permettent d'effectuer la recherche d'une donnée déterminée dans les meilleures conditions, mais ce sont néanmoins les plus difficiles à gérer. Les programmes de gestion des fichiers indexés présentent en effet une certaine complexité et s'adressent donc à des programmeurs déjà expérimentés.

Cela ne diminue en rien l'intérêt de l'organisation indexée, et nous étudierons, dans le chapitre suivant, la méthode ISAM de gestion des fichiers indexés, méthode utilisable sur les micro-ordinateurs et les ordinateurs individuels.

STRUCTURE D'UN FICHIER A INDEX SECONDAIRES



L'informatique au bureau

Le bureau est aujourd'hui encore le centre vital d'une entreprise industrielle ou commerciale. La diminution progressive, depuis les années cinquante, de la main-d'œuvre employée par les industries manufacturières, l'élargissement du secteur tertiaire et l'avènement de l'électronique ont entraîné la création de nombreux emplois de bureau. Aux Etats-Unis, par exemple, le pourcentage des personnes employées dans le secteur du traitement de l'information est aujourd'hui supérieur à celui des autres branches d'activité.

L'introduction du traitement électronique des données a permis la première phase d'une évolution qui aboutira peut-être à la disparition du papier comme support de l'information. Elle débouche déjà sur d'intéressantes innovations en matière de représentation graphique et de traitement de texte.

L'ordinateur a également supprimé les travaux ennuyeux de manipulation des chiffres. Les comptables et les directeurs des ventes mettent désormais en œuvre des méthodes de traitement des données et d'étude statistique autrefois inimaginables.

On est forcé de constater que le progrès dans les habitudes de travail passe souvent par l'automatisation du calcul et de l'écriture. Le prix d'un ordinateur personnel est d'ailleurs de plus en plus modéré. Certes, sa rapidité, la capacité de sa mémoire et la portée de ses analyses ne peuvent être comparées à celles de ses frères aînés ; néanmoins, il est tout à fait adapté à de nombreuses tâches courantes, telles que la tenue des livres de comptes dans un service de comptabilité.

L'utilisation systématique d'ordinateurs dans les bureaux entraîne très souvent une restructuration des locaux eux-mêmes. Les « postes de travail » sont de petites unités destinées à une ou deux personnes au sein d'un vaste bureau sans cloisons fixes. Chaque unité comprend le matériel et les instruments nécessaires, et notamment un écran, un clavier et une interface de connexion à la mémoire de l'ordinateur et au système de télécommunications de l'entreprise.

L'espace réservé à chaque poste de travail est conçu et aménagé pour assurer le calme indispensable à de bonnes conditions de travail.

Des études ergonomiques approfondies ont été menées (souvent à l'aide de l'ordinateur), et ont permis de souligner l'importance de certains facteurs inhérents à l'utilisation de ce matériel, telles l'intensité de l'éclairage, la distance entre les postes de travail ou la chaleur produite par les machines. Quant au bruit, si la plupart des ordinateurs fonctionnent en silence, certaines parties de la machine émettent toutefois des ultrasons qui, en général, ne sont pas perceptibles par l'oreille humaine, mais peuvent néanmoins provoquer des troubles. Tous ces aspects doivent être étudiés attentivement.

Le traitement automatique des données s'étend aux textes rédigés, et il ne faudra sans doute plus très longtemps pour que la simple machine à écrire soit troquée contre le clavier d'un ordinateur capable d'effectuer toutes les opérations ultérieures de correction et de mise en page. D'ailleurs, prévoyant cette évolution, certains fabricants de machines à écrire électroniques ont prévu l'adjonction à la machine du matériel nécessaire à la mémorisation du texte sur une disquette.

Grâce à l'ordinateur, un message peut être codifié et transmis à quelqu'un par l'intermédiaire du clavier, sur lequel on compose les indications de priorité et de confidentialité du texte, ainsi que l'adresse électronique du destinataire. Une fois arrivé, le message est inséré parmi les autres données en instance et transféré dans la mémoire selon son degré de priorité. Le destinataire sélectionne les messages reçus à l'aide d'un clavier spécial et les lit sur son écran.

De cette manière, aucune feuille de papier ne circule dans le bureau, aucun document imprimé n'a besoin d'être jeté ou classé, et aucun intermédiaire n'est plus nécessaire au transport du message. Il s'agit d'un véritable système de courrier électronique. Ce qui n'empêche pas, d'ailleurs, de déclencher l'impression de documents sur papier à la sortie de l'ordinateur dans les cas où cela est nécessaire.

Mais la plupart du temps on renonce à l'utilisation du papier, et on peut penser que cette tendance se généralisera complètement quand on disposera d'un réseau de télécommunications doté des capacités de mémorisation adéquates. De plus, l'emploi des liaisons par fibres optiques porteuses d'impulsions laser codifiées, ou par satellites de télécommunications, devrait multiplier le nombre d'applications de ces systèmes.

Il va sans dire qu'il faudra prévoir de quoi remplacer l'autorité de la signature, une fois que l'usage du document imprimé sera sorti des habitudes. Il faudra également mettre en place des moyens de prévention efficaces contre l'introduction dans les ordinateurs d'erreurs humaines qui pourraient entraîner des conséquences désastreuses.

Quant aux problèmes spécifiques posés par l'ampleur des télécommunications, par l'archivage des données et par les opérations multilingues, ils seront sans aucun doute résolus bien avant celui de la restructuration des organigrammes de l'organisation du travail de certaines catégories du personnel. Archivistes, dactylographes, employés de bureau, entre autres, devront se recycler en vue d'autres activités s'ils ne veulent pas se trouver privés d'emploi, la machine effectuant désormais leur tâche et de façon plus rapide.

A la faveur de cette adaptation à une nouvelle réalité du travail de bureau, on constate fréquemment que de nouveaux rapports s'établissent entre les salariés d'une même entreprise, les différences de niveau culturel ayant plutôt tendance à s'estomper.

Nombre de sociétés sont déjà équipées d'ordinateurs pour le travail de bureau quotidien. Les maisons d'édition scientifique, en particulier, ont mis en place des centres de traitement destinés à supprimer les pertes de temps (et d'argent) dues au travail de documentation, de consultation des sources d'information et de préparation des rapports nécessaires au travail des rédacteurs.

Désormais, l'auteur transmet son manuscrit sur une bande magnétique et l'adresse à son éditeur par téléphone. Le texte est alors introduit dans une machine de traitement de texte et enregistré dans un fichier. Toute personne intéressée (et autorisée) par l'étude concernée peut ensuite « appeler » ce texte et le voir s'afficher sur l'écran de son poste de travail.

Ce procédé est déjà adopté dans de nombreuses bibliothèques universitaires pour l'archivage sur bandes d'articles de revues scientifiques et techniques.

Les salles de rédaction des journaux ont également connu des changements radicaux dus à l'utilisation de l'ordinateur. Journalistes, rédacteurs et compositeurs disposent de consoles de visualisation reliées à un ordinateur central, et peuvent composer un journal



Brecht Einzig Ltd



Lyndsay Westhead Architects/Environement

L'emploi de l'ordinateur dans les bureaux entraîne souvent une restructuration des locaux eux-mêmes.

entier (titres, textes et illustrations) en opérant uniquement en liaison avec la mémoire du système.

L'image électronique du journal terminé est conservé sur une disquette, puis ensuite transférée à de la pellicule photo qui permet de réaliser les films nécessaires au fonctionnement de la machine à imprimer.

En fait, cette dernière opération est utile parce que l'opération finale est encore l'impression sur du papier, mais on peut très bien envisager un autre mode de diffusion des informations. Des écrans installés dans les moyens de transport en commun, ou même un certain nombre de nos actuels postes de télévision disposés dans les grands magasins, dans les restaurants, les cafés, les salons de coiffure, les laveries automatiques, entre autres, pourraient très bien remplacer les journaux pour la transmission des informations. C'est déjà un peu ce qui se passe mais à une moindre échelle avec les journaux télévisés, et il suffirait de développer le système.

Méthode ISAM de gestion d'un fichier indexé

Cette méthode (Indexed Sequential Access Method) de gestion des fichiers indexés permet d'accéder aux enregistrements aussi bien séquentiellement que directement au moyen d'une clé (ou donnée d'identification), associée à chaque enregistrement.

Selon le type d'application, on aura avantage à utiliser l'un ou l'autre des ces deux modes d'accès. Dans les deux cas, la structure du fichier à clé se révèlera extrêmement utile. Elle est, en effet, beaucoup plus souple que celles qui dérivent des autres formes d'organisation, et se révèle particulièrement adaptée aux applications complexes en temps réel et en mode conversationnel.

Les fichiers indexés sont un outil très commode, notamment pour la lecture des informations contenues dans un enregistrement par simple communication de la clé.

Supposons, par exemple, que dans le cadre de la gestion d'un magasin, on veuille connaître la quantité en stock d'un article déterminé, afin de procéder à une commande éventuelle.

Il suffira d'utiliser un fichier indexé auquel on donnera pour clé le numéro de code de l'article, et de prévoir un programme qui demande d'abord à l'opérateur le code de l'article qui l'intéresse, et qui aille ensuite lire l'enregistrement correspondant, pour enfin afficher sur l'écran la quantité en stock.

L'utilisation d'index secondaires permettra d'attribuer aux enregistrements une seconde clé, qui sera, par exemple, la désignation de l'article. On pourra ainsi, même si on ne connaît pas le numéro de code d'un article, accéder à l'enregistrement qui le concerne.

Un autre avantage du fichier indexé est qu'il permet la vérification des données introduites, dans la mesure où une donnée peut être utilisée comme clé. Ainsi, dans un programme de paie, l'enregistrement où se trouve le nombre d'heures de travail hebdomadaires d'un employé contient également son numéro de matricule.

L'opérateur tape donc le numéro de matricule qui sert de clé d'accès aux informations

L'utilisation de l'ordinateur n'a pas seulement bouleversé la notion traditionnelle d'archive. Elle a aussi apporté de nouvelles méthodes de travail permettant des recherches très rapides.



C. O'Rear/West Light-Grazia Neri

concernant un employé. Si la valeur entrée ne correspond à aucune clé, l'opérateur en est averti par la machine, et peut introduire le matricule exact.

Les opérations d'accès aux données

Cinq méthodes de lecture différentes permettent d'accéder à un enregistrement dans un fichier indexé.

- **Lecture directe par clé.** On doit indiquer une valeur de clé afin d'obtenir le pointeur de l'enregistrement correspondant. Dès que le système a eu connaissance de la clé, il recherche, dans le fichier d'index, le pointeur correspondant à l'enregistrement. En cas de recherche infructueuse, un message d'erreur est émis.
  Cette méthode de lecture directe est utile pour la consultation et la mise à jour des fichiers lorsqu'on veut traiter des enregistrements déterminés dont on connaît la clé.

- **Lecture générale par clé.** Cette méthode est semblable à la précédente à cette différence près que, si l'enregistrement demandé n'existe pas, c'est le pointeur correspondant à l'enregistrement de clé immédiatement supérieure qui sera retenu.
  On peut ainsi opérer des regroupements logiques entre les enregistrements au moyen de clés partielles. C'est un moyen très efficace de sélection des enregistrements de données.

- **Lecture séquentielle par clé.** Cette méthode permet de lire des enregistrements en ordre logique, c'est-à-dire dans l'ordre de leurs clés, indépendamment de leur implantation physique dans le fichier.
  La lecture peut partir de n'importe quel enregistrement logique, selon l'ordre désiré.
  Combinée à la précédente, cette méthode offre la possibilité de lire des groupes cohérents d'enregistrements.

- **Lecture séquentielle en ordre physique.** Elle consiste à lire les enregistrements du fichier de données dans l'ordre physique où ils se présentent, sans se préoccuper de l'index. Cette méthode présente un intérêt lorsque seule compte, pour le traitement, l'analyse des données, et non leur ordre de lecture, car les temps d'accès sont alors réduits dans la mesure où l'on supprime l'étape de recherche dans l'index.

- **Lecture directe par numéro d'enregistrement.** Elle permet d'accéder aux données par leur numéro d'enregistrement. Il est donc possible de consulter un fichier de données aussi bien en accès direct à partir de la clé qu'en mode DAM.

Il existe également plusieurs façons de pratiquer des insertions et des suppressions de données.
Le système de gestion des fichiers indexés offre la possibilité de supprimer soit les enregistrements de données, soit leur clé, soit encore les deux. Le travail de programmation s'en trouve simplifié, on gagne de la place sur le disque, et la réorganisation périodique des fichiers perd de son urgence.
Il existe une instruction spéciale pour éliminer d'un fichier des enregistrements déterminés. Cette fonction supprime la clé des enregistrements dans le fichier index, supprimant aussi les enregistrements correspondants. Mais si l'on veut supprimer la clé d'un enregistrement tout en conservant celui-ci, on doit utiliser la fonction de suppression de clé.
La clé disparaît alors de l'index, tandis que l'enregistrement qui contient les données reste intact.
Cependant, il n'est plus accessible par sa clé mais devra être lu séquentiellement.
C'est donc le programmeur qui est maître de la suppression des données.
La suppression de clé est particulièrement intéressante dans les applications où l'on prévoit de consulter les enregistrements ultérieurement, pour la constitution d'archives, par exemple. De plus, elle est indispensable pour supprimer, des index secondaires, les clés des enregistrements détruits.
La mise à jour des données en temps réel est certainement l'une des meilleures applications de l'organisation indexée.
Une donnée peut être tenue à jour au fur et à mesure du déroulement du programme d'application. Elle devient ainsi une information fiable en temps réel.
Ces modifications immédiates sont très souvent associées à la lecture des enregistrements.

C'est ainsi que, dans une banque, on peut enregistrer directement les prélèvements et les dépôts effectués sur les comptes en lisant les enregistrements à l'aide de leur clé (le numéro de compte) et en déduisant ou en ajoutant les sommes concernées avant de récrire les enregistrements dans le fichier.

La structure générale du fichier ISAM

L'organisation ISAM utilise deux structures de données : l'index et le fichier des données proprement dites. Plusieurs index coexistent lorsqu'on utilise des clés secondaires pour ouvrir différentes voies d'accès aux enregistrements de données. Les fichiers de données en ISAM sont enregistrés selon le format Basic et sont compatibles avec toutes les fonctions du Basic.

L'utilisation de programmes utilitaires comme Sort ou Merge ne présuppose donc aucune modification. Sort est un programme de tri tandis que Merge (fusion) sert à réunir plusieurs fichiers en un seul.

Il faut cependant remarquer que ces programmes entraînent la destruction des relations logiques entre le fichier de données et l'index.

Le fichier de données étant un fichier Basic à accès direct (aléatoire), il est possible de le manipuler par le biais des instructions Basic correspondantes. Quant au fichier index, il est constitué d'enregistrements contenant chacun un certain nombre de clés ; à chaque clé correspond un numéro d'enregistrement.

Lorsqu'une même clé revient plusieurs fois, le mécanisme de recherche est légèrement différent. On distingue donc deux sortes de fichier index, selon qu'ils acceptent les clés dupliquées ou les clés uniques.

Les clés sont toujours enregistrées dans l'ordre physique, mais elles sont reliées par des pointeurs qui rétablissent l'ordre alphanumérique. Une structure « en arbre » permet de maintenir cet ordre logique. L'arbre utilisé par la méthode ISAM s'appelle un arbre binaire.

La structure en arbre binaire équilibré

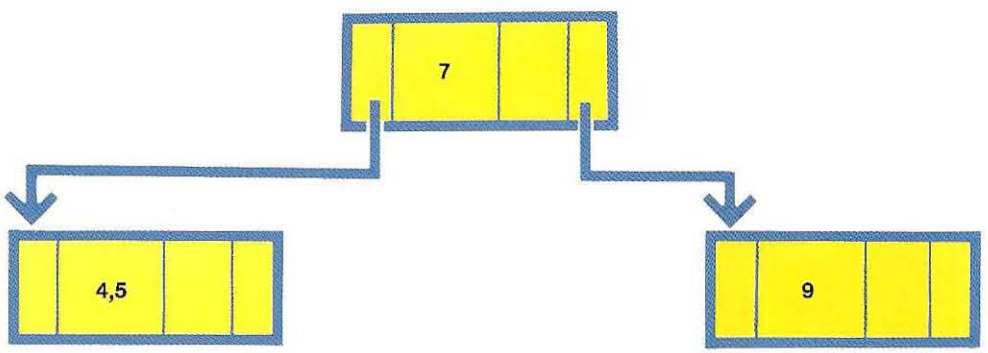
Avant d'aborder l'étude de l'arbre binaire (B-tree, en anglais) utilisé par ISAM, nous allons examiner une forme d'arbre équilibré plus simple, qui nous permettra d'en mieux comprendre le fonctionnement.

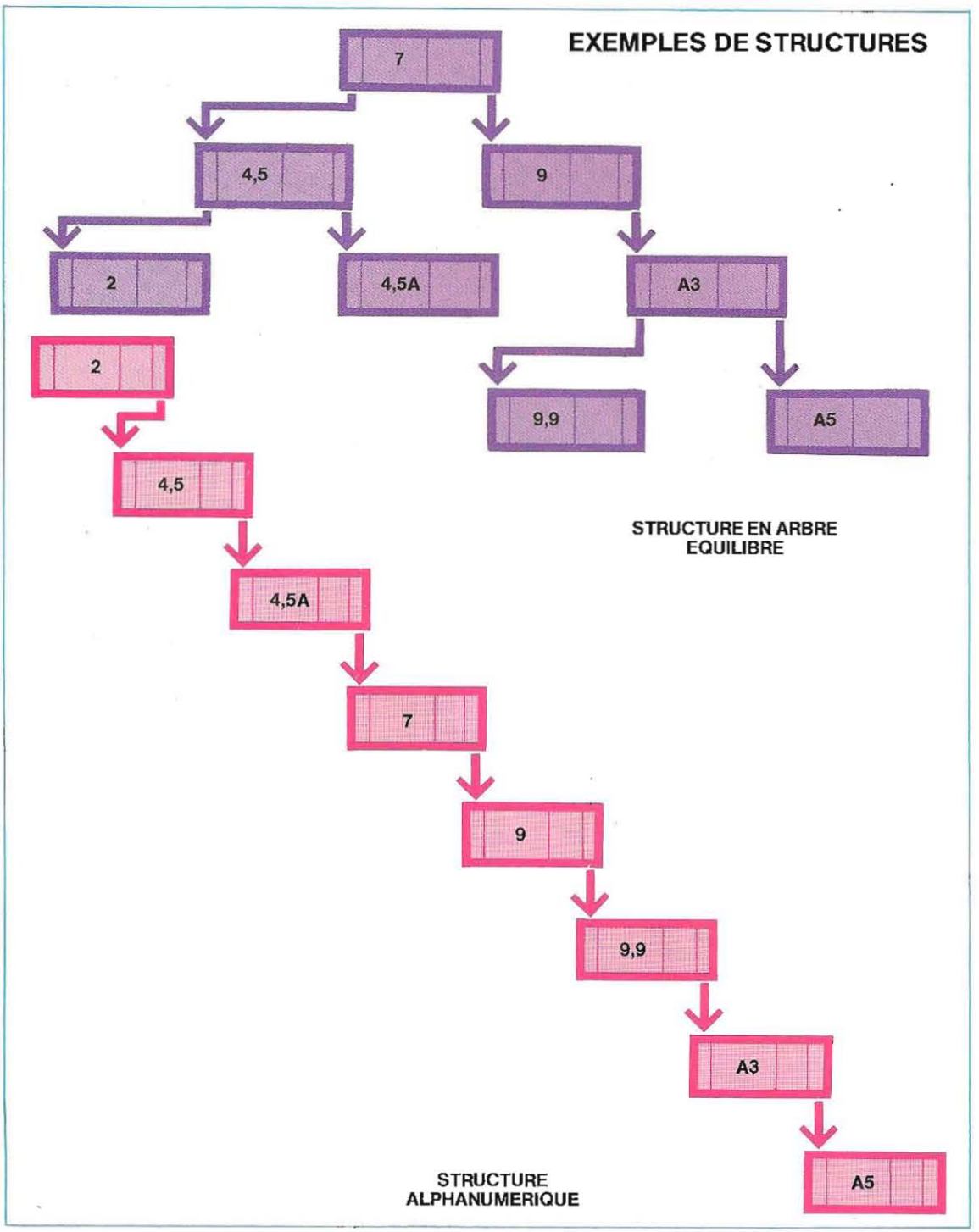
Dans un type particulier d'arbre binaire, chaque clé occupe un embranchement et deux pointeurs lui sont associés. L'un indique la clé qui précède immédiatement, l'autre celle qui suit immédiatement. Ces clés se trouvent, toutes deux, au niveau immédiatement inférieur.

Le schéma ci-dessous montre la structure binaire obtenue par l'ajout des clés 4,5 et 9 à la clé 7.

Chaque fois qu'un enregistrement est ajouté au fichier, la valeur de sa clé est comparée à celle de la clé qui se trouve tout en haut de l'arbre et que l'on appelle, en fait, la racine. Si la clé de l'enregistrement à ajouter est supérieure à la racine, on la compare à nouveau mais cette fois avec la clé indiquée par le pointeur qui part à droite de la racine. Si elle est inférieure, on la compare avec la branche de gauche. On effectue ainsi des comparaisons successives jusqu'à ce qu'on trouve la

EXEMPLE DE STRUCTURE EN ARBRE EQUILIBRE





position qui revient à la clé, au niveau le plus bas de l'arbre.

Ainsi, en ajoutant, par exemple, les clés 2, A3, A5, 4,5A et 9,9 à la structure illustrée page 256, nous obtiendrons l'arbre représenté ci-dessus en haut, à droite. Tant que l'arbre demeure équilibré, c'est-à-dire tant que le nombre de clés situées à la gauche d'une clé est comparable à celui des clés qui se trouvent à sa droite, la durée de recherche d'une clé est optimale.

On parvient ainsi à la clé A5 après trois com-

paraisons seulement (avec les clés 7, 9 et A3). Si, en revanche, les clés étaient disposées en ordre alphanumérique, on obtiendrait la structure représentée en bas, à gauche, du schéma de la page 257, et il faudrait alors sept comparaisons pour parvenir à cette même clé. Il est nécessaire de disposer d'un algorithme pour maintenir l'équilibre de l'arbre et éviter des recherches trop longues.

La structure en arbre ISAM

L'arbre binaire utilisé par ISAM est plus complexe en raison des deux types d'enregistrements qui coexistent dans cette méthode et auxquels correspondront des branches index et des branches terminales.

Les branches terminales sont les plus basses de l'arbre ; ce sont elles qui contiennent les clés et les numéros des enregistrements de données correspondants. Les index occupent, au contraire, tous les niveaux de l'arbre, excepté le niveau inférieur et constituent une voie d'accès aux branches terminales (voir schéma ci-dessous). Chaque ramification terminale possède, en outre, deux pointeurs, qui la rattachent à la branche terminale précédente et à la suivante. On peut ainsi accéder à toutes les clés en respectant l'ordre ASCII.

La longueur des enregistrements du fichier index est fixe, tandis que la longueur des clés est variable.

Le nombre de fichiers ouverts et les zones tampon

L'utilisateur doit déclarer le nombre maximum de fichiers index qui peuvent être ouverts en même temps. L'organisation ISAM bénéficie d'un programme de base* qui permet de définir ce nombre qui sera compris entre 1 et 15, par exemple.

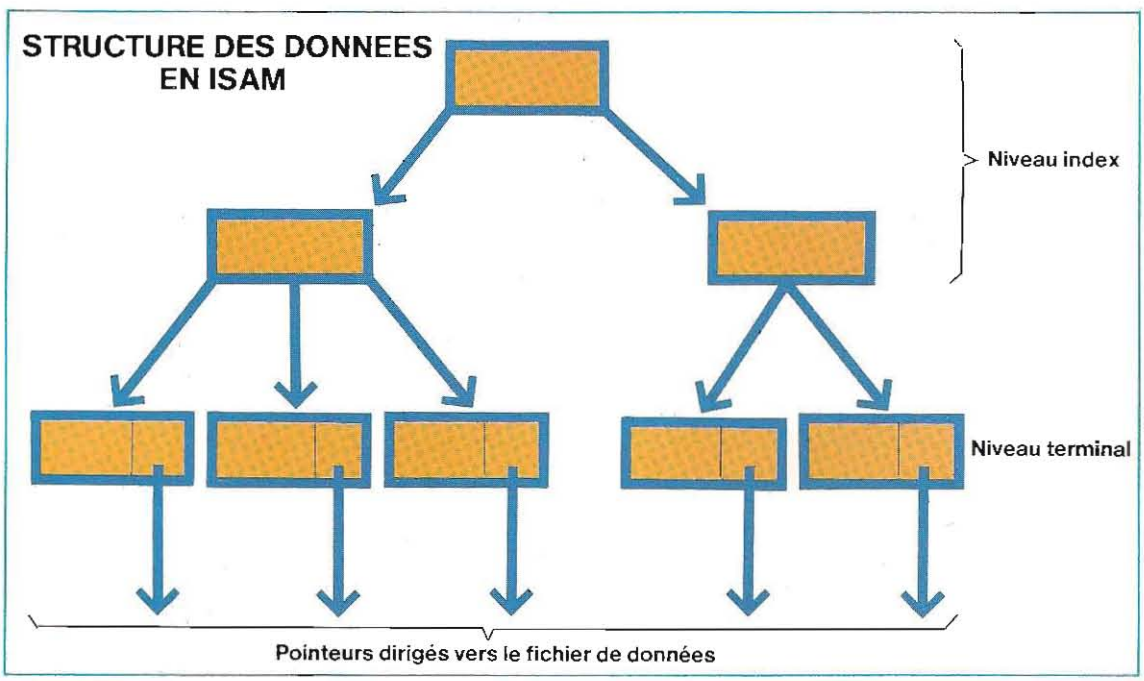
On doit attribuer aux fichiers index utilisés par ISAM des numéros différents de ceux qui sont associés aux fichiers de données. Si on a, par exemple, fait l'initialisation pour un nombre maximum de quatre fichiers en ISAM (soit 1, 2, 3 et 4), seuls restent disponibles pour les autres fichiers les numéros 5 à 15.

Si un programme en Basic a recours aux numéros 1, 2, 3 ou 4, des erreurs imprévisibles risquent de se produire. On peut éviter ces situations « conflictuelles » en prenant l'habitude de numéroter les fichiers par ordre décroissant en partant du plus élevé des nombres des fichiers disponibles.

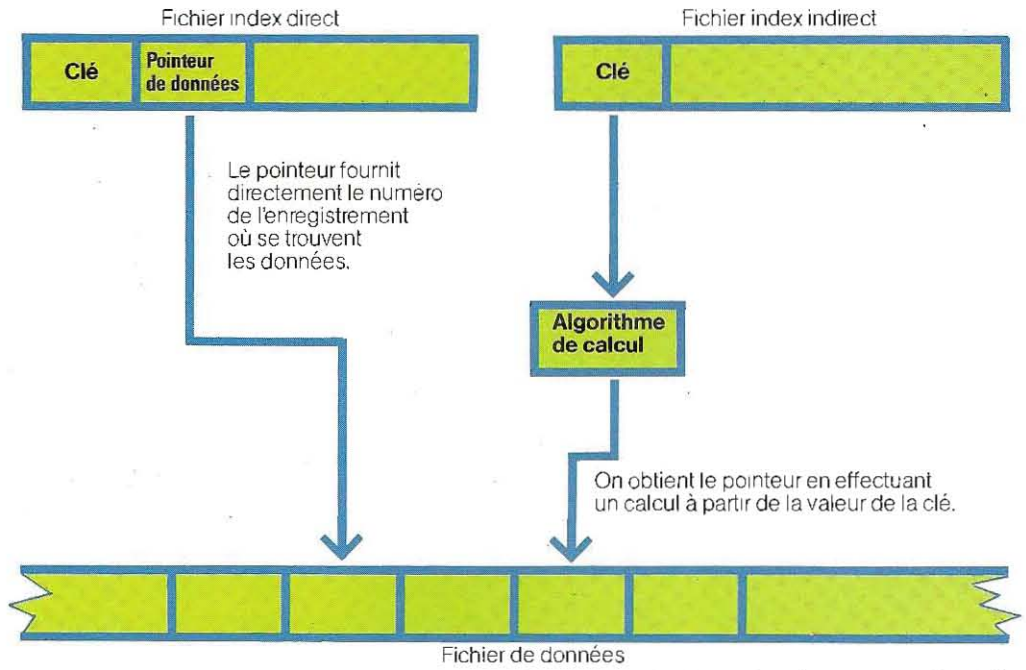
Les enregistrements du fichier index sont lus dans des zones de mémoire intermédiaires, ou zones tampon (buffers, en anglais), situées dans la mémoire centrale.

Le système réserve automatiquement un mi-

* Les programmes de base font partie du système d'exploitation et accomplissent des opérations indispensables au fonctionnement de la machine.



COMPARAISON DE LA GESTION DES DONNEES AVEC DES FICHIERS INDEX DIRECT ET INDIRECT



Dans le second cas, l'index prend moins de place puisqu'il ne contient pas de pointeur mais qu'on obtient ceux-ci en appliquant un algorithme à la valeur de la clé.

nimum de deux zones tampon par fichier afin de permettre l'exécution de l'algorithme d'équilibrage de la structure binaire.
L'utilisateur peut toutefois décider d'augmenter le nombre de ces zones tampon, dans les seules limites de l'espace mémoire disponible. Plus le nombre des fichiers augmente, plus l'espace nécessaire à leur gestion (et notamment, aux zones de contrôle des données) est grand, et moins il reste de place pour les zones tampon. Il est donc conseillé de réduire le nombre des fichiers ouverts en même temps, afin de libérer de la place en mémoire pour ces zones.
ISAM peut fonctionner avec une seule zone tampon par fichier, mais la rapidité de la méthode s'en trouve diminuée. Si ISAM peut faire tout cela, c'est qu'elle dispose d'algorithmes qui attribuent les zones tampon libérées par les derniers fichiers utilisés, tirant ainsi tout le parti possible de l'espace mémoire. Lorsqu'un fichier index a besoin d'une zone tampon supplémentaire et qu'elles sont toutes utilisées, celle qui a servi en dernier lui est affectée.

L'adressage indirect

Le plus simple, devant des fichiers indexés, est d'enregistrer, pour chaque clé, le numéro de l'enregistrement correspondant. Mais on peut également calculer l'adresse d'un enregistrement à partir de la valeur de sa clé. Cette méthode, plus compliquée que la méthode directe, permet de gagner de la place puisqu'on n'a plus besoin d'enregistrer les numéros des enregistrements à côté des clés des données.
Le schéma ci-dessus illustre les deux méthodes : dans la première, les pointeurs des données se trouvent dans l'index; dans la seconde (adressage indirect), on obtient le pointeur en effectuant le calcul adéquat sur la clé.
Les méthodes de calcul de l'adresse à partir de la clé sont parfois très complexes. Nous n'aborderons que les principales qui concernent uniquement les clés numériques.

Subdivision. On sépare les chiffres de la clé en un certain nombre de groupes que l'on

additionne. Le résultat constitue le pointeur. Soit une clé ayant pour valeur 537146. En la divisant en trois groupes de deux chiffres, on obtient 53, 71 et 46, dont la somme est égale à 170. Ce nombre est la valeur du pointeur. Lors de la saisie, les données qui ont pour clé 537146 seront écrites dans l'enregistrement 170. Lors de la recherche, il suffira d'additionner les chiffres de la clé pour retrouver les données.
Selon les applications, la clé est divisée en un plus ou moins grand nombre de groupes, qui contiennent eux-mêmes un nombre variable de chiffres.

Multiplication. Cette méthode est semblable à la précédente, mais on multiplie les chiffres au lieu de les additionner. Reprenons la clé précédente, 537146, et divisons-la en deux parties : 537 et 146. Leur produit est égal à 78402. On peut l'utiliser tel quel comme pointeur, ou n'en retenir qu'une partie, par exemple les trois chiffres du milieu. Le numéro de l'enregistrement sera respectivement 78402 et 840.

Multiplication par 11. On multiplie la clé par 11 et on prend le résultat, ou une partie du résultat, comme pointeur.

Puissance deux. Dans ce cas, le pointeur est une partie du carré de la clé. Si la clé est 271, son carré est 73441. En prenant les trois chiffres centraux, on aura 344 comme pointeur.
Toutes ces méthodes présentent les mêmes inconvénients : elles peuvent produire des « synonymes » et les données ne sont pas toujours bien réparties.
En effet, quelle que soit la méthode employée, il peut arriver que deux ou plusieurs clés différentes aboutissent à la même adresse. Si l'on prend, par exemple, les clés 7421 et 5243, on obtiendra, par la première méthode, 7421 = 74 + 21 = 95 et 5243 = 52 + 43 = 95.
Les deux données devraient donc normalement avoir le même pointeur (95). On surmonte cette difficulté en réservant une zone de l'index aux synonymes, c'est-à-dire à ces clés auxquelles l'algorithme attribue le même pointeur. Il est évident que ces synonymes doivent faire l'objet d'un traitement particulier. Il est également ennuyeux que l'algorithme

La gestion des banques de données exige d'imposantes mémoires de masse.



Maika

produise des valeurs trop proches car une partie du fichier de données est alors presque vide tandis qu'une autre partie est saturée. La seule solution est de changer d'algorithme.

Exemple d'application : la gestion du budget familial

Nous allons illustrer les techniques de manipulation des fichiers exposées jusqu'ici, en développant un programme de création et de maintenance pour le fichier d'un budget familial. La première étape, ou analyse, consiste à déterminer les résultats dont nous aurons besoin en sortie, et les entrées qui nous permettrons de les obtenir.

Le plan des comptes

Puisqu'il s'agit d'un budget, il faut d'abord opérer une première distinction entre deux types de mouvements de fonds, les recettes et les dépenses.
La **justification** de l'opération définit le type de mouvement concerné. Le **plan des comptes** est le tableau qui dénombre les différentes justifications.
Voici donc la liste de ces possibilités :

Type de compte
1 / Dépenses
2 / Recettes

Postes du compte de dépenses (10)	**Postes du compte de recettes (20)**
11 / Loyer	**21** / Salaire principal
12 / Gaz et électricité	**22** / Second salaire
13 / Téléphone	**23** / Rentrées imprévues
14 / Eau	
15 / Garde-robe	**24** / Rapports
16 / Loisirs	**25** / Intérêts
17 / Revues	
18 / Nourriture	

La numérotation constitue le code des différents postes du budget. Les numéros qui commencent par un **1** se rapportent au compte de dépenses tandis que ceux qui commencent par un **2** appartiennent au compte de recettes. L'ordinateur peut, à l'aide de ce code, déterminer s'il s'agit d'une somme à ajouter ou à déduire.
Lors de la saisie des données, il faudra accompagner le montant enregistré d'informations complémentaires :

Compteur progressif	3 caractères
Date	6 caractères
Justification	2 caractères
Montant	8 caractères
Commentaire	20 caractères
Opérateur	10 caractères
Espace disponible pour des développements ultérieurs	11 caractères
	60 caractères au total

Ces zones ont chacune une signification particulière (voir page 262 le schéma du format de cet enregistrement).

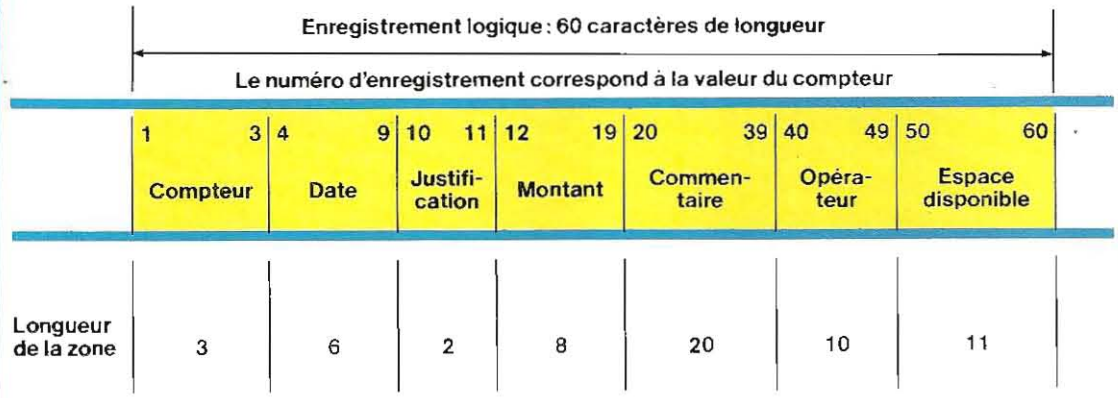
Le compteur. Il permet de retrouver l'ordre dans lequel les données ont été introduites. Il démarre à 1 pour la première opération et est ensuite incrémenté de 1 à chaque nouvelle opération. C'est, en fait, le numéro de l'enregistrement qui contient les données, et la clé qui permet de les relire. Afin d'éviter les erreurs, il doit être calculé automatiquement par la machine. Le programme doit donc, à chaque introduction nouvelle de donnée, ajouter 1 au numéro d'enregistrement et stocker ce numéro dans la zone réservée.

La date. C'est la date de l'opération exprimée sous la forme : jour (2 chiffres), mois (2 chiffres), année (2 chiffres).
Outre sa fonction d'aide-mémoire, cette zone joue le rôle d'indicatif de sélection pour l'impression des bilans. On peut donc demander des bilans des opérations d'une journée à partir de la date.

La justification. C'est la valeur numérique qui correspond à chacun des postes du budget.
Le plan qui nous sert d'exemple ne comporte que des codes compris entre 11 et 25 ; 2 caractères sont donc suffisants. Il faudra vérifier que la valeur indiquée pour cette zone à la saisie est bien comprise dans les limites prévues (de 11 à 18 et de 21 à 25 inclus).

Le montant. C'est le montant d'un mouvement de fonds. On a prévu 8 caractères, c'est-à-dire un maximum de 999 999,99. Cette limite ne joue que pour le montant d'une opération unique ; elle est beaucoup plus élevée pour le calcul des totaux.

STRUCTURE DES DONNEES DANS LE FICHIER DE GESTION DU BUDGET FAMILIAL



Le commentaire. Cette zone de 20 caractères ne sert qu'à commenter une opération ; à expliquer, par exemple, l'emploi d'une somme dans l'une des rubriques du budget.

L'opérateur. Cette zone peut contenir le nom du membre de la famille qui a effectué la dépense ou celui de la personne qui est responsable de l'encaissement. On peut s'en servir pour connaître le total des dépenses d'un membre de la famille ou le total des revenus d'une même source. Si l'on a plusieurs sources de revenus, on pourra, après en avoir précisé l'origine dans cette zone, savoir combien a rapporté, en tout, chacune d'elles.

L'espace disponible. La dernière zone est réservée aux futurs développements du fichier. Il est toujours bon, à la création d'un fichier, de prévoir de la place supplémentaire afin de pouvoir ajouter des données et de ne pas se retrouver avec un fichier inutilisable au cas où surgirait de nouveaux besoins.
Voici les caractéristiques de ces zones (le mot « caractère » est désigné par « ch ») :

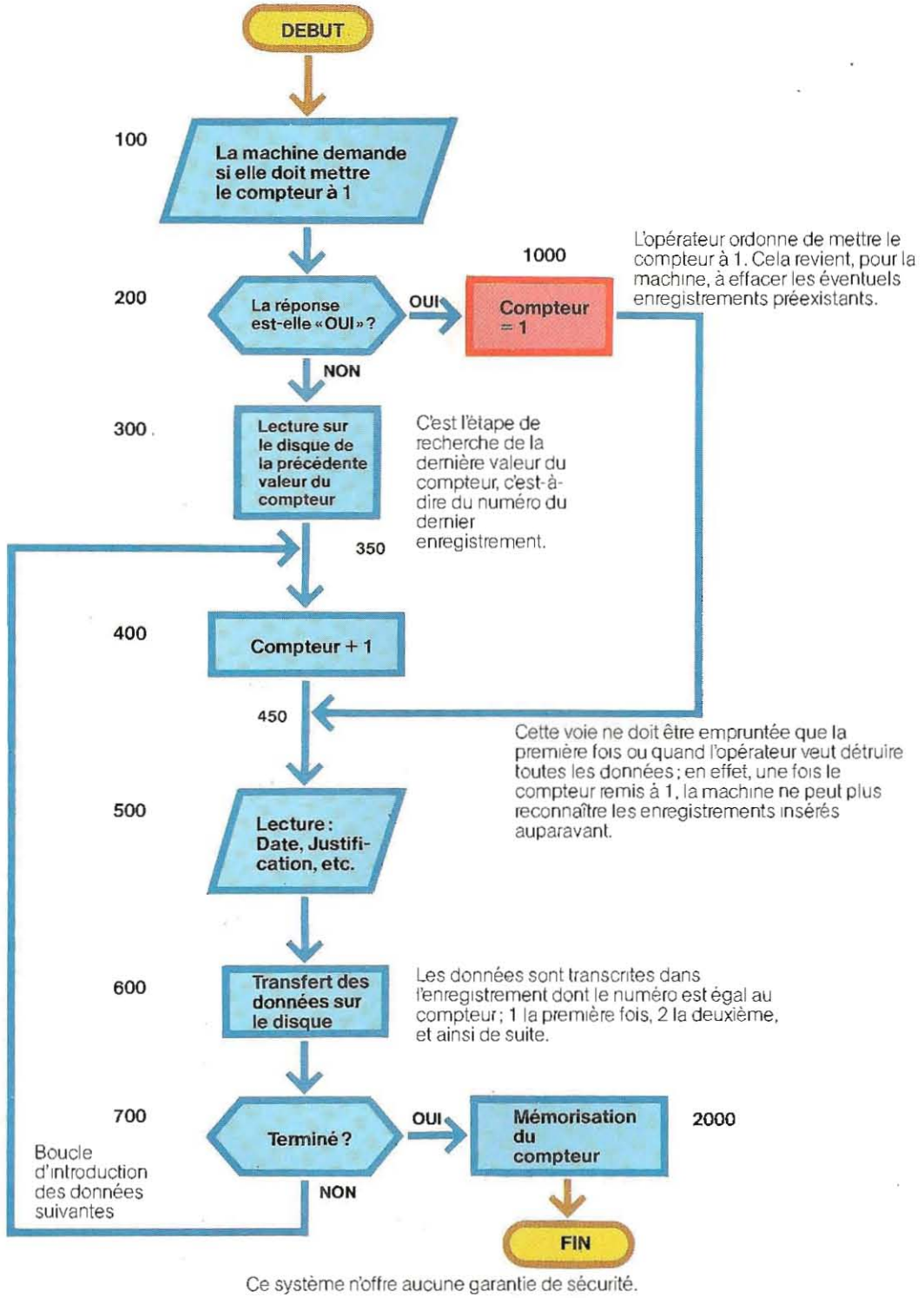
Compteur - 3 ch : augmentation automatique de 1 à chaque entrée.
Date - 6 ch : contrôle de compatibilité entre le jour et le mois (par exemple, il n'y a jamais de 31/02).
Justification - 2 ch : elle va de 11 à 18 inclus et de 21 à 25 inclus.
Montant - 8 ch : cette zone ne peut contenir que des chiffres.
Commentaire - 20 ch : pas de contrôle.
Opérateur - 10 ch : pas de contrôle.

On voit clairement la nécessité de prévoir, dans le programme, deux modules particuliers, l'un pour le calcul automatique du compteur, l'autre pour les contrôles de compatibilité et de numéricité sur la date, la justification et le montant.

Le module de calcul du compteur d'enregistrements

A la première introduction de données, et à ce moment-là seulement, il faut mettre le compteur à 1 ; toutes les saisies qui auront lieu ensuite provoqueront alors son incrémentation de 1.
Le module enregistre la dernière valeur du compteur après chaque entrée et la rappelle lorsque l'utilisateur reprend le travail. La mémorisation du compteur doit obligatoirement se faire sur disque (ou sur bande). En effet, si elle ne se faisait qu'en mémoire centrale, la donnée serait perdue à la première coupure d'alimentation (les mémoires électroniques sont « volatiles », ou non permanentes, c'est-à-dire qu'elles perdent leur contenu dès que le courant est coupé).
L'organigramme de ces opérations est représenté par le schéma de la page 263.
Ce déroulement logique recèle deux graves défauts qui compromettent le fonctionnement du programme.

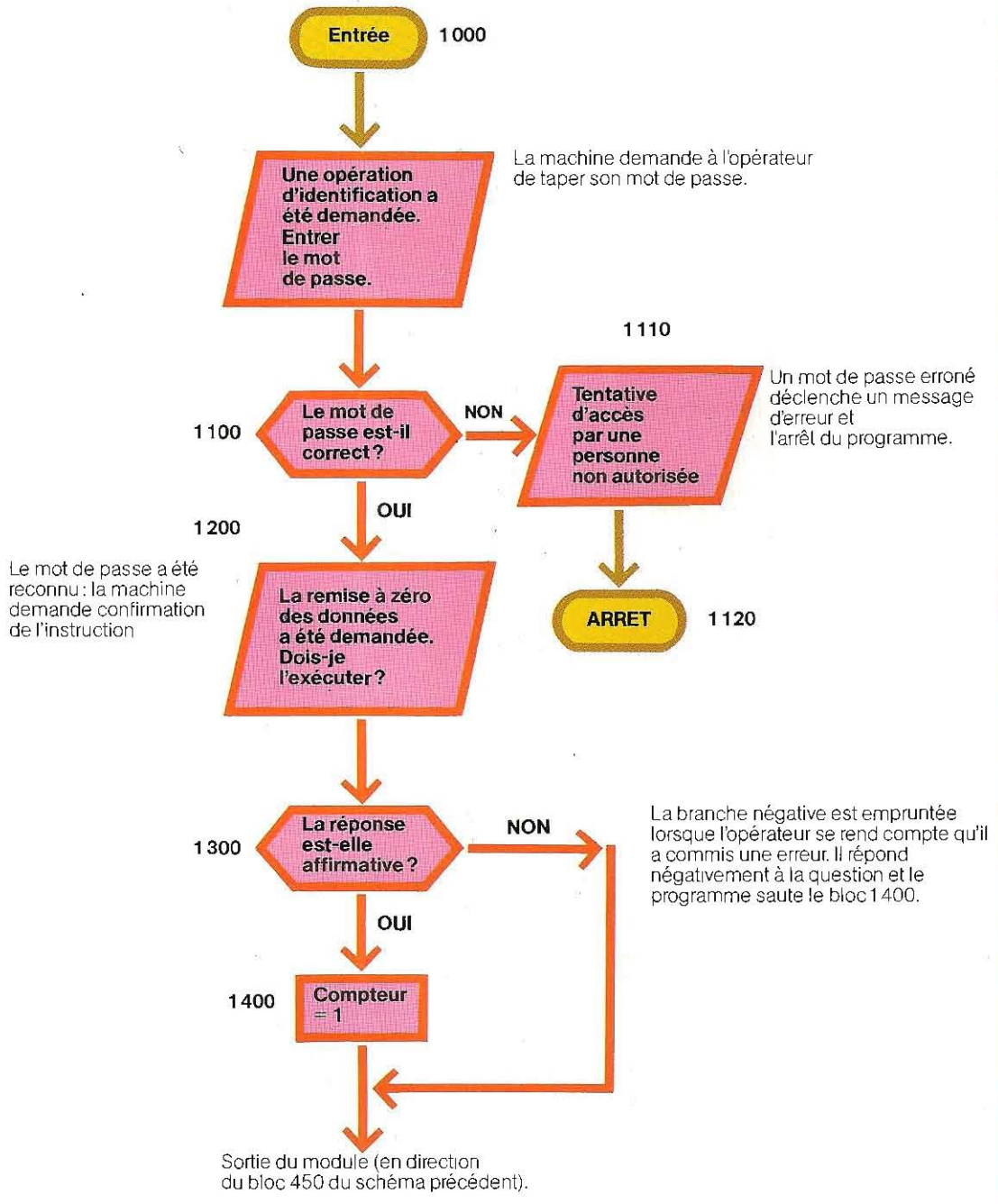
ORGANIGRAMME DE L'INTRODUCTION DES DONNEES

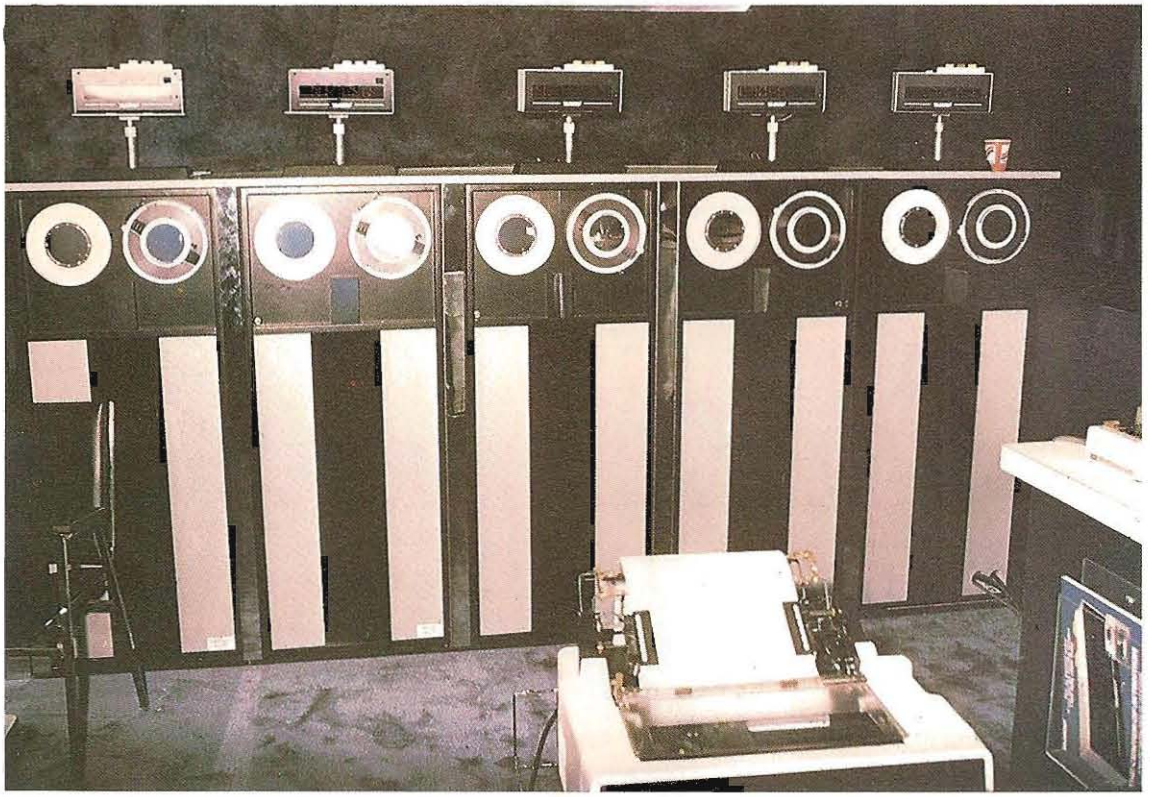


Ce système n'offre aucune garantie de sécurité.

En effet, si le bloc 1 000 (mise du compteur à 1) est utilisé à mauvais escient, tous les enregistrements précédents seront perdus. Supposons que nous en soyons à l'introduction de la donnée numéro 176 mémorisée dans le compteur. La prochaine donnée sera donc enregistrée à l'emplacement 176 + 1, c'est-à-dire dans une zone disponible.

MODULE 1000 DEMANDANT L'ENTREE D'UN MOT DE PASSE ET UNE CONFIRMATION POUR TOUTE OPERATION QUI POURRAIT ENTRAINER LA PERTE DE DONNEES





Marka

Vue partielle (dérouleurs de bande) de la mémoire de masse d'une banque de données.

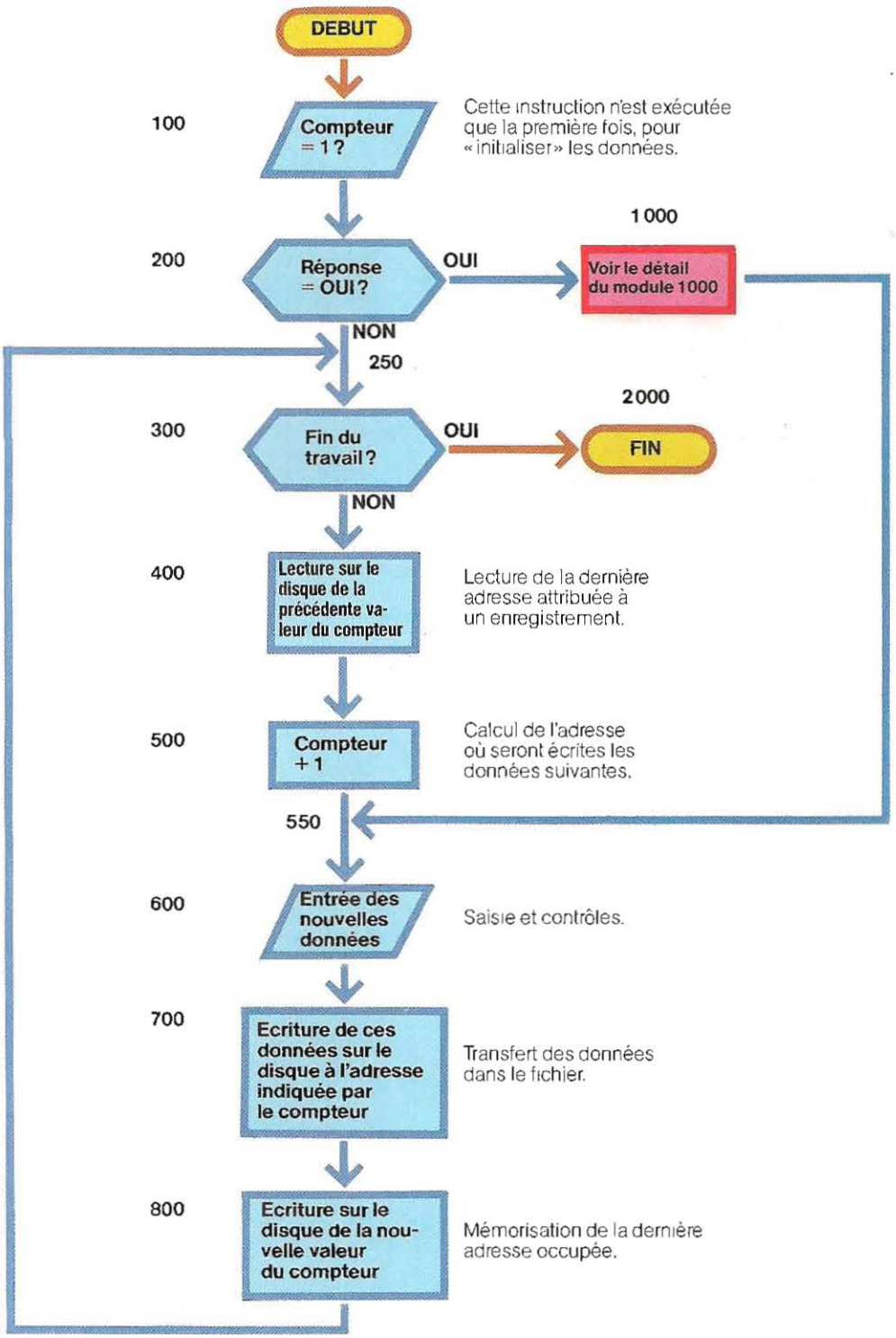
Inversement si, après la donnée 176, nous déclenchons par erreur l'exécution du module 1 000, la donnée suivante sera écrite en position 1 (puisque le compteur aura été remis à 1 par le module 1 000) et non, comme elle aurait dû l'être, en position 177. En s'écrivant par-dessus le précédent contenu de la position 1, elle l'« écrasera », c'est-à-dire l'effacera. En outre, la machine ne connaîtra plus le nombre des données réellement présentes.

La seule façon de réduire ces risques d'erreur est de subordonner l'exécution du module 1 000 à la reconnaissance d'un mot de passe et à une demande de confirmation. On a recours à cette double vérification toutes les fois qu'une distraction ou une mauvaise utilisation risqueraient d'entraîner la perte des données.

Dans notre exemple, la perte des données n'est qu'apparente ou, du moins, limitée, puisqu'il suffit de remettre le compteur à 176 pour les récupérer. Dans d'autres cas, elle est irrémédiable et lourde de conséquences. Qu'on essaie d'imaginer ce que signifie la destruction des fichiers contenant les informations économiques d'une grande entreprise. L'organigramme de la page 264 représente le module 1 000 amélioré : les contrôles appropriés ont été ajoutés pour protéger les données contre les erreurs humaines.

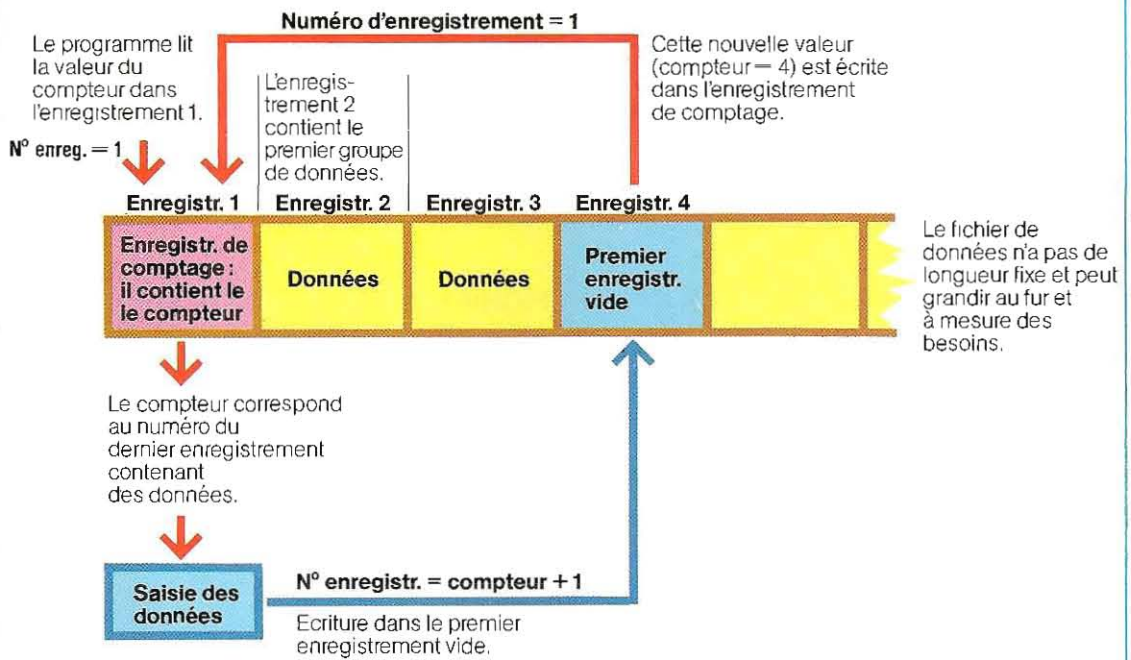
Venons-en maintenant au second défaut du programme. Imaginons que nous en sommes à l'enregistrement 121 au cours d'une saisie de données. Cette valeur n'a pas été mémorisée sur le disque puisque, comme le montre le schéma de la page 263, ce transfert n'a lieu qu'en fin de travail. C'est alors que se produit une panne de courant. Puis le courant est rétabli. Mais le compteur n'est plus à 121 (il est à zéro, à « blanc » ; ou, pire, il contient un nombre aléatoire) et toutes les données saisies avant la panne ne peuvent plus, malgré leur présence dans le fichier, être considérées comme acquises puisque le système en a perdu l'adresse (la position). Contre ce risque, il n'y a qu'une solution : mémoriser le compteur à chaque introduction de données. L'organigramme définitivement corrigé est représenté page 266.

ORGANIGRAMME DE SAISIE DES DONNEES MODIFIE POUR LA MISE A JOUR DU COMPTEUR A CHAQUE ENTREE

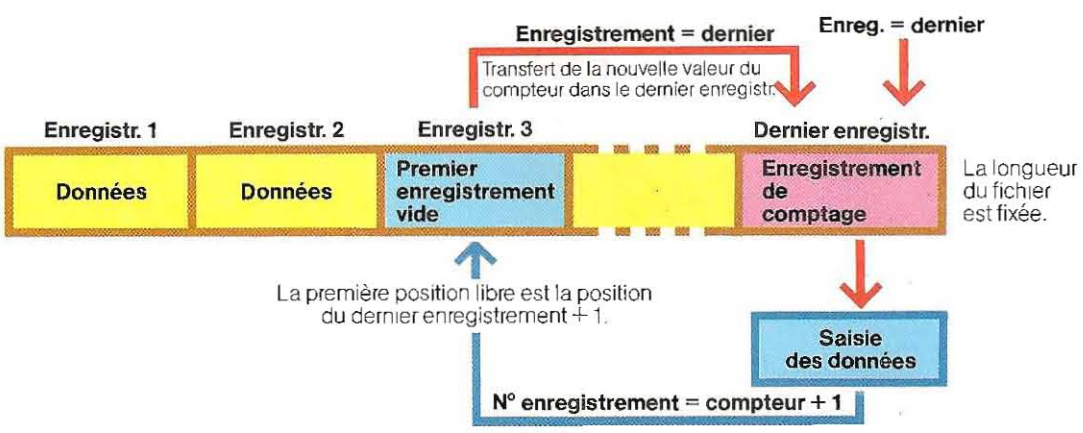


C'est alors que se pose la question de savoir où va s'inscrire le compteur. Il est possible d'utiliser un enregistrement donné du fichier, dont toutes les zones seront vides, excepté celle du compteur. Bien qu'appartenant au fichier de données, cet enregistrement n'est qu'une zone de comptage temporaire dans lequel on peut lire ou écrire le compteur. S'il est placé au début du fichier, le premier enregistrement disponible pour l'écriture des données est alors le numéro 2 et la donnée 1 n'existe pas. Cette solution permet de ne pas spécifier d'avance la longueur du fichier qui pourra s'accroître éventuellement au fur et à mesure des besoins.

STRUCTURE DES FICHIERS DE DONNEES CONTENANT UN ENREGISTREMENT DE COMPTAGE



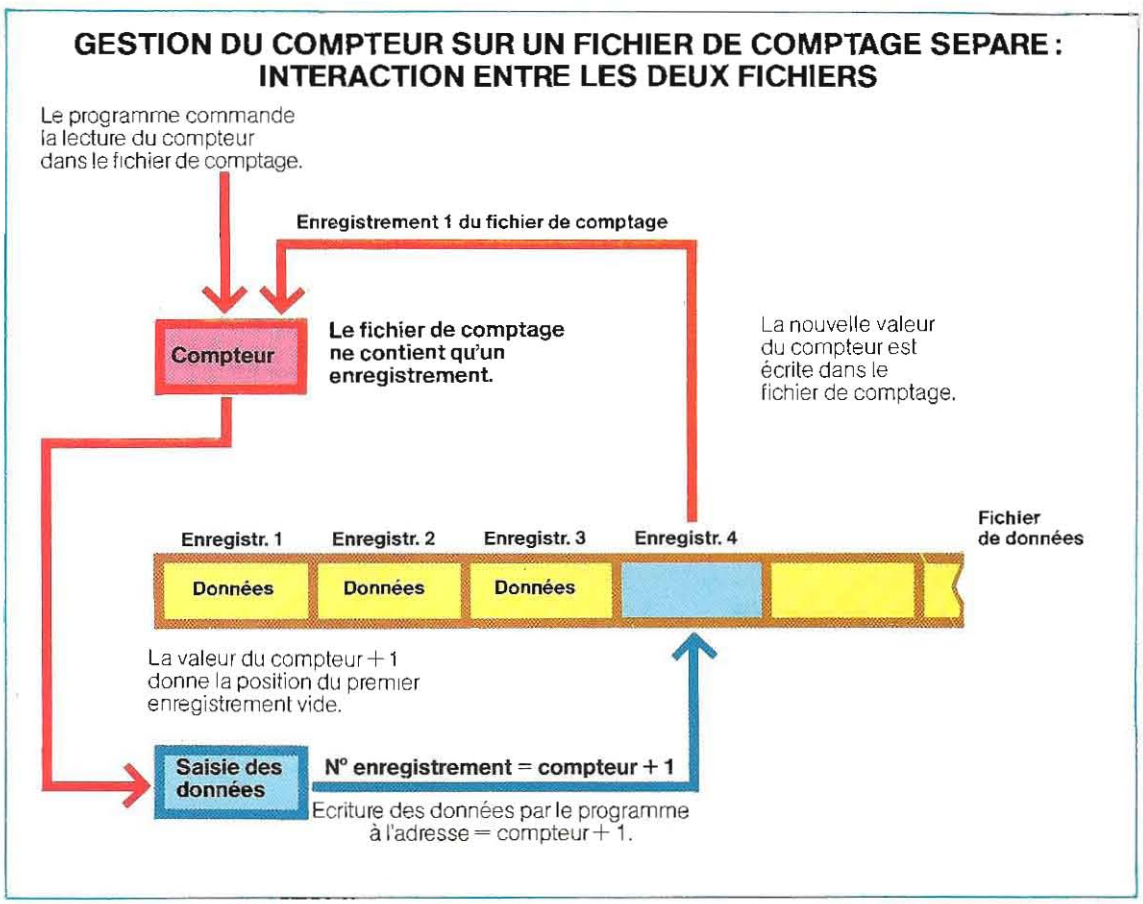
L'enregistrement de comptage se trouve en position 1. Les données commencent au numéro 2 et la longueur du fichier n'est pas prédéterminée.



L'enregistrement de comptage se trouve en fin de fichier. Les données commencent au numéro 1 mais la longueur du fichier est limitée.

En revanche, si l'enregistrement de comptage se situe à la fin du fichier de données, la numérotation commence au numéro 1 puisque le premier enregistrement n'est plus destiné au comptage. Il faut déterminer à l'avance la longueur du fichier, l'adresse de l'enregistrement de comptage étant celle du dernier enregistrement du fichier. Les schémas de la page 267 représentent ces deux possibilités et les voies d'accès aux enregistrements correspondants.
Une seconde solution consiste à créer un nouveau fichier ne contenant qu'un enregistrement : on y mémorise le compteur. Les données sont alors numérotées à partir de 1, et l'on conserve les avantages d'un fichier de longueur indéterminée. Mais on doit alors gérer simultanément deux fichiers : celui des données et celui du comptage.
Le schéma ci-dessous illustre cette structure

GESTION DU COMPTEUR SUR UN FICHIER DE COMPTAGE SEPARE : INTERACTION ENTRE LES DEUX FICHIERS



et son utilisation. Quelle que soit la méthode choisie, lecture et écriture du compteur sont effectuées par les modules 400 et 800 de

l'organigramme général (page 266). En revanche, le contenu des modules varie en fonction de la méthode adoptée. On peut voir ci-dessous le module de lecture tel qu'il se présente quand on a recours à un fichier de comptage appelé DIR ; le module d'écriture (800) se décomposerait de façon analogue.

LECTURE DE LA VALEUR DU COMPTEUR DANS LE FICHIER DIR

Module : 400.

Fonction : lecture de la zone compteur dans le fichier DIR.

Nom du fichier : DIR.

Longueur : 1 enregistrement (60 caractères).

Position de la zone compteur : 1 – 3

Sortie : valeur numérique du compteur.



L'écriture (module 800) se déroule de façon identique.

Le module de contrôle des données en entrée

Lors de l'introduction (saisie) des données, il faut contrôler l'exactitude formelle des zones Date, Justification et Montant.

Cette opération (détaillée page 270) doit s'effectuer dans le module d'introduction des données (600 sur le schéma de la page 266), afin que seules des données exactes puissent être acheminées vers la suite du programme. Le module (ou « bloc ») 620 concernant le contrôle de la date, fera l'objet d'un développement ultérieur car il revient souvent dans les programmes. Considérons, pour l'instant, qu'il permet de vérifier la compatibilité entre le jour et le mois, et de répondre par une variable d'erreur. Si celle-ci est nulle, la date est exacte ; sinon elle est erronée et la

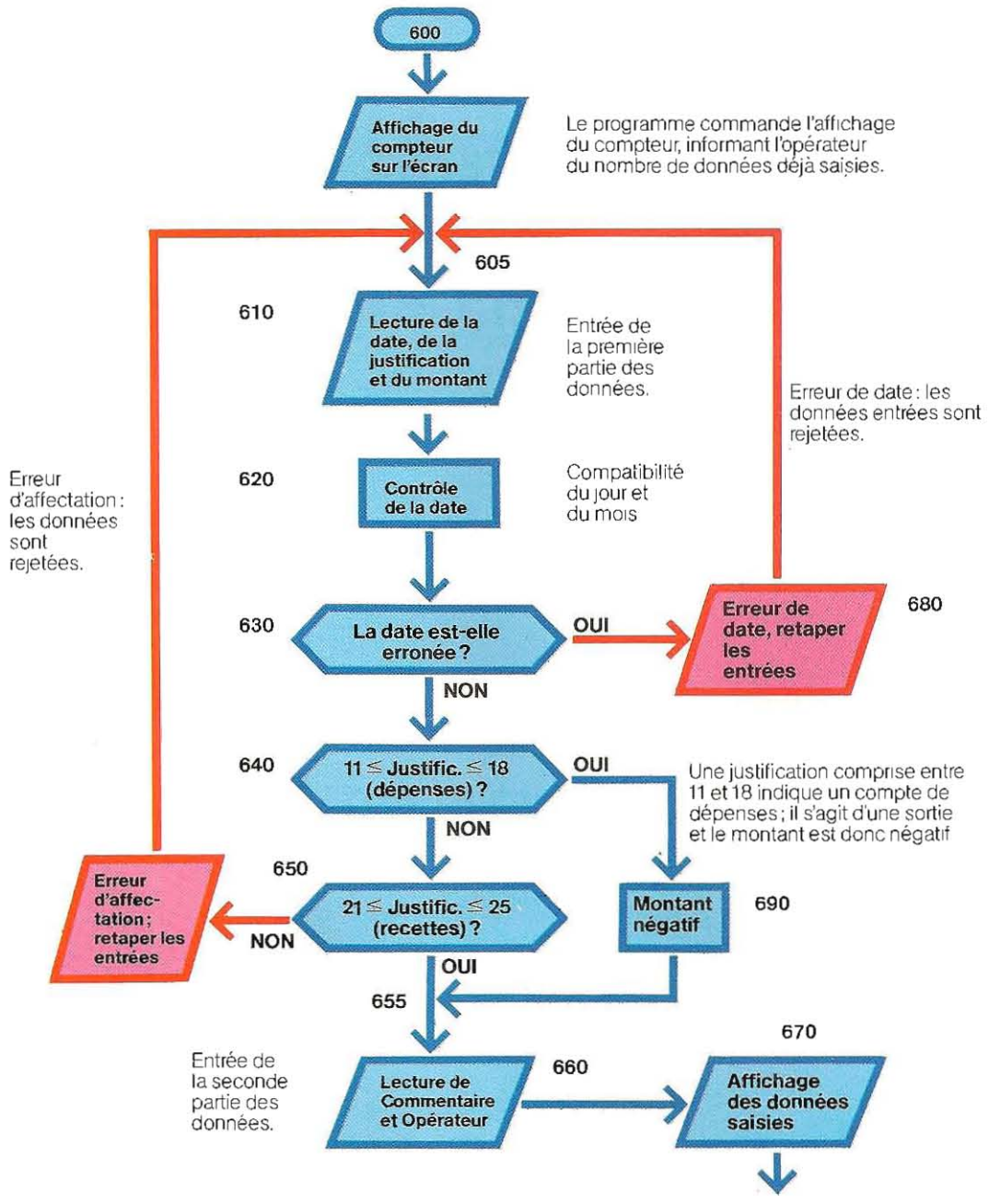
ORGANIGRAMME DETAILLE DU MODULE DE SAISIE ET DE CONTROLE DES DONNEES

Module : 600.

Fonction : entrée au clavier des nouvelles données et contrôle formel.

Entrée : compteur

Sortie : zones de données prêtes à être transférées sur disque.



valeur de la variable renseigne alors sur le type d'erreur commis. Ce module peut également servir à limiter dans le temps l'utilisation des programmes. Si, par exemple, nous posons comme condition que l'année ne peut être postérieure à 1995, le programme n'acceptera plus les saisies en 1996. Dans le cas où un utilisateur ne dispose que de la version compilée du programme (c'est-à-dire en langage machine), il ne peut modifier lui-même cette date limite. Seule peut le faire la personne qui détient le programme source. Naturellement, après les corrections, on devra recompiler le programme.
La seule ressource de l'utilisateur serait de fournir une fausse date à l'ordinateur, en continuant, par exemple, de taper 1995 au lieu de 1996. Naturellement, cette solution n'est envisageable que pour les programmes à usage personnel et certainement pas sur des programmes d'utilisation officielle (fiscale par exemple).
Cette technique de péremption permet de louer des programmes dont la date d'échéance ne sera remise à jour que si l'utilisateur a payé sa redevance.

En résumé :

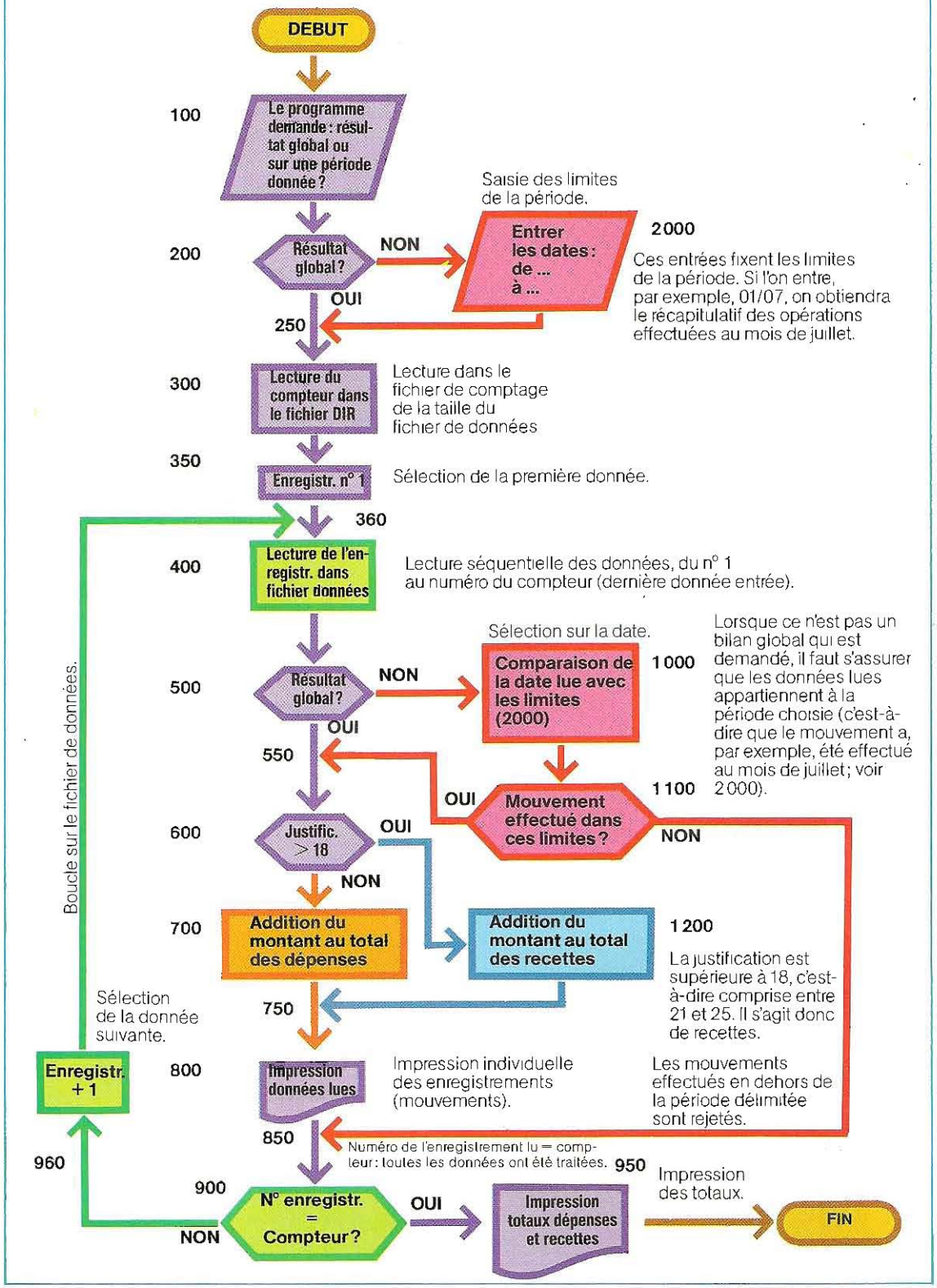
- **la structure de l'enregistrement de données et les longueurs des zones sont schématisées page 262 ;**
- **l'organigramme du programme de saisie et de contrôle des données complet se trouve page 266 ;**
- **l'initialisation est effectuée par le module 1000 dont l'organigramme se trouve page 264 ;**
- **le compteur de données est mémorisé dans un fichier auxiliaire de comptage (voir schémas pages 268 et 269) ;**
- **les opérations de saisie et de contrôle des données sont détaillées page 270. En traduisant dans l'ordre les organigrammes des pages 264, 266, 269 et 270 en instructions, nous obtiendrons notre programme.**

Le module 600 permet la saisie des données. Il faut, à présent, compléter la procédure par le développement des calculs.
Ce nouveau traitement doit lire tout le fichier de données et additionner séparément les montants des dépenses et des recettes. Nous aurons ainsi une vue générale des mouvements effectués et le résultat global (la différence entre le total des recettes et le total des dépenses). En plus de ce résultat, il faut prévoir la possibilité d'effectuer des bilans partiels entre deux dates déterminées. Nous pourrons ainsi demander soit une situation globale (prenant en compte les données du début à la fin), soit la situation d'une période déterminée.
L'organigramme de ce traitement est représenté page 272.
Les programmes « auxiliaires » concernant des données particulières (justification, commentaire, opérateur) feront l'objet d'une explication générale ultérieure.
Quant aux modules qui apparaissent page 272, ils seront développés en détail lorsque nous rédigerons le programme après avoir étudié les instructions Basic nécessaires.

La recherche des données dans les fichiers

Une fois le fichier constitué (phase de création), il faut pouvoir modifier ses données (phase de mise à jour).
Pour mettre à jour un enregistrement, on doit d'abord le lire, puis, en fonction de son contenu, effectuer les ajouts et les modifications qui s'imposent.
Accéder aux données selon une logique déterminée constitue la phase principale d'utilisation d'un fichier.
Pour la mise à jour, l'utilisateur recherche l'enregistrement possédant la clé voulue et en extrait les données correspondantes pour les modifier.
Pour un traitement particulier (listing par exemple), il s'agit d'extraire du fichier une partie des données. On peut ainsi rechercher dans un fichier d'adresses celles qui se situent dans une même ville, ou dans un fichier du personnel tous les employés d'une même catégorie.
Il est donc indispensable de disposer de programmes de recherche qui retrouvent tous les enregistrements dont les données répondent

ORGANIGRAMME DU PROGRAMME DE CALCUL ET D'IMPRESSION DU BUDGET



aux critères voulus.
Les deux méthodes de recherche les plus utilisées sont la recherche séquentielle et la recherche dichotomique.

La recherche séquentielle

La recherche séquentielle d'une donnée implique la lecture successive de tous les enregistrements du fichier et leur comparaison avec la valeur de la clé de recherche. S'il y a égalité, la donnée trouvée est imprimée, sinon la recherche se poursuit jusqu'à la fin des enregistrements.
Le nombre moyen de lectures à effectuer s'obtient par la formule suivante :

$$\text{nombre moyen de lectures} = \frac{\text{longueur du fichier} + 1}{2}$$

Dans un fichier de 1 000 enregistrements, il nous faudra donc en lire 500 en moyenne avant de trouver celui qui nous intéresse. De plus, comme nous l'avons vu, la donnée recherchée peut très bien apparaître dans plusieurs enregistrements ; il faudra donc poursuivre la recherche jusqu'à la fin du fichier. Le seul moyen d'accélérer cette opération est d'ordonner préalablement les données.

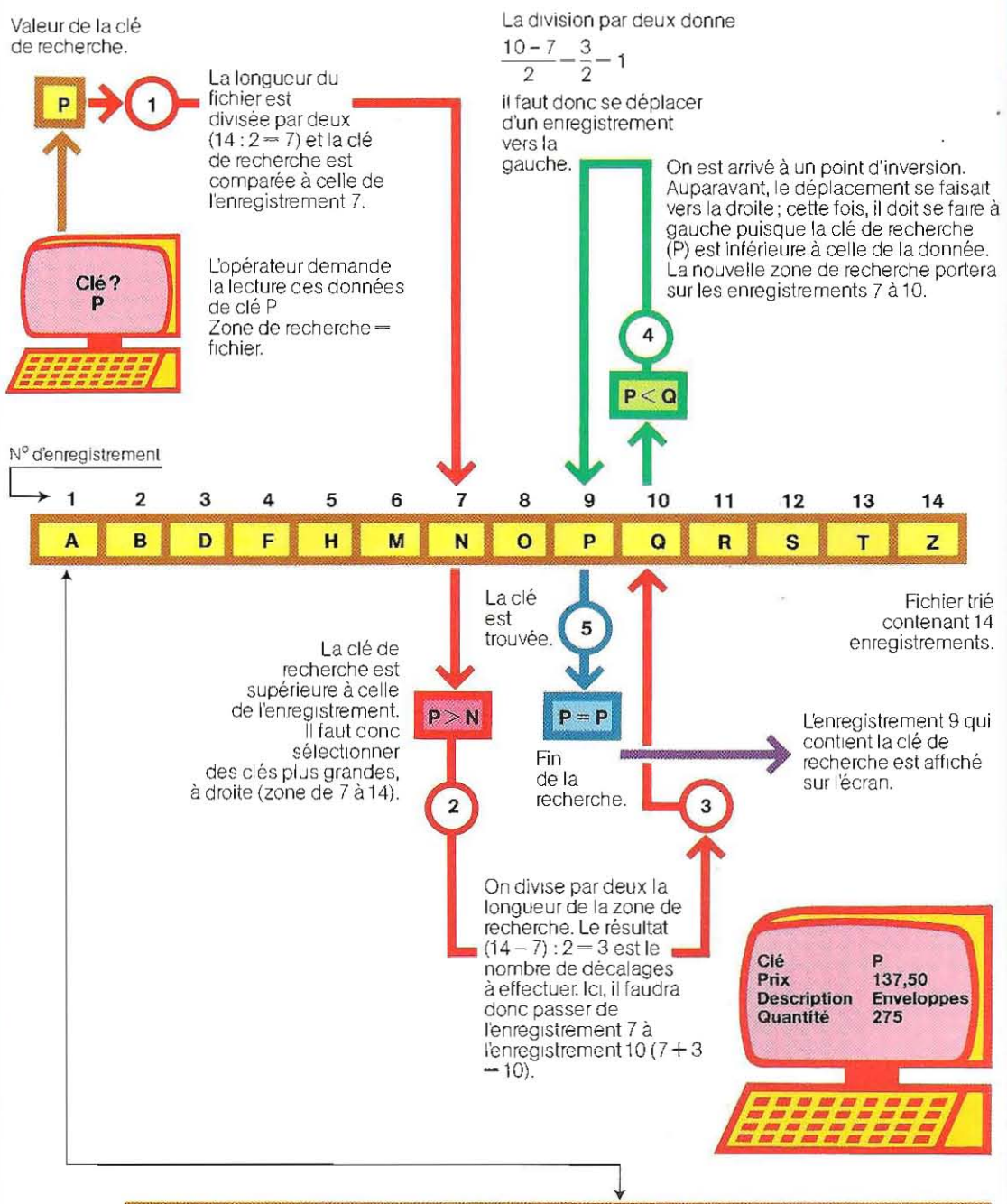
Deux étapes du travail effectué sur les banques de données. L'information doit être parfaitement planifiée et structurée afin d'utiliser au mieux la capacité de la mémoire.





Marka

LOGIQUE D'UNE RECHERCHE DICHOTOMIQUE



Clé	Prix	Description	Quantité
Contenu	A	318	Stylos à bille	106
Nombre de caractères	1	6	20	3

Longueur de l'enregistrement = 1 + 6 + 20 + 3 = 30 caractères.

La recherche séquentielle s'avère très longue. Aussi ne l'applique-t-on qu'à des petits fichiers, pour lesquels le gain de temps permis par des méthodes plus rapides, mais difficilement mises en œuvre, serait négligeable.

La recherche dichotomique

Elle ne peut se pratiquer que sur des fichiers ordonnés. On procède à des divisions binaires (en deux parties) successives du fichier. On lit ainsi l'enregistrement situé au milieu, on compare sa clé à la clé de recherche et on rejette alors l'une des parties (la droite si la clé de recherche est inférieure à celle de l'enregistrement lu, la gauche dans le cas contraire, en supposant le fichier classé par ordre croissant). On répète l'opération jusqu'à la découverte de la donnée recherchée. Prenons un exemple (voir schema page 274).

- Zone clé, 1 caractère alphabétique.
- Prix des articles, 6 caractères numériques*.
- Description de l'article, 20 caractères alphabétiques.
- Quantité en stock, 3 caractères numériques*.

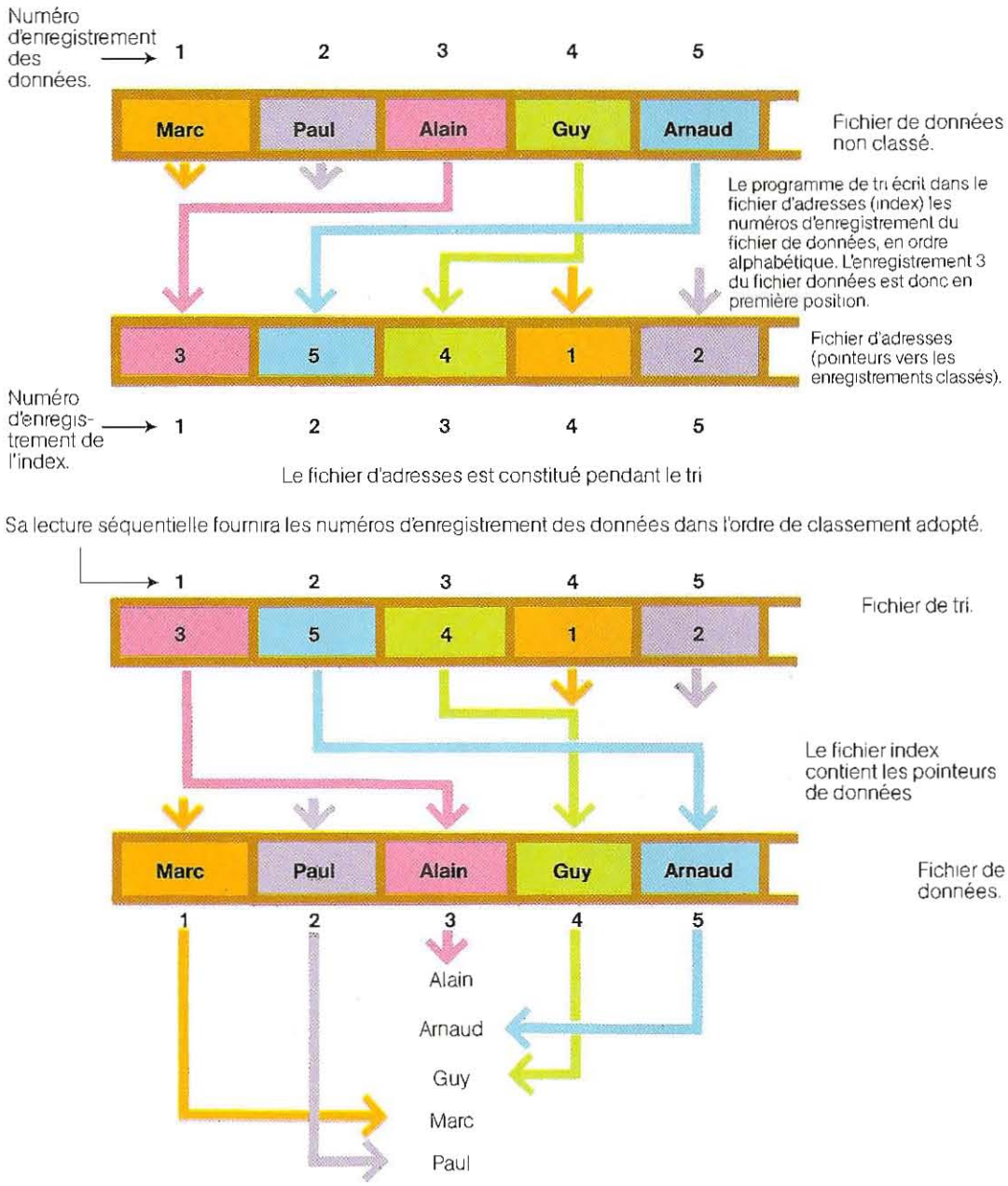
L'utilisateur veut connaître le contenu de l'enregistrement dont la clé est P. La recherche dichotomique s'effectue comme suit.
On divise d'abord (point **1**) la longueur du fichier par 2 ; ici, 14 : 2 = 7.
Ensuite (point **2**), la clé de l'enregistrement 7 (N) est comparée à la clé de recherche (P). Or, P est placé après N (P > N) ; l'enregistrement de clé P se trouve donc à droite, entre les enregistrements 7 et 14 (fin du fichier). Cette zone est à son tour divisée en deux. (14 – 7) : 2 = 3 (on ne prend que la partie entière du quotient). On obtient, en nombre d'enregistrements, l'ampleur du déplacement à effectuer vers la droite, à partir du n° 7.
En se décalant, vers la droite, de trois enregistrements à partir du numéro 7, on va lire l'enregistrement 7 + 3 = 10 (point **3**).
Cet enregistrement ayant une clé supérieure à P (point **4**), celui qui est recherché se trouve, cette fois, à sa gauche. De même que tout à l'heure notre donnée se trouvait entre les enregistrements 7 et 14, on la localise, maintenant entre les enregistrements 7 et 10. Un nouveau découpage en deux de la partie à étudier (10 – 7) : 2 = 1 indiquera le déplacement à effectuer vers la gauche à partir de la position actuelle.
L'enregistrement 10 – 1 = 9 qui est lu contient P dans sa zone clé (point **5**). La recherche s'achève et les données de l'enregistrement 9 sont affichées à l'écran.
Avec cette méthode, on n'a lu que trois des quatorze enregistrements du fichier (7, 10 et 9) alors qu'en recherche séquentielle nous aurions dû effectuer neuf lectures (l'enregistrement de clé P et les huit précédents). L'avantage de la recherche dichotomique est évident.
Une réserve s'impose cependant. On ne peut faire une recherche dichotomique que sur un fichier ordonné. Sinon, il faudra d'abord en classer les données. Cette méthode dichotomique impose un programme plus complexe que la méthode séquentielle et ne fait donc pas forcément gagner de temps lorsque le fichier ne comporte qu'un petit nombre d'enregistrements.

Le classement des données

Le classement des données est l'une des tâches principales de la gestion des fichiers. C'est, en effet, une condition fondamentale pour l'application de certaines techniques de traitement (comme la recherche dichotomique) et pour l'obtention de listings faciles à consulter.
Lorsqu'on ne peut obtenir de listing des données classées, un fichier, même de modestes dimensions, s'avère très difficile à exploiter, car il est pratiquement impossible d'y trouver les éléments que l'on recherche. C'est un peu comme si l'on disposait d'un annuaire téléphonique où les noms ne seraient pas répertoriés par ordre alphabétique mais dans l'ordre chronologique de souscription du contrat d'abonnement ! M. Martin, abonné de longue date, figurerait donc avant M. Leblanc et il serait impossible de retrouver

* Dans les ordinateurs individuels, les nombres sont souvent enregistrés sur les disques comme des caractères. Le nombre 579, par exemple, est enregistré sur trois octets, qui contiennent, dans l'ordre, les chiffres 5, 7 et 9, codés comme des caractères, respectivement par les valeurs hexadécimales 35, 37 et 39 (se reporter au code ASCII). Avec d'autres ordinateurs, on peut, au contraire, enregistrer les nombres tels quels sur les disques, ce qui occupe moins de place. Nous reviendrons plus loin sur cette question.

TRI DES DONNEES A L'AIDE D'UN FICHIER INDEX

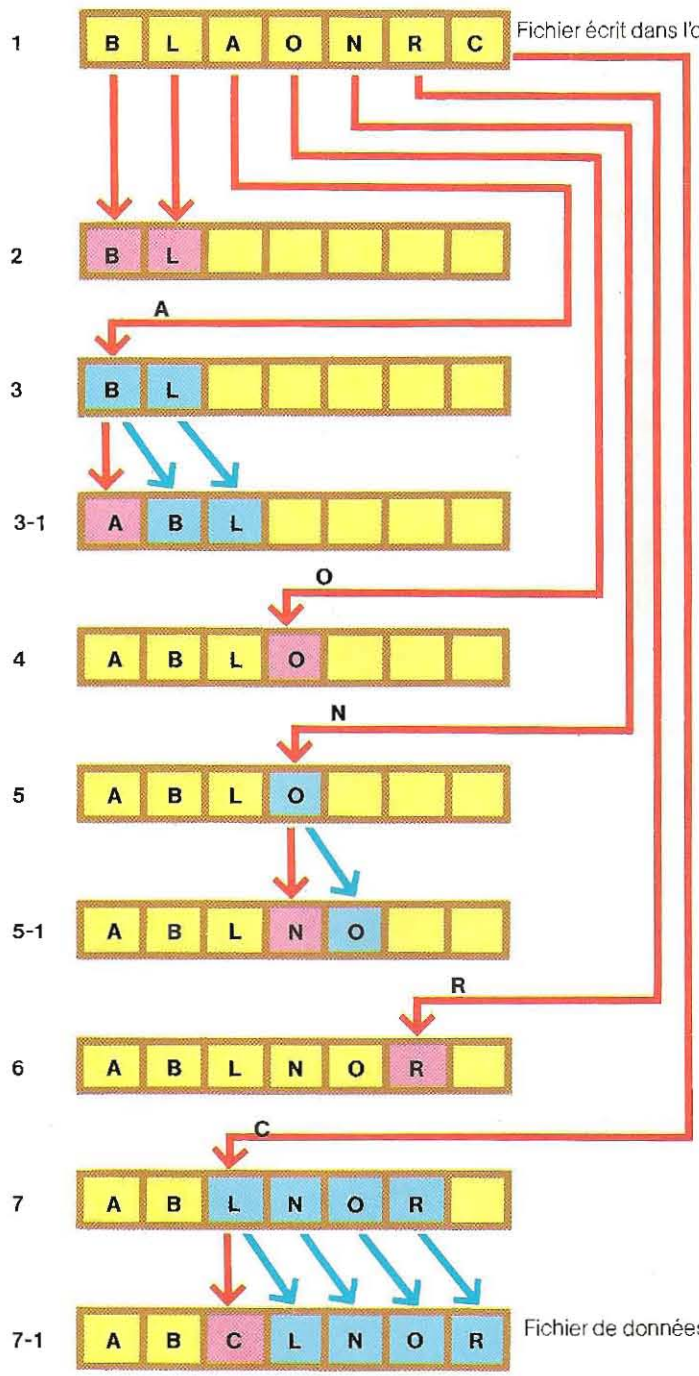


Pour obtenir une liste classée des données du fichier, il faut lire l'index séquentiellement et utiliser son contenu comme pointeur vers le fichier de données.

un numéro de téléphone donné puisqu'aucune logique de recherche ne serait applicable en temps réel. De même, pour l'utilisation des données d'un fichier ; c'est pourquoi toutes les procédures de gestion de fichiers

possèdent des programmes de classement des données, appelés des programmes de tri.
L'importance de ces programmes est telle

TRI DES DONNEES PAR INSERTION DANS UN FICHIER PARALLELE

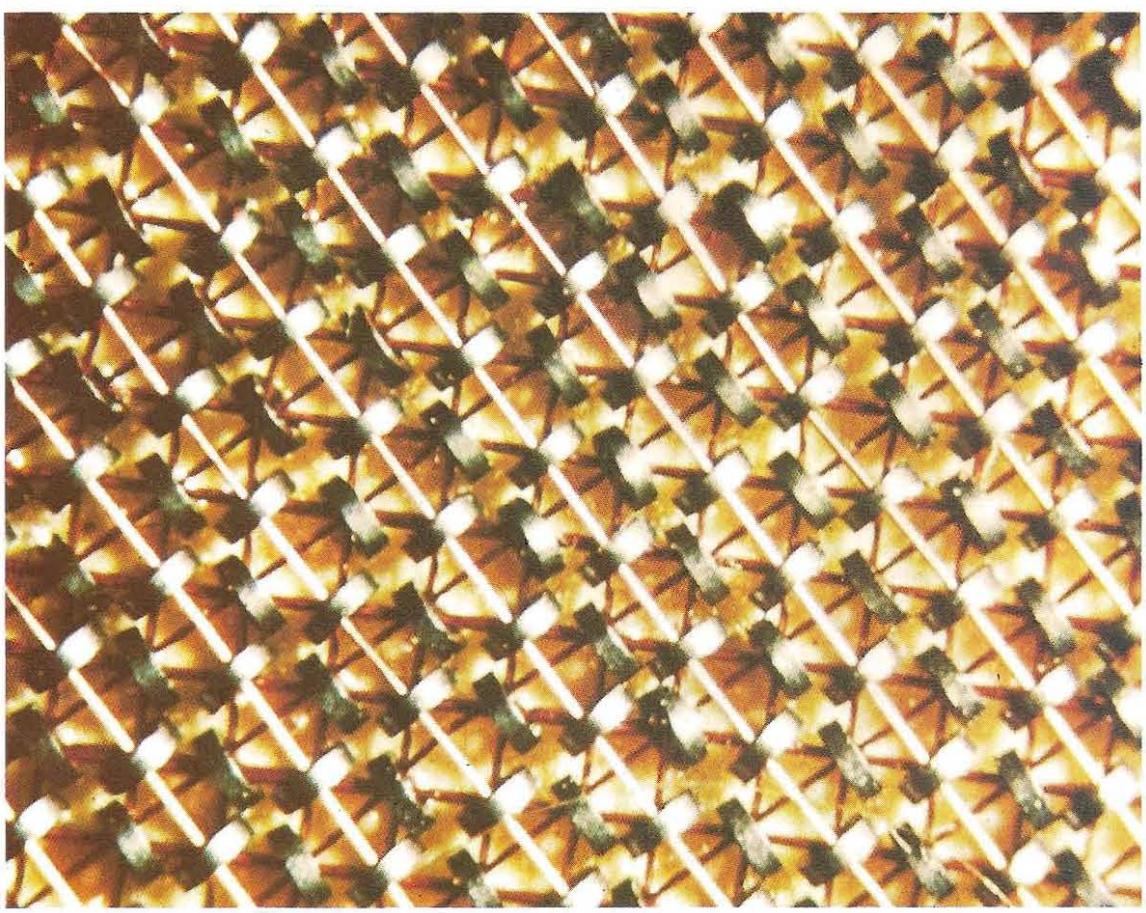


Fichier parallèle dans lequel les données sont transférées de façon ordonnée.

qu'ils sont inclus dans presque tous les systèmes d'exploitation, et généralement écrits en Assembleur pour accroître leur rapidité d'exécution.
A ces programmes de tri « utilitaires », c'est-à-dire accessibles à l'utilisateur, ce dernier ne fournit que le nom du fichier, la structure de l'enregistrement (nombre et longueur des zones de données) et la clé en fonction de laquelle il veut ordonner le fichier.
On distingue deux types de programme de tri suivant la structure des résultats qu'ils obtiennent. Certains recopient le fichier, en l'ordonnant, sur une nouvelle partie du disque, créant ainsi un nouveau fichier ordonné, et sous un nom différent du premier. D'autres constituent un fichier d'adresses qui contient (dans l'ordre) les adresses des données du fichier d'origine.
Pour lire les données dans l'ordre, il faudra donc, dans ce dernier cas, lire d'abord le fichier d'adresses, puis l'enregistrement de données pointé par l'adresse trouvée. Cette méthode est illustrée par le schéma de la page 276.
Cette méthode est plus rapide (puisqu'on n'a pas à recopier tout le fichier mais seulement les adresses) et elle permet d'occuper moins de place sur le disque en évitant de recopier des données déjà existantes (redondance).. Si tous les mini-ordinateurs sont proposés avec des programmes de tri, certains micro-ordinateurs en sont dépourvus. C'est alors le programmeur qui doit se charger de les écrire lui-même. Ce qui implique une méthode de travail rigoureuse, que nous allons étudier en l'illustrant d'un exemple de tri par insertion (algorithme) avec constitution d'un fichier « parallèle » (recopie). On se reportera au schéma de la page 277.

1 / Après avoir lu le premier enregistrement de données (qui contient B), on le transfère dans le premier enregistrement du fichier de sortie, c'est-à-dire du fichier parallèle qui

Détail d'un ancien dispositif de mémoire centrale à tores de ferrite.



Marka

Test 7

1 / Laquelle de ces affirmations s'applique à un fichier séquentiel ?
a) on peut accéder aux enregistrements dans n'importe quel ordre ;
b) les données sont écrites dans l'ordre croissant de la zone clé ;
c) les enregistrements sont écrits et lus l'un à la suite de l'autre.

2 / Lesquelles de ces affirmations s'appliquent à un fichier indexé ?
a) l'index est une zone spéciale qui contient des informations sur les enregistrements ;
b) il permet d'ordonner rapidement les données ;
c) un fichier indexé peut posséder plusieurs index.

3 / En quoi consiste un tri ?
a) à extraire les données dans l'ordre ;
b) à chercher une valeur déterminée parmi les données d'un fichier ;
c) à recopier un fichier.

4 / Enumérez les principales différences entre recherches séquentielle et dichotomique.

5 / Un fichier du personnel doit contenir : le prénom, le nom, la qualification, la catégorie, l'ancienneté, le salaire de base, les jours d'absence.
Définissez le format de l'enregistrement en choisissant des longueurs pour les différentes zones.

6 / Définissez la structure des enregistrements d'un fichier de gestion des stocks qui doit contenir les informations suivantes : article, quantité en stock, cote d'alerte, date du dernier prélèvement, nom et prénom du fournisseur.

7 / Pour la question 6, on peut gagner de la place en codant certaines zones. Essayez de coder la date de prélèvement (jour, mois, année) et le nom du fournisseur (on supposera qu'il y a 100 fournisseurs au maximum).

Les solutions du test se trouvent pages 286 et 287.

contiendra les données ordonnées.

2 / Le deuxième enregistrement (qui contient L) est transféré en deuxième position du fichier de sortie puisque son contenu est supérieur à celui du premier (L > B).

3 / L'enregistrement 3 du fichier de données contient A, c'est-à-dire une donnée inférieure à celle qui sont déjà présentes dans le fichier de sortie (B et L). Il faut donc le mettre en position 1.

3-1 / La position 1 étant déjà occupée par la lettre B, il faut la libérer en décalant le contenu du fichier de sortie d'une position vers la droite. Ainsi, la lettre L passe de la position 2 à la position 3, tandis que B vient en position 2.

L'enregistrement 1 est disponible pour A.

4 / La lettre O (4e donnée) est placée en dernière position puisqu'elle est supérieure à toutes les autres lettres du fichier de sortie.

5 / Cette fois, le nouvel enregistrement à recopier (N) doit être placé avant O que l'on décale donc d'une position (étape 5-1).

6 / La lettre R (enregistrement 6 du fichier de données) est placée en dernière position.

7 / Pour insérer la lettre C (enregistrement 7), il faut décaler les quatre derniers enregistrements du fichier de sortie (L, N. O, R) afin de libérer une place entre B et L.

7-1 / Le dernier enregistrement de données (C) est inséré à cette place et le fichier de sortie est complet.

Les banques de données spécialisées

Dans un monde en constante évolution, la ***demande d'information*** *en matière économique et scientifique, tant dans le secteur privé que public, ne cesse de s'accroître.*
Or la masse de documentation publiée chaque jour dans le monde entier est telle qu'il est devenu impossible de s'en tenir régulièrement informé. A moins d'en faire un véritable métier.
Les organisateurs de bases et de banques de données rassemblent, sélectionnent et mémorisent une abondante documentation sous une forme condensée. Certaines banques de données extrêmement spécialisées ont choisi de centraliser toute la littérature qui se rapporte à un unique secteur, mais elles sont rares et les banques à caractère pluridisciplinaire sont aujourd'hui beaucoup plus nombreuses. Enfin, mentionnons les banques de données collectant des ***informations statistiques, démographiques*** *et* ***commerciales.*** *Elles sont très nombreuses et leur utilité augmente d'autant plus que l'arrivée des informations se fait à un rythme sans cesse croissant.*
Le documentaliste qui travaille dans un petit centre ne peut consulter que quelques-unes des publications susceptibles de l'intéresser. Au contraire, s'il a la possibilité d'interroger une banque de données, il obtient une liste à jour des ***références bibliographiques*** *concernant son sujet et ne laisse ainsi échapper aucune information utile.*
Il existe également des banques de données spécialisées dans la ***diffusion des brevets,*** *et qui permettent aux entreprises, même de petite taille, de se tenir au courant de l'apparition, dans leur secteur d'activité, de procédés ou de produits nouveaux et constituent soit un risque de concurrence soit, au contraire, une heureuse opportunité.*
Les banques de données favorisent également la transmission rapide des informations dans des domaines aussi variés que la médecine, le commerce international, le droit, entre autres.
Le marché de l'information est particulièrement développé dans les pays dont la structure de production privilégie les industries de transformation, et qui sont constamment éperonnés par les progrès accomplis à l'extérieur, tant sur le marché des facteurs de production (nouvelles technologies) que sur celui des produits finis. Une telle économie devrait mobiliser toutes ses ressources dans l'innovation technologique et commerciale afin de conserver et (si possible) d'améliorer sa position. Les premières entreprises qui se sont intéressé à la ***« documentation en ligne »,*** *sont apparues aux Etats-Unis, pays où des systèmes très performants de traitement des données pouvaient s'appuyer sur le réseau de télécommunications le plus moderne du monde. Lorsque, par la suite, ces services se sont développés et que le nombre des utilisateurs potentiels s'est accru, les sociétés américaines ont commencé à proposer leurs fichiers de l'autre côté de l'océan. Entre-temps, dans tous les pays à haut niveau technologique, de telles entreprises d'information ont été créées en Europe sur le modèle américain, proposant des fichiers comparables, ou revêtant (INSEE) un caractère national et rassemblant des données originales.*
Pour permettre à ce patrimoine de se développer sur un plan européen et pour acquérir leur autonomie technologique, les services de Postes et Télécommunications de la CEE ont créé ensemble le premier réseau européen, EURONET. Il est le fruit d'un accord aux termes duquel la libre concurrence est établie entre toutes les banques de données européennes. Les utilisateurs y accèdent, en effet, de façon indifférenciée. Ainsi, l'ouverture des frontières, souvent illusoire dans le domaine des échanges commerciaux, se réalise dans celui des télécommunications.
Opérationnel depuis 1980, EURONET offre le support des télécommunications aux banques de données européennes réunies sous le label DIANE (Direct Information Access Network in Europe). Son caractère d'internationalité est ce qui fait la particularité de ce réseau.
L'utilisateur qui désire interroger un fichier situé dans l'un des pays membres souscrit un contrat d'utilisation d'EURONET et doit disposer d'un terminal connectable au réseau téléphonique. Le raccordement au réseau est ensuite pris en charge automatiquement par les ordinateurs qui constituent les différents

« nœuds » (points d'accès).
Il suffit à l'utilisateur européen de téléphoner au nœud d'une grande ville de son pays pour pouvoir, sans autre frais que celui d'une communication téléphonique interurbaine, se relier aux banques de données d'Allemagne, de France, de Grande-Bretagne, de Hollande, d'Italie, de Belgique ou du Luxembourg. La distance ne retentit pas sur le prix, et l'utilisateur n'a à payer que son appel. De plus, les tarifs d'EURONET sont très souvent inférieurs aux tarifs américains.
Toutefois, il est à peine plus compliqué de consulter les banques de données situées aux Etats-Unis. Il suffit de s'abonner au réseau de télécommunications assurant le lien entre le pays de départ et les nombreux réseaux américains (networks). A ces frais assez peu élevés s'ajoutent ceux du contrat avec le ou les réseaux américains qui, d'ailleurs, comme la plupart des banques de données reliées à EURONET, ne facturent que l'utilisation effective et n'imposent pas de minimum.
Les fichiers accessibles par ces réseaux ont des contenus, et donc des utilisations, extrêmement variés. Il faut établir une distinction fondamentale entre banques de données et bases de données, en fonction de leur contenu.

Les **bases de données** sont des fichiers organisés que l'on peut interroger à partir d'un terminal et qui contiennent ce qu'on appelle des publications secondaires. Il s'agit de comptes rendus synthétiques de revues, de monographies, de livres, d'exposés de conférences, destinés à fournir à des usagers très différents un moyen de sélection efficace des documents traitant d'un sujet précis.
Elles sont souvent établies par des organismes (associations liées à un secteur de production, institutions universitaires ou encore entreprises privées spécialisées dans le traitement de l'information) qui éditent déjà des résumés de revues ou d'articles. Parfois les fichiers couvrent alors plusieurs disciplines.
Pour chaque article, citation, ou monographie recensés, on indique : le titre (en anglais et dans la langue d'origine si elle est différente de l'anglais) ; l'auteur ; la source (revue, article, monographie) et tous les autres éléments qui permettent d'identifier l'original ; une liste de descripteurs ou mots clé, tirés du texte et permettant d'en préciser la portée et de s'assurer qu'il concerne la recherche menée ; et enfin, le résumé, à la fois concis et complet, du document.
Le résultat final d'une recherche parmi ces fichiers fournira l'ensemble des références

L'exploitation des banques de données est particulièrement utile dans le domaine de la recherche scientifique. Ainsi, pour les travaux menés à l'Observatoire de Greenbank, en Virginie (Etats-Unis).



Marka

bibliographiques que le documentaliste aura pu sélectionner en combinant un certain nombre d'éléments qui définissent de façon univoque le sujet choisi.

Ce service s'avère très utile aux personnes éloignées de la source d'information principale, en leur épargnant la difficulté de choisir les publications nécessaires à leur travail. C'est notamment le cas des gens qui ne disposent pas de bibliothèque à proximité de chez eux, ou qui ont besoin, pour une utilisation particulière, d'une revue qu'ils ne parviennent pas à trouver parmi celles qui sont à leur disposition.

Avec les bases de données, ils disposent ainsi d'un outil de sélection rapide des documents en rapport avec leur sujet. Pour préparer l'interrogation d'une base de données, on sélectionne les fichiers spécialisés qui semblent correspondre le mieux au thème de la recherche, et également certains fichiers pluridisciplinaires susceptibles d'être intéressants.

La ***stratégie de la consultation*** *découle de la nature des fichiers interrogés. Si l'on craint de recueillir un trop grand nombre de références par une approche trop générale, on peut restreindre le champ de la recherche en définissant des conditions plus précises (date de mise à jour, type de source, pays d'origine, auteur, etc.). Les instructions de recherche contiennent la suite logiquement ordonnée des mots que le documentaliste s'attend à trouver dans le titre ou dans le texte et qui spécifient les caractéristiques du sujet recherché.*

Voici un bref exemple de stratégie de recherche avec ses instructions d'extraction et l'impression de l'une des références bibliographiques trouvées.

Le but de la recherche, menée sur la base de données PAPERCHEM, développée par The Institute of Paper Chemistry et accessible par le réseau américain DARDO, était de trouver le compte rendu d'une conférence intitulée Energy from Biomass Conference.

L'abréviation CT (conference title) utilisée dans cet exemple est un opérateur qui désigne un certain type de documents et que PAPERCHEM connaît.

Voici les instructions de consultation :

SELECT CT = ENERGY (1W) CT = BIOMASS, ce qui signifie : « Sélectionne, parmi les titres de conférence, le mot ENERGY ; tu dois trouver le mot BIOMASS dans la même zone ». L'opérateur (1W) précise que les deux mots doivent se trouver dans l'ordre indiqué (Biomass doit venir après Energy dans le titre) et qu'ils peuvent être reliés par un troisième mot.

?SELECT CT = ENERGY (1W) CT = BIOMASS
33 83 CT = ENERGY (1W) CT = BIOMASS

L'impression de la référence trouvée permet d'obtenir :

1. le titre du compte rendu ; 2. son auteur ; 3. le titre de la conférence (c'est la zone dans laquelle la recherche a été effectuée) ; 4. les pages du document d'origine qui contiennent ce compte rendu ; 5. la date de la conférence ; 6. le nombre total de comptes rendus ; 7. la classification du document ; 8. la langue d'origine ; 9. le résumé ; 10. les descripteurs.

Quant aux ***banques de données,*** *elles présentent une caractéristique fondamentale. Leur interrogation fournit directement les données finales recherchées, car les fichiers des banques de données ne contiennent pas de publications secondaires (descripteurs d'originaux) mais, essentiellement, des copies de documents originaux consultables à partir d'un terminal.*

On accède ainsi par l'écran aux publications des organismes de statistiques nationales ou des centres universitaires qui rassemblent eux-mêmes des données provenant de leurs propres recherches ou d'institutions internationales telles que la FAO ou l'OCDE.

Ces données peuvent servir à des traitements ultérieurs.

Ces fichiers sont destinés à des utilisateurs spécialisés, et associent souvent aux procédures normales d'interrogation (comparables à celles des bases de données bibliographiques), des fonctions parfois très complexes de traitement numérique des données trouvées. On peut ainsi combiner plusieurs séries numériques, insérer, traiter et conserver des données à usage confidentiel, obtenir des représentations graphiques, intercaler des données manquantes, extrapoler à partir de modèles mathématiques fournis par la banque ou mis au point par l'utilisateur.

(D'après A. Beverini. Extrait de BANCA DATI, in CELLULOSA E CARTA n° 4, juillet-août 1983.)

Les différents points exposés mettent en évidence les phases essentielles d'un programme de tri : comparaison des données et déplacements consécutifs au sein du fichier de sortie. Nous reviendrons sur ce sujet lorsque nous aurons étudié les principales instructions du Basic.
Recopier le fichier de données présente deux inconvénients : d'une part, on occupe davantage de place dans la mémoire de masse puisqu'à l'issue du tri on a deux fichiers au lieu d'un et, d'autre part, l'exécution du tri est beaucoup plus lente puisqu'il faut déplacer les données du fichier de sortie pour inscrire presque tous les enregistrements. En utilisant un index (ou fichier d'adresses), non seulement on économise la mémoire de masse, mais surtout on a la possibilité d'exécuter les déplacements en mémoire centrale (l'index ne contient que des pointeurs : il est beaucoup plus petit et l'ensemble d'un fichier index peut donc rester en mémoire centrale), ce qui est beaucoup plus rapide.

Le compactage des données

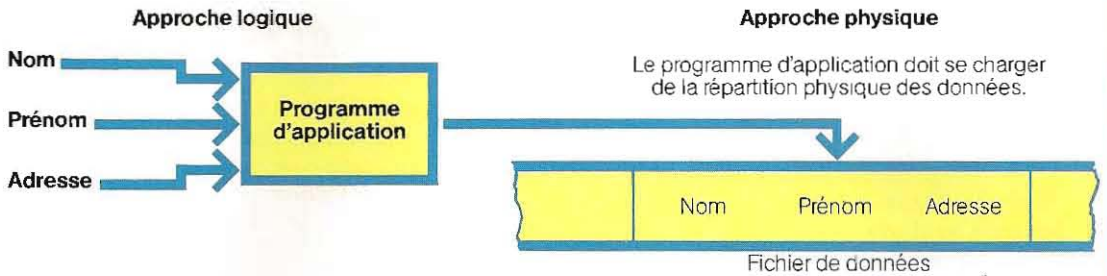
Pour certaines applications, on est obligé de condenser ou « compacter » les données avant de les inscrire sur les supports de mémoire physique.
Sur les gros systèmes qui traitent d'énormes quantités de données, le fait de gagner quelques caractères par donnée peut entraîner une réduction importante de l'espace occupé. Une meilleure exploitation des mémoires justifie alors une plus grande complexité des programmes (qui doivent compacter les données à l'écriture, puis les étendre à nouveau à la lecture). Sur les micro-ordinateurs, au contraire, ces techniques compliquées sont rarement adaptées, exception faite de quelques cas particuliers que nous étudierons par la suite.
Les principales méthodes de compactage reposent sur l'élimination des zones et des caractères vides et sur la codification de certaines combinaisons de caractères.
Les zones de données vides. Il vaut mieux les éliminer que les remplir de blancs. On obtient ainsi un enregistrement plus court que les autres. En contrepartie, on devra utiliser des programmes plus lourds, capables de traiter des enregistrements de longueur variable.
L'élimination des caractères inutiles. Dans les zones de format ASCII, on peut éliminer les blancs qui se trouvent entre deux caractères ainsi que les zéros dans les zones numériques. Une fois encore, le programme sera plus complexe car il devra être conçu de façon à retrouver combien d'espaces ou de zéros ont été éliminés afin de pouvoir reconstituer les données sous leur forme initiale avant de les présenter en sortie.
Le codage des combinaisons de caractères. Lorsque des combinaisons de caractères reviennent souvent, on peut les représenter par un symbole. Ainsi, dans le fichier du personnel, nous trouvons les noms Rousseau, Lenoir et Leblanc.
En associant, par exemple, « ousseau » à 1 et « blanc » à 2, les données deviennent :
R1, Lenoir et Le2.
Cette technique n'est intéressante que si les séquences ainsi codifiées reviennent fréquemment.

Les banques de données des gros systèmes

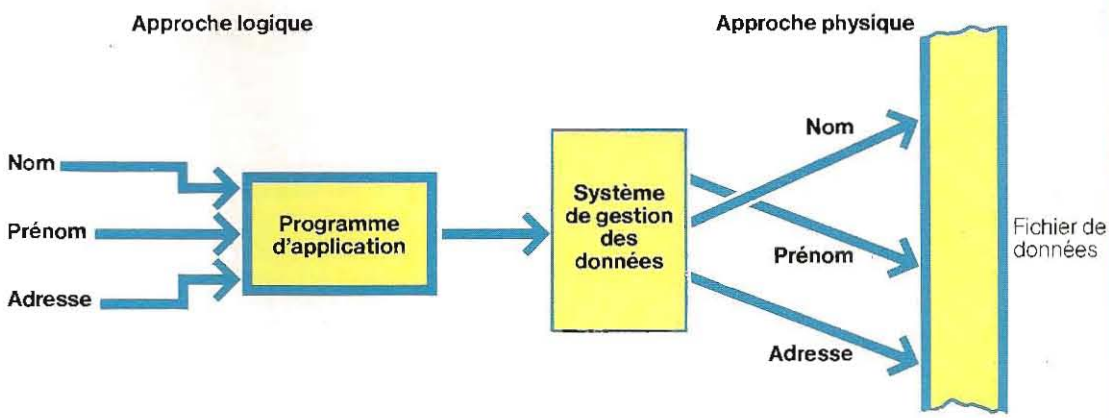
Les gros ordinateurs et certains mini-ordinateurs possèdent des programmes de gestion de données très complexes.
Grâce à ces programmes l'utilisateur n'a plus à s'occuper de la structure physique des fichiers mais simplement à définir leur structure logique.
Les programmes-système se chargeront ensuite de la traduire en schémas physiques et de gérer fichiers, zones et enregistrements. Le programmeur ne connaît plus l'adresse physique des données puisque l'espace mémoire disponible (disques, tambours) est géré par le système. Les schémas de la page 284 illustrent les différences entre ces deux méthodes.
Dans le premier cas (pas de programme de gestion des données), le programmeur doit définir la structure logique de son fichier (ici, le nom, le prénom et l'adresse), la structure physique des enregistrements et leur position dans le fichier.
Dans le second cas, il n'a que la structure logique à définir. C'est le système qui décide où et comment écrire les données, à l'aide des programmes de gestion.
Sur les systèmes les plus évolués, la gestion des données se fait parfois au moyen d'un

langage spécialement étudié, différent des langages d'application.
Dans ce cas, le langage de gestion des données est lié à un langage normal de programmation (Basic, Cobol, etc.) et l'utilisateur peut se servir indifféremment des instructions des deux langages.
L'importance de cette question a été à l'origine de la création à Washington, en 1959, du CODASYL (Conference on Data System Language), organisme chargé d'unifier les recherches effectuées dans le monde entier et de publier des normes à observer dans le développement des systèmes de bases de données (structures logiques et relations entre les données).

GESTION DES DONNEES ET PROGRAMMES D'APPLICATION

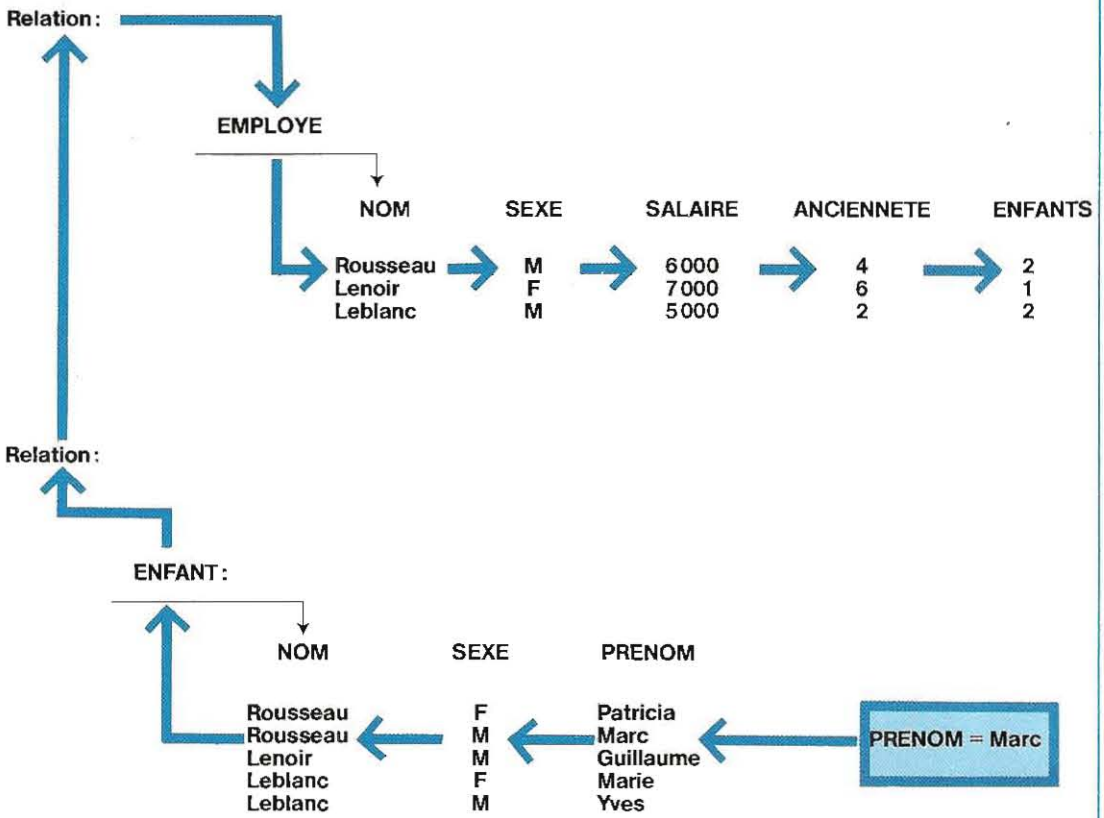


En l'absence de système de gestion des données évolué, les programmes d'application doivent tout prendre en charge, y compris l'adressage des données dans les mémoires de masse.



Sur les gros systèmes, des programmes de gestion des données simplifient beaucoup le travail de programmation. Le système prenant totalement en charge leur gestion physique, l'utilisateur n'a plus à définir que la structure logique de ses données.

EXEMPLE DE RECHERCHE DANS UNE BASE DE DONNEES RELATIONNELLE



Le « domaine » PRENOM (PRENOM = Marc), à travers les relations ENFANT et EMPLOYE, permet de sélectionner toutes les données de l'employé Martin.

Les bases de données relationnelles

Elles permettent l'indépendance totale des structures logique et physique des fichiers. Dans une base de données relationnelle, il n'est défini qu'une structure logique et toutes les données sont reliées entre elles par des « relations ». On peut, en partant de n'importe quelle donnée ou « domaine » (qui constitue la clé de la relation), accéder à toutes les autres données associées, au moyen des relations qui existent entre elles.

Le schéma ci-dessus illustre l'exemple d'un fichier du personnel où coexistent deux relations : Employé et Enfant. A partir d'une valeur (Marc) du domaine PRENOM, on obtient toutes les informations sur l'employé.

Cette base de données, qui est la forme la plus évoluée de gestion des données, n'est proposée que sur de gros ordinateurs.

Solutions du test 7

1 / c) les enregistrements ne peuvent être ni lus, ni écrits sélectivement dans un fichier séquentiel. Les données sont enregistrées séquentiellement, c'est-à-dire dans l'ordre de leur introduction.

---

2 / b et c) l'index est une zone du fichier (ou un fichier à part) où sont enregistrés les clés et les pointeurs vers les enregistrements de données. Il peut, en général, y avoir plusieurs index, correspondant chacun à une clé d'accès différente.
Dans un fichier du personnel qui contient les informations suivantes :
– prénom et nom – qualification, catégorie, service – salaire de base, taux horaire des heures supplémentaires – jours de congé, d'absence, de maladie –
il peut être utile de disposer de plusieurs index : un premier pour les noms (qui constituent la clé principale), d'autres pour les données secondaires, de façon à pouvoir accéder aux enregistrements selon un autre critère que le nom. Ainsi, on peut, grâce à l'index des jours de congé, obtenir la visualisation de tous les employés qui devront bénéficier d'une certaine période de vacances, ou encore se servir de l'index des services pour établir la liste du personnel d'un même service. Sur les systèmes les plus évolués, on peut procéder à une recherche « croisée », qui combine les zones clé à l'aide d'opérateurs logiques. On pourra ainsi, par exemple, sélectionner les employés du service d'entretien qui ont droit à 20 jours de congé et n'appartiennent pas à la catégorie A. D'un point de vue logique, cette instruction correspond à la formule suivante : (service = entretien) ET (congés = 20) ET (catégorie ≠ A). Le symbole ≠ signifie « différent de » (en Basic, on écrira « catégorie < > A »).

---

3 / a) c'est ainsi qu'on appelle l'opération de classement. Rappelons que le tri d'un fichier consiste à extraire les données dans l'ordre, selon une zone définie comme clé. Dans l'exemple précédent (**2**/), l'existence de plusieurs index (nom, congés, service) permet d'obtenir des listes du personnel ordonnées différemment (autant qu'il y a de zones clé). On peut également combiner plusieurs conditions au cours d'un même tri. On pourra ainsi demander une liste du personnel par service, où les noms des employés apparaîtront par ordre alphabétique à l'intérieur de chaque service.

---

4 / La recherche séquentielle consiste à lire tous les enregistrements d'un fichier jusqu'à la découverte de celui que l'on cherche. La recherche dichotomique ne lit que les enregistrements obtenus par divisions binaires successives, ce qui va beaucoup plus vite. On ne peut l'utiliser que sur des fichiers ordonnés, ce qui nécessite souvent un tri préalable. Lorsqu'un fichier contient peu de données, le tri et la recherche prennent autant de temps que la recherche séquentielle.

---

5 / Cette solution n'a qu'une valeur indicative puisque le résultat dépend des longueurs des zones utilisées. Une moyenne raisonnable donnerait :

prénom et nom	50 caractères
qualification	2 caractères (code de deux lettres)
catégorie	2 caractères (code)
ancienneté	2 caractères (de 0 à 99 ans)
salaire	5 caractères (jusqu'à 99 999 francs)
jours d'absence	2 caractères (jusqu'à 99)
TOTAL	63 caractères

On utiliserait donc ainsi 63 caractères. En adoptant une longueur de 80 caractères, on disposera d'une marge de 17 caractères pour d'éventuels ajouts.
Pour avoir un ordre de grandeur de la capacité de stockage d'une disquette, calculons le nombre d'employés dont on pourrait stocker les données sur des disquettes de 250 Ko et d'1 Mo (1 Ko = 1 000 octets ; 1 Mo = 1 méga-octet = 1 000 Ko).
La disquette de 250 Ko contient 250 000 caractères, ce qui correspond à :

$\frac{250000}{80}$ = 3125 données de 80 caractères.

La disquette d'1 Mo contient 1 000 000 de caractères, ce qui correspond à :

$\frac{1000000}{80}$ = 12 500 données.

Une seule disquette peut donc contenir le fichier du personnel d'une entreprise de 12 000 employés.
Dans notre exemple, des données numériques comme le salaire ont été considérées comme des chaînes de caractères. En effet, les systèmes d'exploitation les plus répandus en Basic représentent en mémoire de masse les valeurs numériques sous la forme de caractères. Certaines machines utilisent cependant le format numérique où les nombres sont traités comme tels. On économise alors de la place car le format numérique binaire en occupe moins que le format caractère (ASCII). Ainsi, en binaire le nombre 60 n'occupe qu'un octet (60 en base 10 = 00111100 en base 2), alors qu'en code ASCII, il lui faudrait deux octets, un pour chaque caractère (6 et 0, représentés respectivement par 0110110 et 0110000).

---

6 / Cet exercice est semblable au précédent. Voici un exemple de solution :

article	20 caractères (désignation)
quantité	4 caractères (jusqu'à 9 999)
cote d'alerte	4 caractères
date	8 caractères (jour = 2 caractères, mois = 2, année = 4 ; ex. : 21 071983)
fournisseur	20 caractères
TOTAL	56 caractères, ce qui donne 70 si l'on prévoit une marge raisonnable.

---

7 / Il y a deux façons très simples de gagner de la place dans le fichier précédent. On peut déjà coder la date par les deux derniers chiffres de l'année : 83 au lieu de 1983. On peut également désigner les fournisseurs par des numéros : Rousseau = 0, Lenoir = 1, etc. On établit ainsi une codification des 100 fournisseurs à l'aide des nombres compris entre 0 et 99. Ces deux modifications permettent d'économiser 2 caractères pour la date et 18 pour le fournisseur. Les données n'occupent plus que 36 caractères et l'on pourra donc adopter une longueur de 50 caractères au total pour conserver une marge. On pourra ainsi faire tenir 5 000 enregistrements de ce fichier sur les petites disquettes et 20 000 sur les plus grandes.

GLOSSAIRE

Les mots clés de la gestion des fichiers

Accès. Opération de lecture ou d'écriture des données sur les mémoires de masse (disques durs ou souples, etc.).

Adressage indirect. Il permet de repérer une donnée à l'aide d'un fichier index contenant des pointeurs. La recherche se fait en deux temps : le premier pointeur désigne un enregistrement de l'index qui, lui-même, contient l'adresse de la donnée.

Adresse. Valeur numérique indiquant la position d'un enregistrement.

Algorithme. Méthode de calcul utilisée pour la résolution d'un problème donné.

Banque de données. Ensemble de données relatives à un certain nombre de sujets et organisées en vue de leur utilisation par des programmes correspondant à des applications distinctes. Elles présentent une caractéristique fondamentale : les informations que l'on obtient quand on les interroge d'un terminal correspondent aux données finales recherchées. En effet, ces banques ne contiennent pas de publications secondaires mais des copies de documents originaux.

Base de données. Ensemble de fichiers organisé selon un même schéma logique. On distingue bases et banques de données, même si les deux termes sont parfois confondus. Les bases de données désignent, notamment, les fichiers où sont enregistrées les publications secondaires. On peut les interroger à partir d'un terminal. Ce sont des résumés normalisés et concis de revues, livres, monographies ou rapports de conférences, destinés à fournir à des usagers très divers un outil de sélection efficace de documents en rapport avec leur recherche.

Buffer. Voir **Zone tampon**.

Chaînage. Les enregistrements sont dits chaînés lorsque chacun d'eux contient un pointeur qui désigne l'enregistrement suivant. Cette technique, utilisée notamment dans les banques de données, permet de former une liste dans laquelle chaque enregistrement est chaîné à son successeur, quelle que soit sa position physique en mémoire. Le pointeur qui sert au chaînage constitue alors une zone supplémentaire de chaque enregistrement.

Clé. Zone d'un enregistrement permettant de le rechercher.

Compactage (ou condensation). Technique qui permet d'écrire une donnée sur un nombre plus réduit de caractères

DAM (Direct Access Method). Méthode d'accès direct.

Data base. Voir **Base de données**.

Dictionnaire. Liste des noms ou étiquettes de données prévus dans le cadre d'un fichier ou d'un programme.

Fichier. Espace physique et logique destiné à rassembler un ensemble de données homogènes.

Fusion. Ce terme désigne l'opération (et, par extension, le programme correspondant) qui consiste à réunir les données contenues dans des fichiers différents.

ISAM (Indexed Sequential Access Method). Méthode d'accès séquentiel indexé.

Liste. Voir **Chaînage**.

Mémoire virtuelle. Des blocs entiers du programme ne pouvant tenir simultanément en mémoire centrale sont transférés vers la mémoire de masse, puis ramenés en mémoire centrale lorsqu'ils sont nécessaires à l'exécution. Ces opérations, effectuées par le système, sont extrêmement rapides et permettent d'augmenter artificiellement la taille de la mémoire centrale.

Méthodes d'accès. Méthodes de lecture et d'écriture des données. Les principales sont l'accès aléatoire (random access), l'accès séquentiel (SAM) et l'accès séquentiel indexé (ISAM).

Pointeur. Valeur numérique indiquant la position d'un enregistrement ou d'une donnée.

Random. Voir **Méthodes d'accès**.

Recherche dichotomique. Méthode de recherche des données fondée sur des divisions successives en deux parties.

Relationnelle (Base de données). Forme particulière de structuration logique des données. Les zones (ou domaines) sont liées logiquement par des relations.

Répertoire. Index des données ou des fichiers existants et de leurs caractéristiques physiques (longueur et emplacement dans la mémoire de masse).

SAM (Sequential Access Method). Méthode d'accès séquentiel.

Secteur. Division physique d'un disque. C'est la plus petite unité adressable par le système d'exploitation, lors des opérations de lecture et d'écriture.

Temps d'accès. Temps nécessaire à l'acquisition d'une donnée. Il comprend le temps de positionnement de la tête de lecture et le temps de transfert de la donnée de la mémoire de masse à l'unité centrale.

Tri. Classement d'un fichier en fonction d'une clé.

VTOC (Volume Table Of Content) ou Directory list. Liste du contenu d'un disque.

Zone. Donnée élémentaire. Un enregistrement de fichier stock comprend, par exemple, une zone de description de l'article, une zone pour le prix, etc.

Zone tampon. Zone de mémoire utilisée pour le stockage momentané des données. Sur les micro-ordinateurs, les données transitent toujours par une zone tampon au cours des échanges avec la mémoire de masse. Ainsi, lorsqu'on écrit sur une disquette, la donnée passe d'abord de la mémoire centrale à une zone tampon, avant d'être transférée sur la disquette.

Etablissement des programmes

La rédaction d'un programme est précédée par l'analyse, étape très importante, abordant les problèmes conceptuels, logiques et physiques. Lorsqu'il s'agit d'un gros système, il est indispensable de répartir cette activité entre plusieurs informaticiens, chacun d'entre eux ayant une fonction bien définie à assumer.
Sur les ordinateurs individuels et les micro-ordinateurs, c'est souvent une seule personne, l'utilisateur lui-même, qui effectue à la fois l'analyse et la programmation proprement dite.
La marche à suivre dans cette phase d'analyse est la même pour toutes les applications et peut se résumer en trois points essentiels :

1 / Définition des sorties (résultats) nécessaires.
2 / Vérification des données en entrée et des algorithmes de calcul.
3 / Choix du matériel.

Une fois ces trois points définis, on peut passer à la conception des organigrammes et à l'écriture des programmes.

Définition des sorties nécessaires

La première étape consiste à définir le type et le volume des sorties à effectuer.

1 / Editions : leur nombre, le type de données qu'elles contiennent, et la périodicité requise.
2 / Fichiers : type ; disposition et nature des données à enregistrer sur disque.

Configuration classique d'un ordinateur individuel (familial) de taille moyenne.



IRET

Cette définition précise des problèmes qui se posent permet d'établir non seulement les variables de sortie, mais aussi le type d'imprimante le mieux adapté à l'application.
On peut, en effet, en fonction de la longueur prévisible du listage et du temps nécessaire, définir la vitesse d'impression la plus appropriée (en caractères par seconde), la densité de l'impression et le format.
Pour un ordinateur individuel, le choix de l'imprimante la mieux adaptée à l'écriture des résultats d'un programme est évidemment dicté par la configuration du système. Au moment même de l'achat, il faudra définir à un niveau plus général le type de travail que l'on aura besoin d'effectuer. L'imprimante « idéale » sera celle qui permettra de concilier contraintes économiques et volume de travail à accomplir régulièrement.
Il existe sur le marché de très nombreux types d'imprimantes, mais les plus performantes d'entre elles n'intéressent pas directement le grand public. Nous nous bornerons ici à étudier les plus classiques et leurs caractéristiques fondamentales, déterminant leurs possibilités d'utilisation.

Imprimantes à aiguilles
80 colonnes (impression de 80 caractères par ligne, sur des feuilles de dimensions réduites) ;
100 caractères par seconde (soit 1,25 ligne par seconde, environ).

132 colonnes (papier grand format) ;
140 caractères par seconde (1 ligne par seconde).

132 colonnes ;
200 caractères par seconde.

Les imprimantes à aiguilles sont, en général, bidirectionnelles : leur tête d'écriture imprime les caractères, aussi bien de gauche à droite que de droite à gauche, ce qui permet de gagner du temps.

Imprimantes à marguerite
Leur vitesse d'impression varie de 20 à 50 caractères par seconde, en moyenne.

Imprimantes par ligne (ou parallèles).
Les machines de ce type écrivent plusieurs centaines de lignes (jusqu'à 2 500) par minute.

Les imprimantes à aiguilles sont les plus utilisées sur les micro-ordinateurs, et le choix s'effectuera en fonction de la rapidité d'exécution souhaitée et du format de papier adopté.
Les imprimantes à marguerite coûtent environ deux fois plus cher et ne sont donc employées que lorsqu'une impression d'excellente qualité est nécessaire.
Quant aux imprimantes par ligne, elles coûtent plus de dix fois plus cher que les imprimantes à aiguilles et leur achat n'est donc justifié que si l'on dispose au moins d'un mini-ordinateur très puissant.
La seconde étape de définition des sorties permet de fixer le volume d'informations à mémoriser sur disque. En calculant la somme des emplacements physiques nécessaires à leur stockage, on définit approximativement la taille des fichiers et donc la capacité des disques à utiliser.
Nous avons déjà vu que la capacité des disques souples varie entre 80 000 et 1 200 000 caractères tandis que celle des disques durs est de 5 000 000 d'octets (5 Mo) au minimum, et va jusqu'à 400 000 000 (400 Mo).
Le coût d'un système informatique dépend environ pour moitié du type de disque qui lui est associé. Le choix de ce support a donc des répercussions considérables et ne doit pas être fait à la légère.
Il faut toutefois tenir compte des développements futurs envisagés : opter pour une solution plus coûteuse permet de disposer de la marge de sécurité nécessaire.

Vérification des données en entrée et des algorithmes de calcul

Après avoir défini les sorties, on doit s'assurer que l'on dispose de toutes les données nécessaires en entrée, et que l'on maîtrise les algorithmes indispensables aux calculs éventuels.
Cette étape vise donc essentiellement à :

- vérifier que les données utiles sont disponibles,
- s'assurer de la nécessité de conserver sur disque certaines données,
- définir les méthodes de calcul.

Certaines données, ne servant qu'au cours

des calculs, n'apparaissent jamais sur les listages.
Mais parfois il faut tout de même les conserver sur disque et donc en tenir compte dans l'évaluation de la capacité de mémoire de masse nécessaire.

Choix du matériel

Conditionnée par le type d'application envisagé, cette dernière étape est celle qui pèsera le plus lourdement sur l'écriture des programmes.
A ce stade, on a défini les périphériques du système (imprimante et unité de disque). Toutefois, il reste encore à choisir l'unité centrale.
On distingue principalement trois grands types de micro-ordinateurs aux performances différentes :

- **les micro-ordinateurs 8 bits ;**
- **les micro-ordinateurs 16 bits ;**
- **les micro-ordinateurs 32 bits.**

Si les premiers peuvent adresser 64 000 emplacements mémoires (octets) au maximum, les deuxièmes atteignent couramment 1 000 000 (1 Mo), et jusqu'à plusieurs millions d'octets pour les plus puissants.
Le choix, parmi les deux premiers, dépend donc, entre autres, de la quantité de données à traiter.
Sur les micro-ordinateurs 8 bits, on doit, face à un important volume de données, fractionner les programmes, ce qui accroît leur complexité.
En revanche, les machines 16 bits coûtent un peu plus cher et les extensions de mémoire sont onéreuses.
L'évolution très rapide de la technologie réduit chaque année les coûts de production. De plus, les systèmes d'exploitation ne sont pas toujours en mesure d'utiliser la totalité de leur capacité d'adressage.
Le système d'exploitation constitue en effet le dernier facteur qui guide le choix. Il est préférable d'opter pour des machines munies de compilateurs et utilisant des systèmes d'exploitation largement diffusés. Ainsi, les programmes seront portables et l'on aura l'avantage de disposer d'un système d'exploitation constamment mis à jour.

Les systèmes d'exploitation

Un ordinateur est, au départ, un ensemble de circuits électroniques, capable de recevoir et de traiter des signaux électriques de type binaire. Il peut accepter, sous forme binaire, les données à traiter et les indications indispensables sur l'enchaînement des opérations à effectuer. La suite ordonnée des codes de ces différentes opérations constitue le programme.
Pour utiliser une machine de ce type, nous serions contraints d'employer le système binaire, afin d'écrire directement les codes et les données en langage machine. Même les opérations les plus simples devraient être « décrites » sous cette forme dans chaque programme.
On comprendra sans peine que travailler sur ce type de matériel exigerait des programmeurs dotés de facultés tout à fait hors du commun. Cette situation constituait pourtant la réalité quotidienne des informaticiens il y a seulement vingt-cinq ans. A cette époque, rédiger des programmes signifiait, en effet, combiner des 1 et des 0 en des suites interminables, avec une probabilité d'erreurs extrêmement élevée.
C'est pourquoi les langages symboliques ont été introduits par la suite. Ce sont les langages d'assemblage (Assembleur) et les langages évolués (Fortran, Basic, Cobol, etc.).
Le système d'exploitation représente l'ensemble des programmes de service et des compilateurs ou interpréteurs, destinés à la traduction (compilation ou interprétation) des programmes en langage symbolique.
Il convient de distinguer radicalement les systèmes d'exploitation destinés aux utilisateurs isolés (et travaillant sur micro-ordinateurs), de ceux qui permettent la multi-utilisation ou, plus précisément, l'exécution simultanée de plusieurs programmes (sur mini-ordinateurs et gros systèmes principalement).
Dans ce dernier cas, le système, plus complexe, doit « savoir » répartir les ressources disponibles (unité centrale, mémoires, disques, périphériques) entre les différents programmes qui s'exécutent en même temps et cela, à une vitesse telle qu'aucun utilisateur ne soit gêné par la présence des autres si cela est possible.

Les fonctions du système d'exploitation

Prenons l'exemple plus concret d'un système à microprocesseur de la catégorie des micro-ordinateurs et des ordinateurs individuels, et présentant par conséquent les caractéristiques d'une machine de puissance moyenne :

- un seul utilisateur ;
- mémoire utilisateur de 64 000 octets ;
- unité d'affichage 24 lignes × 80 colonnes ;
- deux unités de disques souples ;
- un clavier ;
- une imprimante.

Supposons que la programmation soit prévue en langage Basic. La première condition que doit satisfaire le système d'exploitation est évidente : il doit être capable de traduire le Basic en langage machine. Un premier choix s'impose entre deux types de systèmes d'exploitation : ceux qui ne contiennent qu'un interpréteur et ceux qui possèdent également un compilateur. L'exécution des programmes est généralement plus rapide sur les systèmes équipés d'un compilateur.
L'analyse de la configuration du matériel détermine les autres fonctions du système d'exploitation. L'unité d'affichage, le clavier et l'imprimante sont des dispositifs d'entrée-sortie, que les programmes d'application doivent pouvoir utiliser en précisant simplement le périphérique concerné et les données à lire ou à écrire. Le système d'exploitation devra se charger de la gestion des fonctions complexes régissant le matériel, c'est-à-dire des mécanismes d'impression et de reconnaissance des touches frappées au clavier.
L'utilisation de disquettes suppose également un contrôle exercé par les programmes du système.
Enfin, il doit nécessairement pouvoir interpréter des instructions particulières entrées au clavier, tels les ordres de mise en marche ou d'interruption de l'exécution d'un programme.
Un système d'exploitation se compose donc, en général, de trois parties principales :

1 / le module de traitement des commandes ;
2 / le module de gestion des unités de disque ;
3 / le système de gestion des opérations d'E/S.

Nous allons expliquer les fonctions correspondant à chacune de ces parties et leur utilisation.

Le module de traitement des commandes. C'est le premier sollicité lorsqu'on branche la machine. La configuration d'un micro-ordinateur comprend souvent deux unités de disque. La première, en général désignée par la lettre A ou le chiffre 0, doit contenir la disquette du système d'exploitation. La mise sous tension déclenche l'exécution d'un programme d'amorçage (bootstrap), qui réside en permanence dans la mémoire. Le seul rôle de ce programme consiste à lire le système d'exploitation sur la « disquette système » (unité A ou 0) pour le transférer en mémoire. Après ce « chargement » du système, la machine peut interpréter et exécuter des instructions. Elle signale à l'opérateur qu'elle est prête à recevoir des instructions par un tableau (le plus souvent) qui résume les principales caractéristiques du matériel*, ou par un symbole d'« état prêt » (prompt). Les illustrations des pages 293 et 294 montrent cette phase d'initialisation sur les écrans de deux machines de type différent. On remarquera, dans les deux cas, l'apparition du symbole de « système prêt » à côté du curseur.
A ce stade, on ne peut donner que des instructions propres au système d'exploitation, commandant des opérations de base indépendantes du langage que l'on veut utiliser.

On a donc deux possibilités :
- demander l'exécution d'opérations de base ;
- entrer un programme en langage symbolique.

Pour écrire un programme en Basic, il faut communiquer (par ex.) l'instruction MBASIC à la première machine (Xerox 820) et l'instruction BA à la seconde (M20 Olivetti).

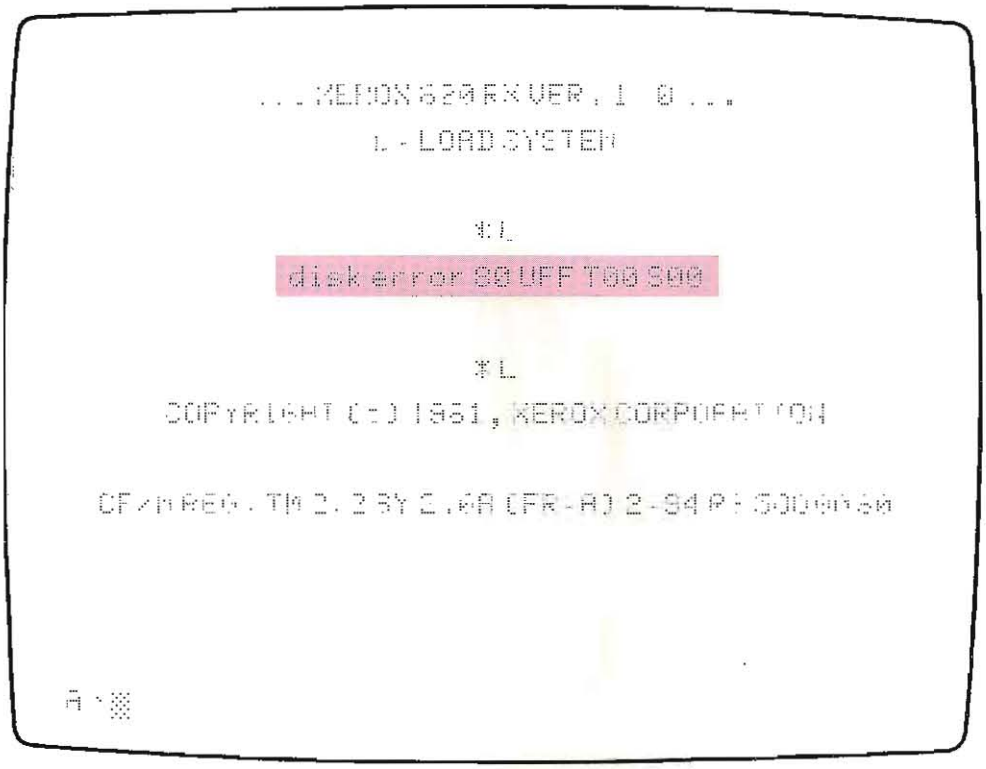
---

*Les caractéristiques du matériel (hardware), c'est-à-dire la capacité de la mémoire, le nombre des unités de disque, etc., définissent ce qu'on appelle la configuration du système. Le tableau résumant les caractéristiques de notre machine en est un exemple. On parle également de la configuration du logiciel (software) d'un système, pour désigner l'ensemble des programmes qui sont à la disposition de ses utilisateurs.

ECRAN DU SYSTEME XEROX 820 LORS DE SA MISE SOUS TENSION

Le Xerox 820 utilise le système d'exploitation CP/M qui réside sur une disquette et n'est accessible qu'après son chargement en mémoire. Lors de la mise sous tension, on a délibérément laissé l'unité de disque A (celle de gauche) ouverte, unité où doit normalement se trouver la disquette système.

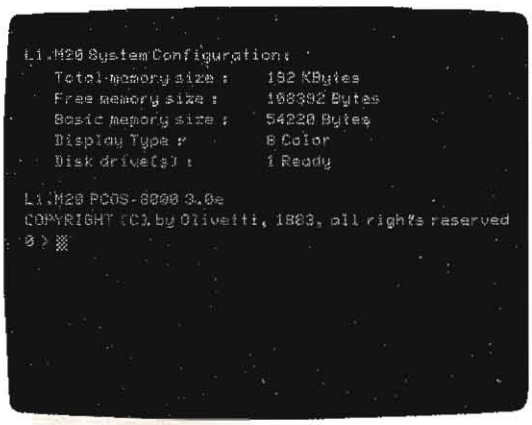


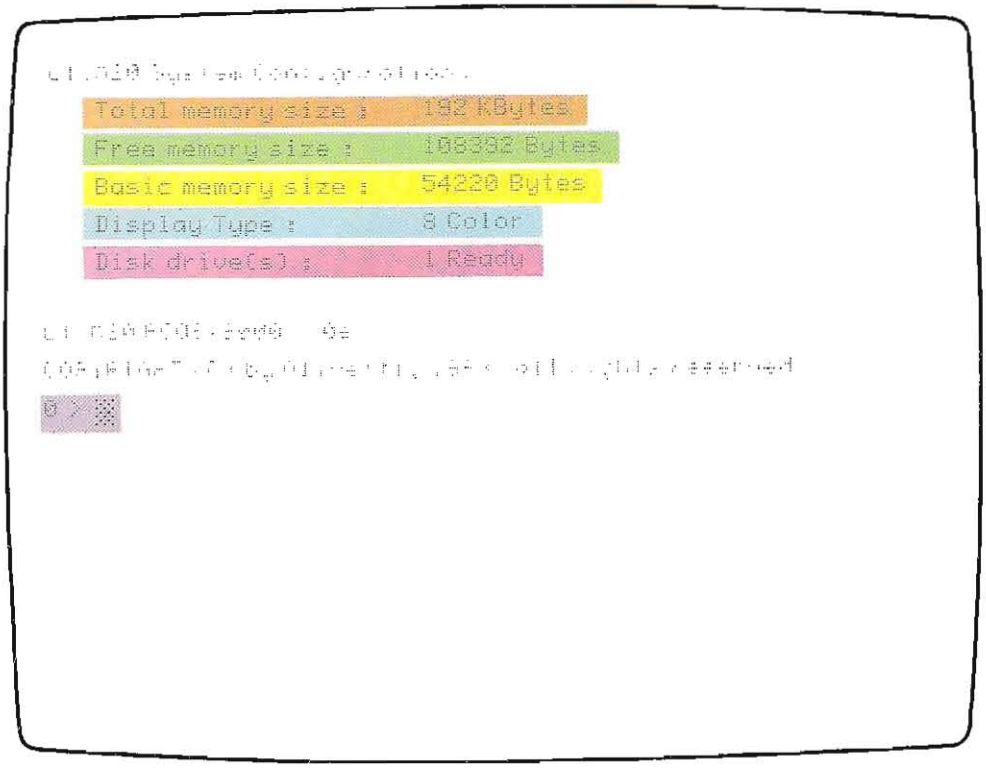


Le programme d'amorçage (ou d'initilisation), qui réside en permanence en mémoire ROM, essaie de lire le CP/M, mais l'unité de disque n'est pas préparée. Il émet donc un message d'avertissement, qui mentionne le disque de gauche (puisque c'est là que devrait se trouver le système d'exploitation).
Sur certaines machines, on peut inverser les unités de disque en modifiant une connexion électrique, ce qui permet de charger le système même si l'unité A est en panne.
Une fois le disque système inséré dans le lecteur de gauche, le système d'amorçage chargera le CP/M en mémoire vive.
L'utilisateur pourra choisir entre le chargement de son programme ou l'envoi de commandes directes.

ECRAN DU SYSTEME M20 OLIVETTI LORS DE SA MISE SOUS TENSION

Les six premières lignes donnent à l'utilisateur des informations sur la configuration du système.





La capacité totale de la mémoire est de 192 Kbytes (Koctets).

La mémoire disponible pour l'utilisateur est de 108392 octets.

La zone mémoire réservée aux programmes en Basic est de 54220 octets.

L'écran peut afficher 8 couleurs différentes.

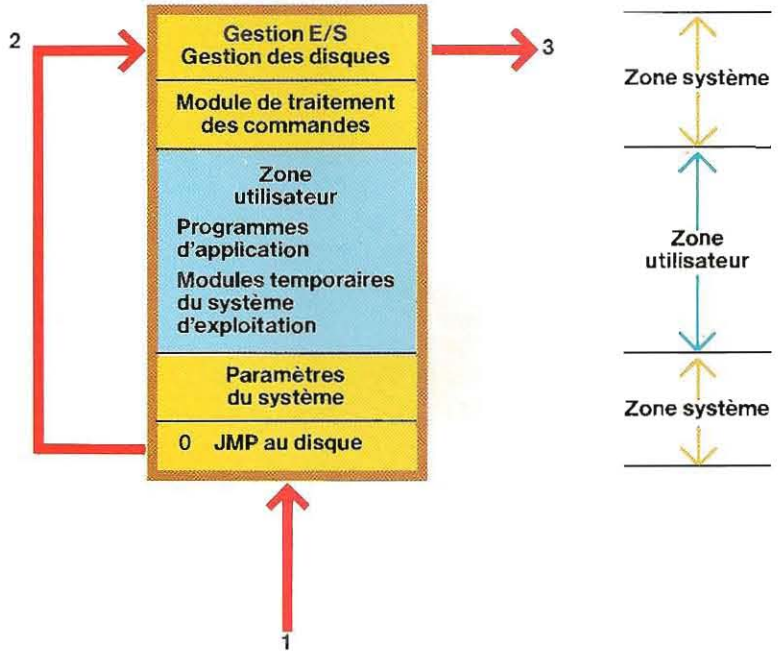
Une seule unité de disque est disponible «en ligne» (connectée). Cela signifie qu'une seule unité est opérationnelle mais la machine peut néanmoins être équipée d'un second lecteur. C'est le cas du M20. Mais, au moment où la photo a été prise, l'une des deux (numéro 1) unités était ouverte.

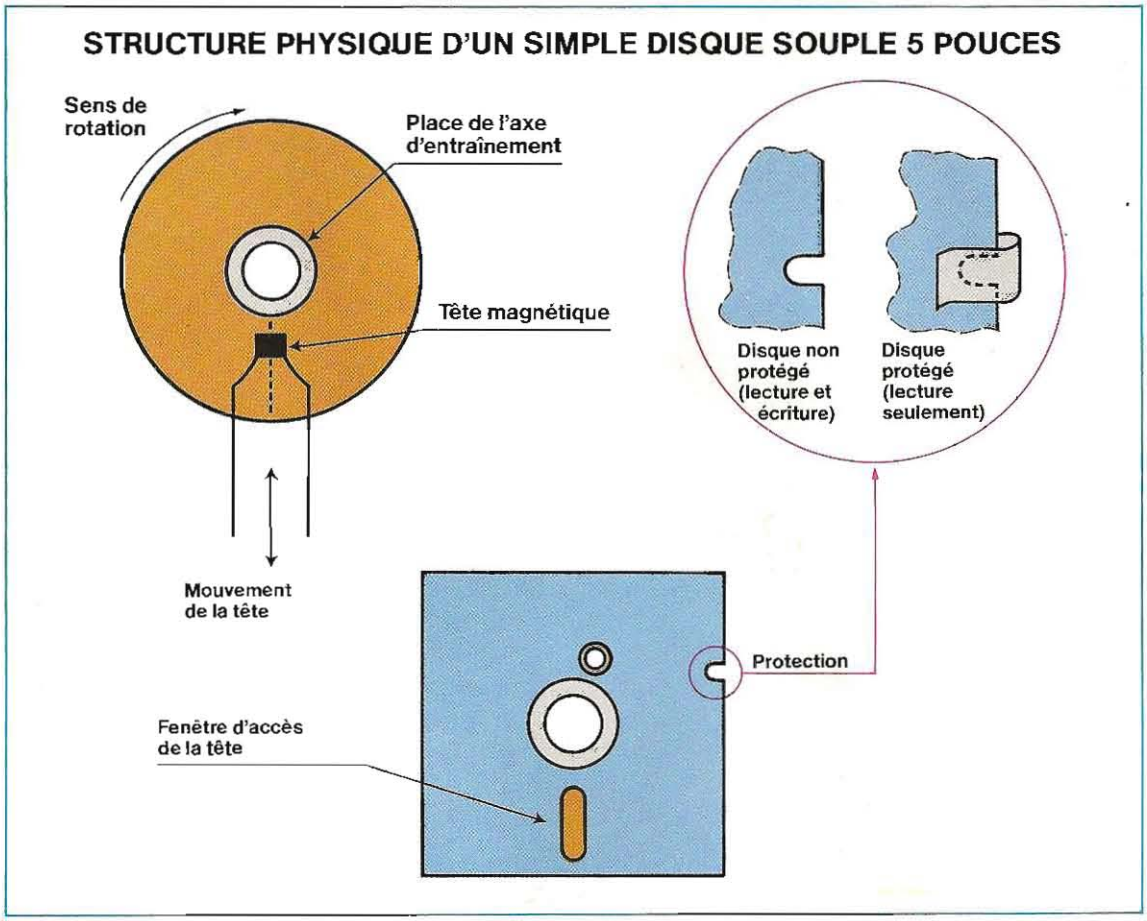
Les deux lignes suivantes indiquent le nom du système d'exploitation (PCOS), sa version ainsi que sa date de création.

Le «prompt» («système prêt») apparaît sur la dernière ligne à côté du curseur. Les ordres transmis au système d'exploitation viendront s'écrire à partir de cette position.

Le module du système d'exploitation chargé de l'interprétation des commandes identifie cette instruction qui demande le passage en Basic. Il charge l'interpréteur en mémoire (en général, le Basic est seulement interprété dans un premier temps ; ce n'est qu'ensuite qu'il est éventuellement compilé) et se prépare à recevoir les lignes du programme.
Si l'on veut exécuter des opérations de base, on entre leur nom symbolique, éventuellement accompagné des paramètres nécessaires.
Après avoir analysé l'instruction, le module de traitement des commandes exécute l'opération demandée.
Les commandes et, donc, les modules correspondants du système d'exploitation, peuvent être résidents ou temporaires.
Les modules résidents, présents en permanence dans la mémoire dès le chargement du système d'exploitation, sont exécutables immédiatement.
Les modules temporaires ne sont chargés que sur demande, et contiennent des commandes propres à l'utilisateur, et stockées dans la partie de la mémoire appelée zone utilisateur. C'est également dans la zone utilisateur que sont chargés les programmes d'application avant leur exécution.
Le schéma ci-dessous montre un exemple de répartition de l'espace mémoire (memory map).
Le premier emplacement de mémoire est indiqué par le symbole 0, et contient une instruction de saut au module de gestion du disque (JMP est l'abréviation de jump, signifiant saut en anglais).
La mise sous tension remettant à 0 le compteur d'instructions (ou compteur ordinal), l'instruction de saut située à l'emplacement 0 (étape **1**) est la première exécutée. Elle entraîne l'exécution du module de chargement du système (gestion des disques, étape **2**) ; la machine attend ensuite les instructions de l'opérateur (étape **3**).
On peut également voir, sur le schéma ci-dessous, illustrant l'espace mémoire, une zone réservée aux paramètres du système (pointeurs, adresses, etc.).

EXEMPLE DE REPARTITION DE L'ESPACE MEMOIRE





La gestion des fichiers sur disque souple. Avant de poursuivre l'étude des opérations exécutées par un système d'exploitation, il nous faut revenir de façon plus détaillée sur la structure physique et logique des disquettes (floppy disks).

Les disquettes sont constituées d'un support souple recouvert d'une fine couche de substance magnétique. L'état magnétisé correspond à la valeur 1 et l'état non magnétisé à la valeur 0.

Lecture et écriture sont effectuées par une tête magnétique qui se déplace le long d'un rayon de la disquette tandis que la rotation de celle-ci permet à tous les points de sa surface de se trouver tour à tour au-dessous de la tête.

Les disquettes sont protégées par une enveloppe percée d'une fenêtre, permettant l'accès de la tête de lecture (voir le schéma ci-dessus).

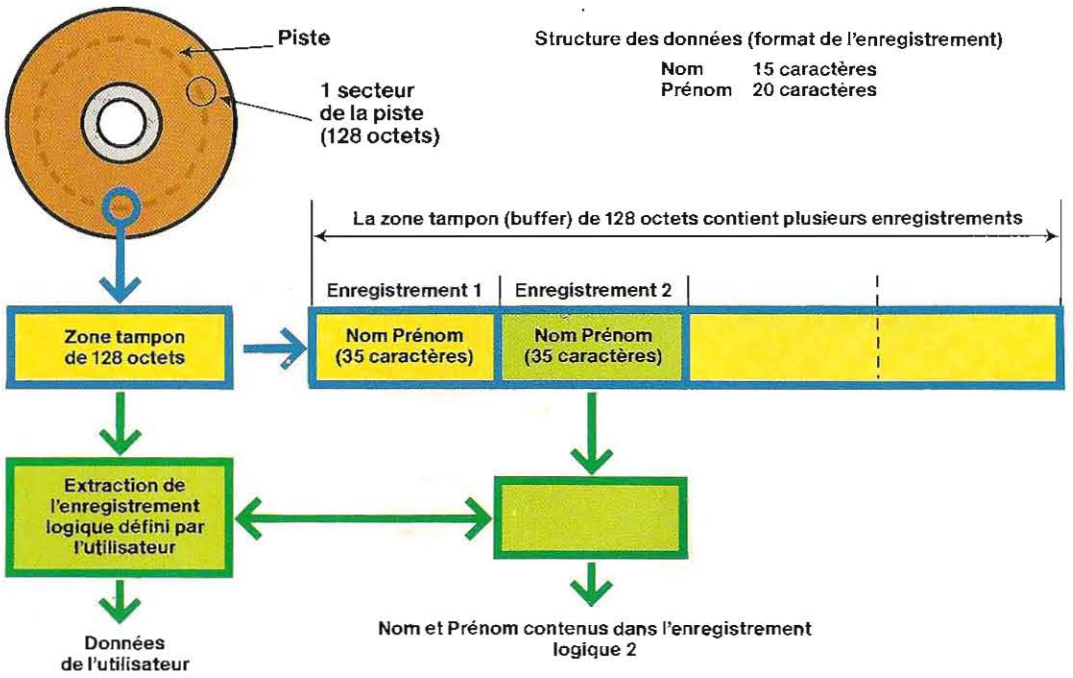
Une encoche de protection est découpée dans cette enveloppe : lorsqu'elle est libre, on peut lire mais non écrire ; si, au contraire, elle est occultée par une étiquette adhésive, l'écriture devient possible (le système de protection fonctionne en sens inverse sur certains matériels).

Les disquettes peuvent être enregistrées sur un ou deux côtés (ce sont alors des disquettes simple ou double face) et leurs zones de magnétisation sont plus ou moins rapprochées, ce qui entraîne la distinction entre disquettes à double densité (souvent désignées par DD) ou disquettes à simple densité (SD). Ce sont, bien sûr, les disquettes double face-double densité qui contiennent le plus d'informations (octets).

Un disque neuf ne peut pas être utilisé tel quel. Il faut d'abord qu'il soit « formaté » par le système d'exploitation. Le **formatage** consiste à diviser le disque en pistes, elles-mêmes divisées en secteurs. On crée ainsi une « map » des espaces disponibles, qui permet d'adresser chaque zone physique de la disquette, en indiquant la piste et le secteur où elle se trouve.

Le secteur est la plus petite unité de mémoire de masse adressable. La lecture et l'écriture s'effectuent donc par blocs dont la longueur est au moins égale à celle d'un secteur.
Un secteur contient, en général, 128 ou 256 octets et les transferts entre la disquette et la mémoire se font toujours par blocs de cette longueur.
C'est seulement ensuite que seront extraits

MECANISME DE LECTURE D'UN ENREGISTREMENT LOGIQUE



Le système prélève sur le disque un secteur entier dont il extrait ensuite les données (enregistrement logique).

les caractères réellement nécessaires au programme d'application (voir le schéma ci-dessus).
La structure d'une disquette, c'est-à-dire le nombre de pistes qu'elle contient, le nombre de secteurs par piste et le nombre d'octets par secteur, dépend à la fois de son type (diamètre, densité, simple ou double face) et du formatage effectué par le système. Voici un exemple de caractéristiques assez courantes :

- 2 faces désignées l'une par 0 et l'autre par 1 (double face) ;
- 35 pistes par face ;

- 16 secteurs par piste ;
- 128 octets par secteur (1D).

La numérotation des secteurs n'est, généralement, pas séquentielle mais intercalée (interleaved). Ainsi, une piste contiendra les secteurs 1, 3, etc., tandis que les secteurs pairs se trouveront sur la piste suivante. Cette technique d'enregistrement est la plus fréquemment utilisée pour réduire les temps d'accès. Cependant, sur certains systèmes, on peut modifier l'ordre de numérotation à volonté, et sauter, par exemple, cinq numéros entre deux enregistrements (secteur 1, secteur 7, etc.). Dans certaines conditions, les temps d'accès s'en trouvent encore réduits.
Après son formatage, la disquette est prête à être utilisée comme mémoire de masse. L'enregistrement des données (ou des programmes) se fait dans des zones contiguës, constituant un fichier identifié par un nom propre. Pour relire des données, le système, connaissant ce nom, se chargera de retrouver l'emplacement physique du fichier et de transférer son contenu en mémoire.
Pour cela, les informations nécessaires, c'est-à-dire le nom, l'adresse et le type de contenu (données, programme) du fichier, ainsi que son emplacement physique (piste et secteur de début, piste et secteur de fin), sont écrites lors de la création des fichiers dans une zone réservée du disque, appelée directory (répertoire).
Le système d'exploitation peut alors positionner la tête de lecture à l'endroit correspondant au début du fichier. Il y trouvera un ensemble d'informations n'appartenant pas aux données proprement dites et constituant le FDB (File Descriptor Block), ou bloc de description de fichier.
En voici le détail pour ce qui concerne le système PCOS* (Olivetti) :

- longueur actuelle du fichier ;
- nombre d'extensions ;
- mot de passe (password) ;
- drapeau (flag) de protection ;
- table des extensions ;
- pointeur vers une extension éventuelle de la table des extensions.

Cette liste diffère légèrement d'un système à l'autre, mais les informations les plus importantes sont toujours présentes. Ainsi, pour le système CP/M*, les principales rubriques sont, dans l'ordre :

- le nom du fichier (8 caractères) ;
- le type de fichier (3 caractères) ;
- le nombre d'extensions (extent) ;
- la taille des extensions ;
- la liste des emplacements sur le disque ;
- le numéro du prochain enregistrement en lecture et en écriture.

Un fichier n'occupe pas constamment le même espace : que ce soit un programme ou une suite de données, sa longueur augmente à chaque nouvelle saisie. Les ajouts ne sont pas placés physiquement à la suite des enregistrements préexistants dans la mesure où d'autres fichiers peuvent se trouver intercalés sur le disque.
Il faut donc mémoriser les **extensions de données** dans une zone spéciale qui contiendra une table de pointeurs (adresses) vers ces zones (ou extents). Cette table redonne une continuité logique au fichier, physiquement « éclaté », dispersé en plusieurs endroits du disque.
La lecture de la table des extensions, le positionnement à l'adresse trouvée et le chargement des données en mémoire sont pris en charge par le système d'exploitation.
Lorsqu'un fichier comporte de nombreuses extensions, sa lecture demande plus d'opérations que celle d'un fichier peu morcelé, et il est donc parfois avantageux de le recopier séquentiellement sur une nouvelle disquette.
Cette opération peut être effectuée lors d'une sauvegarde du fichier sur un disque fraîchement formaté.

On peut conserver sur disque plusieurs types de fichiers dont les principaux sont :

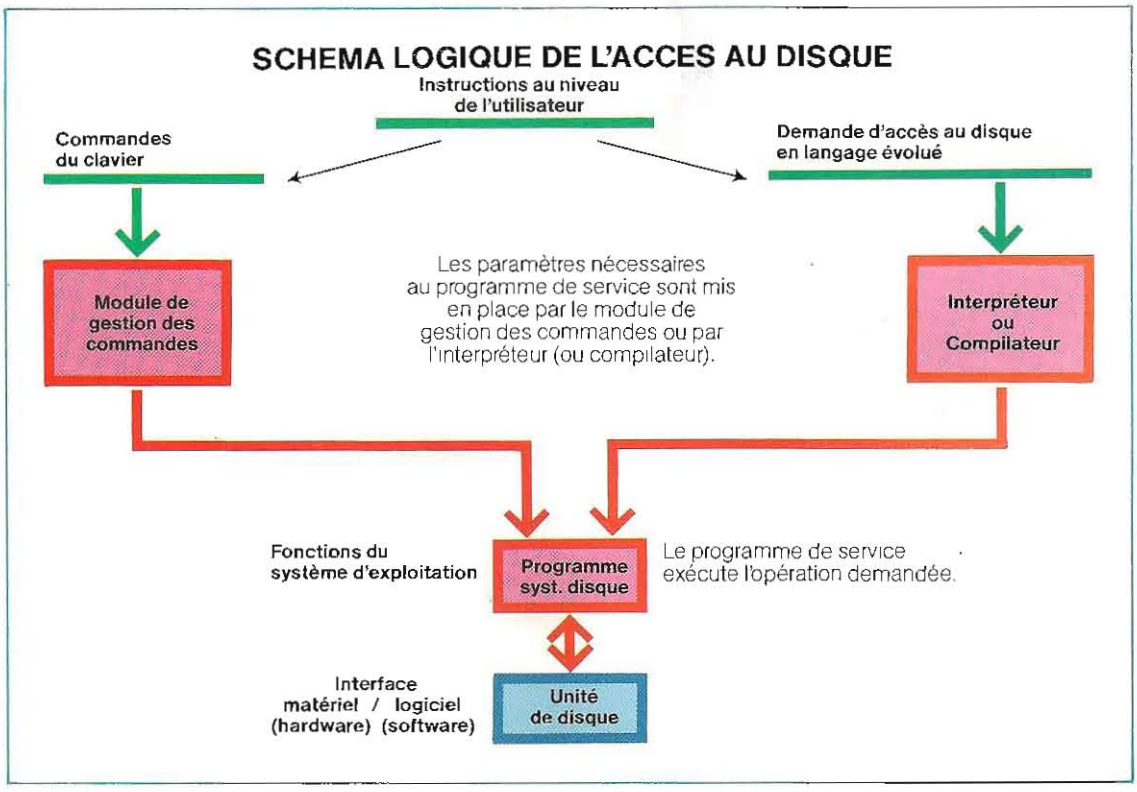
1 / les programmes sources en Assembleur (ASM) ;
2 / les programmes en langage machine avec représentation hexadécimale (HEX) ;
3 / les programmes sources en Basic (BAS), ou Fortran (FOR), entre autres ;

*Les dénominations CP/M et PCOS sont des marques déposées.

4 / les programmes compilés (COM) ou programmes images ;
5 / les fichiers des modules de service ;
6 / les fichiers de données.

Nous avons indiqué pour les fichiers portant les numéros 1 à 4 les sigles de trois caractères servant généralement à les identifier. Les symboles associés aux deux autres types de fichiers varient souvent d'un système d'exploitation à l'autre.
Pour identifier un fichier, il faut indiquer son nom et son type. Par exemple, pour charger un programme en Basic sauvegardé dans un fichier appelé ESSAI, il faut indiquer ESSAI.BAS. La première partie de cette référence donne le nom du fichier (qui, en général, ne doit pas dépasser 8 caractères) et la seconde partie son type (ici programme source en Basic).
Sur certains systèmes d'exploitation, on peut remplacer l'une de ces deux informations par un symbole générique (*, par exemple). La recherche ne se limite plus alors à un nom ou à un type, et tous les fichiers qui répondent à la condition conservée sont pris en considération. Ainsi, la référence *.BAS désigne tous les programmes en Basic quel que soit leur nom. Cela est utile dans de nombreux cas. Par exemple, l'emploi du symbole générique (*.BAS) à la place du nom permet de recopier tous les programmes en Basic, en une seule fois et sans avoir à préciser leur nom. De même, si l'on écrit *.*, tous les fichiers seront concernés, indépendamment de leur nom et de leur type.

Les opérations de gestion des fichiers du système d'exploitation. Le système d'exploitation contient des modules ou programmes utilitaires (ou de service) destinés à l'exécution des principales opérations de gestion des fichiers. Chaque système d'exploitation possède son symbolisme et l'on peut donc retrouver une même opération sous des noms différents. Les dénominations que nous adopterons par la suite sont, soit les plus répandues, soit les plus proches des instructions Basic.
Pendant toute la lecture de ce paragraphe, il faudra garder présent à l'esprit le fait que l'utilisateur n'a pas directement accès aux programmes de service du système (à moins qu'il n'emploie un langage Assembleur), mais



Le dessin animé et l'ordinateur

Il y a peu de temps encore, la réalisation d'un film d'animation à partir de dessins demandait un travail colossal et prenait beaucoup de temps. C'est ainsi qu'il fallut 250 000 dessins et 300 000 heures de travail pour achever « Animal Farm », de Halas et Batchelor, l'un des premiers dessins animés en long métrage.

Aujourd'hui, l'ordinateur permet de diminuer considérablement le temps d'exécution du dessin animé, mais il met également à la disposition des dessinateurs des techniques particulières suggérant la création d'effets nouveaux.
Une caméra prenant 24 images à la seconde, le dessin animé sera obtenu en filmant 24 dessins légèrement différents l'un de l'autre, afin que leur projection à vitesse constante reproduise le mouvement dans son déroulement naturel.

Les techniques d'animation ont connu relativement peu de changements pendant longtemps, et le travail se faisait alors presque complètement à la main. Aujourd'hui, on cherche à rendre moins laborieuse l'élaboration de chaque groupe d'images en perfectionnant des procédés de traitement électronique.
En effet, même si on continue à utiliser la caméra de façon traditionnelle et à photographier des dessins faits à la main et non des images vidéo produites par ordinateur, on peut néanmoins accélérer le processus par des commandes électroniques. On installe alors directement la caméra sur une planche horizontale. On peut déplacer verticalement l'appareil, et horizontalement la table elle-même.
L'opérateur respecte, en général, un enchaînement précis des mouvements de la table ou de l'appareil selon qu'il veut obtenir des effets panoramiques, modifier la taille des personnages ou donner l'impression du mouvement.

On peut ainsi photographier les différentes positions d'un personnage en train de courir, tandis que l'on fait reculer, par à-coups calculés avec précision, le décor du fond, y gagnant ainsi en réalisme.

Lorsque ce déplacement du matériel est commandé par un ordinateur, l'enchaînement des mouvements est beaucoup plus précis, le travail bien moins long et le résultat plus vivant.
Naturellement, chaque image doit tout d'abord être traduite dans le langage de l'ordinateur avant de pouvoir être traitée.

Pour coder une image en langage informatique, il faut la découper en une mosaïque de minuscules carrés de couleur. On attribue à chaque petit carré, ou pixel (élément d'image), un numéro correspondant à sa valeur tonale. Ce numéro varie et donne, par exemple, 0 (0000 en binaire) pour le blanc et 8 (1000 en binaire) pour le noir ; les numéros intermédiaires permettent de traiter les nuances. Pour la mémoire d'un ordinateur, un dessin n'est donc qu'une suite ordonnée de chiffres. Le grand avantage de ce système est qu'on n'a plus besoin de redessiner chaque fois un personnage dans une position donnée : l'ordinateur est programmé pour projeter n'importe quelle image ou ensemble d'images stockés dans sa mémoire. Tous les dessins peuvent donc être réutilisés.

Certains ordinateurs (les systèmes DAO, dessin assisté par l'ordinateur), et notamment le Flair, laissent au dessinateur la liberté de travailler comme il en a l'habitude. Le Flair ne possède pas de clavier et il suffit, pour créer des images, de dessiner avec un stylo spécial sur une table graphique reliée à l'ordinateur. Aucune trace ne reste sur cette table, mais le dessin apparaît sur l'écran de contrôle qui sert de guide au dessinateur.

Un unique instrument remplace ainsi tout un ensemble de pinceaux ou de crayons : il peut tracer des traits fins ou des aplats. Pour changer de style, il suffira d'appuyer sur des touches de sélection spéciales disposées sur le bord de la table.

La table graphique est conçue pour communiquer à l'ordinateur les points touchés par le dessinateur : le stylet émet deux ondes électromagnétiques traversant la surface de la table en largeur et en hauteur. L'ordinateur peut ainsi calculer les coordonnées de chaque point en fonction du temps mis par les ondes pour atteindre les bords de la table.

Pour rendre les nuances de couleur, le dessinateur dispose d'une sorte de pinceau électronique commandé par une touche. Ce pinceau « vaporise » sur l'écran la couleur choisie dans la palette électronique dès que le stylet touche la table graphique. Plus le contact du stylet sur la table se prolonge, plus la couleur est intense. Ces nuances servent généralement à créer l'illusion d'une image à trois dimensions.

La réalisation de ces effets spéciaux demande au dessinateur une habileté comparable à celle requise par le travail traditionnel sur papier avec des pinceaux et de la peinture. Certaines opérations sont toutefois simplifiées. Pour tracer, par exemple, une ligne droite, il suffit d'indiquer ses extrémités à l'ordinateur et de le laisser relier lui-même les deux points.
Le dessinateur peut également demander à sa machine de faire apparaître sur l'écran des figures géométriques telles que cercles ou ellipses. Il suffit pour cela de fournir les points clés (centre et rayon par exemple) à la machine qui se charge de tout le reste.
Pour choisir sa couleur, le dessinateur demande l'affichage de la palette, tableau composé de carrés de couleur et appuie son stylet sur la couleur désirée pour informer l'ordinateur de son choix.

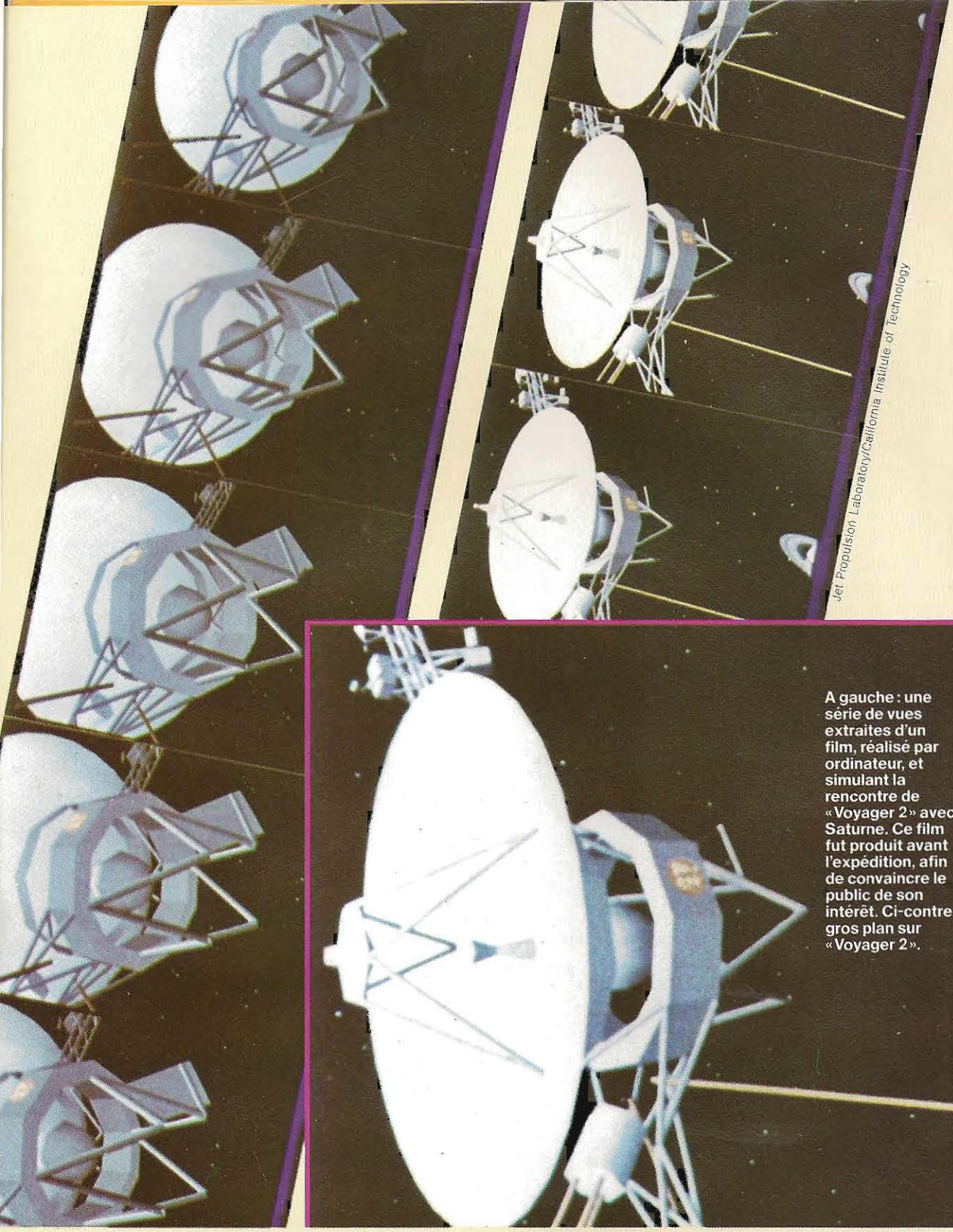
Les informations visuelles sont enregistrées dans la mémoire de l'ordinateur sous forme numérique. L'écran du Flair comporte 576 lignes de 768 points. L'ordinateur emmagasine les informations caractéristiques de chaque point sous la forme de l'une des 256 combinaisons possibles de signaux de base (rouge, vert et bleu) qui, sur l'écran, sont perçues comme une seule couleur. Lorsque, après avoir choisi une couleur, le dessinateur déplace son stylo, le code couleur correspondant est transmis à chaque point de contact.

Un seul microprocesseur, de type 8085A, contrôle l'entrée des informations et la synthèse d'image. Il est relié à la table graphique et à une unité de mémoire de disque. Ainsi,

Table graphique connectée à un micro-ordinateur. Le dessin, exécuté au crayon électronique, apparaît sur l'écran du poste de télévision.



B. Coleman/Marka



Jet Propulsion Laboratory/California Institute of Technology

A gauche : une série de vues extraites d'un film, réalisé par ordinateur, et simulant la rencontre de « Voyager 2 » avec Saturne. Ce film fut produit avant l'expédition, afin de convaincre le public de son intérêt. Ci-contre : gros plan sur « Voyager 2 ».

une fois la conception achevée, on peut conserver le dessin sur disque.
Conscients du fait que le cinéma est l'un des meilleurs supports de diffusion d'un grand nombre d'informations, la NASA et le Jet Propulsion Laboratory ont produit des films d'animation réalisés par ordinateur, et retraçant les trajectoires des sondes envoyées dans l'es-

pace par des fusées.
Ce travail requiert toute la puissance de calcul d'un gros ordinateur, mais l'exactitude mathématique des résultats permet de simuler la modification des paramètres de vol. Les films sont très nets et la précision des ombres, de la perspective et des structures rend ces films extrêmement réalistes et convaincants.

Pour créer des images plus vivantes, l'équipe d'informaticiens du Jet Propulsion Laboratory s'est attachée à la collaboration d'artistes comme David Miller, lequel s'est rapidement passionné pour les possibilités de création offertes par l'outil informatique. Cet artiste travaille souvent la nuit lorsqu'il peut disposer de l'ordinateur pour son usage personnel. Ses créations insolites sont du plus grand intérêt et présentent des images surprenantes pour qui est habitué aux techniques traditionnelles de l'art graphique.

Les chaînes de télévision anglaise et allemande, BBC et ARD, utilisent le système ICON, comprenant un mini-ordinateur programmé dans un langage appelé RTL (en français LTR, langage temps réel) et déjà à l'œuvre dans de nombreuses autres applications, du télex à la comptabilité. Il reçoit les données en entrée et génère les informations qui servent à produire une image télévisée.

L'ICON permet d'obtenir une définition de 1 024 points pour chacune des 574 lignes de l'image. Chaque point apparaît dans la gamme des 64 couleurs de la palette électronique de base. De plus, l'ordinateur peut faire varier chaque teinte de cette palette. La couleur numéro 6, par exemple, pourra passer, en un éclair, du vert océan au bleu indigo. Cette modification des couleurs est un moyen simple et efficace de faire ressortir les détails flous d'une image par contraste avec le fond.

Dans le système ICON, une image n'est pas enregistrée sous forme de nombres indiquant la valeur de chaque point. La plupart des images comportant de larges zones de couleur unie, il est donc plus simple de les définir en indiquant une couleur et le nombre de points correspondants (X et Y). Cette méthode, appelée codification à longueur de séquence, permet de réduire considérablement le nombre des données nécessaires à la reproduction d'une image.

Il faut, en effet, trente fois moins d'informations pour définir une image de cette façon que pour la définir point par point. L'ordinateur n'a plus qu'à calculer les contours des zones où il y a des changements de couleur pour appliquer la teinte qui convient. Lorsque les images sont lues au rythme de défilement de la télévision, les compteurs numériques électroniques passent en revue les points de couleur et changent les numéros de teinte au moment voulu.

Enfin, la palette traduit les numéros en une série de tensions électriques qui correspondent à la combinaison de rouge, de vert et de bleu dont est composée chaque couleur de l'écran.

Pour les longs métrages ou les films spatiaux de la NASA, qui sont des animations très complexes, les images sont produites une à une, puis enregistrées sur une pellicule ou une bande vidéo.

Les données numériques sont communiquées à une mémoire à accès sélectif, selon un rythme relativement lent, puis lues au rythme très rapide des images télévisées.

L'ordinateur peut donc synthétiser l'image à une vitesse choisie et chaque image terminée peut être transférée sur une pellicule. Les photogrammes sont ensuite projetés à la vitesse de 24 images à la seconde de façon à donner l'illusion du mouvement. Cette technique, qui permet de produire des images très complexes, a été utilisée pour réaliser des films comme « la Guerre des étoiles » et « Star Trek ».

Certains ordinateurs offrent de nombreuses possibilités d'effets spéciaux très variés. Ils permettent ainsi de rendre l'effet de volume (trois dimensions) des personnages, des objets, des constructions. Ils se prêtent au rendu de la texture des matériaux (le lisse et le rugueux).
Enfin, ils sont capables de reproduire les effets de lumière et les différents éléments climatiques comme la pluie, le brouillard ou la neige. C'est dire le grand apport du traitement électronique au dessin d'animation.

doit passer par l'intermédiaire de l'interpréteur (ou du compilateur) ou des commandes du système d'exploitation. La logique de ces deux moyens d'accès est illustrée schématiquement page 299. Voici les principaux modules des systèmes d'exploitation :

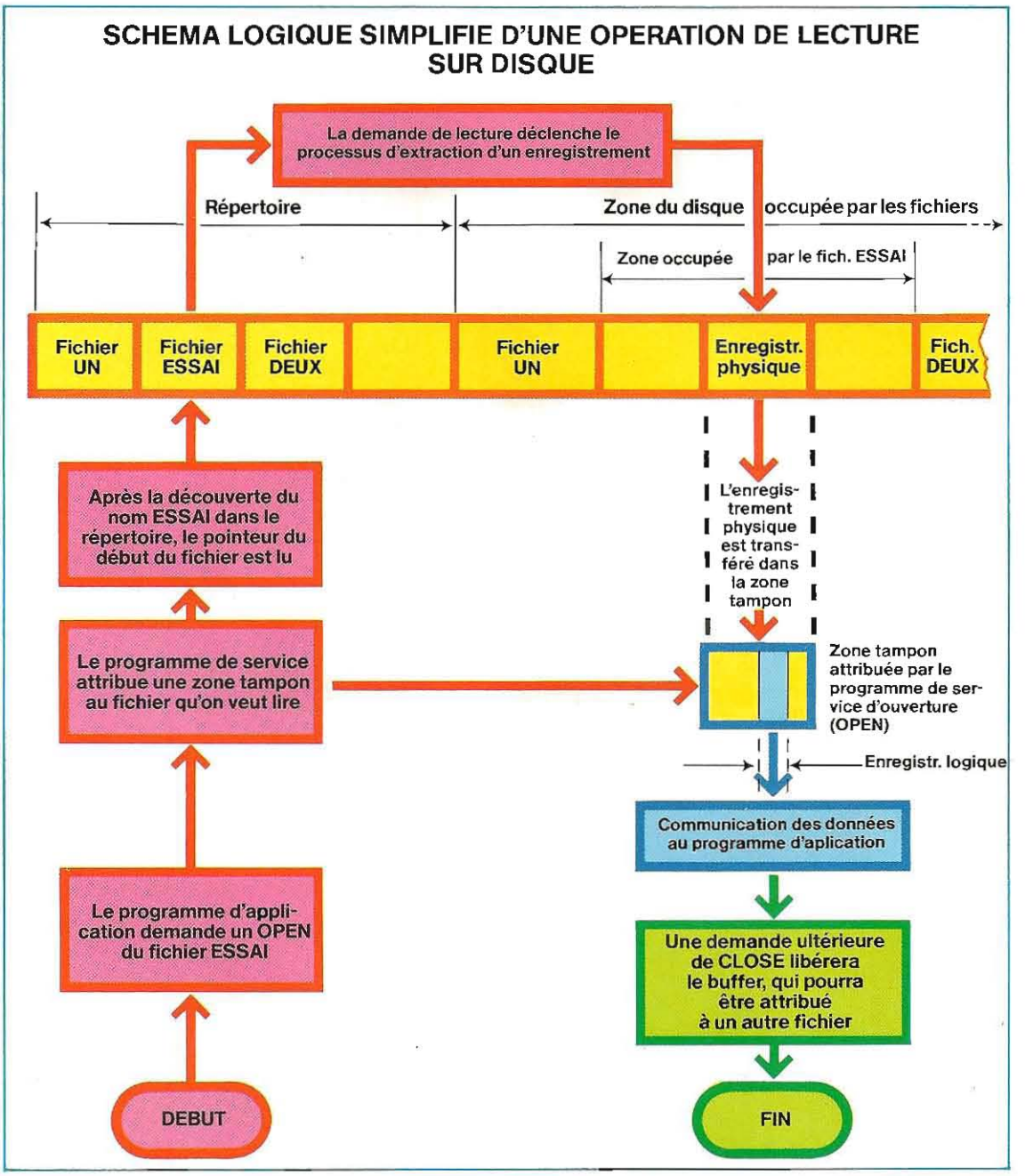
SEARCH : recherche d'un fichier dont on connaît le nom.

OPEN : ouverture d'un fichier. Cette opération permet d'accéder au contenu du fichier. Elle précède presque toujours la lecture ou l'écriture des informations.

CLOSE : fermeture du fichier. Cette opération rend impossible l'accès aux données. Il faudra ouvrir à nouveau le fichier pour manipuler les enregistrements qu'il contient.

RENAME : changement de nom d'un fichier.

SCHEMA LOGIQUE SIMPLIFIE D'UNE OPERATION DE LECTURE SUR DISQUE



READ : lecture d'un enregistrement.
WRITE : écriture d'un enregistrement.
SELECT : sélection de l'une des unités de disques (en anglais, un lecteur de disques s'appelle disk drive)

Les programmes d'application en langage évolué (Basic, par exemple) utilisent des instructions entraînant l'exécution de ces programmes de service chargés de gérer le matériel et d'effectuer les transferts de données demandés par l'application.
L'OPEN et le CLOSE représentent deux fonctions particulièrement importantes du système d'exploitation. Deux instructions Basic portent exactement le même nom.
Comme nous l'avons vu, la lecture et l'écriture impliquent le transfert de secteurs entiers de données, même lorsque l'enregistrement logique est plus petit. Une zone de mémoire tampon (buffer) accueille donc temporairement les données lors des échanges avec le disque ; l'enregistrement logique recherché est ensuite extrait du buffer (enregistrement physique). La fonction OPEN sert essentiellement à attribuer une zone tampon au fichier. Le nombre de buffers, et donc de fichiers utilisables en même temps n'étant pas illimité (il est souvent réduit à 3 ou 4), il faudra, si l'on désire utiliser plus de fichiers, libérer des zones tampons par un CLOSE avant d'ouvrir de nouveaux fichiers. Le schéma de la page 304 illustre le processus de lecture d'un enregistrement ; l'écriture se déroulerait de façon analogue. Les programmes de service sont appelés par des instructions en langage évolué mais aussi par de nombreuses commandes du système d'exploitation. On distingue les commandes temporaires (ou utilisateur) et les commandes résidentes (utilisables seulement lorsqu'on travaille en « mode système »). Dès que l'on a chargé le Basic, on n'y a plus accès directement. Cependant, d'autres instructions permettent d'exécuter les mêmes opérations en Basic ainsi qu'avec tous les autres langages évolués. Le schéma de la page 306 illustre cette répartition des commandes et les opérations correspondantes.
Voici maintenant l'explication des opérations exécutées par ces commandes et des exemples d'utilisation. Si le symbolisme d'écriture de ces commandes dépend du système d'exploitation, les opérations exécutées sont les mêmes sur tous les systèmes. Nous mentionnerons pour chaque commande l'abréviation la plus fréquemment rencontrée.

1 / Destruction d'un fichier (commande = **ERA**)
Le code ERA (abréviation du verbe anglais erase, effacer) est le symbole qui déclenche l'opération de destruction (effaçage) d'un fichier sur le disque.
Il faut fournir les indications suivantes :
- le disque sur lequel se trouve le fichier (soit A ou B, soit 0 ou 1, selon le système) ;
- le nom du fichier ;
- le type du fichier (Basic, compilé, données, etc.).

Ainsi, pour effacer le fichier ESSAI qui se trouve sur le disque A et contient un programme en Basic, l'instruction du système CP/M est la suivante :

ERA ESSAI.BAS

La commande ne provoque ici que la destruction des fichiers situés sur la disquette A (qui est souvent « prise par défaut »). Si le fichier s'était trouvé sur la seconde unité (désignée par B), il aurait fallu préciser :

ERA B:ESSAI.BAS

Avec le système d'exploitation PCOS, l'instruction correspondante est :

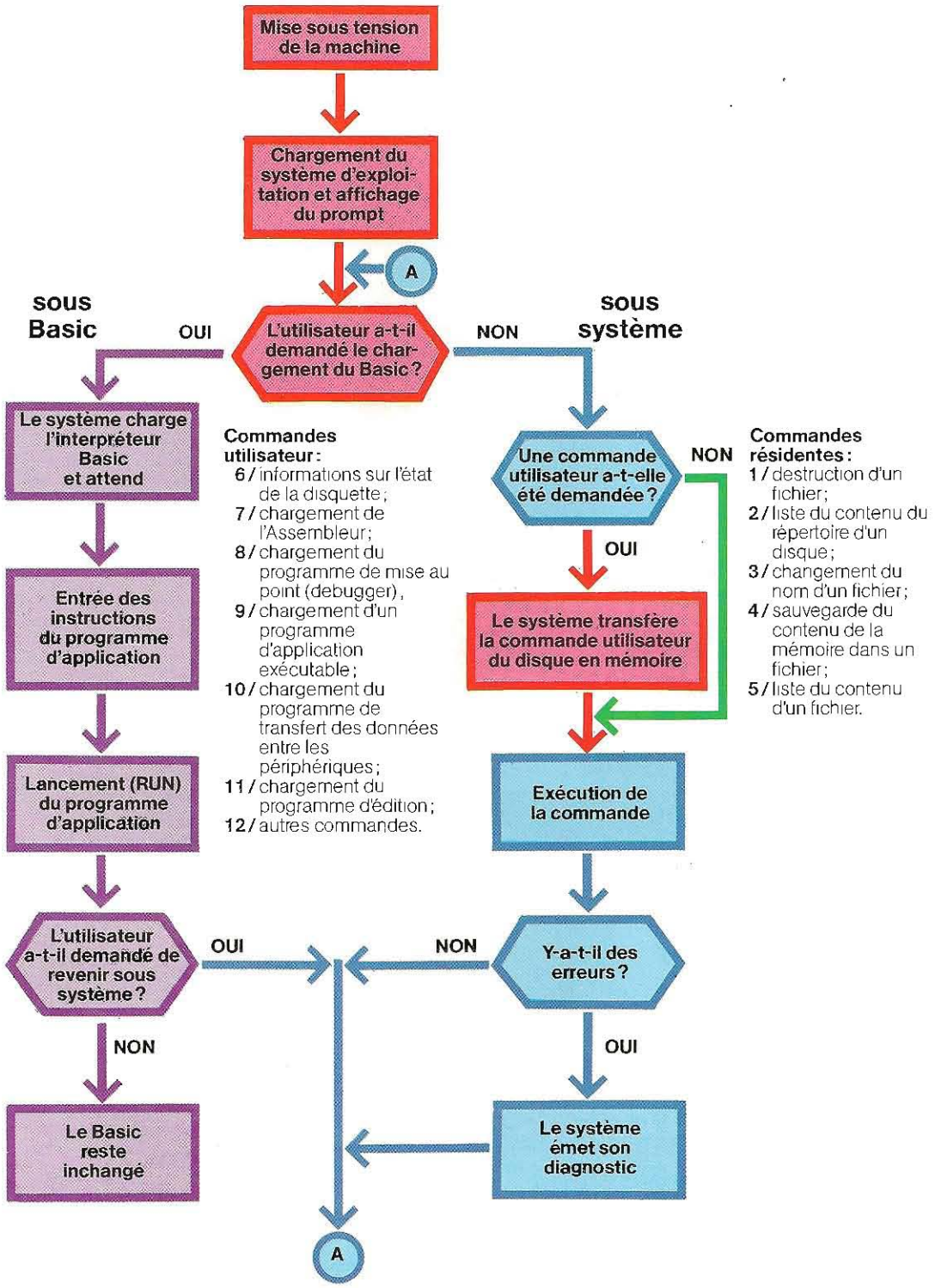
KILL "1:ESSAI"

En effet, sur ce système, les unités de disque sont désignées par 0 et 1 (et non par A et B) et on n'a pas besoin de préciser le type du fichier. La forme générale de ce type d'instruction est donc :

Code opération	Unité de disque	Nom du fichier (et type)
ERA	B:	ESSAI. BAS
KILL	1:	ESSAI
etc.		

2 / Liste du contenu du répertoire d'un disque (commande = **DIR**)
Cette instruction fournit le plus souvent à l'écran la liste des fichiers d'une disquette.

SCHEMA LOGIQUE DES FONCTIONS EXECUTEES PAR UN SYSTEME D'EXPLOITATION



La répartition des commandes résidentes et utilisateur citée en exemple est valable pour le CP/M.

Cette liste indique le type des fichiers, ainsi que, dans certains cas, leur longueur (exprimée en secteurs ou en octets). Le format habituel de cette instruction est :

DIR A : pour lister le répertoire du disque A
DIR B : pour lister le répertoire du disque B

3 / Changement du nom d'un fichier (commande = **REN**)
Cette commande (de l'anglais rename) change le nom du fichier. Son format est :

REN B:NOUVEAU.BAS = ESSAI.BAS

Le fichier ESSAI qui se trouve sur la disquette B, s'appelle désormais NOUVEAU.
Voici la structure générale de cette instruction :

Code commande	Unité de disque	Nouveau nom et type	Ancien nom et type
REN	B:	NOUVEAU.BAS =	ESSAI.BAS

4 / Enregistrement d'un programme sur un disque (commande = **SAVE**)
Cette commande est utilisée en Basic pour mémoriser sur disque un programme enregistré en mémoire centrale. Nous reviendrons plus loin sur les différentes formes de cette instruction dont la formulation est :

SAVE "A:ESSAI"

Par cette commande, le programme qui se trouve en mémoire va être enregistré sur la disquette A, sous le nom ESSAI.

5 / Liste du contenu d'un fichier (commande = **TYPE**)

Le fichier ESSAI.BAS apparaîtra avec la commande suivante, à condition qu'il ait été enregistré sous la forme d'un fichier ASCII (SAVE « A : ESSAI », A).

Code commande	Unité de disque	Nom du fichier (et type)
TYPE	B:	ESSAI.BAS

L'instruction TYPE B:ESSAI.BAS commande l'affichage du contenu du fichier ESSAI.

6 / Etat du disque (commande = **STAT**)
Elle donne des informations sur la disquette de l'unité indiquée : remplissage des fichiers, espace disponible, etc. Il suffit, en général, de préciser le nom ou le numéro de l'unité de disque et l'instruction se présente comme suit :

Code commande	Unité de disque
STAT	B:

Sur certains systèmes, dont le CP/M, cette commande fournit également des informations sur les périphériques.

7 / Chargement de l'Assembleur (commande = **ASM**)
Sur les systèmes où il est possible d'utiliser l'Assembleur, cette commande charge en mémoire un programme écrit en Assembleur et le compile (traduction en langage machine hexadécimal).
La syntaxe de cette instruction est :

Code commande	Unité de disque	Nom du programme
ASM	B:	TEST

Par cette commande, le programme TEST qui se trouve sur la disquette B va être chargé en mémoire (si toutefois il est bien écrit en Assembleur). Après sa compilation, il est enregistré sur une disquette sous le même nom mais son type est désormais HEX (langage machine) : TEST.HEX.

8 / Programme de mise au point (debugger ; commande = **DDT**)
C'est une fonction très complexe qui permet de tester et même de corriger des programmes d'application pendant leur exécution.
On peut ainsi ajouter des instructions, en supprimer, examiner et modifier le contenu des zones mémoire, etc. Toutes les instructions doivent être données en langage machine (hexadécimal) puisque cette mise au point a lieu pendant l'exécution et qu'un programme en langage évolué n'est exécutable qu'après sa compilation.
L'utilisation de cette commande demande une grande expérience.

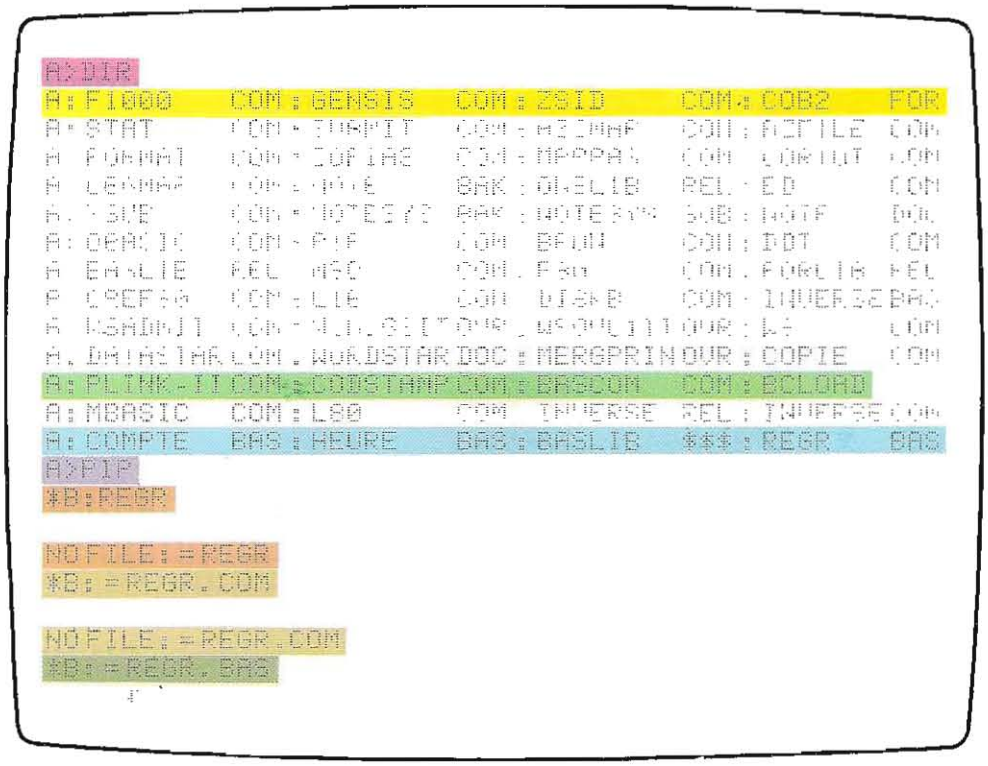
9 / Chargement d'un programme exécutable (commande = **LOAD**)
Un programme n'est exécutable qu'une fois

CONTENU DU REPERTOIRE D'UN DISQUE ET COPIE D'UN FICHIER

Le nom et le type de tous les fichiers présents sur disque sont mémorisés dans le répertoire.



La commande DIR entraîne l'affichage sur l'écran du contenu du répertoire. Ici, on a choisi le disque de l'unité A.



Chaque ligne affiche la référence de l'unité de disque choisie suivie du nom et du type de quatre fichiers.

Cette ligne nous informe que les fichiers F1000 (compilé), GENSIS (compilé), ZSID (compilé) et COB2 (programme source en Fortran) se trouvent sur la disquette A.

Le type du fichier BCLOAD n'est pas indiqué : il s'agit d'un fichier de données.

Les fichiers COMPTE, HEURE et REGR contiennent des programmes source en Basic.

Nous voyons également, sur cet écran, la commande PIP utilisée pour transférer un fichier d'une unité à une autre (unité de disque ou autre périphérique : imprimante, par exemple).

Lorsqu'on tape la commande PIP, le système répond par le symbole *. L'opérateur doit alors fournir des indications supplémentaires.

L'opérateur demande la copie du fichier REGR sur le disque de l'unité B, mais il oublie de préciser son type. Le système ne trouve pas le fichier (répertorié en BAS, comme on le voit sur l'écran) et prévient l'opérateur.

L'opérateur demande alors la copie du fichier REGR. COM. Ce fichier n'existe pas et l'ordinateur repose la question,

La demande de l'opérateur est enfin correcte et la commande est exécutée.

EMPLOI DE LA COMMANDE STAT

La commande STAT provoque l'affichage à l'écran de l'état d'un fichier.

```
A: DATASTAR COM : WORDSTAR COM : MBSPRIN COM : INVERSE COM
A: FLINK-II COM : CODSTAMP COM : SASCOM   COM : BCLOAD
A: MBASIC   COM : LSO      COM : INVERSE REL : INVERSE COM
A: COMPTE   BAS : HEURE    BAS : BASLIB  $$$ : REGR    BAS
A>STAT COMPTE

File not Found
A>STAT COMPTE.BAS

Recs  Bytes  Ext Acc
   5    4k      1 R/WA: COMPTE.BAS
Bytes Remaining On A: 476k

A>STAT
A: R/W, Space: 476k

A>STAT BCLOAD

Recs  Bytes  Ext Acc
   1    4k      1 R/WA: BCLOAD
Bytes Remaining On A: 476k
```

Après l'affichage du répertoire, on a demandé l'état du fichier COMPTE sans préciser son type. Le système le considère par défaut comme un fichier de données, et ne le trouve donc pas.

La commande est reformulée correctement. Le système présente donc l'état du fichier COMPTE : celui-ci contient cinq enregistrements (logiques) ; sa longueur totale est de 4096 octets (4 Ko) ; il ne comporte pas d'extension et peut être lu ou écrit. (R et W sont les initiales de read et write). La dernière ligne indique le nombre de K octets encore disponibles sur le disque.

On aurait également pu obtenir cette information par l'ordre STAT, sans indiquer aucun nom de fichier.

On a demandé l'état du fichier de données BCLOAD. Dans ce cas, on n'a pas besoin de préciser le type du fichier (voir le répertoire), ce fichier comporte une extension

compilé ou interprété : le programme source (écrit, par exemple, en Basic) devient un programme objet en langage machine.
Le résultat de la compilation est un fichier sur disque qui contient la codification hexadécimale des instructions du programme source.
Pour travailler sur le programme compilé, il faut transférer les codes du disque vers la mémoire, par la commande LOAD dont la syntaxe est :

Code commande LOAD	Unité de disque B:	Nom du programme ESSAI

La commande LOAD B:ESSAI extrait la version compilée du programme (de type HEX) et la transfère dans la zone utilisateur de la mémoire. Le programme est alors prêt à être exécuté.
Notons toutefois que, sur les systèmes sans compilateur, l'ordre LOAD effectue le chargement des programmes écrits en Basic.
Si l'on veut charger un programme source en Basic sur un système muni d'un compilateur, il faut d'abord passer sous Basic, puis utiliser l'instruction LOAD. L'opération se déroule alors de la même manière mais c'est le programme en Basic qui est chargé.
La commande LOAD a donc plusieurs rôles :
- sur les systèmes dépourvus de compilateur, elle charge un programme source.
- sur les systèmes munis d'un compilateur, elle charge le programme compilé et le programme source sous Basic.

10 / Echange de données entre les périphériques (commande = **PIP**)
Elle permet le transfert de données ou de programmes d'un périphérique à un autre. On s'en sert essentiellement pour échanger les programmes entre les deux unités de disque.
Voici sa syntaxe :

```
> PIP
* B: = ESSAI.BAS
```

L'instruction PIP sollicite le programme d'échange. Le système répondant par le symbole * attend la suite de l'instruction indiquant entre quels périphériques l'échange doit avoir lieu.
Dans notre exemple, le programme en Basic ESSAI va être transféré sous le même nom de la disquette A à la disquette B.
Cette commande s'applique à tous les types de fichiers prévus par le système d'exploitation, quel que soit leur nom :

```
> PIP
* B: = *.*
```

Cette instruction transfère tous les fichiers de A sur la disquette B (le symbole * qui remplace le nom et le type a une valeur générique : « quel que soit »).

```
> PIP
*B: = *.BAS
```

Cette fois, seul le type est précisé (BAS). Tous les fichiers en Basic seront donc transférés.

```
> PIP
* B: = *.
```

Le type est laissé en blanc : il s'agit de fichiers de données de A, à recopier sur la disquette B.

11 / Editeur (commande = **ED,** ou **EDIT**)
Cette fonction permet de lire, de charger en mémoire, puis de modifier le contenu d'un fichier.
Ce programme complexe effectue de nombreuses opérations de correction telles que :
- le remplacement, la suppression et l'insertion de caractères ou de lignes ;
- la recherche de mots dans un texte ou dans un programme ;
- le remplacement de groupes de caractères.

Le travail terminé, un nouveau fichier contenant les données corrigées est créé, ou remplace l'ancien fichier.

12 / Autres commandes
Il existe de nombreuses autres commandes importantes propres à chaque système d'exploitation, tels le formatage des disquettes (pour lequel il suffit d'entrer le code de la commande, FORMAT par exemple) et la copie du système d'exploitation sur une disquette de secours.
Nous allons donner en exemple les principales fonctions du système PCOS (M 20), sans faire de distinction entre les programmes temporaires et les programmes résidents.

Test 8



1 / Enumérez les principaux éléments du système d'exploitation d'un micro-ordinateur.

2 / Laquelle de ces affirmations est vraie ?
a) la mémoire d'un ordinateur est à l'entière disposition de l'utilisateur ;
b) les programmes utilisateurs sont chargés dans la zone système de la mémoire ;
c) le système d'exploitation peut être chargé tout ou partie dans la mémoire utilisateur et tout ou partie dans la zone système.

3 / Que représentent les tables des domaines (extent) des fichiers ?
a) des tables qui contiennent les longueurs des fichiers ;
b) des listes des fichiers présents sur un disque ;
c) des tables qui contiennent les adresses des différentes parties d'un fichier.

4 / A quoi sert le répertoire (directory) ?
a) à donner des informations sur tous les fichiers ;
b) à contenir les différents éléments du système d'exploitation ;
c) à contenir les programmes d'application.

5 / En quoi consiste le formatage des disquettes ?
a) à effacer leur contenu ;
b) à les diviser en pistes et en secteurs ;
c) à vérifier leur bon état.

Les solutions du test se trouvent page 313.

- formatage de disquettes ;
- copie du système d'exploitation ;
- détermination d'un mot de passe (empêchant les personnes non autorisées d'accéder aux données) ;
- création d'un fichier ;
- détermination d'un mot de passe (password) pour un fichier particulier (seuls sont protégés les fichiers possédant un mot de passe ; tout le monde a accès au reste du disque).
- commandes spéciales modifiant le fonctionnement des périphériques (format d'impression par exemple).

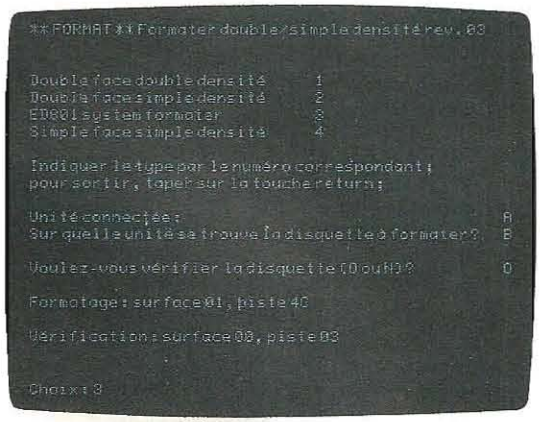
Les opérations de passage sous Basic

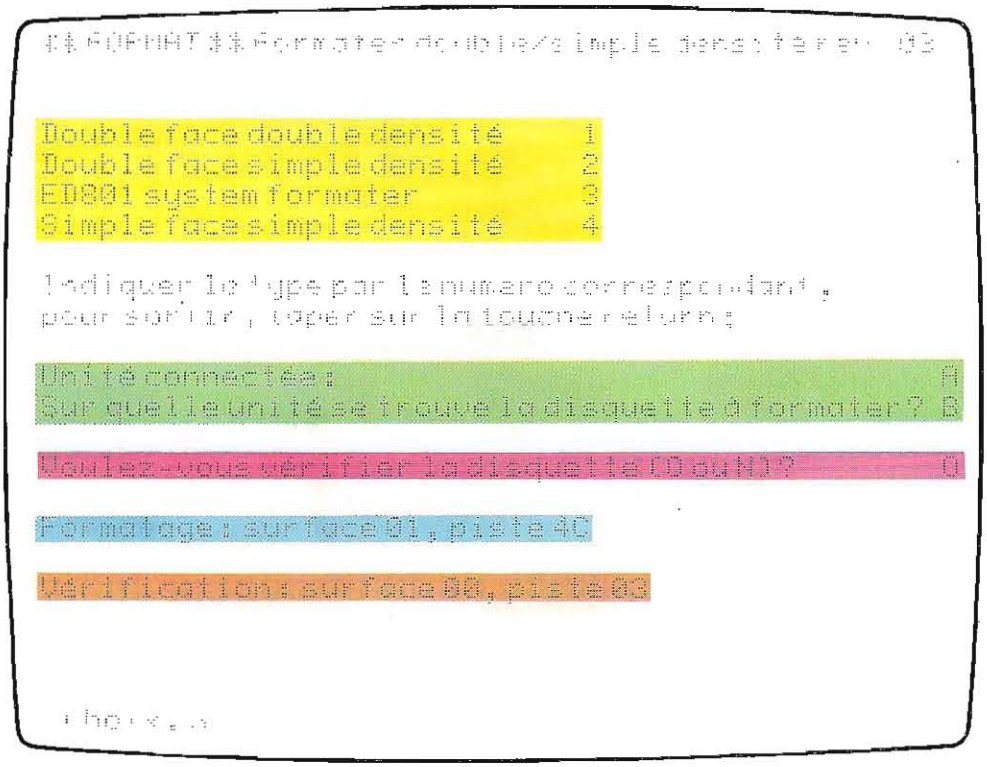
Une fois branché, l'ordinateur est prêt à fonctionner dès que le système d'exploitation est chargé. Toutefois, il est sous le contrôle du système et ne peut accepter les instructions en Basic. Il faut donc « passer sous Basic » avant d'entrer un programme.

Cette instruction, qui diffère selon les systèmes d'exploitation (le CP/M utilise l'instruction MBASIC, tandis que celle du PCOS est BA) provoque le chargement de l'interpréteur du Basic et on peut alors commencer la programmation.
Une fois le travail terminé, il suffira, pour revenir sous système, d'entrer une autre instruction, souvent symbolisée par SYSTEM. Toutefois, ces opérations ne sont pas nécessaires sur les systèmes qui ne peuvent travailler qu'en Basic.
Pour les autres langages compilés, comme le Fortran, la procédure est différente. On commence par écrire les instructions comme s'il s'agissait de simples données, en utilisant, par exemple, le programme EDIT, et on demande ensuite sa compilation. Le programme est traduit en langage machine et peut alors être exécuté.
On suit la même procédure pour compiler un programme en Basic si le système d'exploitation possède un compilateur.

EMPLOI DE L'ORDRE FORMAT EN CP/M

L'ordre FORMAT, qui peut prendre des noms différents selon les machines, INIT par exemple, permet de préparer un disque neuf à son utilisation ultérieure. Sur un disque déjà formaté cette commande entraîne la perte des données qu'il contenait éventuellement. Le système d'exploitation dirige, pas à pas, la procédure de formatage afin de réduire les risques d'erreurs. Voici ce qui apparaît à l'écran lorsqu'on transmet l'ordre FORMAT au système CP/M d'une machine ayant de nombreuses possibilités.





Le système indique à l'opérateur le choix entre quatre possibilités.

Le système prévient qu'il travaille pour l'instant sur l'unité A et demande à l'opérateur sur quelle unité est montée la disquette à formater. La réponse est B.

Le système demande si l'on désire vérifier la disquette. Si la réponse est oui (O), le système contrôlera le bon état de tous les secteurs de la disquette après son formatage.

Cette ligne rend compte de la progression du formatage. (sur face 01, piste 4C).

Cette dernière ligne apparaît une fois le formatage terminé, lorsque la vérification de la disquette a été demandée. Elle indique où en est la vérification. A ce stade, le système est en train de vérifier la piste 3 de la face 00. Le travail s'achèvera à la piste 4C de la face 01.

Solutions du test 8

1 / Les principales fonctions du système d'exploitation sont :
– la gestion des opérations E/S. Tous les programmes qui « interprètent » les instructions d'entrée/sortie et activent les contrôles et les fonctions nécessaires pour piloter les périphériques (disques, imprimantes, etc.) font partie du système ; un module peut être préposé à chaque périphérique ;
– l'interprétation des commandes. Ce module analyse les commandes entrées au clavier (inhérent au système d'exploitation) et contrôle leur exécution.
Ces deux composants du système d'exploitation assument les fonctions de base et doivent toujours être présents (espace mémoire résident, ou noyau résident).
Il existe de nombreuses autres fonctions (liste de l'index, échange de données entre périphériques, etc.) qui ne sont pas sujettes à un usage permanent. Les programmes associés sont stockés sur disque, et chargés en mémoire sur demande.

2 / c) l'explication est inclue dans la réponse à la question 1. La mémoire de la machine est divisée entre la zone système (résidente) et la zone utilisateur. En outre, dans le cas du Basic interprété, l'interpréteur aussi doit être chargé en mémoire. Au contraire, dans le Basic compilé, seul le programme utilisateur est chargé, car, étant déjà traduit (en phase de compilation) en langage machine, il ne nécessite pas d'interpréteur. On voit ici un autre avantage de la compilation : comme l'interpréteur n'est pas chargé, il reste à la disposition de l'utilisateur un espace mémoire plus grand, ce qui n'est pas insignifiant si l'on considère que l'interpréteur peut occuper 12 000 octets (12 Ko).
Considérons, par exemple, un ordinateur ayant 64 Ko de mémoire centrale, les répartitions sont :

Système d'exploitation	16 Ko environ	(Cela dépend de la complexité du système d'exploitation)
Interpréteur	12 Ko	
Zone utilisateur	36 Ko	

En l'absence d'interpréteur, pour le Basic compilé, la zone utilisateur est de 48 Ko.

3 / c) un fichier, qu'il contienne des programmes ou des données, ne conserve pas toujours sa longueur initiale.
Les mises à jour successives du contenu peuvent demander un nombre d'enregistrements supérieur à celui dont le fichier dispose déjà. En ce cas, les nouveaux enregistrements sont écrits sur des emplacements du disque éloignés du fichier auquel ils appartiennent. Pour réunir logiquement les différents morceaux, on utilise la table des extensions (table des domaines) qui contient, pour chaque fichier, la position physique des différentes extensions rajoutées.

4 / a) Le répertoire est une zone du disque gérée par le système d'exploitation, et dans laquelle sont mémorisées les informations du contenu du disque (emplacement, nom et type de fichier, mots de passe, etc.).

5 / Les trois réponses (a, b et c) sont globalement bonnes. Le but principal du formatage est de partager le disque en pistes et secteurs, pour créer une méthode d'accès à l'espace physique. Pendant cette opération, tout le contenu du disque est effacé. En outre, dans beaucoup de systèmes d'exploitation, une fonction de vérification lit et écrit tout le disque, de façon à déceler les éventuels espaces défectueux.

Mais on a l'avantage de pouvoir tester le programme de manière interactive. Les erreurs éventuelles peuvent être corrigées avant compilation. Au contraire, dans les langages sans interpréteur, il faut compiler le programme source après chaque modification.

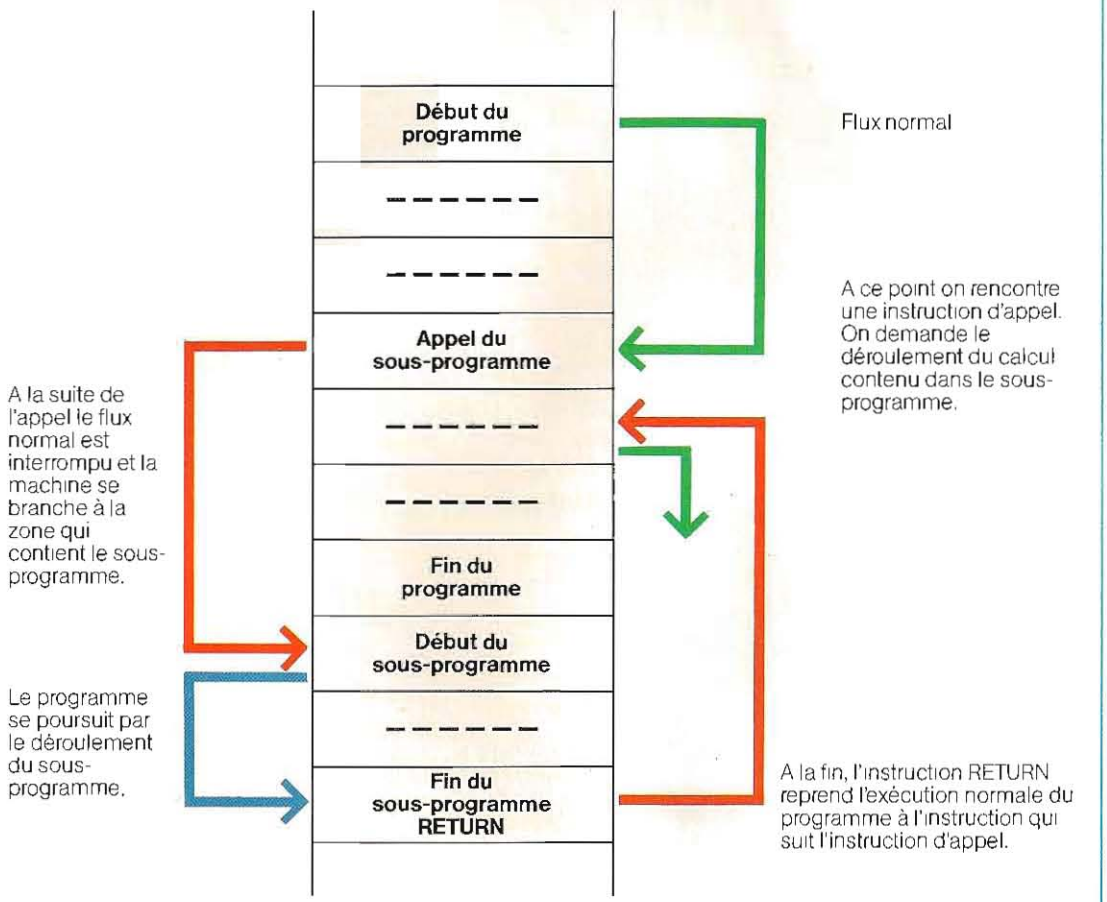
Critères d'évaluation d'un système d'exploitation

Les fonctions principales d'un système d'exploitation constituent les critères auxquels on se réfère pour juger le système.
En règle générale, un système d'exploitation est très complexe et il faut une solide expérience et des connaissances approfondies pour savoir en apprécier la valeur. En effet, un logiciel peut contenir des erreurs.
La phase d'essai est une opération délicate, et même si l'on a recours à des méthodes de contrôle très fiables, on n'est jamais assuré de mettre en évidence tous les défauts éventuels d'un programme. C'est pourquoi on se limite le plus souvent à formuler une estimation globale des fonctions du système d'exploitation. Généralement, le compilateur Basic est prioritaire, le reste étant à évaluer cas par cas, selon les besoins de l'utilisateur.

Subdivision des programmes

Lors de l'écriture de programmes complexes, il faut tenir compte de deux conditions fondamentales : éviter d'écrire plusieurs fois les mêmes séquences d'instructions, et faire tenir le programme entier dans l'étendue maximale de l'espace utilisateur (qui fait partie de la mémoire transitoire ; si le programme est en Basic interprété, une partie considérable de la mémoire transitoire sera occupée par l'interpréteur).
Un programme donné peut, en divers points de son développement, recourir à des calculs identiques, même s'ils s'appliquent à des valeurs numériques différentes. Les écrire à nouveau toutes les fois qu'ils servent constituerait un gaspillage inutile de la mémoire ; il vaudra mieux réutiliser les mêmes calculs, mais il faudra pour cela commencer par préparer d'abord une version unique de chacune des procédures en l'écrivant de façon généralisée. Une partie de programme ainsi structurée prend le nom de **sous-programme** (ou routine). Pour obtenir le déroulement des calculs contenus dans celui-ci, il suffit de l'« appeler » au cours du programme avec les instructions appropriées.
Pour la machine, cet appel détermine un saut, du point (point d'appel) du programme voulant utiliser le sous-programme, au point par lequel commence le sous-programme appelé. A la fin du sous-programme, il faut écrire l'instruction RETURN (retour). En la reconnaissant, le contrôle est redonné à l'instruction qui suit l'appel et se poursuit normalement jusqu'à la fin du programme. D'un point de vue logique, c'est comme si un bloc à part (routine) était inséré aux emplacements nécessaires chaque fois que cela s'impose.
Dans le schéma de la page 315, on a reproduit le déroulement logique d'un appel de sous-programme. Dans les organigrammes, l'appel d'un sous-programme s'indique le plus souvent avec le symbole d'action générale (rectangle), dans lequel est spécifié le sous-programme appelé. Dans les systèmes d'exploitation plus complexes, les sous-programmes peuvent ne pas faire partie du programme.
Ils résident généralement sur disque et sont chargés en mémoire quand il le faut. Pendant le chargement, ils peuvent prendre la place d'un autre sous-programme qui n'est pas utilisé à ce moment. Il se produit alors un recouvrement (en des moments successifs) de différents sous-programmes dans le même espace mémoire, avec une occupation très inférieure à celle qui serait nécessaire si tous les sous-programmes étaient simultanément en mémoire.
On peut trouver des exemples de sous-programmes dans les fonctions mathématiques de tous les langages de haut niveau. En Basic, comme dans les autres langages, on peut développer des calculs complexes (exemple : la racine, les logarithmes, etc.) grâce aux sous-programmes du système, qu'on appelle plus précisément fonctions.
La symbolique des sous-programmes système est différente de celle employée pour les sous-programmes écrits par l'utilisateur, mais le mécanisme est le même. Généralement, pour appeler une fonction système il suffit d'en indiquer le nom (ainsi la racine est appelée SQR), alors que, pour les programmes de l'utilisateur, il faut une instruction spécifique prenant en charge le mécanisme de saut et de retour.

MECANISME D'APPEL D'UN SOUS-PROGRAMME (CONTENU DANS LE PROGRAMME)



Malgré l'analyse correcte du problème en phase de conception et l'utilisation des sous-programmes, on arrive fréquemment à une occupation mémoire excessive. L'obstacle peut être contourné en utilisant deux techniques similaires : l'enchaînement et le lancement d'un programme à partir d'un autre programme.

L'**enchaînement** (CHAIN) a des aspects voisins de la gestion des sous-programmes non résidents et il consiste à charger en mémoire certaines parties du programme seulement quand elles sont nécessaires, en les remplaçant de temps à autre. A la différence des systèmes plus évolués, dans les ordinateurs personnels et les micro-ordinateurs, l'utilisateur doit lui-même gérer tous les transferts.

Le schéma de la page 316 illustre l'exécution d'un enchaînement. Par exemple, le programme utilisateur est divisé en trois parties, respectivement PROG-1, PROG-2, PROG-3. La première partie (PROG-1) constitue le « corps » principal, et réside en permanence en mémoire ; les deux autres parties sont chargées par l'opération d'enchaînement seulement quand elles servent.

La seconde méthode consiste à exécuter un autre programme au moyen d'une instruction de **lancement** (RUN) contenue dans le programme d'appel. La mémoire ne garde pas

trace du programme d'appel, et le nouveau bloc remplace le précédent.

Transfert des paramètres

Pour qu'un programme puisse fonctionner, tous les points du programme doivent « connaître » les variables utilisées. Par exemple, pour écrire un programme qui calcule la superficie d'un cercle, il faut entrer la valeur du rayon. Si ce programme est écrasé par un autre module, la valeur du rayon est perdue, à moins que le programmeur ait transmis la valeur au nouveau module, c'est-à-dire, ait opéré la transfert du paramètre. Ce transfert se déroule de façon différente selon la technique utilisée pour écrire les différentes parties du programme.

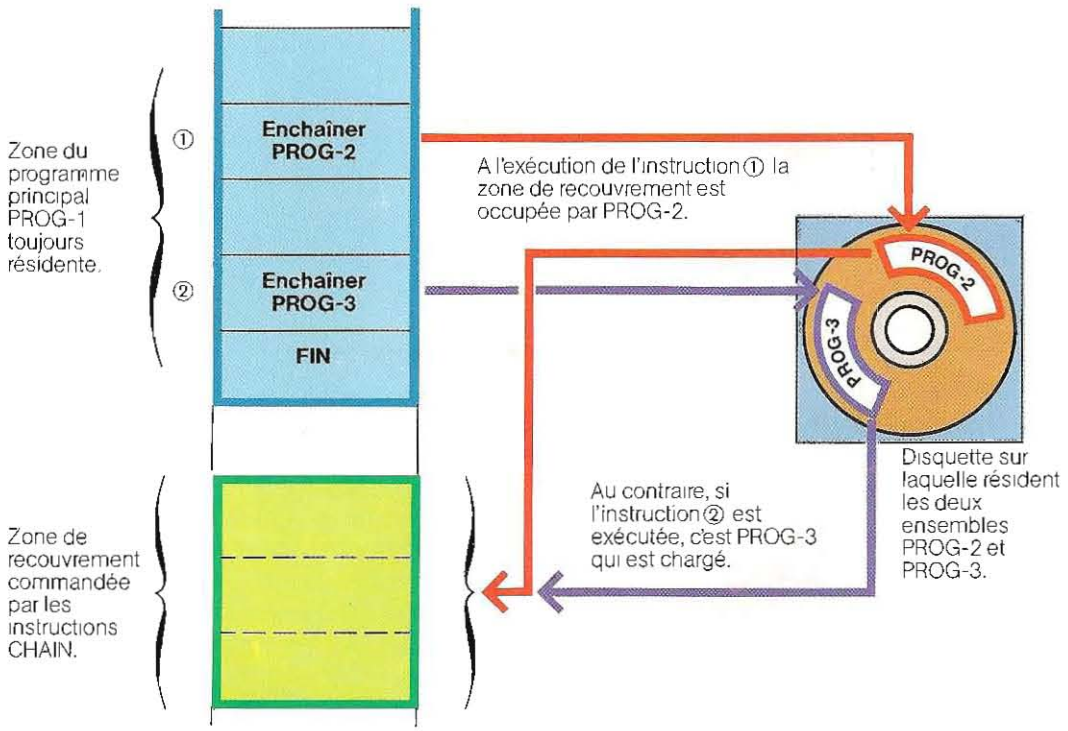
Dans les sous-programmes (à l'exception des cas qui seront mentionnés dans l'explication du langage Basic), il ne faut aucune instruction particulière.

Les sous-programmes (dans la forme standard du Basic) sont parties intégrantes du programme principal, et connaissent donc toutes les valeurs (paramètres).

L'opération CHAIN nécessite une instruction particulière (COMMON) déclarant quelles sont les valeurs utilisées en commun par tous les modules enchaînés. Avec l'instruction RUN, on ne peut passer aucun paramètre en utilisant le disque comme mémoire temporaire. Les valeurs à transférer sont écrites sur disque par le programme appelant (qui contient l'instruction RUN) et sont lues, après chargement, par le programme appelé.

L'étude des instructions Basic de segmentation des programmes sera approfondie dans le chapitre consacré à la syntaxe du langage. Dans les autres langages, même ceux très semblables au Basic, comme le Fortran, les méthodes diffèrent par la forme. Les variantes principales seront exposées dans les chapitres sur les différents langages.

SCHEMA D'EXECUTION D'UN ENCHAINEMENT



La zone de recouvrement est occupée tour à tour par l'un des deux segments (PROG-2 ou PROG-3) selon qu'est exécutée l'instruction ① ou l'instruction ②

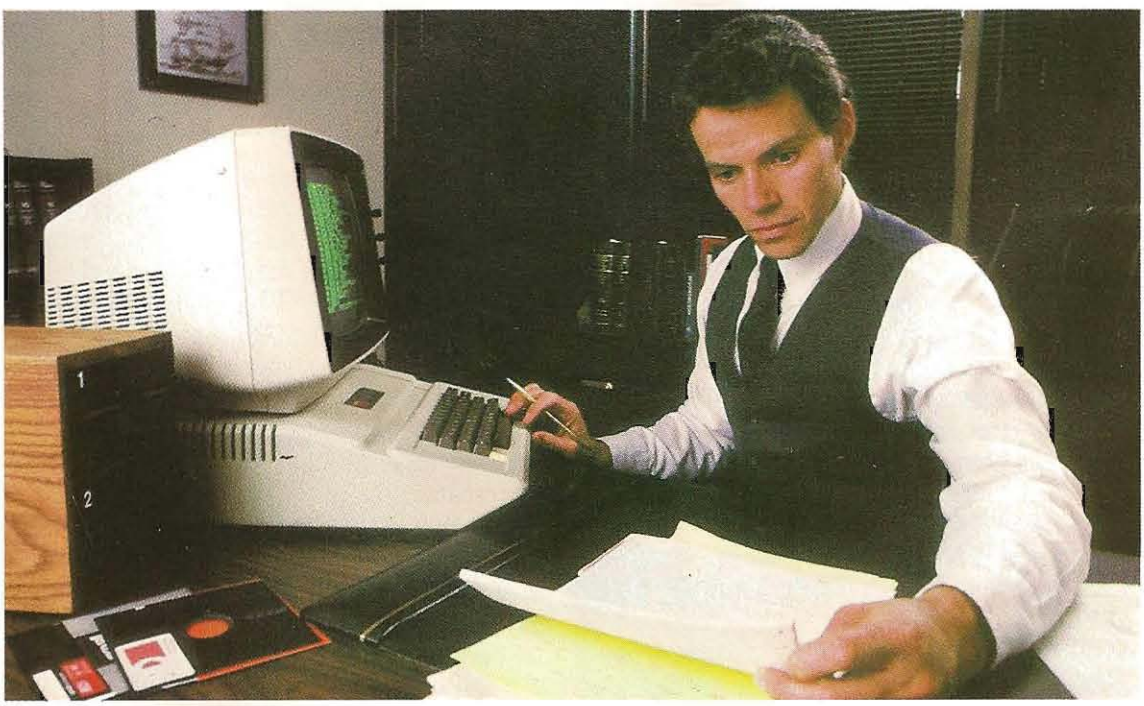
Le langage de programmation Basic

Les seuls moyens pour dialoguer avec les micro-ordinateurs et les ordinateurs personnels sont le clavier, comme dispositif d'entrée, et l'écran (ou l'imprimante), comme dispositif de sortie. Toutes les commandes et les instructions sont entrées au clavier ; la réponse éventuelle de la machine est affichée sur l'écran vidéo ou écrite sur l'imprimante.

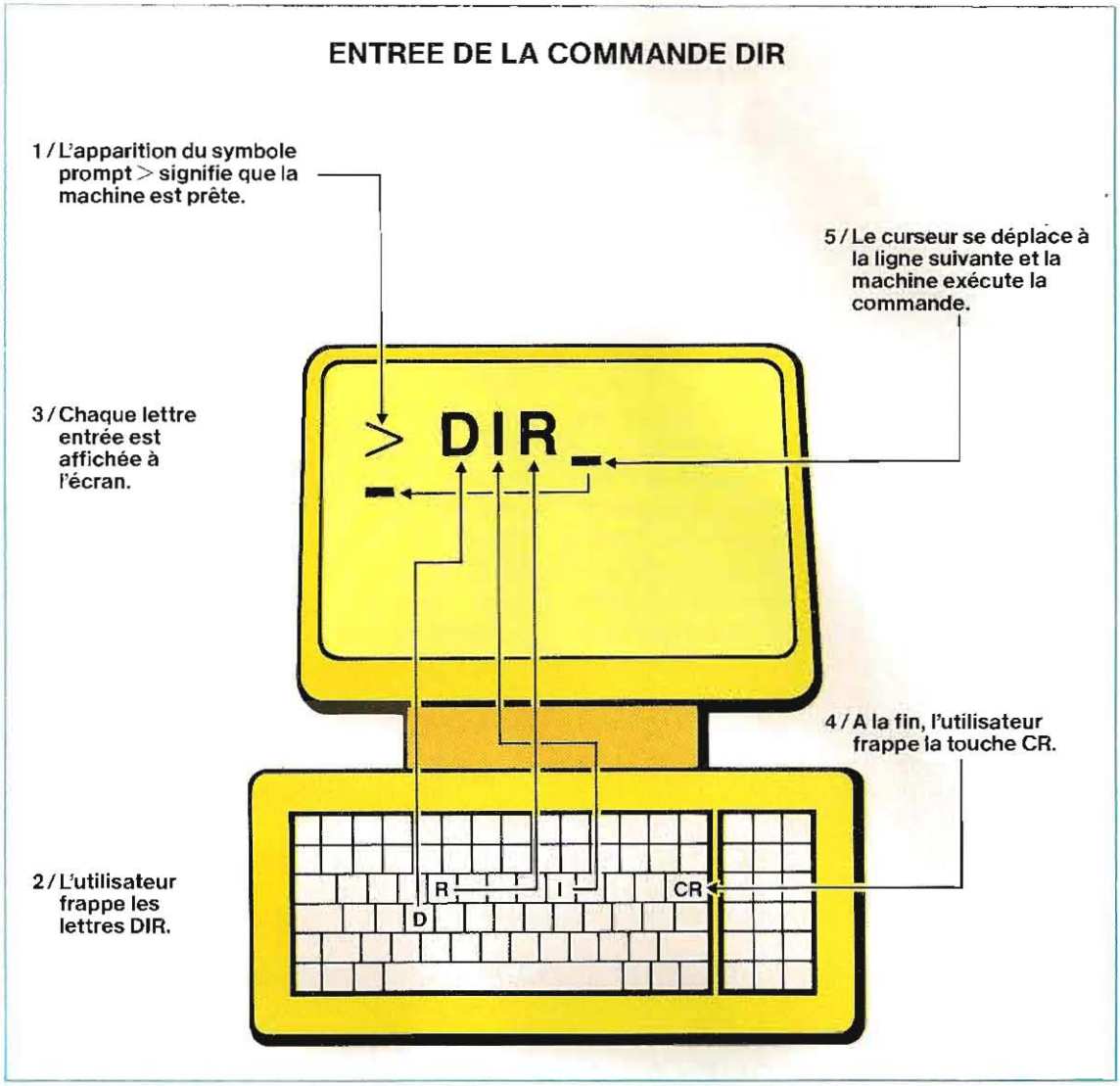
L'entrée au clavier se fait en tapant la commande ou l'instruction comme s'il s'agissait d'une simple machine à écrire. Elle transforme chaque symbole en un signal numérique. A la fin de l'entrée, il faut presser la touche CR (sur certains claviers elle est indiquée par RETURN, ou par le symbole ↵). Supposons que nous sommes en système d'exploitation et que nous voulons obtenir la liste des fichiers contenus sur le disque. L'instruction à entrer est DIR ; dont il faudra frapper (entrer au clavier) les trois lettres DIR et à la fin presser la touche CR (ou RETURN, ou ↵, selon le type de clavier). La touche CR permet d'avertir la machine que l'on a terminé de composer l'instruction et que l'on peut procéder à son exécution.

Toute commande entrée au clavier apparaît à l'écran (ainsi les lettres DIR) : de cette façon, il est possible de vérifier l'exactitude de ce que l'on écrit et d'apporter d'éventuelles corrections. A la fin de la phrase, le curseur s'arrête à côté de la dernière lettre. En pressant la touche CR le curseur se positionne au début de la ligne suivante et l'ordre frappé est exécuté. Cette séquence est illustrée par le schéma de la page 318. Le fonctionnement ainsi exposé se nomme **écho** (l'affichage à l'écran est l'écho de la frappe au clavier), et c'est la façon normale d'opérer. En certaines circonstances (par exemple, à l'entrée d'un mot de passe), on peut supprimer l'écho. Tout ce qui a été frappé au clavier n'est pas restitué à l'écran, et un observateur éventuel ne pour-

L'ordinateur est devenu le matériel de pointe du secteur tertiaire, au niveau le plus élevé.



Marka



rait pas lire ce que l'on écrit. Deux formats type de clavier sont représentés page 319. L'emplacement des lettres diffère selon les claviers, qui sont « AZERTY » en France, ou « QWERTY » (clavier international de langue anglaise). Certains claviers présentent en outre des touches (de fonction) programmées pour exécuter des fonctions particulières. On peut ainsi éviter parfois de frapper toutes les lettres d'une commande donnée.

Sur certains claviers, il existe des touches non utilisées, qui restent libres de toute inscription et sont réservées pour d'autres applications (par exemple PRINT, REPT, CLEAR, EOL, BREAK).

Sur tout clavier, une barre d'espacement (SPACE) a la même fonction que celle d'une machine à écrire, c'est-à-dire l'insertion d'un blanc (dans un ordinateur, un blanc équivaut à un caractère, et, comme tout caractère, occupe une position mémoire). Les lettres des touches sont minuscules. Or les commandes et les instructions d'un programme se composent normalement de lettres majuscules ; c'est pourquoi le clavier prévoit aussi des majuscules ; pour les obtenir, il faut simplement tenir pressée la touche SHIFT lorsqu'on tape une lettre. Pour écrire une longue suite de lettres majuscules, on bloquera le sélecteur des lettres majuscules en poussant la touche LOCK (on débloquera LOCK par simple pression). Certaines touches portent un double symbole : le second symbole sera obtenu par pression préalable de la touche





Daniel Bouquignaud

Le clavier d'un ordinateur ressemble beaucoup à celui d'une simple machine à écrire, mais il comporte plus de touches, certaines d'entre elles étant programmables. Ces touches spéciales sont souvent désignées par la lettre F (Fonction) suivie d'un chiffre. En haut : clavier du M20 (Olivetti). En bas : celui du HP150 (Hewlett-Packard).

SHIFT qui prend donc aussi en charge la sélection des caractères des touches doubles. Il existe enfin deux touches particulières, ESC (escape) et CTRL (contrôle), dont la mise en œuvre provoque l'envoi de caractères particuliers ayant un rôle que nous étudierons par la suite.
En résumé, les touches du clavier d'un micro-ordinateur ou d'un ordinateur personnel sont :

- **alphabétiques et numériques :** pour l'entrée des données ;
- **fonctionnelles :** il s'agit de touches dont la fonction est déterminée par programme ;
- **spéciales** (ESC, CTRL) : elles ont des fonctions particulières.
- Certaines touches **non employées** sont laissées libres pour d'autres applications au clavier ;
- la touche **CR** permet de replacer le curseur au début de la ligne suivante ;
- la touche **SHIFT** sert à obtenir les majuscules, ou les caractères du haut des touches ayant deux symboles ;
- la touche **LOCK** bloque l'état SHIFT.

L'utilisation d'une touche quelconque génère le code correspondant ASCII. A la fin de l'instruction, avec l'entrée du code CR (touche CR), les caractères (codifiés ASCII), sont envoyés à l'unité centrale et traités par le programme résident en cours.
Certains symboles spéciaux (flèches, barres, etc.) sont typiques du Basic alors que dans d'autres systèmes d'exploitation ils ne sont pas reconnus. En ce cas, la machine rend un diagnostic mettant l'erreur en évidence et demandant une nouvelle saisie.

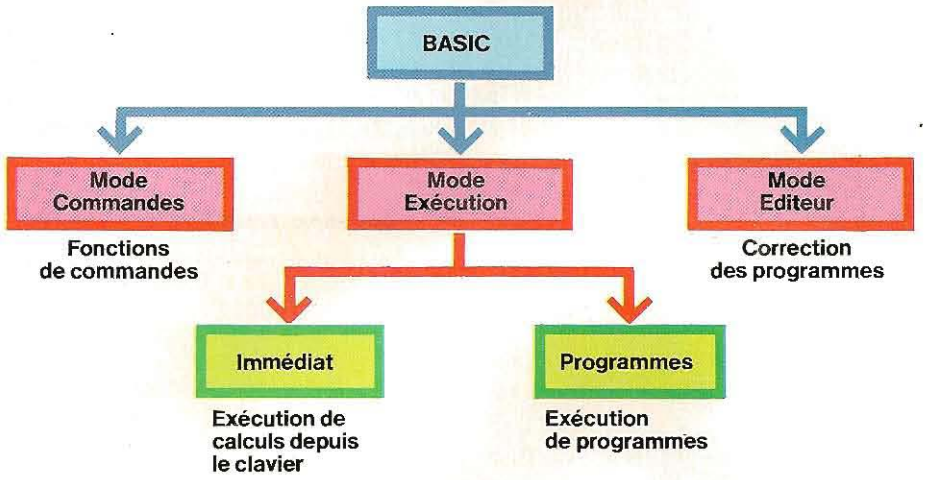
Généralités

Le langage Basic (Beginner's All purpose Symbolic Instruction Code) est né en 1963, développé par Kemmeny et Kurts sous forme interprétée, et doté d'instructions très restreintes. Il était destiné à des utilisateurs inexpérimentés en programmation, et était limité à un usage spécifique.
En peu d'années, il devait pourtant jouir d'un essor considérable, consécutif au développement toujours plus étendu des micro-ordinateurs et des ordinateurs individuels jusqu'à devenir un langage complet, compilable, et constituer une alternative valable aux langages déjà existants (Fortran, Cobol, RPG, etc.). Il a toutefois conservé ses caractéristiques initiales de simplicité et de compréhension immédiate qui en font le langage le plus adapté à ceux qui débutent en informatique. Sa structure particulière et la grande variété des instructions qu'il rassemble permettent aussi au Basic moderne d'être un langage très utilisé, même par des professionnels de l'informatique. L'étude du Basic n'est pas seulement une forme d'introduction à la programmation, elle donne aussi les moyens de développer toutes sortes d'applications, qu'elles soient de type scientifique ou de gestion. En outre, cette étude n'est pas limitée aux ordinateurs individuels et aux micro-ordinateurs, car la majorité des machines peuvent travailler en Basic.
Comme tout langage, le Basic doit aussi correspondre aux normes qui en décrivent les fonctions et les modalités d'exploitation. Les fabricants de logiciel et de matériel doivent au moins respecter les normes correspondant à chaque type de système.
La version du Basic que nous allons étudier est la version standard définie dans le document officiel ANSI - BSRX 3.60 - 1978, émis par l'office américain pour la définition des standards, et augmentée des fonctions propres au Basic 80.
Nous présenterons également quelques particularités adoptées par les fabricants les plus connus, et les caractéristiques du Basic compilé.

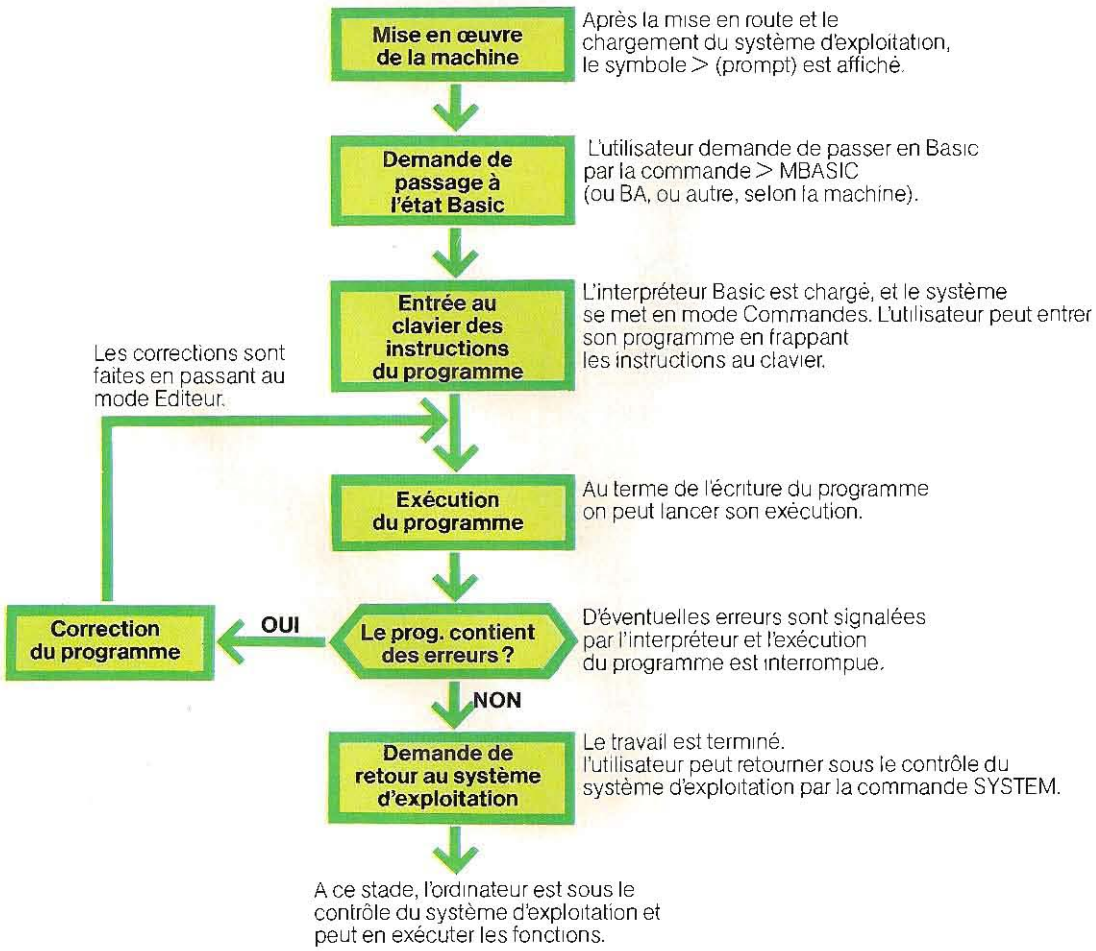
Les modes opératoires

Le Basic interprété possède trois modes opératoires (voir schéma en haut de la page 321) : un mode Commandes, un mode Exécution, et un mode Editeur.
Le mode Commandes permet de saisir toutes les fonctions de commande, telles la demande d'édition du listing d'un programme, les fonctions de gestion des fichiers, l'entrée d'un programme, etc. On a recours au mode Exécution pour lancer les programmes d'application ou exécuter des calculs immédiats. Le mode Editeur sert à la correction des erreurs dans les programmes d'application. Le schéma en bas de la page 321 illustre la procédure d'écriture et de lancement d'un programme d'application.

MODES OPERATOIRES DU BASIC



PROCEDURE COMPLETE D'ECRITURE ET DE LANCEMENT D'UN PROGRAMME D'APPLICATION



Structure des instructions Basic

Un programme Basic est constitué d'une succession de lignes (instructions) qui spécifient la suite logique des actions à accomplir. L'entrée de chaque ligne survient au terme de son écriture en poussant la touche CR, comme pour toute saisie.

Chaque ligne doit être précédée d'un numéro qui la distingue des autres. Ce numéro est l'adresse de l'instruction, et l'exécution du programme (mis à part l'intervention d'instructions de saut) va du plus petit numéro de ligne jusqu'au plus grand.

La numérotation peut partir d'une valeur quelconque et se poursuivre avec un pas quelconque, même variable. Les seules limitations concernent la valeur maximale (normalement elle est de 32 767, donc la ligne numérotée 40 000 ne peut exister) et la séquence de progression. Si l'on rentre les instructions numéros 10, 35, 160, 2 000, puis l'instruction numéro 20, l'interpréteur Basic place cette dernière ligne à sa place dans l'ordre croissant (qui n'est pas celui d'entrée). Le programme sera constitué de la séquence : 10, 20, 35, 160, 2 000, même si la ligne 20 a été la dernière entrée. Le format des instructions est celui-ci :

nnnn	Instruction
1 260	PRINT 6 + 4 (CR)

Le symbole nnnn est le numéro de ligne ; le signe **CR** indique que, pour terminer et saisir la ligne, il faut frapper la touche CR.

On peut écrire différentes instructions de programme (le maximum de caractères qu'il est possible d'entrer sur chaque ligne est 72 ou 255, selon la version du Basic) sur la même ligne physique, les différentes instructions étant séparées par un symbole déterminé. Le plus employé est :, mais certaines machines font usage du symbole backslash, \ . Par exemple le programme

```
10 PRINT  6 + 4
20 PRINT 10 - 2
30 PRINT  8 + 3
```

peut être écrit de la manière suivante :

```
10 PRINT 6 + 4 : PRINT 10 - 2 : PRINT 8 + 3
```

Si la machine adopte l'autre symbole \ , on aura :

```
10 PRINT 6 + 4 \ PRINT 10 - 2 \
PRINT 8 + 3
```

Les fonctions de la touche (CTRL)

Dans la présentation du clavier, on a mentionné la présence de certaines touches spéciales qui ont des fonctions particulières.

Une d'entre elles est la touche CONTROL,

Codes commandes du Basic 80

CTRL + A	La machine se met en mode éditeur (correction des programmes).
CTRL + C	Interrompt l'exécution d'un programme en cours (BREAK) et met la machine en mode commande.
CTRL + G	Sert à activer le signal acoustique habituellement présent sur l'unité vidéo. Cette fonction est également utilisée pour attirer l'attention du programmeur quand son intervention est requise. En ce cas, c'est le programme d'application qui doit émettre, au terminal, le même code que celui généré par la saisie simultanée des touches CTRL + G.
CTRL + H	Déplace le curseur (vers la gauche) et supprime le dernier caractère. Si, par exemple, apparaît sur l'écran le mot PARIS, la commande CTRL + H efface le dernier caractère ; on a PARI. En phase de correction, on effacera autant de caractères que l'on voudra en répétant l'entrée de cette instruction.
CTRL + I	Déclenche la tabulation. Le curseur se déplace de 8 colonnes à chaque commande CTRL + I. L'écran comportant généralement 80 colonnes, on peut effectuer 10 tabulations au maximum.
CTRL + O	Ferme momentanément la sortie d'un programme d'application qui tourne (par exemple, l'impression) ; une seconde entrée de cette même commande réactive la sortie du programme interrompue précédemment.
CTRL + R	Réécrit une ligne (on l'emploie pour dupliquer une ligne sans avoir à la retaper).
CTRL + S	Suspend l'exécution d'un programme (ne pas confondre avec CTRL + C, qui interrompt définitivement le programme).
CTRL + Q	Réactive le programme suspendu par CTRL + S.
CTRL + U	Efface une ligne.

habituellement indiquée par le sigle CTRL.
Comme on peut voir sur le tableau de la page 322, les touches dites spéciales ne sont pas des touches uniques, mais une combinaison de la touche CTRL, et de l'une des lettres (A, C, G, H, I, O, R, S, Q ou U). Utilisées conjointement, elles deviennent une série de commandes interprétées et exécutées par le Basic. La symbolisation CTRL + une lettre indique que, pour activer la fonction, il faut presser simultanément la touche CTRL, et celle de la lettre relative souhaitée.
Toutes les commandes possibles figurent sur le tableau à côté de chaque code.

Utilisation interactive du Basic (mode direct ou immédiat)

Le langage Basic, grâce à son interpréteur, permet l'exécution immédiate de calculs, sans d'autre instruction que la saisie au clavier du calcul à exécuter. Cette façon de procéder n'est pas possible dans d'autres langages, car il faut d'abord saisir la totalité de la formule du calcul à exécuter, comme s'il s'agissait d'une ligne source (c'est-à-dire écrite en symbolique de haut niveau, Fortran, Cobol, etc.), puis compiler la ligne, comme s'il s'agissait d'un programme ordinaire et l'exécuter ensuite. La caractéristique fondamentale du Basic réside dans son programme d'interprétation, qui analyse, et traduit en langage machine chaque ligne au moment de la saisie. Cela constitue un avantage appréciable surtout pour la recherche des erreurs.
Pour vérifier l'emplacement d'une erreur, on peut développer les calculs un par un (mode immédiat). L'utilisateur suivra le déroulement du programme en simulant les fonctions au clavier et en procédant instruction par instruction ; ainsi, il vérifiera les résultats et trouvera l'instruction erronée.
Les opérations qui peuvent se dérouler de façon interactive sont celles prévues dans le fonctionnement normal (programmation), et on peut utiliser les parenthèses. Le calcul à exécuter doit être précédé d'une commande spécifiant la fonction demandée.
Si on veut voir apparaître le résultat sur l'écran, il faut donner l'instruction :

PRINT formule du calcul (CR)

alors que sur l'imprimante, elle devient :

L PRINT formule du calcul (CR)

Rappelons que le symbole CR indique que l'exécution d'un calcul s'obtient en pressant la touche CR. Par exemple l'instruction :

PRINT (7 + 3) / 2 (CR)

donne sur l'écan le résultat : 5.
En mode interactif, on peut aussi utiliser les opérateurs logiques (AND, NOT, OR, etc.).
Par exemple, PRINT 5 OR 2 donne 7 (en binaire 5 = 101 et 2 = 10, dont 101 OR 10 = 111 = 7 en décimal).
Le langage Basic (comme tout langage) dipose d'une « bibliothèque » de fonctions, c'est-à-dire d'un groupe de programmes exécutant des fonctions mathématiques spéciales et pouvant être employées de façon immédiate.
Par exemple, la racine carrée d'un nombre est calculée par un de ces programmes, appelé **algorithme.**

Symbolique, signification et priorité des opérateurs

Les opérateurs reconnus par le Basic sont de quatre types :

- Arithmétiques
- Relationnels
- Logiques
- Fonctionnels

Chaque groupe possède sa propre hiérarchie, c'est-à-dire un ordre de priorité des calculs. L'ordre hiérarchique peut être modifié par l'emploi de parenthèses. Le calcul s'effectue alors en commençant par l'expression entre parenthèses comme dans une expression algébrique classique.
Dans un calcul mixte, c'est-à-dire contenant différents types d'opérateurs, la priorité suit l'ordre des opérateurs mentionnés ci-dessus. Par exemple, si un calcul contient des opérateurs logiques et des opérateurs arithmétiques, les opérateurs arithmétiques seront considérés en premier.

Opérateurs arithmétiques

Les opérateurs arithmétiques prévus en Basic, par ordre de priorité, sont les suivants :

UTILISATION DU BASIC EN INTERACTIF



```
PRINT 8*(12.3+(65.8-21)*5)/3
 627.467
Ok
?8*(12.3+(65.8-21)*5)/3
 627.467
Ok
PRINT 32 AND 78
 8
Ok
PRINT 10 OR 50
 58
Ok
? 10 OR 50
 58
Ok
PRINT ?(128 AND 255) OR (72 OR 83) AND (44 OR 15)
 11
Ok
?(128 AND 255) OR (72 OR 83) AND (44 OR 15)
 11
Ok
```

■ On peut exécuter des opérations arithmétiques et logiques en mode interactif au moyen de la seule commande PRINT suivie de l'expression à calculer. Une fois l'expression écrite on doit frapper la touche RETURN.

■ Le système répond par le résultat de l'opération qu'on lui a soumise et se prépare à recevoir une nouvelle entrée.

■ L'instruction PRINT peut être remplacée par le symbole « ? » qui a la même signification.
En mode interactif toutes les fonctions du Basic peuvent être utilisées.

Opérateur	Symbole	Exemple
Elévation à une puissance	∧	3∧2
Changement de signe	–	– 3
Multiplication	*	5*2
Division	/	8/3
Addition	+	2+7
Soustraction	–	6–4

Dans certaines machines, l'exposant peut aussi être indiqué par le symbole ↑, ou par deux symboles de multiplication **. Par exemple, le calcul 3^2 peut être symbolisé : 3∧2, 3↑2, ou 3**2.

Dans l'écriture des expressions à calculer, soit en mode interactif, soit en programmation, il faut prêter une attention particulière à la priorité des calculs. Les opérations se déroulent selon l'ordre indiqué par le tableau ; le premier calcul à faire est l'élévation à la puissance. Viennent ensuite le changement de signe, la multiplication, etc. L'ordre peut toutefois être changé par l'usage des parenthèses.
Par exemple l'instruction :

PRINT 5∧2 + 3*7 / 2

donne 35.5*. Sur le schéma de la page 325

EXEMPLES DU DEROULEMENT DE CALCULS EN BASIC

5∧2+3*7/2

Etape 1 25+3*7/2

Etape 2 25+21/2

Etape 3 25+10.5

Etape 4 35.5

5∧2+3*7/2=35.5

Le premier calcul exécuté est 5∧2 = 5^2 = 25, car l'exposant a la priorité la plus haute ; il est suivi de l'opérateur * (multiplication), puis de l'opérateur / (division) et enfin de l'opérateur + (addition).

5∧(2+3)*7/2

Etape 1 5∧5*7/2

Etape 2 3125*7/2

Etape 3 21875/2

Etape 4 10937.5

5∧(2+3)*7/2=10937.5

Dans ce cas le calcul entre parenthèses (5∧5 = 5^5 = 3125) est exécuté en premier.

Expression algébrique	Expression en Basic	
(5x3) – 7 : 2	(5*3) – 7/2	
$(9^2+2x6):4^3$	(9∧2+2*6)/4∧3	(9^2 = 9∧2 ; 4^3 = 4∧3)
$\frac{3^2+4}{6}$	(3∧2+4)/6	
$7^{2/3}+9$	7∧(2/3)+9	
$\frac{5x3+2-1}{8^2}$	(5*3+2–1)/8∧2	
$\frac{AxB}{C}+C^2$	(A*B)/C+C∧2	

Le calcul peut être indiqué par des symboles (A, B, C, etc.) auxquels on assignera des valeurs numériques.

est reproduite la succession des calculs.
En modifiant la priorité par l'usage des parenthèses, on aura :

PRINT 5 ∧ (2 + 3) * 7 / 2

dont le résultat est 10937.5. Pour le développement d'un calcul (expression algébrique), il suffit de remplacer les opérateurs arithmétiques (somme, produit, etc.) par les symboles correspondants. Le schéma de la page 325 en fournit quelques exemples.

Opérateurs relationnels

Ils sont utilisés pour comparer deux grandeurs. Le résultat peut avoir la valeur 0, si la condition est fausse, la valeur − 1, si la condition est vraie.
Les opérateurs relationnels sont habituellement employés dans les instructions de « décision »



et le résultat de l'opération (vrai ou faux) est évalué par l'instruction elle-même ; l'utilisateur l'emploie implicitement. On trouvera ci-dessous, classés par ordre de priorité, les opérateurs relationnels disponibles en Basic.

Opérateur	Symbole	Exemple	
Egalité	=	A = B	met la valeur de A égale à celle de B, ou bien la condition est vraie si A est égal à B.
Inégalité	<>	A <> B	vrai si A est différent de B.
Inférieur	<	A < B	vrai si A est inférieur à B.
Supérieur	>	A > B	vrai si A est supérieur à B.
Inférieur ou égal	<=	A <= B	vrai dans les deux cas, A inférieur à B ou A égal à B.
Supérieur ou égal	>=	A >= B	vrai dans les deux cas, A supérieur à B ou A égal à B.

Les programmes de recherche de données emploient ces opérateurs. Pour déterminer si la valeur cherchée se situe à gauche ou à droite de celle lue, la recherche dichotomique sur des archives triées implique une série de comparaisons effectuées au moyen des opérateurs relationnels > (supérieur), < (inférieur), = (égal). Le schéma en haut de la page 327 représente l'organigramme d'une recherche binaire (dichotomique) de données.
Les opérateurs relationnels peuvent être inclus dans des expressions qui contiennent aussi des opérateurs arithmétiques (∧, *, /, +, −), mais ces derniers sont prioritaires. Par exemple, dans l'expression 3 + 5 < 7 * 2, on exécute d'abord les calculs arithmétiques (3 + 5 et 7 * 2). L'expression équivaut à 8 < 14.

Opérateurs logiques

Les opérateurs logiques du Basic sont ceux déjà énoncés page 73. Leur priorité d'application est la suivante :

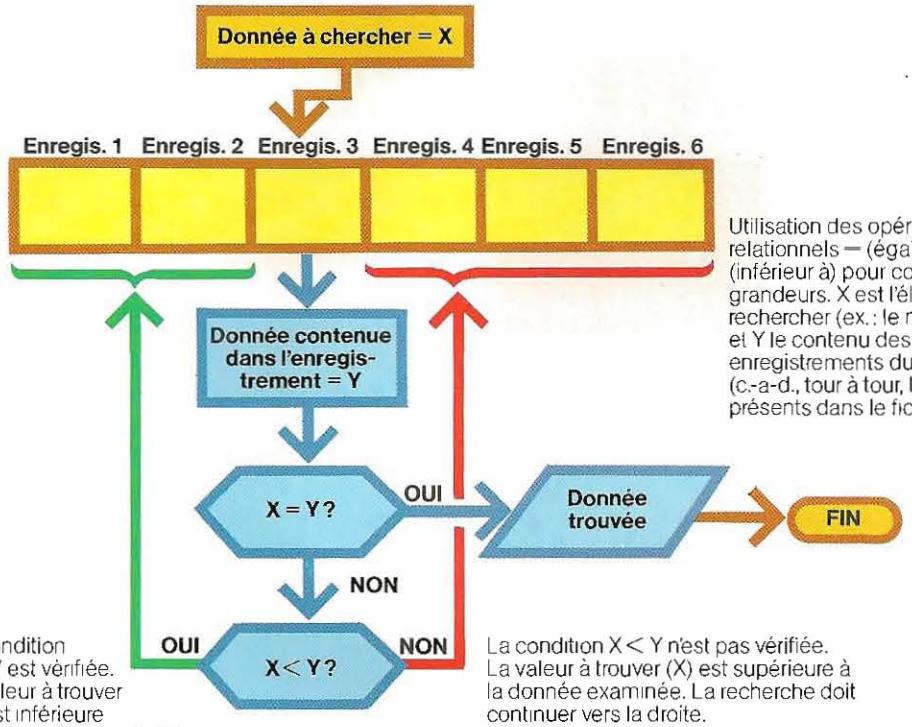
Opérateur	Données	Résultat
NOT	A = 1	NOT A = 0
AND	A = 1, B = 0	A AND B = 0
OR	A = 1, B = 0	A OR B = 1
XOR	A = 1, B = 0	A XOR B = 1

Outre ces quatre opérateurs d'usage fréquent, le Basic 80 en prévoit deux autres pas encore traités : IMP et EQU. Les tables de vérité correspondantes sont reproduites au bas de la page 327. Ces deux opérateurs ne figurent pas dans toutes les versions du Basic et ils sont rarement utilisés dans les programmes d'application.
Les opérateurs logiques peuvent s'appliquer à des nombres entiers (ceux que la machine peut contenir en mémoire sont compris entre − 32 768 et 32 767) et opèrent sur les bits dont sont composés les nombres. Les opérateurs arithmétiques, relationnels et logiques peuvent être employés dans la même expression. Par exemple, la condition :

(A + B) > C AND F = B

*Pour la représentation des nombres réels, on utilisera désormais le point décimal de préférence à la virgule. C'est la représentation standard en calcul informatique (par ex., 35.5 plutôt que 35,5).

EXEMPLE D'APPLICATION DES OPERATEURS RELATIONNELS A LA RECHERCHE DE DONNEES



Utilisation des opérateurs relationnels = (égal) et < (inférieur à) pour comparer deux grandeurs. X est l'élément à rechercher (ex.: le nom MARTIN) et Y le contenu des différents enregistrements du fichier (c.-a-d., tour à tour, les noms présents dans le fichier).

La condition X < Y est vérifiée. La valeur à trouver (X) est inférieure à la demande examinée. La recherche doit continuer vers la gauche.

La condition X < Y n'est pas vérifiée. La valeur à trouver (X) est supérieure à la donnée examinée. La recherche doit continuer vers la droite.

TABLES DE VERITE DES OPERATIONS IMP ET EQU

Opérateur IMP ⟹

A	B	A IMP B
1	1	1
1	0	0
0	1	1
0	0	1

L'opérateur IMP reconnaît la séquence 1 0 et donne 0 en sortie. Dans tous les autres cas, la sortie (résultat) est 1.

Opérateur EQU ⟺

A	B	A EQU B
1	1	1
1	0	0
0	1	0
0	0	1

L'opérateur EQU reconnaît le cas d'égalité des deux entrées (toutes deux 1 ou 0); dans ce cas, le résultat est 1, sinon 0.

La modélisation sur ordinateur

La mise en service d'un nouvel appareil relevant de technologies avancées exige de nombreux contrôles (parfois très poussés), nécessaires pour vérifier que la machine présente bien la fiabilité souhaitée.

Ainsi, les fabricants de plate-formes pétrolières, de moteurs d'automobiles, de réacteurs, ou de circuits électroniques sont tenus de prévoir avec la plus haute précision possible quel sera le comportement de leurs produits après leur sortie d'usine.

Ces appareils sont soumis à de véritables traitements de choc dans des conditions particulières destinées à mettre à l'épreuve leur résistance. Ces procédés coûtent d'ailleurs fort cher mais ils sont indispensables.

De nos jours, ces expériences peuvent être menées, non plus « sur le terrain », mais simulées par ordinateur. C'est là que le calculateur électronique se révèle d'une aide précieuse dans l'application de ces méthodes pouvant être mises en œuvre grâce à des modèles mathématiques reproduisant le comportement de l'appareil à l'essai.

Si les calculs révèlent l'existence de points faibles dans la structure de l'appareil, on modifie le modèle directement sur la table à dessin, et on recommence ainsi jusqu'à obtention des résultats qui correspondent au fonctionnement désiré.

Lorsqu'il s'agit d'équipements particulièrement complexes, comme une tour de forage pétrolier ou un avion, les modèles mathématiques de simulation s'avèrent eux aussi très compliqués, et les calculs deviennent tellement longs et fastidieux qu'ils sont confiés directement à l'ordinateur.

Le travail des concepteurs s'en trouve considérablement allégé et ces derniers peuvent ainsi passer plus de temps à concevoir de nouveaux modèles et à élaborer les méthodes appropriées pour les tester.

Cette utilisation de l'ordinateur qui permet de prévoir le comportement d'une structure se nomme **simulation par ordinateur.**

Dans un plus vaste domaine d'application de l'électronique, c'est l'ordinateur lui-même qui dessine et modifie les détails d'un projet ; il le représente en deux ou en trois dimensions de façon à n'ignorer aucun de ses aspects.

Ces méthodes de « prototypage » se divisent en deux étapes principales : le traitement numérique des équations mathématiques qui décriront le comportement de la structure, et la visualisation des résultats du calcul, permettant au concepteur de pourvoir à la correction des erreurs.

Les résultats du traitement numérique sont communiqués au technicien, non pas en chiffres, mais sous forme de dessins quelquefois très élaborés.

Les systèmes de modélisation informatisée font une large utilisation de la **table traçante** *(plotter). Cet appareil électronique exécute le dessin à partir des instructions fournies par l'ordinateur ; les tracés apparaissent à l'écran. Quand une table traçante fait office de terminal de sortie d'un ordinateur, on peut disposer d'un enregistrement permanent du dessin.*

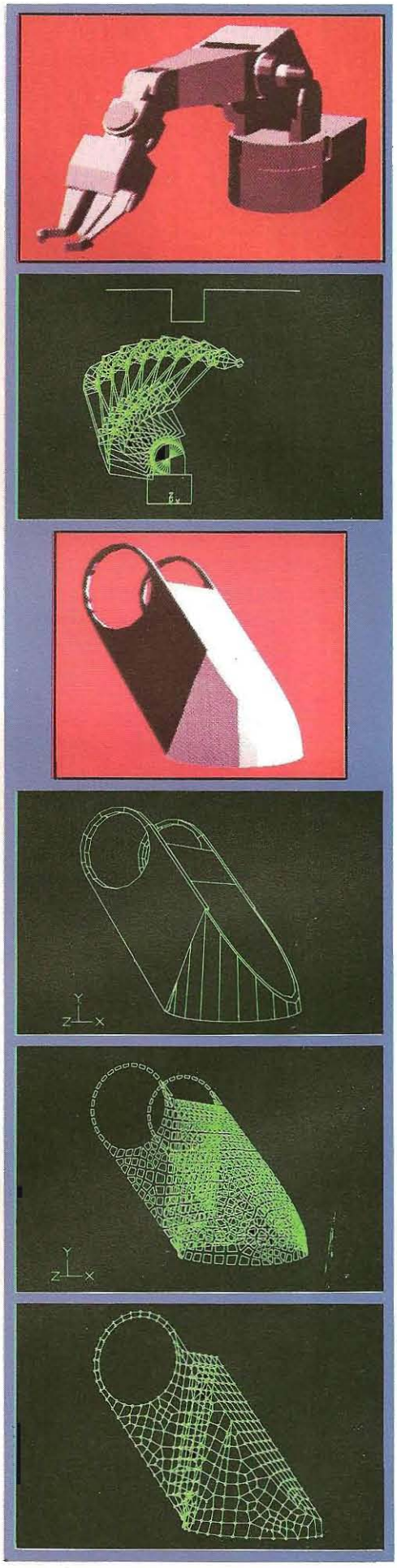
Lorsque les schémas sont très complexes, il est nécessaire de les faire apparaître sur la table traçante, le résultat visible sur l'écran étant trop peu précis pour être exploité. Notons toutefois qu'une image se dessine beaucoup plus rapidement sur l'écran que sur la table traçante, et qu'elle peut en outre être modifiée assez vite si l'on veut représenter l'objet sous d'autres angles. Ainsi, le déroulement du processus graphique peut être contrôlé à chaque étape.

La conception des circuits électroniques est un des domaines où l'ordinateur apporte à l'homme la contribution la plus appréciable. Les circuits de type numérique sont formés de milliers de **portes logiques** *qui contrôlent de faibles impulsions électriques permettant (ou non) au courant de traverser les différentes parties du circuit, suivant la présence (ou l'absence) d'autres impulsions. En fait, les circuits peuvent devenir extrêmement compliqués et, si le concepteur néglige le fonctionnement d'une seule de ces portes, le fruit de son travail sera anéanti par cette unique erreur. L'ordinateur peut, en revanche, manipuler avec aisance les milliers d'éléments sans commettre ni erreur ni oubli.*

Les modèles simulent aussi le fonctionnement des structures mécaniques.

Il serait en effet inimaginable de construire (par exemple) de véritables tours de forage à seule fin de vérifier leur fiabilité dans les vraies conditions de leur implantation future (pleine mer, tempête, etc.). D'autant plus que, dans certains cas, il faudrait peut-être attendre

longtemps pour que soient effectivement rassemblées les conditions souhaitées.
La méthode par « éléments finis » est utilisée comme outil mathématique pour l'analyse des tensions auxquelles peuvent être soumis les appareils, qu'il s'agisse de l'ossature d'un avion ou de la simple dent d'un engrenage.
Les éléments finis permettent de schématiser les pièces par des formes élémentaires, triangles, rectangles ou lignes.
L'ingénieur introduit dans l'ordinateur les données décrivant le modèle de la structure à concevoir afin que la machine en construise les éléments finis. Ces éléments apparaissent alors à l'écran comme des sortes de toiles d'araignées dont l'entrecroisement de fils reproduit l'image de la structure réelle.
L'utilisateur peut demander à l'ordinateur une rotation du modèle afin de l'examiner sous un autre angle, ou un agrandissement de tel ou tel élément qui doit être étudié plus en détail.
Pour simuler la situation dans laquelle une charge est exercée sur l'un ou l'autre point de la structure à l'étude, le concepteur communiquera à la machine la valeur des pressions envisagées et leur point d'impact. La machine fera apparaître à l'écran les déformations consécutives de la structure, superposant éventuellement cette image de l'élément déformé à celle de l'élément intact.
L'ordinateur sait aussi analyser les effets produits sur la structure par des anomalies de fonctionnement ou par des situations accidentelles : on peut ainsi étudier ce qui se passerait, par exemple, si la plate-forme pétrolière perdait un de ses piliers de soutien.
Naturellement, les résultats ne sont intéressants que si les paramètres de base communiqués à la machine sont précisément ceux de la réalité.
La précision dans ce domaine sera d'autant améliorée qu'on aura recours à de petits éléments, produisant à l'écran des réseaux très serrés. Toutefois, l'emploi de ces réseaux revient très cher, leur mise au point étant longue et nécessitant aussi un long traitement de l'ordinateur. Ils ne seront donc pas toujours recherchés, d'autant plus que des réseaux à mailles larges procurent la plupart du temps

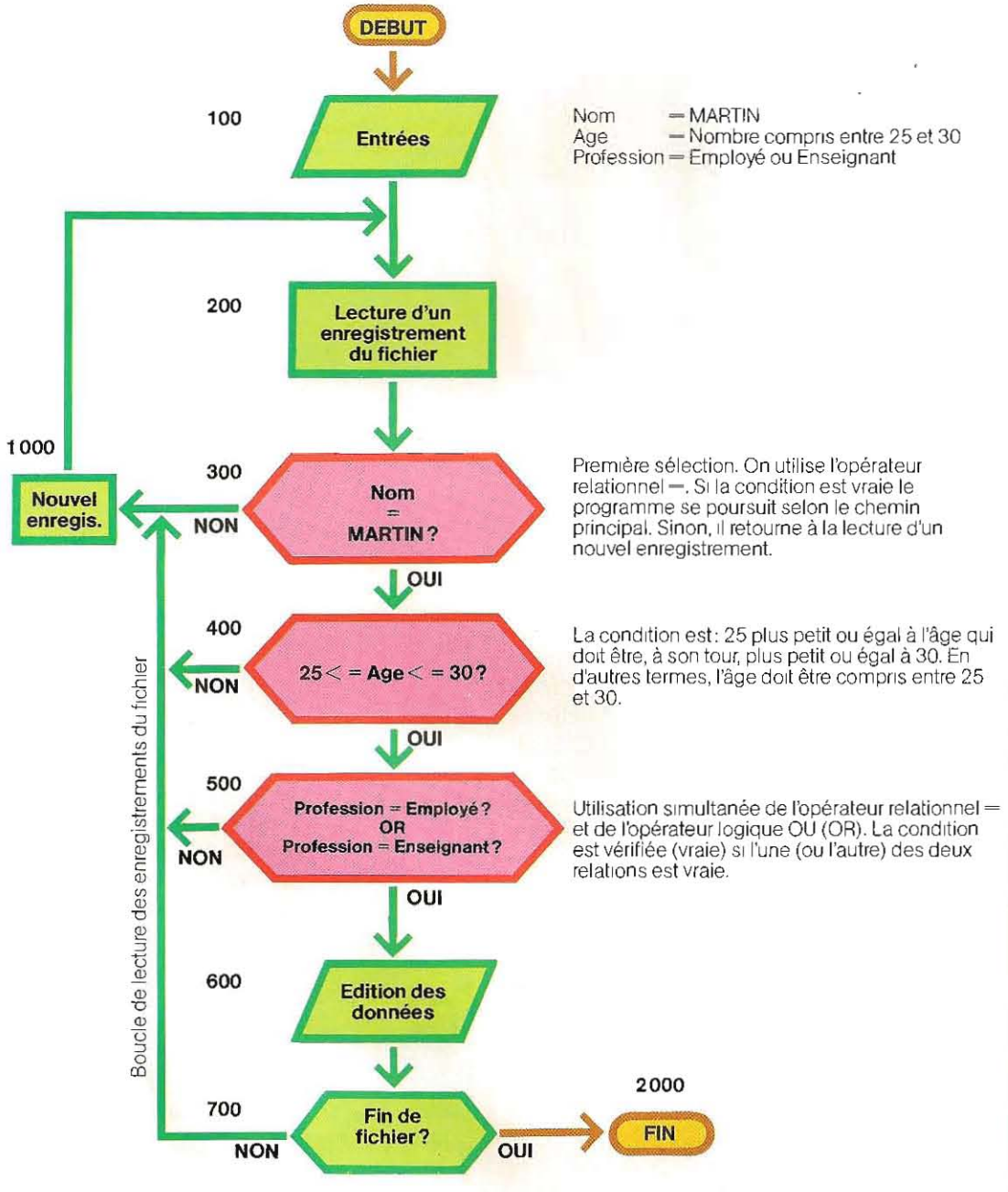


SDRC Engineering Services Ltd

Sur les deux premières images : visualisation à l'écran d'un bras articulé et du déroulement de son mouvement. Sur les autres : une pièce et différents réseaux représentant ses éléments finis.

des résultats suffisamment précis.
Les modèles informatisés se révèlent fort utiles dans l'étude des appareils en mouvement. Une application particulièrement spectaculaire en est donnée avec notre exemple de la plate-forme pétrolière. Dans la réalité, la tour est traînée (couchée sur le flanc) par un remorqueur jusqu'au lieu de son implantation. On doit ensuite la redresser en lui faisant accomplir un mouvement de 90° pour l'arrimer à l'endroit voulu. La traduction graphique de cette simulation est un ensemble de dessins reproduisant les positions successives de la tour lors de son installation. Les dessins sont tridimensionnels et on peut, à volonté, examiner la tour sous tous ses angles quand on le désire.
De la même façon, on utilise ces modèles pour des programmes d'étude des mouvements effectués par le corps humain dans différentes situations, et tout particulièrement, lors de la manœuvre d'une machine. Cela permet de construire des machines adaptées à leur emploi par l'homme. SAMMIE (sigle de System for aiding man-machine interaction evaluation, soit, en français, Système d'aide à l'évaluation de l'interaction homme-machine) est l'un de ces programmes. Il permet de dessiner, en trois dimensions, des personnages en situation, superposés à des machines, des appareils divers, ou des édifices. Les personnages peuvent se déplacer sur l'écran, s'asseoir ou se lever, et l'on peut ainsi vérifier de quelle manière ils réussissent à atteindre les diverses commandes d'une machine.
Le modèle (ayant cette fois la silhouette humaine) est réalisé de façon à se mouvoir exactement comme un être humain, c'est-à-dire disposant des mêmes articulations corporelles. Si, par erreur, un mouvement impossible pour l'homme était imposé au modèle (par exemple, la torsion excessive d'un bras, ou la rotation de la tête à 100°), la machine le signalerait.
De même que la stature humaine varie d'une personne à l'autre, celle du mannequin sera modifiée dans certains cas. Cependant, il est plus facile de concevoir un personnage de taille moyenne et de mener la plupart des essais en fonctions de ses mensurations.
L'étape suivante consiste donc à dessiner le milieu dans lequel le mannequin doit évoluer, que ce soit un bureau à cloisons mobiles, la cabine d'un tracteur agricole, l'intérieur d'une voiture, ou simplement une cuisine équipée de ses différents appareils électroménagers et de ses éléments de rangement. Une fois le décor en place, le concepteur peut, par le jeu de commandes simples, indiquer à l'ordinateur les déplacements ou les mouvements du mannequin, selon ce qu'il a à accomplir.
Les dessins réalisés par SAMMIE sont en trois dimensions, et le programmeur a toute possibilité de demander à l'ordinateur de lui montrer d'autres aspects de la scène. Il peut également obtenir des vues rapprochées de certaines parties du dessin, autorisant une étude plus en détail de certaines situations. Là aussi, l'ordinateur transmettra un message si une situation ne correspond pas à ce qu'un être humain pourrait effectivement faire. Le concepteur devra répéter son essai.
Tous les éléments du décor ou de l'habitacle pourront être déplacés, dessinés avec de nouvelles dimensions, ou conçus différemment selon les problèmes précis qui auront été mis en évidence. Ainsi, on changera le sens d'ouverture d'une porte de placard, on déplacera l'emplacement des commandes du tracteur pour que les manœuvres soient facilitées, on modifiera la ligne des sièges d'une automobile pour garantir une plus grande sécurité au conducteur.
Dans toutes ces applications, l'ordinateur, en assumant vite et bien tous les innombrables calculs de base et de corrections, offre la possibilité de répéter les essais autant de fois qu'il le faut tant qu'on n'a pas obtenu toute la fiabilité nécessaire à l'appareil étudié, et permet de réaliser ensuite des produits d'une très haute précision, et réunissant tous les critères souhaitables de sécurité.
En outre, il laisse à l'ingénieur tout le temps utile à sa recherche de création. Le fait que l'ordinateur puisse ensuite donner l'analyse de situations fictives vient encore encourager l'imagination du concepteur qui se consacre à l'étude de matériels de plus en plus performants. La généralisation de la modélisation informatisée conduira à l'élaboration de prototypes qui, bénéficiant toujours davantage des progrès de l'informatique, apporteront à leur tour des données de plus en plus sophistiquées au travail de l'ordinateur, accélérant ainsi le progrès technique dans son ensemble.

UTILISATION SIMULTANEE D'OPERATEURS RELATIONNELS ET LOGIQUES



est vraie (elle donne comme résultat — 1), si la somme est A + B supérieure à C et si, en même temps, la valeur de F égale celle de B. Le schéma ci-dessus illustre un exemple d'utilisation simultanée de différents types d'opérateurs. Le problème examiné dans l'exemple traite de l'impression statistique du contenu d'un fichier. On veut imprimer les données de tous les individus dont le nom est Martin, dont l'âge est compris entre 25 et 30 ans inclus et qui sont employés ou enseignants.

Opérateurs fonctionnels

On peut transformer une séquence de calculs en **fonction**, représentée par la suite des opérations à effectuer sur une donnée opérande. Ainsi, ce calcul du montant correspondant à 18 % d'un coût donné :

$$\text{montant} = \frac{\text{coût} \times 18}{100} = \text{coût} \times 0.18$$

peut être transformé en une fonction qui consiste à multiplier par 0.18, le coût donné. Mathématiquement, la fonction est représentée ainsi :

$$\text{montant} = f(\text{coût})$$

Le symbole f (coût), signifie, en ce cas, la multiplication : coût × 0.18. Un symbolisme très voisin est adopté en Basic. Le symbole d'une fonction est formé de deux lettres de la fonction. Ainsi le symbole FNA représente une fonction de nom A. Dans l'exemple précédent l'opérande est coût et la formulation complète de la fonction est :

FNA (coût)

où FNA (coût) n'est autre que le produit coût × 0.18. Une fois définie, la fonction peut être utilisée dans un calcul quelconque. Ainsi, si les 18 % représentent une remise, pour calculer le prix on devra d'abord calculer le montant de la remise et le déduire du coût (prix sans remise) :

$$\text{remise} = \frac{\text{coût} \times 18}{100} = \text{coût} \times 0.18$$

prix = coût − remise = coût − coût × 0.18.

Ayant défini FNA (coût) comme montant de la remise, on a :

Définition (DEF) :
remise = FNA (coût) (soit coût × 0.18)

Calcul
prix = coût − FNA (coût)

Ce genre de fonction est dit **fonction utilisateur** car elle n'est pas définie à priori dans le Basic et doit donc être définie par le programmeur.

L'instruction de définition d'une fonction doit être établie avant l'utilisation de cette fonction, sinon le système émet un diagnostic de fonction inconnue et s'arrête dans l'attente d'une correction. Pour indiquer au système l'existence de cette nouvelle fonction FNA (coût), on a besoin de l'instruction de définition DEF FNA (soit définition de la fonction A). Comme dans presque toutes les instructions Basic, le symbole indique le nom même de l'opération à exécuter. L'opération étant la **définition**, le code Basic correspondant est DEF.
Pour définir la fonction de l'exemple précédent, l'instruction est :

DEF FNA (coût) = coût * 0.18

soit
- codage de l'instruction de définition
- symbole indiquant une fonction à la machine
- nom de la fonction
- paramètre
- calcul effectué par la fonction

Les fonctions définies par l'utilisateur peuvent être de trois types :

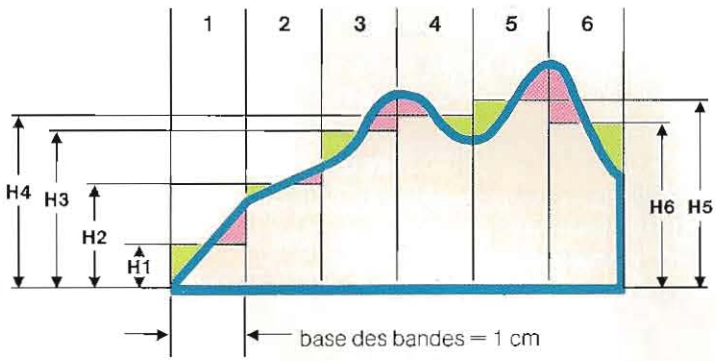
- **réelles :** toutes celles qui opèrent sur des nombres réels, c'est-à-dire formés d'une partie entière et d'une partie décimale.
- **entières :** toutes celles qui n'opèrent que sur des nombres entiers.
- **de chaîne :** les fonctions opérant sur des caractères (lettres ou chiffres).

Les caractéristiques de ces trois types et la façon de définir les fonctions dans les trois cas énumérés seront exposées par la suite. En plus des fonctions définies par l'utilisateur, il existe des fonctions de bibliothèque, déjà prêtes, qui exécutent les opérations mathématiques les plus courantes. Ces fonctions définies au niveau système, sont « appelées » (c'est-à-dire utilisées) par un nom symbolique (défini dans le Basic), sans la contrainte du sigle FNX (X = nom de la fonction).
Par exemple, la fonction qui détermine la racine carrée d'un nombre a pour nom le sigle SQR (abréviation de square root = racine carrée) ; ainsi pour extraire la racine du nombre 1 673 on devra écrire :

SQR (1673)
code paramètre

Si l'on veut comprendre pourquoi un ordinateur requiert une fonction (et non un calcul direct) pour le calcul d'une racine, il faut analyser la structure propre à la machine. Le système exécute les calculs en utilisant les microprogrammes (microcodes) contenus dans l'UAL (unité arithmétique et logique). Les signaux qui représentent les opérandes (les nombres) et les opérateurs (les opérations à exécuter) y sont interprétés, puis transmis à des circuits spéciaux qui fournissent les résultats en sortie.
Toutefois ces circuits ne sont en mesure d'exécuter que quelques types de calculs par

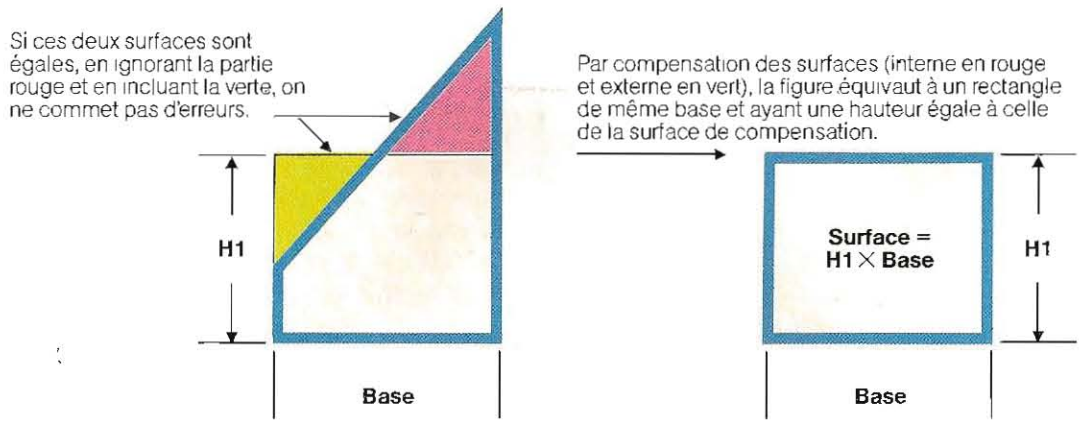
PROCEDE GEOMETRIQUE DE CALCUL DES SURFACES



La figure plane (ligne bleutée) est divisée en un certain nombre (6) de bandes verticales. La surface totale est la somme des surfaces élémentaires, de chaque bande ; cette surface élémentaire s'obtient en multipliant la base par la hauteur correspondante (H1, H2, etc).

complément, essentiellement l'addition et la soustraction. Les opérations de multiplication et de division sont exécutées par des microprogrammes habituellement extérieurs à l'UAL. On dit que ces opérations sont exécutées par logiciel, car elles sont indépendantes des circuits. Tous les autres calculs, par exemple la racine carrée, sont exécutés au moyen des quatre opérations fondamentales.
A titre d'exemple, appliquons le concept de fonction à un programme de calcul de surfaces. Le problème est le suivant : étant donné une figure plane délimitée par une droite horizontale, deux droites verticales et une courbe, calculer la surface de la figure.
La première étape consiste à diviser la figure en un certain nombre de bandes verticales

METHODE DE COMPENSATION POUR LE CALCUL DES SURFACES



(voir le schéma de la page 333) ayant toutes une base égale (par ex., 1 cm). Chaque bande peut être considérée comme un rectangle. On choisit la hauteur du rectangle (H1 sur le schéma ci-dessus), de façon que la zone de la figure exclue du calcul (superficie rouge) soit équivalente à une nouvelle surface supplémentaire (zone verte). De cette façon, tout en excluant une partie de la figure (rouge), on en inclut une autre (verte), de même surface, mais qui ne faisait pas partie de la figure auparavant. En d'autres termes, la surface de chacune des bandes verticales est obtenue comme s'il s'agissait d'un rectangle ayant comme hauteur H1, H2, etc. La subdivision en un certain nombre de bandes s'avère nécessaire pour minimiser l'erreur qui serait commise si l'on dessinait arbitrairement la droite de compensation (hauteur moyenne) sur la totalité de la base. Après avoir préparé le dessin comportant la mention des différentes hauteurs, on peut dessiner l'organigramme du programme. Les fonctions à développer sont les suivantes :

1 / demande du nombre de divisions verticales (6 divisions sont un minimum).
2 / demande de la dimension de la base (dans l'exemple 1 cm).
3 / boucle de recherche des hauteurs (en nombre égal au nombre des bandes), calcul des surfaces de chacun des rectangles et de leur somme.
4 / présentation du résultat (somme des surfaces élémentaires).

Le calcul le plus souvent employé est celui de la surface d'un rectangle ; pour le développer rapidement, on peut définir une fonction opérant sur les deux paramètres base et hauteur.

surface = base × hauteur

La fonction correspondante (par exemple nommée A) est :

DEF FNA (base, hauteur) =
= base × hauteur

soit
- instruction de définition
- symbole de fonction
- nom de la fonction
- paramètres
- calcul.

Le schéma de la page 336 présente l'organigramme de ce calcul. Cette méthode est connue comme « intégration graphique d'une fonction ». La courbe qui sous-tend la surface que l'on cherche à connaître est une fonction (au sens mathématique du terme) et le calcul de la surface est une méthode de calcul de l'intégrale de la fonction, étendu à l'intervalle de base. L'intégration représente un des aspects les plus complexes des mathématiques, au point que le calcul d'une intégrale par des méthodes analytiques peut être impossible. Dans les calculs scientifiques, la méthode que nous venons d'exposer est employée couramment, même quand il s'agit d'une forme qui permet d'éliminer la partie graphique à la charge de l'utilisateur.

Test 9

1 / Ecrire l'instruction qui, en mode interactif, permet de calculer la moyenne des trois nombres 5.2, 15.21, 7.43, de façon que le résultat apparaisse à l'écran et sur imprimante.

2 / Ecrire le résultat des expressions suivantes (attention à la priorité des opérateurs) :
3 + 2 AND 3 × 4 ; 5 + (2 AND 3) × 4

3 / Imaginer un fichier contenant les archives d'une bibliothèque de 1 000 volumes (1 000 enregistrements). Les données sont distribuées dans les champs (ou zones) suivants :

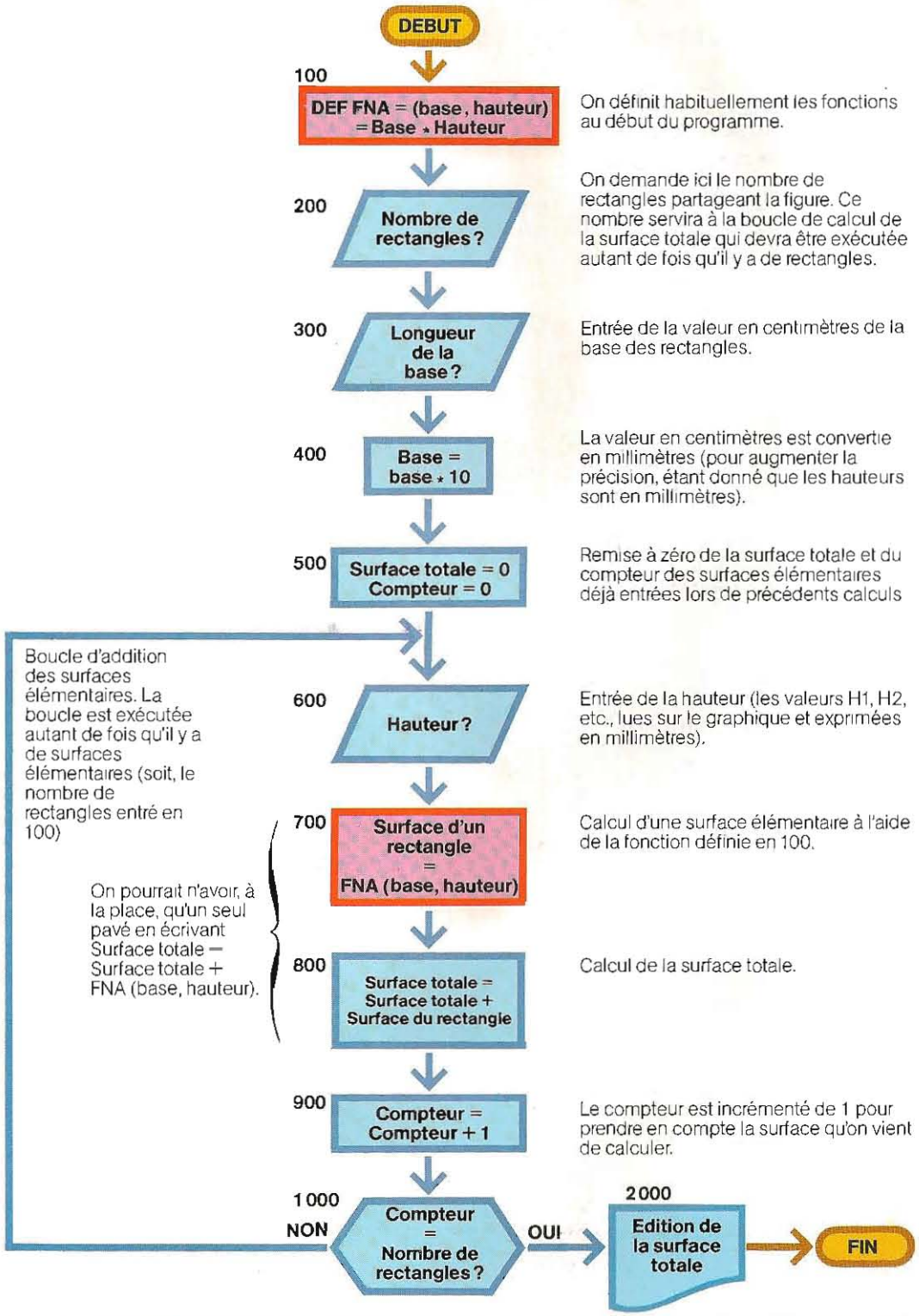
1 / Titre du livre	30 caractères
2 / Sujet	10 caractères
3 / Spécialité	10 caractères
4 / Langue (code)	3 caractères
5 / Disponibilité	1 caractère

Les rubriques (champ n° 2) représentent une subdivision à un niveau général (par exemple histoire, géographie, etc.), la spécialité (champ n° 3) est une subdivision du sujet (pour l'histoire, ce sera, par exemple, la préhistoire, l'époque romaine, etc.). Le champ n° 4 est relatif à la langue dans laquelle l'ouvrage est écrit (FRA = Français, ITA = Italien, etc.).
La disponibilité (champ n° 5) est un indicateur qui peut valoir 0 si le volume a été prêté, ou 1 s'il est présent dans la bibliothèque.
Dessiner l'organigramme d'un programme recherchant tous les livres présents (non prêtés) écrits en français (FRA) ou en anglais (ANG), et traitant de poésie. Le sujet est Art, la spécialité Poésie.

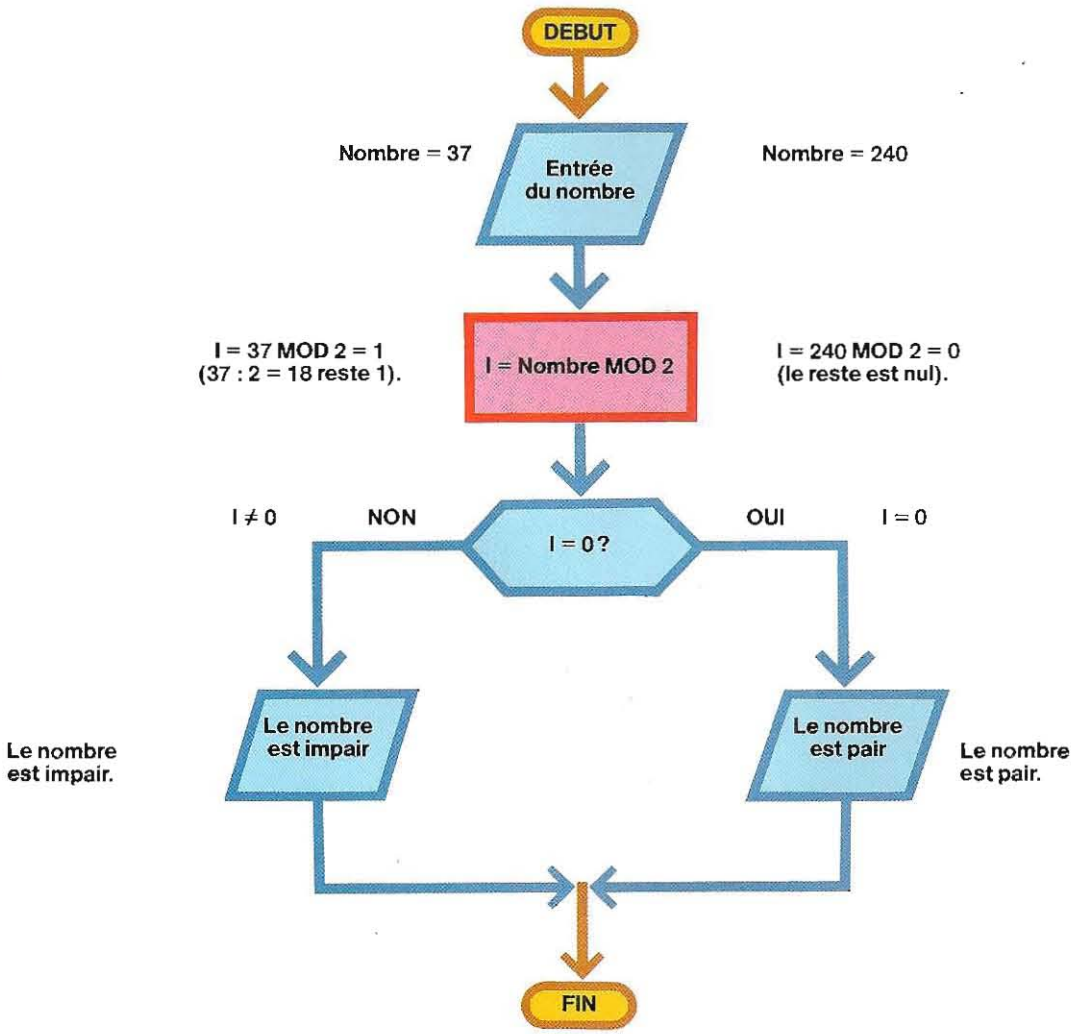
4 / Combien parmi les volumes contenus dans les archives (question n° 3) peuvent être stockés sur une disquette de 1 Moctet ?

Les solutions du test se trouvent pages 350 et 351.

ORGANIGRAMME DE CALCUL DES SURFACES



UTILISATION DE L'OPERATEUR MOD POUR DETERMINER LA PARITE D'UN NOMBRE



Opérateurs de division entière

Dans la division entière indiquée en Basic par le symbole \, les opérandes sont arrondis à une valeur entière avant l'exécution de la division, dont le résultat est lui-même un nombre entier. Ainsi :

35.7 \ 4.7 est évalué à 36 / 5 = 7

Si la conversion des opérandes conduit à un résultat dépassant les limites d'un entier (— 32 768, + 32 767), une erreur de débordement est diagnostiquée. Par exemple, (75.4 * 56 621) \ 5.6 ne peut être calculé car le numérateur excède la valeur 32 767.

Opérateur modulo

L'opérateur modulo (MOD) fournit une valeur entière égale au reste de la division de 2 nombres.
Ainsi :

7 MOD 2 = 1 car 7 : 2 = 3 reste 1
9 MOD 3 = 0 car 9 : 3 = 3 reste 0

L'opérateur MOD permet de s'assurer qu'un nombre est divisible par un autre : dans ce cas le résultat de MOD est 0.
Le schéma ci-dessus présente l'organigramme d'un programme déterminant la parité d'un nombre.

En résumé :
Le Basic possède trois modes opératoires :
– mode Commandes : pour l'entrée des commandes.
– mode Exécution : exécution interactive de programmes et de calculs.
– mode Editeur : pour d'éventuelles modifications de programmes.
Les opérateurs du Basic (par ordre de priorité) sont :
– arithmétiques : les opérations mathématiques habituelles (∧, changement de signe, *, /, +, −) ;
– relationnels : égal, supérieur, inférieur et leurs combinaisons ;
– logiques : les opérateurs logiques déjà énoncés dans le chapitre qui leur est consacré, plus deux autres, disponibles dans certaines versions ;
– fonctionnels : fonctions définies par l'utilisateur (DEF FNX) ou fonctions système ;
– division entière et modulo : opérateurs \ et MOD.
Le programmeur peut définir une fonction quelconque en utilisant la notation : DEF FNX (X = nom de la fonction, une lettre quelconque). Le Basic met à la disposition de l'utilisateur toute une bibliothèque de fonctions mathématiques déjà prédéfinies, et qu'il suffit d'appeler par leur nom symbolique.

Le contrôle s'effectue en vérifiant la divisibilité de ce nombre par 2 (si le nombre est divisible par 2, il est pair, sinon il est impair).

Constantes et variables

Comme dans tout langage symbolique, les données peuvent être de deux types : **constantes** et **variables.**
Les constantes ont une valeur fixe, alors que les variables prennent des valeurs qui peuvent changer au cours du traitement.
Les variables sont représentées par des noms symboliques, les constantes sont directement désignées par leur valeur.

Par exemple, si l'on veut calculer la circonférence d'un cercle, la formule est : circonférence = 2 * π * R. Le symbole π est une constante numérique (on verra qu'il existe un autre type de constante : la chaîne de caractères), tandis que le symbole R (rayon du cercle) est une variable, pouvant prendre une valeur quelconque (les chaînes peuvent également être des variables).

Constantes numériques

Il existe cinq types de constantes numériques en Basic :
1 / entières (integer) : représentant un nombre entier (et son signe) compris entre − 32 768 et + 32 767 (ou − 32 768 ÷ + 32 767), suivi du symbole %.
2 / réelles en virgule fixe (fixed point) : ce sont des nombres réels (contenant des décimales) et associés à des signes négatifs ou positifs (on les appelle des nombres décimaux signés).
3 / réelles en virgule flottante (floating point) : ce sont des nombres réels en représentation exponentielle.
La représentation exponentielle permet d'exprimer un nombre à l'aide de puissances de 10. Ainsi, le nombre 1 400 peut être écrit $14 * 100 = 14 * 10^2$.
De la même façon, le nombre 121.46 devient $12146 / 100 = 12146 / 10^2 = 12146 \times 10^{-2}$.
On doit se souvenir qu'écrire 1 : 100 équivaut à 1 : 10^2, soit encore 10^{-2}.
En Basic, le format à utiliser pour l'expression d'un nombre en virgule flottante est : Nombre (mantisse) E (symbole d'exposant) Exposant.
Par exemple :

Valeur numérique	Notation exponentielle	Notation en Basic
1400	14×10^2	14 E 2
1400	1.4×10^3	1.4 E 3
1400	0.14×10^4	0.14 E 4
121.46	12146×10^{-2}	12146 E − 2
121.46	1.2146×10^2	1.2146 E 2

Les trois représentations du nombre 1400 sont équivalentes.

4 / hexadécimales (HEX) : ce sont des nombres en représentation hexadécimale (base 16). Pour indiquer à la machine que l'on entend utiliser la notation hexadécimale, le

nombre doit être précédé des symboles &H. Ainsi, la valeur de nombre &H70 est 112 en décimal (70 en hexadécimal valant $7 \times 16 + 0 = 7 \times 16 = 112$).

5 / octales (OCT) : il s'agit de nombres représentés en base 8, et précédés du symbole & (« et » commercial), ou, sur certaines machines, &0 (lettre « o »). Ainsi, le nombre &70 vaut 56 en décimal.

Les notations octale et hexadécimale sont proches de la symbolisation binaire de l'ordinateur et on y aura recours pour toute référence à une adresse mémoire. Pour extraire le bit d'une donnée de nom X, on appliquera l'opérateur AND entre la donnée et un « masque », nombre binaire dont seul le bit 8 est égal à 1.

On convertit ce masque en octal :

Valeur	4 2 1	4 2 1	4 2 1
Binaire	1 0	0 0 0	0 0 0
Octale	2	0	0

10 000 000 (binaire) = 200 (octal)

Dans le programme, la fonction d'extraction s'écrira X AND & 200.

Précisions

Les nombres entiers sont représentés en machine sur 16 bits (2 octets), aussi leur taille est-elle limitée aux valeurs décimales que l'on peut écrire sur 16 bits, soit à l'intervalle $-32768, +32767$ ($-2^{15}, +2^{15} - 1$). En fait les bits utilisés pour décrire le nombre ne sont qu'au nombre de 15, le bit numéro 16 indiquant le signe du nombre (négatif si le bit 16 vaut 1, positif si le bit 16 vaut 0). Rappelons qu'en binaire, pour passer d'un nombre négatif à sa valeur absolue, il faut prendre son complément à 1, puis ajouter 1. Ainsi, pour connaître la valeur du nombre négatif 1111110011000011 (négatif, car le bit 16 est à 1), il faut d'abord compléter à 1 chaque bit.

La démarche est la suivante :

valeur d'origine	1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1
complément	0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0
	+
ajouter 1	1
	1 1 0 0 1 1 1 1 0 1

On obtient : $1100111101 = 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 829$.

Les nombres réels peuvent être mémorisés en simple ou double précision. En simple précision, on utilise 32 bits et le nombre peut aller de 1×10^{-38} à $1 \times 10^{+38}$, avec 7 chiffres significatifs.

Par exemple, le nombre $135.794397 \times 10^{25}$ est en réalité tronqué à 135.7943×10^{25}.

Les sept chiffres significatifs sont utilisés pour les calculs, mais à l'affichage on n'en retient que 6.

La simple précision est indiquée par le symbole !. Par exemple, 1.35 E2 et 135! sont deux notations du même nombre en simple précision.

Pour accroître le nombre de chiffres significatifs, et donc la finesse des calculs, on emploie la double précision. Dans ce format, le nombre sera représenté sur 64 bits, et pourra varier de 1×10^{-38} à $1 \times 10^{+38}$, avec 16 chiffres significatifs (15 à l'affichage).

On symbolise un nombre en double précision comme un nombre en simple précision, en remplaçant le « E » par un « D ».

Le nombre 135.794397E25, en simple précision perd ses derniers chiffres (9 et 7) ; écrit ainsi, 135.794397D25, il est en double précision et on ne perd aucun chiffre.

Les nombres en simple ou double précision sont affichés avec un chiffre de moins que dans le format de stockage en mémoire, car le Basic effectue automatiquement un arrondi à la valeur la plus proche.

Le symbole D définit un nombre en double précision sous forme exponentielle ; si l'on ne veut pas utiliser l'exponentiation, on doit se servir du symbole # (dièse).

Ainsi, 372.0, en simple précision, s'écrit 372.0 # en double précision.

Ici, les deux notations 372.0 et 372.0 #, sont identiques (aucun chiffre ne déborde du format en simple précision) ; cependant, si dans le programme, des calculs associent ces nombres et des valeurs en double précision, il est conseillé d'utiliser une notation homogène.

Le tableau en haut de la page suivante résume les différents formats de représentation des nombres en machine.

En résumé, les différents types de constantes numériques rencontrées en Basic sont :

PRECISION DE LA REPRESENTATION DES NOMBRES

Type	Espace mémoire	Valeurs limites	Nombre de chiffres significatifs	Nombre de chiffres affichés	Exemple
Entier	2 octets	$-32768 \div +32767$	5	5	21743
Simple précision	4 octets	$\pm 10^{-38} \div \pm 10^{+38}$	7	6	121.3E7
Double précision	8 octets	$\pm 10^{-38} \div \pm 10^{+38}$	16	15	275.8D9 127.41 #
Hexadécimal	2 octets	0 ÷ F F F F			&H57
Octal	2 octets	0 ÷ 177777			&75

Illustrons cela par quelques exemples :

Valeur d'entrée	Signification
42745 %	Erreur. Un nombre entier ne peut être supérieur à 32767.
&12*5	Le résultat est 50 car le nombre octal &12 correspond à 10 décimal.
&12 + &H12	Résultat : 28 (&12 = 10 &H12 = 18 ; le symbole H indique que le nombre est codé en hexadécimal.
1357.92486E21	Les deux derniers chiffres (8 et 6) sont perdus car le nombre est en simple précision.
1357.92486 #	Tous les chiffres sont conservés car le symbole # indique la double précision.
13.5792486D2	Représente la même valeur que précédemment en notation exponentielle.
579.3E41	Erreur. L'exposant en simple précision doit être compris entre −38 et +38.
579.3D41	Représentation exacte du nombre précédent.
157.2!	Le symbole ! indique une valeur en simple précision.
157.2	Autre représentation du nombre précédent. En l'absence du symbole #, le nombre est en simple précision, même si le symbole ! manque.

Constante	Symbole	Exemple
Entière	%	127 %
Exponentielle	E	127E2 = 12700
Double précision	D	127D2 = 12700
Double précision	#	12700 #
Octale	&	&21
Hexadécimale	&H	&H21

Variables numériques

Les variables sont des grandeurs, représentées par un nom symbolique, dont la valeur peut changer en cours d'exécution d'un programme.

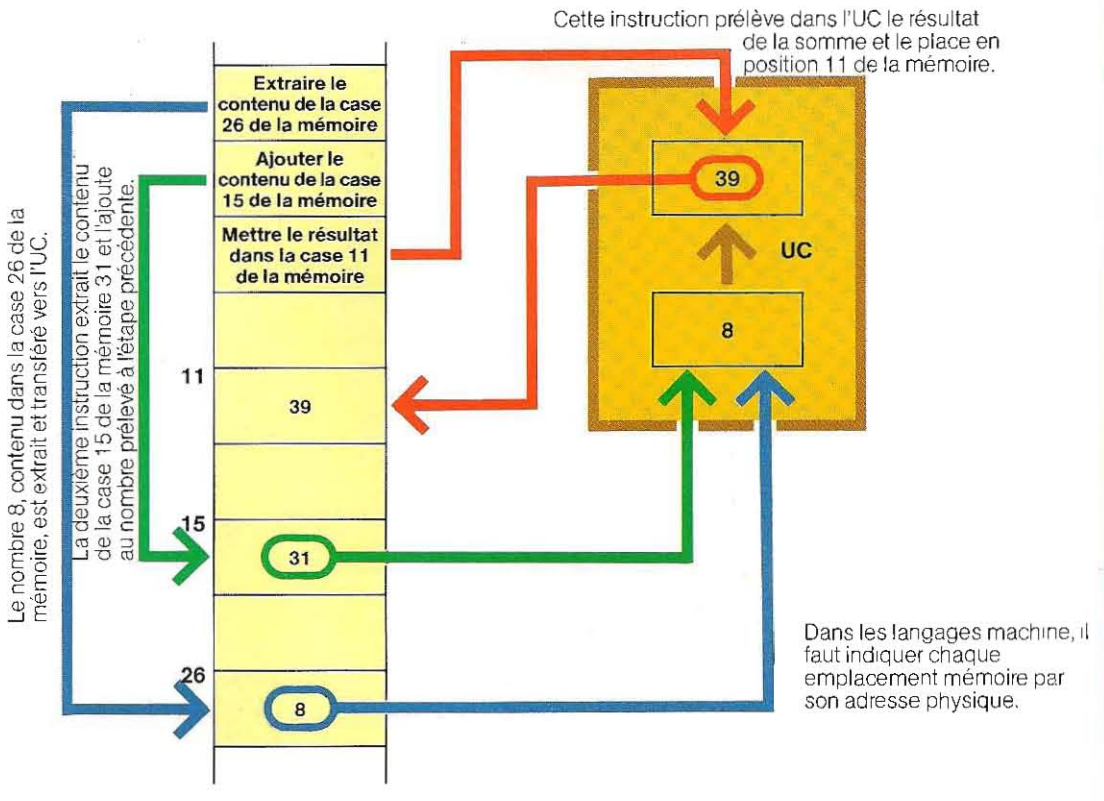
Dans les versions les plus simplifiées du Basic, les noms de variables ont deux caractères au maximum.

Des noms de longueur supérieure à deux caractères sont tronqués automatiquement (troncature automatique) et réduits à leurs deux premières lettres.

Ils risquent alors d'être confondus avec des noms différents mais commençant par les deux mêmes lettres.

Par exemple, les noms MOIS et MOYENNE, définissant deux variables différentes sont tous deux réduits, dans les versions simplifiées du Basic, au terme MO ; lors de l'exécution du programme, les deux variables prendront les mêmes valeurs, amenant à de graves erreurs dans les calculs.
Le nom d'une variable est équivalent à une adresse mémoire. La mémoire de la machine contient toutes les instructions des valeurs sur lesquelles elles opèrent : dans une première zone, résident les codes associés aux instructions, dans la seconde, les valeurs. Pour indiquer la valeur à utiliser il faut fournir l'adresse (position de la case mémoire qui la contient). Par exemple, pour additionner les nombres 31 et 8, les instructions (voir schéma ci-dessous), doivent faire référence aux adresses mémoire contenant ces valeurs (respectivement 15 et 26) ; de même, il faudra donner l'adresse où doit être stocké le résultat. Dans les langages de haut niveau, les

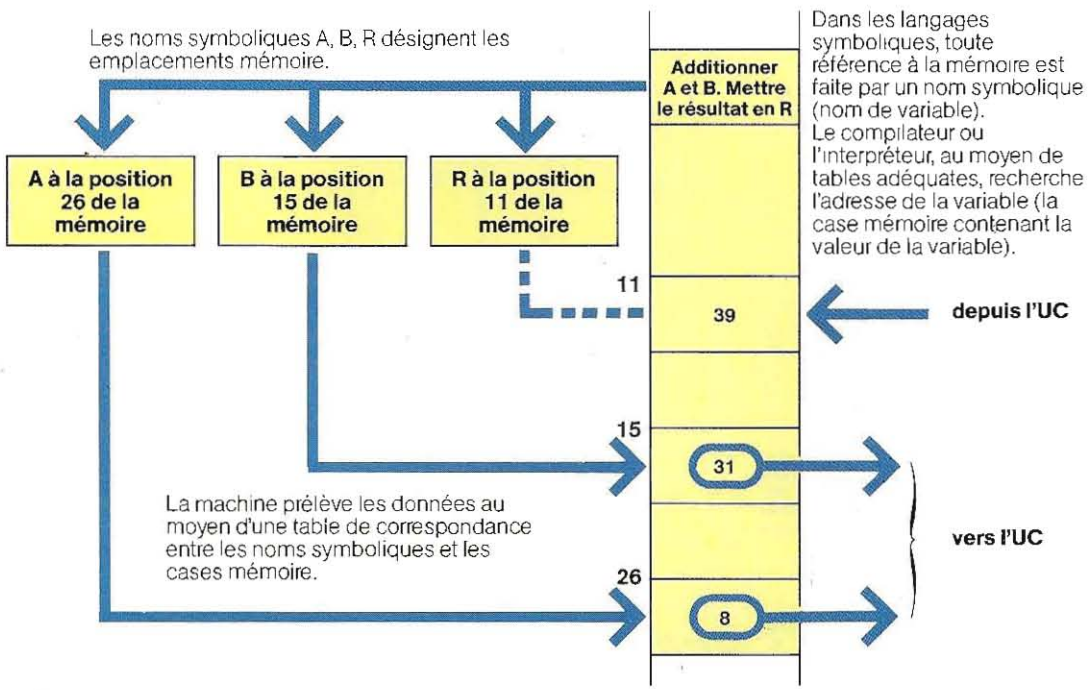
REFERENCE MEMOIRE PAR ADRESSAGE DIRECT



adresses sont représentées par des noms symboliques (A, B, R, dans notre exemple) et le système crée une **table de correspondance** associant à chaque nom symbolique du programme une adresse en mémoire.
Le programmeur, sans se préoccuper des positions mémoires des données, désignera ces dernières par leur seul nom symbolique. L'interpréteur Basic prendra en charge la recherche d'adresses.
En Basic (comme dans d'autres langages évolués) l'addition décrite précédemment s'écrit simplement R = A + B. L'interpréteur identifie les symboles + et =, et traduit l'instruction (R = A + B) en une séquence appropriée d'instructions en format machine, et ce en recherchant les adresses des variables dans la table de correspondance (voir schéma ci-dessous).
Les variables peuvent être, comme les constantes numériques :

- **entières** (symbole % accolé au nom)
- **en simple précision** (symbole !)
- **en double précision** (symbole #).

REFERENCE MEMOIRE PAR NOMS SYMBOLIQUES



L'occupation mémoire et la précision (le nombre de chiffres significatifs) sont les mêmes que pour les constantes numériques. On attribue sa valeur à une variable par une instruction d'assignation, (A = 21, B = 185, etc.), ou bien par un calcul (A = 3 * B + 2).
En début de programme, toutes les variables sont initialisées à zéro et conservent cette valeur tant qu'elles ne sont pas utilisées. C'est ce que figure le bloc 500 de l'organigramme de la page 336, qui remet à zéro les variables Surface totale et Compteur.
Cette réinitialisation est automatiquement exécutée au début du programme mais généralement il est bon de l'insérer en vue de futures modifications dans le programme.
En l'absence du module 500, si on modifiait (par ex.) le programme par une boucle, pour calculer successivement deux surfaces, le second calcul serait faux du fait de l'absence de remise à zéro. La première fois, au démarrage du programme, la mise à zéro serait faite automatiquement ; mais la seconde fois, les valeurs se cumuleraient aux précédentes.

Exemples de variables

R! = 356.8 — La variable R, définie en simple précision (symbole !), reçoit la valeur numérique 356.8.

A! = 37E20*8E20 — Faux. La variable A est définie en simple précision alors que le résultat du calcul ne peut être contenu que dans une variable en double précision :
37E20 = 37×10^{20}, 8E20 = 8×10^{20} et le produit $37 \times 10^{20} \times 8 \times 10^{20}$ donne 296×10^{40}, soit 296E40 :
L'exposant E40 ne peut être accepté en simple précision (le maximum étant 38).

B! = 5 / 2E9 — Faux. Le résultat de 5 divisé par 2E9 (c'est-à-dire 2000000000) est trop petit pour être accepté en simple précision (7 chiffres significatifs qui seront tous nuls).

C% = 721.3 — Faux. Le symbole % définit la variable C comme entière et il ne peut donc contenir de valeurs décimales.

C! = 721.3 — Exact. On peut également utiliser la notation C = 721.3, dans la mesure où l'absence de symboles implique la simple précision.

R = 1240.2145# — Faux. A une variable R en simple précision on ne peut affecter un nombre en double précision (symbole #) car certains chiffres du nombre sont perdus.

Conversion des différents types de variables

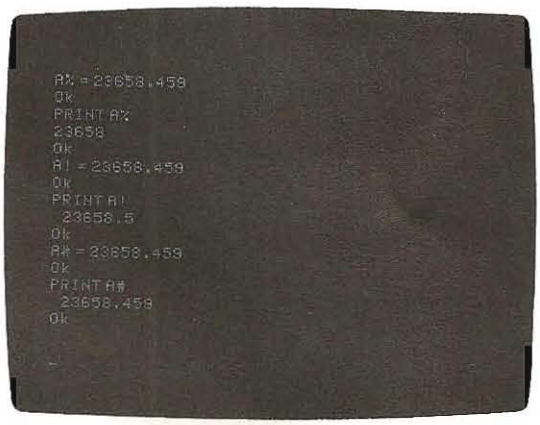
Au cours de l'exécution d'un programme, il peut s'avérer nécessaire de convertir une variable d'un type à un autre. C'est durant cette conversion que sont effectués des arrondis ou des troncatures de valeurs numériques. Les conversions ont lieu lors :

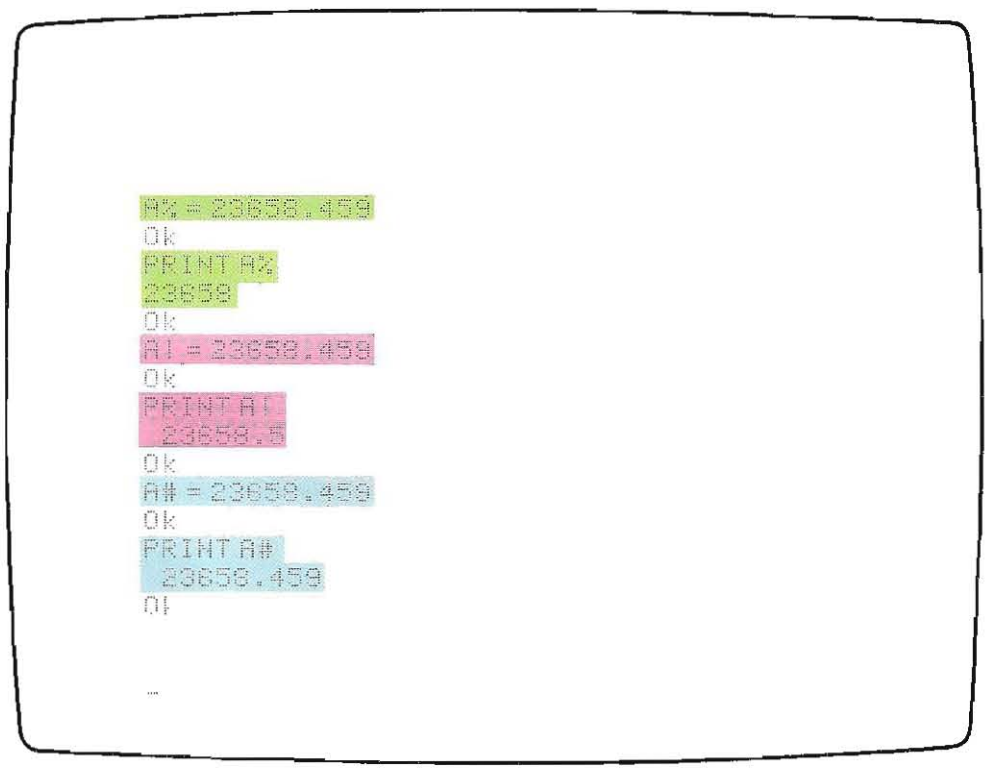
- d'instructions d'affectation : A = 275
- des calculs : A = 3.21 * 75.8 + B#
- d'opérations logiques : (5.2 + 4) AND (3.7 + 2)

Arrondis. Dans les procédures de calcul automatique, l'exactitude des résultats numériques est fortement influencée par les arrondis que la machine exécute systématiquement sur les données et les résultats eux-mêmes.
Aussi est-il essentiel de connaître, a priori, l'ordre de grandeur de l'erreur à laquelle on peut s'attendre.
La première source d'erreur réside dans l'arrondi exécuté par la machine lors de la phase d'affectation des constantes. On la minimi-

PRECISION DES VARIABLES

Sur l'écran on voit différentes assignations typiques pour une même valeur A, chacune étant suivie par une phase d'édition.





La variable A est entière.

La variable A est définie en simple précision.

La variable A est définie en double précision. Cette phase d'édition met en évidence un nombre de chiffres significatifs supérieur aux cas précédents, bien qu'une même valeur numérique ait été attribuée.

sera en utilisant la précision la plus appropriée.

Arrondis dans les affectations. La valeur numérique est mémorisée selon le format défini pour la variable.
Par exemple :

A! = 152.6D3 La valeur 152.6E3 est affectée à la variable déclarée en simple précision, et les chiffres au-delà du 7e sont perdus (par exemple 1359.743298D4 devient 1359.743E4).

A% = 28.3 Dans la variable entière A%, seule est transférée la partie entière du nombre ; le résultat est A% = 28.

A# = 562.5E4 La variable est en double précision (#) alors que le nombre est en simple précision : les chiffres significatifs non explicitement déclarés dans le nombre sont mis à zéro.

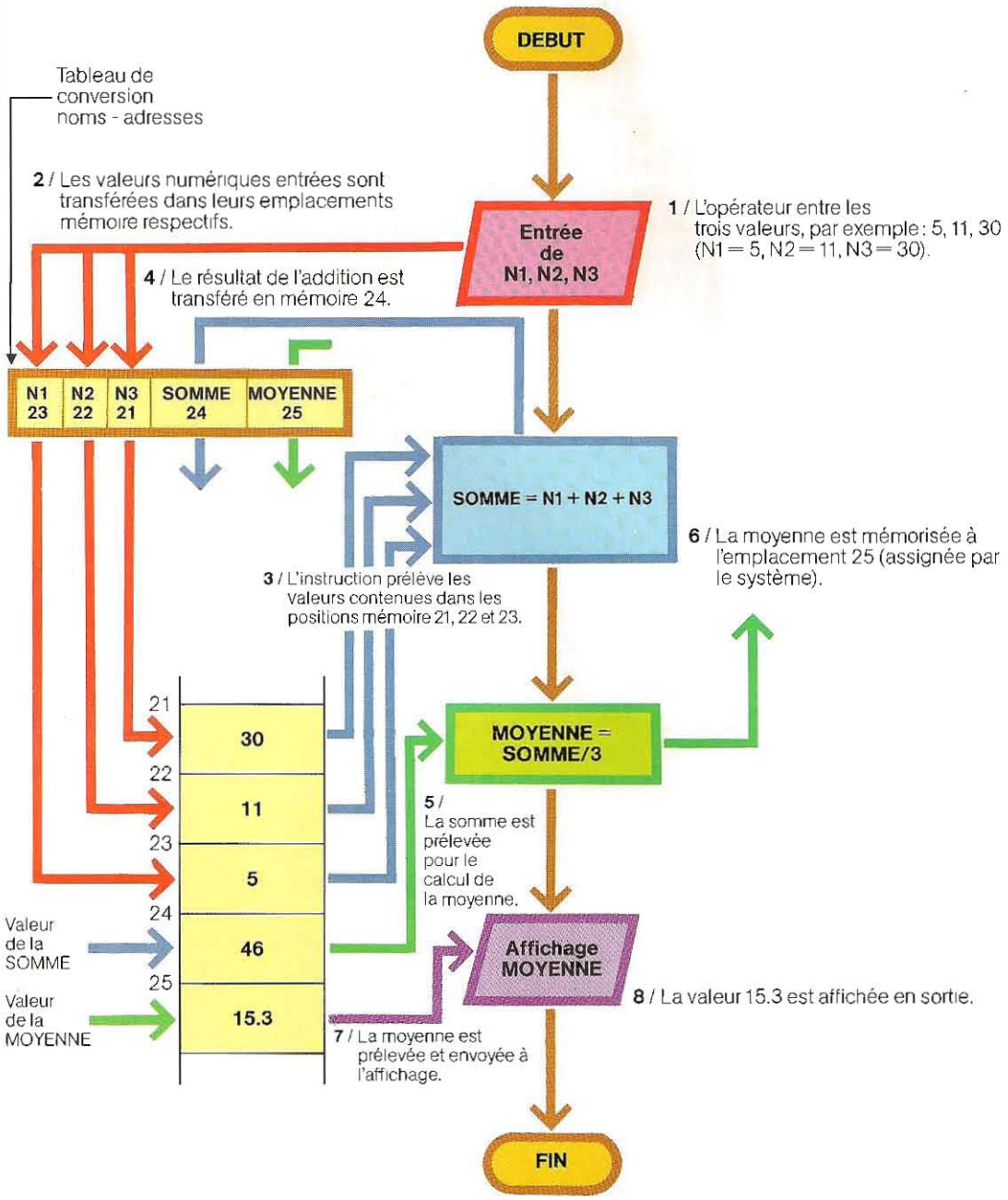
A% = 28.7 La décimale étant supérieure à 5, l'arrondi se fait à la valeur supérieure. Le résultat est A% = 29.

Arrondis dans les calculs. Dans les expressions, tous les opérandes sont évalués avec la même précision, à savoir, la plus haute en cours. Par exemple, l'expression A# = (7.56! + 8.26E4)/121D7 est calculée comme si tous les nombres étaient en double précision, comme l'opérande 121D7. Le résultat aussi est mémorisé en double précision.
En revanche, l'expression A! = (7.56! + 8.26E4)/121D7 (égale à la précédente) mémorisée en simple précision, celle de la variable A!.

En résumé :

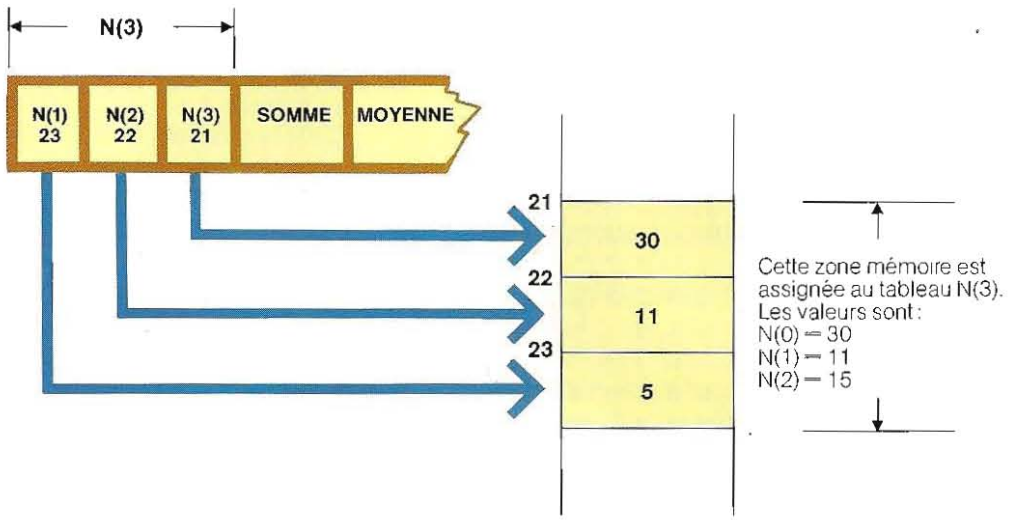
SI L'OPERATEUR VEUT...	IL DOIT ENTRER...
passer sous Basic	MBASIC (ou BA, ou autre)
définir une variable entière	le nom de la variable suivi du symbole % (exemple : A%, VALEUR%).
définir une variable en simple précision	le nom de la variable suivi du symbole ! (exemple : C!).
définir une variable en double précision	le nom de la variable suivi du symbole # (exemple : R #).
définir un nombre entier	la valeur du nombre sans décimales
définir un nombre en simple précision	la valeur du nombre suivie du symbole ! (exemple : 1265.3! ou, en notation exponentielle, 12.753E2).
définir un nombre en double précision	la valeur du nombre suivie du symbole # (exemple 1275.3 #, ou en notation exponentielle, 12.753D2).
définir une fonction	DEF FNX = ... où X est le nom de la fonction (une lettre).
déterminer le reste de la division de deux nombres	reste = Premier nombre MOD Second nombre, exemple 7 MOD 3
faire un calcul immédiat	PRINT calcul, exemple PRINT (7 + 3)/4
affecter une valeur à une variable	A = 7, TOTO = 121.3.
passer sous système d'exploitation	SYSTEM

ADRESSAGE MEMOIRE PAR PLUSIEURS NOMS SYMBOLIQUES



Mécanismes de traduction entre les noms symboliques des variables (N1, N2, N3, SOMME, MOYENNE) et les emplacements mémoire contenant les valeurs numériques de ces variables.

ADRESSAGE MEMOIRE PAR UN SEUL NOM SYMBOLIQUE



Plutôt que trois variables N1, N2, N3, on peut définir un seul nom N les contenant, (occupant 3 emplacements mémoire).
Le tableau symbolisé par N(3), a une « dimension de 3 ».

Arrondis dans les opérations logiques. Les opérandes qui apparaissent dans une expression logique sont convertis en nombres entiers. Par exemple, l'expression (5.2 + 4) AND (3.7 + 2) est évaluée 9 AND 6 (3.7 + 2 = 5.7 arrondi à 6). Si la conversion génère un nombre hors de l'intervalle − 32768, + 32767 il y a dépassement de capacité (overflow).

Les variables structurées

En programmation, on distingue deux types de variables : les variables simples et les variables structurées. Dans le second cas, le logiciel du système offre la possibilité de se référer à un ensemble de données distinctes, tout en utilisant un nom commun. Cela permet de réserver pour une variable structurée un certain nombre d'emplacements mémoire, où seront mémorisées, dans l'ordre, les composantes de la variable. Les variables structurées les plus simples, existant en Basic, sont les **tableaux** (array) et les **chaînes de caractères** (string).

D'autres types de structures plus complexes seront exposés par la suite.

Tableaux. Par ce terme, on désigne un ensemble de zones mémoire auxquelles on fait référence par le même nom symbolique. Imaginons un programme calculant la moyenne de trois nombres introduits au clavier.
Les fonctions à exécuter sont :

- lecture des trois nombres ;
- calcul de leur moyenne (somme des trois nombres/3) ;
- affichage du résultat.

Le schéma de la page 346 représente l'organigramme correspondant, en désignant les trois nombres par N1, N2, N3. Ces noms symboliques sont associés à trois zones mémoire distinctes. A la saisie (entrée) des valeurs numériques à affecter aux trois variables, le système réserve des adresses mémoire et y stocke les valeurs entrées au clavier.

Les trois variables N1, N2, N3 peuvent également être appelées par le nom unique N(3), le symbole (3) indiquant qu'à ce nom correspondent trois emplacements mémoire (voir schéma de la page 347). La variable N(3) ainsi dimensionnée peut contenir simultanément plus d'une valeur (ici, trois). Pour faire référence à l'une d'elles, il faut indiquer son numéro. La numérotation interne des tableaux partant de zéro, on a :

N(0) = premier élément du tableau ; correspondant à N1 ;
N(1) = deuxième élément, correspondant à N2.
N(2) = troisième élément, correspondant à N3.

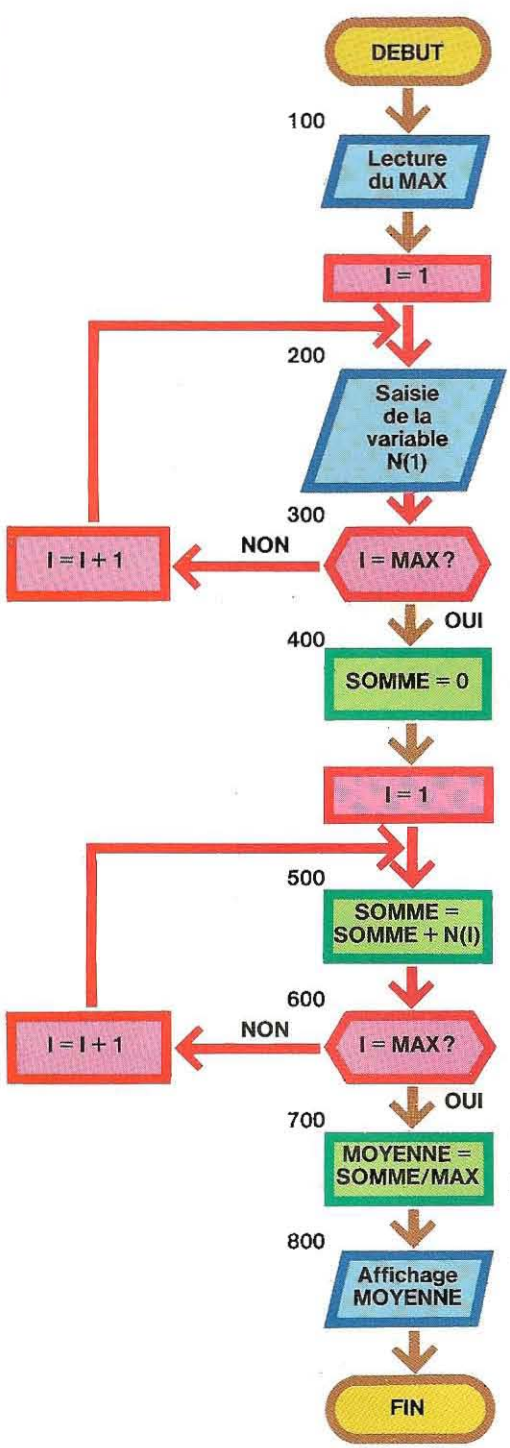
La notation N(3) sert, lors de la définition de la variable, à indiquer au système qu'il existe trois valeurs (toutes sous le nom N), alors que les symboles N(0), N(1), N(2) sont utilisés pour extraire ou écrire chacune des valeurs qui constituent le tableau N(3).
Appeler le premier élément d'un tableau par le nombre 0 peut sembler contraire à notre logique, c'est pourquoi, en Basic, une instruction est prévue, qui modifie la base de numérotation en la faisant débuter à 1.
Ainsi les éléments du tableau précédent deviennent N(1), N(2), N(3).
Pour faire référence à un élément d'un tableau, on peut « indiquer » le paramètre qui le repère. Par exemple, l'élément N(2), qui vaut 11, peut être appelé N(1) avec 1 = 2. En faisant varier l'indice 1 (1 = 1, 2, 3), on appelle tous les éléments du tableau. Cette méthode, très utilisée, permet de « paramétrer » les programmes.
Dans le calcul de la moyenne (exemple précédent) les instructions utilisées sont :

SOMME = N1 + N2 + N3
MOYENNE = SOMME/3
dans le cas de variables séparées.
SOMME = N(1) + N(2) + N(3)
MOYENNE = SOMME/3
dans le cas d'un tableau.

Si le nombre des variables change, (si par exemple il passe à 8), ces instructions ne sont plus valables. Il faut pratiquement récrire le programme. Avec un tableau indicé, le programme est paramétré et s'adapte sans aucune modification aux différentes applications. Le schéma de la page 349 présente l'organigramme (paramétré) du calcul de la moyenne d'un nombre quelconque de valeurs.
La limite de ce nombre est celle de la dimension de la mémoire fixée pour le tableau N. Avec N(3000), on peut calculer la moyenne d'un nombre de variables compris entre 2 et 3 000 (on ne calcule pas la moyenne d'une seule variable) ; le programme s'adapte alors automatiquement.

Les chaînes de caractères. Une chaîne est une suite de caractères alphanumériques (en code ASCII) délimitée par des guillemets. Ainsi, « NOM, 1, 3, 5 » est une chaîne qui contient les caractères NOM, 1, 3, 5. Les nombres figurant dans les chaînes de caractères ne sont pas représentés en format numérique. A moins de les convertir, on ne peut donc s'en servir pour effectuer des calculs.
Les chaînes de caractères peuvent être des constantes ou des variables, la distinction étant la même que pour les constantes et les variables numériques. Le symbole $ (dollar) informe le système que tel nom de variable désigne une chaîne de caractères (NOM$, A$, par exemple). La longueur maximale d'une chaîne, variable ou constante, est de 255 caractères. Si, par exemple, A$ = « ADC », la chaîne de nom A$ contient les caractères A, D et C. On appelle « chaîne vide » une chaîne qui ne contient aucun caractère. Par exemple lorsque A$ = « », le nom A$ désigne une chaîne vide. Notons que la chaîne A$ = « » n'est pas vide puisqu'elle contient un caractère, le blanc, ou espace, (en anglais, blank ou space).
La longueur d'une chaîne varie avec le nombre de caractères qu'elle contient. La chaîne définie en un point du programme par NOM$ = « Ravel », contient 5 caractères. Si, par la suite, on lui affecte une nouvelle valeur, NOM$ = « Hector Berlioz », elle aura alors une longueur de 14 caractères. Cette caractéristique, propre au Basic, s'appelle l'« allocation dynamique ». Son intérêt est de ne pas obliger le programmeur à définir à l'avance la longueur de chaque chaîne. Cette commodité présente toutefois un risque.
C'est le cas lorsqu'un long programme occupe la quasi totalité de l'espace mémoire

EMPLOI D'UN TABLEAU PARAMETRE POUR LE CALCUL D'UNE MOYENNE



Le programme demande le nombre de valeurs à saisir pour calculer leur MOYENNE. Ce nombre (MAX) ne doit pas dépasser la dimension prévue pour le tableau [N(3000), et donc MAX < = 3000]. Prenons, par exemple, MAX = 128.

Dans cette boucle, le programme demande les valeurs des variables. Il commence à I = 1 et affecte donc la valeur saisie à la première variable [N(I) — N(1) quand I = 1]. La boucle s'achève quand I = MAX = 128. Il faut donc absolument entrer les 128 valeurs annoncées.

Remise à zéro de SOMME.

La boucle cumule les 128 valeurs dans la zone SOMME. Au départ, on a posé SOMME = 0.
Au premier tour (I = 1), on a donc SOMME = SOMME + N(1) — N(1). Au tour suivant, (I = 2), on a SOMME = SOMME + N(2) = N(1) + (N2).

On additionne ainsi les 128 nombres.

On calcule la moyenne de cette façon :

$$\text{MOYENNE} = \frac{\text{Somme des valeurs}}{\text{Nombre des valeurs}} = \frac{\text{SOMME}}{\text{MAX}}$$

Affichage du résultat.

Solutions du test 9

1 / La moyenne de trois nombres est le quotient de leur somme par 3. On l'obtient à l'écran par l'instruction :
PRINT (5.2 + 15.21 + 7.43)/3
et sur l'imprimante par :
LPRINT (5.2 + 15.21 + 7.43)/3

---

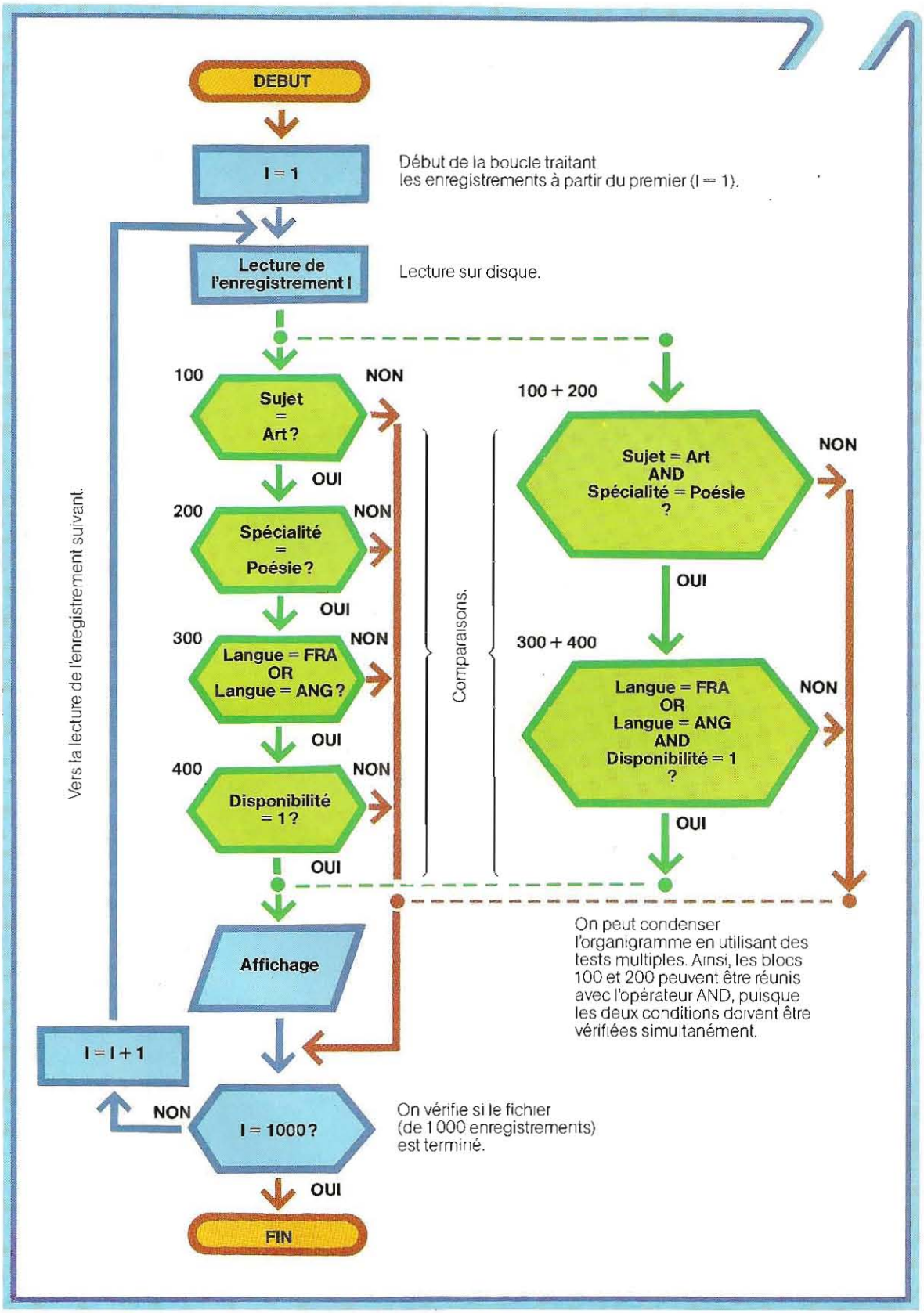
2 / L'opérateur logique AND a une priorité inférieure à celle des opérateurs arithmétiques. C'est pourquoi, dans l'expression 3+2 AND 3×4, les calculs (3+2 et 3×4) sont effectués en premier. On obtient l'expression 5 AND 12. Il faut alors convertir 5 et 12 en binaire, et leur appliquer l'opérateur AND.
5 décimal = 101 binaire
12 décimal = 1100 binaire
5 AND 12 = 0100 = 4 décimal
On a donc 3 + 2 AND 3 × 4 = 4
Dans la seconde expression, les parenthèses modifient l'ordre de priorité. La première opération, 2 AND 3, donne 10 AND 11 = 10 en binaire, soit 2 en décimal. L'expression devient donc :
5 + 2 × 4 = 13

---

3 / Il faut effectuer les opérations suivantes :
1 - lire un enregistrement du fichier de données ;
2 - regarder si le sujet est Art ;
3 - regarder si la spécialité est Poésie ;
4 - regarder si le livre est rédigé en français ou en anglais (FRA OR ANG) ;
5 - vérifier si le livre est en rayon (disponibilité = 1) ;
6 - vérifier si les conditions 2, 3, 4 et 5 sont remplies ; afficher le titre de l'ouvrage ;
7 - lire un nouvel enregistrement (si le fichier n'est pas terminé).
L'organigramme de cette succession de traitements se trouve à la page 351.

---

4 / La longueur d'un enregistrement correspond à la somme des caractères des différentes zones, soit, pour cet exemple, 30 + 10 + 10 + 3 + 1 = 54 caractères (rappelons qu'un caractère correspond à un octet). Si l'on fixe cette longueur à 70 pour conserver une marge, un disque d'un Mo (1 000 000 de caractères) pourra contenir 1 000 000 / 70, soit 14 285 enregistrements. Il est donc théoriquement possible de mémoriser les données concernant 14 000 volumes.
Toutefois, il ne faut pas surestimer les possibilités du système : les temps de traitement seraient très longs pour un tel fichier. L'accès au disque requiert, en moyenne, 0.1 seconde par enregistrement (notamment pour le positionnement de la tête). La lecture de 14 000 enregistrements dure donc environ 25 minutes, auxquelles il faut ajouter le temps nécessaire à l'exécution des instruction proprement dites. On peut estimer à 30 minutes la durée totale du traitement.
En outre, l'impression (pour le cas où on voudrait imprimer la liste de ces titres) est souvent la phase la plus longue. Un bon modèle d'imprimante (micro-ordinateur) a une vitesse de 120 lignes à la minute. Si on décide de faire figurer dans le listing seulement 10% des ouvrages, on imprimera 1 400 lignes (une par ouvrage), ce qui représente 1 400 / 120 = 11 fois 1 minute, soit 110 minutes. C'est-à-dire qu'il faudrait presque 2 heures pour imprimer toutes les données du fichier. La durée de l'exécution de ce programme peut donc varier entre 45 minutes et 3 heures.


DEBUT
I = 1
Début de la boucle traitant les enregistrements à partir du premier (I = 1).
Lecture de l'enregistrement I
Lecture sur disque.
100
Sujet = Art ?
NON
OUI
200
Spécialité = Poésie ?
NON
OUI
300
Langue = FRA OR Langue = ANG ?
NON
OUI
400
Disponibilité = 1 ?
NON
OUI
Comparaisons.
100 + 200
Sujet = Art AND Spécialité = Poésie ?
NON
OUI
300 + 400
Langue = FRA OR Langue = ANG AND Disponibilité = 1 ?
NON
OUI
Affichage
On peut condenser l'organigramme en utilisant des tests multiples. Ainsi, les blocs 100 et 200 peuvent être réunis avec l'opérateur AND, puisque les deux conditions doivent être vérifiées simultanément.
I = I + 1
NON
I = 1000 ?
On vérifie si le fichier (de 1 000 enregistrements) est terminé.
OUI
FIN
Vers la lecture de l'enregistrement suivant.

disponible et que plusieurs chaînes de caractères sont prévues en entrée (les noms et les adresses d'un répertoire, par exemple). Tant que les données saisies n'occupent pas trop de place en mémoire, l'exécution se déroule normalement. Mais si l'opérateur introduit une donnée trop longue dont l'allocation demande plus de place qu'il n'en reste en mémoire, l'exécution s'interrompt. Pour éviter ces incidents, on contrôle le nombre de caractères entrés, et sans interrompre le programme, on signalera toute longueur excessive à l'opérateur, qui abrégera alors la donnée.

Il est possible de dimensionner les chaînes, comme les variables numériques. En écrivant NOM$(10), on réserve l'espace mémoire nécessaire à 10 chaînes, chacune d'une longueur comprise entre 0 (chaîne vide) et 255 caractères. Chaque caractère occupant un octet en mémoire, l'espace mémoire total occupé varie donc de 0 (toutes les chaînes sont vides) à 10 × 255 = 2 550 octets. Mais il est impossible, à cause de l'allocation dynamique, de déterminer à l'avance l'espace réellement occupé.

On peut appliquer aux chaînes quelques opérateurs :

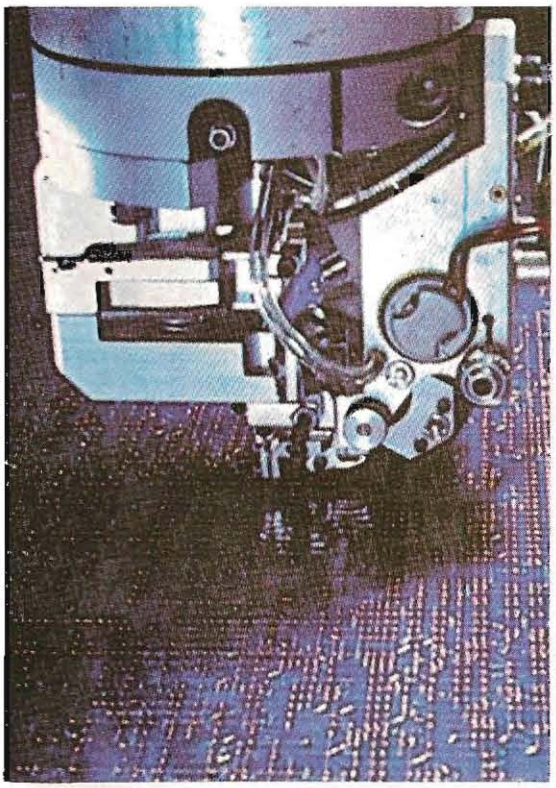
- **l'opérateur arithmétique de somme** (+), qui permet la concaténation de deux ou plusieurs chaînes. Ainsi, lorsqu'on pose : A$ = « Hector », B$ = « Berlioz », C$ = B$ + A$, C$ contient « Berlioz Hector ». Les chaînes B$ et A$ ont été mises bout à bout pour constituer une troisième chaîne : elles ont été concaténées.
  Sur certaines machines, le symbole de concaténation est & ; on écrira alors C$ = B$&A$.
- **tous les opérateurs logiques** (=, <>, <, >, <=, >=). Ainsi la condition NOM$ < « Berlioz » est vérifiée par toutes les chaînes dont la valeur lexicographique est inférieure à Berlioz (Bach, par ex.) et elle est fausse pour toutes les autres (Brahms, ou Mendelsshon, etc.). La valeur d'un caractère est fonction de sa place dans l'alphabet.

Les blocs 300 et 500 (organigramme de la page 331) constituent un exemple d'utilisation des chaînes.

Dans le bloc 300, on doit vérifier l'égalité entre le nom lu dans le fichier et la valeur « Martin ». Le nom est une variable de type chaîne (son contenu varie au cours du programme) ; on peut la désigner par NOM$. « Martin » est, au contraire, une constante qui sera écrite entre guillemets dans les expressions. La condition d'égalité s'écrira, tout simplement, NOM$ = « MARTIN ».

De même, dans le bloc 500, la profession constitue une variable qu'on désignera par PROF$ tandis que les valeurs Employé et Enseignant sont des constantes. La relation s'écrit donc :

PROF$ = « EMPLOYE » OR PROF$ = « ENSEIGNANT ».



B. Coleman/Marka

Robot pratiquant des soudures sur un panneau électronique.

Les portions de chaînes. Une portion de chaîne représente un groupe de caractères extaits d'une chaîne. Tout à l'heure, en écrivant A$ = « Hector », B$ = « Berlioz », C$ = B$ + A$, nous avons formé la chaîne C$ à partir de A$ et B$. L'inverse, c'est-à-dire l'extraction des 7 premiers caractères de la chaîne C$, puis les suivants, nous fournira respectivement les deux parties de chaîne.

En Basic, des instructions permettent d'extraire d'une chaîne le nombre désiré de caractères, à partir d'une position donnée.

Les commandes

Tout langage possède deux types d'« ordres » : les commandes et les instructions. Les premières déclenchent des fonctions générales du système, telles l'exécution ou l'interruption d'un programme. Les instructions représentent le codage des opérations à effectuer, leur ensemble constituant le programme.
En Basic, les commandes sont plus nombreuses que dans les autres langages et presque toutes sont utilisables au cours même du programme. L'emploi des commandes est beaucoup plus délicat dans les autres langages.
On peut les répartir en quatre groupes, selon les opérations qu'elles effectuent :

1 / écriture et sauvegarde de programmes ;
2 / visualisation à l'écran et impression de listages ;
3 / exécution et interruption de programmes ;
4 / fusion.

Commandes d'écriture et de sauvegarde des programmes

Les instructions d'un programme Basic sont numérotées en ordre croissant, avec un écart arbitraire entre deux numéros. Ainsi, nous pourrions numéroter les instructions d'un programme de 5 en 5 à partir de 10 (10, 15, 20, 25, etc.) et celles d'un autre programme de 10 en 10 (10, 20, 30, etc.). En écrivant ces programmes l'un après l'autre, sans éteindre la machine, on obtiendra un curieux mélange d'instructions : les instructions paires du second programme remplaceront les instructions de même numéro appartenant au premier programme, tandis que les instructions impaires de celui-ci (15, 25, etc.) seront inchangées.
La commande NEW (nouveau) évite ces confusions entre le programme en cours et les précédents (encore présents en mémoire), en remettant à zéro toute la zone utilisateur de la mémoire. L'introduction de la commande NEW pendant l'édition d'un programme entraîne sa destruction.
Une fois la mémoire réinitialisée, les instructions écrites constitueront un nouveau programme.
On peut se dispenser d'écrire le numéro de chaque ligne en passant en numérotation automatique, avec la commande AUTO, dont la syntaxe est :

AUTO ligne de départ, pas

Le premier terme (« ligne de départ ») désigne le numéro de la première ligne que l'on va écrire, et, le second (« pas »), le pas de numérotation. Ainsi, AUTO 120, 5 déclenche la numérotation automatique des lignes de 5 en 5 à partir de 120. La production automatique des numéros de ligne peut aboutir parfois à un numéro déjà existant. Il s'affiche alors à l'écran, suivi d'un astérisque, et l'opérateur choisit d'interrompre la numérotation automatique pour conserver l'ancienne ligne, ou de continuer pour la remplacer par une nouvelle.
Lorsqu'on écrit de longs programmes, la commande NEW et la possibilité de numéroter à partir de n'importe quel nombre sont très utiles. Grâce à la commande AUTO, on peut interrompre et reprendre à tout moment l'écriture d'un programme, en faisant repartir la numérotation au numéro immédiatement supérieur au dernier utilisé.
Le programme doit être sauvegardé sur disquette ou sur bande magnétique car la plus brève coupure de courant, ou même, parfois, une simple variation de tension, peut entraîner sa perte (la mémoire de l'ordinateur est volatile). Pour une sauvegarde sur disque, la commande est :

SAVE « nom de l'unité de disque :
nom du programme »

Si le code SAVE est commun à la plupart des machines, les paramètres (nom de l'unité et du programme) dépendent du système d'exploitation. Ainsi, pour mémoriser le programme résident en mémoire dans un fichier de nom ESSAI, on utilisera, par exemple, en CP/M, la commande SAVE « A:ESSAI », et, en PCOS, SAVE « O:ESSAI ».
Le système d'exploitation CP/M détectera que le programme est écrit en Basic, et le mémorisera avec la mention BAS. Si l'on demande l'affichage du répertoire du disque

par la commande DIR (voir le chapitre sur les systèmes d'exploitation), on obtiendra le nom complet du programme : ESSAI.BAS.
En fait, la commande SAVE est encore plus complexe car on peut mémoriser les programmes sous deux formes : en code ASCII et en binaire condensé.
En code ASCII, lettres et chiffres constituant les lignes du programme sont mémorisés en tant que symboles ASCII, c'est-à-dire comme de simples caractères alphanumériques, tandis que, dans le second cas, tous les caractères sont codés en binaire.
Tous les caractères frappés au clavier sont codés en ASCII mais, pour être exécutable, une instruction en ASCII doit d'abord être traduite en binaire par l'interpréteur. Un programme peut être mémorisé indifféremment dans l'un ou l'autre code, et l'interpréteur décide au moment de l'exécution si la traduction est (ou non) nécessaire.
Une mémorisation sous forme ASCII devra être spécifiée par la lettre A, tandis qu'en binaire, il n'y a rien à indiquer. En CP/M, la commande SAVE peut donc prendre deux formes :

SAVE « A:ESSAI »,A
mémorisation en ASCII
SAVE « A:ESSAI »
mémorisation en binaire

Il s'avère parfois nécessaire de protéger des programmes. En indiquant P au lieu de A, le programme sera sauvegardé en binaire mais on ne pourra plus le lister ou le corriger. Il est impossible d'annuler cette protection, aussi est-il conseillé de conserver une copie non protégée des programmes. Le binaire, plus compact que le code ASCII, convient généralement mieux à la sauvegarde. Cependant, certaines opérations comme la compilation, nécessitent une version en code ASCII.
Voyons maintenant en détail l'enchaînement des commandes et des instructions qu'il faut communiquer à la machine, après sa mise sous tension, pour écrire et sauvegarder un programme :

> MBASIC Après la mise sous tension, le système d'exploitation est chargé et envoie le symbole > (prompt). MBASIC commande le chargement de l'interpréteur Basic.

NEW Demande d'effacement du contenu de la mémoire.

AUTO 50, 5 Demande de mise en numérotation automatique de 5 en 5 à partir de 50.

50 A = B + C
55 D = A * 6
60 K = C − B Les numéros de ligne s'affichent dans l'ordre croissant et l'utilisateur entre les instructions du programme (A = B + C, etc.).

CTRL + C Clôture de la phase d'écriture par la frappe simultanée des touches CTRL et C (voir les fonctions de la touche CTRL p. 322). Le MBASIC attend une nouvelle commande.

SAVE « A:PROG−1 »,A Le programme qui vient d'être écrit est enregistré en code ASCII, sous le nom PROG-1, sur la disquette A.

NEW La mémoire est réinitialisée. Il n'y reste aucune trace de ce qui vient d'être écrit.

Le programme PROG-1, ainsi stocké sur disquette, sera rappelé en mémoire pour être exécuté ou modifié. On utilise pour cela la commande LOAD « A:PROG-1 », identique sur tous les systèmes.
Le programme réside alors à nouveau en mémoire, et l'on peut le corriger ou l'exécuter. On dispose de trois commandes de correction : RENUM, EDIT et DELETE. RENUM permet de renuméroter un programme, en partie ou en totalité, avec la syntaxe suivante :

RENUM [nouveau numéro'] [,ancien numéro'] [,pas

ECRITURE ET MEMORISATION D'UN PROGRAMME

Les différentes phases de la procédure s'affichent à l'écran.



```
BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977-1981 (C) by Microsoft
Created: 28-Jul-81
32824 Bytes free
Ok
NEW
Ok
AUTO 50,5
50 A = B + C
55 D = A * B
60 K = C - B
65 IC
Ok
SAVE "A:PROG-12,A
Ok
RENUM 80,55,2
Ok
LIST
50 A = B + C
80 D = A * B
82 K = C - B
Ok
_
```

Le système indique à l'utilisateur qu'il se trouve sous Basic et dispose de 32 824 octets.

La commande NEW (qui, ici, n'est pas nécessaire) réinitialise la mémoire. Le système signale que la commande a bien été exécutée (Ok).

L'opérateur peut désormais taper son programme.

On achève la phase d'écriture avec la commande CTRL + C (↑ C sur l'écran). Cette ligne 65, éliminée par l'interpréteur, ne fera pas partie du programme.

La commande SAVE provoque le stockage sur la disquette A du programme résidant en mémoire. Ici, on a frappé le caractère 2 au lieu des guillemets qui indiquent la fin d'une chaîne, à la suite du nom qu'on voulait donner au programme. Cette erreur est assez courante, car les deux symboles se trouvent sur la même touche et sont sélectionnés par l'utilisation de la touche des majuscules.
La machine n'a pas remarqué l'erreur (c'est une lacune du système), et le programme n'est donc pas enregistré sous le nom désiré.

Ces lignes illustrent l'emploi des commandes RENUM et LIST.

Par « nouveau numéro », on entend celui à partir duquel commencera la nouvelle numérotation.
L'ancien numéro désigne la première ligne à renuméroter et le pas, l'écart entre deux numéros consécutifs de la nouvelle numérotation.
La commande RENUM 80,55,2 renumérote de 2 en 2, à partir de la ligne 55, le programme que nous venons d'écrire.
Les nouveaux numéros de ces instructions seront donc 55, 80 et 82 (le numéro 50 reste inchangé puisque la renumérotation démarre à 55).
Pour corriger une ligne, on peut écrire son numéro, suivi de son nouveau contenu. L'ancienne instruction se trouve ainsi effacée, quelle que soit sa longueur.
Ainsi, en écrivant 55 D = A/6, nous remplaçons l'ancienne instruction 55 (D = A*6) par la nouvelle.
La commande EDIT fait passer le Basic en mode éditeur, et permet ainsi de corriger des lignes du programme sans les réécrire entièrement.

Avec EDIT, il aurait suffi de remplacer en ligne 55 le symbole de multiplication (*) par le symbole de division (/).
Superflu dans des cas aussi simples, ce mode de correction devient un outil très précieux pour la modification de lignes longues ou complexes.
En mode EDIT, nous disposons de cinq commandes, très semblables à celles de l'Editeur du système d'exploitation :

1 / déplacement du curseur le long d'une ligne ;
2 / insertion de caractères ;
3 / suppression de caractères ;
4 / recherche de caractères ou de groupes de caractères ;
5 / remplacement de caractères.

La connaissance approfondie des fonctions d'édition n'est utile que dans des cas particuliers ; à titre d'information, elles sont détaillées dans l'encadré ci-dessous.

Par ailleurs, on utilise en mode EDIT deux touches spéciales : RUBOUT (ou DEL) et ESCAPE.

Principales fonctions du mode EDIT (quelques exemples sont illustrés page 358).

1 / Déplacement du curseur
Le curseur (tiret ou rectangle lumineux qui indique la position où se trouvera le caractère entré) peut se déplacer le long de la ligne à « éditer » (c'est-à-dire à compléter ou à corriger).
On le déplace vers la droite, en appuyant sur la barre d'espacement, ou vers la gauche, avec la touche RUBOUT.

2 / Insertion de caractères dans une ligne
Pour insérer un ou plusieurs caractères dans une ligne, on positionne le curseur au caractère précédent, on frappe la touche I et on tape les caractères à insérer. La touche RUBOUT mettra fin à l'insertion.
Pour ajouter des caractères en fin de ligne, la touche X avance le curseur à la bonne position. L'insertion se déroule automatiquement.
On terminera l'insertion en appuyant sur la touche ESCAPE.

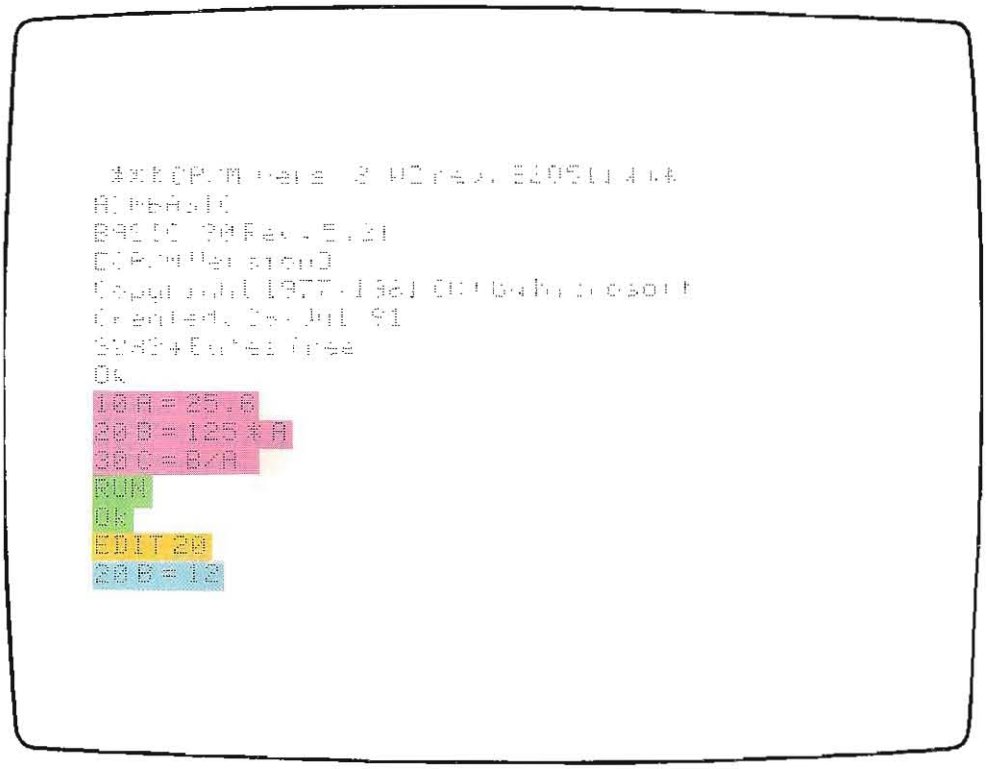
3 / Suppression de caractères
On l'effectue avec la touche D (delete) après avoir positionné le curseur sous le caractère à supprimer ; on efface ainsi les caractères un à un. Cette commande peut se compléter par un facteur de répétition : en tapant 4D, on supprimera quatre caractères à partir du curseur.
Pour effacer tous les caractères à droite du curseur, on utilisera la lettre H.

4 / Recherche de caractères
Elle est déclenchée par la touche S (search). Si l'on tape, par exemple, Sx, la machine cherche la lettre x dans la ligne éditée, à partir de la position du curseur.
L'utilisation de K au lieu de S permet en plus de supprimer le caractère trouvé.

5 / Remplacement de caractères
Si l'on appuie sur la touche C (conversion), le caractère sous lequel se trouve le curseur sera remplacé par celui qu'on entre aussitôt après. Ainsi, Cr remplace l'ancien caractère par la lettre « r ».

UTILISATION DE LA COMMANDE EDIT





Ces lignes constituent le programme entré au clavier.

La commande RUN déclenche l'exécution du programme qui réside en mémoire. Le message Ok indique que l'exécution est terminée.

La commande EDIT 20 affiche à l'écran la ligne 20 du programme.

L'opérateur positionne alors le curseur sous le caractère voulu et apporte, en mode éditeur, les corrections désirées. Lorsqu'il a terminé, il appuie sur la touche de retour chariot ; la nouvelle ligne 20 remplace désormais l'ancienne dans le programme.

EXEMPLES D'OPERATIONS D'EDITION

10 V = A * B − (C/D + 6)

position initiale du curseur → position finale

1 / Déplacement du curseur
d'une position à chaque pression sur la barre d'espacement

---

10 V = A * B − (C/D + 6)

On peut insérer des caractères après avoir frappé la touche I.
I suivi de Z modifie la ligne comme suit

10 V = A * B − (C/D Z + 6)

caractère inséré

2 / Insertion d'un caractère
Commande = I

---

10 V = A * B − (C/DZ + 6)

La touche D supprime le caractère en l'entourant de deux barres de division inversées (\)

10 V = A * B − (C/D \Z\ + 6)

Le caractère Z a été supprimé.
La ligne a retrouvé sa forme première (**1**).

3 / Suppression d'un caractère
Commande = D

---

10 V = A * B − (C/D + 6)

La commande S + déclenche la recherche du caractère +

Le curseur se place sous le caractère cherché

4 / Recherche d'un caractère
Commande = S

---

10 V = A * B − (C/D + 6)

C * remplace par * le caractère +

10 V = A * B − (C/D * 6)

5 / Remplacement d'un caractère
Commande = C

En utilisant la commande EDIT, on doit préciser le numéro de la ligne à corriger : on écrit, par exemple, EDIT 55 pour modifier la ligne 55.
Tous les claviers possèdent la touche ESCAPE (généralement indiquée par l'abréviation ESC).
En revanche, la touche RUBOUT est parfois remplacée par la touche DELETE (DEL) ou encore par la touche curseur comportant le symbole ↰.

La dernière commande de correction des programmes, RUBOUT ou DELETE (qui, en anglais, signifie biffer, rayer), permet de supprimer des lignes, de tel à tel numéro. Ainsi, DELETE 10-28 supprime les lignes 10 à 28 inclus.
Sur certains systèmes, les deux numéros sont séparés par un tiret, alors qu'on utilise toujours la virgule pour les autres commandes (exemple : RENUM 20, 28, 5).
Cette particularité diminue le risque d'utiliser la commande DELETE par erreur.

Wafer, chip and Co.

Le microprocesseur est sans doute le produit le plus remarquable de la technologie du silicium. Il concentre sur une « puce » (ou chip) de silicium une capacité de calcul qu'on aurait obtenue, dans les années cinquante, avec 40 tonnes de matériel. En outre, il se fabrique en grande série, à un coût unitaire de quelques dollars seulement.
Ajoutez au microprocesseur quelques microcircuits de mémorisation, une alimentation, quelques accessoires (clavier, écran, entre autres), et vous obtiendrez un calculateur électronique.

* * *

Mais qu'est-ce qu'une puce ? Que peut-elle faire et de quoi est-elle faite ?
C'est une mince plaquette de silicium à la surface de laquelle s'étend le microscopique dessin du microcircuit, en une succession de couches de silicium pur ou impur (dopé), d'oxyde de silicium et d'aluminium. L'élément fondamental du chip est le transistor, plus précisément le transistor à effet de champ (FET : Field Effect Transistor).
Un tel transistor possède trois connections électriques appelées ***source, drain*** *et* ***porte.***
Le courant peut passer de la source au drain si une tension adéquate est appliquée à l'électrode porte. Le transistor à effet de champ constitue donc un interrupteur électronique commandé par une tension à la porte.
En associant un certain nombre de ces transistors, on réalise les opérateurs logiques simples qui sont à la base du calcul automatique.
Un transistor à effet de champ est formé de deux types de silicium différents, les matériaux « n » et « p ».
Le silicium « n » est dopé par des impuretés (comme le phosphore), qui produisent, au sein du métal, un excès d'électrons. Pour le type « p », d'autres impuretés (comme le bore) créent au contraire un déficit d'électrons. Dans les deux cas, le déséquilibre électrique assure la conductivité du silicium.
Dans le transistor, deux couches de type « p » forment la source et le drain, tandis qu'une électrode appliquée sur la couche intermédiaire de type n, constitue la porte.

En pratique, un microcircuit qui peut comprendre des milliers de transistors à effet de champ, est réalisé sur une plaquette (appelée wafer) d'un type de matériau déterminé (par exemple p).
Par « dopage », on crée des îlots de type opposé, en masquant soigneusement les zones qui ne doivent pas être dopées.

* * *

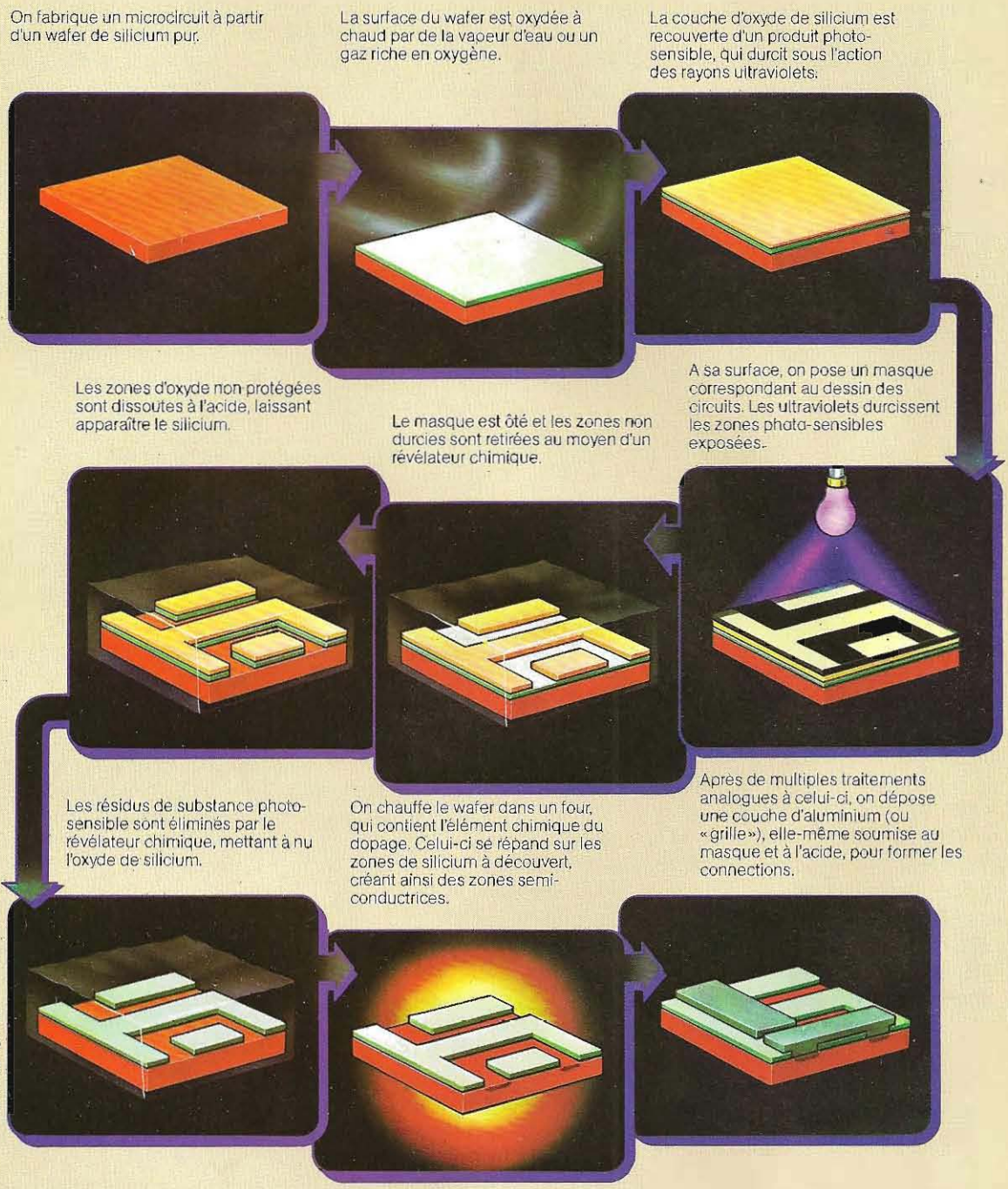
La production d'un nouveau microcircuit passe par plusieurs étapes : spécification du produit par une équipe de conception, dessin des circuits conformément à leur cahier des charges, test d'efficacité au moyen de simulateurs.
Les circuits satisfaisants sont ensuite tracés par un ordinateur graphique, et on attribue aux divers éléments du circuit des couches précises du bloc de silicium. L'ordinateur fournit un plan de contrôle à grande échelle et une bande magnétique codifiée, permettant la réalisation des masques.
Ces derniers servent au tracé du circuit sur les différentes couches du bloc. Autrefois réalisés par procédé photographique et réductions successives d'un plan à grande échelle, les masques sont aujourd'hui dessinés directement par un rayon électronique, commandé par la bande magnétique.

* * *

Comme on l'a mentionné, les microcircuits sont gravés sur des wafers de silicium. Pour préparer les wafers, on ajoute au silicium pur fondu des impuretés du dopage. Une lente extraction transforme le matériau fondu en un gros cristal cylindrique, que l'on sciera de façon à obtenir des éléments circulaires (wafers) d'un diamètre de 100 ou 120 mm. Après un polissage de haute précision, la surface du wafer est prête à recevoir les couches du circuit. Chaque wafer peut contenir jusqu'à 500 microcircuits.

* * *

Une couche du microcircuit s'obtient classiquement de la façon suivante. On crée tout d'abord une couche de bioxyde de silicium (envoi de vapeur d'eau ou de gaz riche en oxygène) à la surface du wafer placé dans un four. En réglant la température et la quantité de

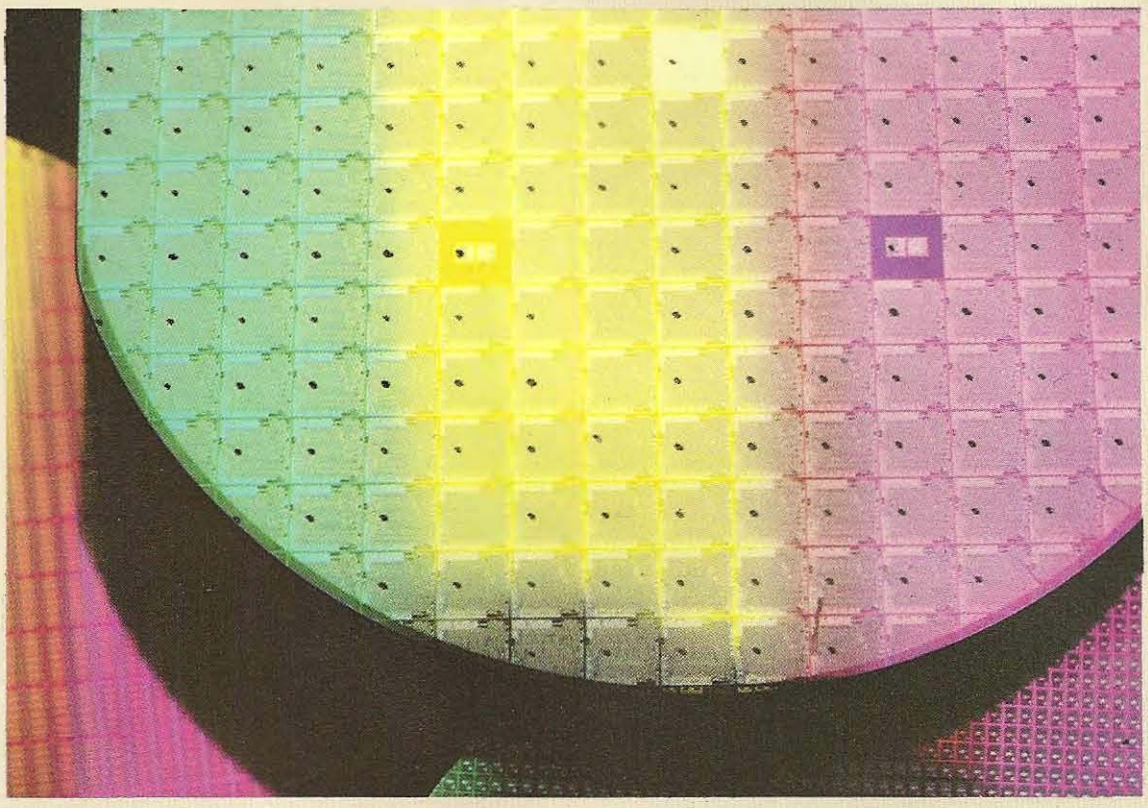


Advertising Arts

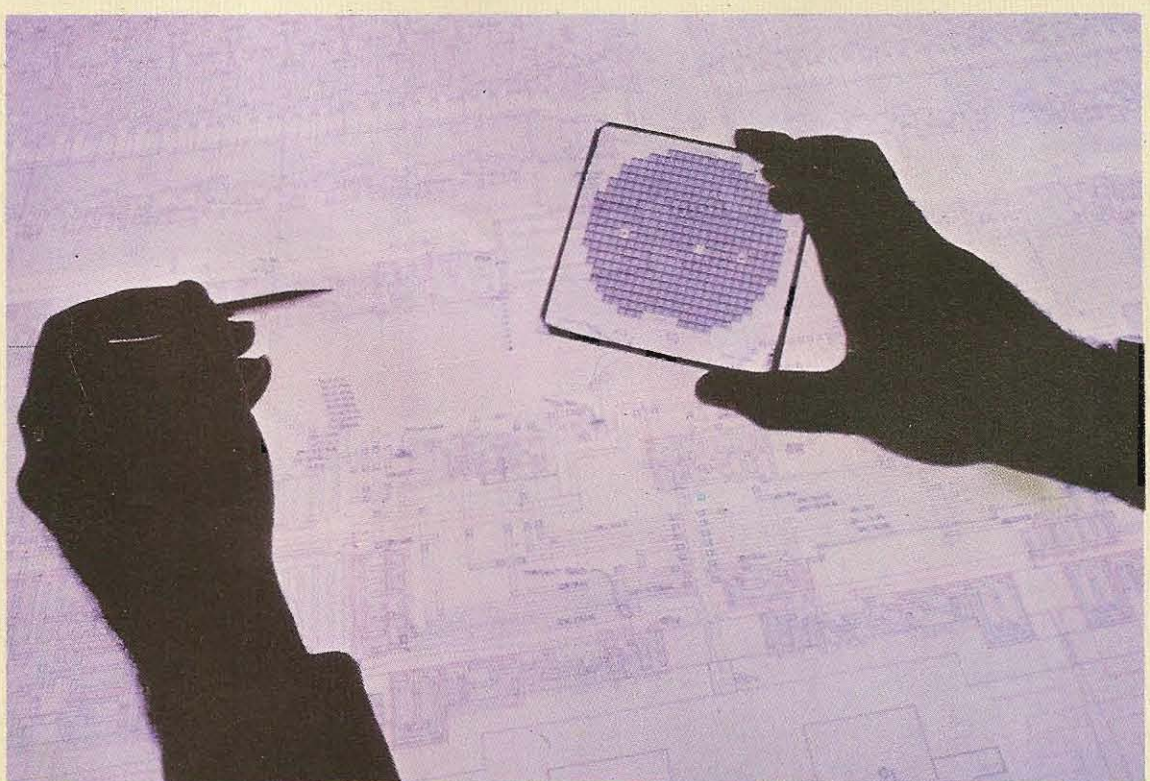
gaz, on détermine exactement l'épaisseur de la couche oxydée. On la recouvre ensuite d'un produit chimique photo-durcisseur, solidifié par exposition aux rayons ultraviolets.
Cette lumière est envoyée sur le wafer au travers d'un masque approprié où se répète des centaines de fois un motif de base.
On procède par contact ou par projection : dans le premier cas, le masque est directement posé sur le wafer, dans le second cas, un système optique projette le dessin. Quel que soit le système employé, le matériau durcit là où la lumière parvient.
Ailleurs, il sera dissout et retiré au moyen d'un révélateur chimique.
L'opération suivante est celle de la gravure par l'acide.

La gravure de la plaquette est réalisée généralement par attaque à l'acide fluorhydrique. Dans les zones exposées, l'oxyde est attaqué à l'acide fluorhydrique, tandis qu'il demeure intact là où il est protégé par une mince couche photo-résistante. Puis les résidus de la substance photo-sensible sont éliminés à l'aide d'un révélateur.
Des zones d'oxyde masquent ainsi les parties de la plaquette qu'on veut préserver du dopage.

Microcircuits imprimés sur une plaquette de silicium avant leur découpage. En bas : le circuit dessiné sur le plan est reproduit sur chaque puce de la plaquette.



B. Coleman/Marka



B. Coleman/Marka

Le wafer sera chauffé dans un four, en atmosphère borée ou phosphorée selon que l'on veut obtenir des zones actives p ou n. Il y restera le temps nécessaire à la diffusion de la substance dopante (impuretés) dans le silicium.
Une couche du microcircuit vient donc d'être réalisée. Cette succession d'opérations se répètera plusieurs fois, avec des variantes, pour créer les autres couches du circuit intégré (11 en moyenne, qui nécessitent à peu près 60 opérations).
On dépose ensuite une couche métallique (d'aluminium en général), gravée sur masquage, qui établit un réseau de connexions entre les différentes zones actives. Cette couche possède également des contacts reliant le circuit à l'extérieur.
Une dernière couche d'oxyde est enfin appliquée pour protéger le circuit.

* * *

La moindre pollution, à n'importe quel stade de la fabrication des puces, peut compromettre leur fonctionnement: la production s'effectue donc en atmosphère soigneusement filtrée et dans des limites de température parfaitement contrôlées. Les techniciens doivent même endosser des vêtements de protection stériles.
Les puces sont alors testées l'une après l'autre à l'aide de pointes reliées à un ordinateur, puis elles sont ensuite séparées les unes des autres.
Les puces défectueuses sont marquées à l'encre, puis éliminées lors du découpage. Ce dernier, qui suivait autrefois les lignes de coupe du cristal, utilise aujourd'hui les techniques modernes du diamant et du laser. Les chips non défectueux sont collés sur un socle et leurs pattes de contact sont soudées par de fins fils d'or à celles du socle. Ils sont ensuite encapsulés dans un boîtier de plastique ou de céramique avant d'être soumis à un nouveau contrôle.
Dans le cas d'une nouvelle fabrication, le pourcentage de puces sans défaut peut ne pas dépasser 5 %; le rendement d'une bonne production n'est d'ailleurs pas toujours supérieur à 25 %.
D'importants progrès sont accomplis tous les jours dans le domaine de la fabrication des microprocesseurs: on conçoit de nouveaux circuits en redessinant les zones défectueuses sur les précédents, on modifie les procédés de fabrication et on réduit la taille des puces.
Ce dernier point permet d'augmenter la production de chips au cours d'un même cycle de fabrication et d'en améliorer le rendement. De plus, on accroît ainsi la puissance, la compacité et la vitesse de fonctionnement des circuits en réduisant la longueur des connexions, ce qui est fondamental pour des ordinateurs qui effectuent des milliers de calculs à la seconde.

* * *

Un bref regard vers l'évolution passée s'impose. A l'apparition des circuits intégrés, dans les années soixante, Gordon Moore, le fondateur d'Intel, prédit que la complexité (nombre de composants intégrables dans un circuit) doublerait chaque année.
Bien peu de gens le crurent, et pourtant, la « loi de Moore » s'est, jusqu'ici, assez bien vérifiée. En effet, la densité d'intégration, qui était de 32 composants en 1965, atteignait 1 000 en 1970, 1 million en 1980, et 40 millions en 1984. Il est probable que les circuits intégrés vont continuer à obéir à cette loi. Cela sera d'autant plus facile que le schéma des éléments de mémoire sera plus régulier.
Les techniques de production devront être adaptées à l'augmentation constante du nombre d'éléments que l'on rassemble dans un même circuit.
Par exemple, lors des opérations de masquage, les ultraviolets, de trop grande longueur d'onde pour les circuits les plus compliqués (à haute intégration), devront bientôt être remplacés par les rayons X, dont la longueur d'onde beaucoup plus courte permettra de résoudre ce problème.
Dans un avenir plus lointain, les chercheurs envisagent de remplacer le support de silicium, aujourd'hui principal support des circuits intégrés.
*Pour cela, on utiliserait l'**arséniure de gallium**, plus délicat à manipuler, mais qui rend possible la réalisation de circuits plus rapides en consommant moins d'énergie.*
Cette baisse de la consommation, se traduisant par une diminution de la chaleur dissipée, permettra de réduire encore la taille des composants et d'en augmenter la complexité en faisant tenir plus de composants sur chaque puce.

Commandes de la bande magnétique. La bande magnétique peut remplacer le disque comme support de mémoire de masse. Elle possède ses propres commandes de sauvegarde (SAVE) et de chargement de programmes en binaire (LOAD).
Voici leur syntaxe :

CSAVE "NOM"
CLOAD "NOM"

NOM désigne le programme qu'on veut sauver ou charger. Toutefois, pour une bande magnétique, seul le premier caractère du nom (ici, N) est pris en compte : CSAVE "NOM" et CSAVE "N" sont donc équivalents. Dans les deux cas, le programme est stocké sous le nom N. Il faudra donc être très attentif à ne pas utiliser des noms commençant par la même initiale pour deux programmes différents.
Les instructions CSAVE et CLOAD, servent également, sous une forme légèrement différente, pour enregistrer des données en chaînes de caractères.

Affichage et listage des programmes

Le Basic standard possède deux périphériques de sortie : l'écran et l'imprimante. On peut demander l'affichage à l'écran ou l'impression sur papier, aussi bien de données que de programmes.
Pour les programmes, on utilise les commandes suivantes :

LIST ligne de début, ligne de fin
(pour l'écran)
LLIST ligne de début, ligne de fin,
(pour l'imprimante)

« Ligne de début » et « ligne de fin » correspondent aux numéros des instructions qui délimitent la partie du programme à lister : LIST 20, 100 provoque l'affichage à l'écran du programme résident de la ligne 20 à la ligne 100. La commande équivalente pour l'imprimante est LLIST 20, 100.
Il existe deux commandes NULL et WIDTH peu usitées, que nous ne mentionnerons que pour mémoire.
NULL sert à indiquer, pour les machines équipées d'un perforateur de bande, le nombre de blancs que l'on veut laisser entre deux lignes. Si l'on précise, par exemple, NULL 4, la machine laissera quatre blancs après chaque ligne.
Quant à la commande WIDTH, elle détermine la longueur maximale des lignes à écrire. La commande WIDTH 20 pour l'écran (ou WIDTH LPRINT 20 pour l'imprimante), limite à 20 le nombre de caractère par ligne, en créant une « fenêtre » de 20 emplacements excluant le caractère supplémentaire éventuel.

Exécution, interruption et mise au point des programmes

On lance l'exécution du programme résidant en mémoire par la commande RUN.
Tout programme contient une instruction de fin (END) et l'exécution s'arrête donc, en principe, à cette instruction. Cependant, il arrive, lors des premiers essais surtout, que des erreurs de programmation conduisent à éviter systématiquement l'instruction END.
La machine ne trouvant pas cette instruction de fin, le programme continue alors à tourner indéfiniment si on ne l'interrompt pas en appuyant simultanément sur les touches CNT (ou CTRL) et C (control + C).
La commande CONT (continue), qui relance l'exécution, sert également à tester pas à pas le déroulement d'un programme.
Une instruction STOP, insérée aux endroits « stratégiques » du programme, en interrompra l'exécution.
L'opérateur pourra alors demander interactivement l'affichage des résultats intermédiaires, et les analyser.
La commande CONT fera reprendre l'exécution là où elle aura été interrompue. De cette manière, on peut examiner les étapes de chaque opération et s'assurer de leur validité.
On ne peut pas employer CONT si on a profité de l'arrêt du programme pour y apporter des corrections.
Les deux dernières commandes de mise au point, utilisables interactivement (commandes) ou dans le programme (instructions), sont TRON et TROFF, qui régissent le mode « trace ».
TRON (trace on) affiche au fur et à mesure les numéros des instructions dans l'ordre de leur exécution.
On peut ainsi connaître avec précision le chemin suivi par la machine, sans avoir à consulter l'organigramme ou le listing du programme.

Un programme pouvant contenir un grand nombre de décisions (tests) en fonction desquelles un parcours est choisi plutôt qu'un autre, la logique devient parfois si compliquée qu'il est difficile de discerner d'où viennent les erreurs. Il est alors plus simple d'utiliser la commande TRON que d'essayer de « dérouler » le programme à la main.
La commande TROFF (trace off) annule la commande TRON (trace on).

Concaténation

Lorsqu'on fractionne les programmes en sous-programmes, il est souvent possible de réutiliser, dans une nouvelle application, des suites d'instructions écrites pour une autre.
Ces « routines » d'usage général seront enregistrées sur des fichiers séparés, comme s'il s'agissait de programmes indépendants. Leur réemploi est ainsi facilité. Pour s'en servir, il suffit de les prélever sur le disque et de les fusionner au programme résident.
On utilise pour cela la commande MERGE suivie du nom du fichier. Cette instruction ne concerne que les programmes (fichiers en Basic) stockés en format ASCII ; utilisée sur un fichier binaire ou sur des données, la machine émet un message d'erreur et la commande n'est pas exécutée.
La commande MERGE « ESSAI » charge la routine ESSAI enregistrée sur la disquette et l'unit au programme résident, réalisant ainsi leur concaténation. Notons que le programme à fusionner doit avoir été sauvegardé en code ASCII.
Si ces deux programmes possèdent des numéros de ligne en commun, ce sont les instructions du sous-programme qui se trouvait sur le disque qui, lors de la fusion, remplacent celles du programme résident.
Prenons un exemple :

Programme en mémoire
```
10 A = B + C
20 D = A * 2 + SQR (B)
30 E = A + D
40 TOTAL = E * 2
50 T1 = E — TOTAL
```

Programme sauvegardé sur disque sous le nom PROG-2
```
30 F = A + B + C
50 SOMME = TOTAL + F
60 END
```

L'instruction MERGE « PROG-2 » charge le programme PROG-2 et le fusionne avec le programme résident. Celui-ci possède désormais l'instruction 60 et ses lignes 30 et 50 sont remplacées par celles de PROG-2.
Un programme « structuré » se compose d'une partie principale et de routines (sous-programmes).
Le programme principal esquisse la structure générale du programme, tandis que les sous-programmes traitent en détail l'exécution des différentes opérations.
La commande MERGE permet de réaliser cette structuration.
Lors de l'écriture, on prépare et on mémorise séparément chaque module (programme principal et sous-programmes) au lieu de tout réunir, dès le départ, dans un même fichier.
Il suffit ensuite de charger en mémoire le programme principal et de lui adjoindre, par fusions successives, tous les sous-programmes nécessaires.
On se réserve ainsi la possibilité de réutiliser les routines dans d'autres applications, en les fusionnant à d'autres programmes principaux.
Enfin, la commande CLEAR effectue les opérations suivantes :

- clôture des fichiers (un programme ne peut accéder à un fichier tant qu'il n'a pas été réouvert) ;
- remise à zéro des variables numériques et de chaînes ;
- réservation de l'espace mémoire disponible pour l'utilisateur (en particulier réinitialisation de la zone réservée aux variables).

Généralement, cette dernière commande s'effectue en début d'exécution d'un programme. Elle est prise en charge par le système, dans la mesure où réservations d'espace et initialisations des variables se produisent automatiquement à l'exécution d'un programme.

Les instructions

Les instructions représentent la codification, en langage de programmation, des opérations que l'ordinateur doit effectuer.
Dans ce chapitre seront exposées toutes les instructions qui traitent des variables et des constantes.

Test 10

1 / On a mémorisé un programme sur disque de trois façons différentes, en utilisant les commandes suivantes :

a) SAVE "A:ESSAI"
b) SAVE "A:TEST", A
c) SAVE "B:ESSAI2", P

Quelle est la version en ASCII ?
Sur quel disque le programme nommé ESSAI2 est-il mémorisé ?
Lesquelles de ces trois versions peut-on corriger ?

---

2 / Le programme suivant :

```
10 A = 6
20 B = 8
30 C = 10
40 D = C * (A + B)
50 E = 2 * D/A
```

a été renuméroté par la commande REN 20, 10, 2, puis fusionné à la routine par la commande MERGE :

```
20 B = 2
30 F = D + E
```

Donner les valeurs des variables D et E dans les cas suivants :
a) exécution du premier programme seul et avant la renumérotation ;
b) exécution du programme obtenu après renumérotation et fusion.

---

3 / Ecrire le programme obtenu par la séquence suivante :

```
NEW
10 A = 3
20 B = 2
30 C = 3 * (A + B)
REN 10,10,2
AUTO 20,5
20 F = 2 * (C + A)
25 PRINT F
```

---

4 / Comment remplacer la ligne 20 F = 2 * (C + A) par F = 4 * (C + A) ?

---

5 / A l'exécution du programme suivant que lira-t-on à l'écran ?

```
10 A = 6
20 TRON
30 B = A + 2
40 C = 2 * B
50 TROFF
60 D = B + C
```

Quelle commande doit-on utiliser pour sauvegarder ce programme sur le disque A, en format ASCII et sous le nom ESSAI ?

Les solutions de ce test se trouvent pages 374 et 375.

Tableau récapitulatif des commandes

Commande	Effet	Exemple
NEW	Remet à zéro le contenu de la mémoire.	NEW
AUTO n, m	Déclenche la numérotation automatique à partir de la ligne n, et avec un pas m.	AUTO 10, 5
CLEAR	Réinitialise les variables numériques et les chaînes de caractères.	CLEAR
SAVE "X:YYY", Z	Sauvegarde sur le disque X (A/B ou 0/1) le programme de nom YYY, avec l'option Z (en binaire par défaut ; en ASCII si Z = A ; protégé si Z = P).	SAVE "A:PROG-1", A SAVE "A:PROG-2" SAVE "0:NOM", P
RENUM n1, n2, m	Renumérote le programme de m en m, à partir de la ligne n2 qui prend le numéro n1.	RENUM 10,5,5 RENUM 300,2,12
EDIT n	Edite la ligne n.	EDIT 10
LIST n, m	Affiche à l'écran les lignes n à m.	LIST 5, 50
LLIST n, m	Effectue la même opération sur l'imprimante.	LLIST 3, 25
RUN	Lance l'exécution du programme résident.	RUN
TRON	Fait passer en mode « trace » l'affichage des numéros des instructions exécutées.	TRON
TROFF	Annule la commande TRON.	TROFF
MERGE "YYY"	Charge la routine YYY stockée sur disque, et la concatène au programme résident.	MERGE "PROG-1"

Les instructions qui concernent les mémoires de masse (bandes ou disques) feront l'objet d'une étude particulière.

En exposant la syntaxe des instructions, nous avons souvent inséré des blancs entre les mots clés, les noms de variables et les opérateurs par souci de clarté, afin de mieux distinguer les différentes parties d'une même instruction. Toutefois, il faut savoir dès maintenant que ces espaces n'ont aucune signification particulière pour l'interpréteur (ou le compilateur) Basic, qui reconnaît les éléments d'une instruction où qu'ils se trouvent dans une ligne, pourvu que l'orthographe soit correcte et l'ordre respecté.

Début et fin des programmes

De même que les organigrammes commencent par un bloc de début et se terminent par un bloc de fin, un programme comporte nécessairement une instruction START et une instruction END.

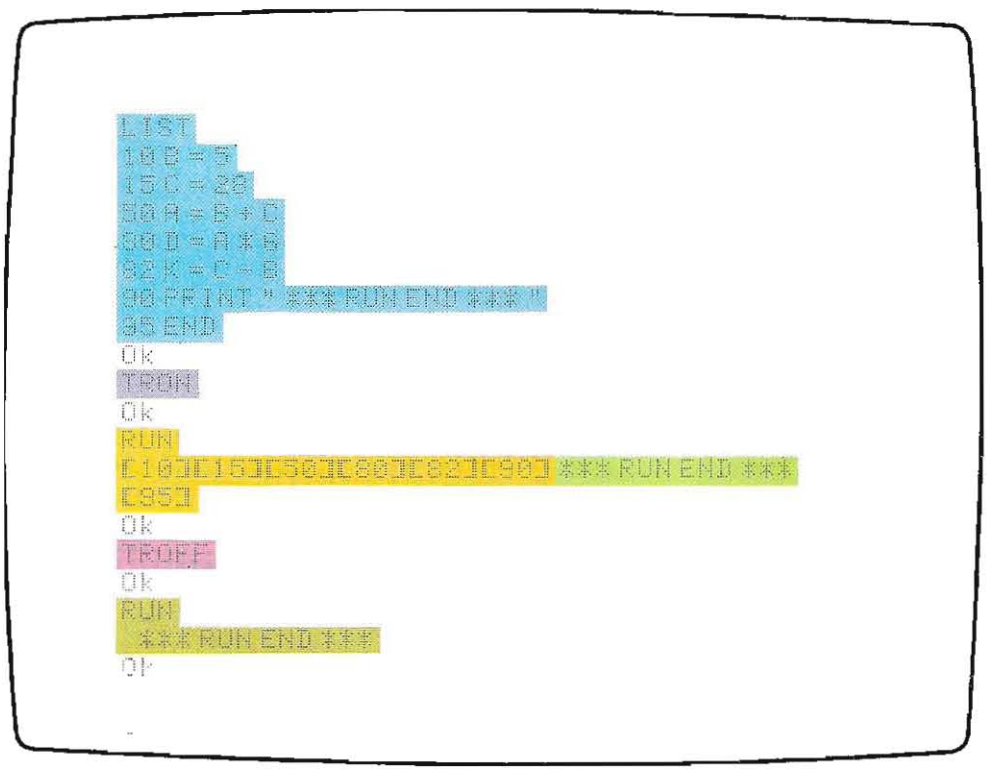
Cependant, dans certains langages, et notamment le Basic, il n'existe pas d'instruction de début : l'exécution du programme démarre automatiquement à la ligne portant le plus petit numéro.

Cependant, aucun automatisme similaire ne peut déterminer la fin du programme car les instructions qui portent les plus grands numéros ne sont pas toujours celles qui doivent être exécutées les dernières.

Comme nous l'avons vu, un programme se décompose en plusieurs modules : le programme principal (portant le nom de Main, en anglais) et les sous-programmes (appelés aussi routines, de l'anglais Subroutines). Le programme principal appelle les sous-programmes au fur et à mesure de ses besoins. A la fin de chaque sous-programme, l'exécution

EMPLOI DES COMMANDES RUN, TRON ET TROFF





La première commande (LIST) demande le listage du programme qui se trouve en mémoire. Ce programme s'affiche à l'écran, suivi du message Ok qui indique que l'exécution de la commande est terminée.

Avec TRON, l'opérateur passe en mode trace, demandant la visualisation des numéros des instructions exécutées.

La commande RUN lance l'exécution du programme. Les numéros de ligne apparaissent successivement à l'écran (entre crochets).

Cet affichage de caractères résulte de l'exécution de l'instruction de la ligne 90 (PRINT).

L'exécution terminée, l'opérateur annule le mode trace par la commande TROFF.

Lors de la nouvelle exécution, la seule sortie sera celle de la ligne 90.

du programme principal reprend juste après l'instruction d'appel. Certaines lignes sont donc « sautées » pour être reprises ensuite. L'exécution d'un programme n'est donc pas obligatoirement séquentielle.
Le schéma de la page 369 détaille un programme structuré en deux sous-programmes (A et B). Arbitrairement, la routine A est la première appelée, mais elle peut, en effet, s'insérer à n'importe quel endroit du programme (par exemple, de 3700 à 4580). De même, B, bien qu'appelée plus tard, pourrait très bien la précéder (elle occupe ici les lignes 1350 à 1920).
Le choix de l'ordre d'appel des routines est déterminé par les lignes 50 et 95 du programme principal. A la fin du premier sous-programme (A), se trouve l'instruction RETURN (ligne 4580) qui rend la main au programme principal à partir de la ligne 60.
L'ordinateur effectue les instructions 60, 65 et 95. Cette dernière appelle la routine B. L'exécution se poursuit donc de la ligne 1350 à la fin de la routine B, c'est-à-dire à la ligne 1920 où se trouve l'instruction de retour au programme principal (ligne 100).
A partir de cette ligne, l'exécution se poursuit dans l'ordre des numéros (instructions 100, 110, etc.). Si la machine ne rencontre pas auparavant l'instruction de fin (END), elle exécute à nouveau le sous-programme B (bien qu'il n'ait pas été rappelé). Elle rencontre alors, à la ligne 1920, l'instruction RETURN, exécute à nouveau la partie du programme comprise entre les instructions 100 et 1920 et ainsi de suite. Puis elle signale une erreur.
Sur le schéma de la page 369, on a placé l'instruction END au terme logique de l'exécution, c'est-à-dire après les deux sous-programmes. Elle devrait, en fait, se trouver « physiquement », (selon son numéro de ligne) dans le programme principal, avant le sous-programme A. L'organigramme c) détaille le schéma a) et résume le mécanisme décomposé en b), en représentant l'ordre effectif des instructions.
L'exécution du programme démarre automatiquement à l'instruction de plus petit numéro. C'est donc ici que le programmeur devra éventuellement spécifier les options qu'il choisit, comme par exemple, la base de numérotation des tableaux, avec l'instruction OPTION BASE 1. L'ordinateur numérote habituellement les tableaux à partir de 0, ce qui peut s'avérer gênant pour certaines applications.
Ainsi, pour un programme en Basic, les éléments d'un tableau de cinq valeurs sont numérotées de 0 à 4. Si le programmeur le désire, l'instruction OPTION BASE 1 insérée en tête de programme fera débuter la numérotation à 1. Dès lecture de cette instruction, la machine cessera d'utiliser l'indice 0.
Il ne s'agit, en fait, que d'une question de présentation : la machine fonctionne toujours de la même façon et décrémentera de 1 chaque indice, pour retrouver son propre système de numérotation.
Alors que l'instruction OPTION BASE 1 peut commencer un programme, l'instruction END le termine obligatoirement, constituant ainsi la dernière ligne au sens logique (et non selon l'ordre de numérotation).
Il est toujours utile, lors de la rédaction d'un programme, d'écrire des commentaires expliquant les opérations effectuées et décrivant les méthodes employées.
Un programme « nu » s'avère, en effet, fort difficile à ré-examiner et à comprendre après quelque temps. On y insère donc un certain nombre de lignes qui ne constituent pas des instructions, mais de simples commentaires. Cette différence est spécifiée à l'ordinateur par REM (abréviation de « remark »), placé en tête de ligne, ou par un symbole particulier (apostrophe, point d'exclamation, étoile, « c » minuscule, etc., selon les systèmes et les langages).
Voici un exemple :

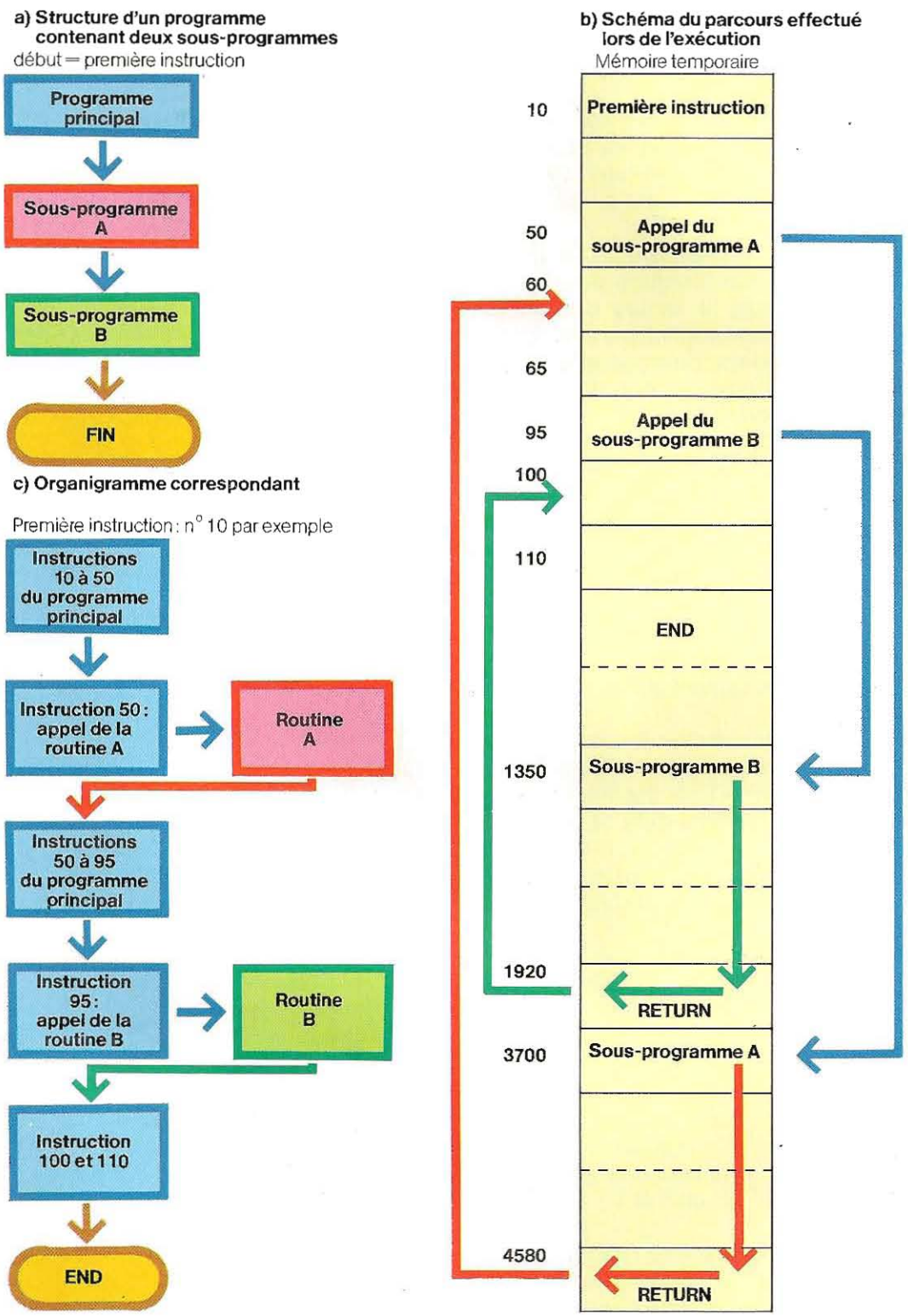
```
10 OPTION BASE 1
20 REM Cette ligne est un commentaire
30 A = B + C 'Somme
(30 A = B + C !Somme)
40 END
```

A la ligne 20, le commentaire est indiqué par REM et par les symboles ' ou ! à la ligne 30. Tout ce qui se trouve à droite du symbole est considéré comme commentaire et n'est donc pas traité. Cependant, ces commentaires occupent de la place en mémoire.

Déclaration du type de variables

Le programmeur dispose de quatre types de variables ou de constantes : entiers, réels en

STRUCTURATION D'UN PROGRAMME ET DEROULEMENT DE SON EXECUTION



« simple précision » et en « double précision », chaînes de caractères. Pour déclarer un type, le nom (ou la valeur) sera suivi du symbole correspondant (% = entier, ! = simple précision, # = double précision, $ = chaîne).
Les noms de variables seront toujours accompagnés de l'un de ces symboles (sauf le symbole ! qui peut être omis, car, en l'absence d'indication, la machine considère « par défaut » qu'il s'agit d'une variable en simple précision).
Il serait très fastidieux de répéter continuellement ces symboles lorsqu'on écrit un programme. De plus, la lecture d'un listage encombré de symboles finit par être difficile.
Pour éviter cela, on peut remplacer la « déclaration explicite » (spécification du symbole) par une « déclaration implicite ».
Une instruction DEF (à ne pas confondre avec DEF FN vu précédemment), précisera une fois pour toutes le type d'une variable ou d'un groupe de variables.
En voici la formulation :

DEFINT pour les entiers
DEFSNG pour les réels en simple précision
DEFDBL pour les réels en double précision
DEFSTR pour les chaînes de caractères

On indiquera, dans l'instruction, les lettres par lesquelles commenceront lesdites variables. Par exemple, DEFINT I-N signifie que toute variable du programme dont le nom commence par une lettre entre I et N est entière.
La définition peut contenir plusieurs groupes de lettres séparés par des virgules. Ainsi :

DEFINT I-N, X-Z — Les noms ayant pour initiale une lettre comprise entre I et N ou X et Z désignent des variables entières.
DEFDBL A-D, P-T — Les noms des variables en double précision commencent par une lettre comprise entre A et D ou P et T.

Cette déclaration implicite n'existe pas sur tous les systèmes. En outre, il arrive parfois qu'on ne puisse définir un intervalle de lettres mais qu'on doive déclarer le nom des variables en entier et donc définir séparément chaque variable.
La déclaration implicite est surtout utile pour les entiers et les réels en double précision.
La déclaration facultative des réels en simple précision est cependant conseillée pour plus de clarté.
L'instruction DEFINT, citée en exemple, est très utile pour une programmation rigoureuse.
Dans une boucle (voir page 174), on utilise, en effet, un indice qui part d'une valeur entière et subit des incrémentations successives, entières elles aussi, jusqu'à la valeur finale de fin d'exécution de la boucle.
Le déclarer comme réel ne constituerait pas vraiment une erreur, mais ralentirait l'exécution de la boucle.
En outre, on utiliserait plus d'espace mémoire que nécessaire (seule la partie entière de l'indice sera utile).
Le Fortran (langage de haut niveau d'où dérive le Basic), prévoit d'ailleurs à cet effet que tous les noms qui commencent par une lettre comprise entre I et N désignent systématiquement des variables entières, sans qu'il soit nécessaire de les déclarer.
C'est donc avec profit qu'on adoptera la déclaration DEFINT I-N et qu'on réservera ces lettres initiales aux indices de boucles.

Instructions d'affectation

Ce sont les instructions qui servent à attribuer à des variables ou à des constantes une valeur déterminée, numérique ou alphanumérique et résultant éventuellement d'un calcul.
Toute grandeur se mentionne dans un programme par son nom symbolique, auquel le système d'exploitation associe une place déterminée en mémoire.
Utiliser ce nom symbolique revient à prélever le contenu de l'emplacement de mémoire ou à y écrire une donnée.
Schématiquement, tout se passe un peu comme si le nom d'une variable représentait une adresse de mémoire où l'on stocke, dans une mise à jour continuelle, les valeurs que cette variable prend successivement.

LET

L'instruction d'affectation la plus simple est :

LET nom = valeur

On demande ainsi à la machine d'écrire la donnée « valeur » dans l'emplacement de mémoire « nom ».
L'instruction LET A = 2.5 place la valeur 2.5 dans un emplacement mémoire qui prend alors le nom A.
Le symbole A sera par la suite mentionné à la place de sa valeur numérique.
Dans la plupart des langages, on peut omettre LET et écrire simplement A = 2.5.
Le signe = (prononcé « reçoit » et non « égale ») suffit alors à symboliser une instruction d'affectation.
On mémorisera ainsi le résultat de n'importe quelle opération dans une variable, qu'on pourra ensuite afficher, imprimer, ou utiliser dans de nombreux calculs.
L'instruction d'affichage (qui sera étudiée plus loin) sera mentionnée ici, et pour l'instant, sous sa forme la plus simple : PRINT nom.
Cette instruction provoque l'affichage à l'écran de la valeur associée au symbole « nom ».
Dans le cas de l'exemple précédent (LET A = 2.5), l'instruction PRINT A affichera à l'écran le nombre 2.5.
Calculons, par exemple, la surface d'un cercle de 3 cm de rayon (R).
Nous appliquerons la formule Surface = $3.14 \times R^2$, puis remplacerons R par 3, ce qui donne : Surface = 3.14×3^2.
Cette instruction constitue, à elle seule, presque tout le programme :

```
10' OPTION BASE 1 est inutile puis-
15' qu'aucun indice n'est utilisé
20 SURFACE = 3.14 × 3 ∧ 2 ' le symbole
25' ∧ indique l'élévation à une puissance
30 PRINT SURFACE
40 END
RUN (lancement de l'exécution)
28.26 (résultat du calcul)
```

La ligne 10 est un commentaire (symbole ').
La ligne 20 contient le calcul de l'affectation du résultat à la variable SURFACE.
A la ligne 30 se trouve l'instruction d'affichage de SURFACE, et la ligne 40 termine le programme.
Plusieurs instructions peuvent tenir sur une même ligne, à condition de les séparer par le symbole : (ou / sur certains systèmes). Ainsi, en omettant les commentaires, on obtient le programme suivant :

```
20 SURFACE = 3.14 × 3 ∧ 2 : PRINT
SURFACE : END
RUN
28.26
```

En fait, ce programme n'a aucune utilité pratique car il faut le récrire pour chaque nouvelle valeur du rayon.
Or, un programme d'application, pour être adapté à d'autres données, doit toujours avoir été paramétré.
Dans le programme ainsi récrit :

```
10 R = 3
20 SURFACE = 3.14 x R ∧ 2
30 PRINT SURFACE
40 END
```

le rayon n'est plus désigné, en ligne 20, par sa valeur numérique mais par un symbole : il est devenu une variable. En changeant seulement la valeur numérique affectée à R en ligne 10, on pourra calculer la surface de n'importe quel cercle.
Cependant, le programme n'est pas encore réellement parfait, puisqu'il faut modifier une instruction (ligne 10) pour chaque nouveau calcul.
Le seul moyen de le généraliser à toutes les valeurs de R est de donner à l'utilisateur la possibilité de fixer lui-même, à chaque fois, la nouvelle valeur de R.
Pour cela, on utilise l'instruction INPUT R, qui stoppe la machine jusqu'à ce que la valeur de R soit entrée au clavier.
Une fois saisie, cette valeur est affectée à la variable et l'exécution se poursuit comme précédemment.

```
10 INPUT R
20 SURFACE = 3.14 × R ∧ 2
30 PRINT SURFACE
40 END
```

Lors du déroulement du programme, la machine s'arrête à l'instruction 10 et affiche le signe ? sur l'écran, pour informer l'utilisateur qu'elle attend l'entrée d'une donnée.
L'opérateur tape la valeur numérique de R au clavier, puis la valide en actionnant la touche RETURN.

La variable R prend alors la valeur saisie, valable dans toutes les instructions où elle apparaît, et ce, jusqu'à l'affectation suivante. Le symbole ^, utilisé à la ligne 20, désigne l'élévation à une puissance. Toutes les machines ne comportent pas cet opérateur, et on doit alors expliciter la multiplication (ici $R^2 = R \times R$) ; la ligne 20 devient :
SURFACE = 3.14 × R × R.
La page 373 reproduit le listing d'une partie d'un sous-programme qui calcule, à partir de paramètres en entrée, la surface et le périmètre de figures planes.
La valeur de l'indicateur K indique la figure choisie, et c'est en fonction de cette valeur (1, 2, etc.) qu'est sélectionnée la formule appropriée (lignes 1670, 1700, etc.). Nous verrons plus loin comment s'effectue cette sélection.
Le symbole : a permis d'aligner plusieurs instructions correspondantes à la suite l'une de l'autre sur une seule ligne.
Toutes les lignes contenant le symbole ' après leur numéro sont des commentaires.
En entrée, on fournit les données suivantes : type de figure géométrique (K) et valeurs nécessaires au calcul ; on obtient en sortie la surface de la figure (variable SURF) et son périmètre (variable PER).
Si l'on pose, par exemple, K = 5 (cercle) et COTE1 = 7 (dans un cercle, Côté1 désigne le rayon et les autres variables sont donc inutiles), on obtient SURF = 153.86 et PER = 43.96 (PER désigne la circonférence). Si l'on veut ensuite calculer la surface d'un losange (K = 4), il faudra fournir trois données à la machine, le côté et les deux diagonales.
Remarquons que l'instruction END n'apparaît pas dans le listing, car elle est remplacée par RETURN, puisqu'il s'agit d'un sous-programme. L'instruction de fin se trouve donc dans le programme principal correspondant.
LET, qui est facultatif, ne figure presque jamais dans les instructions d'affectation. On écrit donc, (par ex.) A = 21 au lieu de LET A = 21. Cette forme implicite est le seul exemple d'instruction qui ne commence pas par un « mot réservé » du langage, c'est-à-dire un mot désignant une opération donnée.
Dans tous les autres cas, le mot réservé doit apparaître derrière le numéro de ligne ou le symbole : .
Comme leur nom l'indique, les mots réservés (codes d'instructions ou mots clés) ne peuvent pas être utilisés pour désigner des variables dans les programmes. Si nous avions, par exemple, utilisé PRINT au lieu de PER pour le périmètre, nous aurions provoqué une erreur, car l'interpréteur aurait reconnu un mot réservé et aurait tenté d'exécuter une instruction d'affichage erronée en lui affectant une valeur comme une variable.
Certains interpréteurs Basic sont capables de discerner si un mot désigne une instruction ou une variable, selon son contexte. Ils restent cependant l'exception.

DATA, READ et RESTORE

Les instructions DATA et READ permettent également d'affecter des valeurs à une ou plusieurs variables. S'il faut toujours les utiliser conjointement, elles peuvent cependant ne pas se trouver au même endroit du programme.
La machine sait que le code DATA est suivi d'une liste de nombres ou de lettres, constituant des valeurs à affecter à des variables. Ces dernières sont mentionnées après l'instruction READ. Les instructions

```
10 READ X
20 DATA 3.5
```

affectent à X la valeur 3.5, de même que l'instruction X = 3.5.
Il existe cependant des différences entre DATA et l'affectation ordinaire :
- on peut attribuer des valeurs à plusieurs variables par une seule instruction DATA ;
- les valeurs du DATA peuvent être utilisées plusieurs fois pour des variables différentes.

L'instruction DATA permet donc des affectations multiples. Si l'on veut, par exemple, affecter respectivement les valeurs 2, 7.3, 15 et 174 aux variables R, VAL, NOM et ETAT avec la première instruction, on écrira :

```
10 R = 2 : VAL = 7.3 : NOM = 15 : ETAT = 174
```

et les valeurs numériques 2, 7.3, etc. ne pourront plus être réutilisées.
Si une autre variable du programme, T par exemple, doit prendre la valeur 2, nous devrons ajouter une nouvelle instruction d'affectation (T = 2) sans pouvoir nous servir de l'instruction 10.

SOUS-PROGRAMME DE CALCUL DE SURFACES ET DE PERIMETRES

```
1500 '   ** Sous-programme de calcul de surfaces et de périmètres
1510 '
1520 '      SYMBOLES UTILISES :
1530 '                          SURF  = Valeur de la surface calculée
1540 '                          PER   = Valeur du périmètre
1550 '                          COTE1 = Longueur du premier côté (en entrée)
1560 '                          COTE2 = Longueur du second côté (en entrée)
1570 '                          COTE3 = Si nécessaire
1580 '                          COTE4 = Si nécessaire
1590 '   K = FLAG indiquant le type de figure
1600 '                              K = 1 CARRE
1610 '                              K = 2 RECTANGLE
1620 '                              K = 3 TRAPEZE
1630 '                              K = 4 LOSANGE
1640 '                              K = 5 CERCLE
1650 '
1660 '   Carré (K = 1)
1670 SURF = COTE1 * COTE1 : PER = 4 * COTE1 : RETURN
1680 ' Rectangle (K = 2)
1690 ' COTE1 = Longueur   COTE2 = Largeur
1700 SURF = COTE1 * COTE2 : PER = 2 * (COTE1 + COTE2) : RETURN
1710 ' Trapeze (K = 3)
1720 ' COTE1 = Pte base  COTE2 = Gde base  COTE3 et COTE4 = côtés obliques  HAUT = hauteur
1730 SURF = (COTE1 + COTE2) * HAUT/2 : PER = COTE1 + COTE2 + COTE3 + COTE4 : RETURN
1740 ' Losange (K = 4)
1750 ' COTE1 = Petite diagonale  COTE2 = Gde diagonale  COTE3 = Côté
1760 SURF = COTE1 * COTE2/2 : PER = 4 * COTE3 : RETURN
1770 ' Cercle (K = 5)
1780 ' COTE1 = Rayon
1790 SURF = 3.14 * (COTE1 * COTE1) : PER = 6.28 * COTE1 : RETURN
1800 '
```

Solutions du test 10

1 / La version en ASCII est celle qui porte le nom TEST ; elle est mémorisée sur le disque monté dans l'unité A.
Le programme ESSAI2, qui est protégé (P = protégé) se trouve sur la disquette qui était montée dans l'unité B quand on a entré la commande.
On ne peut apporter de corrections qu'aux versions sauvegardées sous les noms ESSAI et TEST (versions a et b).

---

2 / A l'exécution du programme principal seul, la valeur des variables D et E s'obtient (lignes 40 et 50) en remplaçant A, B et C par les nombres qui leur ont été affectés aux lignes 10, 20 et 30 :

$$D = 10 * (6 + 8) = 140$$

$$E = 2 * \frac{140}{6} = 46.666... = 46.\overline{6}$$

Le tiret placé au-dessus du 6 signifie que ce chiffre se répète indéfiniment. Ce résultat mathématiquement exact est arrondi par le système qui ne conserve qu'un nombre restreint de décimales, en fonction de la précision voulue.
En renumérotant le programme de 2 en 2 à partir de 20, nous obtenons :

```
20 A = 6
22 B = 8
24 C = 10
26 D = C * (A + B)
28 E = 2 * D/A
```

Fusionnons maintenant ce programme avec :

```
20 B = 2
30 F = D + E
```

La ligne 20 du premier programme (A = 6) est remplacée par celle du second (B = 2) et l'instruction 30 est ajoutée :

```
20 B = 2
22 B = 8
24 C = 10
26 D = C * (A + B)
28 E = 2 * D/A
30 F = D + E
```

Deux valeurs sont affectées successivement à la variable B (lignes 20 et 22). La dernière efface la première et B vaudra donc 8 dans les calculs qui suivent. En Basic, une variable est initialisée à 0 tant qu'aucune valeur ne lui est affectée. A est donc nulle aux lignes 26 et 28. La ligne 26 donne

$$D = 10\ (0 + 8) = 10 * 8 = 80$$

mais il se produit une erreur (débordement ou overflow) à l'exécution de la ligne 28, car on tente d'effectuer une division par 0 :

$$E = 2 * \frac{80}{0}$$

3 / La commande REN 10, 10, 2 entraîne la renumérotation des lignes 10, 20 et 30. On obtient :

```
10 A = 3
12 B = 2
14 C = 3 * (A + B)
```

La commande AUTO 20,5 provoque la numérotation automatique des lignes, de 5 en 5 et à partir de 20. Les instructions suivantes, F = 2 * (C + A) et PRINT F, portent donc les numéros 20 et 25. Voici le programme qui résulte de ces deux commandes :

```
10 A = 3
12 B = 2
14 C = 3 * (A + B)
20 F = 2 * (C + A)
25 PRINT F
```

La variation de pas entre les numéros de ligne importe peu puisque la numérotation est totalement arbitraire (mis à part le respect d'un ordre de progression).

---

4 / Il y a deux façons de procéder :
a) retaper complètement la ligne 20,
b) passer en mode éditeur.
Dans le premier cas, il suffit de taper, à la fin du programme, la nouvelle ligne 20 F = 4 * (C + A), qui remplacera automatiquement l'ancienne. Dans le second cas, il faut entrer la commande EDIT 20 pour se mettre en édition ; l'ancienne ligne F = 2 * (C + A) s'affiche à l'écran et on peut alors remplacer localement le 2 par un 4.

---

5 / L'instruction 20 TRON, provoquant l'affichage des numéros d'instructions exécutées, sera annulée à la ligne 50 par l'instruction TROFF. Les numéros de toutes les instructions exécutées comprises entre ces deux commandes vont apparaître à l'écran. Nous verrons ainsi s'afficher :

```
[30] [40]
```

car seules ces deux lignes se trouvent entre le TRON et le TROFF. Pour sauvegarder le programme, il faudra entrer la commande :

```
SAVE "A:ESSAI", A
```

Il n'en est pas de même pour les valeurs qui suivent l'instruction DATA : elles peuvent servir à des variables différentes.
Avec READ et DATA, la précédente instruction d'affectation devient :

```
10 READ R, VAL, NOM, ETAT
.....
.....
273 DATA 2, 7.3, 15, 174
```

L'instruction DATA peut se trouver à différents endroits du programme mais, sur certaines machines, on doit la placer à la fin. Il ne s'agit pas, en effet, d'une instruction réellement exécutable, mais d'une zone de mémoire où sont enregistrées des valeurs (numériques ou non) et cela pourrait provoquer des erreurs si elle se trouvait dans le corps du programme.
L'affectation des valeurs se déroule dans l'ordre d'écriture : la première donnée (2) est donc attribuée à la variable R, la deuxième à VAL et ainsi de suite.
L'instruction READ peut être divisée en plusieurs blocs, contenant chacun un certain nombre de variables.
Au fur et à mesure, le système tient le compte

Concerto pour clavier et microprocesseur

C'est à juste titre que le stand de la Moog Company à la foire musicale de Francfort s'enorgueillissait de cette inscription : « From the men who started it all », faisant référence à l'idée, alors novatrice, de Robert Moog. Si elle nous semble aujourd'hui d'une simplicité déconcertante, l'idée de contrôler par des tensions électriques, et non manuellement, des dispositifs électroniques (oscillateurs, filtres, amplificateurs, etc.) servant à générer et à transformer des sons, était alors inédite.
Des traitements qui demandaient auparavant des heures, voire plusieurs jours de travail dans un studio d'enregistrement, devenaient dès lors presque instantanés, rendant possible la construction des premiers synthétiseurs, appareils pouvant servir en direct sur scène.

Le ***synthétiseur*** *constitue l'instrument électronique par excellence. Offrant à l'utilisateur la plus large liberté d'action sur la production et la modification des timbres, il est aussi, par là-même peut-être, l'instrument le plus difficile à utiliser. En effet, s'il arrive qu'on trouve par hasard une combinaison intéressante, il est pratiquement impossible de produire un effet précis en procédant par tâtonnements, à moins de connaître parfaitement le fonctionnement de la machine et d'improviser à la perfection.*
Pour bien se servir d'un synthétiseur, il faut posséder des notions de base en acoustique et en mécanique ondulatoire si c'est possible, afin d'avoir une réelle maîtrise des différents éléments de l'appareil (oscillateurs, filtres, générateurs d'enveloppe, amplificateurs, modulateurs).

Contrairement à l'orgue électrique, le synthétiseur n'est pas nécessairement muni d'un ***clavier*** *semblable à celui d'un piano. Néanmoins, la plupart des modèles en possèdent un, car c'est tout de même le périphérique d'entrée le plus commode pour un musicien habitué aux instruments traditionnels.*
Cependant, il en existe de nombreux autres types, tels des systèmes modulaires (le Moog et Roland en particulier) qui peuvent parfaitement fonctionner sans clavier.
Le VCS3, qui dut sa célébrité aux Pink Floyd, à Klaus Schulze et à bien d'autres, possédait accessoirement un clavier, mais constituait à lui seul un véritable petit studio de musique électronique.

Le clavier peut notamment être remplacé par un ***séquenceur,*** *ou contrôleur de bande. C'est un instrument très répandu qui mémorise des tensions de contrôle successives (s'il est analogique), ou des informations numériques. On enregistre le plus souvent des instructions correspondant à des notes et à des timbres particuliers. Le séquenceur guide ensuite le synthétiseur et lui fait jouer, à la vitesse désirée, les notes préprogrammées.*
Il est possible de programmer le séquenceur à partir du clavier en jouant une suite de notes qu'on peut ensuite faire reproduire par le séquenceur pendant qu'on exécute un autre morceau au clavier.
Il est facile de résoudre le problème de l'enregistrement avec une machine numérique : il suffit d'emmagasiner les informations dans des mémoires (RAM, bandes et autres supports magnétiques) semblables à celles d'un ordinateur.
Un tel instrument peut, par exemple, posséder seize mémoires de travail. Il est possible de mémoriser dans chacune d'elles l'ensemble des réglages des différents éléments de la machine correspondant à un timbre particulier. Il suffit d'appuyer sur un bouton pour passer très vite d'un registre à un autre. Le contenu de ces mémoires se conserve généralement même après qu'on ait éteint l'instrument, et, sur les machines les mieux conçues, il est possible de le sauvegarder sur bande, avec un magnétophone ordinaire. Le musicien, ayant programmé toutes les combinaisons qui l'intéressent, les archivera pour les rappeler au fur et à mesure de ses besoins. Les informations sont toutes enregistrées sous forme numérique, ce qui garantit la précision et la fiabilité propres à l'informatique.

*Avec leur séquenceur et leur batterie électronique, les synthétiseurs les plus sophistiqués ne sont pas loin de répondre à l'idéal de l'***homme-orchestre***: le séquenceur peut travailler en polyphonie et en synchronisa-*

tion avec la batterie et enregistrer des milliers de notes. Pendant que ces deux instruments répètent les morceaux enregistrés à une vitesse modifiable à volonté, le musicien exécutera solos, mélodies et improvisations.
Avec une bonne amplification, l'effet obtenu est souvent impressionnant.
Korg a également proposé une version améliorée du Polysix, le Poly-61, dont les douze oscillateurs à contrôle numérique permettent de produire six voix (c'est-à-dire de jouer simultanément jusqu'à six notes sur le clavier). Il est muni d'un interface pour enregistrer les programmes sur bande et possède plusieurs effets sonores automatisés. L'un d'eux permet de faire durer une note en point d'orgue. Un autre dispositif permet au Poly-61 d'enregistrer et de restituer arpèges et accords, en synchronisation avec le séquenceur ou une batterie électronique externe. La mémoire de travail peut contenir soixante-quatre programmes en même temps.

Les modèles DX Yamaha appliquent un procédé original de génération des sons, la « synthèse numérique à modulation de fréquence par algorithmes programmables ». De plus, ils autorisent le doigté dynamique : comme un piano, le clavier tient compte de la force avec laquelle les touches sont enfoncées. Contrairement au piano, pour lequel seule compte l'impulsion initiale, le clavier de ces synthétiseurs est sensible aux variations ultérieures de pression. On peut donc augmenter le volume d'une note de faible intensité au départ en appuyant plus fort sur la touche.
Parmi les nouveaux synthétiseurs, remarquons également le Chroma, conçu par Arp, l'un des plus grands fabricants d'instruments électroniques.
C'est un synthétiseur polyphonique entièrement programmable, basé sur une technologie à mi-chemin entre l'analogique et le numérique. Il est équipé de deux microprocesseurs, dont l'un (7 K de RAM et 16 K de ROM) prend en charge tous les contrôles du son, tandis que l'autre veille à la sensibilité du doigté. Il mesure, en effet, au 1/1000 de seconde près, la vitesse à laquelle une touche est frappée et simule le doigté en fonction de cette vitesse. La RAM du microprocesseur principal permet d'emmagasiner une cinquantaine de programmes, qu'on sélectionne sur un tableau de commandes. Des batteries de secours permettent de conserver les programmes si on éteint le synthétiseur. On peut également les enregistrer sur des cassettes. L'appareil est fourni avec une cassette de 150 programmes.

Le secteur le plus fascinant, où beaucoup reste encore à découvrir, est celui de la **musique assistée par ordinateur**. On trouve notamment dans le commerce de nombreux micro-ordinateurs munis d'oscillateurs produisant mélodies ou effets sonores. Les possibilités deviennent réellement intéressantes lorsque l'ordinateur se complète de cartes enfichables spéciales, ou s'il est couplé à un synthétiseur dont il contrôle tous les réglages. Extensions et programmes prometteurs ont déjà été conçus, par exemple, pour l'Apple II. Citons notamment le Music System et l'Alpha Syntauri. Inspiré de deux produits de Mountain Computer Music System, ce dernier fonctionne comme un synthétiseur numérique polyphonique à huit voix. Il permet de créer des timbres (par synthèse additive ou en dessinant sur l'écran la forme de l'onde recherchée) et possède un clavier avec contrôle du doigté.
Signalons, parmi les dernières nouveautés, un programme de composition musicale mis

Synthétiseur à modulation de tension, à mémoire numérique et à mécanisme de track tape.



© BBC



Barry Plummer

En 1972, le groupe « Yes » enregistra « Close to the edge », l'un des premiers albums de musique rock produite avec synthétiseur.

au point par Amdek pour l'Apple II, mais dont les versions compatibles avec d'autres ordinateurs individuels sont sérieusement étudiées : certaines sont déjà commercialisées. Un périphérique spécial (le CMU-800, version améliorée du Microcomposer de Roland) permet d'écrire morceaux et arrangements, et de contrôler jusqu'à huit synthétiseurs.

Si les machines très perfectionnées sont encore réservées aux grands studios et aux musiciens célèbres (elles coûtent un minimum de 100 000 F), l'évolution dans ce domaine se poursuit à un rythme rapide et le prix de certaines d'entre elles devrait diminuer d'ici quelques années.

Citons en exemple les performances de l'une de ces machines : le Wave 2.2 et son complément, le Waveterm.

Le Wave 2.2 comporte seize oscillateurs et trente « tables » de soixante-quatre formes d'onde chacune (soit, au total, deux mille formes d'onde environ). Ces tables sont produites numériquement ; on peut les filtrer et les mixer par microprocesseur, grâce à un tableau de commande mi-numérique, mi-analogique, qui satisfait à toutes les exigences et favorise un réglage plus poussé de l'utilisateur.

Le Digital Recording System permet d'enregistrer les sons et les airs joués au clavier sur huit pistes différentes, réglables indépendamment lors du play-back.

Déjà très achevé par lui-même, le Wave 2.2 donne toute sa mesure en association avec le Waveterm, qui comprend un terminal vidéographique à disquettes, et une unité de traitement des signaux audio.

Cette dernière accepte en entrée les sons captés par un microphone, les analyse et affiche le tracé de leur onde ; avant de les transmettre au Wave 2.2, on peut les corriger à l'aide de dix touches qui se trouvent au-dessous de l'écran.

On peut donc enregistrer une voix humaine, la reproduire au clavier en modifiant sa fréquence et obtenir ainsi une voix de basse, de ténor ou de soprano, quelle que soit sa hauteur initiale.

On peut d'ailleurs obtenir une reproduction d'une qualité extraordinaire. Toutefois, le Wave 2.2 ne se limite pas à reproduire les sons, et les traitements qu'on peut réaliser à l'aide de filtres, générateurs d'enveloppe et dispositifs de mixage produisent les effets les plus surprenants.

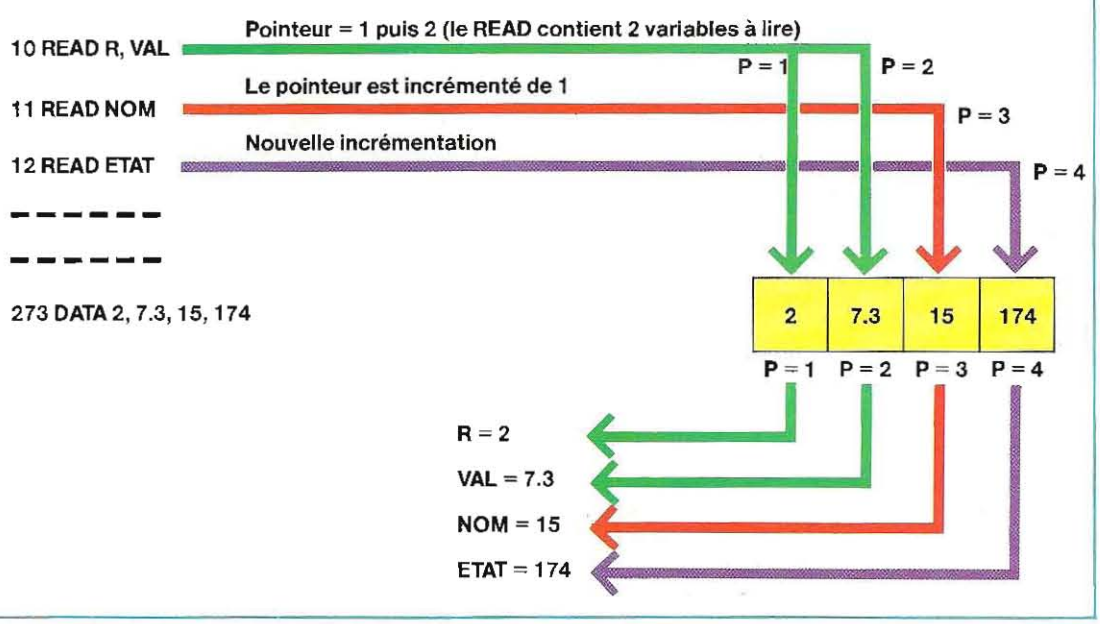
des valeurs déjà affectées au cours de l'exécution afin de pouvoir continuer dans l'ordre. On peut décomposer la ligne 10 de la façon suivante :

```
10 READ R, VAL
11 READ NOM
12 READ ETAT
...
273 DATA 2, 7.3, 15, 174
```

Le résultat est le même, mais les affectations sont réparties sur trois lignes.
Le mécanisme d'affectation des valeurs est illustré par le schéma ci-dessous. Le système associe à l'instruction DATA un pointeur (appelé P* sur la figure) qui indique la valeur à lire. P est initialisé à 1 par le premier READ et désigne d'abord la première valeur de la donnée. A chaque nouvelle affectation, une incrémentation de 1 le fait pointer vers la valeur suivante.
Comme cela apparaît sur le schéma, la première instruction READ concerne deux variables. La valeur 2 est associée à la première (P = 1). P est alors incrémenté (P = P + 1 = 2) ; il associe donc la seconde valeur (7.3) à la variable suivante, et ainsi de suite. A la dernière lecture, P prend sa valeur maximale (P = 4). D'autres tentatives d'affectations provoqueraient une erreur car P a atteint son maximum. Si l'on veut réutiliser les valeurs de la donnée, il faut réinitialiser le pointeur, ce qui sera fait avec l'instruction RESTORE.
Après l'exécution de cette instruction insérée en n'importe quel point voulu du programme, on peut de nouveau accéder à toutes les valeurs du DATA en partant de la première. L'emploi de RESTORE est illustré en haut de la page 380. Les lectures successives de cinq variables (A, B, C, X, Y), demandées aux lignes 20 et 30, déplacent le pointeur de la première valeur de l'instruction DATA (8) à la cinquième (25). La prochaine valeur lue devrait donc être 73. Mais la ligne 50 contient une instruction RESTORE qui remet le pointeur à 1 et lui fait donc indiquer la première valeur du DATA. L'instruction de lecture 60 entraîne donc l'affectation des valeurs 8, 19 et 11 aux variables E, F et G. En l'absence de RESTORE, le pointeur ne serait pas réinitialisé et il associerait aux variables de la ligne 60, la sixième valeur de DATA et les suivantes. Les instructions des lignes 40 et 70 impriment le contenu des variables.

* Le nom P et les valeurs attribuées au pointeur ne sont donnés qu'à titre d'exemples destinés à illustrer le mécanisme physique d'affectation par l'instruction DATA, qui est pris en charge par l'interpréteur et qui reste totalement transparent pour le programmeur.

READ ET DATA : MODE D'ACTION DU POINTEUR DES DONNEES



EMPLOI DE L'INSTRUCTION RESTORE (1)

```
10 OPTION BASE 1
20 READ A,B,C
30 READ X,Y
40 LPRINT "A = ";A, "B = ";B, "C = ";C, "X = ";X, "Y = ";Y
50 RESTORE
60 READ E,F,G
70 LPRINT "E = ";E, "F = ";F, "G = ";G
80 DATA 8,19,11,2.7,25,73,80,9.1
```

```
A = 8     B = 19    C = 11    X = 2.7    Y = 25
E = 73    F = 80    G = 9.1
```

En supprimant la ligne 50, on obtient

```
A = 8     B = 19    C = 11    X = 2.7    Y = 25
E = 73    F = 80    G = 9.1
```

EMPLOI DE L'INSTRUCTION RESTORE (2)

```
10 OPTION BASE 1
20 READ A,B,C
30 LPRINT "A = ";A, "B = ";B, "C = ";C
40 READ X,Y
50 RESTORE 90
60 READ K,Z
70 LPRINT "X = ";X, "Y = ";Y, "K = ";K, "Z = ";Z
80 DATA 7,9,200
90 DATA 26,48,11,90
```

```
A = 7     B = 9     C = 200
X = 26    Y = 48    K = 26    Z = 48
```

En supprimant 90 en ligne 50, on obtient :

```
A = 7     B = 9     C = 200
X = 26    Y = 48    K = 7     Z = 9
```

S'il existe plusieurs DATA, on peut préciser, derrière l'instruction RESTORE, le numéro de ligne concerné. Seules les valeurs contenues dans cette ligne et les suivantes sont alors réutilisables. Dans le programme ci-dessus, l'instruction RESTORE 90 qui a été écrite à la ligne 50, rend accès aux valeurs de DATA, seulement à partir de la ligne 90. L'instruction 40 affecte à X et Y les valeurs 26 et 48 : le pointeur désigne ensuite la troisième valeur de la ligne (11). Mais, l'instruction RESTORE 90 réinitialisant le pointeur à la première valeur du DATA, l'instruction de lecture suivante (ligne 60) attribuera donc les valeurs 26

MAUVAISE UTILISATION DES INSTRUCTIONS READ ET DATA

```
10 OPTION BASE 1
20 READ A,B
30 X=A+B
40 READ C,D,F
50 PRINT X,C,D,F
60 DATA 3,7,9,11


Out of DATA in 40
```

EMPLOI DE L'INSTRUCTION DATA POUR DES CHAINES DE CARACTERES

```
10 OPTION BASE 1
20 READ A$
30 READ B$
40 READ C$
50 READ D$
60 READ E$
70 F$=A$+C$+B$+C$
80 G$=D$+C$+C$
90 H$=E$+C$
100 PRINT F$
110 PRINT G$
120 PRINT H$
130 L$=F$+G$+H$
140 PRINT L$
150 DATA "Nom"
160 DATA "Prénom"
170 DATA ".........."
180 DATA "Rue"
190 DATA "Commune"




Nom.........Prénom.........
Rue.....................
Commune.........
Nom.........Prénom.........Rue.....................Commune....
```

et 48 à K et à Z. Si l'on ne précisait pas de numéro de ligne après RESTORE, le pointeur se positionnerait au début du premier DATA (ligne 80) et la ligne 60 déterminerait l'affectation des valeurs 7 et 9 (et non 26 et 48) aux variables qu'elle contient.

Le nombre des valeurs contenues dans une instruction DATA est au moins égal au nombre de variables à lire (s'il est supérieur, les valeurs superflues sont ignorées). Dans le cas contraire, le système diagnostique une erreur et stoppe l'exécution. Le programme présenté en haut de cette page est celui d'un programme où on tente de lire cinq variables

dans un DATA de quatre valeurs, et sans utiliser de RESTORE. L'interpréteur interrompt le programme et émet le message « Out of DATA ». On peut également utiliser l'instruction DATA pour traiter les variables alphanumériques. Les valeurs de DATA s'écrivent alors entre guillemets (le symbole " marque le début ou la fin d'une chaîne de caractères. Le second programme de la page 381 utilise des chaînes et leur affecte, par les instructions 20 et 60 les valeurs contenues dans les DATA. Les lignes 70 et 80 concatènent ces variables en insérant des points aux endroits désirés (la chaîne C$ contient 10 fois le signe), créant trois chaînes destinées à l'impression de formulaires d'état-civil. L'impression peut se faire sur une ou plusieurs lignes, selon la façon dont on combinera les différentes chaînes.

LSET et RSET

Ces instructions permettent de déplacer des variables en les alignant à gauche (LSET; L = Left), ou à droite (RSET; R = Right).

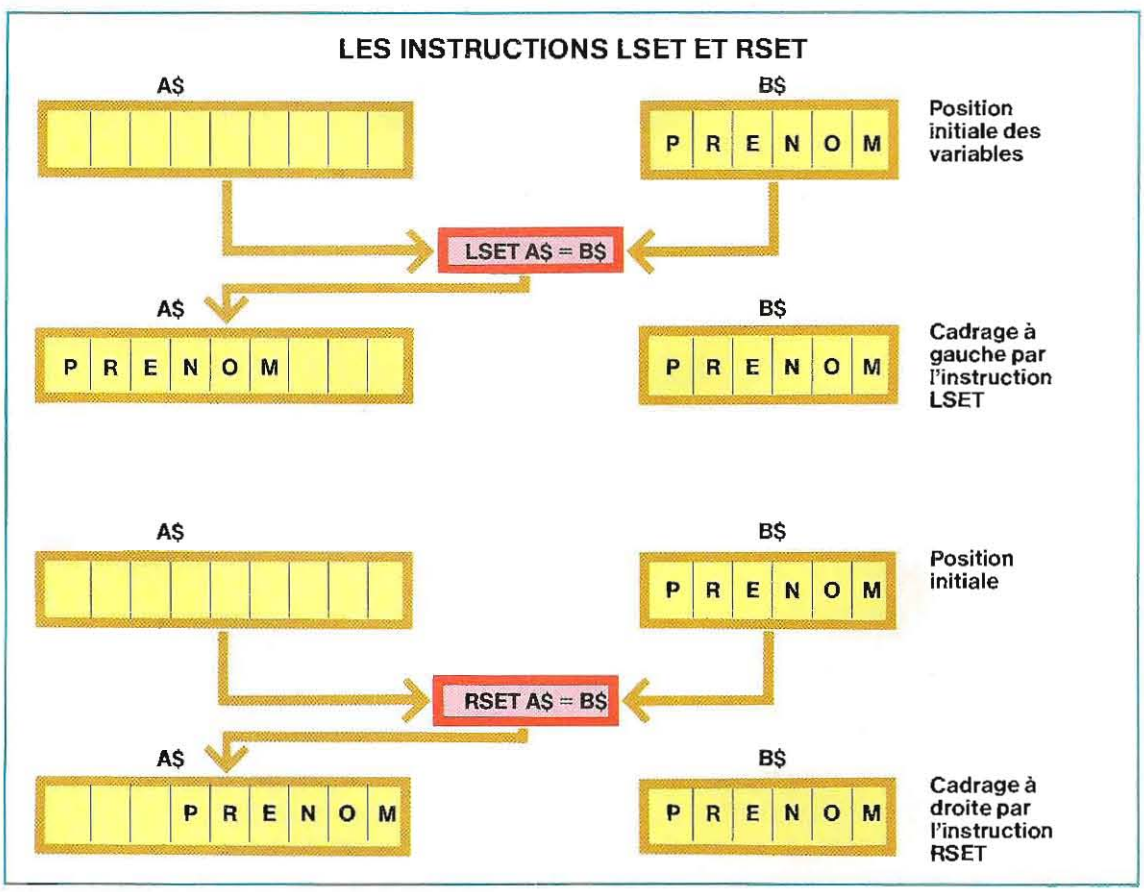
Ainsi, l'instruction

LSET A$ = B$

affecte à A$ le contenu de B$ en le cadrant à gauche (B$ demeure inchangée).
On utilise principalement ces instructions pour préparer les données à l'impression ou à la mémorisation sur disque.
On évite ainsi d'avoir recours à des commandes de mise en page spéciales quand on a besoin d'imprimer les données. De plus, LSET et RSET convertissent, tout en les cadrant, les variables numériques et alphanumériques. La plupart des systèmes ne pouvant stocker que des données en ASCII (chaînes), ces instructions s'avèrent très précieuses : on est assuré de retrouver les valeurs exactes lors de la conversion des caractères en nombre après lecture du disque.

SPACE$(N)

Cette instruction représente en réalité une fonction d'une variable entière N. Elle attribue



UTILISATION DES INSTRUCTIONS SPACE$, LSET ET RSET

```
1000 '
1010 ' ** Emploi des instructions : SPACE$ LSET RSET
1020 '
1030 ' Attribution aux variables PRÉNOM1$, PRÉNOM2$, PRÉNOM3$, PRÉNOM4$
1040 ' d'une longueur de 20 caractères et initialisation à blanc
1050 PRENOM1$=SPACE$(20):PRENOM2$=SPACE$(20):PRENOM3$=SPACE$(20)
1060 :PRENOM4$=SPACE$(20)
1070 ' Lecture de 4 prénoms. On peut écrire les 4 instructions (1090 à 1120)
1080 ' sur une seule ligne en utilisant le signe ":"
1090 INPUT A1$  ' Ces instructions lisent des chaînes de caractères et
1100 INPUT A2$  ' les affectent aux variables (A1$,A2$,A3$,A4$)
1110 INPUT A3$
1120 INPUT A4$
1130 '
1140 ' Cadrage à gauche des valeurs de A1$,...A4$
1150 ' On traite 2 variables par ligne, mais on pourrait toutes les
1160 ' traiter sur une seule ligne, ou chacune sur une ligne séparée
1170 LSET PRENOM1$=A1$:LSET PRENOM2$=A2$
1180 LSET PRENOM3$=A3$:LSET PRENOM4$=A4$
1190 '
1200 ' Impression des données telles qu'on les a entrées (A1$..A4$)
1210 '
1220 PRINT A1$:PRINT A5$:PRINT A3$:PRINT A4$
1230 PRINT:PRINT  ' Ces instructions servent à sauter 2 lignes entre les résultats
1240 ' Impression des données cadrées à gauche (PRENOM1$, etc.)
1250 '
1260 PRINT PRENOM1$:PRINT PRENOM2$:PRINT PRENOM3$:PRINT PRENOM4$
1270 PRINT:PRINT  ' Ces instructions servent à séparer les résultats
1280 ' Cadrage à droite
1290 '
1300 RSET PRENOM1$=A1$:RSET PRENOM2$=A2$
1310 RSET PRENOM3$=A3$:RSET PRENOM4$=A4$
1320 '
1330 ' Impression des données cadrées à droite
1340 '
1350 PRINT PRENOM1$:PRINT PRENOM2$:PRINT PRENOM3$:PRINT PRENOM4$
1360 '
1370 END
```

```
PIERRE
CATHERINE
JULIE
OLIVIER

PIERRE
CATHERINE
JULIE
OLIVIER

            PIERRE
         CATHERINE
             JULIE
           OLIVIER
```

à une chaîne de caractères la longueur indiquée entre parenthèses (soit N caractères) et la remplit de blancs (code héxadécimal 40, selon la table de correspondances du code ASCII). En écrivant A$ = SPACE$(10), on fixe la longueur de A$ à 10 caractères initialisés à blanc.

L'allocation des chaînes est dynamique en Basic : la longueur d'une variable alphanumérique (nombre de caractères qu'elle contient), peut donc varier au cours du programme. Les problèmes d'alignement, lors de l'impression ou de la mémorisation, sont résolus par les instructions SPACE$ et LSET (ou RSET).

Le programme de la page 383 accepte quatre prénoms entrés au clavier (A1$, etc., instructions 1090 à 1120)*. L'instruction 1220 imprime les données telles qu'on les a entrées. La deuxième écriture (1260) intervient après le cadrage à gauche et la dernière (1350) après le cadrage à droite (voir listing).

L'instruction SPACE$(N), nécessaire au paramétrage de la longueur d'une chaîne (en fonction du résultat d'un calcul, par exemple), n'existe pas sur toutes les machines. On devra alors, dans une boucle, construire une chaîne d'espaces blancs de longueur donnée. L'organigramme correspondant se trouve ci-dessous mais nous présenterons les instructions de boucle lors d'une prochaine étude.

En affectant à N la valeur 20 avant d'appeler cette routine, nous obtiendrons le même résultat qu'avec l'instruction A$ = SPACE$(20).

On trouvera page 386 un programme commenté qui établit des listes d'adresses.

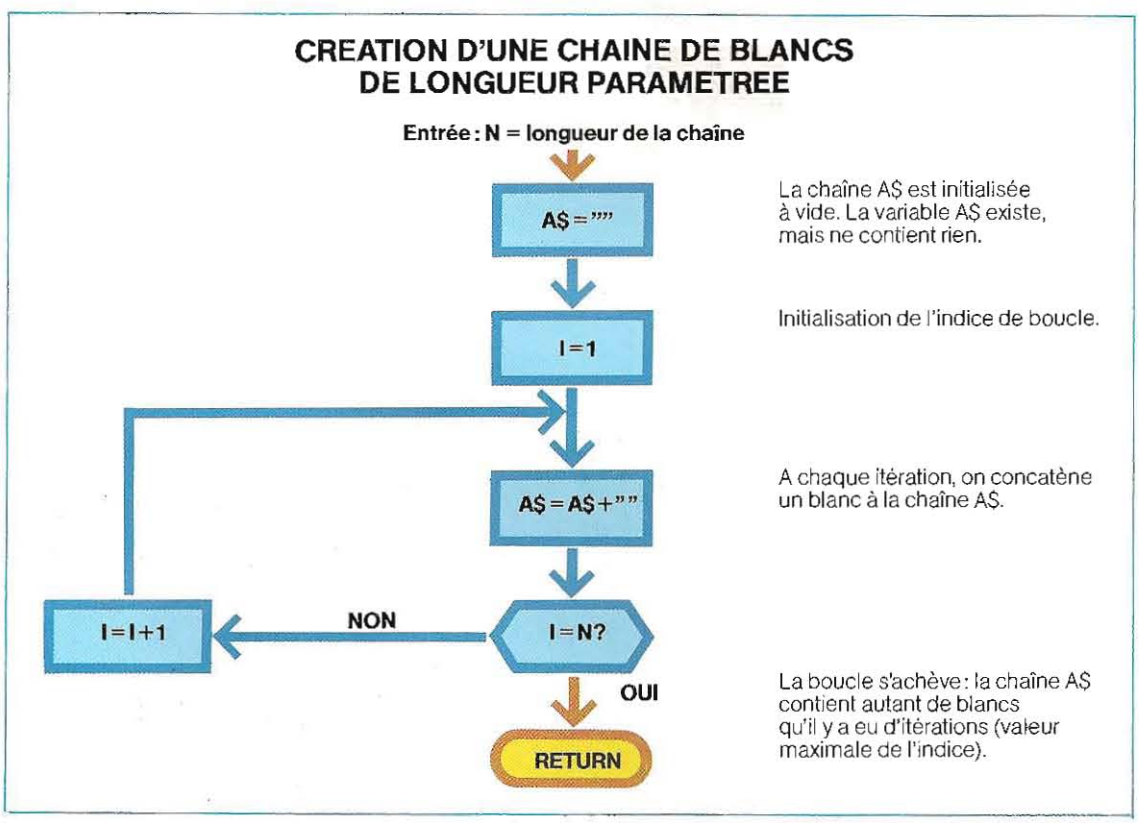
Tel qu'il est présenté, ce programme ne peut lire qu'un nom à la fois car les instructions d'itération ont été remplacées par des lignes d'astérisques (lignes 150 à 410).

SPC(N)

Comme SPACE$(N), cette instruction est une fonction. Elle a un effet semblable et c'est pourquoi nous la traitons ici, même si elle ne réalise aucune affectation.

SPC(N) permet d'imprimer ou d'afficher N blancs.

* Nous n'avons pas encore vu l'instruction INPUT. Elle permet la saisie des données au clavier et leur affectation dans une variable.



Test 11

1 / Classer les constantes suivantes selon leur catégorie (entière, simple et double précisions, chaîne)

a) 12.57 b) "Allo" c) 79.3D4 d) 9159E6 e) &H12
f) 92735 g) &177 h) &HAF i) "AF" j) %2121

2 / Si l'on pose A% = 3, B = 3, C# = 3, D$ = "3", lesquelles parmi les expressions suivantes sont erronées ?

a) D% = A% + 7 b) D% = A% + 46721 c) D% = C# * B!
d) R! = B! * 120 e) D# = D$ + D# f) H$ = "7" + D$

3 / Si le tableau A (10) est dimensionné sans autre indication, combien compte-t-il d'éléments ?

4 / A l'aide de la fonction MOD dessiner l'organigramme d'un programme déterminant la divisibilité par 2, 3, 5, 7, 11 d'un nombre quelconque entré au clavier.

5 / Généraliser le programme précédent en prévoyant l'entrée au clavier des diviseurs (10 nombres au maximum). Le programme s'articulera selon les étapes suivantes :

1) lecture du nombre de diviseurs à saisir (NMX) ;
2) boucle de lecture des diviseurs (de 1 à NMX) et leur mémorisation dans un tableau A dimensionné à 10 éléments ;
3) entrée de la valeur dont on veut vérifier la divisibilité (par chacun des nombres précédemment introduits) ;
4) contrôle de divisibilité et impression des résultats ;
5) préparation du programme pour une nouvelle entrée.

Le programme se termine lorsque le chiffre 0 est introduit comme valeur de NMX.

Les solutions du test se trouvent pages 398 et 399.

Ainsi, l'instruction PRINT SPC (6) provoque l'affichage de 6 blancs de la même manière que l'ensemble des deux instructions A$ = SPACE$(6) : PRINT A$. Dans ce cas, le contenu de la chaîne de caractères A$ (6 blancs) est imprimé. L'instruction SPC (N) reste cependant assez peu utilisée.

SWAP

Cette instruction, qui sert à échanger les valeurs de deux variables, peut être appliquée à un format quelconque (entier, réel en simple ou double précision, chaîne de caractères) à condition que les deux variables soient homogènes.

Exemple :

SWAP A, B	Echange entre eux les contenus de A et B.
SWAP A, B#	Erreur. La variable A est en simple précision alors que B est en double précision.
SWAP A$, B$	Echange les contenus des chaînes de caractères A$, B$.

En insérant l'instruction SWAP A$, B$ dans le programme de la page 386 avant l'impression, par exemple en ligne 355, le nom s'écrit (deuxième colonne) à la place du prénom, et vice versa.

Instructions de boucle

Dans ce paragraphe, seules sont présentées les deux principales d'entre elles :

FOR... NEXT...
WHILE... WEND...

prévues en Basic 80. Les instructions propres à d'autres versions du Basic feront l'objet d'une étude ultérieure.

LECTURE DES DONNEES D'ETAT CIVIL ET IMPRESSION SOUS FORME DE TABLEAUX

```
10  ' ** LECTURE DES DONNEES D'ETAT CIVIL ET IMPRESSION SOUS FORME DE TABLEAUX **
20  '
30  ' DECLARATIONS :
40  '
50  OPTION BASE 1              ' Les indices partent de la valeur 1
60  DEFINT I-N                 ' Tous les noms ayant pour initiale les lettres de I à N
70  '                           représentent des variables entières
80  ' ENTREES :
90  '
100  INPUT PRENOM$
110  INPUT NOM$
120  INPUT NUMERO$
130  INPUT RUE$
140  ' Les données brutes ne permettraient pas une impression ordonnée en tableaux
150  ' **********
160  ' Le format d'impression doit être :
170  '       Prénom          15 caractères dans la chaîne A$
180  '       Nom             15 caractères dans la chaîne B$
190  '       Numéro           4 caractères dans la chaîne C$
200  '       Rue             20 caractères dans la chaîne D$
210  '
220  ' Création des fenêtres d'impression A$, B$, etc.
230  '
240  A$ = SPACE$(15)            ' devra contenir PRENOM$
250  B$ = SPACE$(15)            ' devra contenir NOM$
260  C$ = SPACE$( 4)            ' devra contenir NUMERO$
270  D$ = SPACE$(20)            ' devra contenir RUE$
280  '
290  ' Cadrage des données lues dans les fenêtres d'impression
300  '
310  LSET A$ = PRENOM$
320  LSET B$ = NOM$
330  RSET C$ = NUMERO$    ' les chaînes de caractères représentant des valeurs
340  LSET D$ = RUE$       ' numériques sont justifiées à droite pour être lues en colonne
350  '
360  ' Les fenêtres d'impression A$, B$, etc. sont réunies pour former une seule chaîne
370  ' appelée LIGNE$
380  '
390  LIGNE$ = A$ + B$ + C$ + D$
400  PRINT LIGNE$
410  ' **********
420  END
```

```
LEON        ROUSSEAU          123 rue d'AQUITAINE

HERVE       TISSERAND        1234 rue du MAINE

ARMAND      ROY                 6 place de la VICTOIRE
```

FOR... NEXT...

Etudions plus en détail l'organigramme d'une boucle indicée classique exposée page 387. L'indice I varie entre les valeurs N1 et N2 (N2 > N1) avec un pas P. A la première exécution de la boucle, l'indice I prend la valeur N1. Successivement incrémenté du pas P, l'indice prend les valeurs I + P, I + 2P, etc., jusqu'à la valeur finale N2. La programmation de la boucle nécessite deux instructions : la première spécifie le pas (STEP), et les valeurs (N1 et N2) entre lesquelles l'indice doit varier, la seconde indique le point de « sortie de la boucle ». A ce point du programme, la valeur de l'indice est comparée à la limite supérieure (N2). Selon le résultat, on a la sortie de la boucle ou l'incrémentation de l'indice pour un nouveau passage.

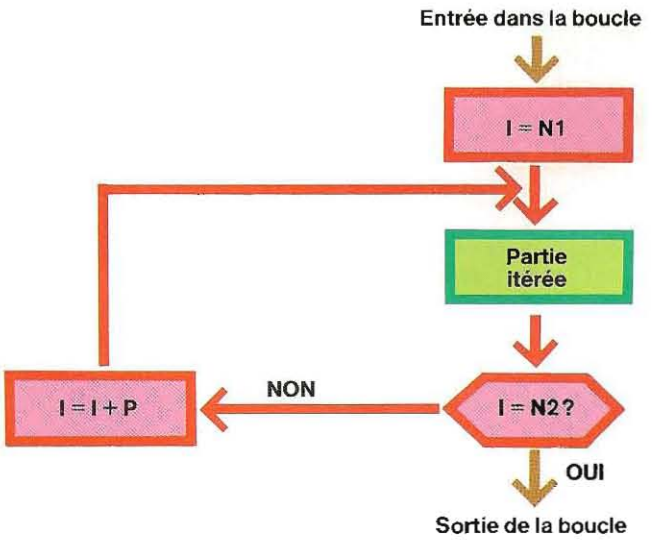
L'instruction spécifiant les paramètres est :

FOR I = N1 TO N2 STEP P

Le code opération FOR indique à la machine que l'instruction est une boucle ; l'indication I = N1 TO N2 STEP P en spécifie les modalités d'exécution (bornes, pas).
L'instruction de fin de boucle est NEXT I (NEXT = prochain).
Par conséquent, la traduction en Basic de l'organigramme est :

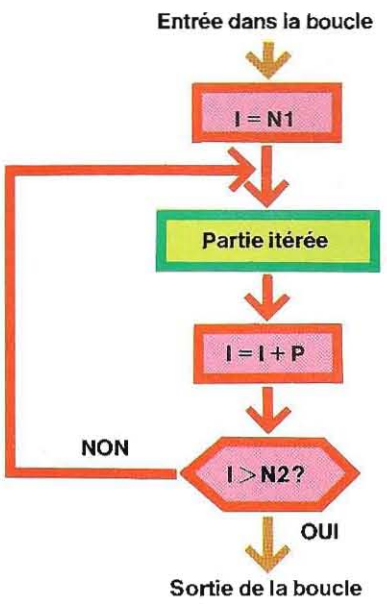
20 FOR I = N1 TO N2 STEP P

SCHEMA LOGIQUE D'UNE BOUCLE



I = Indice de boucle
N1 = Valeur initiale de l'indice
N2 = Valeur finale
P = Pas

ORGANIGRAMME DE DEROULEMENT D'UNE BOUCLE





(Partie itérée)
1270 NEXT I

Le premier schéma de la page 387 est l'équivalent logique du programme en langage machine, mais non sa forme exacte. Une comparaison avec le mécanisme d'exécution réel (schéma au bas de la page 387) met en évidence une différence dans la position du calcul de la nouvelle valeur de I (I = I + P).
Logiquement, et selon le premier schéma, ce calcul s'effectue avant le retour dans la boucle, tant que la condition I = N2 n'a pas encore été vérifiée. Lorsque l'indice I est parvenu à la valeur N2, la boucle s'interrompt avant une nouvelle itération. De cette manière, à la sortie de la boucle, I vaut donc N2.
En réalité, l'interpréteur (ou compilateur Basic) procède de manière différente: d'abord, l'indice est incrémenté (I = I + P, schéma du bas de la page 387) et on procède ensuite au test de comparaison interrompant éventuellement la boucle. En comparant les deux organigrammes, il paraît évident que dans le second cas, on ne peut pas sortir avec le test I = N2, car I prend sa valeur finale (I = N2) dans la boucle, juste après les calculs correspondant à I = N2 − 1. On sortira de la boucle sous la condition I > N2.
Précisons ici qu'à la sortie de chaque boucle, l'indice n'est pas égal à la limite supérieure (N2), mais à la limite supérieure augmentée du pas (N2 + P). Par exemple, la boucle d'indice désignée par la lettre J:

FOR J = 3 TO 11 STEP 2
(Partie itérée)
NEXT J

se termine avec la valeur J = 11 + 2 = 13.
Le déroulement de la boucle est transparent pour le programmeur: tous les tests, décisions et incrémentations sont effectués automatiquement par l'interpréteur.
Les paramètres d'une boucle (bornes et pas) doivent, dans la plupart des langages, être obligatoirement de type entier: l'instruction FOR I = 2.9 TO 3.1 STEP 0.1 est donc généralement erronée. Pour programmer une boucle à valeurs non entières, on transformera les valeurs en entiers pour l'instruction FOR, par exemple en les multipliant par 10, et on les divisera ensuite par 10 à l'intérieur de la boucle.
Le programme listé ci-dessous illustre cette technique en imprimant les nombres de 2.1 à 3 avec un pas de 0.1. L'interpréteur Basic considère les paramètres d'une boucle comme entiers. S'ils sont représentés par des noms symboliques non explicitement déclarés comme entiers (DEFINT), le système n'évaluera de toute façon que leur partie entière.
Bien qu'admise, l'utilisation de variables non entières n'est jamais avantageuse: elles occupent inutilement davantage d'espace

EXEMPLE DE BOUCLE A VALEURS NON ENTIERES

```
100  '
110  ' ** EXEMPLE DE BOUCLE A VALEURS NON ENTIERES **
120  '
130  ' Le programme imprime les nombres compris entre 2.1 et 3 par pas de 0,1 :
140  ' Valeur initiale de l'indice = 2,1  est portée à 21
150  ' Valeur finale                = 3    est portée à  30
160  ' Pas                          = 0.1  devient       1
170  ' On a donc introduit un facteur multiplicateur de 10, et il faudra
180  ' diviser l'indice avant de l'imprimer.
190  '
200  '
210  N1 = 21 : N2 = 30 : P = 1 : 19      ' Paramètres multipliés par 10
220  FOR I = N1 TO N2 STEP P             ' BOUCLE
230  R = I/10
240  LPRINT R;       ' Le symbole ; indique qu'il ne faut pas passer à
250  NEXT I          ' La ligne après l'impression
260  END


 2.1   2.2   2.3   2.4   2.5   2.6   2.7   2.8   2.9   3
```

EXECUTION D'UNE BOUCLE SUR ECRAN



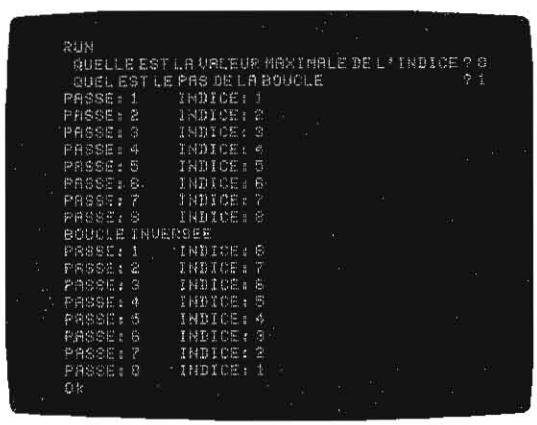
```
LIST
0  '          FICHIER = PHOTO 1
10  DEFINT A-Z
20  INPUT "QUELLE EST LA VALEUR MAXIMALE DE L'INDICE ";NF
30  INPUT "QUEL EST LE PAS DE LA BOUCLE              ";S
40  K = 0
50  FOR I = 1 TO NF STEP S
60  K = K + 1
70  PRINT "PAS: ";K,"INDICE: ";I
80  NEXT I
90  K = 0
100  PRINT "BOUCLE INVERSEE"
110  FOR I = NF TO 1 STEP -S
120  K = K + 1
130  PRINT "PAS: ";K,"INDICE: ";I
140  NEXT I
150  END
Ok
```

L'écran ci-contre fait apparaître le listage d'un programme illustrant l'utilisation d'une boucle FOR... NEXT... Le programme demande à l'opérateur les paramètres de la boucle : valeur maximale (NF) de l'indice et pas (S) à prendre en considération. Notons que le pas S de la boucle est l'incrément de l'indice I, tandis qu'en phase d'affichage (et éventuellement d'impression), le mot PASSE introduit la valeur du compteur K, qui s'incrémente de 1 à chaque itération et permet de suivre le déroulement de la bande. PASSE équivaut donc au nombre d'itérations.



```
RUN
 QUELLE EST LA VALEUR MAXIMALE DE L'INDICE ? 8
 QUEL EST LE PAS DE LA BOUCLE               ? 2
PASSE: 1     INDICE: 1
PASSE: 2     INDICE: 3
PASSE: 3     INDICE: 5
PASSE: 4     INDICE: 7
BOUCLE INVERSEE
PASSE: 1     INDICE: 8
PASSE: 2     INDICE: 6
PASSE: 3     INDICE: 4
PASSE: 4     INDICE: 2
Ok
```

Cet écran et le suivant montrent deux exécutions différentes du programme. Ici, à la question : « quelle est la valeur maximale de l'indice ? », l'opérateur indique le chiffre 8, et impose, par la saisie suivante, un pas de 2. A chaque incrémentation, le programme affichera la valeur de l'indice I à côté du compteur, dans la boucle directe, et dans la boucle inversée.



```
RUN
 QUELLE EST LA VALEUR MAXIMALE DE L'INDICE ? 8
 QUEL EST LE PAS DE LA BOUCLE               ? 1
PASSE: 1     INDICE: 1
PASSE: 2     INDICE: 2
PASSE: 3     INDICE: 3
PASSE: 4     INDICE: 4
PASSE: 5     INDICE: 5
PASSE: 6     INDICE: 6
PASSE: 7     INDICE: 7
PASSE: 8     INDICE: 8
BOUCLE INVERSEE
PASSE: 1     INDICE: 8
PASSE: 2     INDICE: 7
PASSE: 3     INDICE: 6
PASSE: 4     INDICE: 5
PASSE: 5     INDICE: 4
PASSE: 6     INDICE: 3
PASSE: 7     INDICE: 2
PASSE: 8     INDICE: 1
Ok
```

Ici, on pose NF = 8 et S = 1. Puisque le pas est divisé par deux et NF inchangé, on exécutera deux fois plus d'itérations (passes) que précédemment. A la fin de l'exécution, le système répond par le message de commande exécutée (Ok).

EXECUTION D'UNE BOUCLE INTERPRETEE ET COMPILEE

```
10  '** COMPARAISON DES TEMPS D'EXECUTION D'UNE BOUCLE
20  '    AVEC INDICE REEL ET ENTIER
30  OPTION BASE 1  ' Cette instruction sert à faire débuter
40  '                les indices à la valeur 1
50  INPUT "N avec R = REEL ";N  ' Cette instruction sera expliquée
60  '                            par la suite
70  FOR R=1 TO N STEP 2
80  NEXT R
90  PRINT N
100  DEFINT R     ' Impose à R et N le type entier
110  DEFINT N
120  INPUT "N avec R = ENTIER";N     ' Cette instruction sera
130  '                                 expliquée par la suite
140  FOR R=1 TO N STEP 2
150  NEXT R
160  PRINT N
170  END
```

Nombre d'itérations	Programme interprété (temps en secondes)		Programme compilé (temps en secondes)	
	R = Réel	R = Entier	R = Réel	R = Entier
10000	5	4	3.5	0.5
15000	8	6	5	1
25000	14	9.5	9	1
32000	17.5	12	12	1

mémoire et obligent l'interpréteur à exécuter des opérations supplémentaires (ici, extraction de la partie entière), avec la perte de temps qui en découle dans l'exécution de la boucle. Etant donné la lenteur de l'exécution du Basic interprété, toute surcharge se répercute lourdement sur le temps de déroulement des programmes. Le tableau ci-dessus, présentant, en secondes, les temps nécessaires à l'exécution d'une simple boucle, en donnera l'ordre de grandeur.

Dans les boucles, la valeur du pas, si elle n'est pas déclarée, est égale à 1, par défaut. Ainsi,

```
FOR I = N1 TO N2 STEP 1
FOR I = N1 TO N2
```

sont deux instructions équivalentes.

Les valeurs limites de l'indice (N1, N2) peuvent être calculées dans l'instruction même. Par exemple :

```
FOR I = 5 + N  TO  20 + N
```

est une forme valide de limites paramétrées. Les bornes de la boucle dépendent alors de la valeur de N, variable à laquelle on affecte une valeur avant d'entrer dans la boucle. Une forme particulière de calcul des bornes d'une boucle se présente lorsque l'une des deux est calculée à partir de la variable d'indice I. Par exemple, avec les instructions :

```
10  I = 2
20  FOR  I = 6  TO  I + 25
```

la limite supérieure (I + 25) est obtenue d'après la valeur affectée à l'indice en ligne 10

(instruction 10 I = 2). Généralement l'interpréteur Basic calcule en premier la valeur de la limite supérieure, et ensuite seulement celle de la borne inférieure. L'instruction 10 affecte à I la valeur 2; par conséquent, en ligne 20 (I + 25), la limite supérieure est 2 + 25 = 27. La boucle se déroulera entre les valeurs I = 6 (I = 6 est établi à la ligne 20 après le calcul I + 25) et I = 27.
D'autres interpréteurs, au contraire, calculent d'abord la valeur initiale et ensuite celle finale; en ce cas, l'instruction 10 ne servirait à rien car, à la ligne suivante, on écrit I = 6. L'autre limite vaudra donc I + 25 = 31, et la boucle se déroulera entre 6 et 31.

Cette dernière méthode est caractéristique des interpréteurs Basic antérieurs aux versions actuelles, qui proposent parfois l'autre système.
Les boucles peuvent être « imbriquées » c'est-à-dire contenues les unes dans les autres, à condition que les premières ouvertes soient fermées en dernier (voir le chapitre consacré aux organigrammes).
Page 391, on a reproduit le listing de deux programmes : le premier est correct, le second contient une erreur dans la clôture des boucles. La page 384 illustrait un organigramme de génération d'une chaîne de longueur paramétrée, contenant seulement des blancs. La

EXEMPLES DE TROIS BOUCLES IMBRIQUEES

```
100 '
110 ' ** EXEMPLE DE TROIS BOUCLES IMBRIQUEES **
120 '
130 DEFINT I-N          ' Les indices sont déclarés comme entiers
140 FOR I = 1 TO 5      ' Première boucle (sera fermée la dernière en ligne 230)
145 PRINT               ' Sert à séparer la boucle de 1 à 5
150 PRINT I             ' Imprime l'indice de boucle de plus haut niveau
160 FOR J = 3 TO 5 STEP 2    ' Deuxième boucle (fermée ligne 220)
170 K = I * J
180 PRINT K;            ' Le symbole ; impose l'écriture sur une seule ligne
190 FOR L = 10 TO 12    ' Troisième boucle (fermée ligne 210)
200 PRINT L;
210 NEXT L
220 NEXT J              ' Comparer la séquence des fermetures avec la position
230 NEXT I              ' de début des boucles correspondantes

RUN


 1
 3  10  11  12  5  10  11  12  7  10  11  12
 2
 6  10  11  12  10  10  11  12  14  10  11  12
 3
 9  10  11  12  15  10  11  12  21  10  11  12
 4
 12  10  11  12  20  10  11  12  28  10  11  12
 5
 15  10  11  12  25  10  11  12  35  10  11  12


100 DEFINT I-N
110 FOR L = 3 TO 21
120 A = 2 * L + 10
130 FOR N = 7 TO 15
140 B = N * A
150 NEXT L              ' Les clôtures des deux boucles sont inversées,
160 NEXT N              ' générant ainsi une erreur

RUN

NEXT without FOR in 160
```

SOUS-PROGRAMME 1000 DE GENERATION DE CHAINES

```
1000 '
1010 ' ** SOUS-PROGRAMME DE GENERATION DE CHAINE= **
1020 '
1030 ' ENTREES :
1040 '          LS = Longueur (en caractères) de la chaine
1050 '          B$ = Symbole qui doit remplir la chaine
1060 '
1070 ' SORTIES :
1080 '          A$ = Chaine de longueur désirée remplie par le
1090 '               caractère B$
1100 A$=""       ' La chaine est initialisée à vide
1110 FOR I=1 TO LS       ' BOUCLE sur la LONGUEUR
1120 A$=A$+B$            ' Le symbole B$ est concaténé à A$, un
1130 '                     nombre de fois égal à LS
1140 NEXT I
1150 RETURN
1160 '
```

EXEMPLE D'UTILISATION DU SOUS-PROGRAMME 1000

```
100 '
110 ' ** EXEMPLE D'UTILISATION DE LA ROUTINE 1000 (generation d'une chaine) **
120 '
130 DEFINT I-N
140 B$="*"       ' La chaine voulue sera constituée du caractère *
150 LS=15        ' répété 15 fois
160 GOSUB 1000   ' Appel du sous-programme 1000
170 LPRINT A$
180 '
190 B$="X"       ' Nouveau symbole
200 GOSUB 1000   ' On n'assigne pas de nouvelle valeur à LS
210 LPRINT A$    ' La longueur reste donc 15
220 '
230 B$="1"
240 LS=5
250 GOSUB 1000   ' A cet appel, symbole et longueur ont tous deux changé
260 LPRINT A$
270 '
280 LS=280
290 GOSUB 1000   ' Cet appel générera une erreur dans la routine ; LS=280
300 LPRINT A$    ' alors que les chaines ont au maximum 255 caractères
310 '
320 END
```

traduction en Basic de cet organigramme passe par l'utilisation des boucles ; l'unique adjonction nécessaire est un paramétrage ultérieur.

Ce paramétrage permettra de transmettre la longueur de la chaîne, mais aussi le symbole (unique) que l'on désire y faire apparaître. De cette façon, le programme pourra générer des chaînes de longueur voulue, et contenant un symbole quelconque. Sur cette page, figure le listing de cette routine, suivi d'un programme illustrant un exemple d'utilisation de cette routine. Le dernier appel au sous-programme, avec un paramètre de valeur erronée (LS > 255), générera une erreur si on ne prévoit pas un contrôle avant la boucle créant la chaîne, et le programme s'arrêtera. La meilleure parade consiste à prévoir, en entrée du sous-programme, un contrôle de validité des paramètres transmis par le programme appelant.

En cas d'erreur, on affectera aux paramètres une valeur conventionnelle, et un indicateur d'erreur sera activé.

De cette manière, l'exécution du programme n'est pas stoppée et l'indicateur d'erreur,

transmis au programme appelant, permettra de déterminer quelle routine a été appelée avec des valeurs erronées, et quelles sont les corrections possibles. Le schéma du bas de cette page et celui de la page 394 reproduisent les modifications apportées à l'organigramme initial pour inclure le contrôle du paramètre LS.

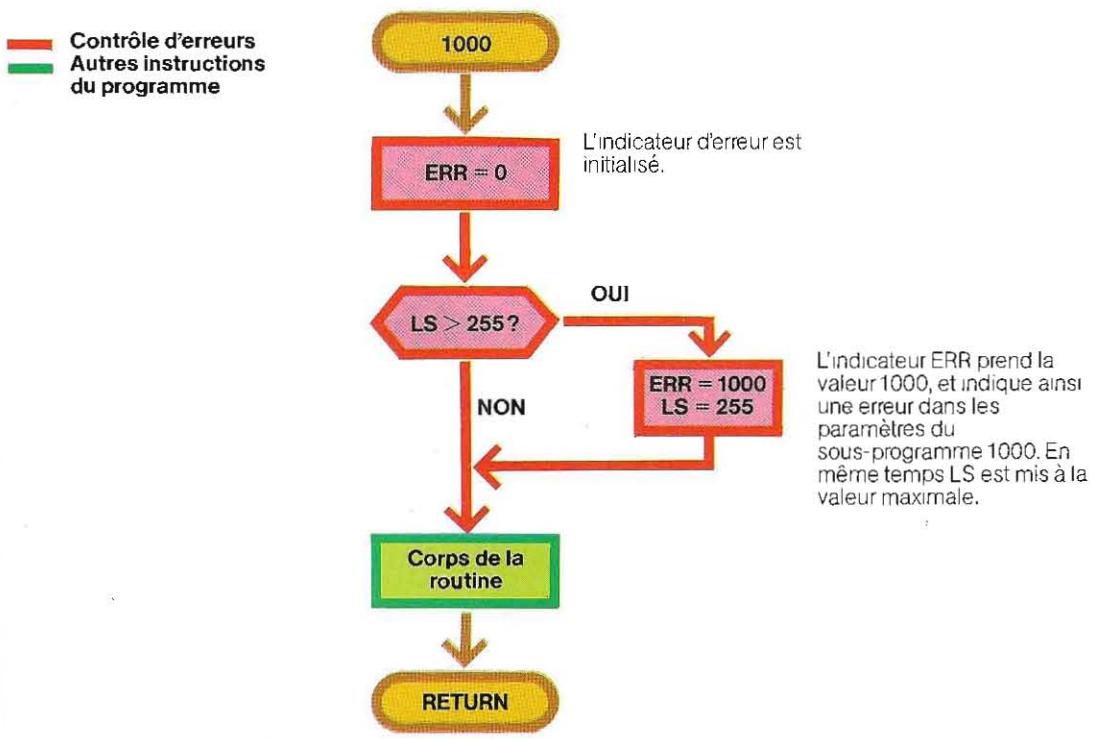
WHILE... WEND

La séquence des instructions WHILE... WEND active une autre forme de boucle : son exécution est conditionnée par la valeur (vrai ou faux) d'une expression donnée (résultat d'un calcul, valeur d'un indicateur, etc.). Une condition est « vraie » quand sa valeur est différente de zéro ; au contraire, elle est « fausse » quand sa valeur est zéro. La condition de vérité est souvent indiquée par le terme anglais true (vrai), signifiant non-zéro ; la condition opposée est false (faux) signifiant zéro.
La forme de cette instruction est :

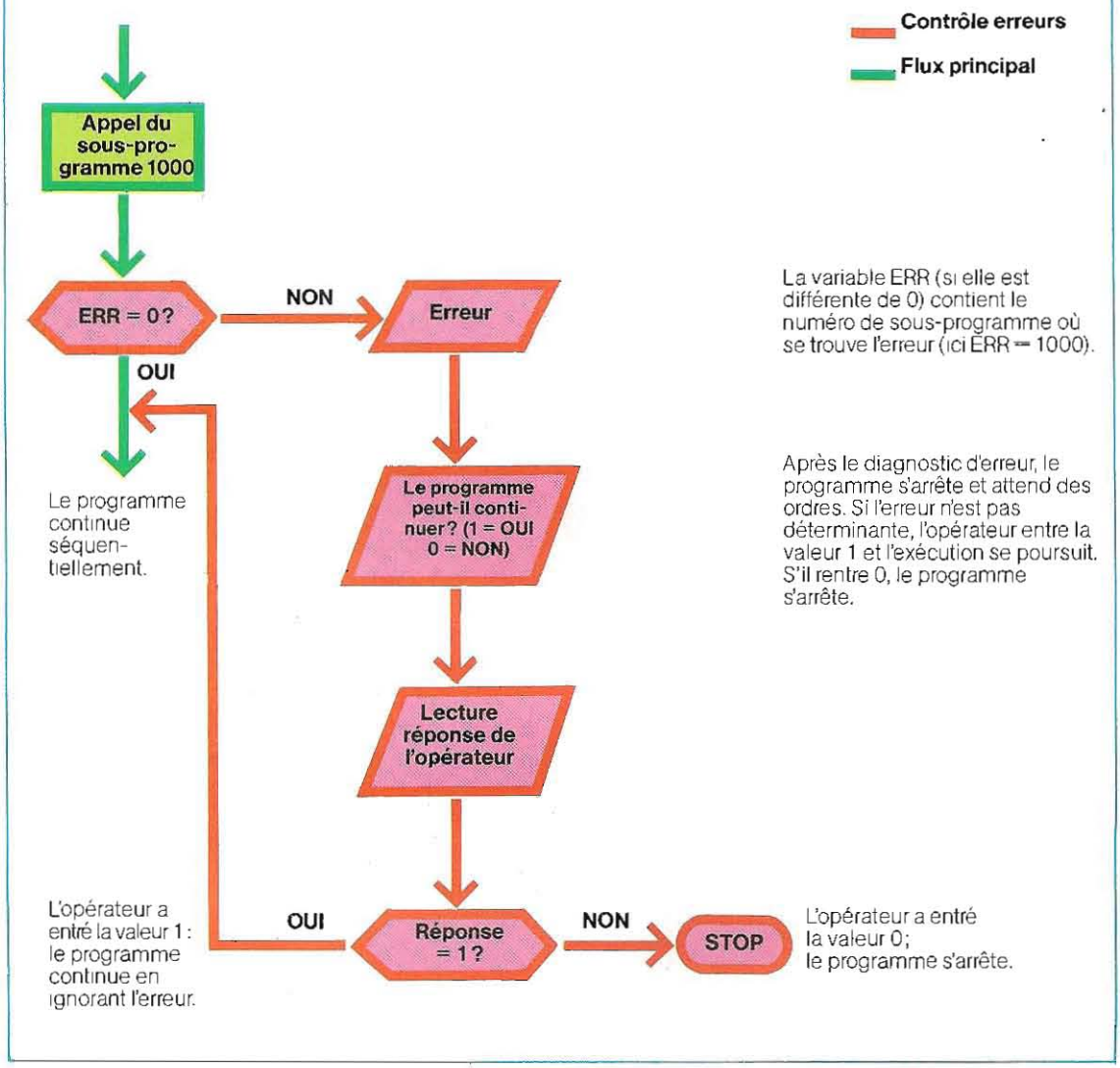
```
10  WHILE condition
(Partie itérée)
120 WEND
```

Le calcul (ou le paramètre) indiqué par le terme général de « condition » est évalué en début de boucle. S'il a une valeur différente de zéro, toutes les instructions comprises entre WHILE (ligne 10) et WEND (ligne 120) sont effectuées.
A ce point, l'exécution du programme remonte ensuite à l'instruction WHILE pour un nouveau test sur la valeur de « condition ». Si le résultat est différent de zéro, les instructions sont à nouveau exécutées. L'exécution du programme suit le chemin décrit jusqu'à ce que la condition soit fausse. A ce point, la boucle est abandonnée et le programme continue en séquence, à partir de la ligne suivant l'instruction WEND (while end).
Le schéma de la page 395 présente l'organigramme de la boucle, dans lequel la variable C représente le terme condition. La structure et le fonctionnement de cette boucle sont très différents de l'autre boucle FOR... NEXT. Sur cette dernière, les limites, et donc le nombre d'exécutions de la boucle, sont prédéterminées (bornes fixes de l'indice), tandis qu'avec

EXEMPLE D'UTILISATION DU SOUS-PROGRAMME DE GENERATION DE CHAINES



EXEMPLE D'APPEL DU SOUS-PROGRAMME 1000 AVEC CONTROLE DES ERREURS



l'instruction WHILE l'itération peut se poursuivre un nombre de fois déterminé, jusqu'à la modification de la valeur représentée par « condition ». Le traitement itératif doit naturellement contenir des instructions appropriées qui, à un moment donné, modifieront la valeur prise par la condition, et provoqueront ainsi la sortie de la boucle. Le schéma de la page 396 représente l'organigramme d'un exemple d'application du programme listé en bas de la page 395. Le programme doit lire les enregistrements d'un fichier de données contenant les noms et les adresses d'un répertoire. Il doit ensuite imprimer les valeurs lues, tant qu'il s'agit de la ville de Paris.

Dès que le nom de la ville n'est plus Paris, la boucle de lecture de données sera abandonnée, et l'exécution du programme se poursuivra séquentiellement.

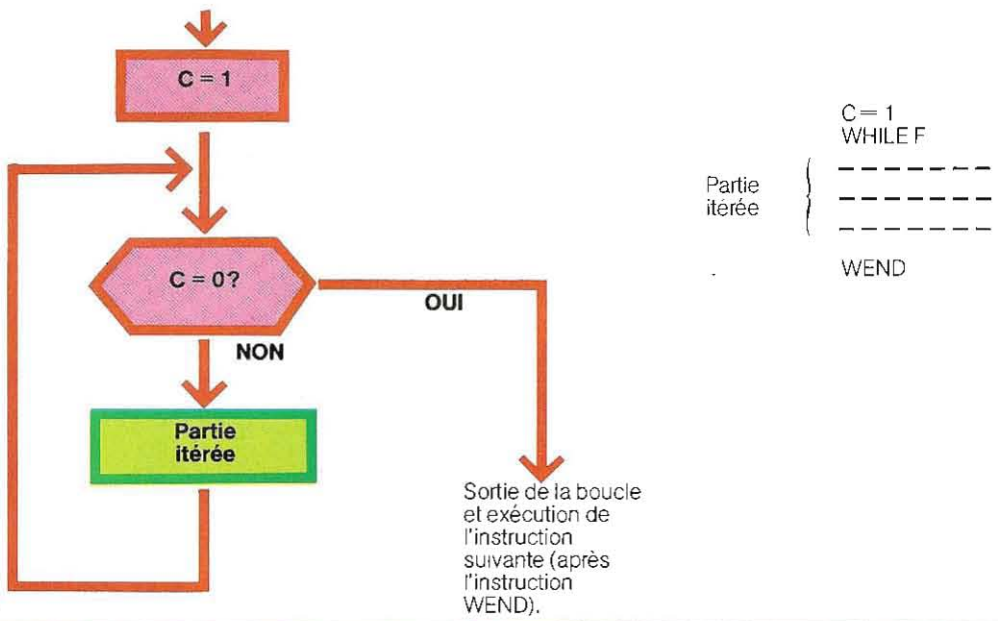
Pour simplifier le programme, on a omis, dans le listing de la page 395, toutes les instructions de gestion du disque mentionnées sur l'organigramme qui figure page 396, et les données sont entrées au clavier grâce à l'instruction 70 .

Instructions de saut

On désigne ainsi les instructions qui permettent de « sauter » d'un point à l'autre du pro-

gramme. Ce type d'instruction ne doit pas être confondu avec l'appel d'un sous-programme, même s'il lui ressemble.
Avec l'appel d'un sous-programme, le contrôle est transféré aux instructions de la routine, qui peuvent se trouver en un point quelconque du programme, mais RETURN fait reprendre l'exécution à la ligne qui suit celle qui contient l'appel. En fait, aucune instruction n'est « sautée ». L'exécution d'une partie du programme sera seulement reportée au retour de la routine. Par contre, dans les instructions de saut (ou de branchement), il n'existe pas de retour automatique. L'exécution du programme se poursuit séquentiellement depuis la ligne du saut en direction des

ORGANIGRAMME DE L'INSTRUCTION WHILE... WEND

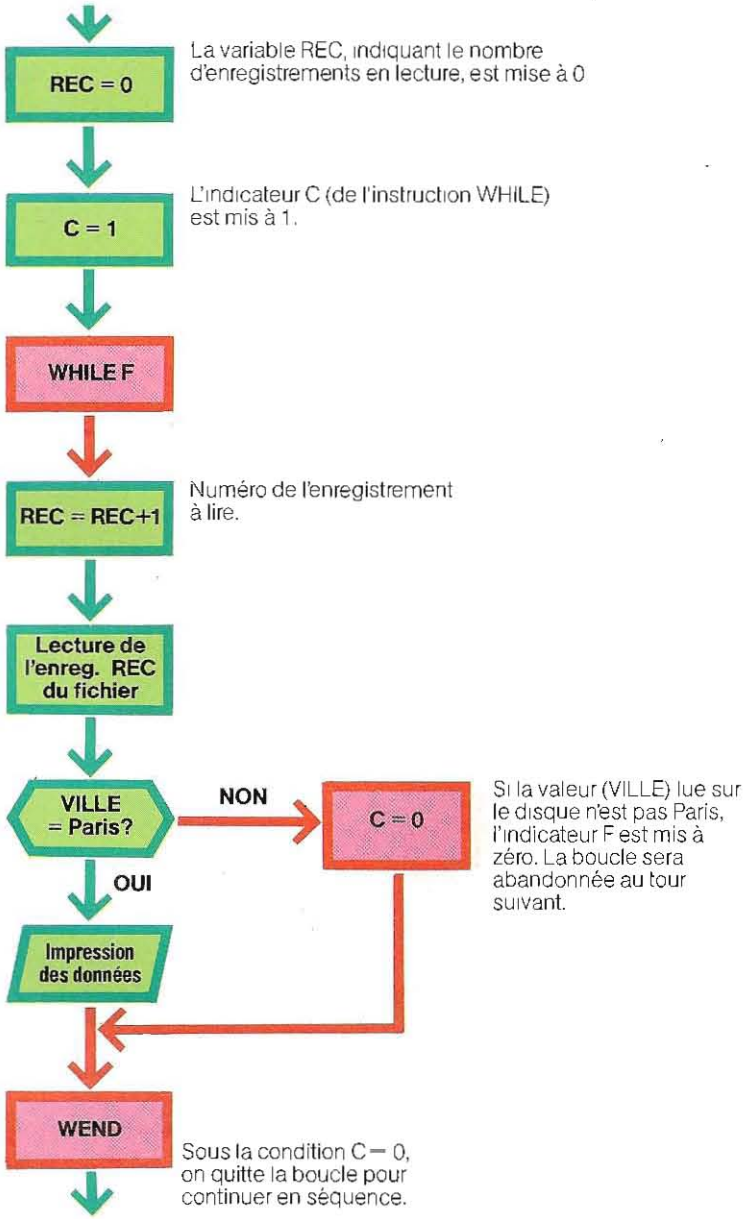


PROGRAMME EMPLOYANT LES INSTRUCTIONS WHILE... WEND

```
10  ' ** EXEMPLE DE BOUCLE WHILE-WEND **
20  '
30  ' FICHIER = TEST
40  '
50  C=1             ' Initialisation de l'indicateur
60  WHILE C         ' Début de la boucle
70  INPUT "VILLE";VILLE$
80  LPRINT  "VILLE = " ;VILLE$
90  IF VILLE$ <> "PARIS" THEN LPRINT "CONDITION FAUSSE" :C=0
100  ' La condition de non-egalite arrete la boucle
110  IF VILLE$="PARIS" THEN LPRINT "CONDITION VRAIE"
120  WEND
130  LPRINT  "FIN DU PROGRAMME, VILLE = ";VILLE$, "CONDITION FAUSSE"
140  END
```

```
VILLE = PARIS CONDITION VRAIE
VILLE = PARIS CONDITION VRAIE
VILLE = PARIS CONDITION VRAIE
VILLE = LILLE CONDITION FAUSSE
FIN DU PROGRAMME, VILLE = LILLE          CONDITION FAUSSE
```

APPLICATION DE L'INSTRUCTION WHILE... WEND



numéros de ligne plus élevés, et les instructions du programme qui ont été sautées ne sont plus exécutées, à moins qu'elles ne soient expressément rappelées par le programmeur grâce à une autre instruction de branchement.

Le but principal de ces instructions de saut est de permettre des parcours différents à l'intérieur d'un même programme.

Le choix d'un parcours déterminé peut se faire en fonction du résultat de calculs particuliers ou en fournissant des valeurs clés à la machine durant la saisie des données.

GOTO...

Littéralement traduit par « allez en... », c'est l'instruction de saut la plus simple. La syntaxe est GOTO n, où n représente le numéro de la ligne à laquelle doit se poursuivre l'exécution du programme. Par exemple, GOTO 153 con-

duit le programme en ligne 153, et la partie comprise entre GOTO et l'instruction 153 ne sera pas exécutée. Dans l'utilisation de cette instruction, il est important de se souvenir de l'existence de la ligne vers laquelle on achemine l'exécution (il suffit d'écrire un simple mot). L'instruction GOTO est dite « de saut inconditionnel », car elle est toujours exécutée. Il existe également des instructions de saut « conditionnel », qui ne sont exécutées que sous certaines conditions.

ON... GOTO...

Cette nouvelle instruction de saut se comporte comme la précédente, avec, en plus, la possibilité d'adresser simultanément des lignes différentes, selon la valeur d'une variable. Une forme complète de l'instruction pourrait être, par exemple :

ON V GOTO 100, 250, 300, 1610, 2000

qui brancherait l'exécution selon la valeur de V à l'une des lignes explicitées.

Pour V = 1, on saute à l'instruction 100 ; pour V = 2, on saute à la ligne 250 et ainsi de suite, jusqu'à V = 5 qui provoque un saut à l'instruction 2000.

Si V est nul, ou supérieur au nombre de lignes d'arrivée figurant dans l'instruction (5 dans cet exemple), le programme continue avec l'instruction qui vient immédiatement après. Par exemple, dans l'instruction précédente, si V vaut 0 ou 6 (aucun saut n'étant demandé), l'exécution du programme se poursuit séquentiellement.

Les limites de validité du paramètre (V dans l'exemple) sont 0 et 255 ; des valeurs négatives ou supérieures à 255 génèrent une erreur, et le programme s'arrête.

La variable utilisée comme élément de sélection (V dans notre exemple) est toujours considérée comme entière.

L'instruction ON... GOTO... peut contenir aussi une expression à la place de la variable. En ce cas, l'évaluation de l'expression a d'abord lieu, puis le saut est exécuté en prenant le résultat comme indicateur. Par exemple, dans l'instruction :

ON K + 6 GOTO 10, 12, 21, 70

la somme K + 6 est d'abord évaluée, et ensuite, se fondant sur le résultat, le saut correspondant est exécuté (pour K + 6 = 1, on saute à 10, pour K + 6 = 2, on saute à 12, etc.).

L'exemple précédent nous indique une méthode pour utiliser des paramètres négatifs de l'indice. On ne peut obtenir le résultat K + 6 = 1 qu'avec K = − 5 (− 5 + 6 = 6 − 5 = 1). Si, à cette valeur négative, qu'on ne peut mentionner directement dans l'instruction, on ajoute une quantité opportune, on retrouve des paramètres positifs (de 1 à n). Par exemple, si l'on veut sélectionner un saut par un nombre compris entre − 5 et − 10, il faudra ajouter 11, obtenant ainsi un paramètre qui variera entre 1 (11 − 10) et 6 (11 − 5).

La solution qui apparaît la plus évidente, consistant simplement à inverser le signe, ne donne pas le même résultat.

En cas de variation de − 5 à − 10, un changement de signe produirait une valeur comprise entre 5 et 10, adressant depuis le cinquième numéro de ligne de GOTO jusqu'au dixième. En revanche, en ajoutant 11 à notre variable, le résultat est compris entre 1 et 6, et permet l'adressage du premier numéro de ligne jusqu'au sixième. Les deux situations sont comparées ci-dessous :

- **Première méthode (addition d'une valeur positive)**
  ON K + 11 GOTO 120, 36, 100, 10, 21, 60
  Pour K variant entre − 10 et − 5, la quantité K + 11 varie de 1 (l'instruction 120 est exécutée) à 6 (l'instruction 60 est exécutée).

- **Seconde méthode (changement de signe)**
  L = − K

  ON L GOTO ?, ?, ?, ?, 60, 21, 10, 100, 36, 120

  La valeur de L est l'opposé de K (elle est donc positive). Pour K variant entre − 10 et − 5, le paramètre L varie de 10 à 5. Pour respecter la correspondance entre les valeurs de K et les lignes à exécuter, il faut insérer quatre numéros de ligne fictifs et inverser l'ordre des numéros valides par rapport au cas précédent. Les quatre premières lignes indiquées par le symbole ?, ne seront jamais adressées, étant donné le champ possible des valeurs prises par L (de 10 à 5).

La première méthode (addition d'une valeur positive) est plus fiable et présente des risques d'erreur moindres.

Netteté et simplicité restent cependant deux conditions essentielles à une bonne programmation. Il convient donc parfois d'ajouter quelques lignes de plus, et d'allonger ainsi le

Solutions du test 11

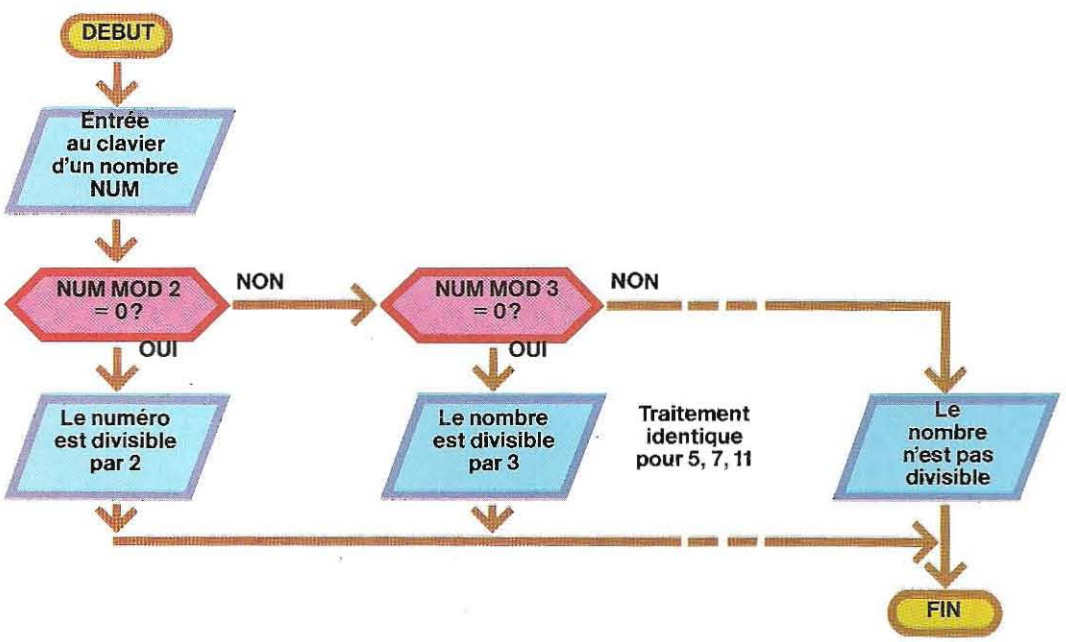
1 / a, d, f = réels en simple précision ; j = entier ; b,i = chaînes de caractères ; c = réel en double précision ; g = octal ; e,h = hexadécimaux.
Notons toutefois que, selon le Basic utilisé, certaines différences sont possibles (& peut marquer un nombre hexadécimal et &O un nombre octal, par ex.).

2 / Les expressions erronées sont :
b) D% et A% sont entières alors que le montant 46721 ne peut pas l'être (il dépasse 32767). Une erreur de débordement sera diagnostiquée par le système ;
c) D% est entière alors que C# est un réel en double précision et B! un réel en simple précision. Toutefois, le produit donne un résultat égal à 9 et donc, dans ce cas particulier, le calcul est quand même exécuté ;
e) D# est un réel alors que D$ est une chaîne. On ne peut effectuer aucune opération entre une chaîne (représentée en ASCII) et une variable numérique.
En revanche, l'expression H$ = "7" + D$ donne comme résultat H$ = "73". Pour les chaînes de caractères, le symbole + signifie concaténation, et il génère une chaîne formée de la succession des caractères des deux chaînes.

3 / La numérotation des tableaux part de 0. Aussi, sauf indication contraire, A(10) contient 11 éléments (de l'élément numéro 0 à l'élément 10).

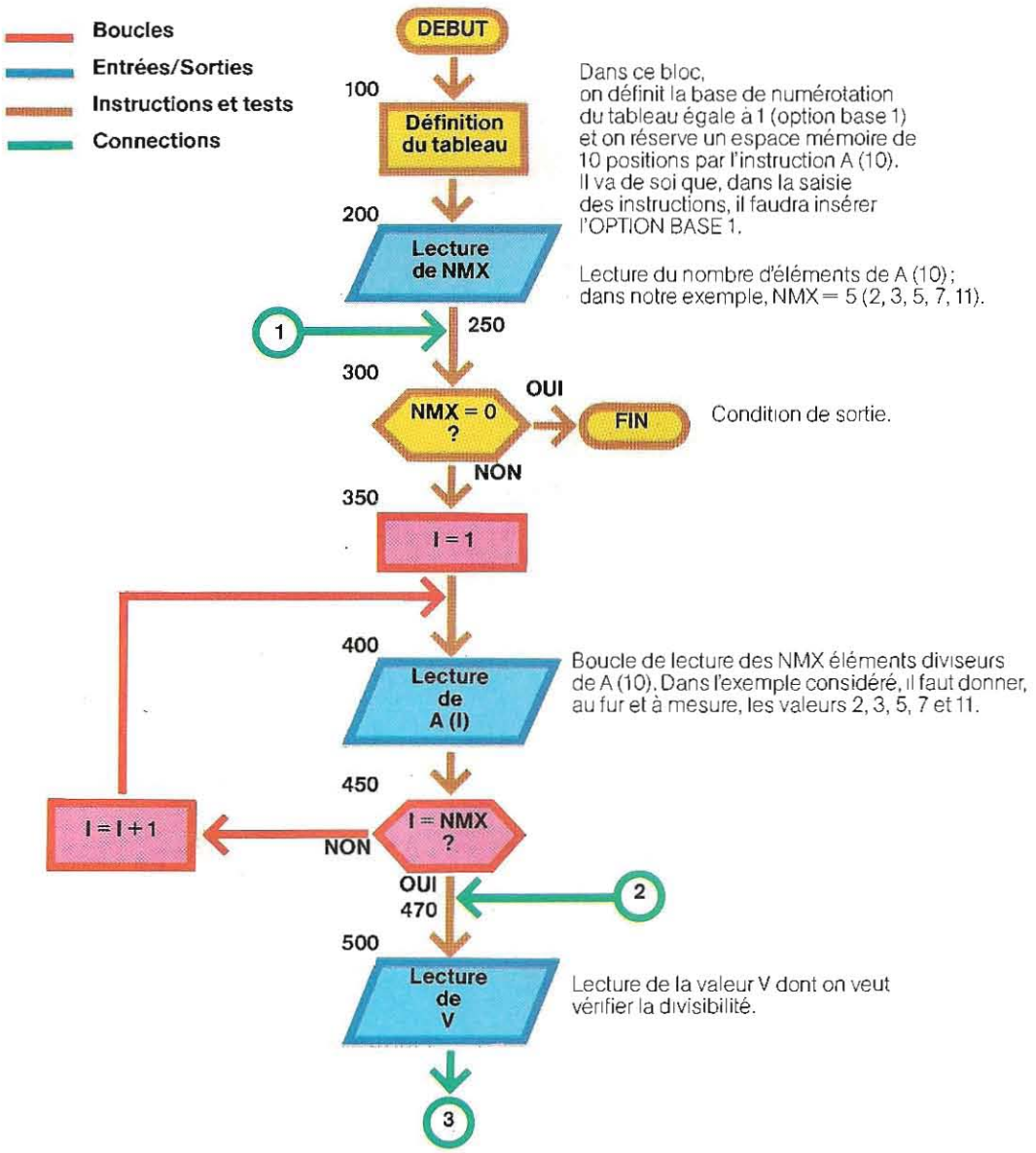
4 / L'organigramme ci-dessous prévoit une série de calculs des valeurs NUM MOD d (pour les diviseurs d valant successivement 2, 3, 5, 7, 11). Chaque évaluation est suivie d'un test : si le résultat est nul, le nombre saisi (NUM) est divisible par d. On ne le testera pas avec les diviseurs suivants, qui, cependant, répondent peut-être à la question (si NUM = 165, par exemple, il est divisible par 3, mais également par 5 et par 11). Cette façon de dérouler le programme n'est donc donnée qu'à titre indicatif.

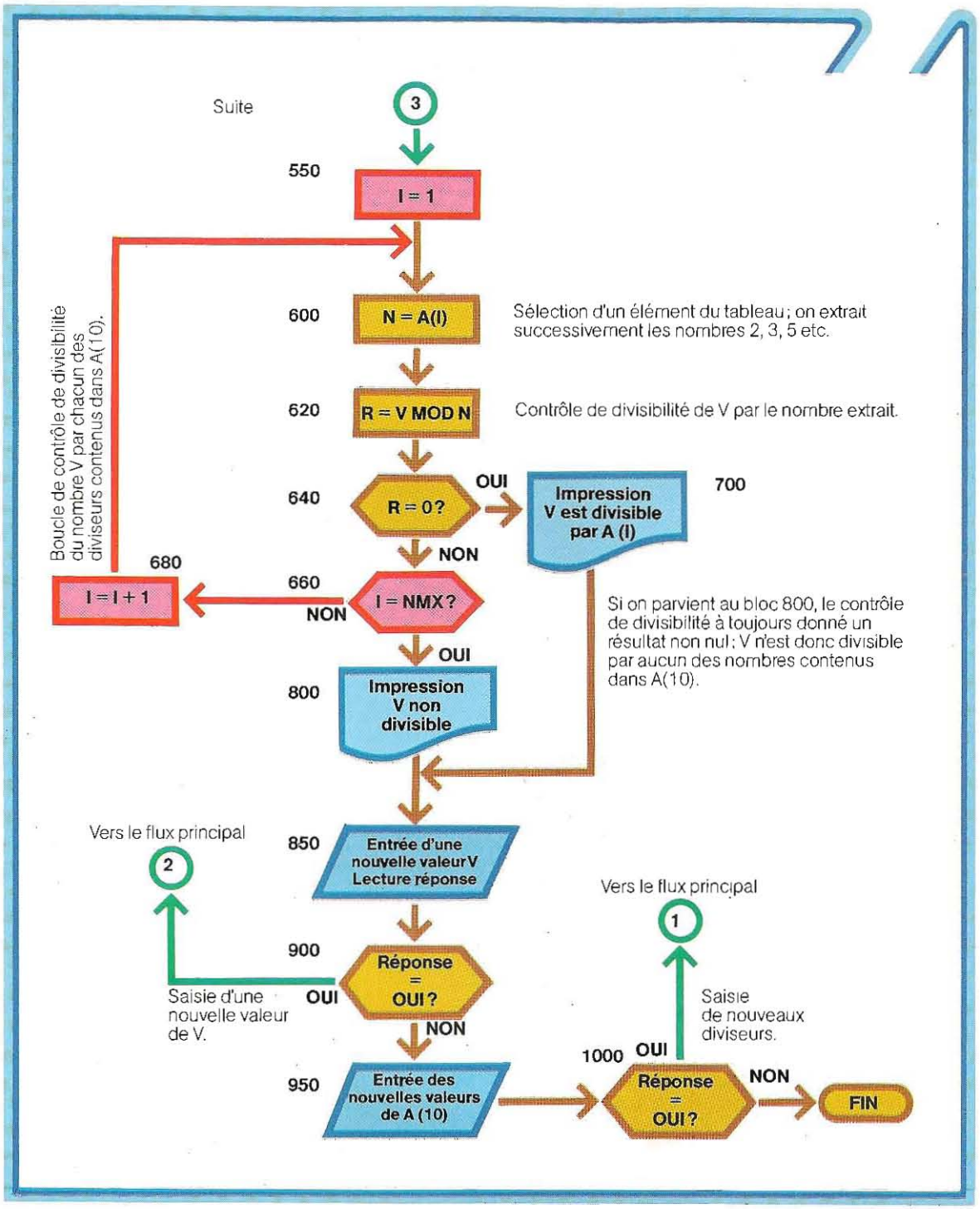
PROCEDURE DE CONTROLE DE DIVISIBILITE (PREMIERE VERSION)



5 / Sur l'organigramme précédent, le même calcul et le même test sur zéro sont répétés cinq fois. Pour les écrire une seule fois, nous allons paramétrer les fonctions utilisées (MOD et test zéro). De cette façon, dans des situations différentes, il suffira d'actualiser les valeurs numériques (paramètres) propres au calcul en cours. Sur ce point, le second organigramme (pages 399 et 400) peut être considéré comme optimisé de façon satisfaisante.

PROCEDURE DE CONTROLE DE DIVISIBILITE (SECONDE VERSION)





programme, afin d'éviter des instructions trop complexes et peu lisibles.
Une application possible de cette instruction ON ... GOTO ... est suggérée dans le sous-programme de calcul des surfaces de certaines figures planes (voir page 373).

Sur le listing correspondant, on s'est borné à indiquer que la sélection du type de figure se fait au moyen du paramètre K. Pour K = 1, on veut le carré (instruction 1660) ; pour K = 2, le rectangle (instruction 1680), et ainsi de suite jusqu'à la valeur K = 5, correspondant au

cercle (instruction 1770).
La sélection se programme par l'instruction :

ON K GOTO 1660, 1680, 1710, 1740, 1770

Le schéma ci-dessous présente l'organigramme complet et le listing comprenant la nouvelle instruction. Tous les commentaires précédents ont été supprimés, et les instructions de calcul (1670, 1700, etc.), qui restent inchangées, n'ont pas été reproduites. La ligne 1656 est un piège (trap) qui arrête le programme si une valeur de K erronée (hors de l'intervalle 1-5) survient.

Instructions conditionnées

On dit d'une instruction qu'elle est conditionnée quand son exécution dépend de la valeur de certaines variables.
Toutes les instructions peuvent être conditionnées sous la forme suivante :

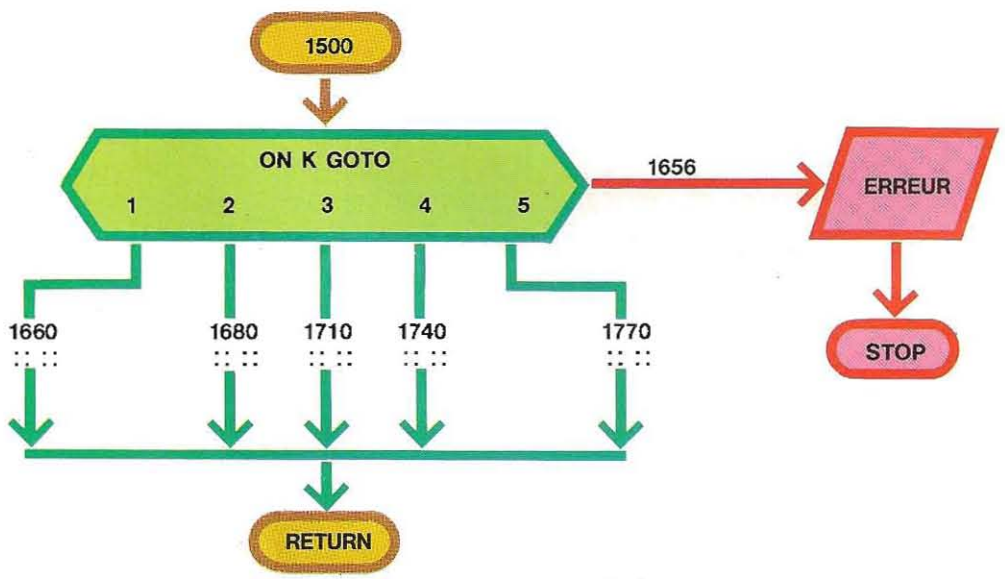
IF condition THEN instruction

Le mot IF (si) et THEN (alors) indiquent à la machine que l'instruction est subordonnée à une condition ; il faudra donc évaluer et vérifier cette condition à l'exécution. L'instruction qui suit le code THEN sera alors exécutée ou sautée.
La condition peut être représentée par une simple variable, par un calcul arithmétique (dans ces deux cas, la condition est vraie si sa valeur est différente de zéro), ou par une expression logique. D'autres exemples typiques seront donnés page 406.

IF V < > 0 THEN PRINT "Non zéro"
Si V est différent de zéro, imprimer la phrase "Non zéro"
IF A > B THEN C = A + B
Le calcul C = A + B est effectué seulement si A est supérieur à B
IF A = B THEN 1020
Si A = B, le programme saute à l'instruction 1020.

Cette dernière instruction est un exemple de saut conditionnel (voir instruction GOTO) et peut être écrite sous une autre forme, plus explicite : IF A = B GOTO 1020.
On remarquera que seule l'instruction GOTO

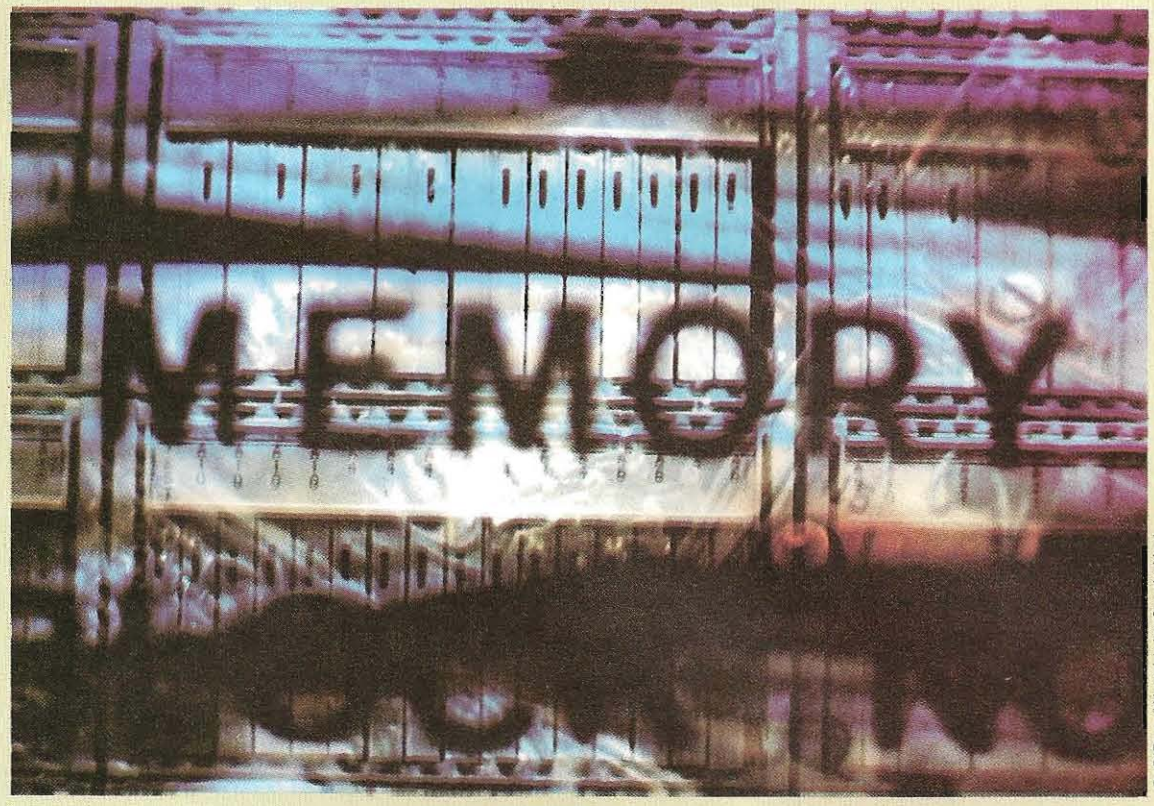
ORGANIGRAMME ET LISTING D'UNE INSTRUCTION ON ... GOTO ...




```
1500 '
1655 ON K GOTO 1660, 1680, 1710, 1740, 1770
1656 PRINT "ERREUR": STOP
1660 ' CARRÉ (K = 1)
.. .....
.. .....
```

Quel avenir pour l'ordinateur ?

L'enthousiasme suscité par le développement des applications de l'informatique au cours de ces dernières années a amené certains spécialistes à imaginer une évolution quasi illimitée de leurs ordinateurs.
Les résultats obtenus aujourd'hui, par rapport aux perspectives et aux ambitions qui orientèrent la recherche il y a quelques années, ont convaincu H.R.J. Grosch qu'il était temps de mettre en garde les « inconditionnels enthousiastes » contre un triomphalisme exagéré.
Le Dr Grosch a été président de l'ACM (Association for Computing Machinery) et a occupé de nombreux postes dans l'administration fédérale et dans quelques grandes compagnies américaines. Aussi, vaut-il la peine de s'arrêter un instant sur ses considérations.
J'ai passé une bonne partie de ma vie à recommander l'usage de l'ordinateur et cela en a valu la peine. Mais malgré mon enthousiasme, je suis resté prudent.
Il y a quelques années, on nous avait promis un langage de programmation indépendant de la machine et commun à tous les gros ordinateurs. On nous avait promis également la traduction du langage machine et l'apparition de systèmes de gestion intégrés. On nous avait dit que les programmeurs disparaîtraient, que les ordinateurs pourraient lire les caractères imprimés et même l'écriture, et qu'ils joueraient aux échecs mieux que l'homme.
On nous avait dit aussi qu'un gros ordinateur aurait les dimensions d'une montre et qu'il comprendrait le langage parlé. Aujourd'hui, on nous dit que le clavier tombera bientôt en désuétude, on répète que les programmeurs ne vont pas tarder à disparaître, que les ordinateurs traiteront des concepts plutôt que des données, qu'un million de processeurs reliés en série résoudront des problèmes d'une complexité inimaginable, et que de nouveaux langages spécialement étudiés permettront à nos enfants d'apprendre plus facilement et plus rapidement que par les méthodes d'enseignement traditionnelles.
Combien de ces promesses ont-elles été tenues ? Et, en ce qui concerne celles que l'on nous fait encore, combien nous faudra-t-il attendre avant que le nouvel hardware si puissant, le software et ses capacités de traitement atteignent nos bureaux, nos usines et nos écoles ?
Le premier fait à prendre en considération est que la réussite des années 1950, 1960, 1970 est arrivée un peu plus lentement qu'on ne l'attendait.
Il est vrai que pour traiter les données commerciales, il y a le Cobol, complètement standardisé et indépendant de l'unité centrale de traitement, mais aujourd'hui, vingt ans après leur apparition, même les programmes Cobol les plus simples demandent encore une certaine mise au point quand on veut les transférer d'IBM à Burroughs ou à ICL, par exemple.
Ce sont les mécanismes d'entrée/sortie et la complexité du logiciel sous-jacent au système de mise en forme qui l'exigent. Les mini-Cobol et les micro-Cobol qui existent ne sont pas du tout standardisés.
Les tentatives de traduction et de contrôle des langages en langue anglaise sont restées sans succès. Il y a trente ans c'était encore un rêve. Il y a vingt ans de nombreuses équipes y travaillaient avec ardeur dans les universités et les industries : j'en ai moi-même dirigé une. Mais le problème s'avéra trop ardu ; les résultats se faisaient attendre.
Même la réalisation d'un « dictionnaire électronique », qui serait d'une telle utilité aujourd'hui pour la vérification de l'orthographe dans les systèmes de traitement de texte (Word Processing), s'est alors révélée trop lente et trop coûteuse. Cependant, même ceux qui étaient alors conscients de ces difficultés ne voulaient pas céder. Le Pentagone finit par ne plus les subventionner et l'initiative en resta là.
Les SIG (Systèmes Intégrés de Gestion) étaient il y a dix ou quinze ans un concept à la mode.
Les chefs d'entreprise n'auraient plus eu qu'à s'asseoir face aux mises en images des données prévisionnelles et des différentes alternatives simulées qui auraient défilé sous leurs yeux à toute allure. Ce projet se révéla utopique avant même qu'on n'en parlât dans les articles de fond du Business Week. L'enthousiasme suscité apporta toutefois crédits et prestige au projet de réalisation des banques de données. Aujourd'hui, c'est elles qui fournissent toutes les informations que nous désirons recevoir.



C. O'Rear/West Light-Grazia Neri

Les possibilités d'emploi de l'ordinateur sont potentiellement illimitées, mais cela dépend bien évidemment des capacités de leurs mémoires elles-mêmes.

Cependant, les relations entre l'ensemble des données financières et les réalisations variaient selon la situation et le moment ; les décisions relatives au choix de l'emplacement d'une usine dépendaient en partie des problèmes familiaux du président, et la recherche de marchés extérieurs dépendait aussi d'événements politiques ou religieux.

Il est vrai que les ordinateurs jouent très bien aux échecs et qu'ils ont largement dépassé le jeu de dames, mais ils jouent comme en ont décidé leurs programmeurs, même s'ils sont plus patients, plus exhaustifs et extrêmement rapides. Ils n'ont pas inventé une nouvelle façon de jouer aux échecs : leurs coups sont encore ceux étudiés par les hommes.

Bien sûr, l'unité centrale du plus grand ordinateur des années 1950 peut tenir sur une puce de la taille d'une écaille de poisson, mais les disques et les dérouleurs de bandes magnétiques occupent toujours une bonne partie des salles des machines.

Parfois, quand le succès est à portée de la main, il ne sert plus à rien. Par exemple, la lecture optique des caractères, qui dépend presque exclusivement du perfectionnement du hardware, est possible. L'ordinateur peut déjà pratiquement lire tout document imprimé ou même bien dactylographié.

Mais tant de données ont été désormais accumulées par les machines que la capacité de les enregistrer sous une forme nouvelle n'a plus guère d'importance.

Dans ma liste de promesses pour le futur, j'ai mentionné que les plus enthousiastes avaient prédit la disparition prochaine des programmeurs ; c'est un euphémisme pour dire que les espoirs des premiers experts en génie logiciel ont été déçus. Il existe aujourd'hui des milliers de programmeurs et leur nombre ne cessera d'augmenter.

Tout ordinateur appliqué à la gestion, qu'il s'agisse d'un mini ou d'un micro, est potentiellement sous la dépendance d'un programmeur qui commet l'imprudence de s'aventurer dans les méandres du logiciel. Il y a un abîme entre la façon de travailler d'un ordinateur et celle des êtres humains. Cet abîme est comblé des milliers de fois chaque jour par des programmeurs capables, ou médiocres,



A. Nogues/Sygma-Grazia Neri

Grâce à l'ordinateur, tous les travaux de bureau sont simplifiés.

et par le personnel chargé du fonctionnement, de la manipulation et de la saisie des données.

Que peut apporter l'ordinateur aujourd'hui ? Et, plus précisément, que peut-il offrir de neuf et de précieux en dehors des domaines de la comptabilité et des techniques de l'ingénieur, où l'on peut déjà lui faire confiance ? La première réaction est de se demander : qu'y a-t-il qui ne fonctionne pas bien dans la comptabilité et dans les techniques ? Dans le monde entier, à New York, comme à Caracas, à Paris, comme dans le Punjab, des applications pratiques et rémunératrices attendent d'être réalisées dans des secteurs que nous connaissons déjà très bien. Elles nécessitent un personnel qui s'occupe de l'analyse des programmes, des experts en logiciel, des équipes chargées du fonctionnement et de la manipulation. Pour améliorer, dans le monde entier, la sélection, la formation et la gestion de ces organismes, il nous faut un hardware plus rapide, plus performant, plus fiable et moins coûteux. Nous l'aurons plus tôt que nous ne l'espérons et avant même d'être en mesure de l'assimiler. Nous pourrions supprimer ce problème, ne serait-ce que pour des raisons de choix.

En effet, les inventeurs, les producteurs, les vendeurs, les experts vous diront « achetez le mien » et il y a des centaines de possibilités et des milliers de combinaisons. En tout cas, il y a une particularité que les vétérans n'ignorent pas et que les victimes d'aujourd'hui devraient connaître : presque tout fonctionne. Vous pourrez, parfois, en être pour vos frais, ou il peut arriver que vous ayez choisi un cabinet de consultants en faillite, mais, en général, l'ordinateur fonctionne vraiment. Son coût représente en outre une faible part – non négligeable, mais dans tous les cas, faible – du montant du devis qu'il faudra établir pour résoudre un problème spécifique. La phase de choix d'un problème bien posé et le fait de pouvoir compter sur un personnel compétent sont des moments autrement plus difficiles et importants que ceux du match Honeywell - IBM.

Aujourd'hui, nous pouvons nous servir d'un instrument d'une importance capitale et dont nous étions démunis il y a seulement dix ans : le microprocesseur.

Qu'il s'agisse d'une machine indépendante, d'un terminal intelligent ou d'un contrôleur de communications, le processeur « mono-chip » peut faire des prodiges.

Quand on l'utilise pour une tâche simple, c'est-à-dire si l'on ne fait pas appel à plus de un ou deux programmes, alors il excelle. Mais il arrive de plus en plus souvent que l'apparition d'un système coïncide avec celle d'autres systèmes, et c'est alors que les problèmes de compatibilité et de standardisation deviennent énormes. A ce niveau, mon point de vue diffère de l'avis général : servez-vous, si vous le pouvez, d'un grand système central, géré de façon hautement professionnelle et qui définit des coûts bien précis. Si vous ne le pouvez pas, conservez le type de gestion, les standards et les façons de programmer propres au centre, même si vous décentralisez le hardware.

Ne laissez chaque fonction et chaque disposition suivre son propre chemin qu'en dernier recours.

Les professeurs et les jeunes entrepreneurs affirment que d'ici à quelques mois – les plus honnêtes disent quelques années – tout cela

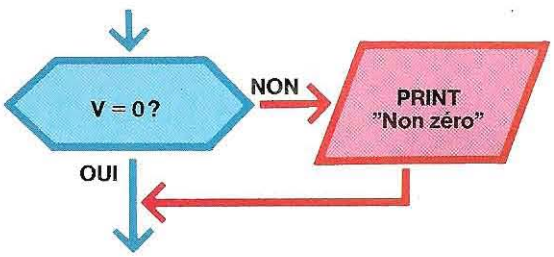
sera balayé par les machines logiques, par la compréhension de la parole et par la programmation en langage naturel. On verra apparaître des architectures nouvelles et fantastiques, des systèmes « amicaux » qui fonctionneront sans commandes ni instructions.
La raison pour laquelle j'ai tellement insisté sur le passé est que, même si nous étions alors peu nombreux, avec une expérience de base limitée, nous avions déjà entendu ces histoires, il y a dix ans. Il est vrai que certaines sont devenues réalité : nous avons le gallium, nous avons les jonctions Josephson, mais tout cela n'a que peu d'importance.
Certains objectifs, tels des interfaces complètes en langage parlé, sont, à la lettre, impossibles à atteindre.
D'autres, tels les réseaux généralisés complètement transparents, les processeurs matriciels pour le graphisme à balayage, même s'ils paraissent très ambitieux, sont abordables, mais il faudra des années avant de songer à une possible application.
D'autres objectifs encore n'auront plus tellement d'importance : le travail avec des langages comme le LOGO ou le PROLOG est fascinant, mais pour la multiplication, rien d'autre qu'une banale instruction n'est prévu. D'autre part, peu importe le caractère « amical » d'un système : il sera toujours plus facile de le connaître avec un bon manuel d'instructions.
La clé pour comprendre les possibilités offertes par l'ordinateur était généralement l'ordinateur lui-même, peu diffusé, faillible, et coûteux. Les gens le regardaient avec admiration et se battaient pour l'approcher.
Aujourd'hui, la clé des possibilités de l'ordinateur, c'est l'homme. Partout, il y a des machines et même si les experts sont très nombreux, on a surtout besoin d'analystes et de programmeurs. Le pays le plus petit et le moins avancé parmi les pays en voie de développement a besoin de centaines de ces personnes ; les pays plus développés en demandent des milliers ou des dizaines de milliers. Les pays hautement industrialisés comme le Japon ou les Etats-Unis pourront en avoir besoin par millions d'ici à la fin de cette décennie. Les universités et les collèges ne pourront en aucune façon en préparer un tel nombre. En ce moment, il ne semble pas que les écoles secondaires puissent y parvenir ni même que les universités puissent former suffisamment de professeurs pour que les écoles puissent y arriver. Cela est encore plus évident dans des pays comme Singapour où la volonté, l'argent et les facultés d'adaptation sociale sont insuffisants devant le travail qui reste à accomplir.
La solution applicable immédiatement est d'utiliser l'ordinateur pour préparer et perfectionner les analystes, les programmeurs, les ingénieurs et les opérateurs chargés de la manipulation. Nous avons à notre disposition le matériel, les logiciels et les systèmes didactiques dont la traduction dans d'autres langues que l'anglais est à l'étude. Moi-même, j'ai fréquenté des cours d'introduction au système Plato et j'ai pu noter comment, à l'aide d'un cours avancé, un expert pouvait se rendre maître rapidement d'un système difficile à acquérir. Au niveau scolaire, j'ai vu à l'œuvre ce système Plato, centralisé dans une grande université américaine, près de Capetown.
Les possibilités du microprocesseur sont par ailleurs d'une aide précieuse. Là où les coûts, l'énergie et les exigences du milieu excluent l'utilisation d'une grosse unité centrale de traitement, on peut actuellement installer des machines de bureau dotées d'un système didactique capable d'enseigner le Basic et le Fortran, la lecture et l'orthographe, l'arithmétique et la grammaire, l'algèbre et la géométrie, la comptabilité et l'ingénierie.
Ces machines n'ont pas encore accès aux énormes banques de données dont pourrait bénéficier l'enseignement en général (et plus particulièrement universitaire), et l'on se heurte également à certains obstacles culturels. Malgré tout, les progrès réalisés sur le hardware résoudront certainement le premier aspect de ce problème et je veux espérer qu'au fur et à mesure que les techniques informatiques et les types de problèmes traités par l'ordinateur se multiplieront, au moins certaines tensions d'ordre culturel s'atténueront.
Pour conclure, on peut dire que les possibilités de l'ordinateur sont aujourd'hui énormes. Il n'est plus nécessaire d'attendre les processeurs d'informations ou les jonctions Josephson. Les coûts sont désormais accessibles et ils continueront à baisser. Mais il reste les personnes – les personnes adéquates – et c'est l'ordinateur qui nous aide à les former.

(Extrait de « Le pouvoir de l'ordinateur de nos jours » de H.R.J. Grosch, Agora 1983).

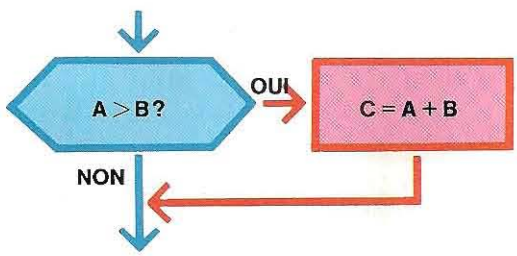
peut s'écrire sous condition (IF ... GOTO) sans le terme THEN.
Ci-dessous, sont reproduits les organigrammes des trois instructions précédentes.
Le dernier organigramme prévoit (ligne 10) la vérification de deux conditions simultanées : A = B et (AND) C > D ; dans un cas, le calcul à exécuter est F = 3 * B, dans l'autre, F = 5 * B. En ligne 10, il faut prévoir un branchement permettant de sauter l'instruction 20, faute de quoi on exécuterait, dans les deux cas, le calcul F = 5 * B. En effet, en l'absence de GOTO 126, et si les deux conditions de la ligne 10 sont vraies, on exécute F = 3 * B, mais, aussitôt après, l'instruction F = 5 * B écrase le résultat du calcul précédent. Afin d'éviter cela, il faut prévoir le saut conditionnel de la ligne 20 (si le test IF ... a donné un résultat positif), isolant ainsi le calcul F = 5 * B.

Les instructions conditionnelles sont souvent utilisées pour accélérer le déroulement des boucles. Une boucle peut être employée pour chercher une valeur déterminée dans un ensemble de données. La démarche la plus logique à adopter en ce cas consiste à programmer une boucle réalisant l'extraction successive de toutes les données présentes (en partant de la première) et leur comparaison avec la valeur désirée ; si la comparaison se vérifie, la donnée est trouvée et la boucle s'interrompt. Un indicateur initialisé à 0, prendra la valeur 1, dès que la condition de recherche sera vérifiée ; le test IF s'effectuera sur la valeur de cet indicateur, permettant ainsi de quitter la boucle.
A la page 407, on voit un exemple de recherche dans un fichier du personnel. Le fichier contient le nom et le prénom, le matri-

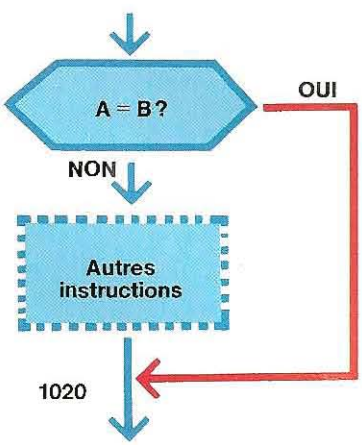
ORGANIGRAMMES DE CERTAINES INSTRUCTIONS CONDITIONNELLES



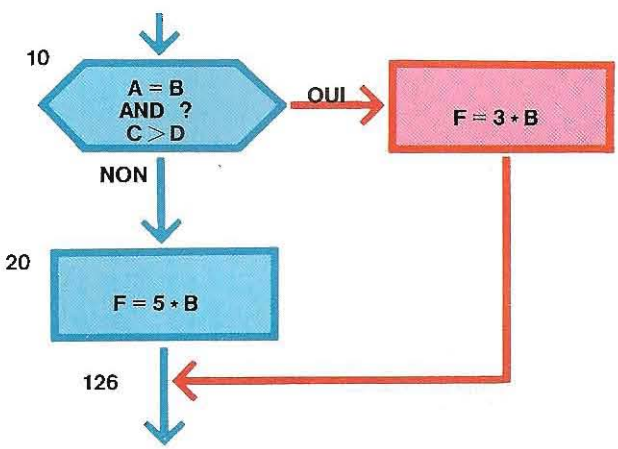
IF V < > 0 THEN PRINT "Non zéro"
(Le symbole < > signifie différent de...)



IF A > B THEN C = A + B



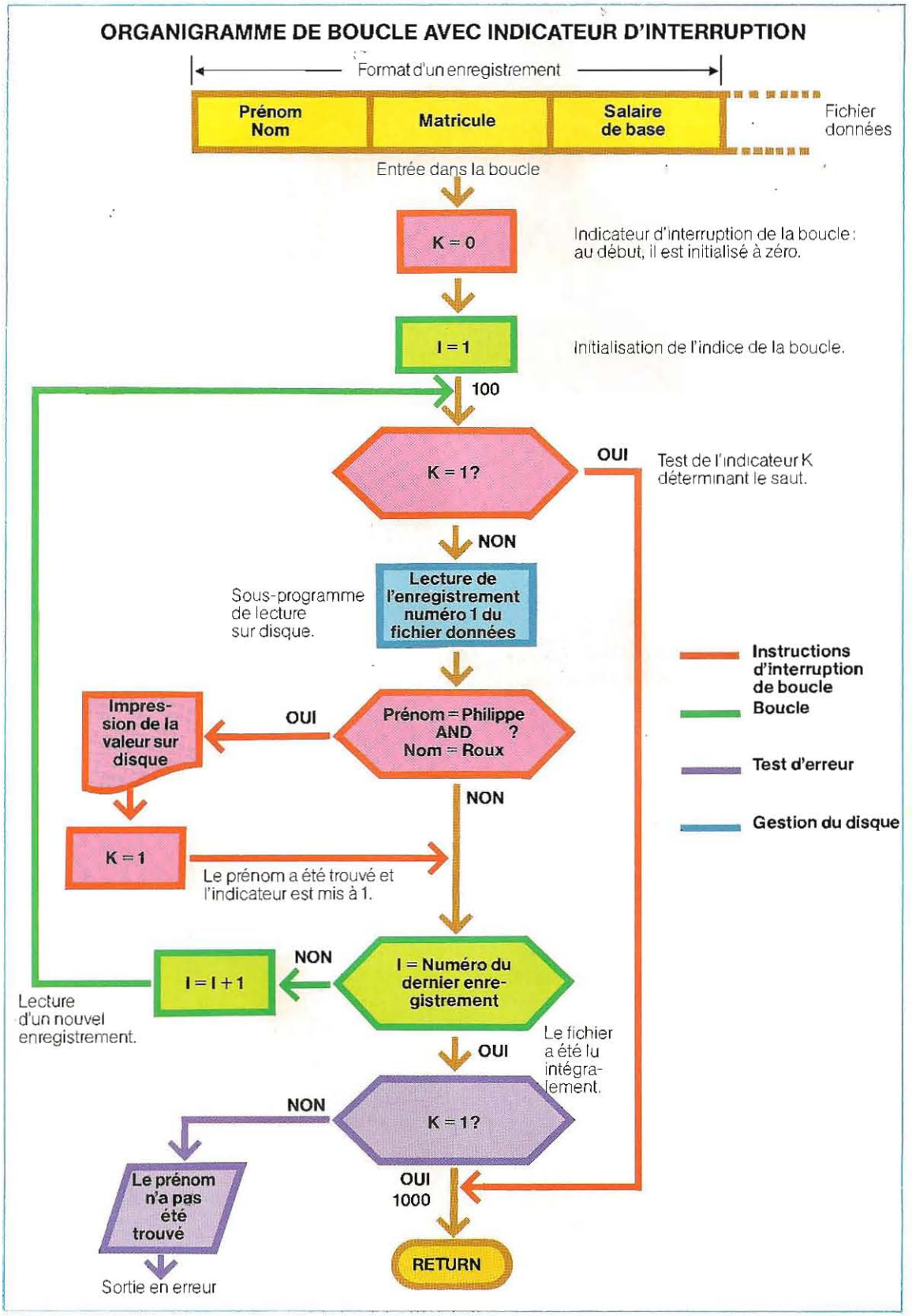
IF A = B THEN 1020
ou aussi
IF A = B GOTO 1020



10 IF A = B AND C > D THEN F = 3 * B: GOTO 126
20 F = 5 * B ' On est en 20 : la condition était fausse

126 ' On arrive de 10 : elle était vraie

ORGANIGRAMME DE BOUCLE AVEC INDICATEUR D'INTERRUPTION



cule et le salaire de base de chaque employé, et on veut connaître le salaire de Monsieur Philippe Roux. Au début, l'indicateur (K dans l'exemple) est mis à 0, cela signifiant que la recherche n'est pas terminée.
A l'entrée de la boucle (instruction 100) un contrôle testant la valeur de K est effectué : si la valeur de K est 1, on quitte la boucle ; sinon, la lecture d'un nouvel enregistrement s'effectue, ainsi que le contrôle concernant le nom lu. En cas de résultat positif, c'est-à-dire quand il y a égalité entre le nom lu et celui qu'on cherche, la donnée voulue (salaire) est imprimée et l'indicateur K prend la valeur 1.
Ainsi, lors de l'itération suivante de la boucle, l'instruction 100 s'exécute avec K = 1 et provoque un branchement vers la fin du programme (instruction 1000).
En d'autres termes, la valeur K = 1 indique que le résultat de la recherche a été positif au passage précédent et l'exécution du programme peut se terminer. Les premiers exemples du schéma de la page 406 reproduisent les formes d'instructions conditionnelles les plus simples : test sur une valeur et décision qui en découle d'effectuer ou non l'instruction. En général, une instruction conditionnelle (IF...) est double et possède deux issues possibles : la première si le contrôle est vérifié, la seconde si le test donne des résultats négatifs. Dans nos trois exemples simplifiés, la première voie se résume à une instruction, la seconde étant absente.
A l'opposé, le dernier exemple du même schéma propose les deux chemins. Il faut recourir à une instruction de saut (GOTO 126 à la fin de la ligne 10) pour ne pas exécuter l'instruction de la ligne 20, en cas de test positif sur la 10.
Cependant, cette structure n'est pas optimale. Il existe une autre instruction, qui sera préférée dans des cas analogues : il s'agit de l'instruction ELSE. Elle indique la marche à suivre dans le cas où le test de la condition n'a pas été positif.
En d'autres termes, elle restitue les deux suites possibles à cette vérification, correspondant respectivement à test = vrai (THEN), et à test = faux (ELSE).
La structure à adopter est la suivante :

IF condition à vérifier

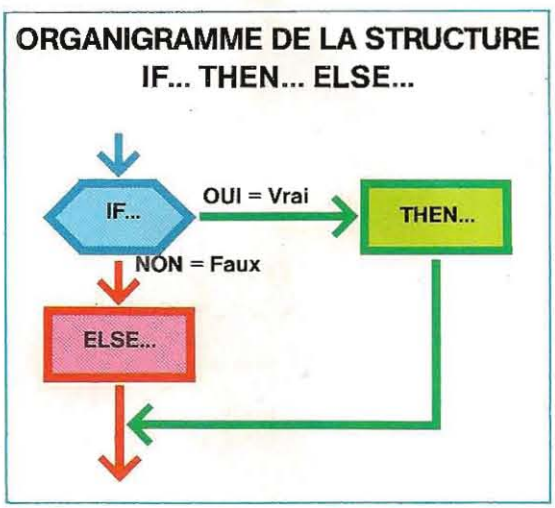
THEN instructions à exécuter si la condition est vraie

ELSE instructions à exécuter si la condition est fausse

Avec cette nouvelle instruction, la codification du dernier organigramme de la page 406 devient :

```
10 IF A = B AND C > D THEN F = 3 * B
15 'A exécuter si la condition est vérifiée.

20 ELSE F = 5 * B
25 'A exécuter dans le cas contraire.
```

Les deux instructions peuvent être symbolisées par le schéma ci-dessous.



Les instructions IF... THEN... ELSE... peuvent être alignées l'une à la suite de l'autre sur la même ligne sans être séparées par le symbole :, car elles constituent un bloc fonctionnel unique :

```
IF A > B THEN C = 1 ELSE C = 0
```

est une instruction valide, de même que

```
IF A > B THEN C = 1
        ELSE IF A = B THEN C = 0
                      ELSE C = 1
```

(qui peut aussi être écrite sur une seule ligne) analyse toutes les situations possibles pouvant se présenter dans une comparaison entre les variables A et B.

De manière plus explicite, cette dernière instruction* s'écrira comme suit :

```
IF A > B THEN C = 1
SI A supérieur à B ALORS C = 1
ELSE IF A = B THEN C = 0
SINON SI A = B ALORS C = 0
      ELSE C = -1
      SINON C = -1
```

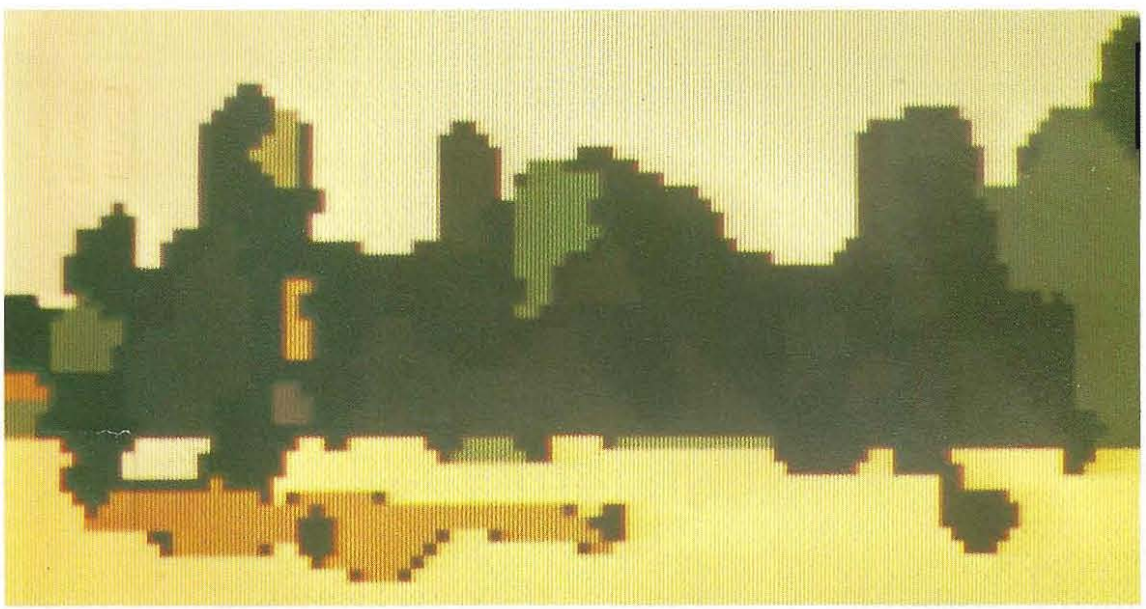
* Comme on le voit, les mots clés en langue anglaise peuvent être traduits de la manière suivante :
IF = SI THEN = ALORS ELSE = SINON

Utilisation de variables réelles dans les instructions conditionnelles

La représentation standard d'une variable réelle (c'est-à-dire qui possède une partie décimale) est la virgule flottante (floating point), suivie d'un nombre de chiffres significatifs qui dépend de la machine utilisée. Lorsqu'on affecte à la variable le résultat d'un calcul, celui-ci peut être arrondi ; l'imprécision qui en découle, si minime soit-elle, peut néanmoins fausser l'issue du test.

Pour vérifier que le résultat d'une opération est zéro, on nommera M la variable contenant ce

Deux vues de l'île de Manhattan (New York) réalisées par ordinateur en synthèse numérique.



Pierce/Wheeler Pictures-Grazia Neri



Pierce/Wheeler Pictures-Grazia Neri

Test 12

1 / Après les instructions DEFINT I-N et DEFDBL Z, quelles affectations parmi les suivantes, sont erronées?
a) IND = 3.14 b) R = 3.5 c) Z = P + 7.5 d) CIRC = I * ZA

2 / Quelles sont les valeurs affectées aux variables A, B, C, D, E, F, G par les instructions:

```
10 READ A, B, C
20 RESTORE
30 READ D
40 RESTORE 80
50 READ E, F, G
60 DATA 5, 7, 21, 40, 60
70 DATA 3, 9, 10
80 DATA 15, 20, 75
```

3 / Le programme reproduit ci-dessous contient une erreur: laquelle?

```
10 X = A + B
20 READ C, D
30 RESTORE
40 READ E, F, G
50 DATA 3, 7
```

4 / Comment modifier le programme de la question numéro 2 pour affecter aux variables E, F et G les valeurs 7, 21 et 40 (DATA 60)?

5 / Le programme listé ci-dessous contient une erreur: quelle ligne faut-il supprimer?

```
10 DEFINT I-N
20 DEFDBL Z
30 READ A, B, C
40 X = C * (A + B)
50 RESTORE
60 READ A$
70 PRINT X
80 DATA 5, 3, 2
90 DATA "Ceci est un essai"
```

Les solutions du test se trouvent page 413.

résultat, et l'instruction s'écrira :

```
IF M = 0 THEN ...
```

Sous cette forme (exacte si M était un entier), un manque de précision sur M peut engendrer des erreurs. Si, par exemple, M vaut 0.001, on assimilera, pour des raisons pratiques, cette valeur à 0, alors que la machine répondra au test IF M = 0 par un résultat négatif.

Dans une instruction conditionnelle à paramètre réel, on indiquera toujours, pour éviter ce genre d'erreur, la précision avec laquelle on veut que le contrôle soit exécuté. Dans l'exemple précédent, en écrivant :

```
IF M < 0.001 THEN ...
```

la condition est vérifiée pour toutes les valeurs de M inférieures à 0.001, ce nombre étant la précision avec laquelle le test est réalisé.

Mais l'instruction est encore inexacte: la variable M pourrait prendre des valeurs néga-

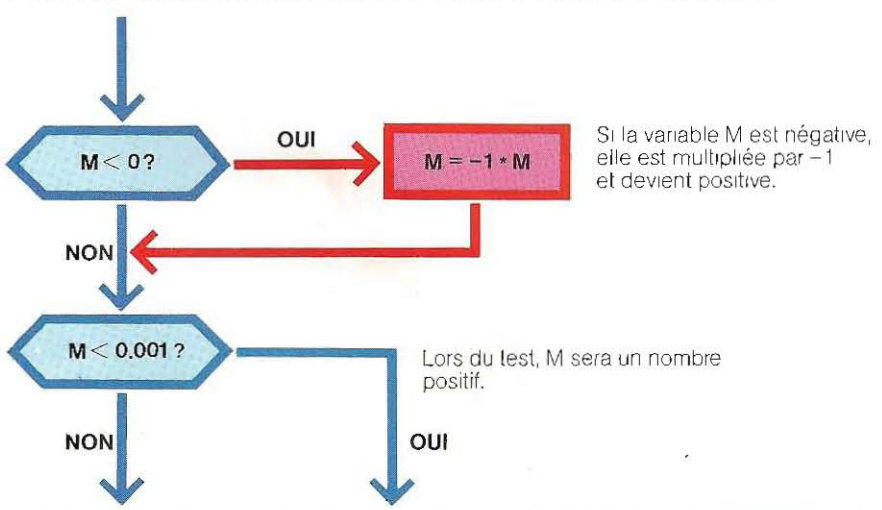
tives très éloignées de zéro, mais toujours inférieures à 0.001 (−10000 est inférieur à 0.001) et le test donnerait des résultats erronés. Pour formuler correctement cette instruction, il faut considérer la « valeur absolue » de M, c'est-à-dire sa valeur privée de signe. De cette manière (en enlevant le signe −), une éventuelle valeur négative deviendra positive et le test sera cette fois parfaitement satisfaisant.
Il existe une instruction spéciale (fonction) qui permet d'extraire la valeur absolue d'une variable : nous l'étudierons prochainement. Pour l'instant, on adoptera la méthode illustrée par le schéma au bas de cette page, et qui consiste à toujours comparer un nombre positif (M ou −M) à la valeur critique (ici 0.001).

Appel d'un sous-programme et enchaînement de programmes

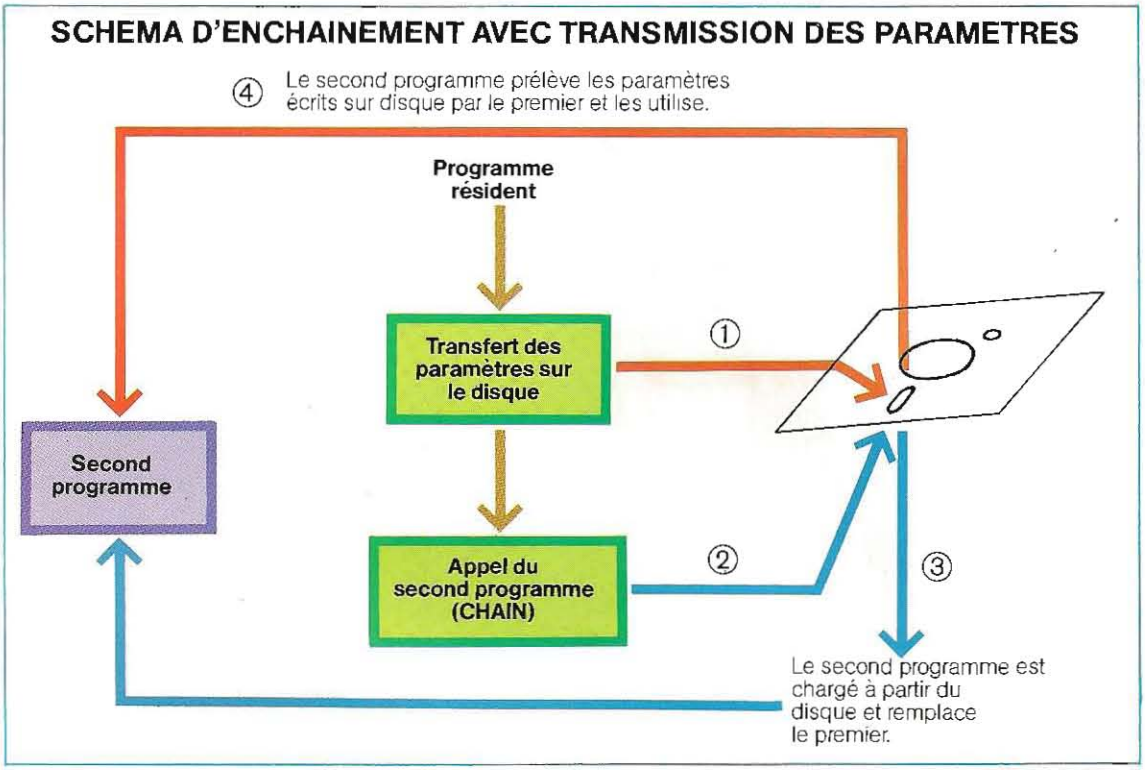
L'instruction « d'appel » d'un sous-programme est GOSUB n, n étant le numéro de la ligne à partir de laquelle débute le sous-programme (**sub**routine en anglais). Cette instruction peut être conditionnelle. Par exemple :

GOSUB 1250
Dans ce cas, le sous-programme 1250 est appelé sans aucune condition, mais, si on a :
IF A > B THEN GOSUB 1250
l'exécution saute à 1250 seulement si A est supérieur à B. Si on a :
ON M GOSUB 1250, 720, 500
l'instruction donne le contrôle à un des trois sous-programmes 1250, 720 ou 500, selon la valeur de M (voir instructions ON ... GOTO ...).
L'instruction GOSUB ... ressemble à l'instruction GOTO ..., mais ici le point d'arrivée du saut est le début d'un sous-programme, au terme duquel l'instruction RETURN ramène l'exécution au programme appelant, à la ligne suivant le GOSUB.
Les sous-programmes (appelés aussi routines) faisant logiquement partie du programme appelant, seront tous chargés en mémoire avant l'exécution. Leur utilisation, bien que recommandée pour une meilleure structuration des programmes, ne permet pas de gain en espace. Il ne faut pas confondre cette subdivision logique en routines avec le découpage physique d'un programme trop volumineux pour être intégralement contenu dans la mémoire de l'ordinateur. Dans ce dernier cas, on divisera le programme en parties plus petites, résidant sur disque, et qui seront chargées en mémoire seulement au moment de leur utilisation (voir **les systèmes d'exploitation**, page 291).
Chacun de ces modules constitue un programme à part entière ; au moment de son chargement, il remplacera en mémoire le programme précédent.
L'instruction réalisant le chargement et l'exécution d'un programme à la suite d'un autre est CHAIN « NOM », où NOM désigne le fichier dans lequel est mémorisé (sur disque) le programme à charger. Par exemple :

TEST DE LA VALEUR ABSOLUE D'UNE VARIABLE REELLE



SCHEMA D'ENCHAINEMENT AVEC TRANSMISSION DES PARAMETRES



CHAIN « ESSAI »
charge le programme ESSAI.
CHAIN « B : XY »
charge le programme XY résidant sur disque B.

L'instruction CHAIN peut aussi indiquer le numéro de la ligne à partir de laquelle doit commencer l'exécution du programme chargé. CHAIN « ESSAI », 275 charge le programme ESSAI et l'exécute à partir de l'instruction 275. Si le numéro est omis (exemples précédents), l'exécution commence à partir de la première instruction du nouveau programme.
Les programmes successivement appelés et exécutés peuvent posséder certaines variables en commun. Il faut alors transmettre (« passer ») ces paramètres de l'un à l'autre. Le transfert se réalise de trois façons différentes :

- avec un fichier intermédiaire comme support des valeurs,
- par l'option ALL,
- par l'instruction COMMON.

Dans le premier cas, le programme résident écrit dans un fichier spécial du disque tous les paramètres qui doivent être transmis au programme suivant. Une fois chargé, ce dernier lit les valeurs du fichier, afin de les utiliser.
Le schéma ci-dessus décrit les opérations que nécessite cette technique. La méthode est laborieuse, mais reste parfois la seule possible, car la majeure partie des petits systèmes d'exploitation ne proposent pas d'autre solution.
La deuxième méthode consiste à spécifier l'option ALL dans l'instruction CHAIN.
Elle permet de transmettre toutes les variables d'un programme au suivant. Par exemple,

CHAIN « NOM », 152, ALL

signifie : charger le programme NOM et l'exécuter à partir de l'instruction 152, en conservant toutes les valeurs des variables. Tous les compilateurs ne prévoient pas cette option. En général, elle est possible en Basic interprété et non en compilé.
L'emploi d'une instruction COMMON est la solution la plus classique et aussi la plus souple pour transmettre des paramètres. Elle peut être utilisée aussi bien en Basic interprété qu'en compilé, mais elle n'est disponible que sur les systèmes d'exploitation relativement évolués.

Le code COMMON déclare « communes » toutes les variables énumérées; par exemple, COMMON A, X, N définit les variables A, X et N comme paramètres communs. Une zone mémoire spéciale, qui ne sera pas écrasée par le chargement d'autres programmes, est réservée aux paramètres du COMMON.
L'instruction CHAIN possède deux autres options : MERGE et DELETE.
MERGE permet de charger le nouveau programme en « overlay », (recouvrement), c'est-à-dire comme s'il s'agissait d'un morceau (segment) du programme appelant. Ainsi, toutes les déclarations des types de variables (par exemple DEFINT ..., etc.) qui, autrement, devraient être répétées dans le programme appelé, demeureront valables. Avec l'option MERGE, les segments appelés doivent être en format ASCII.
L'option DELETE permet d'effacer le segment chargé en overlay avant d'en appeler un autre. Toutefois, de telles options ne sont généralement pas prévues dans les compilateurs.

En résumé, la formulation complète d'une instruction d'enchaînement est :

CHAIN MERGE « NOM », 1350, ALL, DELETE 185 – 2730

qui prendra la signification suivante :
charger en mode overlay le programme NOM, puis déclencher l'exécution à la ligne 1350 avec transfert de toutes les variables ; effacer ensuite les instructions de 185 à 2730.
Les sous-programmes externes, n'appartenant pas au programme principal, peuvent être écrits dans les langages différents du Basic (Fortran, Cobol, Assembleur) ; en ce cas, des instructions spécifiques assureront la transmission des paramètres et l'appel des segments. Ces instructions spéciales seront étudiées plus en détail dans le chapitre traitant des techniques de programmation en langages mixtes.

Exemple d'application : gazole ou essence ?
Lors de l'achat d'une voiture, le choix du type de moteur (à gazole ou à essence), est décisif, et dépend de nombreux facteurs, tels :

Solutions du test 12

1 / Les affectations erronées sont : a) car une variable qui commence par la lettre I est définie entière (DEFINT I-N) ; d) qui contient la variable CIRC en simple précision alors que ZA est définie implicitement en double précision (DEFDBL Z).
Une troncature (un arrondi) s'effectuera dans les deux cas.

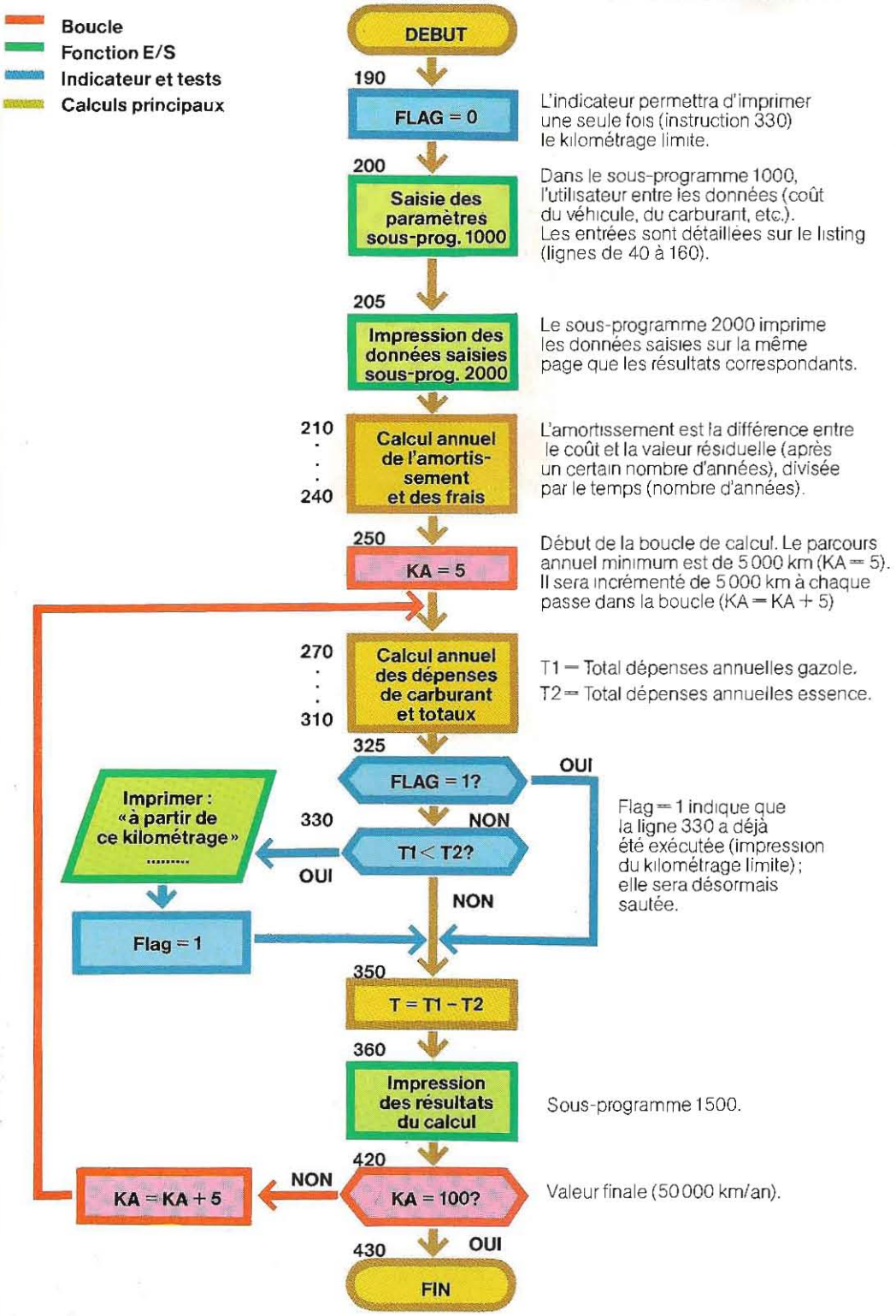
2 / L'instruction 10 exécute les affectations A = 5, B = 7, C = 21, et positionne le pointeur de DATA sur la quatrième valeur de la ligne 60. L'instruction 20 remet à zéro ce pointeur et le repositionne en début de DATA ; la valeur 5 est à nouveau affectée à la variable D (ligne 30). L'instruction 40 positionne le pointeur en début de DATA, apparaissant à la ligne 80 ; la lecture suivante (ligne 50) donnera E = 15, F = 20, G = 75.

3 / Après le RESTORE (ligne 30), le READ de la ligne 40 cherche à lire trois variables dans un DATA contenant seulement deux valeurs.

4 / Si on supprime la ligne 40, le pointeur de DATA reste sur la deuxième valeur de la ligne 60 (la première valeur a été affectée à la variable D en ligne 30). Le READ suivant (ligne 50) exécutera les affectations E = 7, F = 21, G = 40.

5 / Le RESTORE (instruction 50) ramène le pointeur en début de DATA numérique (ligne 80) ; la ligne 60 provoquera une erreur car la variable (chaîne) et les données (numériques) ne sont pas homogènes. Si l'on supprime la ligne 50, le READ de la ligne 30 positionnera le pointeur au début du DATA de chaîne (ligne 90), et la variable A$ (instruction 60) pourra être lue.

ORGANIGRAMME DU PROGRAMME DE BILAN ECONOMIQUE ENTRE UNE VOITURE A GAZOLE ET A ESSENCE



VOITURE A GAZOLE ET VOITURE A ESSENCE

```
5 ' *** COMPARAISON ENTRE VOITURES GAZOLE/ESSENCE ***
6 '
7 '   FILE - GAES
8 '
10 OPTION BASE 1   ' Les indices partent de 1
20 '
30 '
32 A$ = "£££££££££.£"
35 B$ = "£££££££££"
36 C$ = "£££££££££.££"
40 ' VALEUR DES VARIABLES
50 '   CG = Prix d'achat voiture Gazole
60 '   CE = Prix d'achat voiture Essence
70 '   KG = Kilometres parcourus avec 1 litre de Gazole
80 '   KE = Kilometres parcourus avec 1 litre d'Essence
90 '   LG = Prix du litre de gazole
100 '   LE = Prix du litre d'essence
110 '   TG = Frais divers v. gazole (ex. vignette)
120 '   TE = Frais divers v. essence (ex. vignette)
130 '   AA = Nombre d'annees d'utilisation
140 '   KA = Kilometrage annuel
150 '   VG = Valeur de revente voiture a moteur diesel
160 '   VE = Valeur de revente voiture a moteur essence
170 '
180 '
190 FLAG = 0
200 GOSUB 1000   ' SUBROUTINE ENTREE DATA
205 GOSUB 2000   ' SUBROUTINE D'EDITION
210 C1 = (CG-VG)/AA   ' Cout amortissement voiture diesel
220 C2 = (CE-VE)/AA   ' Cout amortissement voiture essence
230 S1 = C1 + TG        ' Cout amortissement voiture diesel + frais
240 S2 = C2 + TE        ' Cout amortissement voiture essence + frais
250 FOR KA = 5 TO 100 STEP 5
260 '
270 R1 = LG*KA*1000/KG   ' Depense annuelle de gazole
280 R2 = LE*KA*1000/KE   ' Depense annuelle d'essence
290 '
300 T1 = S1 + R1           ' Depense globale voiture diesel
310 T2 = S2 + R2           ' Depense globale voiture essence
320 '
325 IF FLAG = 1 GOTO 350
330 IF T1 < T2 THEN LPRINT   " A partir de ce kilometrage annuel "
332 IF T1 < T2 THEN LPRINT   " la voiture diesel est rentable " : FLAG = 1
340 '
350 T = T1 - T2
360 GOSUB 1500    ' Subroutine pour l'impression des resultats
390 '
400 PRINT " taper un caractere pour continuer "
410 S$ = INPUT$(1)
411 '
420 NEXT KA
430 END
1000 INPUT " COUT AUTO DIESEL " ; CG
1010 INPUT " COUT AUTO ESSENCE " ; CE
1020 INPUT " KM PAR LITRE DE GAZOLE " ; KG
1030 INPUT " KM PAR LITRE D'ESSENCE " ; KE
1040 INPUT " COUT LITRE GAZOLE " ; LG
1050 INPUT " COUT LITRE ESSENCE " ; LE
1060 INPUT " VIGNETTE AUTO. DIESEL " ; TG
1070 INPUT " VIGNETTE AUTO. ESSENCE " ; TE
1080 INPUT " NOMBRE D'ANNEES D'UTILISATION " ; AA
1090 '
1100 INPUT " VALEUR REVENTE AUTO DIESEL " ; VG
1110 INPUT " VALEUR REVENTE AUTO ESSENCE " ; VE
1120 '

1130 RETURN
1500 '  ***** IMPRESSION DES RESULTATS *****
1510 LPRINT " KM = " ; KA*1000
1520 LPRINT " DEPENSE ANNUELLE AUTO GAZOLE = " ; : LPRINT USING B$ ; T1
1530 LPRINT " DEPENSE ANNUELLE AUTO ESSENCE = " ; : LPRINT USING B$ ; T2
1540 LPRINT " DIFFERENCE DES DEPENSES       = " ; : LPRINT USING B$ ; T
1550 LPRINT
1560 LPRINT
1570 RETURN
2000 '
2010 '  ***** IMPRESSION DES VALEURS SAISIES *****
2020 '
2030 N$ = " ***** CALCUL ET COMPARAISON DIESEL ESSENCE ***** "
2040 D$ = "                                    TYPE DE VEHICULE "
2050 E$ = " VALEURS                          DIESEL       ESSENCE "
2060 F$ = " COUT ACQUISITION "
2070 G$ = " KM PARCOURUS AVEC 1 LITRE "
2080 H$ = " COUT DU LITRE "
2090 J$ = " FRAIS (VIGNETTE) "
2100 K$ = " NOMBRE D'ANNEES D'UTILISATION "
```

```
2120 M$ = " VALEUR DE REVENTE "
2130 LPRINT N$ : LPRINT
2140 LPRINT D$ : LPRINT
2150 LPRINT E$ : LPRINT
2160 LPRINT F$; : LPRINT TAB(44) : LPRINT USING C$ ; CG ; CE : LPRINT
2170 LPRINT G$; : LPRINT TAB(44) : LPRINT USING A$ ; KG ; KE : LPRINT
2180 LPRINT H$; : LPRINT TAB(44) : LPRINT USING C$ ; LG ; LE : LPRINT
2190 LPRINT J$; : LPRINT TAB(44) : LPRINT USING C$ ; TG ; TE : LPRINT
2200 LPRINT K$; : LPRINT TAB(44) : LPRINT USING B$ ; AA ; AA : LPRINT
2220 LPRINT M$; : LPRINT TAB(44) : LPRINT USING C$ ; VG ; VE : LPRINT : LPRINT : LPRINT
2230 RETURN
```

```
***** CALCUL ET COMPARAISON DIESEL ESSENCE *****

                                        TYPE DE VOITURE

VALEURS                                 DIESEL      ESSENCE

COUT D'ACQUISITION                     90000.00     60000.00

KM PARCOURUS AVEC 1 LITRE                 22.7         13.4

COUT DU LITRE                             3.87         4.94

FRAIS (VIGNETTE)                        330.00       225.00

NOMBRE D'ANNEES D'UTILISATION             5            5

VALEUR DE REVENTE                      30000.00     15000.00


KM = 5000
DEPENSE ANNUELLE AUTO GAZOLE   =     13182
DEPENSE ANNUELLE AUTO ESSENCE  =     11068
DIFFERENCE DES DEPENSES        =      2114


KM = 10000
DEPENSE ANNUELLE AUTO GAZOLE   =     14035
DEPENSE ANNUELLE AUTO ESSENCE  =     12912
DIFFERENCE DES DEPENSES        =      1123


KM = 15000
DEPENSE ANNUELLE AUTO GAZOLE   =     14887
DEPENSE ANNUELLE AUTO ESSENCE  =     14755
DIFFERENCE DES DEPENSES        =       132

A partir de ce kilometrage annuel, la voiture diesel est rentable
KM = 20000
DEPENSE ANNUELLE AUTO GAZOLE   =     15740
DEPENSE ANNUELLE AUTO ESSENCE  =     16598
DIFFERENCE DES DEPENSES        =      -858


KM = 25000
DEPENSE ANNUELLE AUTO GAZOLE   =     16592
DEPENSE ANNUELLE AUTO ESSENCE  =     18441
DIFFERENCE DES DEPENSES        =     -1849
```

- le prix respectif des deux types de voiture ;
- le coût de leur vignette et de leurs frais fixes respectifs ;
- les dépenses en carburant ;
- la cotation à l'argus de chacun des deux types de voiture.

Le programme de bilan économique pour les deux voitures devra tenir compte de tous les facteurs cités, en considérant aussi la marque et le modèle choisis.
L'organigramme de la page 414 fournit, en fonction de la valeur des paramètres caractérisant la dépense (prix, frais fixes, consommation), le coût total annuel pour différents parcours kilométriques, et met en évidence le kilométrage minimum à partir duquel le moteur diesel est le plus avantageux.
Le programme correspondant est listé pages 415 et 416. Un exemple de calcul l'accompagne, dans l'hypothèse de deux voitures coûtant respectivement 90 000 et 60 000 francs à l'achat, et pouvant être revendues 30 000 et 15 000 francs après cinq ans. Comme on peut le voir, l'intérêt économique se situe en faveur du véhicule diesel et pour un parcours annuel minimum de 15 000 kilomètres.
En modifiant les conditions initiales lors de la saisie des données, on obtiendra une nouvelle estimation, par exemple dans le cas d'un changement de cylindrée. La décision d'achat tiendra compte du meilleur rapport qualité-prix.

Les diagnostics d'erreurs

Lorsqu'on écrit un programme, il est inévitable, au moins dans la première phase de rédaction, de commettre des erreurs. Le travail de mise au point présente des difficultés qui augmentent avec le nombre des instructions et leur complexité. Pour cette raison, on cherchera à structurer les programmes en les subdivisant en sous-programmes, chacun pouvant alors être exécuté séparément.
Les techniques de test des programmes seront traitées par la suite. Ici, nous présenterons seulement les instructions particulières de recherche des erreurs.
On distinguera deux catégories principales :

- les erreurs relevées en phase d'écriture du programme ;
- les erreurs relevées en phase d'exécution du programme.

Les erreurs du premier type, appelées aussi erreurs de syntaxe, proviennent d'une formulation inexacte des instructions, par exemple d'une faute de frappe ou du non-respect des espaces entre deux mots. Elles sont reconnues immédiatement et signalées par l'interpréteur Basic. Leur correction est facile : il suffit de récrire l'instruction sous sa forme exacte. Il faut toutefois se rappeler que ce premier diagnostic nous assure de l'exactitude formelle des instructions, mais pas du bon fonctionnement du programme en phase d'exécution.
Par exemple, lors de la programmation d'une boucle, on peut oublier son instruction de fermeture (NEXT). A ce niveau de diagnostic, l'instruction initiale (FOR ... TO ...) est formellement exacte, et l'interpréteur ne signale aucune erreur. L'omission du NEXT ne sera mise en évidence que par la suite, lors de l'exécution du programme.
Les erreurs du second type sont plus complexes et n'apparaissent qu'à l'exécution du programme. Les méthodes qui s'appliquent à leur recherche et à leur correction dépendent principalement de ce qui les a causées.
Les principales erreurs relevées en phase d'exécution peuvent être :

- des erreurs de programmation ;
- des erreurs de logique ;
- des erreurs dans le déroulement des calculs ;
- des erreurs système ;
- des erreurs commises par l'utilisateur lors de la saisie des données.

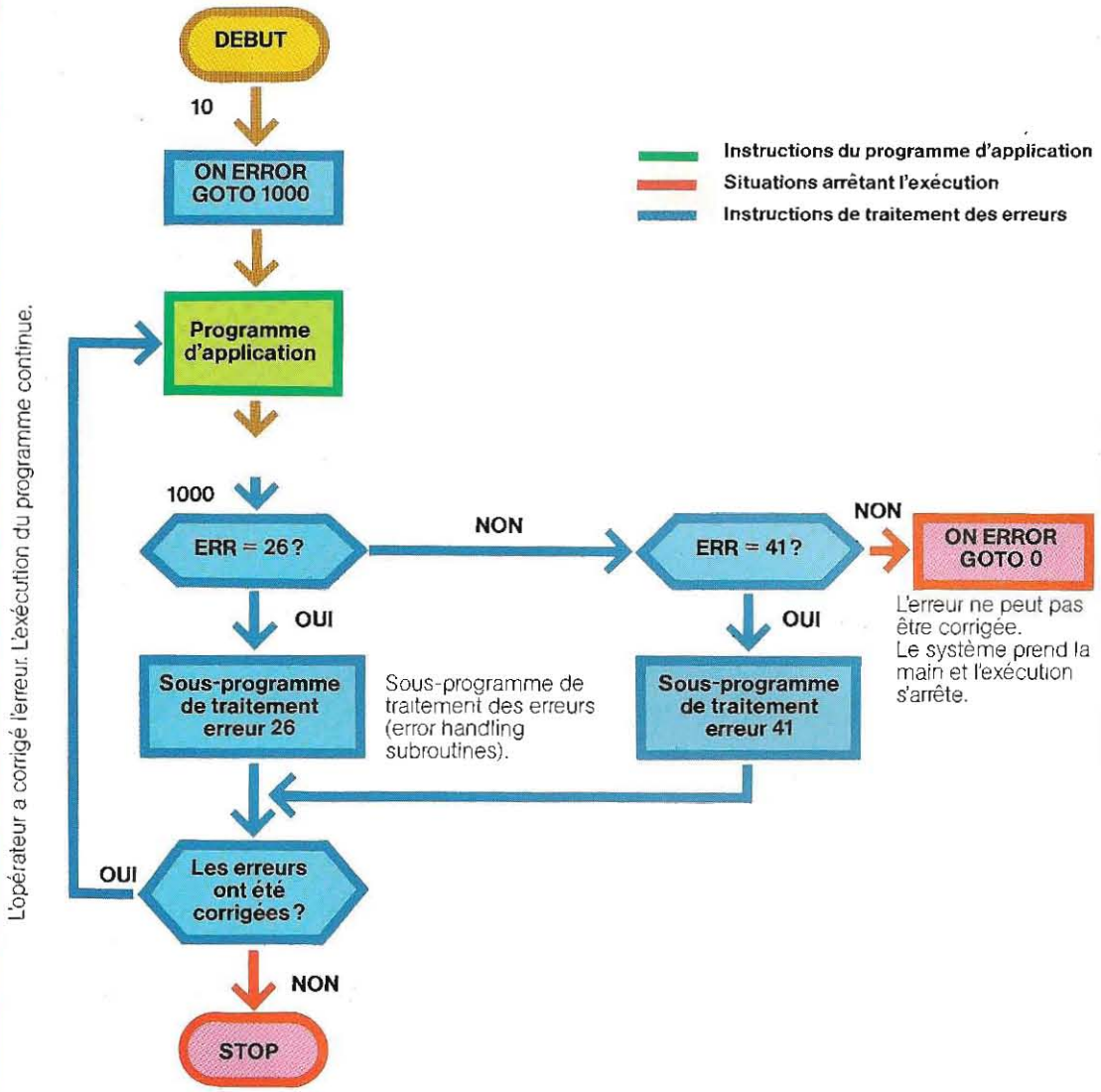
Les **erreurs de programmation** proviennent d'une mauvaise utilisation des instructions. Citons quelques exemples comme l'oubli de la fermeture d'une boucle, l'omission de l'instruction RETURN à la fin d'un sous-programme, une incohérence entre le nombre de variables d'une instruction READ et le nombre de données de l'instruction DATA associée, un numéro de ligne inexistant dans l'instruction GOTO ..., etc.
Normalement, l'interpréteur Basic possède un bon module de diagnostic, et les messages qui s'affichent à l'écran suffisent à l'identification de l'erreur.
Les **erreurs de logique** sont souvent très complexes, et ne génèrent pas de message d'erreur. Leur recherche passe obligatoirement par une analyse attentive des organigrammes, ou par l'exécution d'un cas déjà résolu manuellement. La comparaison entre les résultats obtenus par la machine et ceux calculés à la main peut aider à repérer l'endroit du programme contenant l'erreur. Le mode « trace » (TRON) constitue un autre moyen de recherche des erreurs de logique. En lançant un programme après avoir tapé la commande TRON, on verra apparaître à l'écran les numéros des instructions qui seront successivement exécutées. De cette manière, on suivra le déroulement du programme en vérifiant que l'exécution se poursuit normalement.
Les **erreurs survenant dans l'exécution des calculs** sont semblables aux précédentes. Elles proviennent d'une utilisation incorrecte des parenthèses ou des opérateurs, ou de l'utilisation d'un type de variable inapproprié. Si, par exemple, une variable déclarée comme entière (avec l'instruction DEFINT ...) prend une valeur supérieure à 32767 (ou inférieure à – 32768), (type erroné, débordement), le système donne son diagnostic. Dans les autres cas, il appartiendra au programmeur seul de rechercher la cause première des erreurs.
Les **erreurs système** sont imputables à un mauvais fonctionnement de la machine, ou à des causes accidentelles extérieures,

comme une mauvaise manipulation des mémoires de masse (l'opérateur, par exemple, oublie d'insérer une disquette, ou se sera trompé d'unité). On reviendra plus tard sur ce dernier type d'erreur.
Un programme bien conçu comportera au moins une routine de traitement des erreurs (diagnostic et, éventuellement, corrections). On accèdera à ce sous-programme de deux façons : par l'instruction d'appel ON ERROR GOTO ... (pour les erreurs détectées par le système), ou par des instructions de diagnostic spécifiques (écrites par le programmeur). L'instruction ON ERROR GOTO n, n étant le numéro de ligne où commence le « piège » (trap) de traitement de l'erreur, ne peut être utilisée qu'une fois au cours du programme.
A l'apparition d'une erreur quelconque (détectée par le système), si le programmeur n'adopte pas l'instruction ON ERROR GOTO, le système donnera son propre diagnostic et arrêtera l'exécution du programme, même si l'erreur est minime. Le programme quittera la phase d'exécution et ne pourra la reprendre qu'à partir du début. En revanche, l'instruction ON ERROR GOTO n provoquera l'exécution d'un module de traitement des erreurs, débutant à la ligne n. En analysant le type d'erreur, le module décidera d'arrêter l'exécution, ou de la poursuivre en demandant à l'opérateur une correction opportune.
Le code numérique qualifiant le type d'erreur commise est contenu dans une variable nommée ERR, alors que le numéro de la ligne où l'on a constaté l'erreur est représenté par la variable ERL. Ces deux variables (dont le nom diffère selon le type de Basic) sont réservées au système (« variables système »).
En haut de la page 419 est présenté l'organigramme général d'implantation de ce module. L'instruction ON ERROR GOTO 1000, insérée au début du programme, mènera en ligne 1000 lorsqu'une erreur apparaîtra. Le module 1000 prévoit la correction de deux erreurs (types 26 et 41). L'apparition d'une des deux erreurs déterminera l'appel du sous-programme de traitement correspondant. Si, au contraire, l'erreur détectée n'est pas d'un type prévu, l'instruction ON ERROR GOTO 0 (qui annule ON ERROR GOTO 1000), rendra le contrôle au système ; ce dernier, après avoir émis le diagnostic Basic normal, arrêtera l'exécution. Le contrôle de bon fonctionnement des sous-programmes de traitement des erreurs peut être fait avec l'instruction ERROR (n), qui simule une erreur de type n.
Les **erreurs commises en phase de saisie des données** ne sont normalement pas détectées par le système.
Le programme d'application contrôlera lui-même l'exactitude des données saisies et, en cas d'erreur, affichera un message avant de formuler une nouvelle demande. Les seuls contrôles effectués par le système concernent le nombre et le type des variables demandées en entrée.
En phase de saisie des données, le contrôle de validité sur les valeurs fournies en entrée du programme est déterminant. Deux logiques différentes pourront être mises en œuvre. La première prévoit un contrôle immédiat, dans le sous-programme de saisie lui-même ; la seconde consiste à implanter une routine de contrôle appropriée, qui sera appelée par chaque instruction de saisie.
La première méthode sera appliquée à un contrôle non répétitif, qui reste lié à une seule phase de saisie. Sa programmation ne présente aucune difficulté : on effectuera les tests nécessaires immédiatement après la saisie. Si le contrôle s'avère négatif, on demandera une nouvelle saisie.
La seconde technique est employée pour regrouper dans un seul module toutes les instructions de diagnostic. Ce sous-programme généralisé, appelé par plusieurs points du programme, sera, au contraire, très complexe. Il faudra, en effet, envisager et traiter toutes les erreurs possibles. Malgré cela, l'adoption de cette méthode sera conseillée dans les programmes d'une certaine complexité, pour obtenir une programmation claire et structurée (diagnostics regroupés en un seul point du programme).

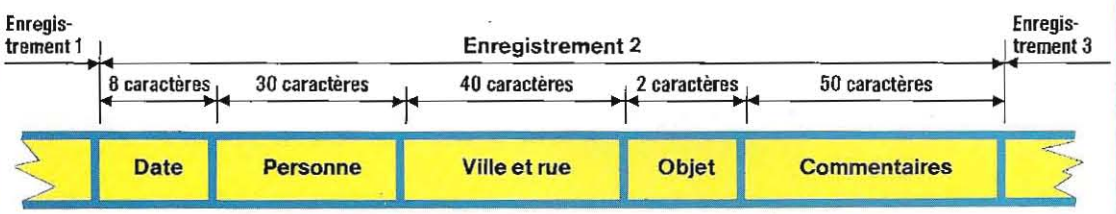
Exemple d'application = gestion d'un agenda de rendez-vous

Illustrons la mise en place de la seconde technique qui vient d'être exposée, par le programme de gestion d'un agenda de rendez-vous.
Les données sont stockées dans un fichier appelé AGENDA, ordonné chronologiquement.
Le format d'enregistrement est le suivant (voir schéma en bas de la page 419) :

DETECTION ET CORRECTION EVENTUELLE DES ERREURS



FORMAT D'ENREGISTREMENT DU FICHIER AGENDA



La « tourismatique » est née

Nul se songerait à nier l'importance décisive du rôle joué par le tourisme dans l'économie de certains pays d'Europe et son importance dans l'équilibre de leur balance des paiements.
Selon des sources officielles, on a dénombré, pour l'année 1983, 54 millions de nuits d'hôtels, sur Paris et sa région.
Pour faire face aux perspectives de crise qui semblent atteindre même le secteur touristique des pays industrialisés, pour affronter la concurrence d'autres pays particulièrement équipés dans le domaine du tourisme organisé, il a fallu améliorer la structuration des activités touristiques à l'aide de techniques adéquates parmi lesquelles l'informatique joue un rôle fondamental.

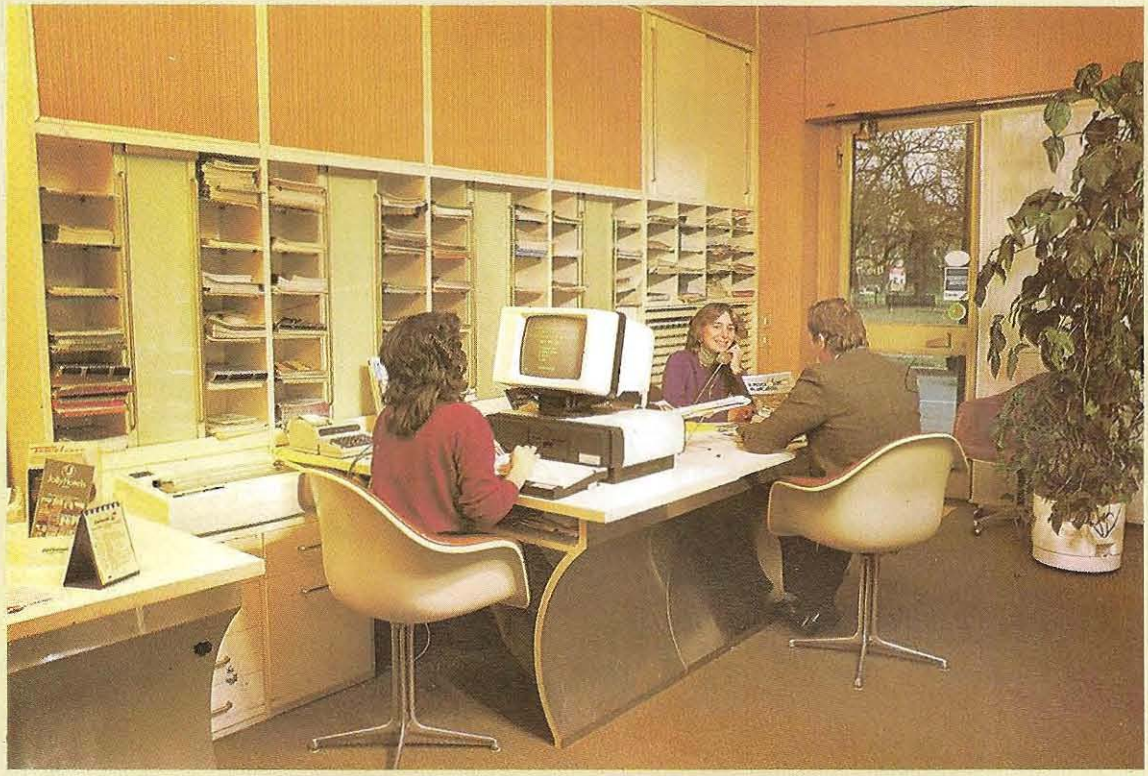
* * *

Aujourd'hui, les progrès de la technologie et la demande d'un public toujours plus exigeant ont provoqué le rapprochement de l'informatique et des télécommunications, deux mondes qui, jusqu'à présent, s'étaient développé séparément.
Dans plusieurs pays d'Europe (France, Italie, Hollande, entre autres), de nouveaux services destinés aux secteurs du tourisme et de l'hôtellerie ont été mis en place.
Ils fonctionnent en étroite collaboration avec des sociétés d'informatique qui ont fourni micro-ordinateurs et mini-ordinateurs.

La ***gestion d'un hôtel****, par exemple, est d'une grande complexité et s'avère difficile à rationaliser.*
D'une part, il faut tenir compte d'un élément fondamental, directement issu de la tradition hôtelière, qui est la prestation du personnel, les services qu'il rend et ses qualités d'amabilité et de courtoisie. Cet élément représente encore l'un des critères du choix d'un hôtel et une garantie de sa qualité.
D'autre part, le secteur hôtelier ne peut faire abstraction aujourd'hui des problèmes et des objectifs économiques fondamentaux de notre société moderne :

– détermination du rapport qualité-prix pour l'optimisation correcte du chiffre d'affaires ;
– rationalisation de la structure d'organisation interne en vue d'une diminution des coûts.

La réalisation de ces deux objectifs est liée à l'expérience et aux capacités d'entreprise de la direction, mais elle peut être améliorée par l'utilisation d'un système de gestion automatisée des informations.
Ce système, appliqué aux différents secteurs de l'activité hôtelière (accueil, logement et restauration), offrira au personnel une base de travail plus efficace. Grâce au réseau public de transmission de données (Transpac en France), accessible à tous les usagers, de n'importe quel point géographique du pays (la distance n'ayant pas d'incidence sur la tarification), le simple particulier peut à tout moment s'informer de la disponibilité des places sur les vols de compagnies aériennes et dans les hôtels de toute catégorie, pour peu qu'ils soient équipés du matériel informatique adéquat.
L'usager peut non seulement ***s'informer****, mais aussi* ***réserver*** *une place d'avion ou une chambre d'hôtel sans se déranger, par l'intermédiaire d'un terminal ou du téléphone.*
Le système est semblable à Télétel, qui fournit, grâce à un petit terminal, des informations sur les spectacles ou le prix des voitures d'une marque donnée, mais le mode est interactif, dans la mesure où la réservation est enregistrée auprès de l'organisme (compagnie aérienne ou hôtel), grâce à l'ordinateur, sans que les personnes se soient parlé.
Les clients ont aussi à leur disposition un système de réservation automatique, qui bénéficie de toutes les possibilités de traitement de l'ordinateur. La question est, naturellement, de savoir dans quelle proportion le simple particulier est disposé (et capable) de s'équiper d'un terminal.
C'est pourquoi les ***agences de voyages*** *voient leur rôle d'intermédiaire se développer chaque jour davantage.*
Devenant elles-mêmes de véritables bureaux de tourisme, elles proposent à leurs clients des prestations de service de plus en plus sophistiquées, et pour lesquelles le support informatique se révèle plus que jamais nécessaire.
Aujourd'hui, l'agent de voyages entreprenant, et son équipe, doivent faire face à de multiples problèmes :

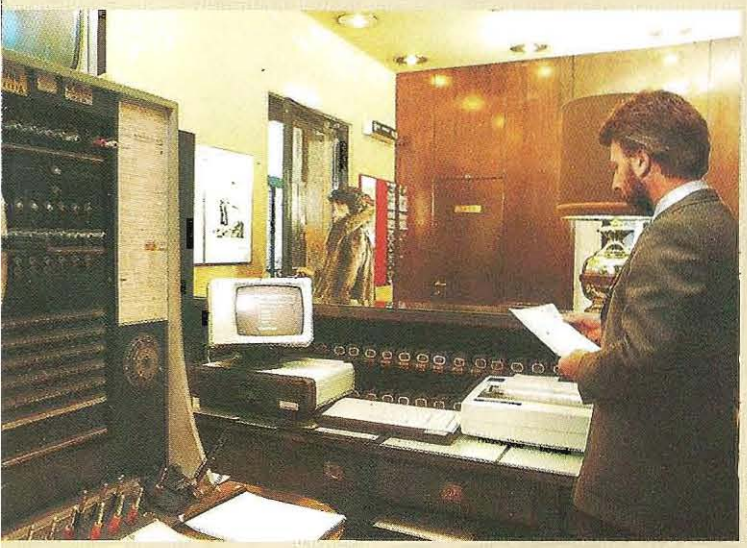


Honeywell



Honeywell

Micro-ordinateur personnel Honeywell Questar/M9050, installé dans les bureaux d'une agence de voyages.



Honeywell

Projet « tourismatique » : un micro-ordinateur Questar/M9050 à la réception d'un hôtel.

– engagement toujours plus poussé dans la mise en place de voyages organisés et donc la gestion des services fournis lors des pointes saisonnières ;
– normes de plus en plus strictes dans les domaines fiscaux et administratifs ;
– nécessité absolue d'offrir à leur clientèle un service public d'excellente qualité, passant par un personnel hautement sélectionné.

* * *

Il semblerait que l'informatisation des agences de voyages ait précédé celle des structures hôtelières. Au cours de ces dernières années, on a constaté néanmoins un fort développement de la demande de matériel et de logiciel dans le domaine de la gestion hôtelière.
Il s'agit d'une part de la gestion des comptes de l'hôtel (paie du personnel, budget, restauration, investissements éventuels pour l'entretien et l'aménagement des locaux), mais également de la gestion des réservations et des comptes clients.
L'informatisation des services hôteliers implique une restructuration du travail des diverses catégories de personnel, mais il est toutefois nécessaire que le changement se réalise dans un contexte qui valorise le capital de compétence et de professionalisme des employés.
Il faut donc comprendre en quoi consistent les nouvelles techniques d'automation et comment elles peuvent être appliquées à la réalité touristique.
Le terme ***tourismatique*** *ne doit pas faire croire qu'il suffit d'appliquer aux hôtels et aux agences de voyages des techniques informatiques déjà adoptées par d'autres secteurs. Il s'agit en fait de développer des moyens spécifiques permettant la rationalisation et l'évolution du service touristique.*

* * *

En Italie par exemple, c'est précisément ce que la société Honeywell a présenté aux professionnels du secteur touristique : deux solutions d'application sur micro-ordinateurs et mini-ordinateurs, l'une étudiée pour la gestion des hôtels, l'autre pour les agences de voyages.

La solution appliquée aux hôtels fonctionne soit sur micro-ordinateur Honeywell Questar/M, soit sur mini-ordinateur Honeywell DPS 6, chacun desservant plusieurs postes de travail et s'articulant sur les cinq modules suivants :

– réservations ;
– check-in ;
– gestion des besoins du client sur place ;
– check-out ;
– administration et comptabilité.

Le module de ***réservation*** *permet de gérer la disponibilité des chambres sur une période de deux ans, d'effectuer des réservations multiples (plusieurs chambres avec une seule réservation), de tenir compte des préférences des habitués, d'imprimer la liste des départs et des arrivées pour tous les services de l'hôtel qui en ont besoin.*
Il permet aussi la gestion de ce qu'on appelle l'over-booking (quand la demande est trop élevée) et, au contraire, en période creuse, le regroupement des réservations dans certaines parties ou étages de l'hôtel, de façon à limiter à ces parties les dépenses (chauffage et services).
Le module ***check-in*** *assure l'enregistrement des clients qui arrivent (s'il s'agit de clients réguliers, les données sont automatiquement prélevées dans les archives), la mise à jour des différents fichiers, le relevé des papiers d'identité pour la Préfecture, évitant ainsi que,*

à la suite d'un oubli ou d'une erreur, ces documents soient mal établis.
Le module de **gestion des besoins du client** sur place se charge de la mise à jour de la situation comptable du client à tout moment sur la base des services qu'il a demandés (blanchisserie, téléphone, mini-bar), prenant en compte aussi les conventions spéciales dont il pourrait bénéficier. Il prévoit, en outre, la gestion des changements de chambre éventuels et, au moment du départ, met à jour les différentes archives parmi lesquelles l'archive réservations.
Le module **check-out** sert à émettre pour chaque client, au moment de son départ, la facturation des divers mouvements de son compte. Simultanément, ce module assure la mise à jour des archives comptables de l'hôtel.
Ces dernières sont gérées par le module **administration et comptabilité de gestion**, assurant de plus l'impression des différents documents utiles :

- registre de caisse ;
- registre de débit ;
- bordereaux pour le service comptable ;
- situation et récapitulation des comptes.

D'une part, le système est caractérisé par une utilisation facile, respectant les traditions hôtelières, puisqu'il doit être utilisé par un personnel généralement dénué d'expérience en informatique et habitué à un autre type de langage et de comportement. D'autre part, il se distingue par sa grande flexibilité, particulièrement nécessaire dans un milieu où la règle est, depuis toujours, de s'adapter à des situations non standardisées. Son utilité est reconnue, par exemple, dans le cas d'arrivées ou de départs non prévus, de comptes séparés pour des occupants d'une même chambre ou, au contraire, de débit de chambres différentes sur le même compte, et enfin, de rectification de dernière minute du compte d'un client, etc.
La solution informatique proposée aux agences de voyages fonctionne sur micro-ordinateurs Honeywell Questar/M avec différents terminaux. Elle est fondée sur quatre modules :

- gestion des dossiers de voyage ;
- gestion des guichets ;
- gestion des réservations faites ailleurs.
- comptabilité générale.

Le module **de gestion des dossiers de voyage** traite aussi bien le transport que l'hébergement, ainsi que les différents services qui y sont liés, tant pour les particuliers que pour les groupes. Sont prévus tous les changements de date, calculs de quota et éventuellement, montant de devises par voyageur.
Les documents émis par le module sont : bons de réservation, confirmations de voyage, listes de voyageurs, moyens de transport, hôtels et services, listes de billets à émettre. Il est relié, après confirmation préalable de l'opérateur, au module de comptabilité générale pour l'enregistrement comptable des opérations, l'émission de factures, les extraits de compte.
Le module **gestion des guichets** concerne les billets d'avion, de chemin de fer, etc., qu'ils soient pris sur place ou ailleurs. Ce module assure une véritable gestion de magasin avec contrôle des stocks. Il gère en outre une archive billets avec mémorisation des diverses données : émission, remboursement ou annulation éventuelle du billet.
Le module de **gestion des réservations par intermédiaire** concerne les opérations de réservation effectuées par un client auprès d'une autre compagnie (nationale ou étrangère). Il assure l'émission de bons de réservation analogues à ceux émis par le module dossiers de voyage (ces derniers formant une seule archive « bons de réservation » pour les contrôles d'exploitation et de comptabilité ultérieurs), ainsi que l'émission de documents propres à la comptabilité de l'intermédiaire lui-même (commission sur ventes, etc.).
Enfin, le module de **comptabilité générale**, au moyen d'un plan de compte personnalisé, gère la comptabilité clients-fournisseurs à comptes ouverts, permet l'enregistrement de la T.V.A., comptabilise les coûts pour les services comptables (dossiers de voyage), effectue les bilans de vérification, de récapitulation et de clôtures périodiques.
Comme nous l'avons déjà dit, la majeure partie des mouvements comptables étant automatiquement exécutée par les autres modules, elle permet une réduction maximale du travail manuel et limite la possibilité d'erreurs.

1 / Dates des rendez-vous :
Mois = 2 caractères (entier, de 1 à 12).
Jour = 2 caractères (entier, de 1 à 31).
Heure = 2 caractères (entier, de 8 à 20 ; seules sont considérées les heures diurnes).
Minutes = 2 caractères (entier, de 0 à 50 par pas de 10).
On arrondira systématiquement les minutes à la dizaine supérieure.

2 / Personne à rencontrer : 30 caractères (Nom et Prénom).

3 / Ville et rue : 40 caractères.

4 / Objet du rendez-vous : 2 caractères (code).

5 / Commentaires : 50 caractères.

La longueur du fichier sera mémorisée dans l'enregistrement 1.
Le programme aura à exécuter les fonctions suivantes :

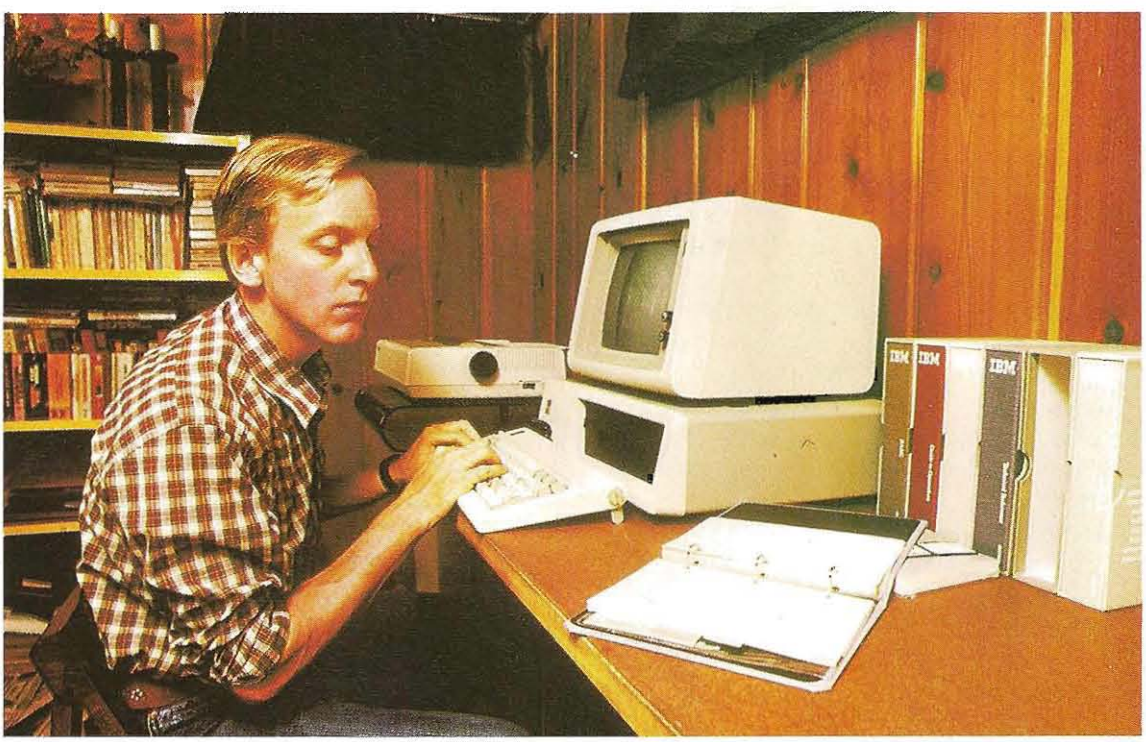
a) impression de la liste des rendez-vous entre deux dates fixées ;
b) sélection des rendez-vous se passant dans une ville déterminée ;
c) sélection des rendez-vous selon un objet précis.

Un tel programme s'articulera ainsi :

- initialisation ;
- lecture des paramètres de sélection ;
- recherche et impression des données.

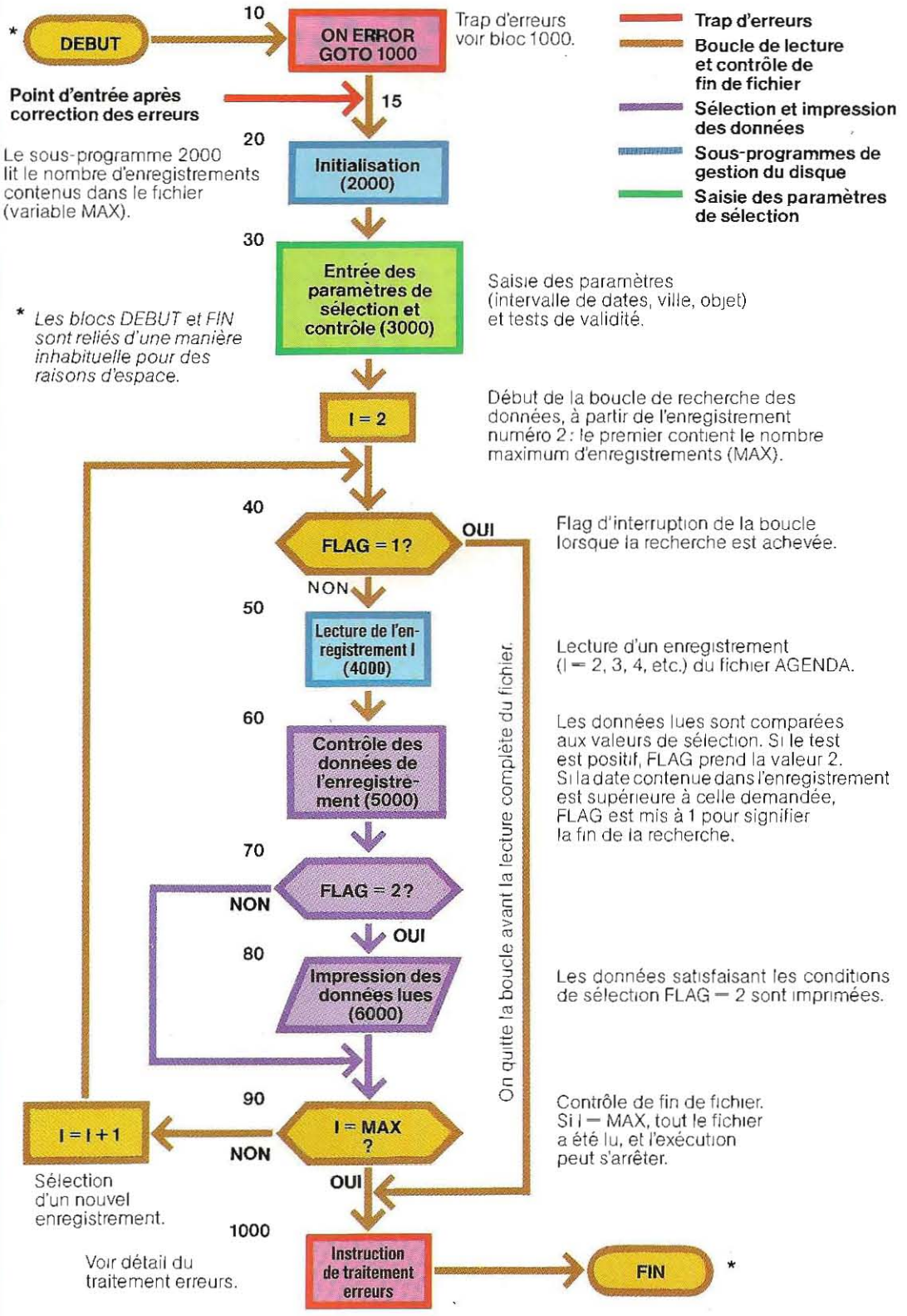
En phase d'initialisation, on vérifiera l'existence du fichier AGENDA, et on lira sa longueur dans l'enregistrement 1 (voir p. 231).
La lecture des paramètres de sélection consistera concrètement à saisir l'intervalle de dates, la ville ou l'objet souhaités (ces deux derniers paramètres étant optionnels).
La dernière partie prévoit la lecture séquentielle du fichier et, pour chacun des enregistrements, la vérification de sa correspondance avec les données saisies. Si les critères de recherche sont satisfaits, on imprimera l'enregistrement avant de traiter le suivant.
La page 425 présente l'organigramme de base : un indicateur a été prévu pour sortir de la boucle avant la lecture complète du fichier, dans le cas où la date contenue dans l'enregistrement lu est supérieure à celle demandée (FLAG = 1).
Le parcours du fichier met en œuvre la technique des boucles à rupture de séquence : le fichier est lu séquentiellement, à partir du premier enregistrement. Si la date contenue dans l'enregistrement lu est inférieure à la valeur maximale saisie, il faudra continuer la

Avec un micro-ordinateur, la gestion d'un agenda devient une simple formalité.



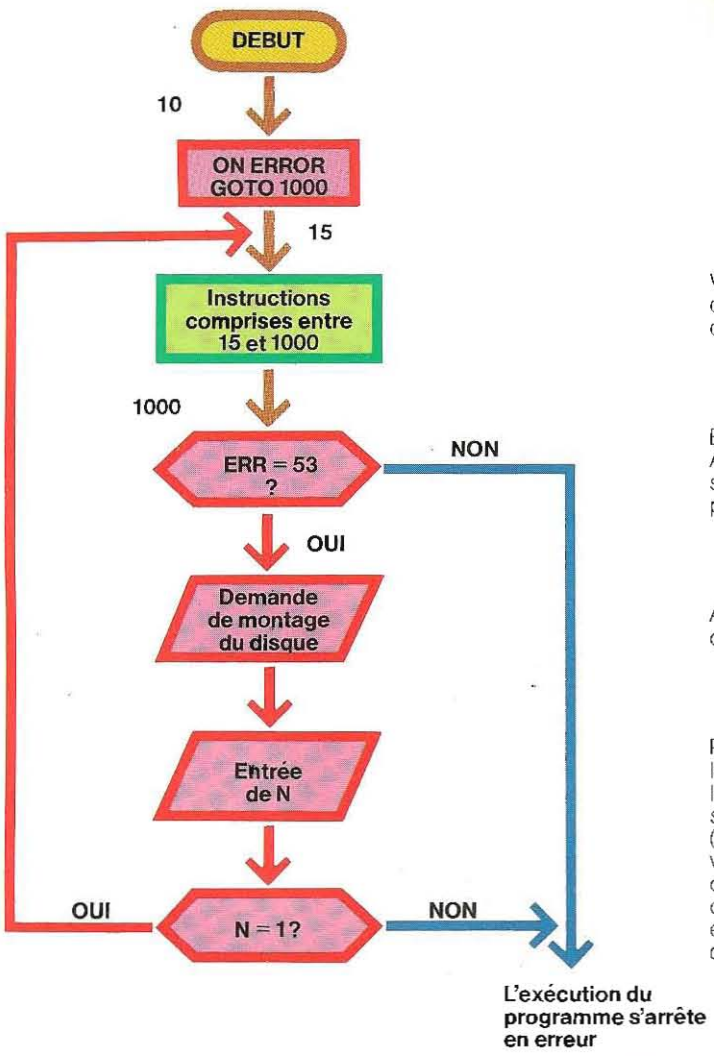
Honeywell

ORGANIGRAMME SIMPLIFIE DU PROGRAMME DE GESTION AGENDA



lecture afin de trouver les autres enregistrements éventuels à sélectionner. La recherche peut être abandonnée au premier enregistrement contenant une date supérieure à la date maximale saisie.
Ci-dessous, on a détaillé le schéma du module de traitement des erreurs (bloc 1000). En ce cas, la seule erreur système correspond à l'absence du fichier AGENDA sur disque (code 53, à titre d'exemple). A l'apparition de cette erreur, on affiche un message d'avertissement; la machine attendra ensuite la fin des opérations de correction que l'utilisateur doit effectuer (en ce cas particulier le remplacement de la disquette). On suspendra l'exécution en insérant une instruction d'entrée à la console; l'opérateur disposera alors de tout le temps nécessaire pour remplacer le disque; en tapant la valeur choisie (ici 1), il réactivera le traitement.
L'organigramme de la page 427 détaille la saisie des paramètres de sélection (routine 3000); alors que le suivant (page 428) correspond au sous-programme 5000, prévu pour le traitement des données lues dans le fichier, en fonction des paramètres établis.
Le sous-programme 5000 lira les données MOIS, JOUR, HEURE, MINUTE, VILLE, OBJET, de chaque enregistrement du fichier AGENDA, après avoir reçu en entrée les para-

DETAIL DU MODULE DE TRAITEMENT DES ERREURS



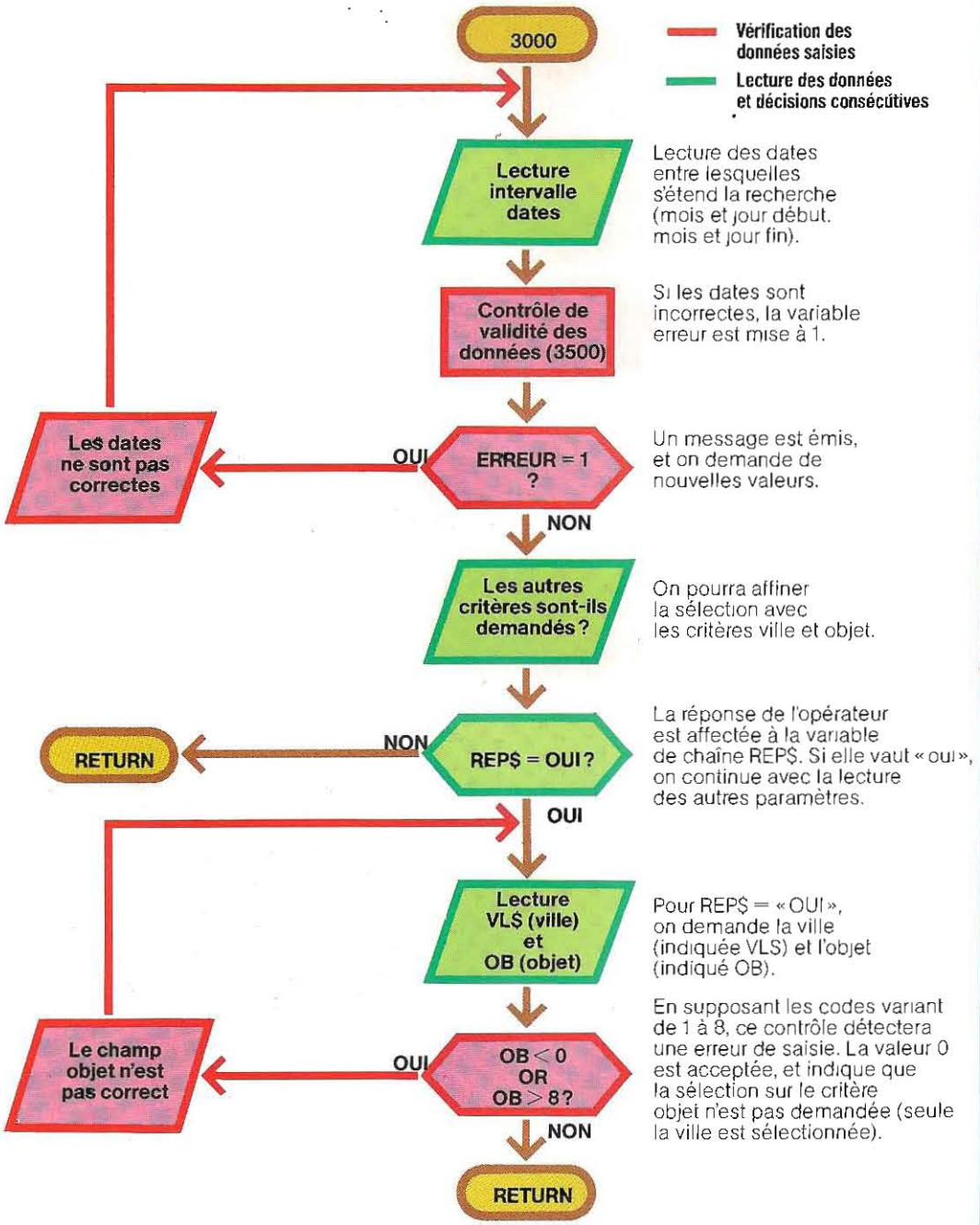
Voir organigramme simplifié du programme de gestion du fichier AGENDA.

Erreur de code 53 : le fichier AGENDA n'a pas été trouvé sur le disque. Autre erreur : pas de traitement prévu.

Affichage d'un message d'avertissement.

Pour donner à l'opérateur le temps de monter le disque, le système se met en attente sur la saisie d'une variante quelconque (par ex., N). L'opérateur peut vérifier le disque et décidera de continuer (en entrant la valeur 1) ou d'arrêter l'exécution (en entrant une valeur quelconque différente de 1).

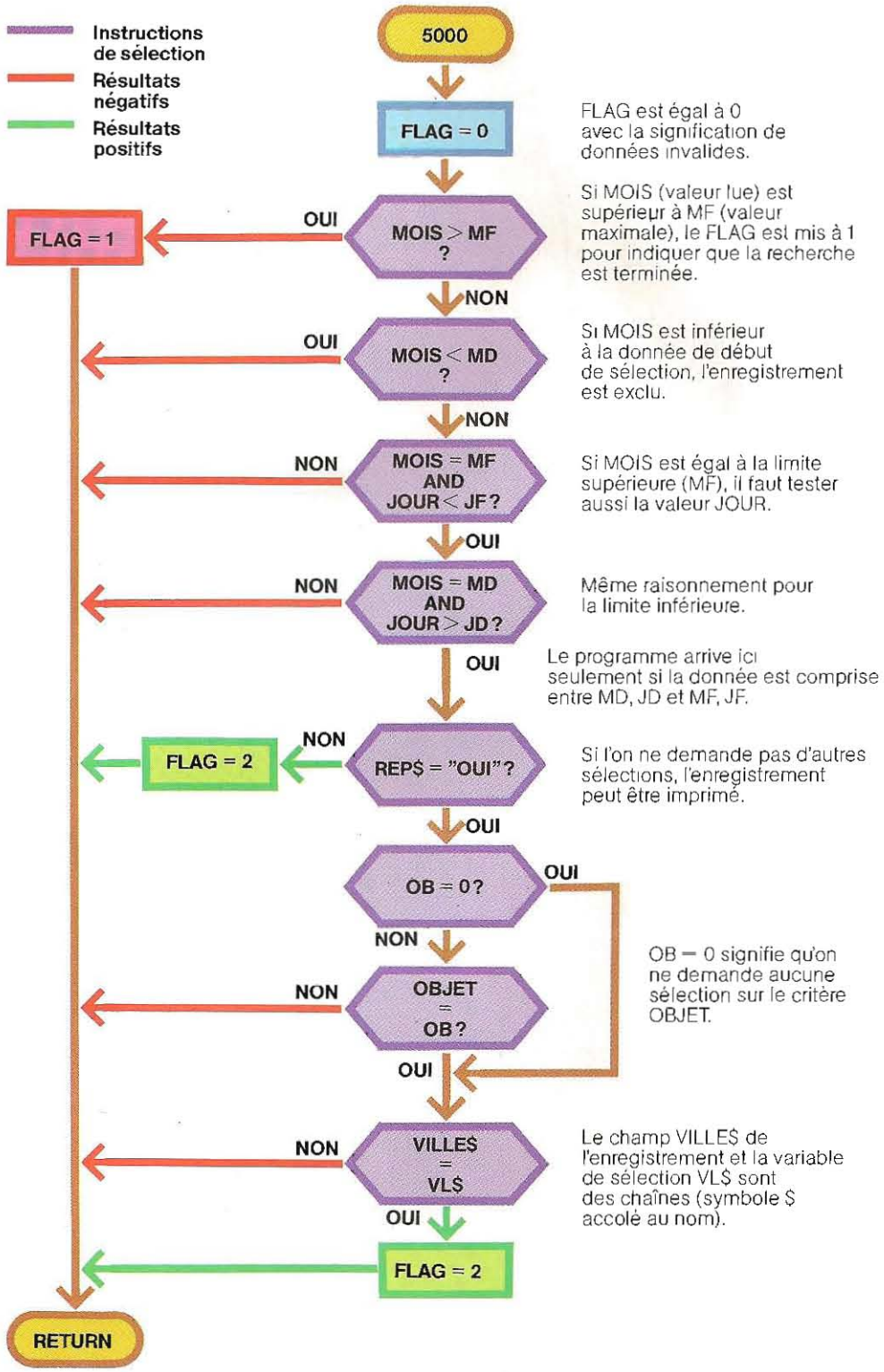
SOUS-PROGRAMME DE SAISIE DES PARAMETRES DE SELECTION



Paramètres fournis en sortie :

JD (jour), MD (mois) = date de début de sélection
JF (jour), MF (mois) = date de fin de sélection
REP$ = "OUI" si d'autres sélections sont demandées
VL$ = ville à sélectionner (chaînes de caractères)
OB = objet (code)

SOUS-PROGRAMME DE TRAITEMENT DES DONNEES



mètres (date de début de la recherche, date de fin de la recherche, ville, objet) acquis par le sous-programme 3000.
Dans le sous-programme 5000, la comparaison entre les deux chaînes VILLE$ et VL$ se résume à un simple test :

(IF VILLE$ = VL$ GOTO ...).
En sortie du sous-programme 5000, le paramètre FLAG prendra les valeurs :

0 = enregistrement non sélectionné ;
1 = date lue supérieure à la date maxi-

PROGRAMME DE GESTION D'UN AGENDA

```
]LIST

1 REM **********************
2 REM ** PROGRAMME AGENDA  **
3 REM **********************
4 REM
5 REM
6 REM programme execute sur APPLE II
7 REM en Basic Applesoft
8 REM ** PROGRAMME PRINCIPAL **
9 REM
10 ONERR GOTO 1000
11 REM
20 GOSUB 2000: REM lecture de la taille du fichier AGENDA
21 REM
30 GOSUB 3000: REM saisie des parametres de selection
31 REM
40 FOR I = 2 TO MAX: REM debut de boucle
41 REM
50 IF FLAG = 1 GOTO 1070: REM on abandonne la lecture
51 REM
60 GOSUB 4000: REM lecture enregistrement
70 GOSUB 5000: REM controle des donnees
71 REM
80 IF FLAG < > 2 GOTO 100
81 REM
90 GOSUB 6000: REM impression des donnees selectionnees
91 REM
100 REM
101 REM
102 REM
103 REM
104 REM ** traitement des erreurs **
105 REM
1000 IF ERR < > 53 THEN PRINT "erreur non prevue": STOP
1001 REM
1010 PRINT "erreur disque"
1020 PRINT "montez la disquette contenant le fichier AGENDA"

1030 PRINT "tapez 1 pour continuer, 2 pour arreter"
1031 REM
1040 INPUT N
1050 IF N = 1 GOTO 20
1051 REM
1060 PRINT "sortie en erreur": STOP

1061 REM
1070 END
1071 REM
1072 REM
1073 REM * routine de test et lecture longueur d'AGENDA *
1074 REM (detaillee par la suite)
2000 MAX = 5: REM valeur fixe pour simulation
2010 RETURN
2011 REM
2012 REM
2013 REM
2014 REM * routine de saisie des parametres *
2015 REM
3000 PRINT "dates limites ?"
3010 INPUT "mois debut ";MD: INPUT "jour debut ";JD
3020 INPUT "mois fin ";MF: INPUT "jour fin ";JF
3030 REM controle des dates saisies
3040 IF MD > 12 OR MF > 12 OR JD > 31 OR JF > 31 THEN PRINT "erreur date": GOTO 3000
3050 INPUT "autre selection? (oui/non) ";REP$
3060 IF REP$ = "NON" GOTO 3100
3070 INPUT "ville de selection ";VL$
3080 INPUT "code objet, entre 1 et 8 ";OB
3090 IF OB < 0 OR OB > 8 THEN PRINT "code errone": GOTO 3060
3100 RETURN
3101 REM
3102 REM
3103 REM
```

```
3104 REM *routine de lecture des enregistrements*
3105 REM detaillee par la suite, elle est ici simulee
3107 REM
4000 PRINT "simulation de lecture sur disque"
4020 INPUT "mois ";MOIS: INPUT "jour ";JOUR
4030 INPUT "ville ";VILL$: INPUT "objet ";OBJET
4040 RETURN
4041 REM
4042 REM
4044 REM *routine de selection*
4045 REM
5000 FLAG = 0
5010 MO$ = "date hors des limites de selection"
5011 REM
5020 IF (MOIS > MF) OR (MOIS = MF AND JOUR > JF) THEN FLAG = 1: PRINT MO$: GOTO 5090
5030 IF (MOIS < MD) OR (MOIS = MD AND JOUR < JD) THEN PRINT MO$: GOTO 5090
5031 REM
5040 IF REP$ < > "OUI" GOTO 5090
5050 IF OB = 0 GOTO 5070
5060 IF OB < > OBJET THEN PRINT "objet non selectionne": GOTO 5090
5070 IF VILL$ < > VL$ THEN PRINT "ville non selectionnee": GOTO 5090
5080 FLAG = 2
5090 RETURN
5091 REM
5092 REM
5094 REM *routine d'impression des resultats*
5095 REM
6000 PRINT "RENDEZ-VOUS PREVUS"
6010 PRINT "DU : ";JD;"/";MD;" AU : ";JF;"/";MF
6020 PRINT "donnees memorisees"
6030 PRINT "mois ";MOIS
6040 PRINT "jour ";JOUR
6050 PRINT "objet ";OBJET
6060 PRINT "ville ";VILL$
6080 RETURN

]RUN
dates limites?
mois debut 3
jour debut 14
mois fin 5
jour fin 16
autre selection? (oui/non) OUI
ville de selection PARIS
code objet, entre 1 et 3 3
simulation de lecture sur disque
mois 4
jour 12
ville LYON
objet 3
ville non selectionnee
simulation de lecture sur disque
mois 3
jour 22
ville PARIS
objet 3
RENDEZ-VOUS PREVUS
DU : 14/3 AU : 16/5
donnees memorisees
mois 3
jour 22
objet 3
ville PARIS
simulation de lecture sur disque
mois 5
jour 1
ville LYON
objet 2
date hors des limites de selection
```

male choisie; il faut dorénavant cesser la recherche;

2 = sélection réussie; impression des données de l'enregistrement.

Le listing de la page 429 réunit les différents modules :
programme principal : organigramme de la page 425;
sous-programme 3000 : organigramme de la page 427;
sous-programme 5000 : organigramme de la page 428,
ainsi qu'un exemple d'exécution. Les parties manquantes du programme (modules de gestion du disque et de lecture des enregistrements), seront présentées par la suite. Pour l'instant, quelques instructions les simuleront.

En ajoutant une routine de saisie des données, nous aurons ainsi constitué un programme complet de gestion d'un agenda.

L'homme face à la machine

Depuis l'avènement de la société industrialisée, l'homme n'a cessé de s'adapter aux machines qui se faisaient de plus en plus nombreuses autour de lui. De nos jours, la mode est à l'adaptation inverse, et l'on accorde plus d'attention à la «machine humaine».

*Une science se charge des problèmes posés par cette nouvelle préoccupation : c'est l'**ergonomie**, ou « l'art et la science de concevoir des systèmes capables de supprimer ou de limiter les sensations douloureuses, physiques ou mentales, réelles ou imaginaires ».*

L'ergonomie étudie les conditions de confort, de santé, de sécurité et d'équilibre mental des personnes vivant dans un milieu de travail, et considère le rapport de l'homme et de la machine comme un problème à résoudre par des moyens physiologiques, psychologiques ou techniques. Si les premières tentatives d'automatisation du travail de bureau ont été l'objet d'une certaine résistance, c'est que les rapports de l'homme avec la machine ont été initialement considérés comme antagoniques. On pensait, en effet, qu'il appartenait aux opérateurs de s'adapter à ces bureaux climatisés, extrêmement soignés, certes, mais également dépourvus de vie. Cette conception représentait l'obstacle le plus important à surmonter : l'univers de la machine était imposé à l'homme.

C'est seulement depuis peu que, sous l'influence de la recherche ergonomique, les conditions de travail commencent à se modifier, dans les bureaux du moins.

La technologie de pointe n'est plus autant perçue aujourd'hui comme l'ennemi à abattre. On comprend de mieux en mieux que les machines dotées de capacités de plus en plus performantes sont destinées à faciliter le travail de l'homme. La plupart des problèmes physiques ont reçu une solution directe et immédiate dans l'élaboration électronique. Les angles des machines de bureau se sont arrondis, devenant ainsi moins dangereux.

Des études précises ont été réalisées afin de mettre en évidence la réaction de la rétine lorsque l'œil fixe un écran lumineux pendant plusieurs heures.

Le NIOSH (National Institute of Occupation Safety and Health) et l'Académie américaine pour les Sciences sont arrivés à la conclusion que la quantité de radiations émise par les écrans est trop faible pour provoquer des cataractes, ou pire, des dommages génétiques. Elles n'ont aucune répercussion sur la santé. En revanche, d'autres problèmes n'ont pas été aussi faciles à résoudre. Une enquête, également menée par le NIOSH, a recueilli quantité de données sur différents troubles réels ou imaginaires.

La cause n'est pas toujours et nécessairement imputable à la machine elle-même : beaucoup de ces troubles peuvent être provoqués par un mauvais éclairage, un siège mal adapté ou une ambiance psychologiquement désagréable. En effet, la solution d'un problème lié à la machine révèle souvent l'existence d'autres problèmes, d'origine psychologique, ceux-là.

James Martin, auteur de « Man – Computer Dialogues », affirme que chaque utilisateur nécessite une interface homme – machine différente. Les hommes appartiennent, selon lui, à des catégories hétérogènes, incapables parfois de « communiquer » entre eux et qui, devant un ordinateur, peuvent passer à l'improviste d'une catégorie à l'autre.

Martin les divise en couples : attentif - distrait, expert - inexpert, intelligent - non intelligent, actif - passif, affirmant que pour chaque utilisateur, il est nécessaire de trouver un moyen idéal de communication : alphanumérique, sonore, ou graphique. Martin écarte immédiatement le langage naturel, celui-ci demandant un logiciel trop complexe ; sa sympathie ne va pas non plus aux langages spécifiques de programmation tels le Cobol et le Fortran, ceux-ci exigeant des niveaux de professionnalisme et de motivation trop élevés pour la masse des utilisateurs.

Dans un article publié en 1982, Paul Heckel a proposé à son tour une classification d'utilisateurs d'ordinateurs, en soutenant qu'un bon concepteur de logiciel doit connaître à fond son public.

« Tout le monde parle sa propre langue, mais chacun de nous a ses propres expressions et, selon sa profession, un jargon lié à une activité donnée. De ce fait, des individus se consacrant à des professions différentes tendront à penser aussi d'une manière différente. Combien de fois nous a-t-on demandé de faire des programmes utilisables par tous,

même par des individus sans aucune connaissance en la matière. Si seulement cela était possible ! La plupart de nos problèmes proviennent au contraire de ce que l'on a implicitement pensé que l'utilisateur pourrait se défaire facilement des connaissances acquises auparavant dans l'exercice de sa profession et s'adonner spontanément à un nouveau jargon ».

Le jargon lui-même, cauchemar des puristes, peut faciliter l'approche avec le nouvel usager, à condition que l'on emploie avec lui le jargon qui lui est familier. Il faut parler avec la secrétaire le langage de la secrétaire, avec le médecin, le langage du médecin, avec le commerçant, le langage du commerçant. Faute de quoi l'informaticien ne saura pas cerner clairement le problème précis auquel le traitement informatique doit être appliqué. Ce qui est vrai du logiciel l'est aussi du matériel.

Un manuel, une page vide ou un menu mal écrit peuvent nuire à l'esprit, davantage qu'une mauvaise adaptation du clavier à la main. Un curseur qui clignote peut, seulement en théorie, attendre éternellement qu'on appuie sur la touche requise, car, pour l'esprit humain, le problème se pose en d'autres termes. Selon « Convergent Technologies », une société de projets ergonomiques, un opérateur ne peut attendre une réponse pendant plus d'une seconde et demie. En revanche, dans « Man – Computeur Dialogues », Martin affirme que si le temps de réponse est trop bref, l'utilisateur inexpérimenté suspecte une inexactitude des données. Le fait que l'attention humaine n'est soutenue que pendant un laps de temps limité constitue de toute manière un des aspects les plus critiques du face à face homme-machine. Ces derniers temps, on a longuement discuté de l'opportunité de l'utilisation de ***menus*** *au lieu de commandes. Avec les menus, l'utilisateur n'a qu'à choisir parmi un certain nombre de propositions, sans avoir à se souvenir de toutes les commandes, l'ordinateur lui présentant automatiquement une liste de choix possibles. De cette manière, le temps passé à apprendre est considérablement réduit puisqu'aucun effort de mémoire n'est plus nécessaire. Très appréciés des débutants, les menus sont terriblement ennuyeux pour les utilisateurs expérimentés, obligés de subir, écran après écran, des choix qu'ils connaissent par cœur. C'est pour cette raison qu'un bon « package »* doit avoir plusieurs sous-programmes de « by-pass », offrant des raccourcis aux utilisateurs plus expérimentés. Même les jeux vidéo de type Dragons et Donjons tiennent compte de ce principe.*

Quand on parle avec un inconnu, généralement on cherche d'instinct à évaluer son niveau intellectuel : comprend-il ce qu'on lui dit ? Participe-t-il à la discussion ? De cette manière, on se fait une opinion qui pousse à adopter une certaine intonation et un certain type de langage. Pourquoi donc un logiciel ne devrait-il pas réagir de la même manière ? Selon Gerry Gassman, responsable d'ergonomie des systèmes de gestion de Hewlett-Packard, on doit surtout définir avec le plus grand soin le contexte dans lequel la nouvelle machine devra être utilisée, chaque milieu nécessitant une approche différente.

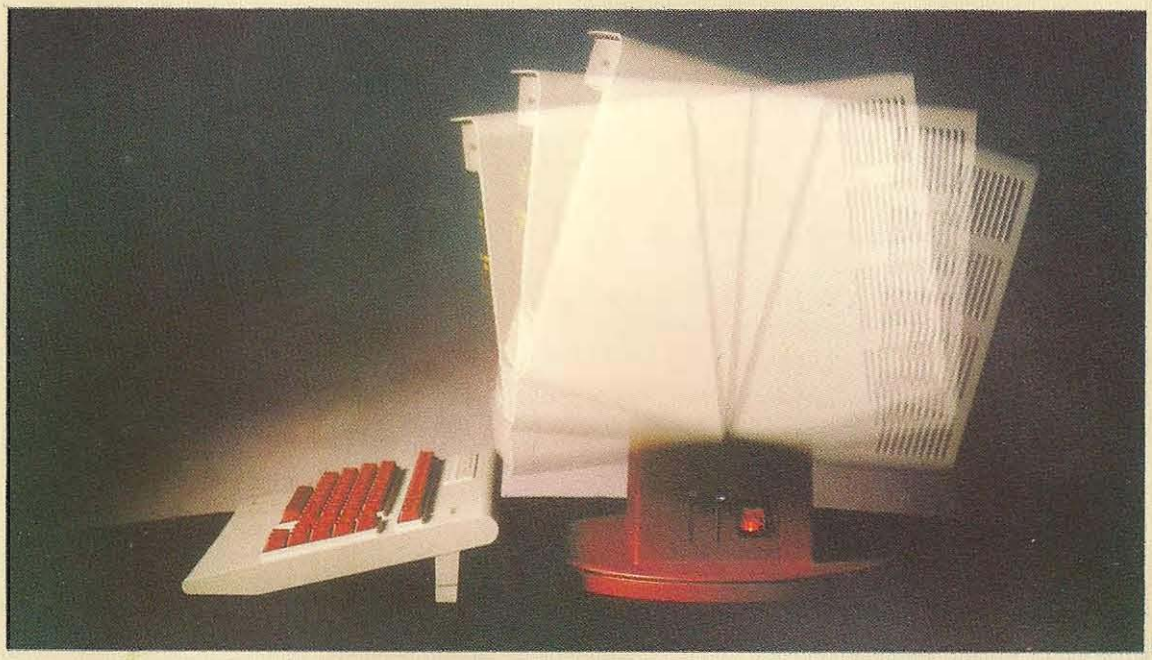
Par exemple, dans un centre informatique ou dans une zone de service, les opérateurs travaillent le plus souvent debout.

Disposition ergonomique des terminaux d'un centre de calcul de la Société Hewlett-Packard aux Etats-Unis.



K. Reese/Marka

* Les « packages » sont des ensembles de programmes d'application, destinés à la solution de problèmes scientifiques ou de gestion et mis en vente en complément des systèmes de base.



Ericsson

L'écran orientable est un élément de confort appréciable.

Dans ce cas, le tableau de commande sera plus visible et plus facile à atteindre s'il se trouve sur le devant et vers le haut de l'appareil. Au contraire, si l'utilisateur travaille assis à un bureau, les commandes seront à hauteur des yeux ; de même le pupitre de commande et les unités d'E/S devront être à portée de la main. Une soigneuse analyse des modalités d'utilisation accompagne toujours la recherche de ceux qui conçoivent la forme des ordinateurs. Confiée à des spécialistes d'esthétique et de psychologie industrielles, cette recherche particulière implique des entretiens avec des utilisateurs potentiels, la construction de prototypes et l'essai de ces machines en particulier par des utilisateurs néophytes. Selon M. Gassman, la société Hewlett-Packard s'impose des contraintes encore plus sévères que celles établies par la loi, afin de pouvoir être toujours à l'avant-garde et de ne jamais être prise au dépourvu par une nouvelle norme.

De nombreux problèmes ergonomiques ont été résolus sur le plan technique, mais il reste encore beaucoup à faire en pratique. Tandis qu'aux Etats-Unis on se fie essentiellement à la loi du marché, avec la certitude qu'il est plus facile de vendre des machines à la fois belles et pratiques, les Européens sont plutôt enclins à légiférer. C'est ainsi qu'en octobre 1980, l'Institut d'assurances contre les accidents du travail d'Allemagne Fédérale a publié une liste de normes de sécurité concernant le matériel de bureau. Depuis janvier 1982, la RFA est le premier pays au monde à disposer d'une ***législation*** *en matière d'ergonomie.*

A la CBEMA (Computer and Business Equipment Manufacturers Association), où l'on étudie ce problème dans une optique mondiale, on reconnaît la nécessité d'une ***normalisation*** *tout en émettant des réserves sur la validité technique et la souplesse des standards de référence. En effet, dans le monde entier, l'ergonomie se contente, le plus souvent, de corriger des erreurs de conception parce qu'elles ont été signalées comme responsables de troubles physiques chez l'utilisateur. L'ergonomie ne se limite pourtant pas à cela, et de nombreux spécialistes cherchent à construire des appareils capables, par leur forme, de mettre l'utilisateur dans les meilleures conditions matérielles susceptibles d'encourager leur créativité.*

On a ainsi pu élaborer une nouvelle conception de l'ergonomie qui ferait de la machine un véritable stimulateur de l'activité cérébrale humaine.

(d'après DATA MANAGER, septembre 1983 ; © INTERACT janvier-février 1983).

Les fonctions

On appelle fonction une relation qui associe une valeur à une ou plusieurs autres.
Le cas le plus simple et que l'on rencontre le plus fréquemment est celui de fonctions qui ne relient que deux variables et qui, à une valeur de l'une, font correspondre une valeur de l'autre.
On peut, par exemple, dire que la distance parcourue par une voiture avec un litre de carburant est fonction de sa vitesse.
Pour déterminer la relation qui associe le trajet accompli à la vitesse, il nous faudra mettre dans le réservoir de la voiture une quantité donnée de carburant (1 litre par exemple) et mesurer le nombre de kilomètres que nous réussirons à parcourir avec ce carburant en maintenant une vitesse constante (ce paramètre doit être fixe également).

En répétant cette expérience avec des vitesses différentes (toujours avec la même quantité d'essence) et en mesurant chaque fois le parcours effectué, on obtiendra une table de correspondance entre la vitesse et la distance couverte :

Vitesse km/h	Distance parcourue (km pour un litre)
40	16
60	14
80	12
100	11
120	10

Ce tableau donne une représentation de la fonction reliant le trajet à la vitesse, mais ne permet pas de déterminer toutes les caractéristiques de cette relation.

Vade-mecum de la console idéale

Ce sont sans doute les terminaux (clavier et écran) qui ont bénéficié des études ergonomiques les plus poussées. Voici les critères de qualité qui reviennent le plus souvent :

1. Les opérateurs sont amenés à changer fréquemment de position durant leur travail face au terminal. Après plusieurs heures de saisie de données et de traitement de texte ou d'une activité comparable, les épaules ont tendance à s'affaisser et la tête à s'incliner d'une vingtaine de degrés par rapport à la verticale. Pour tenir compte du fait que les gens ne restent pas figés dans une position immuable, l'écran devrait former avec l'horizontale un angle moyen situé entre 10 et 40°. En outre, il doit être orientable en fonction d'éventuelles variations d'éclairage.

2. L'écran devrait être placé à une cinquantaine de centimètres des yeux de l'utilisateur ; le clavier et le pupitre aussi. Des changements continuels de longueur focale provoquent en effet une fatigue rapide de la vue.
Sur un système vraiment bien conçu, l'opérateur devrait pouvoir ne régler qu'un de ses paramètres, les autres s'adaptant automatiquement.

3. Toutes les touches de commande doivent être à portée de main, le bras étant demi-fléchi.

4. Les ventilateurs qui assurent le refroidissement des terminaux et des unités de disque risquent de provoquer divers troubles (dessèchement de la cornée, douleurs rhumatismales). Les courants d'air chaud devront, par conséquent, être détournés du visage et ceux d'air froid du reste du corps.

5. L'opérateur doit pouvoir s'installer correctement. Dans ce but, un pupitre à surface anti-reflet prendra place derrière le clavier. Une inclinaison réglable entre 15 et 75° par rapport à la verticale permettra d'y écrire plus facilement.

6. Les écrans à caractères verts ou jaunes semblent les moins fatiguants pour la vue. Si les premiers sont adaptés aux éclairages puissants, les seconds conviennent mieux aux lumières diffuses. Le choix dépend donc non seulement des goûts de chacun mais des caractéristiques du bureau.

7. Selon certains experts, les passages perpétuels d'un texte écrit noir sur blanc à l'écran où le contraste est inversé fatiguerait les pupilles en les obligeant à s'adapter continuellement. Pour d'autres, la lumière émise par l'écran n'a pas le même effet sur l'œil que celle réfléchie par le papier ; la rétine aurait tendance à atténuer le contraste en créant une sorte de halo autour des caractères lumineux.

8. Sur l'écran, les caractères perdent peu à peu leur netteté avant d'être « rafraîchis ». L'effet stroboscopique qui en résulte est particulièrement éprouvant tant pour la vue que pour les nerfs. L'augmentation de la luminosité totale ne fait qu'accentuer ce phénomène. C'est pourquoi la plupart des écrans présentent des caractères clairs sur fond sombre et les réécrivent de 50 à 60 fois à la seconde.

9. Une réflexion importante peut rendre l'écran illisible et donner mal à la tête. On peut parer à cela en utilisant du verre dépoli, des filtres, des revêtements, ou en rendant l'écran réglable. Le verre gravé et les filtres sont néfastes à la netteté et les revêtements coûtent cher. L'adoption d'un écran réglable constitue, en revanche, une solution simple et économique.

Définition d'un diagramme par points

Pour visualiser la fonction plus aisément et de façon plus détaillée, on peut établir un graphique (le diagramme par points de la fonction) à l'aide du tableau que nous venons de voir.
Il convient tout d'abord de fixer une échelle pour la vitesse et une pour la distance.
Par exemple :
1 cm pour 20 km
5 mm pour 2 km parcourus avec un litre de carburant.
Suivant cette convention, la vitesse de 40 km/h sera représentée par un segment de 2 cm et le parcours correspondant (16 km) par un segment de 40 mm.
Pour construire le graphique, nous devons transcrire les valeurs de la table en cm pour les vitesses et en mm pour les distances, selon l'échelle utilisée :

Vitesse km/h	Equivalence en cm	Distance km	Equivalence en mm
40	2	16	40
60	3	14	35
80	4	12	30
100	5	11	27,5
120	6	10	25

Désignons par **y** la valeur du parcours exprimée en mm et par **x** la valeur de la vitesse correspondante (en cm). On peut alors reporter les valeurs de **x** à l'horizontale et celles de **y** à la verticale, et établir ainsi le schéma de la page 436. On appelle axe des **x** (ou des **abscisses**) la droite horizontale supportant l'échelle des vitesses et axe des **y** (ou des **ordonnées**) la droite verticale symbolisant l'échelle des distances parcourues.
On obtient ainsi une représentation graphique de la relation entre la vitesse et la dis-

10. Le contraste est également un point important. Certaines normes européennes préconisent un taux compris entre 3 et 15. De plus, un réglage du contraste doit être possible pour l'adapter aux conditions d'éclairage et aux besoins de chacun.

11. Travailler longtemps sur un clavier est moins fatigant si l'on peut tenir les coudes pliés à 90° et si les poignets forment un angle inférieur à 10° de l'horizontale. La position du clavier devra donc le permettre.

12. Selon certains techniciens (allemands notamment), l'épaisseur du clavier idéal dépasse à peine 3 cm. Des claviers aussi minces sont fort peu répandus mais l'installation d'un appuie-main d'une certaine épaisseur devant le clavier permet de se replacer dans les conditions idéales.

13. L'inclinaison du clavier contribue également au confort d'utilisation en réduisant la fatigue des bras, des poignets et des coudes. Là aussi, les avis sont partagés : on recommande, en Allemagne, une inclinaison inférieure à 15°. D'autres assurent qu'un angle inférieur à 7° est à la fois source d'erreurs et d'inconfort.

14. Opérateurs expérimentés et novices n'ont pas toujours les mêmes besoins et il est pénible d'être placé face à l'écran, alors que l'on porte plutôt son attention sur le clavier. De même, un débutant préférera entendre le déclic de chaque touche.
Si les opérateurs chevronnés s'accommodent mieux d'un clavier moins sensible, tous s'accordent cependant sur la nécessité de percevoir une réaction minimale de la touche afin de ne pas manquer et de ne pas répéter une frappe.

15. Le « rollover » est la faculté de la machine à accepter une cadence de frappe très rapide. Des pointes de vitesse de 200 à 300 caractères/minute peuvent en effet survenir lors de la frappe de mots ou passages familiers à l'opérateur, et selon la « virtuosité » dont il est capable.

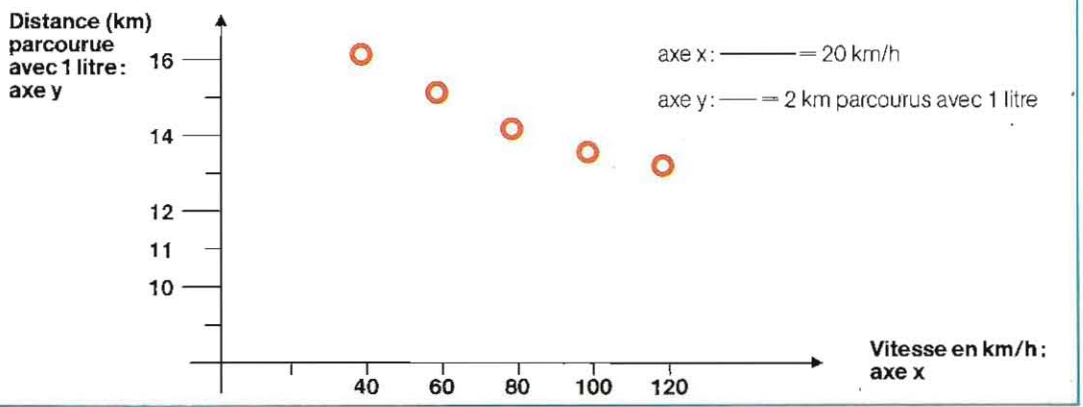
16. La disposition du clavier IBM Selectric fait désormais figure de standard et une configuration différente ne réussit qu'à troubler l'opérateur et à le retarder dans son travail. En outre, il lui faudra des semaines pour s'adapter à ce matériel. Par contre, la touche de majuscules, conçue à l'origine pour les téléscripteurs, présente peu d'utilité en informatique et serait mieux remplacée par une touche de blocage.

17. Un clavier est composé de trois zones. La zone principale comprend les caractères alphanumériques et quelques commandes d'utilisation fréquente. Les commandes d'écran sont regroupées dans une autre zone qui doit être accessible sans détourner les yeux. Quant à la troisième zone, elle nécessite un déplacement du regard, des bras et des mains et ne contrôle donc que des fonctions secondaires moins utilisées. Cette répartition varie bien entendu selon la spécialisation de l'appareil. Un opérateur de traitement de texte n'a pas les mêmes besoins qu'un comptable qui utilisera surtout le pavé numérique. En outre, l'agencement du clavier tient compte, comme celui d'une machine à écrire, des lettres les plus fréquemment employées dans une langue donnée et les répartit en conséquence.

18. Enfin, toujours dans le souci de ménager la vue, certains experts recommandent pour le clavier un coefficient de réflexion compris entre 20 et 50 pour cent.

(d'après DATA MANAGER n° 26, septembre 1983).

DIAGRAMME PAR POINTS DE LA FONCTION DISTANCE-VITESSE



tance en posant :
1 cm = 20 km/h et 5 mm = 2 km.
Ces deux valeurs (1 cm et 5 mm) sont les unités respectives de l'axe **x** et de l'axe **y** qui permettent de convertir les variables en longueurs.
La longueur à reporter sur un des axes pour une valeur donnée se calcule par une règle de trois :

longueur axe **x** : valeur **x** = 5 : 2 (5 mm = 2 km)

longueur axe **y** : valeur **y** = 1 : 20 (1 cm = 20 km/h)

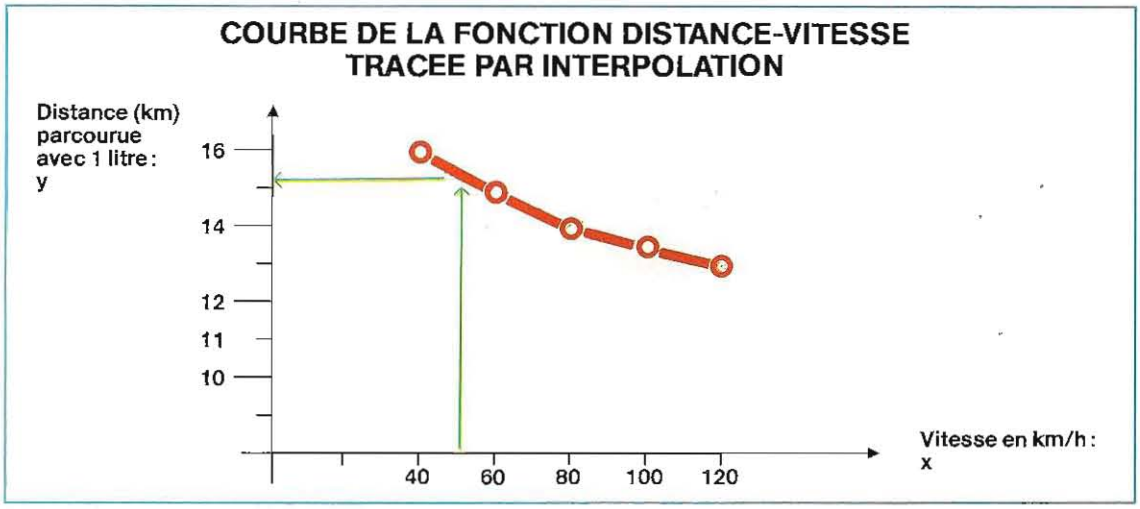
$$\text{longueur axe } \mathbf{x} = \frac{5 \times \text{valeur } \mathbf{x}}{2} = \frac{5}{2} \times \text{valeur } \mathbf{x}$$

$$\text{longueur axe } \mathbf{y} = \frac{1 \times \text{valeur } \mathbf{y}}{20} = \frac{1}{20} \times \text{valeur } \mathbf{y}$$

Ces facteurs $\frac{5}{2}$ et $\frac{1}{20}$, sont appelés **coefficients d'échelle.**
La formule générale est :

$$\text{longueur sur l'axe} = \text{coefficient d'échelle} \times \text{valeur}$$

Nous avons ainsi construit une table et un graphique grâce à la mesure d'un phénomène physique : la variation du parcours en fonction de la variation de vitesse. Et ce, à partir du choix arbitraire d'un certain nombre de vitesses (par ex., 40, 60, 80 km/h) et la mesure des kilométrages correspondants.
Nous voici donc en possession d'une série de résultats ponctuels mais nous n'avons pas déterminé la relation continue associant ces deux variables.
Cependant, ayant pris des mesures assez rapprochées, nous sommes en droit de penser qu'à une vitesse intermédiaire entre deux vitesses étudiées correspond un kilométrage compris entre les deux distances correspondantes.
Nous allons donc relier les points de coordonnées connues par des segments de droite et admettre que tous les couples (**x**, **y**) que nous aurions pu trouver expérimentalement sont situés sur la courbe ainsi obtenue.
Le graphique de cette fonction est alors terminé, et le schéma de la page 437 montre comment trouver la distance couverte à une autre vitesse (50 km/h dans cet exemple).
Cependant, la courbe tracée n'est qu'approximative, car la relation exacte entre **x** et **y** est une formule mathématique qui donne **y** en fonction de **x**.
Néanmoins, il est certain que la courbe réelle passera par les points que nous avons trouvés expérimentalement.
La définition d'une fonction reliant deux variables **x** et **y** permet de calculer la valeur de **y** associée à un **x** donné.
La variable **x**, dont on peut fixer arbitrairement la valeur (ici la vitesse), est dite indépendante.
L'autre variable (ici la distance) est au contraire dépendante de **x** puisque déduite de la formule.

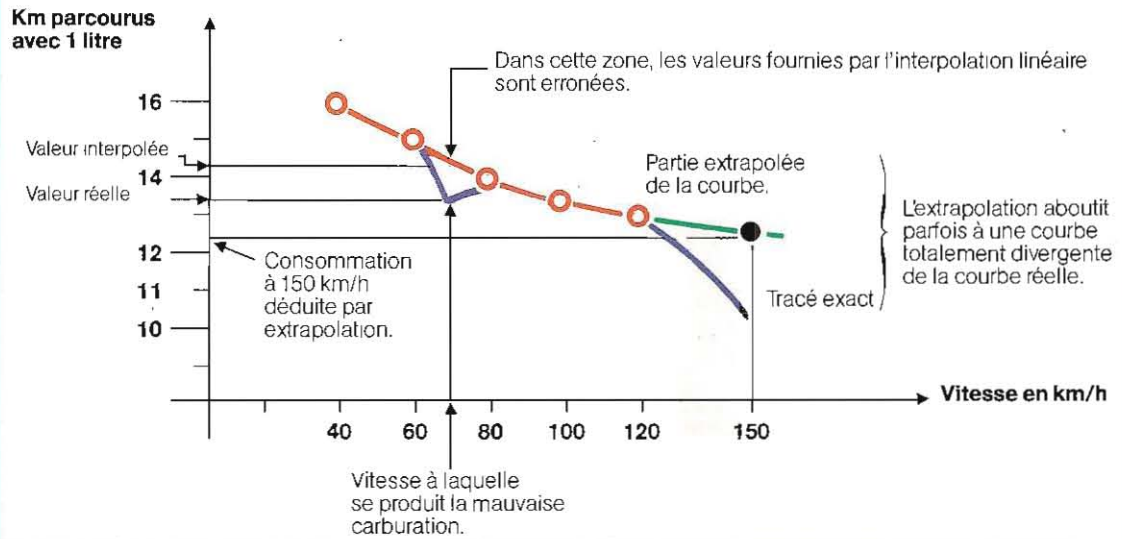


On représente généralement la **variable indépendante** sur l'axe des abscisses (**x**) et la **variable dépendante** sur l'axe des ordonnées (**y**).
Une fonction faisant dépendre une variable **y** d'une autre variable **x** se note généralement : **y = f(x)**, soit, en clair :
« les valeurs prises par la variable dépendante **y** sont fonction des valeurs de la variable indépendante **x** ».
Nous avons vu que, pour représenter graphiquement la fonction **y = f(x)** qui, dans notre cas, correspond à :
distance = f(vitesse),
on choisit des valeurs de la variable indépendante (**x**, vitesse) et on mesure les valeurs correspondantes de la variable dépendante (**y**, distance).
La liberté de choix des valeurs de **x** est toutefois limitée car il n'est pas toujours possible d'effectuer une mesure à 200 km/h (dans le cas où la voiture concernée ne peut atteindre cette vitesse).
On veillera donc à ne pas choisir des valeurs de **x** totalement arbitraires.
L'intervalle dans lequel la variable **x** peut prendre ses valeurs s'appelle le **domaine de définition de la fonction.**

Interpolation et extrapolation

Pour l'exemple que nous avons choisi, nous ne disposions que d'un nombre limité de mesures (40, 60, 80, 100 et 120 km/h, soit cinq valeurs au total) et nous avons tout de même tracé une courbe continue en reliant les points connus.
Cette opération, qui semble aller de soi, relève en réalité d'un procédé mathématique complexe : **l'interpolation linéaire.**
Interpoler signifie déduire de deux ou plusieurs résultats des valeurs intermédiaires.
C'est ce que nous avons fait sur le schéma de cette page en traçant les segments de droite qui sont censés relier tous les points correspondants aux mesures non effectuées.
En reliant les points correspondant aux vitesses de 40 et 60 km/h, nous avons effectué une interpolation linéaire entre ces deux relevés nous permettant d'estimer les consommations intermédiaires (par exemple à 50 km/h).
L'interpolation linéaire est presque toujours acceptable d'un point de vue mathématique, mais elle ne fournit des valeurs proches de la réalité que dans le cas de courbes au tracé régulier.
Imaginons, en effet, que la voiture testée présente un défaut de carburation entre 60 et 80 km/h (voir page 438).
Le pic de consommation ainsi provoqué affecte la régularité de la courbe et on commettrait une erreur grossière en utilisant l'interpolation linéaire pour déduire la distance parcourue à 70 km/h.
On peut toutefois diminuer ce risque en effectuant des mesures intermédiaires de façon à interpoler des points plus rapprochés.
En dépit des risques d'erreurs encourus, l'interpolation est très souvent utilisée dans les calculs scientifiques.
Il existe d'ailleurs d'autres types d'interpolation que l'interpolation linéaire.

INTERPOLATION ET EXTRAPOLATION



Par exemple, on approche beaucoup plus de la réalité en reliant les points par des courbes plutôt que par des droites.

Un autre procédé très employé en mathématiques est **l'extrapolation.**

La vitesse maximale que nous avons testée était de 120 km/h.

Or, si notre véhicule peut atteindre 150 km/h, le domaine de définition de la fonction distance/vitesse s'étend jusque-là aussi.

Mais puisque nous n'avons pas testé la vitesse maximale, il nous est impossible de trouver, par interpolation, les valeurs de la consommation entre 120 et 150 km/h.

Nous allons donc être obligés de le faire par extrapolation, en prolongeant la courbe intuitivement.

Cette méthode peut également s'appuyer sur des calculs mathématiques, mais si, pour interpoler, il suffit d'imaginer le tracé entre deux points connus (avec peu de chances de s'écarter beaucoup du tracé réel), l'extrapolation est beaucoup plus hasardeuse.

En effet, nous ignorons tout de la courbe après le dernier point.

L'évolution de la fonction après la dernière mesure peut très bien présenter une modification, voire une inversion, de la tendance précédente.

Justement, dans notre exemple, la consommation augmente rapidement au voisinage de la vitesse maximale, et le tracé de la

courbe chute de façon imprévisible. Dans ce cas, une extrapolation mènerait à des déductions entachées d'une erreur importante (voir le schéma de la page 438).

Représentation analytique des fonctions

La représentation graphique des fonctions permet une visualisation globale de leur tracé mais ne peut en aucun cas constituer une méthode de calcul.
On pourrait éventuellement estimer les frais de carburant à différentes vitesses, en utilisant le graphique pour déduire la consommation.
Mais ce travail ne peut être effectué par un ordinateur. Ces machines travaillent sur des données précises et sont théoriquement inaptes à « évaluer » une valeur à partir d'un graphique.
La solution est de trouver une formule mathématique dont le tracé se rapproche du graphique. L'application de cette formule permettra alors de calculer la distance correspondant à une vitesse donnée.
Ainsi, pour ce qui est de notre exemple, la formule suivante donnerait une bonne approximation de la fonction :

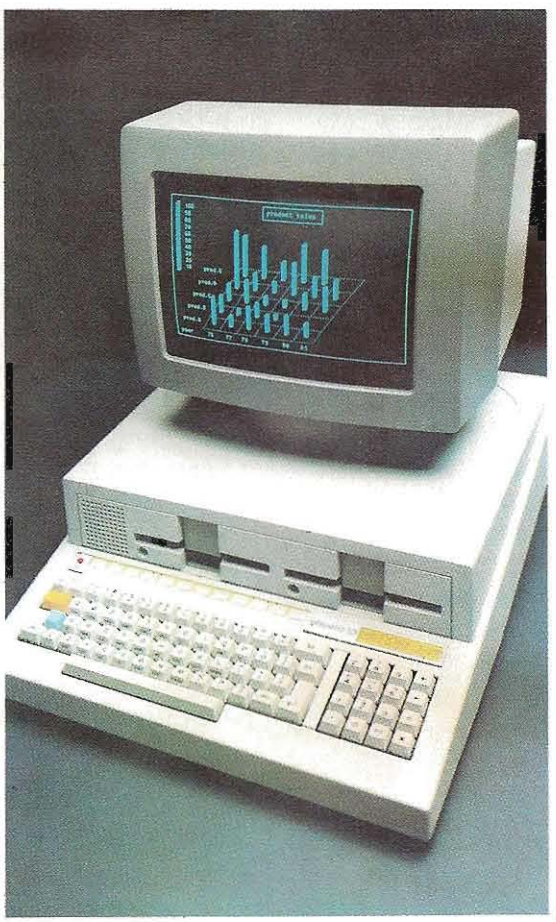
$$\begin{matrix}\text{km parcourus}\\ \text{avec 1 litre}\end{matrix} = 18.6 - 0.07 \times \begin{matrix}\text{vitesse}\\ \text{en km/h}\end{matrix}$$

Il suffit dès lors de reporter la vitesse dans cette formule et d'effectuer les calculs indiqués pour trouver la distance correspondante.
Cette expression arithmétique est la **représentation analytique** de la fonction.
En programmation, on doit exprimer sous forme analytique toutes les fonctions qu'on veut utiliser.
Deux cas peuvent se présenter :
soit il s'agit d'une fonction déterminée et il suffit de la transcrire en langage symbolique, soit on ne connaît que quelques points, et il faudra trouver une fonction approchante (comme nous l'avons fait avec la courbe de consommation).
Il existe alors des méthodes plus ou moins complexes selon la précision requise.
L'interpolation linéaire, ou encore régression linéaire, qui suppose une liaison rectiligne entre les points connus, est la plus simple de ces techniques.
Toute corrélation entre deux (ou plusieurs) variables peut être exprimée sous la forme d'une fonction.
Leur symbolisation est $\mathbf{y = f(x)}$, mais on pourrait très bien remplacer x et y par d'autres caractères.
Tout langage de programmation possède des fonctions intrinsèques et un symbolisme propre, mais on retrouve la plupart du temps cette structure :

$$\begin{matrix}\text{variable}\\ \text{indépendante}\end{matrix} = \text{NOM} \begin{matrix}\text{variable}\\ \text{dépendante}\end{matrix}$$

où NOM est le « mnémonique » correspondant à la fonction (sous la forme d'un sigle ou d'une abréviation).

Aujourd'hui, les possibilités graphiques des micro-ordinateurs personnels permettent une représentation claire de fonctions complexes. Ici : un histogramme en trois dimensions.



Olivetti

Les fonctions du Basic

Le Basic est un des langages les plus riches en fonctions prédéfinies. Destinées pour une bonne part au traitement des chaînes de caractères, ces fonctions facilitent considérablement le travail du programmeur qui les maîtrise bien. Nous vous conseillons donc une lecture attentive de ce chapitre.
Les fonctions du Basic peuvent être classées en trois groupes principaux :

- les fonctions mathématiques ;
- les fonctions permettant le traitement des chaînes de caractères ;
- les fonctions spéciales (mémoire et E/S).

L'application de certaines fonctions mathématiques nécessite des notions de trigonométrie et de calcul logarithmique. Ceci concerne surtout les applications scientifiques et nous ne présenterons ici ces instructions qu'à titre d'information. Les lecteurs intéressés par ce sujet pourrons utilement se référer à un manuel d'analyse et de trigonométrie.

Les fonctions mathématiques

Ces fonctions renvoient un résultat numérique (Y) après traitement de l'argument proposé (X). Elles seront présentées par ordre de difficulté croissante afin de faciliter la lecture de ce chapitre.
Les premières fonctions permettent de modifier la présentation d'un nombre (arrondi, extraction de la partie entière, etc.). Viennent ensuite les fonctions mathématiques courantes suivies des transformations trigonométriques et exponentielles. L'argument à communiquer est noté N ou R selon qu'il s'agit d'un entier ou d'un réel. Cette distinction est purement formelle car le Basic effectue une conversion automatique en cas d'erreur.
Enfin, contrairement au Basic compilé, le Basic interprété n'accepte généralement pas les arguments en double précision.

ABS(R). Cette fonction renvoie la valeur absolue (c'est-à-dire positive) de R. L'argument peut être une expression arithmétique. Ainsi :

```
Y = ABS (-10)
Z = ABS(3*(7-5)-2)
```

sont deux écritures correctes de cette instruction. La première affecte à Y la valeur 10 et dans la seconde, on donne à Z la valeur 4. La fonction ABS(R) s'avère utile avant l'application d'instructions dont l'argument doit être positif.
On pourrait, bien sûr, arriver au même résultat en testant le signe du paramètre et en le multipliant par –1 s'il est négatif. Le nombre obtenu serait alors toujours positif. Il est cependant plus rapide d'utiliser l'instruction ABS(R).
On peut aussi déterminer si une variable X est comprise dans l'intervalle [–0.01, 0.01] par cette instruction :

```
XI = ABS(X)  'XI est la valeur absolue de X
IF  XI < = 0.01 THEN ...
```

ou, plus directement :

```
IF ABS(X) < = 0.01 THEN ...
```

CDBL(R). Convertit R en un nombre en double précision. Exemple :

```
10  R = 56.8
20  DR = CDBL(R)
```

La valeur de R est affectée à une variable DR de format double précision. Ce format réserve pour la variable deux fois plus de place en mémoire, et autorise ainsi l'utilisation de beaucoup plus de chiffres significatifs.
On emploie cette instruction avant les calculs comportant à la fois des variables en double et en simple précision. On a alors un format homogène, ce qui limite les risques d'erreur.

CINT(R). Transforme un réel en entier avec arrondissement de sa partie décimale.
Le résultat étant un entier doit obligatoirement être compris entre –32768 et +32767, sous peine d'un dépassement de capacité (« overflow error »).
Ainsi :

```
A = CINT(7.51) et
B = CINT(3574.2)
```

sont correctement formulés alors que :

```
C = CINT(82754.3)
```

provoquera une interruption du programme.

L'instruction CINT est donc utilisable dans l'intervalle des nombres entiers. Au-delà, on a recours à des méthodes plus complexes que nous étudierons plus loin.

CSNG(R). C'est la fonction inverse de CDBL(R) ; elle produit le passage de double en simple précision. Exemple :

```
R = 135.7943 'constante en double précision
M = CSNG(R) 'affecte à M la valeur de R en simple précision
```

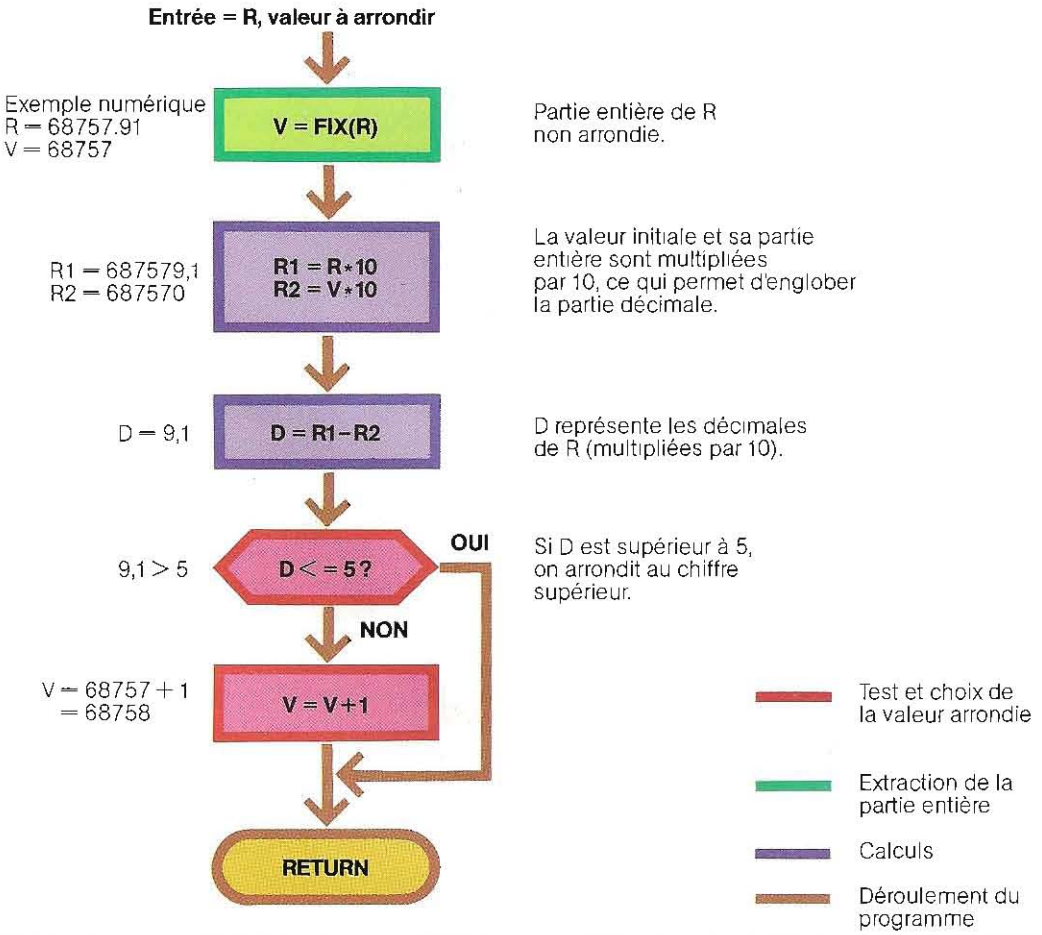
On utilise la fonction CSNG(R) pour alléger l'espace mémoire, ou pour rendre homogènes les facteurs d'un calcul.

FIX(R). Tronque la valeur de R à sa partie entière. Ainsi :

M = FIX(572.85) donne M = 572
M = FIX(–374.6) donne M = –374

Quoique sans décimales, la valeur rénvoyée est un réel. On peut donc utiliser cette fonction en dehors de l'intervalle des entiers (–32768, +32768), et notamment créer un équivalent de la fonction CINT hors de cet intervalle.
Il suffit pour cela d'utiliser l'instruction FIX qui donne la partie entière du nombre, puis d'examiner la partie décimale de l'argument pour savoir si l'on arrondit, ou non, au chiffre supérieur. L'organigramme de cette routine est présenté ci-dessous et son listage page 442. Elle n'est applicable qu'aux nombres positifs,

ORGANIGRAMME DU SOUS-PROGRAMME POUR L'ARRONDI DES NOMBRES



PROGRAMME POUR L'ARRONDI DES NOMBRES

```
10 A$ = "$£££££££££.£"
20 INPUT "VALEUR A ARRONDIR";R
30 '
40 GOSUB 1000
50 '
60 LPRINT "VALEUR A ARRONDIR R = ";:LPRINT USING A$;R
70 LPRINT "VALEUR ARRONDIE    V = ";:LPRINT USING A$;V
80 LPRINT
90 GOTO 20
100 END
1000 ' * SOUS-PROGRAMME POUR L'ARRONDI *
1010 V = FIX(R)     ' PARTIE ENTIERE DE R NON ARRONDIE
1020 R1 = R * 10
1030 R2 = V * 10
1040 '
1050 D = R1 - R2     ' D CONTIENT LES DECIMALES DE R
1060 IF D <= 5 THEN RETURN ELSE V = V + 1 : RETURN
```

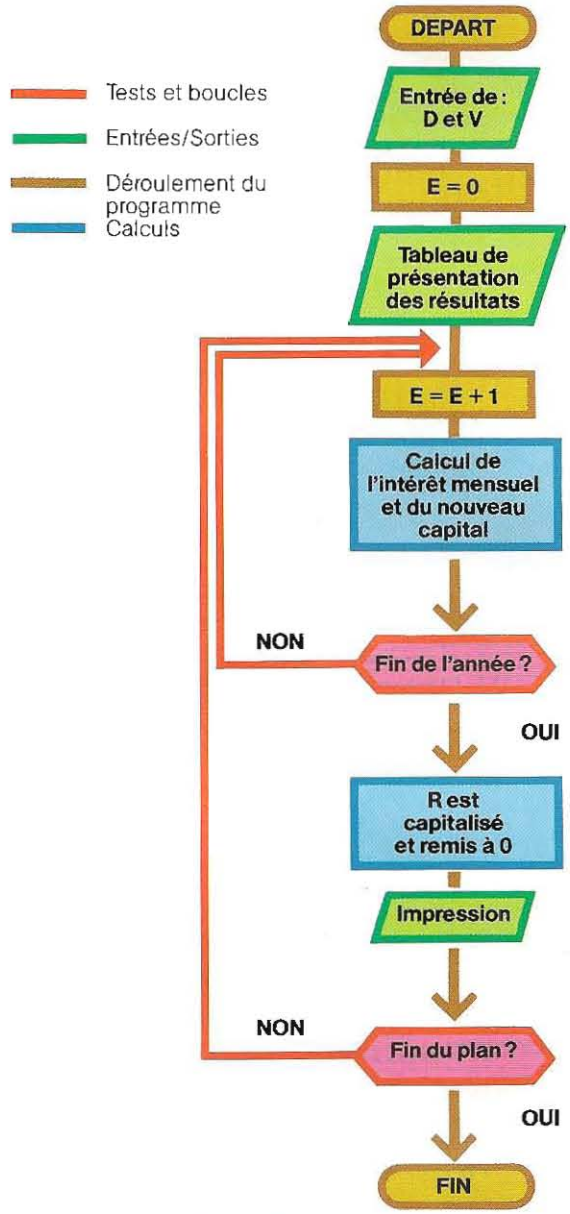
```
VALEUR A ARRONDIR R =          123.5
VALEUR ARRONDIE   V =          123.0

VALEUR A ARRONDIR R =          123.6
VALEUR ARRONDIE   V =          124.0
```

PROGRAMME DE SIMULATION D'UN PLAN D'EPARGNE-LOGEMENT

```
70 A$ = "£££££££££"    ' MASQUE D'IMPRESSION (CETTE INSTRUCTION
80 '                   SERA EXPLIQUEE PAR LA SUITE)
90 INPUT "DEPOT INITIAL:";D
100 C = D    ' CAPITAL EGAL AU DEPOT
110 INPUT "VERSEMENTS MENSUELS:";V
120 E = 0    ' INITIALISATION DE LA BOUCLE : ECHEANCE ZERO
140 ' PRESENTATION DU TABLEAU SUR L'IMPRIMANTE
150 LPRINT "EVOLUTION ANNUELLE DU CAPITAL SUR UN PLAN D'EPARGNE-LOGEMENT"
160 LPRINT "    (TAUX : 10 % SUR 5 ANS)
170 LPRINT
180 LPRINT "AU TERME DE:", "CAPITAL", "SOMMES VERSEES"
200 LPRINT
210 E = E + 1        ' COMPTEUR DES ECHEANCES
230 I = INT(C * 0.1/12)    ' INTERET MENSUEL
240 R = R + I        ' LA REMUNERATION AUGMENTE
260 C = C + V        ' VERSEMENT MENSUEL
270 IF INT(E/12) <> E/12 GOTO 210
280 C = C + R      ' REMUNERATION CAPITALISEE TOUS LES ANS
290 R = 0
300 ' IMPRESSION DES RESULTATS
310 ' *** L'INSTRUCTION "LPRINT USING " SERA EXPLIQUEE ULTERIEUREMENT ***
320 LPRINT "ANNEE ";E/12,:LPRINT USING A$;C,:LPRINT USING A$;D + E * V
330 IF E < 60 GOTO 210
360 END
```

```
EVOLUTION ANNUELLE DU CAPITAL SUR UN PLAN D'EPARGNE-LOGEMENT
    (TAUX : 10 % SUR 5 ANS)

AU TERME DE :        CAPITAL        SOMMES VERSEES

ANNEE : 1             23540             22000
ANNEE : 2             30444             34000
ANNEE : 3             54834             46000
ANNEE : 4             72860             58000
ANNEE : 5             92690             70000
```

PROGRAMME DE SIMULATION D'UN PLAN EPARGNE-LOGEMENT



mais il est très simple de l'étendre aux nombres négatifs. Rappelons en effet que la valeur arrondie de −374.6 n'est pas −375 mais −374 et qu'il convient dans ce cas d'inverser la logique du programme.

INT(R). Renvoie l'entier immédiatement inférieur à R. Ainsi :

N = INT (76.93) donne N = 76

N = INT (−21.2) donne N = −22 (et non −21)
N = INT (−21.0) donne N = −21

A titre d'illustration, nous vous proposons un programme simulant un plan d'épargne-logement. L'évolution du capital sera calculée annuellement en fonction du dépôt initial et des versements mensuels.
L'instruction INT est utilisée lors du calcul des

intérêts (arrondissement au franc inférieur) et pour déterminer la fin d'une année.
Chaque mois sont évalués les intérêts du capital précédent, puis le versement mensuel vient s'ajouter à celui-ci.
Au bout d'un an, les intérêts sont capitalisés et l'ordinateur indique la nouvelle valeur du placement.
En réalité, les intérêts sont intégrés au capital le 31 décembre et non un an après ouverture du plan.
Cette simplification n'induit toutefois qu'une erreur minime : une centaine de francs en plus (pour les valeurs de notre exemple) si le plan a été (par ex.) ouvert en juin.
Vous trouverez l'organigramme et le listage de cette application aux pages 443 et 442.

SNG(R). Renseigne sur le signe de R par un code égal à 1 si R est positif, à 0 si R est nul et à −1 si R est négatif. Ainsi :

Y = SNG(28) donne Y = 1
Y = SNG(0) donne Y = 0
Y = SNG(−75) donne Y = −1

SQR(R). Renvoie la racine carrée de R. Ainsi :

Y = SQR(16) donne Y = 4
Y = SQR(15) donne Y = 3.87298

Le nombre R doit être supérieur ou égal à zéro car les nombres négatifs n'ont pas de racine carrée dans l'ensemble des réels.

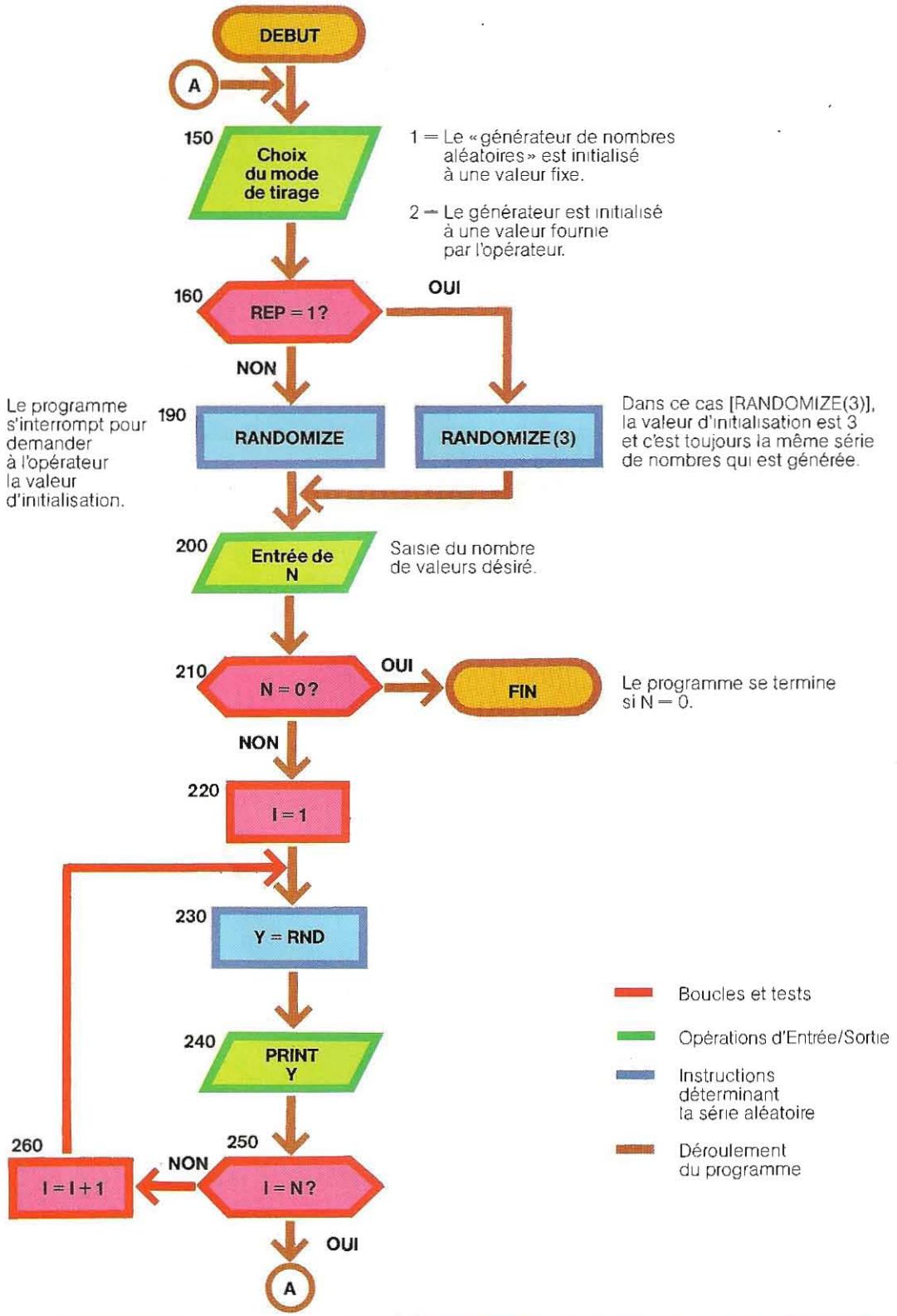
RND(R). Cette fonction génère un nombre aléatoire (random) compris entre 0 et 1.
La notion de nombre aléatoire est assez complexe, aussi renvoyons-nous aux manuels de statistiques le lecteur qui souhaite en étudier une définition plus détaillée.
Du point de vue de l'utilisateur, la fonction RND(R) active un algorithme dont l'application répétée engendre une série de nombres tirés « au hasard ». La probabilité d'obtenir deux fois le même chiffre est donc très faible.
Le tirage de nombres aléatoires se déroule en deux étapes. Un générateur de nombres aléatoires est d'abord initialisé par l'instruction RANDOMIZE(N) (avec N entier).
Cette valeur N sera ensuite utilisée comme

PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES

```
50 BI$=CHR$(27)+"*"+CHR$(7)   ' NETTOYAGE D'ECRAN ET EMISSION D'UN SON
60 PRINT BI$
70 PRINT "*** PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES ***"
80 PRINT
90 PRINT "       1) = GENERATEUR DE NOMBRES ALEATOIRES EST "
100 PRINT "            INITIALISE AUTOMATIQUEMENT "
110 PRINT
120 PRINT "       2) = GENERATEUR DE NOMBRES ALEATOIRES EST "
130 PRINT "            INITIALISE MANUELLEMENT "
140 PRINT
150 INPUT "TAPEZ VOTRE SELECTION";REP
160 IF REP=1 THEN RANDOMIZE(3) : GOTO 200
180 PRINT BI$
190 RANDOMIZE     ' VALEUR D'INITIALISATION DEMANDEE A L'OPERATEUR
200 INPUT " COMBIEN DE NOMBRES ALEATOIRES DESIREZ-VOUS?";N
210 IF N=0 THEN STOP
215 LPRINT " PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES "
220 I=1
230 Y=RND
240 LPRINT Y
250 IF I=N THEN LPRINT : GOTO 60
260 I=I+1
270 GOTO 230
280 END
```

```
 PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES

  .88588
  .484668
  .806328
```

GENERATION DE NOMBRES ALEATOIRES



argument pour la génération d'une série de nombres à chaque appel de RND(R).
Si l'on omet RANDOMIZE, l'initialisation donnera toujours le même résultat et les séries successives seront identiques.
La seconde étape est le processus de génération proprement dit lancé par RND.
Cette instruction s'emploie telle quelle ou avec un argument sous la forme RND(R). Tout argument non nul induira le tirage d'une nouvelle série ; par contre la série précédente sera répétée si R = 0. L'instruction RANDOMIZE peut aussi être employée sans argument ; l'ordinateur interrompra simplement l'exécution pour le demander à l'écran.
Le listing de la page 444 reproduit un programme de génération de nombres aléa-

Test 13



1 / Une ligne de programme comporte le calcul suivant : A=B/(C−D) ; cette instruction peut être à l'origine d'une interruption de programme. Pourquoi ?

2 / Comment peut-on prévenir cette erreur ?

3 / Certaines de ces instructions sont incorrectes. Lesquelles ?
a) IF K = 2.5 THEN 300
b) IF L+M = 6 GOTO 150
c) IF L = 5 RETURN
d) A=−5:B=2:K=A−B:ON K GOTO 10,20,30

4 / Imaginer un programme qui recherche si la variable V est comprise entre 2.6 et 15.8 (dans les deux cas : bornes inclues et bornes exclues).

5 / Quelles valeurs sont affectées aux variables E et F après l'exécution du programme suivant ?

```
10 A=-18.75
20 B=20
30 C=FIX(A)
40 D=INT(A)
50 E=B+C
60 F=B+D
```

6 / Imaginer un programme qui contrôle le signe de la variable V, puis qui présente le résultat sous la forme d'une phrase du type : supérieure à 0, inférieure à 0, égale à 0.

7 / Quelle instruction doit-on ajouter à ce programme pour éviter une erreur ?

```
10 DEFINT A-Z
20 B=-9
30 C=SQR(B)
```

Les solutions du test se trouvent page 451.

toires, correspondant à l'organigramme schématisé page 445.

ATN(Y). L'argument de cette fonction, quoique réel, est symbolisé par Y pour éviter la confusion avec la fonction TAN présentée plus loin.
ATN(Y) fournit l'angle trigonométrique dont la tangente est Y (fonction arctangente).
La valeur renvoyée est donc exprimée en radians et comprise entre $-\pi/2$ et $+\pi/2$.
Le symbole π désigne le rapport entre la circonférence d'un cercle et son diamètre : π = circonférence/diamètre = 3.1415926...
Y peut être donné en double précision mais le résultat est presque toujours en simple précision.

Les fonctions trigonométriques

COS(R). C'est la fonction cosinus. Elle renvoie le cosinus de R qui doit être exprimé en radians. Le résultat est également en simple précision.
L'unité angulaire généralement employée est le degré. On peut toutefois convertir les degrés en radians grâce à cette formule :

$$\text{Angle en radians} = \frac{\pi \times \text{angle en degrés}}{180} = 0.01745 \times \text{angle en degrés}$$

SIN(R). C'est la fonction sinus. Elle calcule le sinus de R qui doit là aussi être exprimé en radians.

TAN(R). C'est la fonction tangente. Elle calcule la tangente de R, toujours exprimé en radians. Notons que les fonctions TAN(R) et ATN(R) sont réciproques l'une de l'autre : TAN(R) donne la tangente de l'angle R, et ATN(Y) donne l'angle dont Y est la tangente. Ou, de façon plus concise :

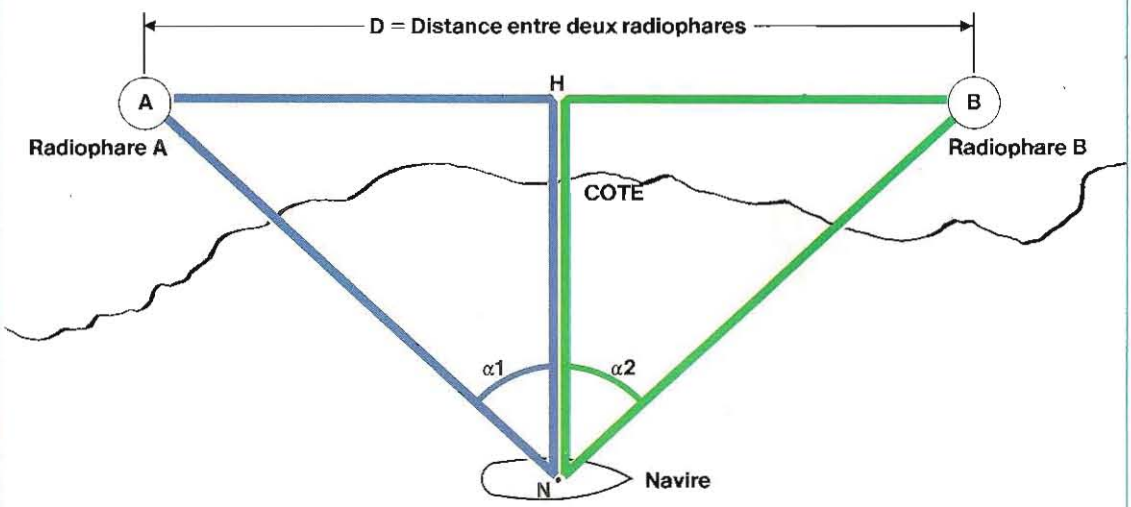
Y = TAN(R)
Y est la valeur de la tangente de l'angle R.
R = ATN(Y)
R est la mesure de l'angle qui a Y pour tangente.

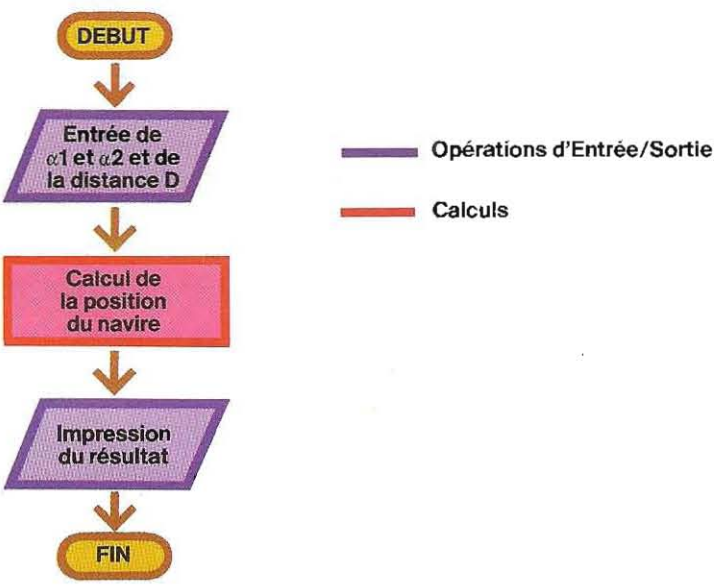
Nous vous présentons ci-dessous le listage et l'exécution d'un programme de démonstration des fonctions trigonométriques.

EXEMPLE D'UTILISATION DES FONCTIONS TRIGONOMETRIQUES

```
40 ' RD = ANGLE EXPRIME EN DEGRES
50 '
60 ' RR = MEME ANGLE CONVERTI EN RADIANS
70 '
100 INPUT "ANGLE (EN DEGRES)";RD
110 IF RD=90 GOTO 1000   ' COS(90°)=0 ; ERREUR EN LIGNE 225
120 IF RD>180 GOTO 1000
130 '
140 ' * CONVERSION EN RADIANS *
150 '
160 RR=.01745*RD
170 '
180 '
190 ' * FONCTIONS TRIGONOMETRIQUES *
200 '
210 S=SIN(RR)        ' SINUS
220 C=COS(RR)        ' COSINUS
230 T=S/C            ' TANGENTE = SINUS/COSINUS
230 A=ATN(T)         ' ARCTANGENTE ; DOIT CORRESPONDRE A RR
240 '
250 ' * IMPRESSION *
260 '
270 LPRINT "ANGLE EN DEGRES =";RD,"EQUIVALENT EN RADIANS =";RR
280 LPRINT "SINUS =";S,"COSINUS =";C,"ARCTANGENTE =";A
285 LPRINT
290 GOTO 100
300 '
1000 END

ANGLE EN DEGRES = 20          EQUIVALENT EN RADIANS = .349
SINUS = .341958       COSINUS = .939715       ARCTANGENTE = .349
```

APPLICATION DES FONCTIONS TRIGONOMETRIQUES A UN PROBLEME DE TRIANGULATION





Applications des fonctions trigonométriques. Les fonctions trigonométriques sont indispensables pour résoudre les problèmes de triangulation ; et notamment dans toute évaluation de distance, d'altitude, etc., à partir de simples mesures d'angle. C'est, par exemple, la méthode employée pour déterminer la position d'un navire à l'aide des signaux émis par deux balises radio.

Dans la réalité, les positions des radiophares et du navire sont caractérisées par leur latitude et leur longitude dans le système géodésique des méridiens et des parallèles. Le calcul se trouve ainsi simplifié. Notre exemple n'ayant qu'une valeur d'illustration, nous négligerons ces possibilités et nous nous contenterons de calculer la distance du bateau à la côte.

Le radiogoniomètre, récepteur spécial installé à bord du bateau, permet de déterminer les angles des signaux émis par les radiophares ($\alpha 1$ et $\alpha 2$ sur le schéma ci-dessus).

On connaît ces deux angles et la distance D séparant les radiophares. Il est alors possible de trouver la distance du bateau à la côte (NH) par des calculs simples sur les triangles rectangles NHA (en bleu) et NHB (en vert).
Voici la formule à employer :

$$NH = \text{distance de la côte} = \frac{\text{distance entre les radiophares}}{tg(\alpha_1) + tg(\alpha_2)}$$

Les expressions $tg(\alpha_1)$ et $tg(\alpha_2)$ désignent les tangentes des angles α_1 et α_2.

Le programme devra donc exécuter les opérations suivantes :

1 / entrée des angles α_1 et α_2 et de la distance D ;
2 / conversion des angles en radians ;
3 / calcul de la tangente de chaque angle ;
4 / calcul de la distance du bateau à la côte ;
5 / présentation du résultat.

Le listage de ce programme se trouve ci-dessous.

Dans les cas où la fonction tangente n'est pas disponible, il est très facile de la déduire de cette formule :

$$tg(R) = \frac{SIN(R)}{COS(R)}$$

Cet exemple montre comment contourner les carences de certains BASIC simplifiés. Quand ne sont disponibles que les trois fonctions sinus, cosinus et cotangente, c'est en effet au programmeur de définir lui-même les autres. Nous reviendrons sur ce point en fin de chapitre.

EXP(R). Calcule la valeur exponentielle de R. En d'autres termes, l'instruction :

Y = EXP(R)
affecte à Y la valeur de e^R. Ainsi, pour R = 2, on a :

Y = EXP(2), $Y = e^2 = 2.7183^2 = 7.389...$

Rappelons que le nombre e de valeur 2.7183 est la base des logarithmes népériens.

CALCUL DE LA DISTANCE D'UN NAVIRE A LA COTE

```
30 PRINT "N.B. On doit entrer les angles en degrés"
40 '
50 INPUT "ALPHA 1";I
55 A=I  ' MEMORISATION DANS "A" DE LA VALEUR DE ALPHA 1 EN DEGRES
60 INPUT "ALPHA 2";II
65 B=II ' MEMORISATION DANS "B" DE LA VALEUR DE ALPHA 2 EN DEGRES
70 INPUT "DISTANCE ENTRE LES RADIOPHARES";D
80 ' CONVERSION DES DEGRES EN RADIANS
90 I=I*.01745
100 II=II*.01745
110 '
120 ' CALCUL DE LA DISTANCE NH=D/(TAN(I)+TAN(II))
130 NH=D/(TAN(I)+TAN(II))
140 '
150 '
160 ' * INSTRUCTIONS D'IMPRESSION *
170 LPRINT "DISTANCE ENTRE LES RADIOPHARES     ";D
180 LPRINT "ANGLE ALPHA 1 = ";A
190 LPRINT "ANGLE ALPHA 2 = ";B
200 LPRINT "N.B. LES ANGLES SONT EXPRIMES EN DEGRES"
210 LPRINT              ' SAUT DE LIGNE
220 LPRINT "DISTANCE DU NAVIRE A LA COTE NH = ";NH
230 LPRINT:LPRINT       ' DOUBLE SAUT DE LIGNE
240 GOTO 50
250 END
```

```
DISTANCE ENTRE LES RADIOPHARES     1455
ANGLE ALPHA 1 = 12,1555
ANGLE ALPHA 2 = 39,4546
N.B. LES ANGLES SONT EXPRIMES EN DEGRES

DISTANCE DU NAVIRE A LA COTE NH = 1401,54
```

Cette fonction est bien entendu réservée à des applications scientifiques. Elle est présentée de façon détaillée dans les manuels d'analyse mathématique.

LOG(R). Renvoie la valeur du logarithme naturel (ou népérien) de R.
Nous ne nous attarderons pas sur la signification du terme logarithme. Rappelons simplement que R doit être strictement positif.

Fonctions permettant le traitement des chaînes de caractères

Une particularité du Basic est sa vaste gamme de commandes concernant les chaînes de caractères. L'emploi de ces instructions procure une grande souplesse dans la présentation des tableaux et facilite la rédaction des traitements de fichiers. Le nombre et la syntaxe de ces commandes varient selon les différentes versions de Basic. C'est pourquoi nous nous limiterons, dans ce chapitre, à étudier les fonctions les plus courantes du Basic 80.
Nous aborderons par la suite certaines structures ou fonctions, absentes de cette version standard, et qui permettent, entre autres, la manipulation de morceaux de chaîne. Rappelons pour finir que certains Basic n'autorisent le transfert sur disque que pour des données en ASCII. On doit alors convertir en caractères toutes les valeurs numériques avant de les sauvegarder et effectuer la conversion inverse (d'ASCII en numérique) après la relecture. Les méthodes de conversion sont développées à la fin de ce chapitre.

ASC(A$). Fournit la valeur numérique (selon le code ASCII) correspondant au premier caractère de la chaîne A$. Ainsi :

```
10  A$ = "17ABC"
(la chaîne A$ contient les caractères 1, 7, A, B et C)
20  X = ASC(A$)
(le code du premier caractère est affecté à X)
30  PRINT X
      49
(la valeur de X s'affiche)
```

Ici, la fonction X = ASC(A$) a affecté à X la valeur 49 qui est le code ASCII du caractère 1 (les symboles numériques contenus dans une chaîne sont traités comme des caractères). On trouvera au bas de cette page le listing du programme correspondant.

CHR$(N). Crée une chaîne d'un seul caractère (celui dont N est le code en ASCII).
Ainsi, par exemple :

```
10  N = 65
20  B$ = CHR$(N)
30  PRINT B$
      A
```

65 est le code de la lettre A. Vous trouverez page 452 le listing d'un programme qui associe à une donnée N le caractère correspondant. Ce programme vérifie que N correspond à un code existant. Il imprime ensuite le caractère concerné en majuscule et en minuscule si on le lui a précisé.

EXEMPLE D'UTILISATION DE L'INSTRUCTION ASC(A$)

```
10 ' **EXEMPLES D'UTILISATION DE L'INSTRUCTION ASC(A$) **
15 '       FICHIER = INSTASC
20 DEFINT A-Z       ' TOUTES LES VARIABLES SONT DEFINIES COMME DES ENTIERS
30 INPUT "ENTREZ UNE CHAINE QUELCONQUE";A$
40 IF A$ = "FIN" GOTO 100     ' ON ARRETE LE PROGRAMME EN ENTRANT LE MOT FIN
50 N = ASC(A$)        ' EXTRACTION DU CODE DE PREMIER CARACTERE
60 LPRINT A$,N        ' IMPRESSION DE LA CHAINE ET DU CODE
80 LPRINT             ' SAUT DE LIGNE POUR AERER LE TEXTE
90 GOTO 30
100 END


ABCDEFGT          65

BCFGGHYK          66

CANNES            67

A                 65
```

Solutions du test 13

1-2 / L'erreur se produit quand C = D. La différence C – D devient alors nulle et la division par zéro est impossible. On peut éliminer cette erreur en « sautant » le calcul quand C = D. On peut écrire le programme de cette façon :

```
210  IF C=D GOTO 2000
220  A=B/(C-D)
225  ' * Point de rebranchement *
... suite des instructions du programme ...
2000 ' * * Erreur * *
2010 PRINT "Erreur : division par zéro"
2020 INPUT "Tapez 1 pour continuer";N
2030 IF N=1 GOTO 225
2040 STOP
```

Lorsque la condition C = D est vérifiée, le programme est détourné ligne 2000, affiche le message d'erreur (2010) et demande s'il faut continuer (2020). Si la réponse est affirmative (N = 1, 2030), il retourne exécuter les instructions 225 et suivantes. Le calcul de la ligne 220 n'est donc pas effectué et le programme ne s'interrompt pas.

---

3 / Les instructions c) et d) sont erronées.

La ligne c) est une instruction de retour conditionnel. Sa syntaxe est IF ... THEN ... THEN ne peut être omis que s'il est suivi de GOTO. L'écriture correcte est donc : IF L = 5 THEN RETURN.

A la ligne d), on veut utiliser la valeur de K pour déterminer le branchement à l'instruction 10, 20 ou 30. Or, l'instruction ON n'accepte que des paramètres positifs. De plus, cette ligne est totalement illogique puisque K est obtenu par la différence de deux constantes (A et B) et est donc elle-même une constante.

---

4 / Il faut vérifier que V est à la fois (AND) supérieur à 2.6 et inférieur à 15.8 :

```
IF 2.6<V AND V<15.8 THEN PRINT "Inclus"
```

Cette instruction exclut les bornes de l'intervalle. Pour les inclure, il suffit d'écrire :

```
IF 2.6<=V AND V<=15.8 THEN PRINT "Inclus"
```

---

5 / L'instruction C = FIX(A) supprime les décimales de A et donne donc C = –18. La ligne 40 affecte à D le premier entier inférieur à A et donc –19 (attention au signe : –18 est supérieur à –18.75).

En conséquence : E = 20 – 18 = 2 F = 20 – 19 = 1

---

6 / Il faut utiliser l'instruction SNG(V). On peut rédiger ainsi le programme :

```
10 K=SNG(V)
20 IF K=-1 THEN PRINT "Inférieur à zéro"
30 IF K=0 THEN PRINT "Egal à zéro"
40 IF K=1 THEN PRINT "Supérieur à zéro"
```

---

7 / Un nombre négatif n'a pas de racine carrée dans l'ensemble des réels. Il faut donc ajouter une instruction qui rende B positif quelle que soit sa valeur. On pourra intercaler cette ligne :

```
25 B=ABS(B)
```

PROGRAMME D'UTILISATION DE L'INSTRUCTION CHR$

```
10 ' ** PROGRAMME D'UTILISATION DE L'INSTRUCTION CHR$ **
15 '    FICHIER = INSTCHR
17 BI$=CHR$(27)+"+"+CHR$(7)
18 PRINT BI$        ' VIDE L'ECRAN ET EMET UN BIP
20 PRINT "DESIREZ-VOUS UNE TRANSCRIPTION EN MINUSCULE (OUI OU NON) ?"
25 '
30 INPUT REP$
40 IF REP$<>"OUI" AND REP$<>"NON" THEN GOTO 300
45 ' DEBUT DE LA BOUCLE
46 PRINT BI$
47 PRINT "TAPER (0) POUR ARRETER"
50 INPUT "NOMBRE A CONVERTIR EN CARACTERE";N
60 IF N=0 GOTO 1000
70 IF 47<N AND N<91 GOTO 200
80 IF 96<N AND N<123 GOTO 90
85 PRINT "ERREUR : CE CODE NE CORRESPOND A AUCUN CARACTERE":GOTO 50
90 C$ = CHR$(N)
100 LPRINT "VALEUR NUMERIQUE ";N,C$
110 GOTO 45
200 C$ = CHR$(N)
205 IF 47<N AND N<58 THEN B$="":GOTO 250
210 IF REP$="OUI" THEN GOTO 220 ELSE B$="":GOTO 230
220 B$=CHR$(N+32)
230 LPRINT "VALEUR NUMERIQUE ";N,B$,C$
240 GOTO 45
300 PRINT "ERREUR : VOUS DEVEZ REPONDRE PAR OUI OU PAR NON"
301 GOTO 25
1000 END
```

```
VALEUR NUMERIQUE 65          a          A
VALEUR NUMERIQUE 66          b          B
VALEUR NUMERIQUE 67          c          C
VALEUR NUMERIQUE 68          d          D
```

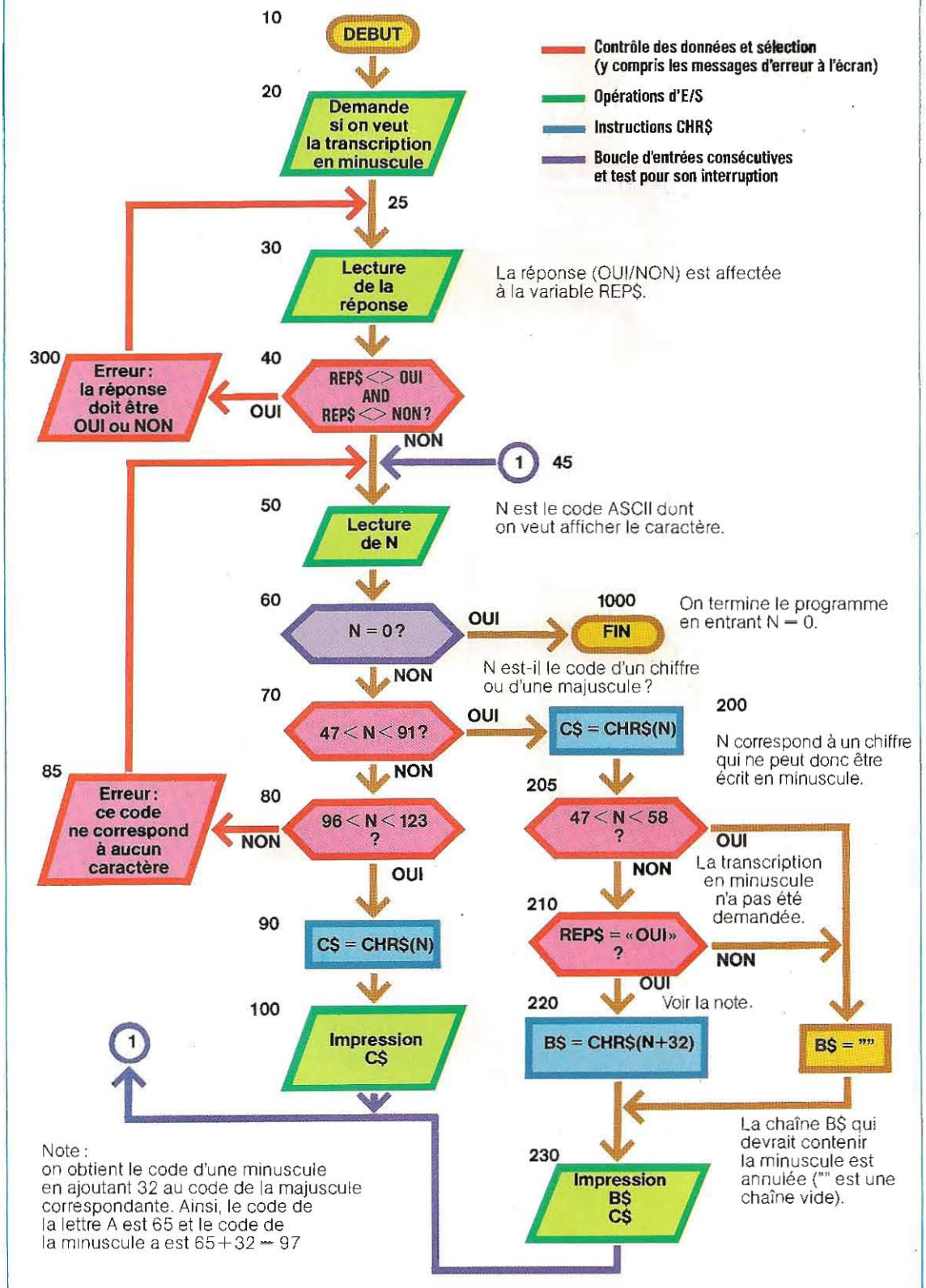
L'organigramme de ce programme se trouve page 453.

On utilise surtout l'instruction CHR$ pour transmettre des caractères de contrôle à l'écran et à l'imprimante. Ainsi, par exemple, la chaîne A$ = CHR$(12) contient un CTRL-L ou FF. La sortie de cette chaîne sur l'imprimante n'écrira rien mais produira un saut de page. Nous reviendrons plus en détail sur ce sujet par la suite.

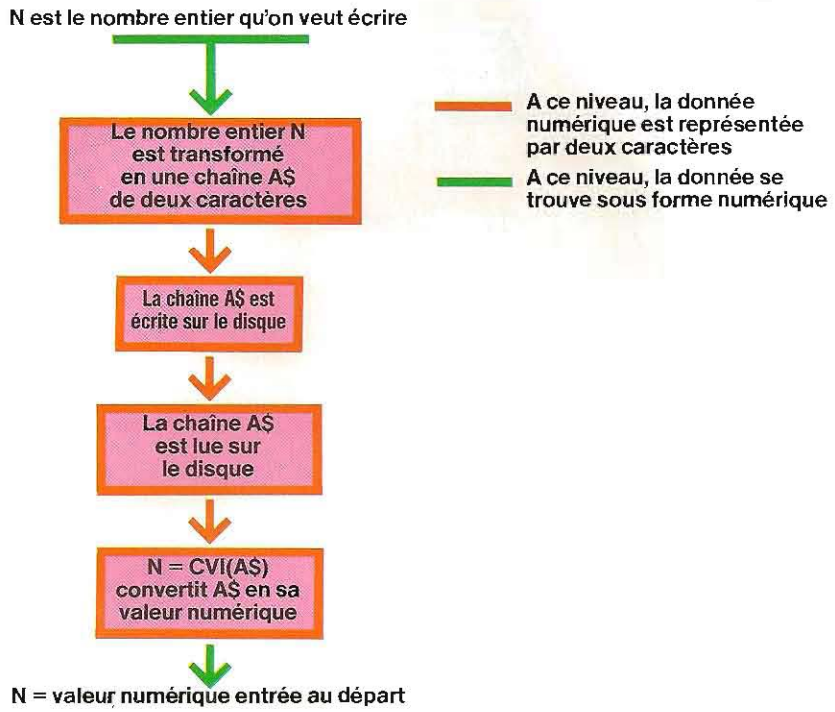
CVI(A$). Convertit une chaîne de deux caractères en son équivalent numérique entier (2 octets). On l'utilise après la lecture de données sur un disque où elles étaient mémorisées sous forme de chaînes. Cette instruction les transforme alors en valeurs numériques et elles devront être retraduites avant sauvegarde.

On trouvera page 454 le schéma décrivant les opérations d'écriture et de lecture de données numériques sur disque dans ce cas.

EMPLOI DE L'INSTRUCTION CHR$



SCHEMA LOGIQUE DE L'ECRITURE ET DE LA LECTURE D'UNE DONNEE NUMERIQUE (ENTIERE) SUR UN DISQUE



CV$(A$). Convertit une chaîne de quatre caractères (4 octets, soit 32 bits) en son équivalent numérique, qui est un réel en simple précision.

CVD(A$). Convertit une chaîne de huit caractères en un réel en double précision.
L'emploi des fonctions CVI, CVS et CVD, ainsi que des fonctions inverses (qui convertissent des réels en chaînes de caractères) est décrit de façon plus détaillée dans le chapitre consacré à la sauvegarde des données sur disque.
Signalons toutefois que ce type de mémorisation, outre ses contraintes, occupe beaucoup plus d'espace qu'une représentation binaire.
Prenons, par exemple, le nombre 12745.
Son écriture en binaire n'utilise que 2 octets (soit 16 bits : 0011000111001001) alors que sa représentation en caractères ASCII occupe 5 octets. En effet, chaque chiffre est alors considéré comme un caractère et mémorisé sur un octet (la chaîne qui représente 12475 est constituée des codes 49, 50, 55, 52 et 53, comme on peut le voir sur la table du code ASCII, page 114).

HEX$(N). Donne une chaîne qui est la représentation hexadécimale (en base 16) de l'entier N. Exemple :

```
10  DEFINT N
20  N = 21
30  E$ = HEX$(N)
```

Ces instructions affectent à la chaîne E$ l'équivalent en base 16 du nombre 21 : soit "15" (1 fois 16 et 5 unités).
Page 455 est listé un programme de démonstration en rapport avec cette instruction et quelques résultats de son exécution.
Cette fonction ne travaille que sur des entiers, mais accepte des réels en argument.
Ceux-ci seront alors convertis automatiquement en entier (ce mécanisme est commun à la plupart des fonctions travaillant sur des

entiers).

INKEY$. Commande la saisie d'un caractère au clavier. C'est donc une instruction d'Entrée/Sortie, et nous étudierons ces fonctions plus en détail dans le chapitre consacré à ce sujet.
Le caractère saisi doit être affecté à une chaîne et la syntaxe exacte est :

C$ = INKEY$

Arrivé à cette instruction, le système indique, par un curseur clignotant, qu'il attend une entrée et suspend l'exécution du programme jusqu'à la frappe d'une touche.

INPUT$(N). Cette fonction est identique à la précédente mais concerne une chaîne de N caractères. Exemple :

C$ = INPUT$(5)

Cette instruction prend les cinq premiers caractères entrés au clavier et les transfère dans la chaîne C$.
On remarquera que INKEY$ n'est qu'un cas particulier de la fonction INPUT$ (c'est le cas où N = 1).

INSTR(N, A$, B$). Permet de déterminer si la chaîne B$ est contenue dans A$.
Le résultat fourni est une valeur numérique (comprise entre 1 et 255) correspondant à la première position où l'on trouve B$.
La valeur 0 indique que la chaîne B$ n'a pas été rencontrée.
Le paramètre N permet de commencer la recherche au N^{e} caractère de A$. S'il est omis, A$ sera inspectée à partir du début.
Les instructions suivantes :

```
10 A$ = "PRENOM INCONNU"
20 B$ = "O"
30 K = INSTR(A$, B$)
```

fixent K à 5 puisque la chaîne B$ (uniquement constituée du caractère "O") a été trouvée à la 5e position (5e caractère) de la chaîne A$.
Attention, les espaces sont considérés comme des caractères. Ainsi, la recherche de "U" aurait affecté 14 à K.
Utilisons maintenant cette commande en fixant un paramètre N.
On peut, par exemple, modifier l'instruction 30 comme ceci :

K = INSTR(6, A$, B$)

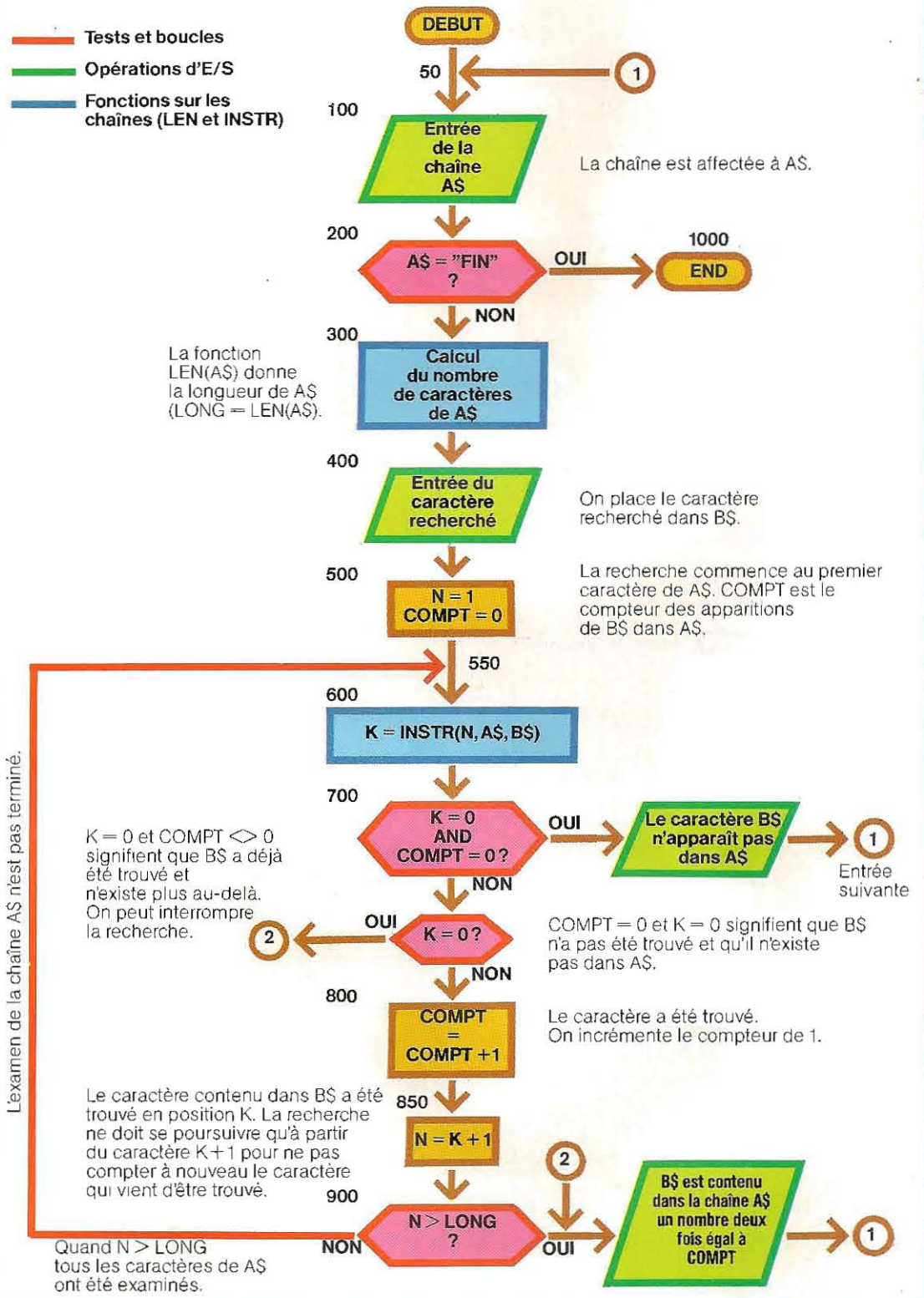
Nous aurons alors K = 11 puisque la recherche a commencé au-delà du "O" de "PRENOM".

EMPLOI DE LA FONCTION HEX$(N)

```
30 DEFINT N
40 BI$=CHR$(27)+"+"+CHR$(7)   ' NETTOYAGE D'ECRAN ET EMISSION D'UN "BIP"
50 '  PREMIER CAS : L'ARGUMENT EST UN ENTIER
60 PRINT BI$
70 PRINT "TAPER (0) POUR SORTIR DE CE PREMIER CAS"
80 INPUT "ARGUMENT : ";N    ' N est défini comme entier à la ligne 30
90 IF N=0 GOTO 130
100 A$=HEX$(N)
110 LPRINT "ARGUMENT : ";N,"VALEUR HEXADECIMALE : ";A$
120 GOTO 80
130 LPRINT
140 ' SECOND CAS : L'ARGUMENT EST UN REEL
145 ' N.B. LA FONCTION HEX$(N) CONVERTIT TOUJOURS LES ARGUMENTS EN ENTIERS
150 PRINT BI$
160 PRINT "TAPER (0) POUR ARRETER LE PROGRAMME"
170 INPUT "ARGUMENT : ";R    ' R est une variable réelle
180 IF R=0 GOTO 220
190 A$=HEX$(R)    ' CONVERTIT R EN HEXADECIMAL ET LE MET DANS A$
200 LPRINT "ARGUMENT : ";R,"VALEUR HEXADECIMALE : ";A$
210 GOTO 170
220 END
```

```
ARGUMENT :   123              VALEUR HEXADECIMALE :   7B
ARGUMENT :   456              VALEUR HEXADECIMALE :   1C8
ARGUMENT :   789              VALEUR HEXADECIMALE :   315
```

ORGANIGRAMME DE COMPTAGE DE L'APPARITION D'UN CARACTERE DANS UNE CHAINE



Page 456, se trouve l'organigramme d'un programme qui effectue les opérations suivantes : il lit une chaîne de caractères saisie au clavier, demande le caractère à rechercher et fournit, en sortie, le nombre de fois où il apparaît. Le programme correspondant, listé sur cette page, utilise la fonction LEN(A$) qui sera étudiée plus tard.

LEFT$(A$,N). Extrait, de la chaîne A$, N caractères à partir de la gauche. Par exemple, la suite d'instructions :

```
10 A$ = "MAURICE RAVEL"
20 B$ = LEFT$(A$,7)
```

affecte à B$ la chaîne "MAURICE", c'est-à-dire les 7 premiers caractères de A$ à partir de la gauche. La valeur de N doit être comprise entre 1 et 255 ; si N est nul, la chaîne B$ sera vide (B$ = "").

Les pages 458 et 459 exposent l'organigramme d'un programme utilisant l'instruction LEFT$ pour la gestion d'un fichier d'adresses.

Les enregistrements du fichier ont la structure suivante :

Champ 1 : Nom et prénom (30 caractères, dont les 20 premiers pour le nom).
Champ 2 : Ville, Numéro, Rue (40 caractères, dont les 10 premiers pour la ville).

Ce fichier, stocké sur disque, contient 1000 adresses ; on demande l'impression de tous les noms d'une ville donnée, ou bien celle de toutes les adresses des personnes de même nom (quelle que soit la ville). Le programme

DETERMINATION DU NOMBRE D'APPARITIONS D'UN CARACTERE DONNE DANS UNE CHAINE

```
1  ' *Calcul du nombre d'apparitions d'un caractère dans une chaîne*
2  ' **********************************************************
3  '
4  '
10 PRINT "POUR SORTIR, ENTREZ LA CHAINE 'SORTIE'"
20 INPUT "Entrez la chaîne de caractères : ";A$
30 IF A$ = "SORTIE" GOTO 200
40 LONG = LEN(A$)
41 '     La variable LONG contient le nombre de caractères de A$
50 INPUT "Caractère à chercher : ";B$
60 N = 1
70 COMPTE = 0
71       ' COMPTE stocke le nombre d'apparitions
72       ' du caractère B$ dans la chaîne A$
80 K = INSTR(N, A$, B$)
90 IF K=0 AND COMPTE=0 GOTO 180    ' on n'a rien trouvé
100 IF K=0 GOTO 140                ' on a fini la chaîne
110 COMPTE = COMPTE+1
120 N=K+1                          ' on entame une nouvelle recherche
130 IF N<LONG GOTO 80              ' à partir du nouveau rang N
140 LPRINT "Le caractère '";B$;"' apparaît"
150 LPRINT COMPTE; "fois dans la chaîne '";A$;"'"
160 LPRINT
170 GOTO 10
171 '
180 LPRINT "Le caractère '";B$;"' n'apparaît pas"
181 LPRINT "dans la chaîne '";A$;"'"
190 GOTO 10
200 END
```

```
Le caractère 'N' apparaît
2 fois dans la chaîne 'VINCENT'

Le caractère 'A' apparaît
2 fois dans la chaîne 'CHANTAL'

Le caractère 'E' apparaît
1 fois dans la chaîne 'ERIC'

Le caractère 'K' n'apparaît pas
dans la chaîne 'VALERIE'
```

doit prendre en compte deux possibilités :

1 / sélection par ville (sur les 10 premiers caractères du champ 2) ;
2 / sélection par nom (sur les 20 premiers caractères du champ 1).

Une partie du listing est représentée page 460 (le module de lecture des données sur disque est omis).

LEN(A$). Cette fonction fournit la longueur de la chaîne A$ (en nombre de caractères), y compris les blancs ou les éventuels caractères spéciaux non imprimables.
L'utilisation de l'instruction LEN est illustrée dans le schéma de la page 456 et en ligne 40 du listing de la page 457.

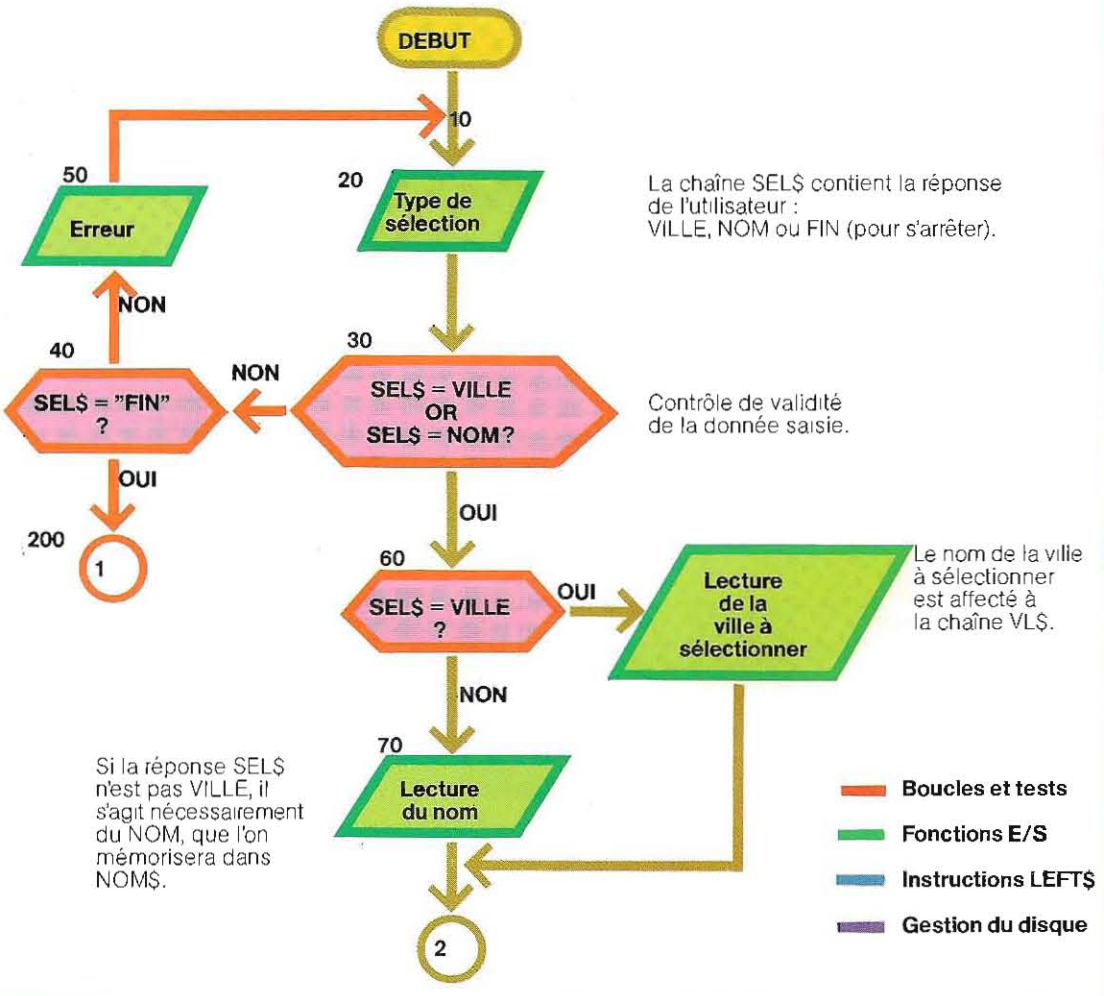
MID$(A$,NS,NC). Extrait de la chaîne A$ un nombre de caractères égal à NC, à partir de la position NS. Par exemple, les instructions :

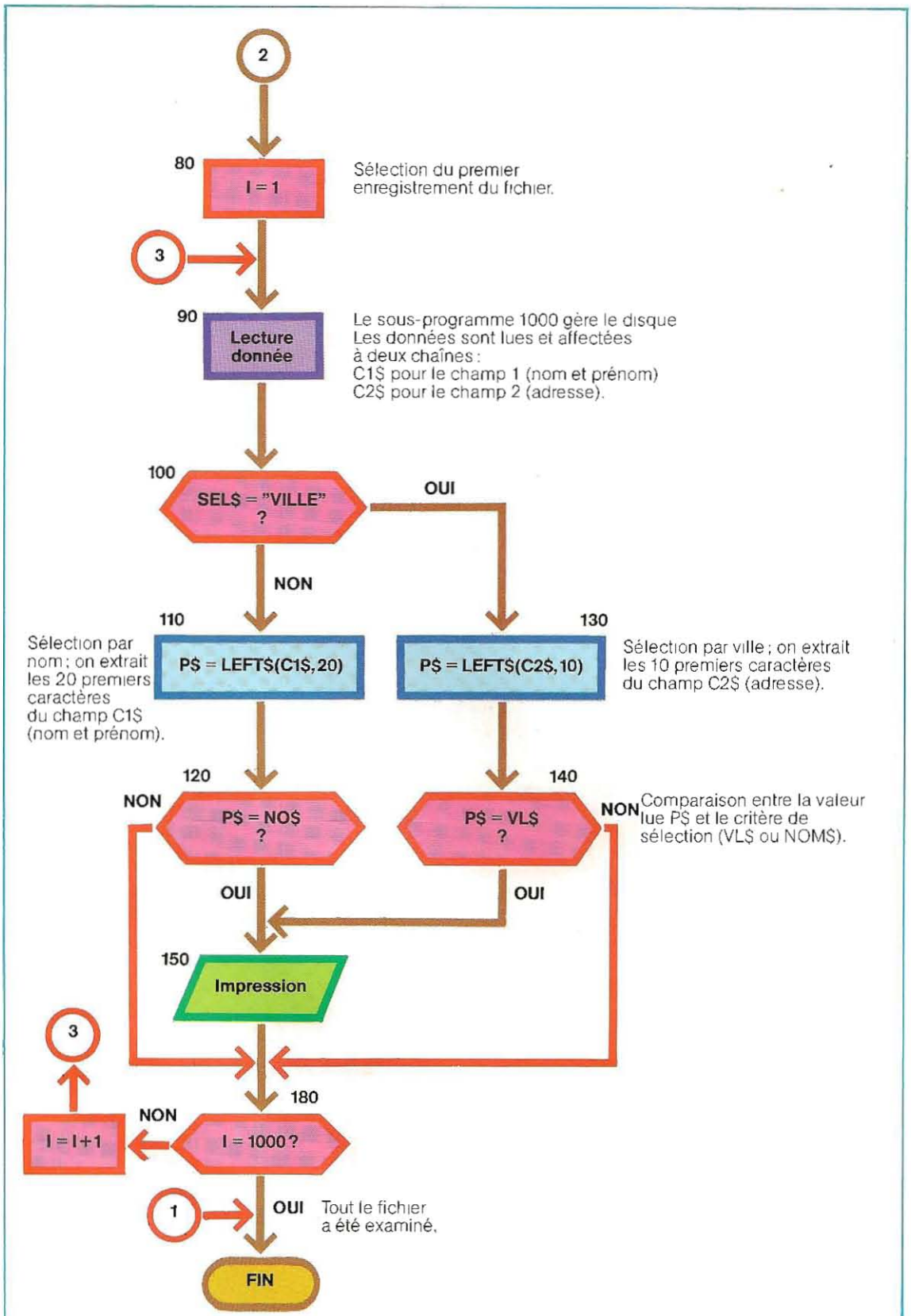
```
10 A$ = "CECI EST UNE CHAINE DE TEST"
20 B$ = MID$(A$,5,7)
```

affectent à B$ sept caractères de A$ à partir du cinquième (de l'espace suivant CECI, jusqu'à la lettre N du mot UNE, les espaces étant comptés comme caractères) ; on aura donc B$ = " EST UN".
L'instruction MID$ peut également effectuer la substitution d'une partie de chaîne. En écrivant :

MID$(A$,NS,NC) = B$

ORGANIGRAMME DE SELECTION DES DONNEES D'UN FICHIER ADRESSES




2
80
I = 1
Sélection du premier enregistrement du fichier.
3
90
Lecture donnée
Le sous-programme 1000 gère le disque
Les données sont lues et affectées à deux chaînes :
C1$ pour le champ 1 (nom et prénom)
C2$ pour le champ 2 (adresse).
100
SEL$ = "VILLE" ?
OUI
NON
110
Sélection par nom ; on extrait les 20 premiers caractères du champ C1$ (nom et prénom).
P$ = LEFT$(C1$,20)
130
P$ = LEFT$(C2$,10)
Sélection par ville ; on extrait les 10 premiers caractères du champ C2$ (adresse).
120
NON
P$ = NO$?
140
P$ = VL$?
NON
Comparaison entre la valeur lue P$ et le critère de sélection (VL$ ou NOM$).
OUI
OUI
150
Impression
3
180
NON
I = I+1
I = 1000 ?
1
OUI
Tout le fichier a été examiné.
FIN

PROGRAMME DE SELECTION DES DONNEES DANS UN FICHIER D'ADRESSES

```
2 REM * Programme de sélection des données dans un fichier ADRESSES *
3 REM ***************************************************
5 REM Exécuté sur EXORSET 33 EN BASICM
6 REM En BASICM, le backslash ( \ ) permet d'insérer un commentaire
7 REM                                          en fin de ligne
10 PRINT "Entrez le critère de sélection, parmi les suivants :"
20 INPUT "VILLE, NOM ou FIN (pour sortir)",SEL$   \ En BASICM, le symbole
30 IF SEL$="VILLE" OR SEL$="NOM" THEN GOTO 70     \ est ',' et non ';'
40 IF SEL$="FIN" THEN GOTO 260
50 PRINT "Erreur : type de sélection non prévu"
60 GOTO 10        \ EN BASICM, on ne met qu'une instruction par ligne
70 IF SEL$="VILLE" THEN INPUT "Ville de sélection",VL$
80 IF SEL$="NOM" THEN INPUT "Nom de sélection",NOM$
90 I=1     \ Sélection du 1er enregistrement
100 GOSUB 300    \ Routine de lecture des données
110 IF SEL$="VILLE" THEN GOTO 170
120 REM Sélection sur le nom
130 P$=LEFT$(C1$,20)     \ extraction du nom (20 caractères)
140 IF P$=NOM$ THEN GOTO 200
150 GOTO 230
160 REM Sélection sur la ville
170 P$=LEFT$(C2$,10)     \ Extraction de la ville (10 caractères)
180 IF P$<>VL$ THEN GOTO 230
190 REM ** Impression des résultats **
200 PRINT "Nom     :";C1$
210 PRINT "Adresse :";C2$
220 REM
230 IF I = 1000 THEN GOTO 260
240 I=I+1
250 GOTO 100     \ boucle sur les enregistrements
260 END
280 REM
290 REM ** Routine de lecture sur disque **
300 REM    (sera détaillée par la suite)
310 RETURN
```

le contenu de la chaîne A$, en partant de la position NS et pour une longueur de NC caractères, est remplacé par le contenu de B$. Par exemple :

```
10 B$ = "75"
20 A$ = "7, rue de Rennes 69"
30 MID$(A$,17,2) = B$
```

donnera : A$ = "7, rue de Rennes 75"
Sur la page ci-contre est listé à titre d'exemple un programme qui utilise les deux formats de cette instruction.

MKI$(N). Convertit un entier (ou une expression entière) N en une chaîne de 2 caractères : c'est la fonction inverse de CVI(A$).

MKS$(R). Convertit un nombre réel R en une chaîne de 4 caractères ; c'est la fonction inverse de CVS(A$).

MKD$(D). Convertit D, réel en double précision, en une chaîne de 8 caractères ; c'est la fonction inverse de CVD(A$).
Le groupe d'instructions MKI$, MKS$, et MKD$ est surtout utilisé pour le transfert sur disque de valeurs numériques dans un format plus compact que l'ASCII, qui transforme chaque chiffre en un caractère. A l'opposé, les opérations de lecture font appel aux fonctions inverses (CVI, CVS, CVD). Ce sujet sera repris et approfondi dans le chapitre consacré aux mémoires de masse.

OCT$(N). Convertit un entier N en une chaîne de caractères correspondant à la représentation octale de N. Après l'exécution des lignes :

```
10 N = 16
20 A$ = OCT$(N)
```

la chaîne A$ contient les caractères 2 et 0 ; 20 est l'équivalent octal du nombre décimal 16. Cette instruction, ainsi que HEX$, a permis de générer les tableaux de conversion des nombres des pages 63 à 67. Le listing du programme correspondant est reproduit page 462.

Certaines instructions, que nous n'avons pas encore étudiées, seront présentées par la suite.

RIGHT$(A$,N). Est le symétrique de LEFT$: on extrait N caractères de la chaîne A$, à partir de la droite. Par exemple, la suite d'instructions :

```
10  A$ = "JEAN MARTIN"
20  B$ = RIGHT$(A$,6)
```

transfère dans B$ six caractères de A$, à partir de la droite ; on obtient : B$ = "MARTIN". Comme avec LEFT$, si N est nul, l'instruction B$ = RIGHT$(A$,0) affecte à B$ la chaîne vide ;

EXEMPLES DE TRAITEMENT DE CHAINES

```
2 REM Programme exécuté sur EXORSET 33 en BASICM
3 REM
4 REM * Extraction d'un groupe de caractères avec MID$ *
6 REM Variables utilisées
7 REM A$ : chaîne initiale
8 REM NS : position du 1er caractère à extraire
9 REM NC : nombre de caractères à extraire
10 INPUT "Chaîne initiale ",A$    \ en BASICM symbole ',' et nom ';'
30 INPUT "Nombre de caractères à extraire      ",NC
40 IF NC = 0 THEN GOTO 240      \ sortie du programme
50 IF NC <= 255 THEN GOTO 80
60 PRINT "Erreur : chaîne trop longue"
70 GOTO 30      \ En BASICM, une seule instruction par ligne
80 INPUT "Position du 1er caractère à extraire ", NS
81 REM Vérification : l'opération demandée est-elle possible ?
85 K = NS+NC-1
90 IF LEN(A$) >= K THEN GOTO 120
95 PRINT "La chaîne '";A$;"' est trop courte"
100 PRINT "pour qu'on manipule ";NC;" caractères à partir du ";NS;"eme"
110 GOTO 20
120 B$ = MID$(A$,NS,NC)
130 PRINT "NS = ";NS;" NC = ";NC
140 PRINT "Chaîne initiale : '";A$;"'"
150 PRINT "Chaîne extraite : '";B$;"'"
151 REM
152 REM * Substitution d'un groupe de caractères avec RIGHT$ et LEFT$
160 INPUT "Caractère(s) de substitution ? ",B$
161 REM Vérification
162 IF LEN(B$) >= NC THEN GOTO 200
170 PRINT "La chaîne '";B$;"' est trop courte"
180 PRINT "Elle doit avoir au moins ";NC;" caractères"
190 GOTO 160
200 C1$ = LEFT$(A$,NS-1)
210 C2$ = RIGHT$(A$,LEN(A$)-NS-NC+1)
215 C$ = LEFT$(B$,NC)
216 A$ = C1$ + C$ + C2$
220 PRINT "Nouvelle chaîne obtenue : '";A$;"'"
225 PRINT
230 GOTO 20
240 END
```

```
NS = 15    NC = 5
Chaîne initiale : 'LE GATEAU EST ENFIN CUIT'
Chaîne extraite : 'ENFIN'
Nouvelle chaîne obtenue : 'LE GATEAU EST ALORS CUIT'

NS = 9    NC = 5
Chaîne initiale : 'IL FAIT CHAUD AUJOURD'HUI !'
Chaîne extraite : 'CHAUD'
Nouvelle chaîne obtenue : 'IL FAIT FROID AUJOURD'HUI !'

NS = 12    NC = 7
Chaîne initiale : 'DONNEZ-MOI L'HEURE'
Chaîne extraite : 'L'HEURE'
Nouvelle chaîne obtenue : 'DONNEZ-MOI LA MAIN'
```

PROGRAMME DE CONVERSION

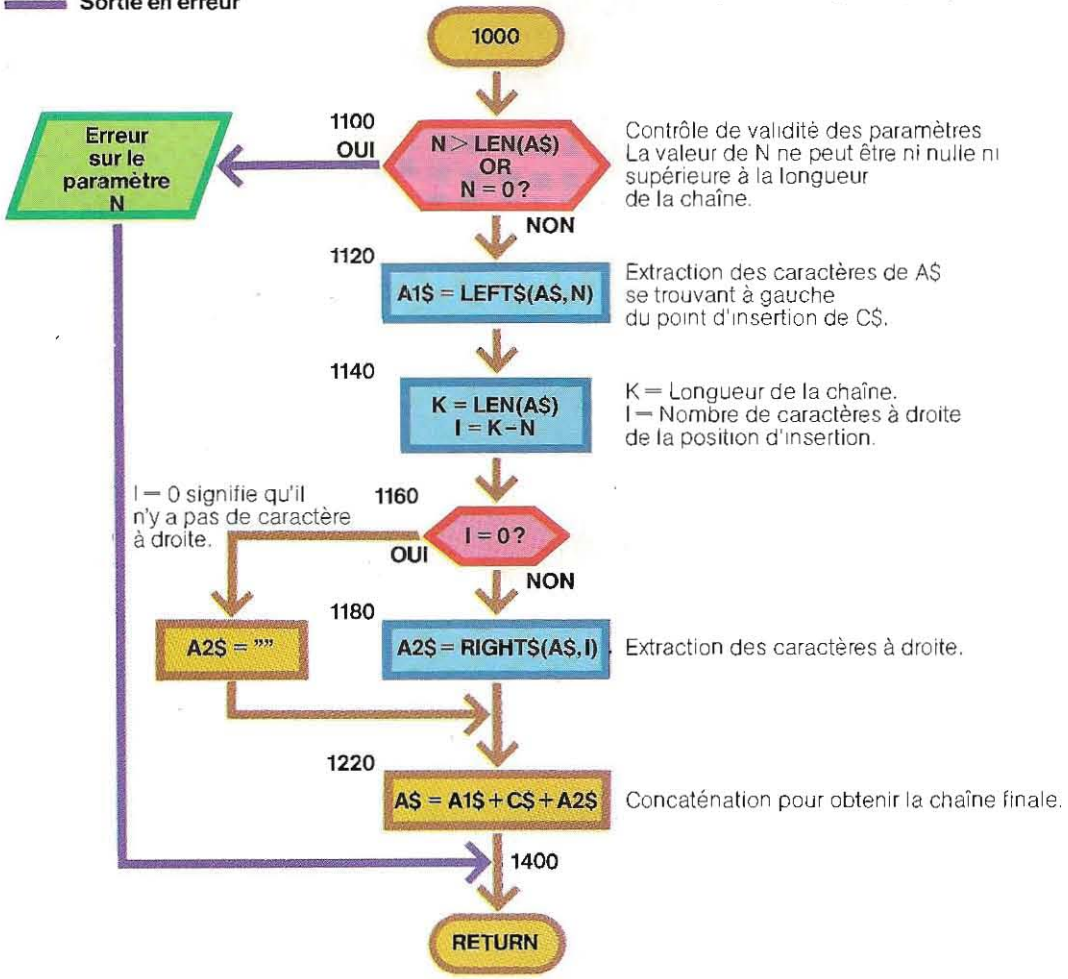
```
15 ' ** DECLARATION DES VARIABLES ET DES TABLEAUX
20 OPTION BASE 1
25 DEFINT A-P
30 DIM B(16)    ' tableau des valeurs à transformer (16)
35 LPRINT TAB(2);"DECIMAL";TAB(20);"BINAIRE";TAB(50);"OCT";TAB(65);"HEX"
36 LPRINT : LPRINT
40 '
45 A1$ = "valeur incorrecte"  ' test des valeurs entrées
100 ' ** SAISIE DES VALEURS ET CONTROLES
105 '
110 INPUT "valeur initiale";NDEB
115 IF NDEB < 0 OR NDEB > 32000 THEN PRINT A1$ : GOTO 110
120 INPUT "valeur finale";NFIN
125 IF NFIN < NDEB THEN PRINT A1$ : GOTO 120
130 '
135 ' ** Début de boucle. N varie de NDEB à NFIN
140 '
145 FOR N = NDEB TO NFIN
150 '
155 ' ** Conversion en binaire
160 '
165 GOSUB 1000
170 '
175 ' ** Conversion en octal
180 '
185 A3$ = OCT$(N)
190 '
195 ' ** Conversion en hexadécimal
200 '
205 A4$ = HEX$(N)
210 '
215 ' ** IMPRESSION
220 '
225 ' ** Conversion de B(16) en chaîne
230 A2$ = ""
235 FOR I = 16 TO 1 STEP -1
240 A2$ = A2$ + STR$(B(I))
245 NEXT I
275 LPRINT TAB(2);N;TAB(20);A2$;TAB(50);A3$;TAB(65);A4$
295 NEXT N
300 END

1000 '
1005 ' ** Conversion d'un entier en binaire
1010 ' ENTREE : N = nombre entier à convertir
1015 ' SORTIE : B(16) = tableau des bits (symboles 0 et 1)
1020 '                  de la représentation binaire de N
1025 '
1030 '
1035 M = N    ' Sauvegarde de la donnée entrée
1040 FOR I = 1 TO 16
1045 B(I) = 0     ' Initialisation
1050 NEXT I
1060 I = 1        ' L'indice I désigne le bit I de B(16)
1065 '             à partir de la droite
1070 D = M/2      ' Division entière (D déclaré de type entier)
1075 K = M-D*2   ' K = reste de la division
1080 B(I) = K     ' Le bit I est à 0 si M est multiple de 2, à 1 sinon
1085 IF M = 1 GOTO 1100   ' Fin de la conversion
1090 M = D
1095 I = I+1 : GOTO 1070  ' On décale d'un bit vers la gauche dans B(16)
1100 RETURN
```

si N est égal à la longueur totale de A$ [N = LEN(A$)], on transfère dans B$ toute la chaîne A$. Page 463 figure l'organigramme de la routine 1000, qui insère un caractère dans une position donnée d'une chaîne ; le listing correspondant se trouve en haut de la page 464. Un autre sous-programme (2000) est ensuite présenté : il permet cette fois de supprimer

ROUTINE 1000 D'INSERTION D'UN CARACTERE DANS UNE CHAINE

Fonctions E/S
Tests
Traitement des chaînes
Sortie en erreur

Données d'entrée du programme.
A$ = Chaîne à modifier
C$ = Caractère à insérer
N = Position d'insertion (en nombre de caractères à partir de la gauche).



Contrôle de validité des paramètres
La valeur de N ne peut être ni nulle ni supérieure à la longueur de la chaîne.

Extraction des caractères de A$ se trouvant à gauche du point d'insertion de C$.

K = Longueur de la chaîne.
I = Nombre de caractères à droite de la position d'insertion.

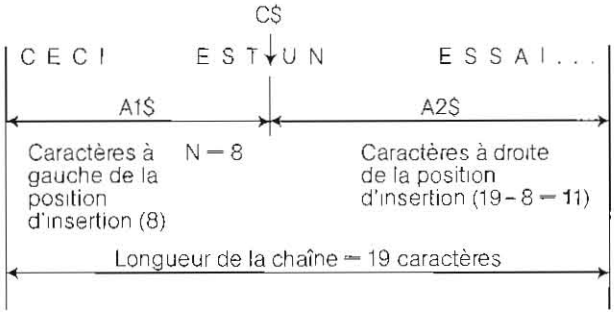
Extraction des caractères à droite.

Concaténation pour obtenir la chaîne finale.

Exemple :

Chaîne saisie = A$ =

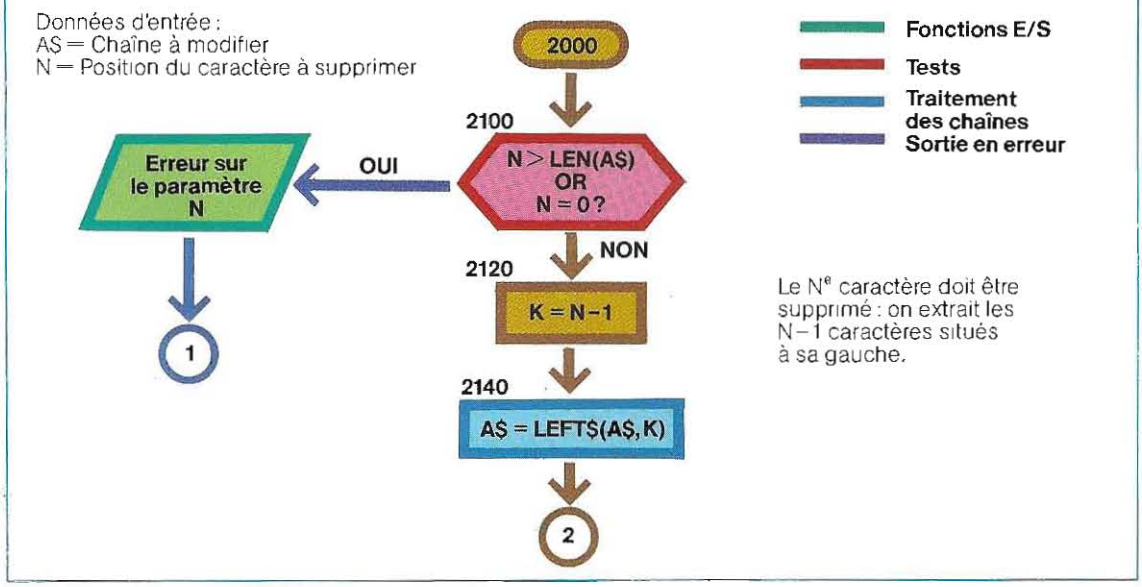
Caractère à insérer :
C$ = blanc

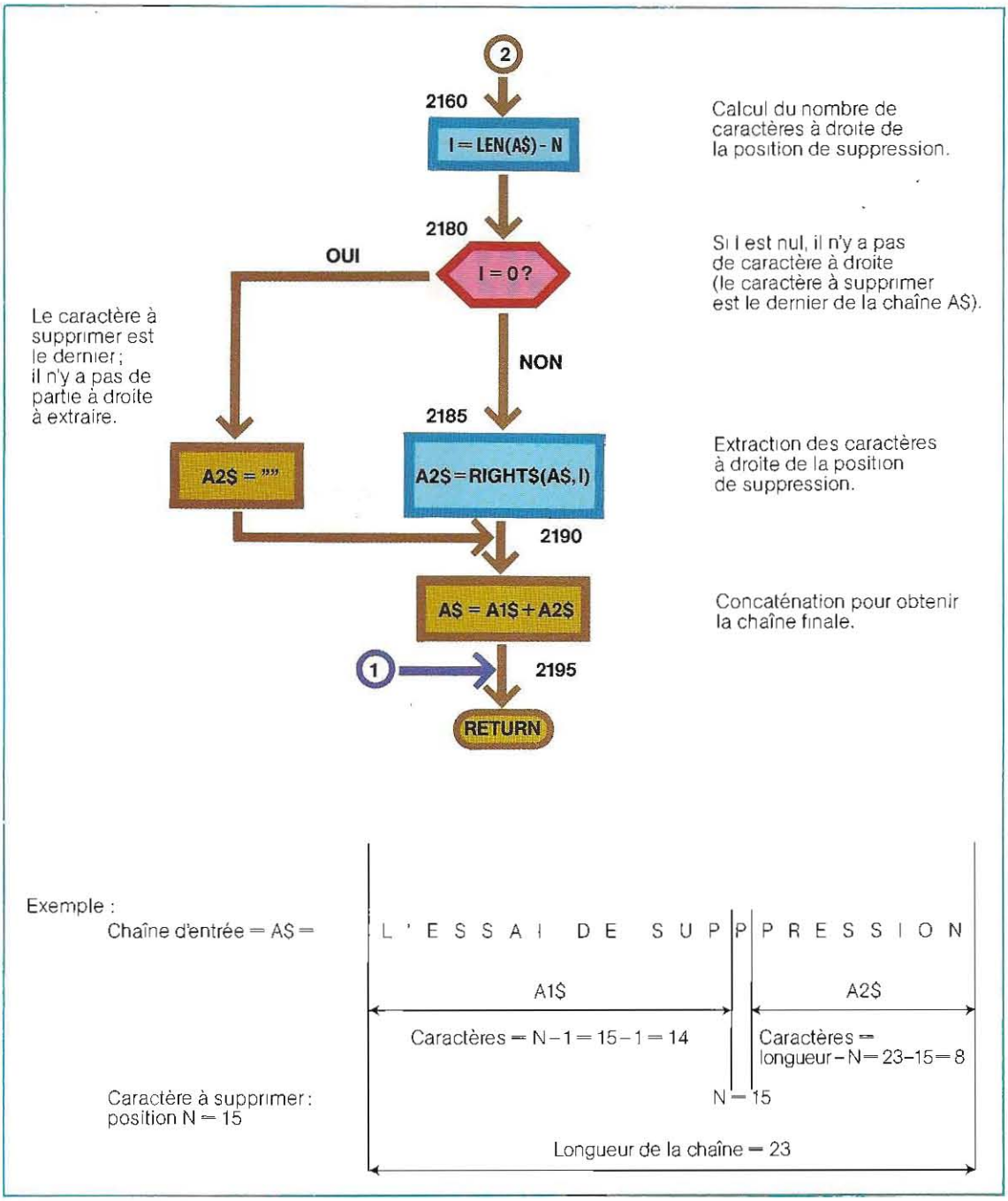


EXEMPLE D'UTILISATION DU SOUS-PROGRAMME 1000

```
10  ' ** PROGRAMME PRINCIPAL **
50  INPUT "CHAINE A MODIFIER"
60  INPUT "CARACTERE A INSERER"
62  ' IF LEN(C$)=0 THEN C$=" "
64  ' SI LE CARACTERE D'INSERTION EST LA CHAINE NULLE,
66  ' L'INSTRUCTION 62 AFFECTE UN BLANC A C$
70  INPUT "QUELLE EST LA POSITION D'INSERTION (nombre de caractères)?"
75  B$=A$ ' SAUVEGARDE DANS B$ DE LA CHAINE D'ORIGINE
80  GOSUB 1000
90  '
100 ' INSTRUCTIONS D'IMPRESSION
110 LPRINT "CHAINE INITIALE :";B$
120 LPRINT "CARACTERE A INSERER (";C$;")"
130 LPRINT "CHAINE MODIFIEE :";A$
132 LPRINT
134 GOTO 50
140 END
1000 ' ** ROUTINE D'INSERTION D'UN CARACTERE DANS UNE CHAINE **
1001 '
1002 '
1003 '
1010 ' DONNEES D'ENTREE DU SOUS-PROGRAMME
1020 '       A$ = CHAINE INITIALE
1030 '       C$ = CARACTERE A INSERER
1040 '       N = POSITION DE L'INSERTION, EN NOMBRE DE CARACTERES
1050 '         A PARTIR DE LA GAUCHE
1100 IF N>LEN(A$) OR N=0 THEN PRINT "ERREUR SUR LE PARAMETRE N"
1120 A1$=LEFT$(A$,N)  ' STOCKAGE DANS A1$ DES CARACTERES A GAUCHE
1140 K=LEN(A$)  ' STOCKAGE DANS K DE LA LONGUEUR DE A$
1145 I=K-N   ' I = NOMBRE DE CARACTERES A DROITE DE LA POSITION D'INSERTION
1160 IF I=0 THEN A2$="":GOTO 1220
1180 A2$=RIGHT$(A$,I)  ' STOCKAGE DANS A2$ DES CARACTERES A DROITE
1220 A$ = A1$+C$+A2$    ' CONCATENATION
1400 RETURN


CHAINE INITIALE : CECI ESTUN ESSAI...
CARACTERE A INSERER ( )
CHAINE MODIFIEE : CECI EST UN ESSAI...
```

ROUTINE 2000 DE SUPPRESSION D'UN CARACTERE DANS UNE CHAINE





(on dit aussi « déléter ») un caractère de rang donné dans une chaîne. Son organigramme se trouve pages 464 et 465, et son listing, suivi d'un exemple d'application, apparaît à la page 466.

Les deux procédures décrites sont programmées à partir des simples fonctions de traitement des chaînes disponibles en Basic. Elles s'avèrent très utiles, car les fonctions d'insertion et de suppression d'un caractère ne figurent pas dans la bibliothèque standard des instructions Basic.

STR$(R). Convertit la valeur numérique R en une chaîne, où chaque caractère représente un chiffre du nombre. Par exemple, les lignes :

```
10  R = 126
20  A$ = STR$(R)
```

EXEMPLE D'UTILISATION DU SOUS-PROGRAMME 2000

```
50 INPUT "CHAINE A MODIFIER (chaine vide pour sortir)";A$
55 IF A$ = "" GOTO 140   ' SORTIE
60 INPUT "POSITION DU CARACTERE A SUPPRIMER"
65 B$ = A$   ' SAUVEGARDE DANS B$ DE LA CHAINE D'ORIGINE
70 GOSUB 2000   ' ROUTINE DE SUPPRESSION D'UN CARACTERE
80 '
90 ' INSTRUCTIONS D'IMPRESSION
100 LPRINT "CHAINE INITIALE:"
110 LPRINT "CHAINE MODIFIEE:"
120 LPRINT
130 GOTO 50
140 END
2000 ' ** SOUS-PROGRAMME DE SUPPRESSION D'UN CARACTERE **
2010 '    DONNEES D'ENTREE
2020 '    A$ = CHAINE A MODIFIER
2030 '    N = POSITION DU CARACTERE A SUPPRIMER
2100 IF N>LEN(A$) OR N=0 THEN PRINT "ERREUR SUR LE PARAMETRE N"
2120 K=N-1
2122 ' K = NOMBRE DE CARACTERES A GAUCHE
2124 '    DE CELUI A SUPPRIMER
2140 A$=LEFT$(A$,K)
2142 ' STOCKAGE DANS A1$ DE CES K CARACTERES
2160 I=LEN(A$)-N
2162 ' I = NOMBRE DE CARACTERES A DROITE DE
2164 '    CELUI A SUPPRIMER
2180 IF I=0 THEN A2$="":GOTO 2190
2185 A2$ = RIGHT$(A$,I)   ' STOCKAGE DANS A2$ DE CES I CARACTERES
2190 A$ = A1$+A2$         ' CONCATENATION
2195 RETURN

CHAINE INITIALE : L'ESSAI DE SUPPPRESSION
CHAINE MODIFIEE : L'ESSAI DE SUPPRESSION
```

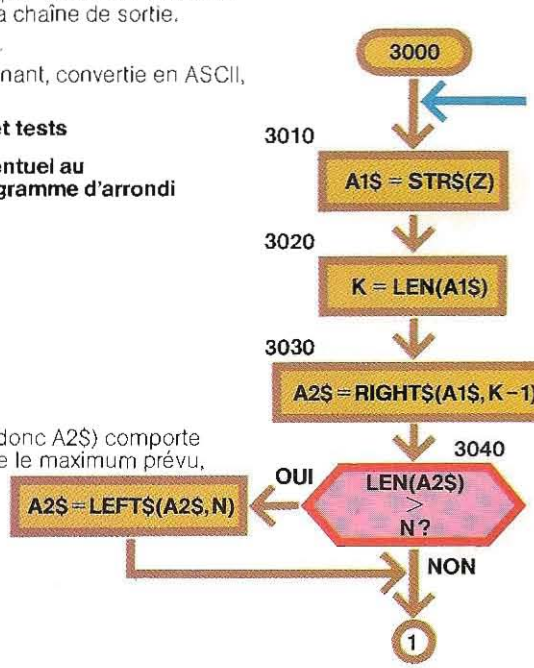
CONVERSION D'UNE VALEUR NUMERIQUE EN ASCII AVEC SUPPRESSION DU PREMIER CARACTERE (SIGNE)

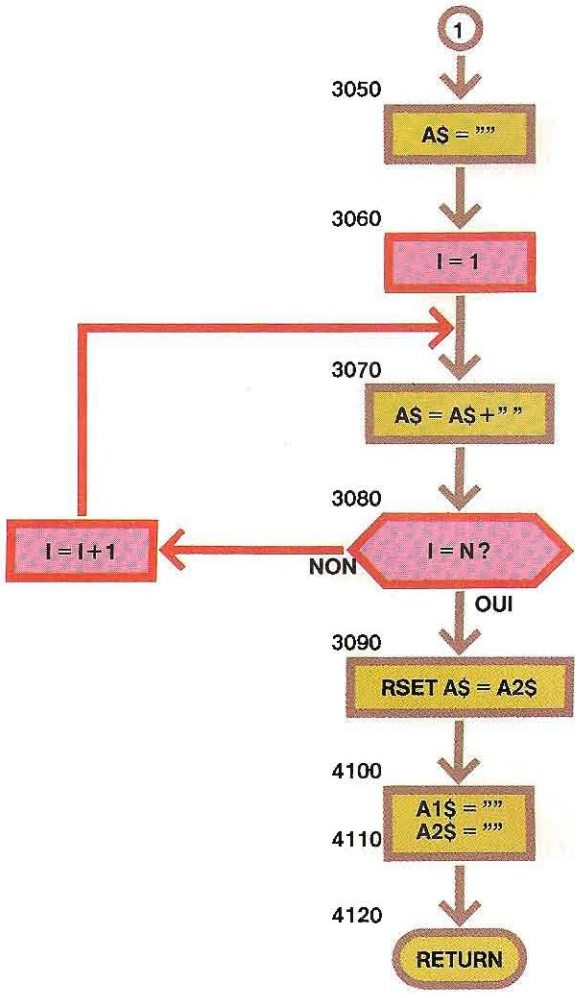
Données d'entrée :
Z = Valeur numérique à convertir en ASCII.
N = Longueur de la chaîne de sortie.

Données de sortie :
A$ = Chaîne contenant, convertie en ASCII, la valeur Z.

Boucles et tests

Appel éventuel au sous-programme d'arrondi





EXEMPLE D'APPLICATION DU SOUS-PROGRAMME 3000

```
50 INPUT "VALEUR NUMERIQUE A CONVERTIR EN ASCII:    ";Z
60 INPUT "LONGUEUR DE LA CHAINE DE SORTIE:    ";N
70 Z1 = Z   ' SAUVEGARDE DANS Z1 DE LA VALEUR INITIALE
80 GOSUB 1000
90 LPRINT "VALEUR NUMERIQUE INITIALE:    ";Z1
100 LPRINT "VALEUR CONVERTIE EN ASCII:    ";A$
110 LPRINT
120 END
3000 ' ** ROUTINE DE CONVERSION D'UN NOMBRE EN ASCII **
3001 '    AVEC SUPPRESSION DU PREMIER CARACTERE
3010 A1$ = STR$(Z)
3020 K = LEN(A1$)
3030 A2$ = RIGHT$(A1$,K-1)
3040 IF LEN(A2$) > N THEN A2$ = LEFT$(A2$,N)
3050 A$=""
3060 FOR I = 1 TO N
3070 A$ = A$ + " "
3080 NEXT I
3090 RSET A$ = A2$
4100 A1$ = ""
4110 A2$ = ""
4120 RETURN
```

Test 14



1 / Certaines des instructions suivantes sont erronées : lesquelles ?
a) A$ = LEFT$(B$,4) b) A$ = MID$(B$,3)
c) A$ = STRING$(3,1) d) A$ = VAL("ABCHD")

2 / Une chaîne contient les 42 caractères suivants :
A$ = "Stylos type A quantité 1250 couleur : rouge"
Sachant que la zone de caractères numériques (1250) débute en position 24 et a une longueur de 4 caractères, écrire un sous-programme réalisant son extraction et sa conversion en valeur numérique.

3 / Écrire un programme qui ajoute une quantité N au nombre obtenu précédemment, et reconstruire la chaîne avec la nouvelle valeur.

4 / Quelles sont les fonctions ou instructions à utiliser pour obtenir :
a) une chaîne de longueur N, contenant N fois le caractère A ;
b) la représentation octale puis hexadécimale d'un nombre N ;
c) la partie entière d'un nombre réel R, obtenue par troncature.

5 / Qu'exécute le programme suivant ?

```
10 A$="*"
20 FOR I=1 TO 20
30 B$= SPACE$(I)
40 PRINT B$;A$
50 NEXT I
```

Les solutions du test se trouvent à la page 473.

transfèrent dans la chaîne A$ les caractères 1, 2, et 6 codés sous forme ASCII. Cette fonction, beaucoup plus utilisée que MK$, MKS$, et MKD$, permet le transfert de valeurs numériques sur disque. Le système d'exploitation utilise une fonction semblable pour afficher un nombre à l'écran.
Les pages 466 (en bas) et 467 présentent l'organigramme d'une routine qui met en œuvre cette fonction, ainsi que le listing du programme correspondant.
Dans ce sous-programme, on a prévu d'éliminer le premier caractère de la chaîne A$. Il représente en effet le signe qui apparaît comme un blanc pour les valeurs positives. Dans beaucoup d'applications, il n'est pas utile de conserver ce signe (toutes les valeurs étant positives), et il convient de récupérer cet espace non significatif.
Dans le cas où l'on voudra conserver le signe, on modifiera le programme localement, en supprimant les instructions 1020, 1030 et 1100, et en récrivant la ligne 1010 (1010 A2$ = STR$(Z). On ne se servira plus de A1$. L'instruction STR$ génère donc une chaîne de longueur variable, en fonction du nombre de chiffres qu'elle doit contenir. On ne peut l'employer directement pour stocker les données en ASCII sur le disque : comme on le verra, elle poserait des problèmes de mise en page (sur listing ou à l'écran), mais surtout de relecture de ces données à format variable.
Dans les programmes d'application, il est préférable d'attribuer toujours la même longueur aux chaînes représentant des valeurs numériques. On définira une longueur maximale des nombres traités, et toutes les chaînes qui les contiendront auront cette même longueur.
Dans l'exemple précédent (routine 1000), la valeur numérique 126 aurait tout d'abord généré une chaîne de 4 caractères (le premier étant un blanc). Pour obtenir un format de 8 caractères (N = 8), on crée une chaîne d'espaces, de longueur N, dans laquelle est transférée (avec cadrage) la chaîne représentant la valeur numérique codée.

L'instruction SPACE$(N) construit automatiquement la chaîne d'espaces voulue. Dans le sous-programme 1000, à titre de démonstration, la chaîne est générée par une boucle.

STRING$(N,K). Crée une chaîne de longueur N, où se répète le caractère correspondant au code ASCII. Par exemple, l'instruction

A$ = STRING$(5,42)

génère la chaîne A$, qui contient cinq fois le symbole * (de code ASCII 42).
Une autre syntaxe est possible :

STRING$(N,B$)

En ce cas, on construit une chaîne de N caractères égaux au premier caractère de B$.

Par exemple, en exécutant les lignes

```
10 B$="***ABC***"
20 A$=STRING$(5,B$)
```

La chaîne A$ obtenue est équivalente à celle de l'exemple précédent : le premier caractère de B$ est en effet représenté par le code 42.

VAL(A$). Fournit la valeur numérique de la chaîne A$. En d'autres termes, c'est la fonction réciproque de STR$(R).
Exemple :

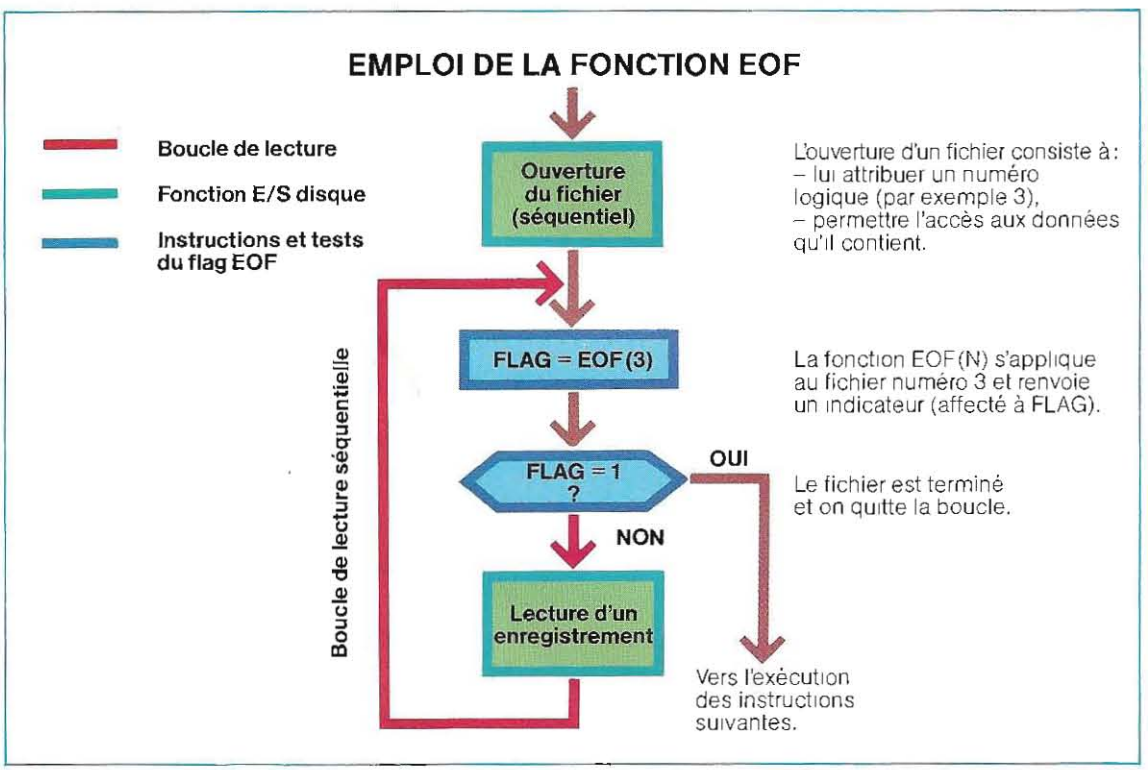
```
10 A$="576.51"
20 R=VAL(A$)
```

Ces instructions transforment le contenu de A$ en valeur numérique et affectent le résultat à la variable R.
On se sert en général de VAL$ après avoir lu des données codées en ASCII sur le disque, afin de les reconvertir en valeurs numériques utilisables par la machine. Notons que l'interpréteur du système réalise une fonction similaire pour saisir un nombre à l'écran.

Fonctions spéciales

A ce groupe appartiennent certaines fonctions particulières gérant les périphériques ou la mémoire. Leur utilisation correcte implique une connaissance approfondie du fonctionnement physique de la machine.

EOF(N). Produit un indicateur qui prend la valeur 1 (ou –1 selon les systèmes) lorsqu'en phase de lecture on rencontre la fin du fichier

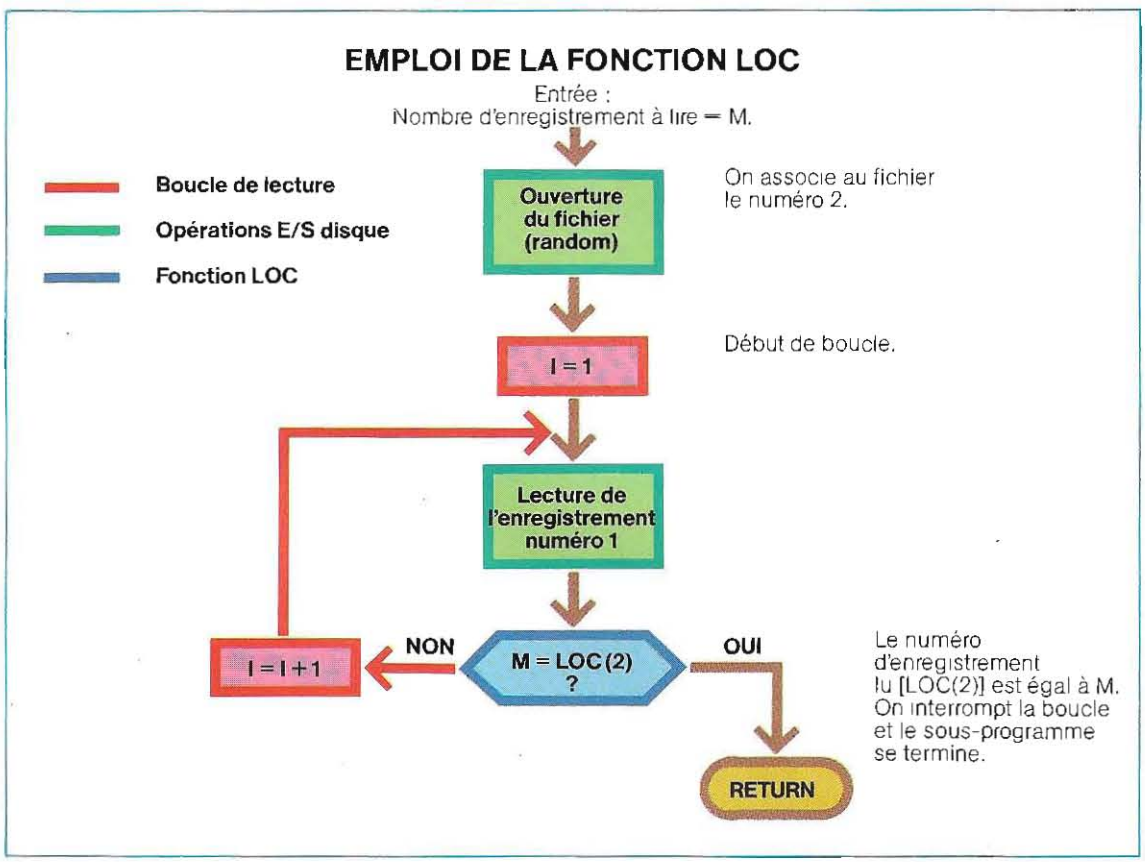


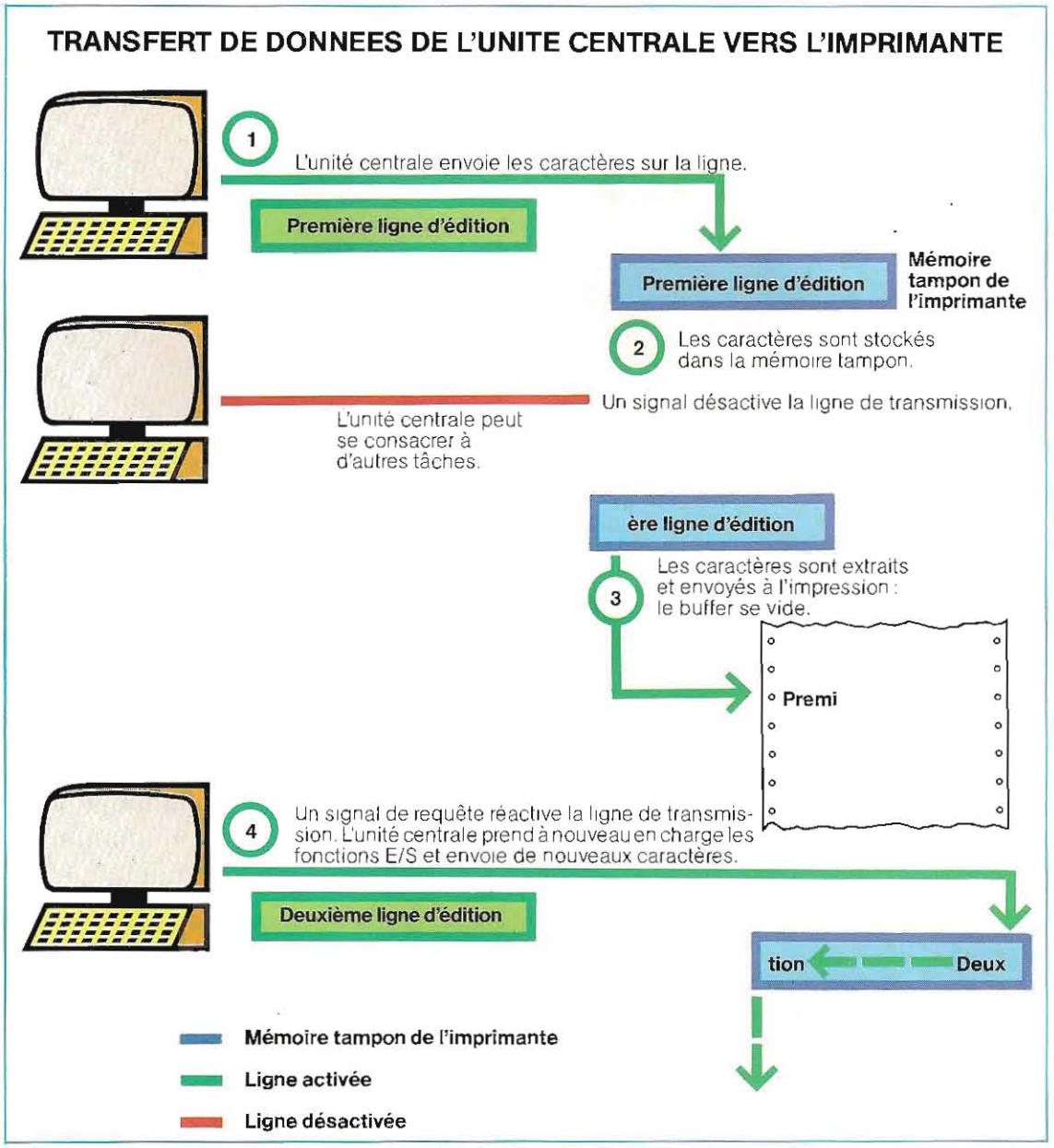
séquentiel N. La valeur N, argument de la fonction, désigne le fichier auquel EOF se réfère. Par exemple, en écrivant EOF(3), on applique la fonction au fichier numéro 3 (ce numéro logique est attribué en phase d'ouverture du fichier). L'organigramme ci-dessous illustre, d'un point de vue logique, la mise en œuvre de la fonction EOF.

FRE(A). Renvoie l'étendue, en nombre d'octets, de la mémoire disponible à un moment donné. L'argument A est « dummy » (fictif) : il n'a aucune signification et n'est mentionné que pour respecter la syntaxe classique d'une fonction.
Ainsi, les instructions FRE(A), FRE(O) ou FRE(C$) seront équivalentes.
Prenons un exemple : supposons une machine ayant 36 K octets de mémoire centrale, où est déjà chargé un programme qui occupe 14 Ko. La fonction FRE(A) fournira la valeur 22000, qui correspond à la place mémoire encore libre.
La syntaxe complète de l'instruction, pour obtenir à l'écran la quantité d'espace mémoire disponible, est PRINT FRE(A).

INP(N). Est une fonction générale d'entrée. Elle renvoie la valeur de l'octet présent sur le « port d'entrée » N. Schématiquement, un port d'E/S peut être représenté par une case mémoire reliée à l'« extérieur » (périphérique). Cette fonction (comme sa « réciproque » OUT) n'est mise en œuvre que pour des applications particulières de gestion des périphériques : l'instruction habituelle permettant la saisie de données au clavier est INPUT.

LOC(N). Retourne le numéro du dernier enregistrement lu ou écrit sur un fichier en accès direct. L'argument N représente le numéro logique attribué au fichier, par exemple en phase d'ouverture.
L'organigramme ci-dessous retrace la lecture d'un nombre déterminé (M) d'enregistrements, à partir d'un fichier en accès direct.
Notre exemple ne constitue pas une application réelle, mais seulement une illustration de l'emploi de la fonction LOC.





LPOS(N) Fournit la position du caractère en cours d'impression à l'intérieur de la mémoire tampon (buffer) de l'imprimante. Comme d'autres périphériques, les imprimantes disposent d'une telle mémoire locale, qui est chargée de stocker les caractères à imprimer. L'unité centrale envoie des caractères au port d'entrées/sorties, qui les transfère dans la mémoire tampon. Lorsque celle-ci est remplie, un signal est envoyé à l'unité centrale, qui se consacre alors à d'autres tâches ; pendant ce temps, l'imprimante extrait les caractères du buffer et les imprime à son propre rythme, son fonctionnement étant totalement indépendant de l'activité de l'unité centrale.

Au fur et à mesure que les caractères seront imprimés, le buffer se videra. Lorsqu'un espace suffisant aura ainsi été libéré, l'imprimante enverra un signal à l'unité centrale, pour demander un nouveau groupe de caractères. Le fonctionnement décrit est schématisé ci-dessus.

La fonction LPOS(N) permet de désynchroniser le fonctionnement de l'unité centrale et de l'imprimante. Ceci prend tout son intérêt si l'on se rappelle que le transfert d'un caractère

(représenté par des signaux électriques) et son impression effective sur papier (qui implique le mouvement d'organes mécaniques), s'effectuent à des vitesses radicalement différentes.
Lorsque LPOS gère les transmissions de données, l'unité centrale* n'est donc pas ralentie par l'imprimante, qui reste le composant le plus lent du système (même avec les modèles les plus sophistiqués comme les imprimantes à laser).

OUT N,M. Cette instruction, présentée au chapitre des fonctions car elle joue le rôle réciproque de INP(M), permet de transférer, vers le port de sortie N, la valeur numérique M. Par exemple :

OUT 3,127

transfère la valeur numérique 127 au port de sortie numéro 3.
Comme INP(N), elle n'est utilisée que dans des cas particuliers : pour les fonctions usuelles d'émission de données, le Basic prévoit d'autres types d'instructions.
L'instruction OUT et la fonction INP sont presque identiques à celles utilisées en langage Assembleur. L'instruction Basic INPUT (qui réalise une lecture de données, comme INP) ne nécessite pas la spécification du périphérique à partir duquel s'effectue la saisie : l'interpréteur Basic (ou le compilateur) désigne systématiquement le clavier pour cette opération. En revanche, en Assembleur (et dans d'autres langages comme le Fortran), cette assignation n'est pas automatique : il appartient au programmeur de spécifier le numéro du port (unité d'entrée) sur lequel doit s'effectuer l'entrée/sortie. Cela explique en partie les raisons qui conduisent à programmer certaines applications en Assembleur plutôt qu'en Basic.
Tous les mini-ordinateurs, ainsi que les micro-ordinateurs modernes, disposent de plusieurs ports d'entrée/sortie et sont dotés d'interfaces spéciales, entre autres pour communiquer par ligne téléphonique avec d'autres ordinateurs. Des programmes en Basic standard ne permettraient pas l'accès à ces dispositifs particuliers.
Les instructions INP et OUT ne s'appliquent pas, elles non plus, à des problèmes particuliers comme la transmission des données, qui mettent en cause la structure hardware de la machine. Le langage le plus adapté sera l'Assembleur, qui reste le plus proche de la machine.
Sur certains micro-ordinateurs de haut de gamme, il existe des instructions utilisables en Basic pour gérer ces interfaces de manière satisfaisante.
De telles machines seront destinées à des applications particulières, comme le contrôle automatique d'appareillages, le recueil de données à l'aide d'instruments, etc.

PEEK(N). Lit et renvoie le contenu de la position N de la mémoire centrale (octet situé à l'adresse N). La valeur retournée est un nombre compris entre 0 et 255. La fonction PEEK requiert donc l'adresse « absolue » de la donnée en question.
Dans tous les langages de haut niveau, en particulier le Basic, chaque emplacement mémoire référencé dans les programmes porte un nom symbolique donné par le système. Aucune adresse physique (position réelle dans l'espace mémoire) n'est donc connue du programmeur ; c'est pourquoi la fonction PEEK, et son instruction réciproque POKE, qui écrit dans la mémoire, sont très peu utilisées dans les programmes d'application classiques.
Cependant, ces fonctions trouvent leur utilité dans la gestion de l'écran vidéo. Si l'on connaît la « cartographie » de la mémoire vidéo, c'est-à-dire la position physique de toutes ses informations, il devient possible de modifier les attributs de l'écran, en changeant, par exemple, la forme du curseur ou sa fréquence de clignotement.

POKE N,M. On a choisi de réserver dans ce chapitre une place à l'instruction POKE, qui opère de manière réciproque vis-à-vis de la fonction PEEK.
L'instruction POKE N,M écrit à l'adresse N de la mémoire, la valeur M (M représentant un octet, est compris entre 0 et 255).

* On mesure la vitesse de l'unité centrale en MIPS (Millions d'Instructions par Seconde). Les vitesses obtenues actuellement sur les machines les plus modernes peuvent atteindre 1 ou 2 MIPS ; la mise en place de techniques particulières, encore à l'étude, permettrait la réalisation de systèmes encore plus rapides.

Solutions du test 14

1 / Les instructions erronées sont :
b) la fonction MID$ requiert deux valeurs entières (par exemple MID$(A$,3,2) ; la première indique la position de début d'extraction, la seconde, le nombre de caractères à extraire ;
d) la fonction VAL, appliquée à une chaîne, renvoie une valeur numérique, qui ne peut être affectée à la chaîne A$.

---

2 / On utilisera la fonction MID$(A$,24,4) pour extraire les caractères numériques et VAL pour les convertir en nombre. Le sous-programme se réduit donc à deux instructions :

```
1000  ' * ENTRÉE DU SOUS-PROGRAMME
1010  B$ = MID$(A$,24,4)
1020  M = VAL(B$)
1030  RETURN
```

---

3 / Ce second sous-programme, plus complexe que le précédent, doit remplacer dans la chaîne le groupe de caractères numériques par le résultat de la somme N+M (N = valeur additionnelle, et M = valeur extraite), lui aussi sous forme de caractères. On scindera A$ en trois : AG$ à gauche, AN$ représentant le nombre converti en chaîne, et AD$ à droite. En concaténant ces trois chaînes, on obtient la nouvelle chaîne A$ complète et contenant la valeur désirée.

```
2000  ' ** ENTRÉE DU SOUS-PROGRAMME
2010  AG$ = LEFT$(A$,23)      ' Partie gauche
2020  AN$ = STR$(N+M)         ' Valeur numérique
2030  AD$ = MID$(A$,28,15)    ' Partie droite
2040  A$ = AG$+AN$+AD$        ' Concaténation
2050  RETURN
```

Puisque l'on a 42 – 27 = 15 caractères à droite de la zone numérique, l'instruction 2030 peut être remplacée par : AD$ = RIGHT$(A$,15).

---

4 / a) STRING$(N,A$), en ayant posé A$ = "A"
b) OCT$(N) et HEX$(N)
c) FIX(R). Attention : la fonction INT(R) n'opère pas par troncature mais par arrondi.

---

5 / La fonction SPACE$(I) crée une chaîne de I caractères blancs (espaces). A l'impression, l'astérisque de A$ se décale à chaque ligne d'une position vers la droite par rapport à la ligne précédente. A la première itération (passe I = 1), on imprime une chaîne ne contenant qu'un seul blanc [B$ = SPACE$(1)], suivie de A$ qui occupe donc la colonne 2. En seconde passe, B$ contient deux blancs et A$ est imprimé en colonne 3, et ainsi de suite.

POS(N). Retourne la position courante du curseur. L'argument N est fictif. Par exemple, la ligne :

```
K = POS(0)
```

affecte à la variable K la position du curseur.

Fonctions définies par l'utilisateur.

On peut enrichir la bibliothèque standard de fonctions Basic en programmant, lors d'une application, de nouvelles fonctions, définies par l'utilisateur.
L'instruction permettant la définition d'une

nouvelle fonction a déjà été étudiée précédemment :

DEF FN NOM = expression

NOM désigne le nom dont on veut baptiser la fonction, et « expression » représente les opérations que la fonction doit effectuer afin de calculer la valeur à renvoyer.
Par exemple, pour évaluer l'hypoténuse d'un triangle rectangle de côtés X et Y, il faut exécuter le calcul suivant (théorème de Pythagore) :

$$H = \sqrt{X^2 + Y^2}$$

qui, une fois traduit en Basic (la racine carrée se calcule par la fonction SQR, l'élévation à une puissance par le symbole Λ), devient :

H = SQR (XΛ2+YΛ2)

Pour exécuter ce calcul plusieurs fois au cours d'un programme, trois possibilités sont envisageables :

1 / écrire chaque fois la formule ;
2 / appeler un sous-programme ;
3 / utiliser une fonction.

La première solution n'est pas rationnelle : elle augmente inutilement l'espace mémoire. Un sous-programme d'une seule ligne (le calcul de H) serait efficace, mais fastidieux : chaque appel nécessite la mention du numéro de ligne où il a été écrit (instruction GOSUB n, avec n = numéro de ligne où commence le sous-programme).
La dernière solution est la plus pratique. En écrivant :

DEF FNA(X,Y) = SQR(XΛ2+YΛ2)

on crée une fonction de nom A, s'appliquant à deux arguments X et Y (côtés), qui exécute le calcul de l'hypoténuse. En mentionnant n'importe où dans le programme :

H = FNA(X,Y)

la valeur de l'hypoténuse sera calculée et

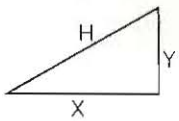
Centre de traitement des données équipé de matériel IBM.



K. Reese/Marka

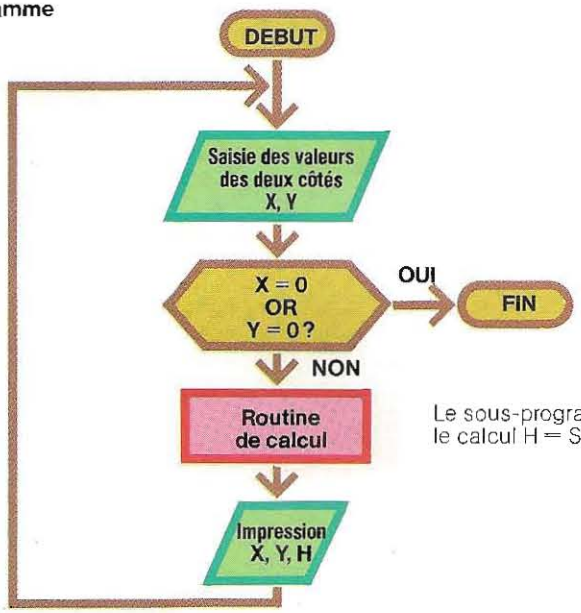
COMPARAISON DES DEUX METHODES DE CALCUL : SOUS-PROGRAMME OU FONCTION

Le calcul à effectuer est
$H = \sqrt{X^2+Y^2}$



Instructions inhérentes à la fonction
Fonctions E/S
Sous-programme

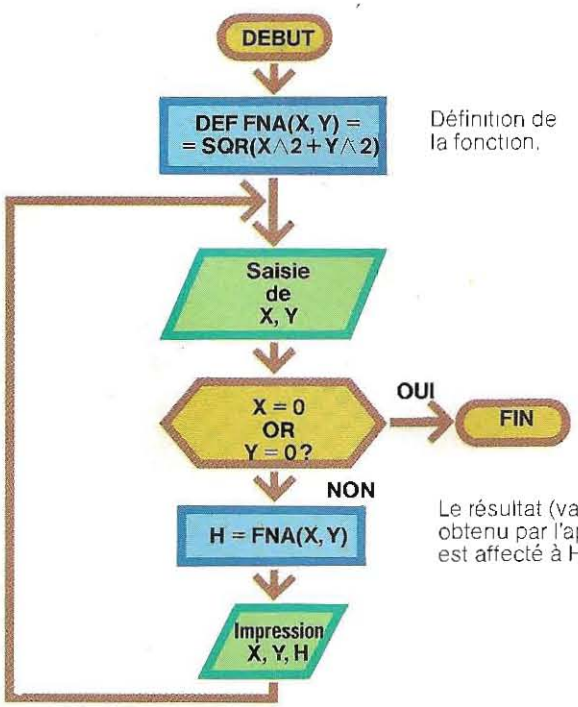
Avec un sous-programme



Le programme se termine lorsqu'un des deux côtés est nul.

Le sous-programme exécute le calcul H = SQR(X<2+Y<2)

Avec une fonction



Définition de la fonction.

Le résultat (valeur de l'hypoténuse), obtenu par l'appel de la fonction, est affecté à H.

affectée à la variable H.
Les organigrammes représentés page 475 correspondent à deux programmes exécutant ce même calcul. Le premier utilise un sous-programme, le second une fonction. Cette page reproduit les listings des programmes ainsi qu'un certain nombre de résultats.
La fonction définie ci-dessus est de type numérique (les arguments et le résultat sont des nombres), mais de manière analogue, on peut envisager des fonctions de chaîne.

PROGRAMME APPELANT UN SOUS-PROGRAMME

```
10 ' ** PROGRAMME UTILISANT UNE SUBROUTINE
20 '
30 ' PROGRAMME PRINCIPAL
40 '
50 PRINT "ENTREZ LES DEUX COTES DU TRIANGLE (X, Y)"
60 INPUT "X =    ";X
70 INPUT "Y =    ";Y
80 IF X = 0 OR Y = 0 GOTO 170
90 GOSUB 200    ' APPEL DU SOUS-PROGRAMME
100 '
110 ' IMPRESSION DU RESULTAT
120 LPRINT "LONGUEUR DU PREMIER COTE X = ";X
130 LPRINT "LONGUEUR DU SECOND COTE Y = ";Y
140 LPRINT "LONGUEUR DE L'HYPOTHENUSE H = ";H
150 LPRINT
160 GOTO 50     ' RETOUR POUR UN NOUVEAU CALCUL
170 END
180 '
190 '
200 ' SUBROUTINE DE CALCUL DE L'HYPOTHENUSE
210 '
220 H = SQR(X*X + Y*Y)     ' FORMULE DE PYTHAGORE
230 '
240 RETURN
```

```
LONGUEUR DU PREMIER COTE     X = 2
LONGUEUR DU SECOND COTE      Y = 2
LONGUEUR DE L'HYPOTHENUSE    H = 2.82843
```

PROGRAMME UTILISANT UNE FONCTION UTILISATEUR

```
10 ' PROGRAMME UTILISANT UNE FONCTION UTILISATEUR
20 '
30 DEF FNA(X, Y) = SQR(X*X + Y*Y)
40 '       L'utilisateur a défini une fonction appelée A(X, Y)
50 PRINT "ENTREZ LES DEUX COTES DU TRIANGLE (X, Y)"
60 INPUT "X =    ";X
70 INPUT "Y =    ";Y
80 IF X = 0 OR Y = 0 GOTO 160
90 H = FNA(X, Y)    ' ON AFFECTE A H LE RESULTAT RENVOYE PAR LA FONCTION
100 ' IMPRESSION DU RESULTAT
110 LPRINT "LONGUEUR DU PREMIER COTE X = ";X
120 LPRINT "LONGUEUR DU SECOND COTE Y = ";Y
130 LPRINT "LONGUEUR DE L'HYPOTHENUSE H = ";H
140 LPRINT
150 GOTO 50     ' RETOUR POUR UN NOUVEAU CALCUL
160 END
```

```
LONGUEUR DU PREMIER COTE     X = 2
LONGUEUR DU SECOND COTE      Y = 2
LONGUEUR DE L'HYPOTHENUSE    H = 2.82843

LONGUEUR DU PREMIER COTE     X = 4
LONGUEUR DU SECOND COTE      Y = 3
LONGUEUR DE L'HYPOTHENUSE    H = 5
```

Un ordinateur sous le capot

L'avènement des microprocesseurs et leur perfectionnement constant résolvent peu à peu tous les obstacles à l'installation, sur une automobile, d'un ordinateur de dimensions compatibles avec les exigences pratiques, et dont la puissance de calcul permet de centraliser l'automatisation complète de toutes les fonctions du véhicule. Le véritable problème est celui de « l'interface » entre l'automobile et son ordinateur.
Afin d'accomplir toutes les fonctions qui lui sont propres, le processeur doit recevoir de façon précise et rapide, les informations, en provenance de l'extérieur, qui sont nécessaires à son intervention finale. Cependant, la variété des tâches à réaliser à bord d'un véhicule et la quantité des paramètres qui les commandent, rendent très complexe le fonctionnement de l'ordinateur.

Un microprocesseur gérant un système d'injection de carburant, par exemple, recevra, afin de fournir un mélange optimal, des informations concernant la vitesse du moteur, son couple, la position des soupapes d'admission, la température et la pression de l'air. Pour chacune de ces valeurs, un ***capteur spécifique*** *sera relié au microprocesseur ; si la conception de ces capteurs n'est pas toujours compliquée – une simple came suffit pour enregistrer la vitesse –, il est évident que leur nombre s'accroît avec l'augmentation des fonctions à contrôler.*
De plus, les informations captées (signaux) doivent être envoyées à l'ordinateur sous la forme adéquate ; autrement dit, toute impulsion enregistrée par le capteur (variation d'une résistance électrique, signaux émis par une cellule photo-électrique, ou toute autre forme d'impulsion) doit être convertie en signal numérique (digital).
Or, l'information issue des capteurs se présente presque toujours sous forme analogique : elle varie en même temps que la quantité mesurée. On obtient ainsi un signal continu qui devra être transformé (échantillonné) en une suite d'informations discrètes (séparées) acceptables par l'ordinateur. Toutefois, avec les techniques actuelles, cette conversion analogique/numérique ne pose plus de problème.

Reste la question de l'interface de sortie, chargée de traduire les décisions de l'ordinateur en commandes numériques appliquées à la voiture. C'est ici qu'une des caractéristiques (sa consommation énergétique très faible) du microprocesseur devient un gros handicap : un signal d'une minuscule fraction d'ampère doit actionner une volumineuse soupape d'alimentation en carburant ou agir sur les freins avec une force suffisante pour arrêter la voiture : il est évident que dans ces cas, le signal doit être considérablement amplifié. Comme en Hi-Fi, on réalise l'amplification en utilisant un petit signal pour en piloter un plus grand. Dans un système en circuit fermé, des oscillations indésirables peuvent alors survenir : une correction, contrebalançant une erreur sans la bloquer et avec un certain retard, peut engendrer une rectification excessive et donc une erreur en sens inverse. Avec un amplificateur Hi-Fi, il se produit alors un bruit aigü ; sur un dispositif de freinage, cela pourrait se traduire par un accident mortel.

Même avec une commande correcte et des oscillations bien amorties, de tels risques sont toujours à redouter. Actuellement, par exemple, les ***freins*** *des voitures sont souvent commandés par des systèmes hydrauliques très sensibles aux oscillations. De plus, dans ce dernier cas, l'interface destinée au contrôle du freinage devra non seulement comprendre un amplificateur, mais aussi un système transformant une impulsion électrique en commande hydraulique.*
Il est évident que les interfaces représentent actuellement le point faible de l'informatisation des voitures, alors que le problème du microprocesseur est déjà résolu : c'est sur la fiabilité des interfaces que porte à l'heure actuelle la majeure partie des recherches dans ce domaine.

Pour ces différentes raisons, l'application des microprocesseurs à l'automatisation des voitures s'est jusqu'ici restreinte à quelques fonctions particulières. Les constructeurs automobiles ont implanté des microprocesseurs dans les modèles les plus sophistiqués de leur gamme, afin d'améliorer leurs performances ou, plus commercialement, à titre d'argument publicitaire. On a assisté de ce fait

à l'informatisation de trois fonctions principales : l'alimentation en carburant, l'allumage et l'équipement du tableau de bord.
Aujourd'hui, certains modèles récents comportent un microprocesseur auquel sont transmises les données telles que la vitesse du moteur, son couple et la position du piston (obtenue à partir de celle de l'arbre à came) ; ces informations constitueront les paramètres d'entrée des programmes stockés initialement dans la mémoire ROM du microprocesseur. Une suite de comparaisons lui permettra de déterminer le moment d'**allumage** optimal, et une interface transmettra les commandes appropriées. Il a déjà été mis en évidence qu'un tel dispositif électronique améliore les performances du moteur, tout en réduisant l'émission des **gaz d'échappement**. Volkswagen, entre autres, a démontré qu'il est possible de maintenir le moteur en régime minimal, en agissant sur la mise en phase.
Toutefois, ces systèmes ne constituent qu'un point de départ : les possibilités d'optimiser les **commandes** par microprocesseur ne se bornent pas là. Depuis les années 1970, certains constructeurs automobiles ont introduit un **capteur de détonation**, afin de rendre les moteurs plus performants en augmentant leur **taux de compression**. A l'heure actuelle, il est rare que les moteurs à essence aient un taux de compression supérieur à 10:1, taux d'ailleurs souvent limité par la teneur en plomb du carburant ; par ailleurs, si ce taux était plus élevé, le moteur serait sujet à **détonation**. Le capteur de détonation est prévenu avant qu'elle ne se produise, par une variation anormale de la vitesse de l'arbre à came, par exemple ; il retarde alors l'allumage. Si l'utilisation du microprocesseur permet déjà une économie considérable de carburant, un taux de compression très élevé et une émission de gaz très peu polluants, il semble que peu de choses puissent être encore améliorées au niveau du moteur lui-même. On reconnaît cependant au microprocesseur des capacités potentielles d'une plus grande portée. Les **soupapes**, par exemple, constituent un vaste champ d'application de la technologie électronique. La société Lucas-CAV a déjà réalisé, en Grande-Bretagne, des soupapes actionnées par des solénoïdes ; depuis 1980, il existe aux Etats-Unis un prototype de moteur doté de soupapes hydrauliques.
Le fonctionnement électronique des soupapes offre de nombreuses possibilités : le conducteur pourrait modifier à volonté les caractéristiques du moteur, en modulant la conduite à l'aide d'un commutateur provoquant une simple variation de la mise en phase et de la durée d'ouverture des soupapes ; inversement, le conducteur pourrait actionner la fermeture simultanée des soupapes, transformant ainsi le moteur en un compresseur assurant un freinage plus efficace. Ce système, selon toute probabilité, permettrait d'améliorer les possibilités de n'importe quel microprocesseur.

Mais le moteur ne constitue pas la seule cible dans un véhicule. Un autre domaine sollicite l'imagination des ingénieurs mécaniciens : c'est celui des transmissions. On prévoit en effet que dans un avenir proche, le gain de consommation obtenu par avertissement électronique de la transmission sera supérieur à celui qu'on peut attendre des microprocesseurs adaptés au bloc moteur. En concevant un système de ce genre, on programmerait les caractéristiques du moteur dans le microprocesseur, ce qui modifierait automatiquement le **rapport de transmission** en l'adaptant à la vitesse en cours et aux conditions de charge.
Pour confirmer définitivement l'intérêt de cette technique, il faudra probablement attendre les résultats des recherches dans le domaine de la **transmission continue** (CVT). Quoiqu'il en soit, le contrôle par microprocesseur permettra une adaptation plus précise du rapport de transmission en fonction de la vitesse et de la puissance.

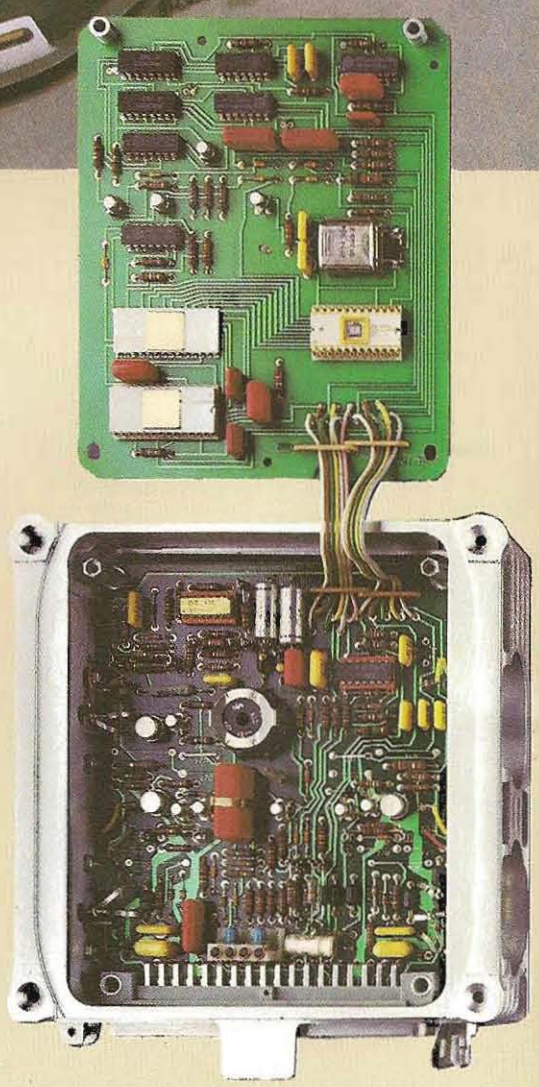
Néanmoins, la présence de l'électronique à bord des véhicules « haut de gamme » se manifeste essentiellement sur le tableau de bord : l'apparition en 1976 de la Lagonda, conçue par l'Aston Martin Company, munie de capteurs intégrés (solid state sensor) et d'un tableau de visualisation numérique, a marqué le début d'un engouement pour les véhicules équipés d'un tableau de bord digne d'un scénario de science-fiction. Ces cadrans n'offrent aucun avantage intrinsèque : une représentation « analagique » traditionnelle indique aussi bien, sinon mieux, la

vitesse, le nombre de tours du moteur ou le niveau du carburant. Reconnaissons tout de même cette tentative comme un pas vers la vulgarisation de concepts tels que l'ergonomie et l'optimisation de la conduite. Quelques constructeurs, d'ailleurs, proposent déjà un contrôle continu de la consommation de carburant et de la vitesse moyenne.
Les concepteurs suggèrent de projeter des informations digitales directement sur le pare-brise, afin que le conducteur n'ait jamais à détourner son regard de la route. Si des indicateurs frontaux de ce type (HUD) sont déjà employés depuis quelques années sur les avions de chasse, leur coût de fabrication et les problèmes liés à leur miniaturisation n'ont jusqu'ici pas permis de les transposer aux véhicules de tourisme ; de plus, on n'a pas encore résolu la question de l'éclairage adéquat pour rendre ces cadrans facilement lisibles même par temps ensoleillé.

Un autre domaine, qui n'a pas retenu l'attention méritée, regroupe les systèmes de freinage et de suspension ; toutefois, BMW et Mercedes offrent actuellement sur leurs modèles de haut de gamme un dispositif de freinage antibloquant. Si l'on considère que ces systèmes équipent les avions depuis plus de vingt ans, on peut s'étonner que leur application au domaine automobile ait nécessité tout ce temps : la force d'adhérence qui s'applique aux roues d'un avion est beaucoup plus importante que dans le cas d'un véhicule léger. Le capteur hydromécanique et le système de contrôle des freins devront donc être plus précis et plus rapides (plage de manœuvre plus réduite) dans le cas d'une voiture.
Sur un système électronique anti-blocage, le capteur mesure la force de décélération appliquée à la roue : si elle est trop forte, la roue risque de se bloquer ; le système commande alors un relâchement rapide des freins, puis, aussitôt après, leur réactivation.
Certains ingénieurs pensent que cette technique d'asservissement des freins, à travers un système de tubes hydrauliques, serait transposable à la suspension. Quelques systèmes ont d'ailleurs été mis au point, qui empêchent la voiture d'osciller dans les virages ; mais une solution électronique serait plus efficace. La limitation majeure des techniques traditionnelles de suspension automobile tient au type même de ressort utilisé. Un ressort idéal devrait agir graduellement : ayant une souplesse importante en début de course, de façon à absorber de petites oscillations, il devrait accroître peu à peu sa rigidité, afin de répondre aux fortes sollicitations dont il devient l'objet. Difficile à réaliser avec des ressorts mécaniques, cela devient possible avec des ressorts à forte pression (air liquide) ; un tel système, muni d'un microprocesseur contrôlant une soupape d'évacuation ou de pompage de l'air pourraient assurer une excellente suspension.

Avec l'apparition des microprocesseurs, qui connaissent une importance croissante dans le contrôle de fonctions spécifiques – alimentation en carburant, émission des gaz d'échappement, allumage, transmission, freinage, suspension, direction et instruments de bord –, on peut désormais envisager de réunir tous ces dispositifs, indépendants à l'origine, sous le contrôle d'un ordinateur central. Il y a quelques années, la firme allemande Bosch commercialisa un système de contrôle (Motronic) qui fut installé sur la BMW 732i. L'intérêt du Motronic est qu'il possède un unique processeur pour gérer à la fois l'allumage, l'alimentation en carburant, et de nombreuses autres fonctions optionnelles, grâce à ses interfaces et ses possibilités de programmation externe.
A ses débuts, l'électronique automobile avait tendance à utiliser des processeurs simples, destinés à des fonctions spécifiques. Le VLSI, au contraire, bien qu'encore coûteux, s'avère universel et ouvre le chemin vers une centralisation totale. Un microprocesseur multifonction peut, en effet, être programmé pour contrôler l'allumage, changer de vitesse, allumer automatiquement les phares, etc. En d'autres termes, un seul VLSI accomplira toute une série de tâches, qui requéraient auparavant autant de processeurs spécifiques.
Toutes les fonctions étant asservies par un ordinateur central, il devient plus facile pour une automobile de réagir à la fois rapidement et de manière globale aux stimuli externes, tels que la densité de circulation ou les virages. On pourrait même doter le véhicule d'un radar à micro-ondes relié à l'ordinateur ; la position et la vitesse des autres véhicules,





Maquette de tableau de bord comprenant un cadran d'indication à capteur intégré (à cristaux liquides) et visualisation par réflexion : les données apparaissent sur le pare-brise. A droite : vue intérieure de l'ordinateur de bord.

la présence des piétons, seraient rigoureusement évaluées, et le microprocesseur (même dans le cas d'un brouillard épais), activerait automatiquement le système de freinage, de façon à respecter la distance de sécurité ou à stopper le véhicule si nécessaire. En outre, le radar pourrait donner au conducteur la possibilité d'effectuer un dépassement sur route sinueuse, en permettant de « voir » au-delà des virages.

Dès la fin des années 1970, des constructeurs allemands ont expérimenté des systèmes à ordinateur de bord recueillant, depuis la route, les informations sur les conditions de circulation et les parcours conseillés, pour les visualiser sur un cadran à l'intérieur de l'automobile. Il n'est pas utopique de penser que de tels systèmes de « conduite assistée par ordinateur » seront commercialisés dans un avenir proche.

Smiths Industries Ltd.

Par exemple, la fonction :

```
DEF FNA$(C$,N) = LEFT$(C$,N) + D$
```

prélève les N caractères les plus à gauche de C$ et y ajoute la chaîne D$. On trouvera ci-dessous et page 482 l'organigramme et le listage d'un programme utilisant la fonction ainsi définie.

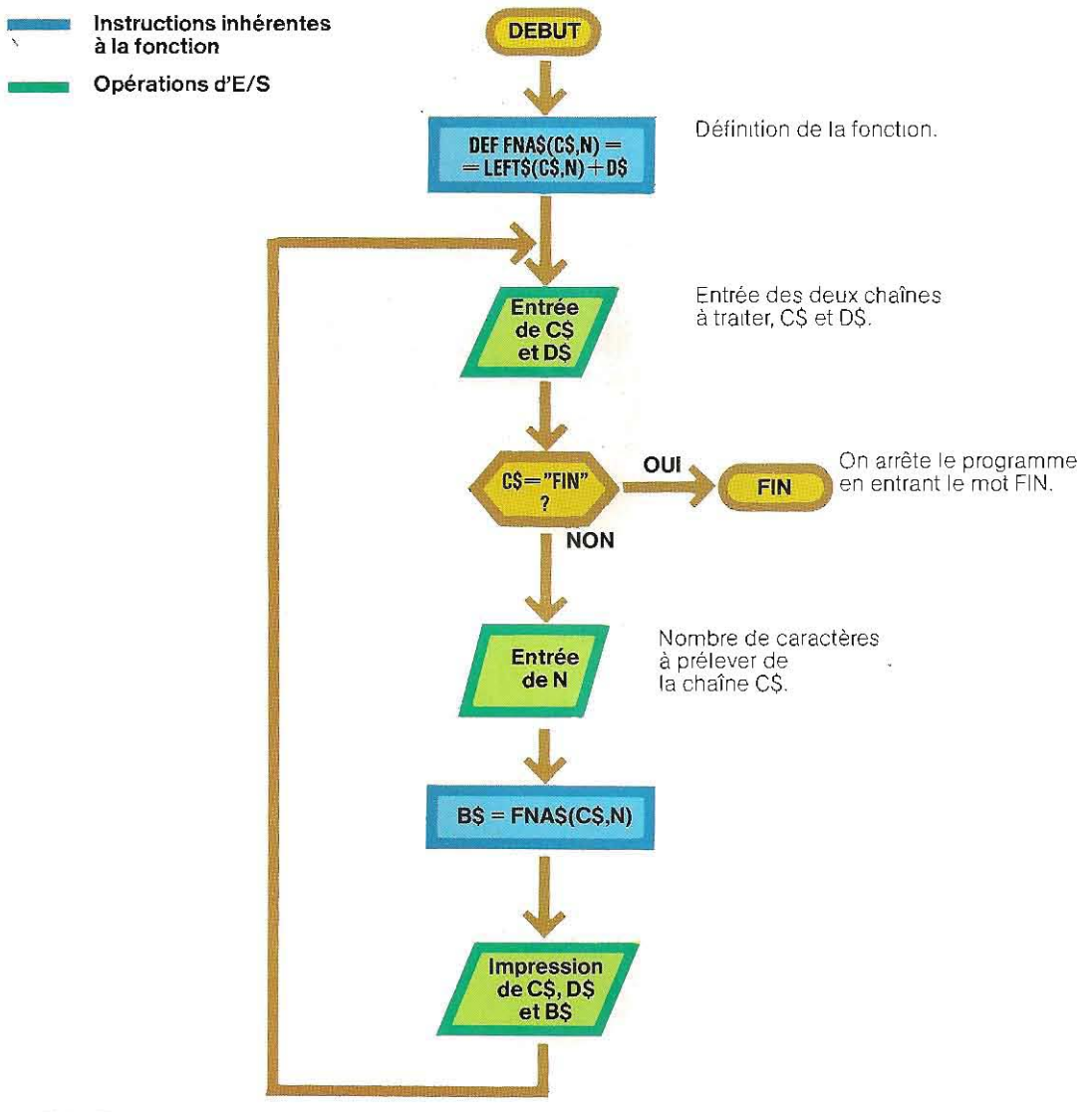
Nous allons maintenant développer deux applications d'usage général relatives aux fonctions numériques, d'une part, et au traitement des chaînes, d'autre part.

Fonctions numériques : résolution d'équations du second degré. Ces équations se présentent sous la forme :

$$A \times X^2 + B \times X + C = 0 \quad .$$

où A, B et C sont des paramètres fixes. Quelle que soit la valeur de ces paramètres, il existe toujours deux « racines » (valeurs particulières de X) pour lesquelles la partie gauche

TRAITEMENT DE CHAINES PAR UNE FONCTION PREDEFINIE



PROGRAMME DU TRAITEMENT DE CHAINES

```
10 ' ** PROGRAMME DE DEMONSTRATION : FONCTION DE TRAITEMENT DE CHAINES **
20 '
30 '                          PROGRAMME = FN/CHAIN
40 '
50 DEF FNA$(C$,N) = LEFT$(C$,N)+D$
60 PRINT "TAPEZ LES DEUX CHAINES (C$ et D$) A TRAITER"
70 PRINT
80 PRINT "POUR SORTIR DU PROGRAMME TAPEZ - FIN -"
90 INPUT "C$ = ";C$
100 INPUT "D$ = ";D$
110 IF C$ = "FIN" GOTO 280
120 PRINT "TAPEZ LE NOMBRE DE CARACTERES A PRELEVER DANS C$"
130 INPUT "N = ";N
140 B$ = FNA$(C$,N)
150 ' La ligne 140 affecte à la variable B$ le résultat
160 ' de la fonction définie en ligne 50
170 '
180 ' INSTRUCTIONS D'IMPRESSION
190 '
200 LPRINT "CHAINES A TRAITER (C$ et D$)"
210 LPRINT C$
220 LPRINT D$
225 LPRINT "NOMBRE DE CARACTERES PRELEVES DANS C$ = ";N
230 LPRINT
240 LPRINT "CHAINE TRAITEE"
250 LPRINT B$
260 LPRINT
270 GOTO 60        ' RETOUR POUR UN NOUVEAU CALCUL
280 END
```

```
CHAINES A TRAITER (C$ et D$)
ALPHABETIQUE
NUMERIQUE
NOMBRE DE CARACTERES PRELEVES DANS C$ = 5

CHAINE TRAITEE
ALPHANUMERIQUE
```

de l'égalité s'annule effectivement. Ces valeurs sont :

$$X1 = (-B+\sqrt{B^2-4\times A\times C})/(2\times A)$$
$$X2 = (-B-\sqrt{B^2-4\times A\times C})/(2\times A)$$

Le programme calcule les racines en utilisant deux fonctions définies selon ces formules.

```
FNX1(A,B,C)=(-B+SQR(B^2-4*A*C))/(2*A)
FNX2(A,B,C)=(-B-SQR(B^2-4*A*C))/(2*A)
```

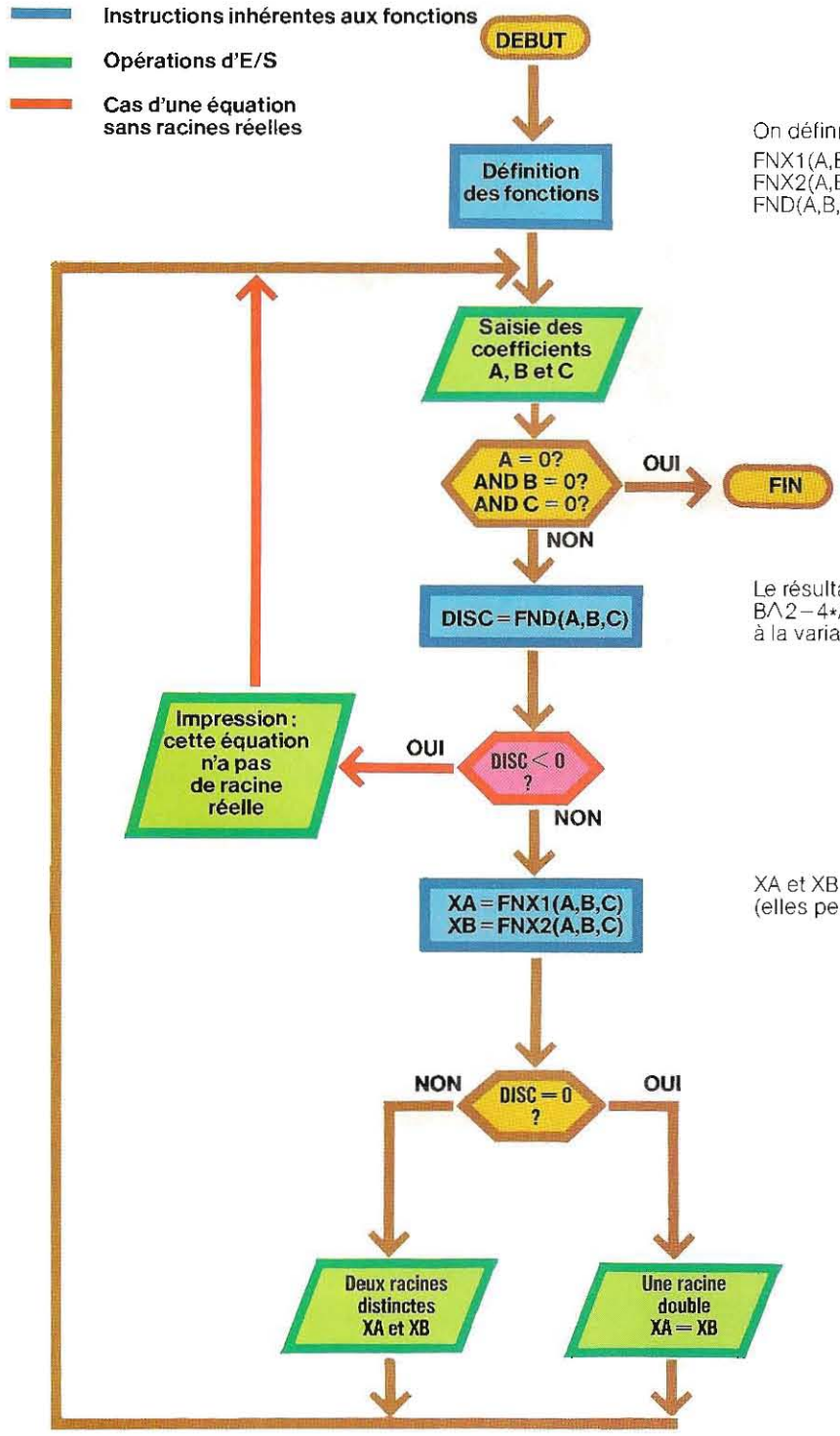
L'utilisation de l'instruction SQR peut toutefois poser un problème quand son argument prend une valeur négative. En effet, le signe de l'expression : $B^2-4\times A\times C$ varie selon les valeurs de A, B et C.
Cette expression, appelée **discriminant** de l'équation, nous oblige donc à distinguer trois cas :

premier cas : le discriminant est positif et sa racine carrée (SQR) appartient à l'ensemble des réels. Il existe alors deux racines à l'équation ;
deuxième cas : le discriminant est nul (et sa racine carrée aussi). Les deux racines sont donc égales (on parle de « racine double ») ;
troisième cas : le discriminant est négatif et sa racine carrée est un nombre complexe et non réel. X1 et X2 n'existent donc pas dans l'ensemble des réels.
Avant d'effectuer les opérations calculant les racines de l'équation, on s'assurera donc de la valeur du discriminant pour savoir dans quel cas on se trouve. La fonction à définir pour déterminer cette valeur est :

```
DEF FND(A,B,C) = B^2-4*A*C
```

L'organigramme de ce programme est présenté page 483 et son listage page 484.

RESOLUTION D'EQUATIONS DU SECOND DEGRE



On définit les fonctions :
FNX1(A,B,C), première racine ;
FNX2(A,B,C), deuxième racine ;
FND(A,B,C), discriminant.

Le résultat du calcul B^2−4*A*C est affecté à la variable DISC.

XA et XB sont les deux racines (elles peuvent être égales).

RESOLUTION D'EQUATIONS DU SECOND DEGRE

```
10 ' ** RESOLUTION D'EQUATIONS DU SECOND DEGRE **
20 '
30 '                    PROGRAMME = EQUAT. 2
40 '
50 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
60 DEF FNX1(A,B,C)=(-B+(SQR(B*B-4*A*C)))/2*A)
70 DEF FNX2(A,B,C)=(-B-(SQR(B*B-4*A*C)))/2*A)
80 DEF FND(A,B,C)=B*B-4*A*C
90 '
100 PRINT "ENTREZ LES COEFFICIENTS DE L'EQUATION (A, B et C)"
110 INPUT "A=";A
120 INPUT "B=";B
130 INPUT "C=";C
140 '
150 IF A=0 AND B=0 AND C=0 GOTO 460        ' EQUATION NULLE POUR SORTIR
160 DISC = FND(A,B,C)          ' On affecte à la variable DISC le résultat
170 '                           de la fonction FND(A,B,C) qui calcule
180 '                           le discriminant
190 IF DISC<0 THEN GOTO 451
200 XA=FNX1(A,B,C)          ' XA et XB SONT LES DEUX RACINES
210 XB=FNX2(A,B,C)
220 IF DISC=0 THEN GOTO 240 ELSE GOTO 400
230 '
240 ' PREMIER CAS: DISCRIMINANT=0, LES DEUX RACINES SONT IDENTIQUES
250 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
260 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
270 LPRINT "DISCRIMINANT=0 : IL Y A DEUX RACINES REELLES EGALES"
280 LPRINT "XA = XB = ";XA
290 LPRINT : LPRINT
300 GOTO 100                    ' RETOUR POUR UN NOUVEAU CALCUL
400 '
410 ' DEUXIEME CAS : DISCRIMINANT POSITIF, IL Y A DEUX RACINES REELLES ET DISTINCTES
415 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
420 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
422 LPRINT "DISCRIMINANT = ";DISC
424 LPRINT "DISCRIMINANT POSITIF : IL Y A DEUX RACINES REELLES ET DISTINCTES
426 LPRINT
428 LPRINT "RACINES"
430 LPRINT "XA = ";XA
440 LPRINT "XB = ";XB
445 LPRINT : LPRINT
450 GOTO 100                    ' RETOUR POUR UN NOUVEAU CALCUL
451 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
452 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
453 LPRINT "DISCRIMINANT = ";DISC
454 LPRINT "L'EQUATION N'A PAS DE SOLUTION DANS L'ENSEMBLE DES REELS"
455 LPRINT "ELLE ADMET POUR SOLUTION DES NOMBRES COMPLEXES"
456 LPRINT : LPRINT
457 GOTO 100
460 END
```

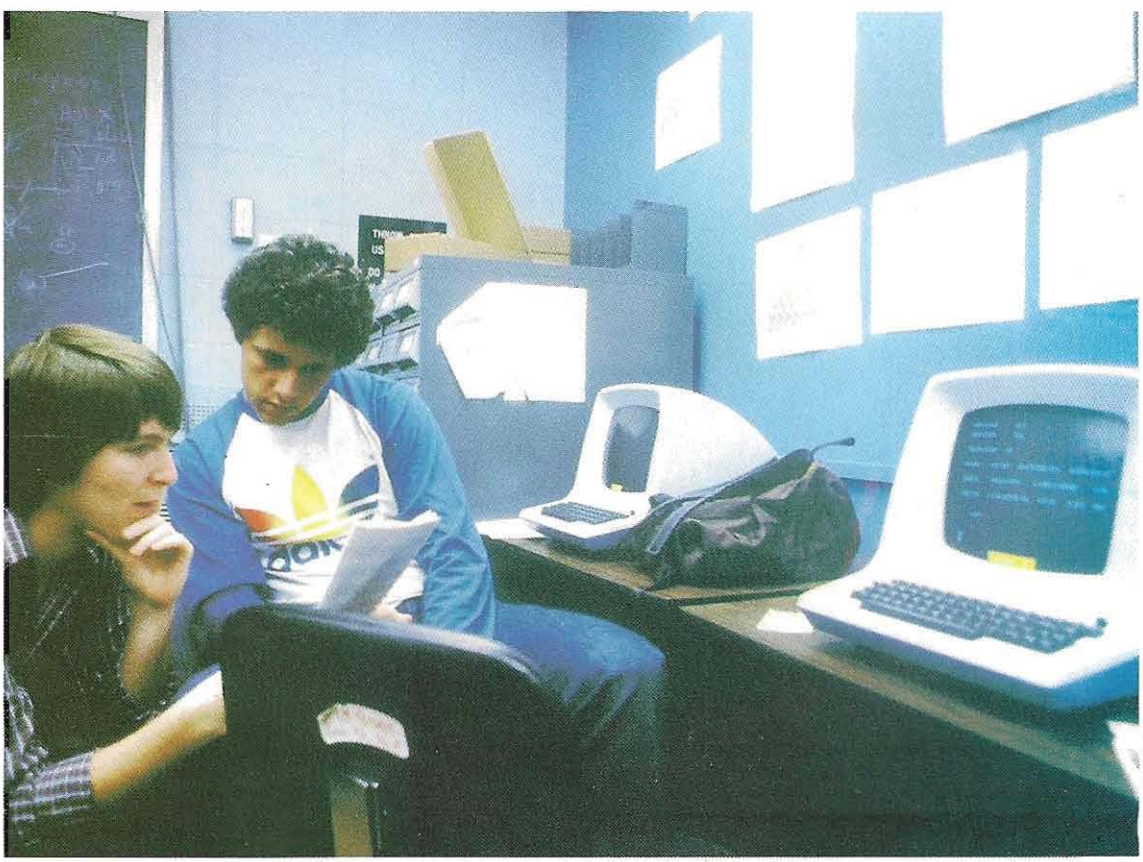
```
COEFFICIENTS DE L'EQUATION AX²+BX+C
A = 2       B = 3       C = 60      ''

DISCRIMINANT = -471
L'EQUATION N'A PAS DE SOLUTION DANS L'ENSEMBLE DES REELS
ELLE ADMET POUR SOLUTION DES NOMBRES COMPLEXES


COEFFICIENTS DE L'EQUATION AX²+BX+C
A = 3       B = -10       C = 1

DISCRIMINANT = 88
DISCRIMINANT POSITIF : IL Y A DEUX RACINES REELLES ET DISTINCTES

RACINES
XA = 3.23014
XB = .103195
```

Fonctions sur chaînes: contrôle des minuscules et des majuscules. Certains programmes traitent principalement des textes ou des fragments de textes (fichiers). L'usage des majuscules en début de phrase ou de nom propre peut alors être impératif. Un sous-programme assurant le contrôle et l'inversion éventuelle (majuscule/minuscule) permet de soulager l'opérateur lors de l'entrée des données. C'est la machine elle-même qui transforme les lettres lors de la frappe.
On trouvera pages 486 et 487 l'illustration (organigramme et listing) d'un tel programme, qui pourrait prendre place dans un traitement de fichiers d'adresses, par exemple.
Les fonctions utilisées sont:

FNP$(I) = MID$(A$,I,1)
(extraction du premier caractère de la chaîne A$)

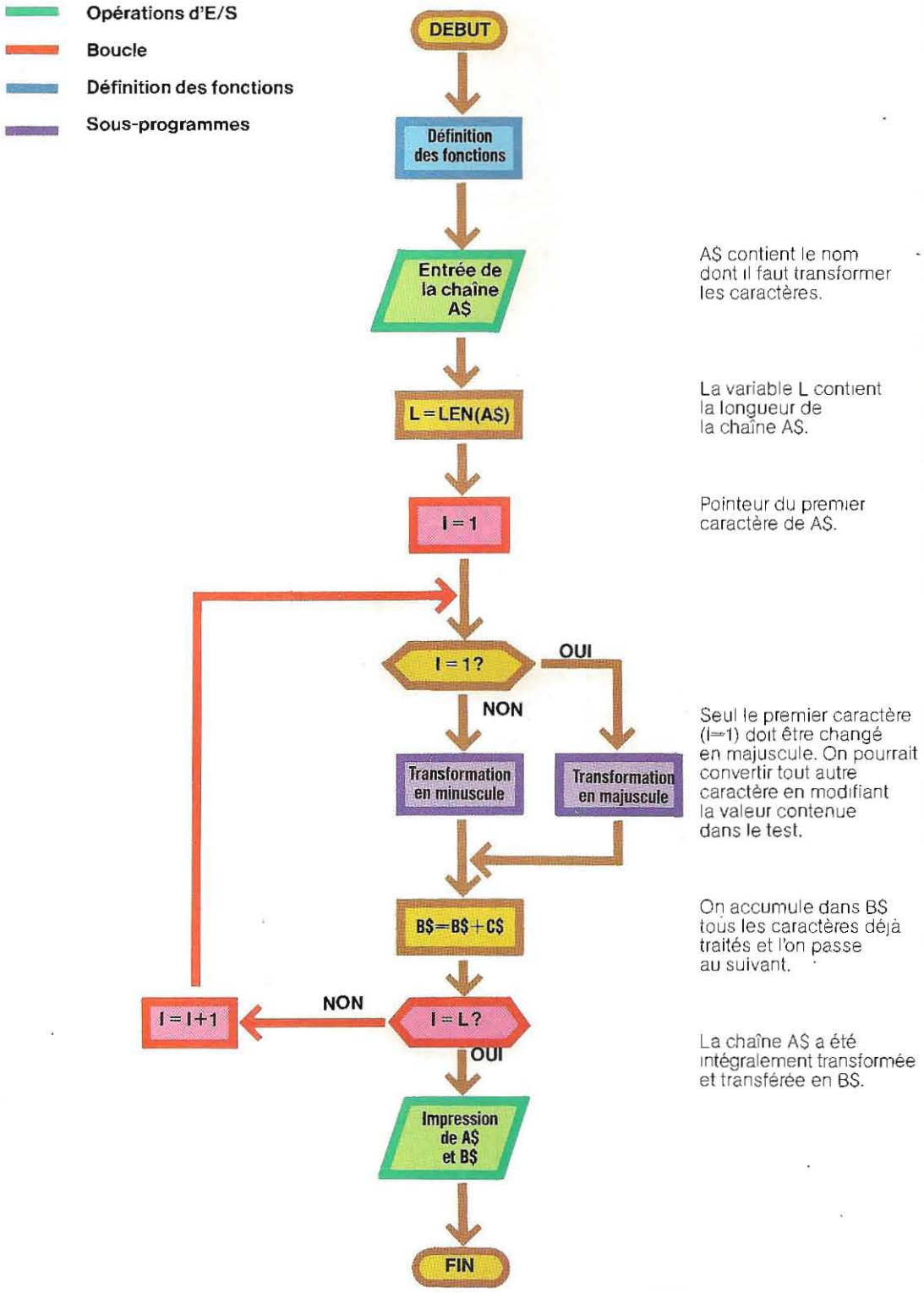
FNV$(V) = CHR$(V – 32)
(passage de minuscule en majuscule)

Aux Etats-Unis, l'enseignement fait de plus en plus appel à l'ordinateur. Sur la photo, des étudiants de la Georgetown University.



J. Pickerell/Marka

PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES



PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES

```
10 '
20 ' ** PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES **
30 '
40 '                PROGRAMME = MAJ/MIN
50 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
70 DEF FNP$(I)=MID$(A$,I,1)     ' EXTRACTION DU CARACTERE DE RANG(I)
80 '                              DANS LA CHAINE A$
90 DEF FNV$(V)=CHR$(V-32)       ' PASSAGE DE MINUSCULE EN MAJUSCULE
100 DEF FNZ$(V)=CHR$(V+32)      ' PASSAGE DE MAJUSCULE EN MINUSCULE
110 '
120 INPUT "TAPEZ LA CHAINE A TRAITER ";A$
130 L=LEN(A$)           ' LA VARIABLE L CONTIENT LA LONGUEUR DE LA CHAINE A$
140 FOR I=1 TO L
150 IF I=1 THEN GOSUB 1000 ELSE GOSUB 2000
160 '
170 ' LES ROUTINES DE TRANSFORMATION EN MAJUSCULE ET EN MINUSCULE
180 ' COMMENCENT RESPECTIVEMENT EN 1000 ET EN 2000
190 '
200 B$=B$+C$            ' UNE FOIS TRAITES, LES CARACTERES SONT STOCKES
210 '                     UN PAR UN DANS B$
220 NEXT I
230 '
240 ' INSTRUCTIONS D'IMPRESSION
250 LPRINT "CHAINE DE DEPART : A$ = ";A$
260 LPRINT "CHAINE TRAITEE : B$ = ";B$
270 LPRINT              ' SAUT DE LIGNE
280 END
1000 ' ROUTINE DE TEST ET DE TRANSFORMATION EVENTUELLE EN MAJUSCULE
1010 '
1020 C$=FNP$(I)         ' Le premier caractère de la chaîne
1030 '                    est mémorisé dans C$
1040 V=ASC(C$)          ' La valeur numérique du caractère C$ en code
1050 '                    ASCII est affectée à V
1060 IF V=>65 AND V=<90 THEN RETURN     ' Le caractère est déjà en majuscule.
1062 '                                    Pas de transformation à faire.
1070 IF V=>97 AND V=<122 THEN C$=FNV$(V)
1080 RETURN             ' SI V n'est pas compris entre 97 et 122, le caractère
1090 '                    n'est pas une lettre et ne peut donc être converti
2000 ' ROUTINE DE TEST ET DE TRANSFORMATION EVENTUELLE EN MINUSCULE
2010 C$=FNP$(I)
2020 V=ASC(C$)
2030 IF V=>97 AND V=<122 THEN RETURN    ' Le caractère est déjà en minuscule.
2040 '                                    Pas de transformation à faire.
2050 IF V=>65 AND V=<90 THEN C$=FNZ$(V)
2060 RETURN
```

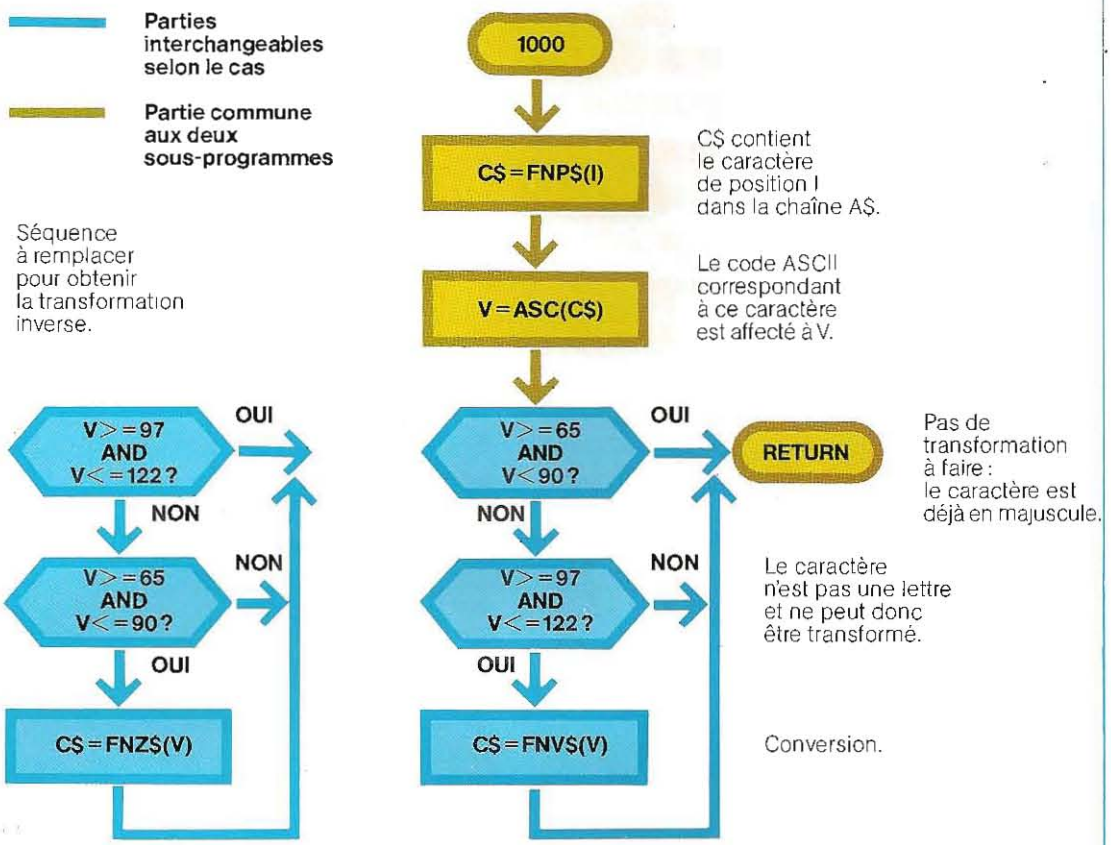
```
CHAINE DE DEPART : A$ = boulevard
CHAINE TRAITEE   : B$ = Boulevard

CHAINE DE DEPART : A$ = HAUSSMANN
CHAINE TRAITEE   : B$ = Haussmann

CHAINE DE DEPART : A$ = 75008
CHAINE TRAITEE   : B$ = 75008

CHAINE DE DEPART : A$ = paris
CHAINE TRAITEE   : B$ = Paris
```

FNZ$(V) = CHR$(V+32)
(passage de majuscule en minuscule).

Le sous-programme de transformation de minuscule en majuscule est schématisé page 488, avec la séquence permettant la transformation inverse. Ci-dessus apparaît le listage complet de cette application.

SOUS-PROGRAMME DE TRANSFORMATION EN MAJUSCULE

Entrée A$ = Chaîne contenant en position I le caractère à transformer.



Domaines d'utilisation. La possibilité, pour l'utilisateur, de définir ses propres fonctions est particulièrement intéressante dans deux secteurs : les applications scientifiques, d'une part, et les manipulations de données nécessitant de nombreuses saisies au clavier, d'autre part. Dans ce dernier cas, on peut créer à l'écran des « masques de saisie », mais ceci réclame une programmation plus élaborée et sera détaillé dans le chapitre concernant la gestion de l'écran.

Nous allons maintenant utiliser les fonctions numériques pour un calcul de surface. La méthode employée, quoique satisfaisante, n'est pas d'une grande rigueur, mais elle illustre bien les applications possibles de l'ordinateur en mathématiques appliquées.

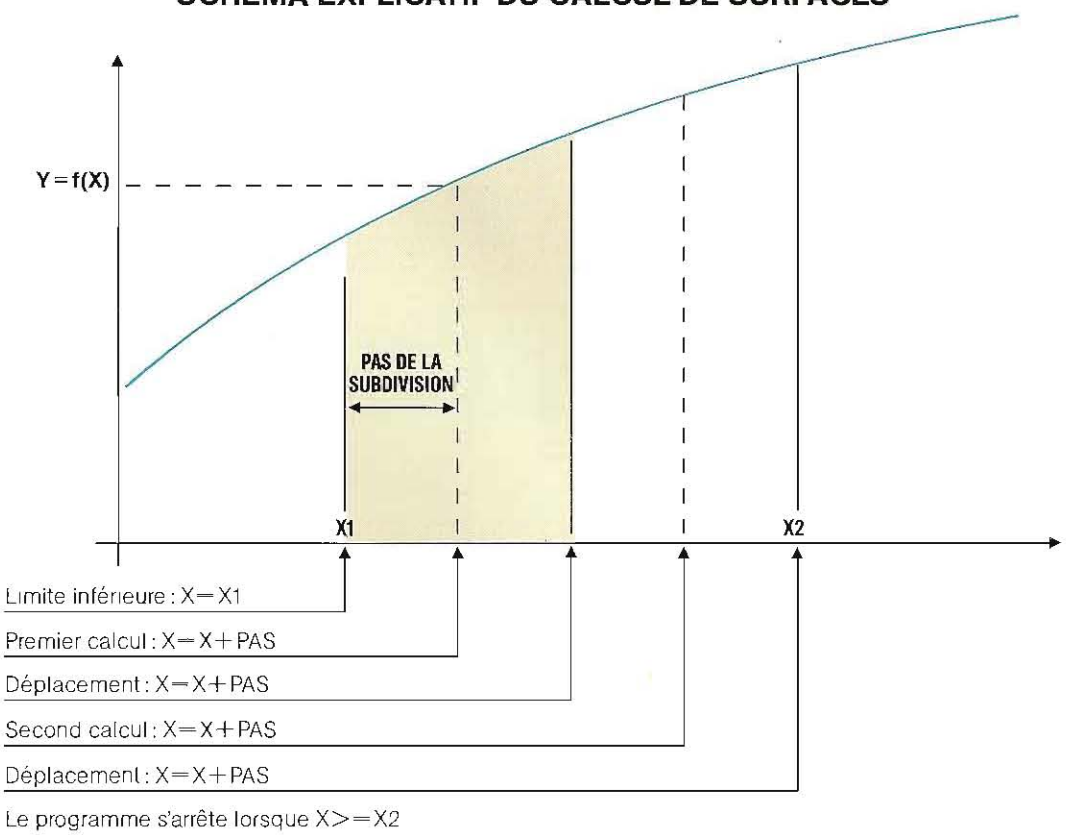
Ce problème a déjà été résolu graphiquement pages 333 et 334. Il s'agissait de décou-

per une surface en petites parties dont on fournissait les coordonnées à l'ordinateur. Celui-ci effectuait alors le calcul de ces surfaces « élémentaires » et les additionnait entre elles. Si l'on connaît l'équation de la courbe (fonction), l'ordinateur peut alors calculer, lui-même et plus précisément, les coordonnées. Il ne reste donc plus qu'à fixer les bornes de l'intervalle et le nombre de subdivisions désiré.

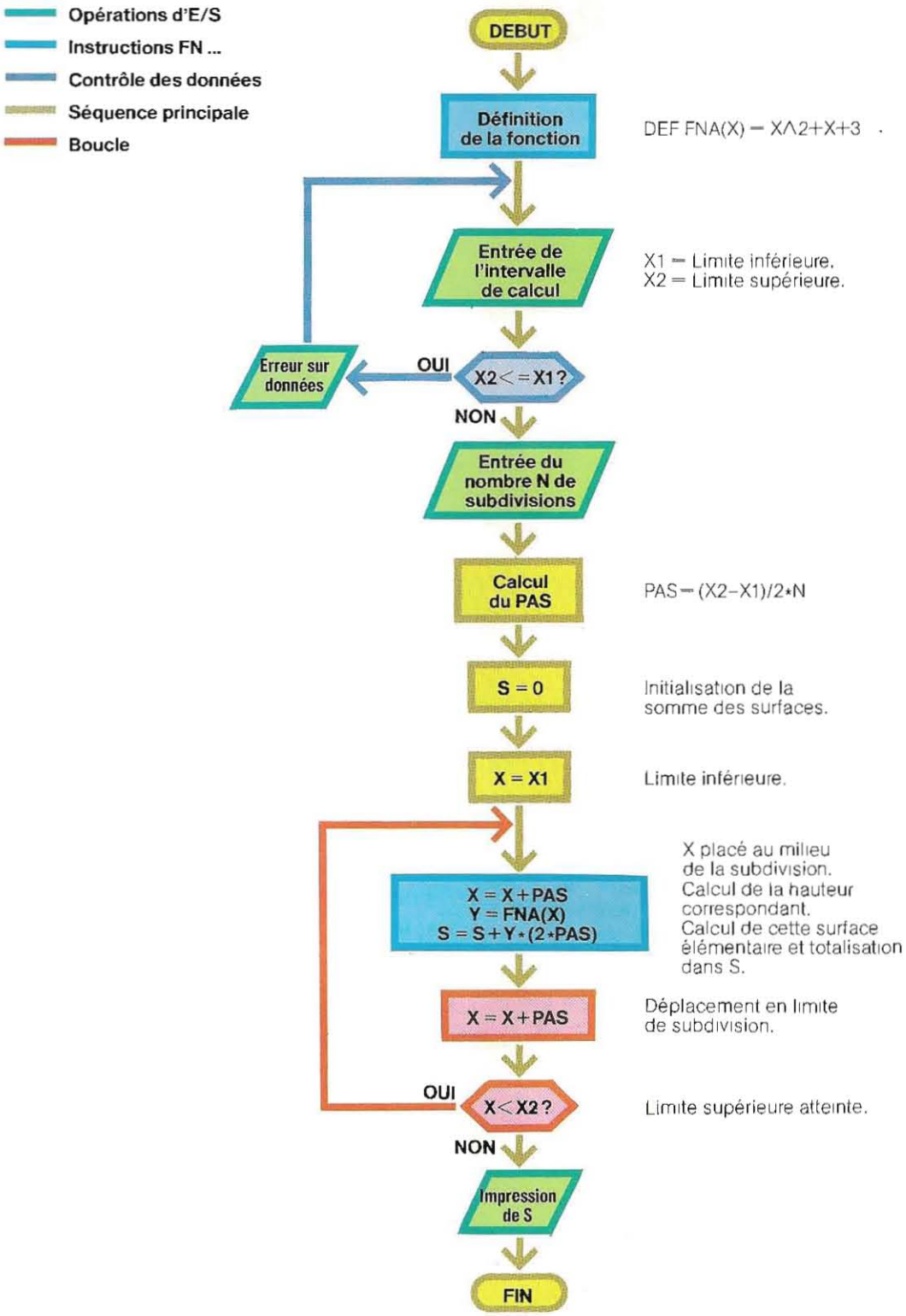
Le listage de la page 491 est accompagné des résultats obtenus pour différents nombres de subdivisions sur l'intervalle [5,25]. La précision du calcul augmente avec ce nombre jusqu'à une limite (80 dans cet exemple) à partir de laquelle l'approximation est suffisante. Il est très difficile de déterminer cette limite a priori. On effectuera donc plusieurs essais afin d'évaluer la précision des résultats et de fixer la valeur qui réalise le meilleur compromis entre fiabilité et durée de traitement.

Dans de tels programmes comportant des

SCHEMA EXPLICATIF DU CALCUL DE SURFACES



ORGANIGRAMME DU CALCUL DE SURFACES



PROGRAMME DE CALCUL DE SURFACES ELEMENTAIRES

```
10 '
20 ' ** PROGRAMME DE CALCUL DE SURFACES ELEMENTAIRES **
30 '
40 '                  PROGRAMME = INTEGRATION
50 '
60 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
70 DEF FNA(X)=X*X+X+3          ' Fonction d'une équation du second degré
80 PRINT "ENTREZ LES LIMITES DE LA SURFACE A CALCULER"
90 INPUT "LIMITE INFERIEURE : X1 = ";X1
100 INPUT "LIMITE SUPERIEURE : X2 = ";X2
110 IF X2=<X1 THEN PRINT "DONNEES ERRONEES":GOTO 80
120 PRINT "ENTREZ LE NOMBRE DE SUBDIVISIONS DE LA SURFACE"
130 INPUT "N = ";N
140 '    CALCUL DU PAS
150 '
160 PAS=(X2-X1)/(2*N)
170 S=0          ' MISE A ZERO DE LA SOMME DES SURFACES ELEMENTAIRES
180 X=X1
190 X=X+PAS
200 Y=FNA(X)
210 S=S+Y*(2*PAS)
220 X=X+PAS
230 IF X<X2 GOTO 190
1000 ' INSTRUCTIONS D'IMPRESSION
1002 LPRINT "NOMBRE DE SUBDIVISIONS DE LA SURFACE : ";N
1010 LPRINT "SURFACE TOTALE = ";S
1020 LPRINT
1030 END
```

```
NOMBRE DE SUBDIVISIONS DE LA SURFACE : 10
SURFACE TOTALE = 5520

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 20
SURFACE TOTALE = 5525

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 40
SURFACE TOTALE = 5526.25

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 80
SURFACE TOTALE = 5526.56

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 800
SURFACE TOTALE = 5526.69
```

calculs répétitifs, une version compilée réduit beaucoup le temps d'exécution et autorise un nombre de subdivisions plus important.

Exemple d'application : le chauffage des locaux

L'emploi de sous-programmes s'avère indispensable au-delà d'un certain degré de complexité du programme. Aborder tous les cas possibles et les calculs correspondants au sein d'un seul grand programme mène à un ensemble confus tout en multipliant les risques d'erreur.

L'utilisation de sous-programmes permet, au contraire, d'entrer progressivement dans les détails et d'appréhender plus clairement chaque cas particulier. Après une analyse globale du problème, on dessine l'organigramme principal. Puis, on envisage le développement de tous les sous-programmes annexes.

Le programme idéal comporte ainsi une partie principale, dont le rôle est d'appeler et d'enchaîner une série de sous-programmes spécifiques. Outre sa facilité d'élaboration, un tel programme présente une clarté qui permettra une relecture aisée. Cet aspect est important pour une personne autre que le concepteur, mais aussi pour celui-ci lors de

L'élève et la tortue

L'intérêt des applications éducatives de l'informatique est désormais reconnu aussi bien dans le corps enseignant que dans les milieux professionnels.

L'enseignement fondamental et les exercices pratiques assistés par ordinateur apparaissent dans les centres d'études pédagogiques, les universités et les établissements de formation, mais aussi dans les entreprises, publiques ou privées, et dans les laboratoires de recherche. Ces dix dernières années ont vu se multiplier les logiciels d'enseignement et aujourd'hui, on produit et on diffuse ces systèmes à l'échelle industrielle. De telles applications peuvent revêtir des formes très diverses selon l'orientation choisie.

■ *Certains systèmes permettent de conserver une structure traditionnelle (formation animée par des enseignants à l'école, à l'université ou dans l'entreprise). L'ordinateur n'intervient alors que comme support d'enseignement ou comme complément didactique. Généralement regroupés sous l'étiquette E.A.O. (enseignement assisté par ordinateur), ces systèmes sont l'aboutissement des recherches et des investissements engagés dès 1955 par les universités américaines. L'E.A.O. englobe plusieurs thèmes : enseignement programmé, exercices d'entraînement et de perfectionnement, résolution de problèmes, simulation, jeux éducatifs et contrôle des connaissances.*

■ *D'autres systèmes sont beaucoup plus orientés vers les moyens audiovisuels ou télévisuels et la télématique (liaison avec les banques de données et les conférences). Ils sont surtout conçus pour l'enseignement à distance grâce à des programmes spécifiques diffusés par les réseaux de télévision.*

■ *Enfin les systèmes dits « intelligents », dérivés des recherches menées dans le domaine de l'intelligence artificielle, établissent un dialogue ouvert avec l'élève. Fondés sur la mise au point de programmes simples, ils visent plus au développement de la logique et de la créativité qu'à un apprentissage particulier. Des langages élémentaires mais évolutifs les mettent à la portée des jeunes enfants ou des élèves en difficulté.*

Depuis 1970, la diffusion des ordinateurs touche, dans les pays les plus industrialisés, un nombre toujours croissant d'établissements scolaires. Ces pays mènent, à un niveau national, des politiques de soutien financier au développement de l'informatique pédagogique. Ainsi, aux Etats-Unis, le gouvernement fédéral avait investi, principalement par le biais de la National Science Foundation, près de 300 millions de dollars. A la fin des années 1960, ce budget fut confié au Ministère de l'Education.

C'est à l'instigation du Centre pour la Recherche et l'Innovation dans l'Enseignement (CERI) de l'O.C.D.E. que des pays comme l'Allemagne, la Grande-Bretagne, le Japon ou la France multiplient les expériences depuis 1970. A l'échelle de la C.E.E., c'est le Fonds Social Européen qui finance les projets pédagogiques orientés essentiellement vers la formation professionnelle.

En France, des associations comme l'ADEMIR, regroupées dans une Fédération Nationale (Microtel) et soutenues par le Ministère de l'Education et l'Agence de l'Informatique, contribuent au développement de l'informatique scolaire en créant des « bibliothèques » de didacticiels (logiciels d'enseignement).

De façon générale, il faut souligner que l'intérêt pour l'enseignement assisté par ordinateur s'accompagne aujourd'hui d'exigences plus profondes de la part d'un public qui avait pu mettre en doute les possibilités des « machines à enseigner », à l'époque où elles n'étaient qu'à l'état de projet.

Lorsqu'au début des années 1960, l'ordinateur fit son apparition dans les écoles et les universités américaines, le milieu enseignant était très attiré par des théories pédagogiques comme celle de l'enseignement programmé. La première génération des systèmes d'E.A.O. naquit, dans ce contexte, des recherches menées par Patrick Suppes et Richard Atkinson à l'Institut de mathématiques appliquées aux sciences sociales de l'Université de Stanford, par Donald Bitzer à l'Université de l'Illinois (système Plato) et par les laboratoires d'IBM (système CAI 1500). Les expériences menées dans les années 1960 furent fortement influencées par la technologie caractéristique de cette période.

Les stratégies fondées sur la notion d'enseignement programmé impliquaient un contrôle rigide du dialogue avec l'ordinateur. Toutefois, dès 1965, on vit se développer une

nouvelle approche théorique et expérimentale de l'utilisation de l'ordinateur dans l'enseignement. A la conception de l'ordinateur, simple machine à « tourner les pages » se bornant à automatiser l'enseignement, succéda une nouvelle conception dont la devise aurait pu être, selon les mots de Seymour Papert : « arrangeons-nous pour que ce soit les enfants qui programment l'ordinateur et non l'ordinateur qui programme les enfants ».
Après plusieurs années de collaboration avec le psychologue suisse Jean Piaget, Papert, spécialiste en cybernétique, assura, aux côtés de Marvin Minsky, la direction du Laboratoire d'Intelligence Artificielle du MIT. Selon lui, l'enfant n'apprend rien de l'ordinateur sinon en le programmant, ce qui l'oblige à exprimer clairement ses « intuitions ». Traduire une idée sous la forme d'un programme, c'est la concrétiser et en faire un objet de réflexion.
Cette approche, telle qu'elle ressort de la biographie de Papert, s'appuie sur les concepts de la science cognitive en matière d'intelligence artificielle, et elle a évolué dans un environnement expérimetal en même temps que les langages interactifs.
Historiquement, le Basic est considéré comme le premier langage ayant permis un enseignement basé sur la programmation. Toutefois, celui-ci se limite à des calculs numériques, du moins dans sa version classique. Des langages interactifs plus récents donnent, par contre, accès à des domaines plus variés. Ainsi, le LOGO et le SMALLTALK offrent à l'enfant des possibilités de rédaction, de graphisme ou de symbolisation.
Tout langage dispose d'un certain nombre d'instructions de base. On réalise alors un programme en introduisant une suite logique utilisant ces commandes.
Avec un langage comme le LOGO, il est possible de définir de nouvelles instructions qui pourront être utilisées dans des programmes ultérieurs.
Une instruction, correspondant à une opération complexe, peut ainsi, au terme d'une analyse attentive, être déterminée comme une suite d'opérations plus simples. Cette manière d'aborder un problème par paliers successifs de complexité décroissante est la base de toute découverte scientifique.
Connaître, c'est traduire l'inconnu par le biais d'un raisonnement logique dont on maîtrise chaque étape. Cette méthode scientifique est aussi bien adaptée à l'enseignement des matières les plus variées et ce, quel que soit l'âge de l'élève. De plus, savoir décomposer une structure complexe est une capacité indispensable à la conception et à l'accomplissement de projets à long terme.
Le SMALLTALK, pour sa part, fut mis au point par Alan Kay au Centre de recherches Xerox de Palo Alto. Tout d'abord destiné à faciliter les conditions de programmation, il fut ensuite développé à des fins didactiques et s'avère aujourd'hui très adapté à la création de dessins animés, de jeux et de compositions musicales.
A l'origine du SMALLTALK, se trouve une assimilation de toute démarche intellectuelle à la coopération et la compétition de plusieurs processus mentaux. Si l'on désire, par exemple, dessiner une figure comportant deux types de symétrie, on peut imaginer une procédure pour créer la première symétrie, puis l'essayer pendant qu'on réfléchit à la seconde. Une fois la seconde procédure écrite, on reprendra la première pour juger des résultats obtenus tandis que se déroulera la seconde. Quand tout est mis au point, l'exécution simultanée des deux procédures fournit le résultat final, c'est-à-dire le dessin désiré. Il est évident que l'utilisation de cette méthode parallèle facilite grandement la programmation de fonctions complexes.
L'emploi des langages interactifs a été expérimenté – et ce, principalement avec des enfants – par Papert et Minsky. Ils s'inspiraient des conceptions de Piaget selon lesquelles l'intelligence est une adaptation à l'environnement et l'éducation une évolution spontanée à travers différents stades intellectuels.
De cette théorie découle l'idée qu'il est nécessaire de mettre le sujet (généralement, un enfant) dans des conditions propices à toute étude. C'est ainsi que furent créés le Laboratoire d'Intelligence Artificielle du MIT et son atelier d'enseignement assisté par ordinateur où fut développé ce langage de programmation très simple qu'est le LOGO.
Toutes sortes de gens ont pu se mesurer à ce langage : des enfants de quatre ans, des classes entières d'élèves venant de régions des environs de Boston culturellement désavantagées, des lycéens surdoués, des étu-

diants en sciences habitués aux ordinateurs du MIT, des chercheurs d'universités latino-américaines (où le LOGO fut importé) et des ingénieurs d'autres laboratoires d'Intelligence Artificielle comme celui d'Edimbourg. Enfin, les chercheurs les plus brillants du laboratoire ont testé le LOGO et sa conception de l'enseignement. Les résultats de ces travaux firent l'objet de plus de deux cents rapports techniques décrivant les réalisations les plus disparates, de l'écriture automatique de poèmes à la simulation de phénomènes biologiques ou comportementaux.

Le LOGO permet de dialoguer avec l'ordinateur et de lui apprendre de plus en plus de choses en fonction de ce qu'il sait déjà.

L'ordinateur devient alors un véritable partenaire qui exécute fidèlement les ordres, à condition qu'ils soient exprimés de façon correcte. Le LOGO ne reconnaît au départ qu'un nombre limité de mots qui permettent de définir d'autres commandes. C'est un robot capable d'apprendre : on peut étendre son langage en associant, à l'aide des instructions de base, des opérations à des mots nouveaux qui seront alors ajoutés à son vocabulaire de base. Orientée au départ vers les matières traditionnelles, l'utilisation du LOGO a rapidement dépassé le cadre de ces disciplines pour parvenir à des expériences plus ouvertes exploitant toutes les possibilités de l'ordinateur.

La plus célèbre de ces expériences, également réalisée avec d'autres langages comme le SMALLTALK ou le LISP, langage de base de l'intelligence artificielle, est celle de la « Tortue ». La Tortue est un véritable petit automate téléguidé par l'intermédiaire du langage LOGO. La Tortue avance en laissant derrière elle une trace qui, au fur et à mesure de ses déplacements, réalise un dessin sur l'écran.

La Tortue est généralement représentée par un point ou un petit triangle lumineux. On peut tracer n'importe quelle figure grâce aux commandes de base qui sont des ordres très simples : AVANCE, RECULE, DROITE, GAUCHE, indicés par la longueur du trajet ou par l'angle en degrés de la rotation à effectuer. Ces commandes simples peuvent être agencées en séquences plus ou moins complexes qui finissent par constituer de véritables procédures. L'utilisateur les imagine et les crée de façon autonome, faisant appel à son intuition et à la logique, au gré d'essais plus ou moins fructueux.

Les instructions AVANCE et RECULE provoquent le déplacement de la Tortue le long de l'axe sur lequel elle se trouve.

Pour modifier cette orientation, on doit employer les commandes DROITE et GAUCHE qui font alors pivoter la Tortue sur elle-même. Ces quatre commandes s'accompagnent d'un indice correspondant soit à la longueur du déplacement, soit à l'angle de rotation. Déjà, dans ce dernier cas, l'expression d'un angle en degrés, si elle semble évidente à des adultes, constitue une véritable découverte pour les enfants.

Ainsi, la séquence d'instructions suivante apprend à l'ordinateur une nouvelle commande (ou mot) appelée « CARRÉ » :

```
POUR CARRÉ
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
FIN
```

ou encore :

```
POUR CARRÉ
RÉPÈTE 4
AVANCE 100
DROITE 90
FIN
```

soit plus généralement :

```
POUR CARRÉ : N
RÉPÈTE 4
AVANCE : N
DROITE 90
FIN
```

On pourra définir le tracé d'un triangle équilatéral d'une façon analogue :

```
POUR TRIANGLE
AVANCE 100
DROITE 120
AVANCE 100
DROITE 120
AVANCE 100
DROITE 120
FIN
```

ou encore :

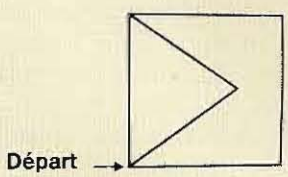
```
POUR TRIANGLE : N
```

RÉPÈTE 3
AVANCE : N
DROITE 120
FIN

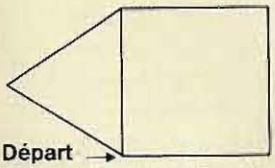
On voit bien que l'intérêt de ce langage réside moins dans le dessin en soi que dans la construction même du programme, élément par élément. Ce travail permet à l'enfant de hiérarchiser ses connaissances et de développer peu à peu une souplesse et une vivacité intellectuelle, qu'il aura ensuite très souvent l'occasion de mettre à profit. Cette façon d'appréhender les problèmes transparaît incontestablement dans les projets entrepris par les enfants après quelques séances de travail avec la Tortue.
La définition du «mot» MAISON est un exemple très simple mais cependant significatif. On devine facilement qu'après avoir défini les mots CARRÉ et TRIANGLE, il suffira pour dessiner une maison de disposer un triangle au-dessus d'un carré. Voici une première ébauche du programme :

POUR MAISON
CARRÉ
TRIANGLE
FIN

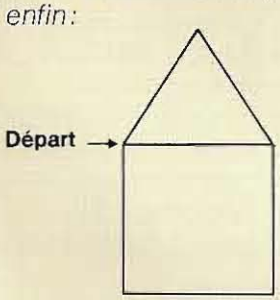
L'exécution de ce programme donne ce résultat...



Le triangle n'a pas été dessiné au-dessus mais à l'intérieur du carré !
Pour comprendre d'où provient l'erreur, il suffit de suivre le trajet parcouru par la Tortue ; on se rendra alors compte que le triangle est mal positionné parce que les deux mots sont définis de manière analogue et que le triangle vient donc se superposer sur le carré. On peut donc résoudre ce problème soit en redéfinissant le TRIANGLE avec des rotations de 120° vers la gauche (et non vers la droite), soit en modifiant l'orientation de la Tortue entre les deux figures.
En insérant l'instruction GAUCHE 60 entre CARRÉ et TRIANGLE, on obtient ceci :



L'apprenti-programmeur fait des progrès. Il réalise que les choses ne sont pas forcément justes ou fausses et qu'il peut y avoir une continuité d'un cas à l'autre. Cette deuxième maison a plus d'allure que la précédente mais ne sera parfaite qu'une fois redressée. Pour cela, il convient de faire pivoter la Tortue d'un quart de tour avant le début du dessin. Avec l'instruction DROITE 90 au début du programme, on obtient enfin :



Outre la Tortue et ses dessins, le LOGO permet : 1) la rédaction de poèmes fondée sur un remaniement créatif de la grammaire et sur la génération aléatoire de mots d'une catégorie déterminée ; 2) l'étude empirique de la régularité des nombres et des opérations arithmétiques ; 3) des expériences de composition musicale avec une « boîte à musique » commandée par un ordinateur. Le LOGO autorise une décomposition de la musique en éléments fondamentaux (mélodie, rythme, harmonie et couleur). En associant ces différents paramètres à l'aide d'instructions appropriées, on réussit à en saisir l'importance et la valeur réelle. Dans cette optique cognitive et interactive qui est à l'origine de langages comme le LOGO ou le SMALLTALK, l'ordinateur devient un instrument dont la programmation permet à chacun de formaliser ses connaissances et d'acquérir des capacités d'analyse et de résolution de problèmes complexes. Cette conception va donc bien plus loin que la notion traditionnelle d'E.A.O. et joue en faveur de l'introduction de l'informatique dans le milieu scolaire, et ce, bien avant toute orientation professionnelle.

(L'exemple est extrait de « Mindstorms », de S. Papert.)

la mise au point ou d'interventions ultérieures. Les étapes préliminaires peuvent paraître fastidieuses et sans grand rapport avec l'utilisation de la machine mais il faut être conscient du fait qu'un ordinateur ne peut traiter que des problèmes dont il connaît toutes les composantes. Un logiciel rédigé à la hâte est, dans le meilleur des cas, peu pratique, et, au pire, incomplet.
Il est possible d'informatiser les tâches les plus variées. Aussi le programmeur se trouve-t-il souvent confronté à des problèmes et à des domaines qu'il ne connaît pas.
L'utilisateur doit alors prendre part à la préparation du logiciel en fournissant au programmeur toutes les informations et les méthodes de calcul dont il pourra avoir besoin pour l'application en question.
Pour illustrer l'analyse d'un problème relativement complexe, nous allons suivre l'élaboration d'un programme calculant les déperditions calorifiques d'un local.
Le travail se déroule en six étapes :

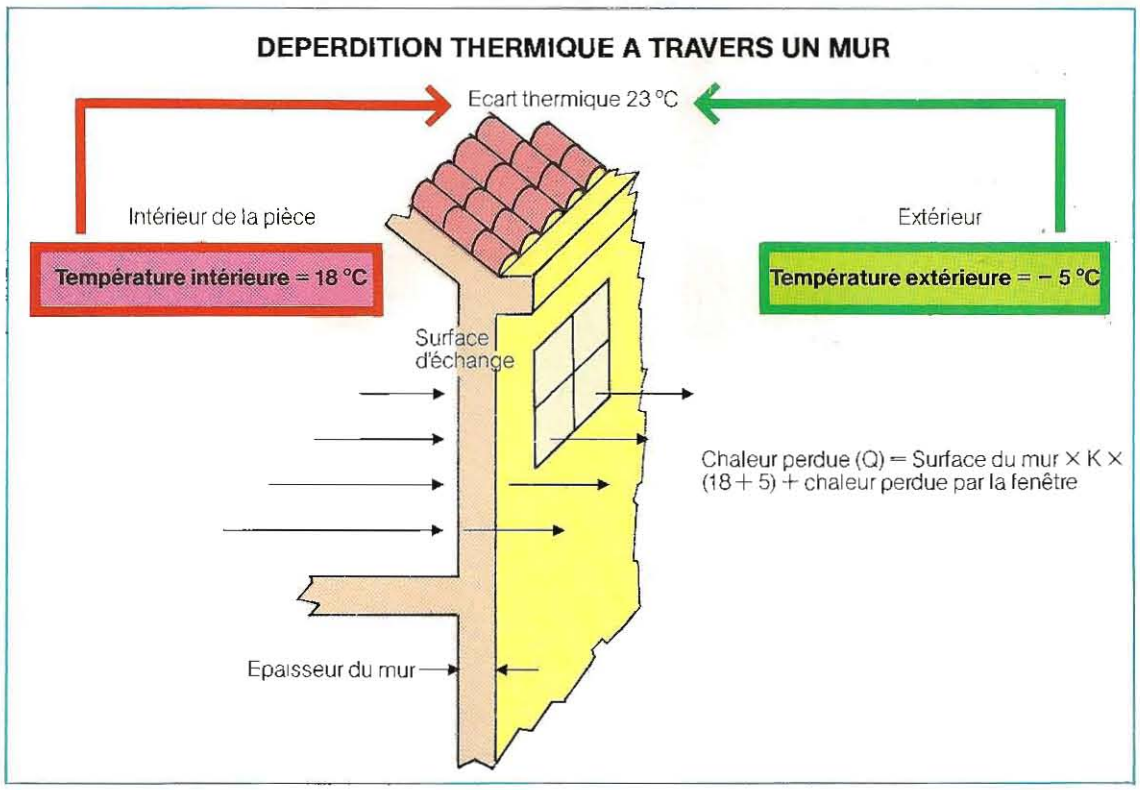
1 / une analyse générale, où sont définies les différentes variables et méthodes de traitement ;
2 / la synthèse, puis la préparation de l'agencement du programme principal ;
3 / une analyse détaillée et l'élaboration des organigrammes de chaque sous-programme ;
4 / l'écriture de ces sous-programmes, puis leur mise au point ;
5 / l'écriture du programme principal ;
6 / et enfin, la réunion du programme principal et des sous-programmes, assortie des derniers tests.

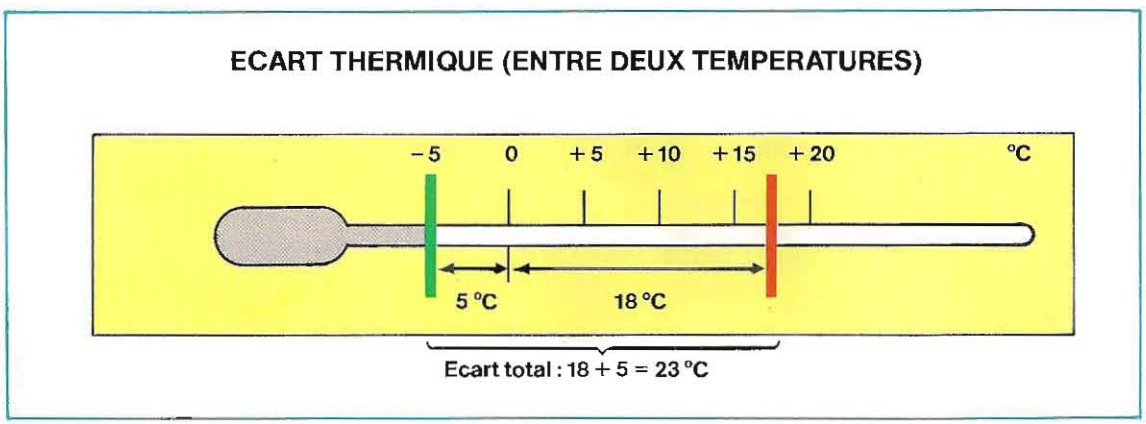
On voit que chaque sous-programme est réalisé et mis au point séparément avant d'être intégré à l'ensemble qui est testé à son tour. Nous verrons plus loin d'autres méthodes de contrôle des logiciels.

Analyse générale du problème. Il faut, tout d'abord, bien cerner le problème à résoudre et considérer l'ensemble des facteurs intervenant. Le programmeur travaillant pour un client a alors besoin de sa collaboration pour bien définir les objectifs à atteindre et les calculs à effectuer.
Dans notre exemple, il est indispensable de connaître les lois physiques régissant les échanges de chaleur.
On peut assimiler une pièce à un volume dont les parois séparent l'air intérieur de l'air extérieur. Les températures de l'air de part et d'autre des murs tendent à s'équilibrer par des déperditions de chaleur plus ou moins rapides selon la nature des parois. La pièce se refroidit donc en permanence et seul un apport de chaleur équivalent permet de maintenir une température constante.
Ainsi, pour une pièce perdant 1250 calories par heure, il est facile de déduire la quantité de mazout (par exemple) nécessaire au maintien de la température.
Ce calcul de consommation peut s'effectuer pour n'importe quelle source d'énergie et le but du programme est de déterminer la déperdition à compenser. Celle-ci dépend de nombreux facteurs et sera évaluée pour chaque pièce du local considéré. Il suffira ensuite d'additionner l'ensemble des pertes calculées pour connaître (en calories par heure) la puissance nécessaire au chauffage de tout le bâtiment. La quantité de chaleur rayonnant vers l'extérieur à travers une paroi est donnée par la relation suivante (voir tableau page 498) :

$$Q = \left[\begin{matrix}\text{Surface} \\ \text{du mur}\end{matrix}\right] \times K \times \left[\begin{matrix}\text{Température} \\ \text{intérieure}\end{matrix} - \begin{matrix}\text{Température} \\ \text{extérieure}\end{matrix}\right] \quad (1)$$

Q est la quantité de chaleur perdue exprimée en calories/heure et K est un coefficient de transmission dépendant du matériau et de l'épaisseur de la paroi.
Ce coefficient représente la capacité d'isolation d'une séparation.
Il est bien évident que les déperditions seront supérieures à travers une cloison qu'à travers un mur, mais moindres que par une vitre.
L'expression [Température intérieure – Température extérieure] représente l'écart de température de part et d'autre de la séparation. Plus grande sera cette différence et plus la pièce aura tendance à se refroidir vite. (Attention au calcul de l'écart thermique pour des températures extérieures inférieures à zéro : voir schéma page 497).
La déperdition est enfin proportionnelle à la surface d'échange entre les deux milieux. Les valeurs de K, indispensables au calcul, sont répertoriées dans des tables spécialisées





concernant les différents matériaux utilisés. Plus nous fournirons de valeurs de K à l'ordinateur, plus le programme sera « universel ». Nous n'utiliserons dans l'exemple qu'une table très rudimentaire, mais la conception du programme permet d'ajouter autant de données que l'on veut. Les valeurs actuellement retenues figurent dans la table de la page 498.

Avant d'aborder la conception des organigrammes, nous allons nous familiariser avec la méthode de calcul en résolvant « à la main » l'un des cas qui peuvent se présenter.

Type de paroi	Epaisseur (en cm) 25	38	50	Symbole	Table entrée dans l'ordinateur
Mur en briques	1.6	1.3	1.0	MB	KB(1), KB(2), KB(3)
Mur en ciment	2.2	1.8	1.6	MC	KC(1), KC(2), KC(3)
Sol		1.0		SL	
Toit		2.0		TO	
Porte		4.0		PO	
Fenêtre		5.0		FE	

KB(1) = 1.6
KB(2) = 1.3
KB(3) = 1.0

KC(1) = 2.2
KC(2) = 1.8
KC(3) = 1.6

Cette table donne les valeurs de K pour un certain nombre de parois dont le programme reconnaît les symboles indiqués dans l'avant-dernière colonne.

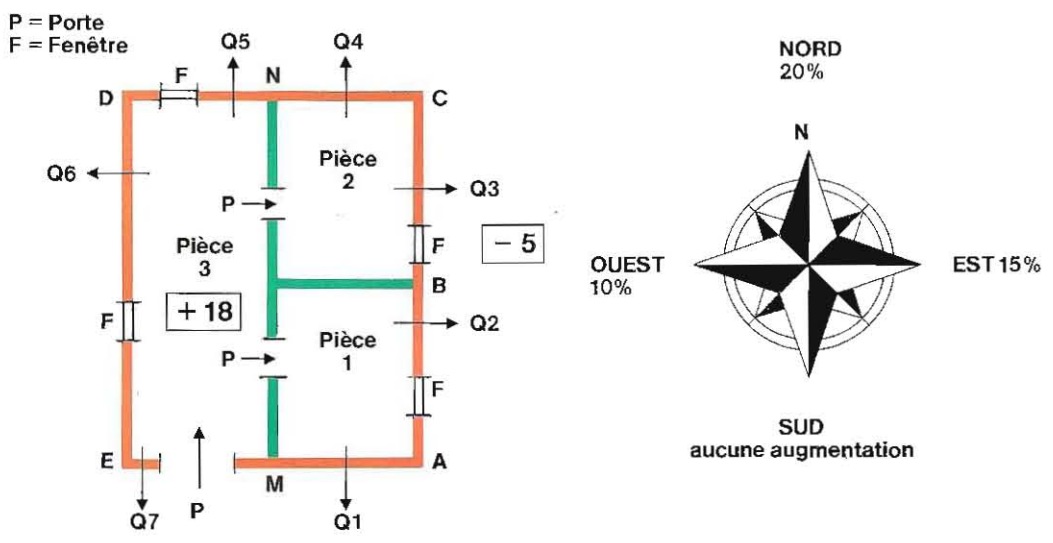
On a choisi une épaisseur moyenne pour les derniers éléments (sol, etc.) car ce paramètre n'influe que très peu sur la valeur de K correspondante.

Le programme est constitué d'une suite d'instructions paramétrées qui feront effectuer les mêmes opérations à la machine. Prenons le cas de l'appartement dont le plan figure ci-dessous.

Les traits verts représentent les murs intérieurs. Aucun échange thermique ne s'y produit puisque les différentes pièces sont à la même température (écart thermique nul). La déperdition de chaleur se fait donc à travers les murs extérieurs (en rouge), le toit et le sol. Le calcul sera effectué pièce par pièce. La maison présente trois salles dont les pertes sont réparties entre :

- les murs externes ;
- les fenêtres et les portes donnant sur l'extérieur ;

PLAN DU TROIS PIECES PRIS EN EXEMPLE



- le sol et le plafond ;
- le renouvellement de l'air (aération).

Ainsi, dans la pièce 1, des déperditions sont occasionnées par :

- les murs : Q2 + Q1 (Q2 sur le côté AB et Q1 sur le côté MA) ;
- une fenêtre ;
- le sol et le plafond : pertes proportionnelles à la surface de la pièce (côté AB × côté MA).

On doit également tenir compte de l'exposition des murs. Ainsi, un mur orienté au nord verra sa perte de chaleur majorée de 20 %. Les corrections respectives pour les murs est et ouest sont de 15 % et de 10 %.
Seule l'exposition au sud n'entraîne aucune perte supplémentaire. Le mur AB de la pièce 1 subira une déperdition de 15 % supérieure à celle tirée de la formule (1).
Supposons, par exemple, que le mur AB mesure 5 m, le mur AM 4 et qu'ils soient tous les deux construits en briques de 38 cm d'épaisseur sur une hauteur de 3,50 m ;
K = 1.3 (voir table page 498).
Pour le mur est (AB) nous aurons donc :

$$\text{Surface} = 5 \times 3{,}50 = 17{,}50 \text{ m}^2$$

dont il faut déduire la surface de la fenêtre (2 m^2) car sa contribution aux pertes sera calculée séparément (K différent de celui du mur).
La surface d'échange est donc :

$$\text{Surface} = 17{,}50 - 2 = 15{,}50 \text{ m}^2$$

Après application de la formule (1) :

$$\text{Chaleur perdue (Q1)} = 15{,}50 \times 1{,}3 \times (18 + 5)$$

on trouve :

$$Q1 = 463{,}45 \text{ kcal/heure}$$

Nous devons encore majorer cette valeur de 15 % en raison de l'exposition du mur :

$$Q1 = 533 \text{ kcal/heure (valeur arrondie)}$$

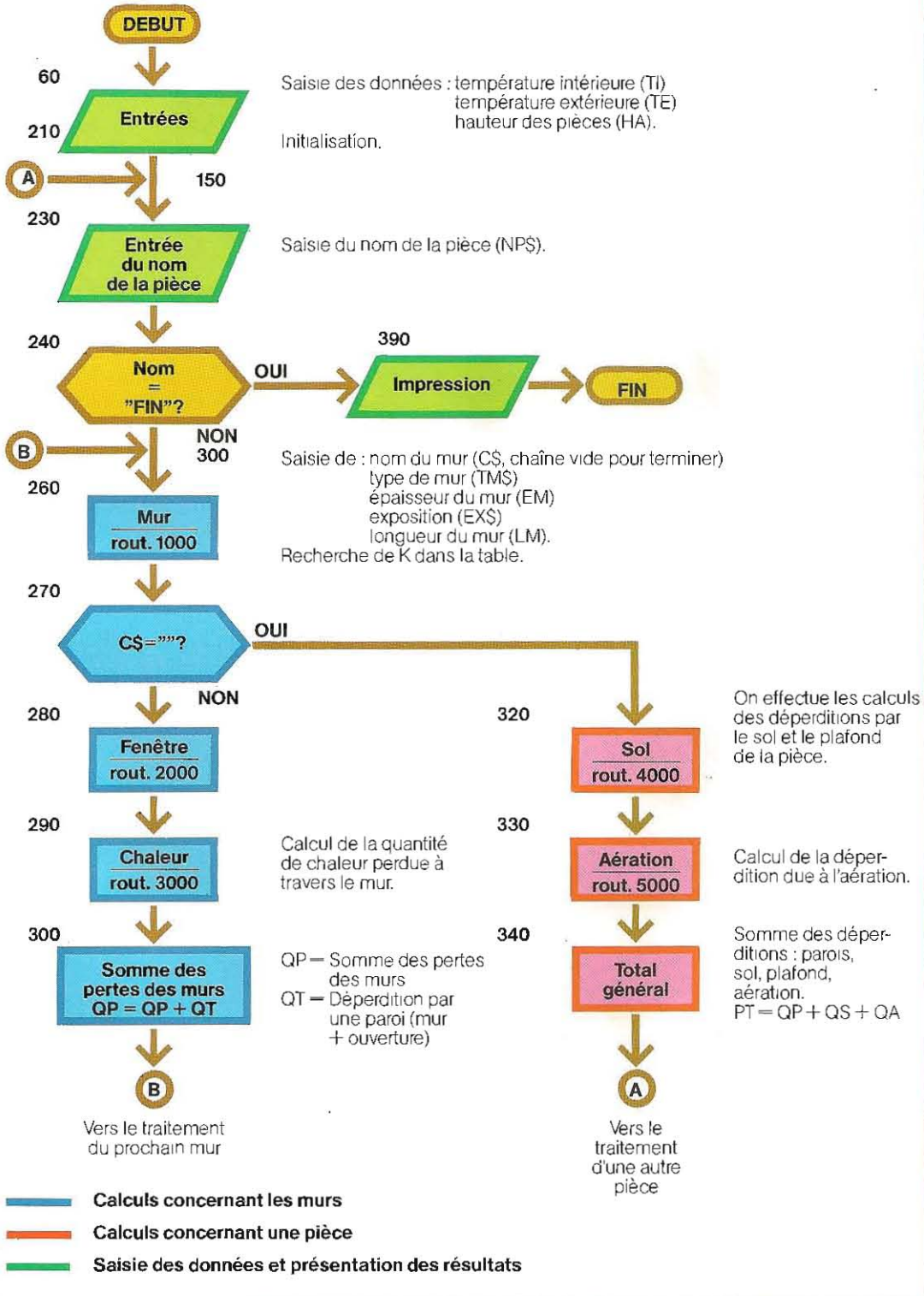
A elle seule, la fenêtre (surface 2 m^2, K = 5) émet 2 × 5 × (18 + 5) soit 230 kcal/heure, et donc, après correction (15 % pour l'est), 265 kcal/heure. La quantité de chaleur rayonnée par la façade AB s'élève ainsi à :
533 + 265 = 798 kcal/heure.
On pourrait effectuer le même calcul pour le mur MA, puis pour le toit (K = 2, surface = côté AB × côté MA) et enfin pour le sol (K = 1, même surface).
Toutefois, la somme de ces résultats ne représente pas la totalité des déperditions de la pièce 1, car il faut aussi tenir compte de la chaleur perdue avec le renouvellement de l'air.
L'explusion d'air chaud et son remplacement par de l'air froid provoque une perte que l'on peut évaluer ainsi :

$$\text{Renouvellement} = \left[\begin{matrix}\text{Volume de} \\ \text{la pièce}\end{matrix}\right] \times 0.3 \times \left[\begin{matrix}\text{Ecart de} \\ \text{température}\end{matrix}\right]$$

La quantité de chaleur nécessaire pour maintenir la pièce à une température constante est donc égale à la somme des calories émises à travers les parois et de celles véhiculées par les courants d'air.
En réitérant ce calcul pour l'ensemble des pièces, on arrive à déterminer la puissance requise pour l'installation de chauffage.
Nous avons maintenant terminé l'analyse générale et on doit en faire la synthèse avant d'élaborer l'organigramme principal.
Récapitulons les étapes de cette procédure :

1 / saisie des constantes : températures intérieure et extérieure, hauteur des pièces ;
2 / puis, pour chaque pièce, saisie du nom de la pièce (ou du mot FIN quand toutes les pièces ont été traitées) ;
3 / pour chaque mur externe, saisie du matériau, de la longueur et de l'exposition ;
4 / saisie de la surface des portes et des fenêtres éventuelles ;
5 / calcul des pertes de chaleur à travers ce mur ;
6 / retour à l'étape 3 pour les autres parois externes de cette pièce ;
7 / saisie de la surface du sol et du toit ;
8 / calcul des déperditions à travers le sol et le toit ;
9 / calcul de la chaleur perdue lors du renouvellement d'air (aération) ;
10 / addition de toutes les pertes de chaleur de cette pièce ;
11 / retour à l'étape 2 pour une autre pièce ;
12 / calcul et impression du total général.

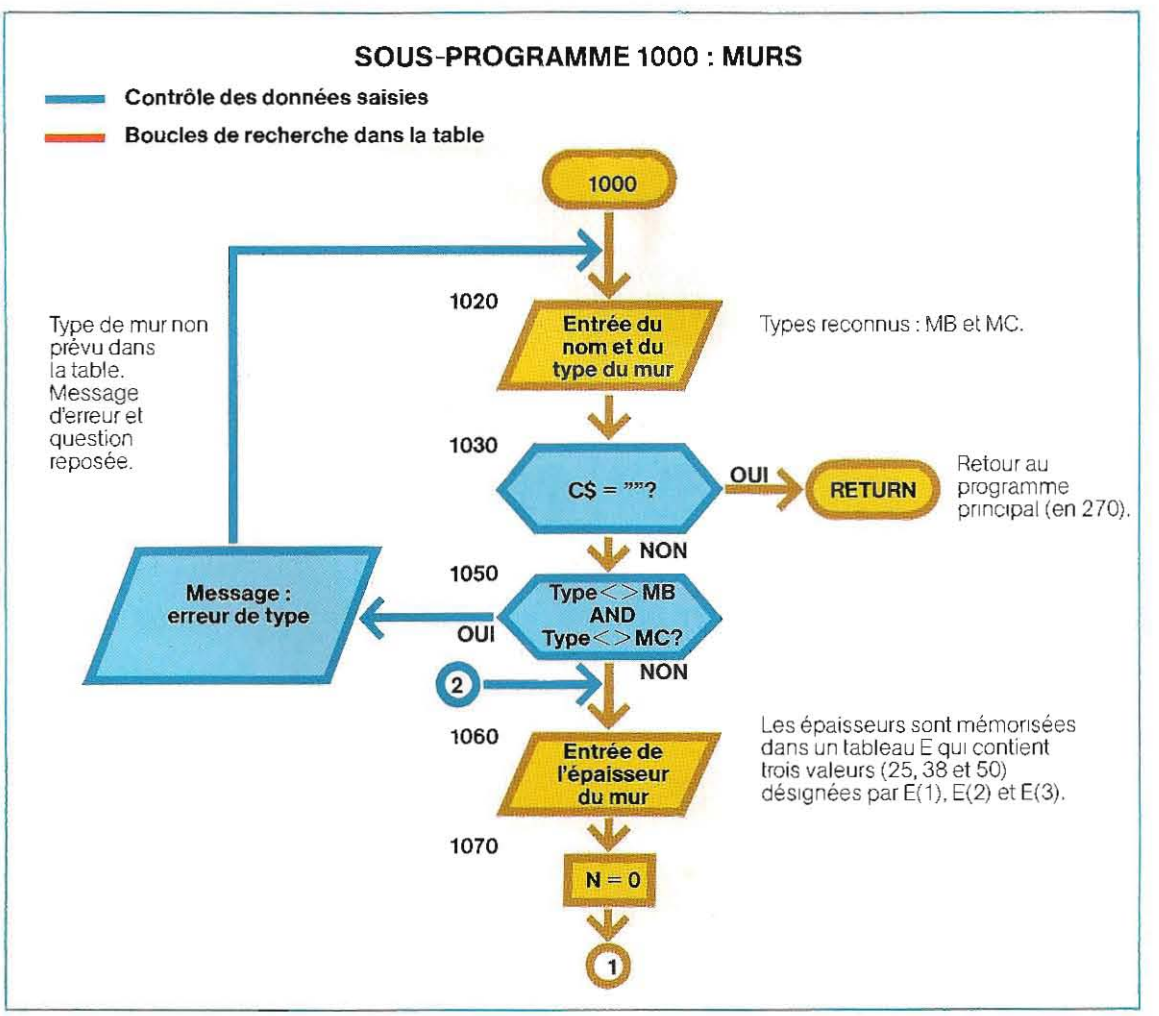
CALCUL DES DEPERDITIONS THERMIQUES
ORGANIGRAMME DU PROGRAMME PRINCIPAL

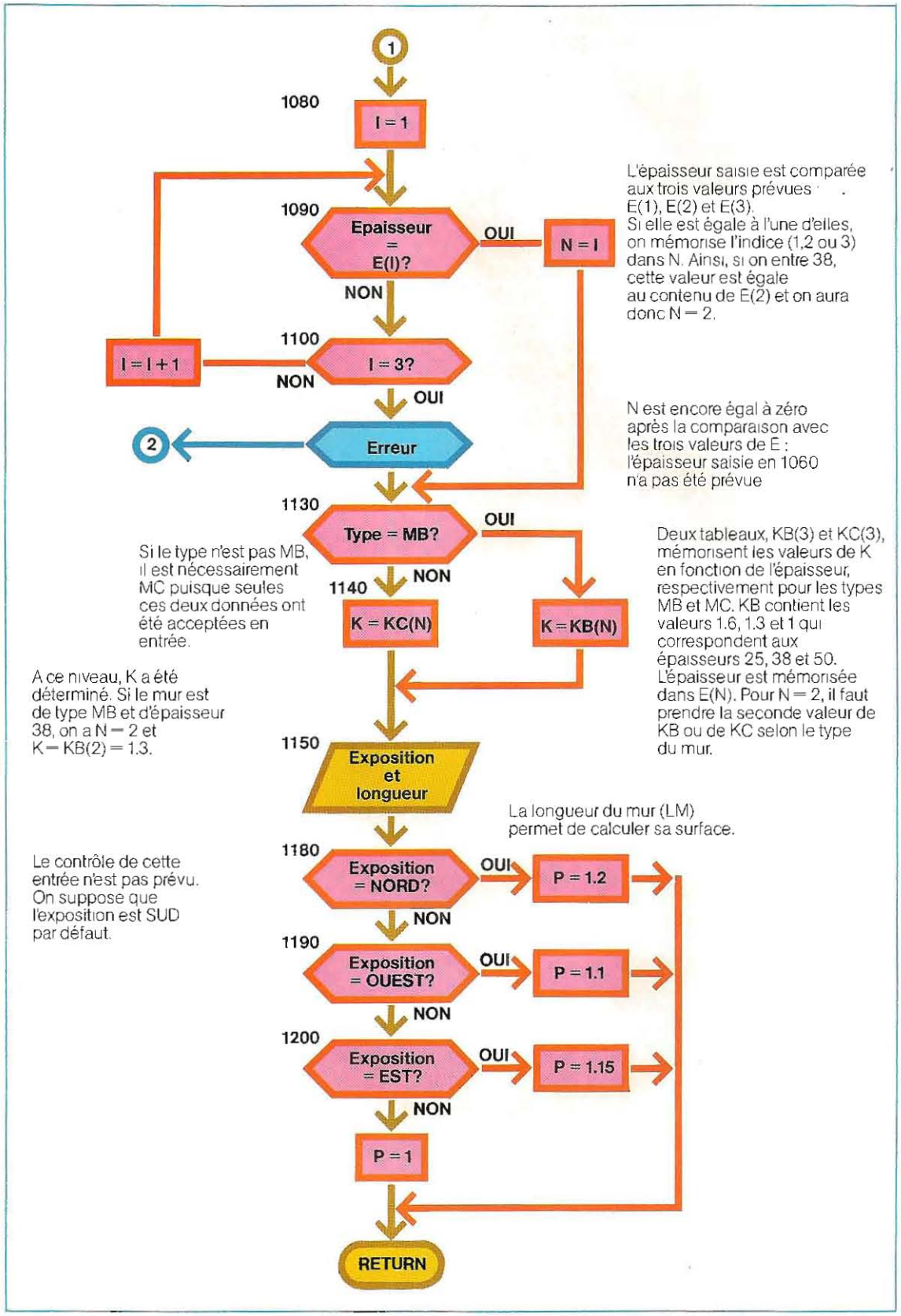


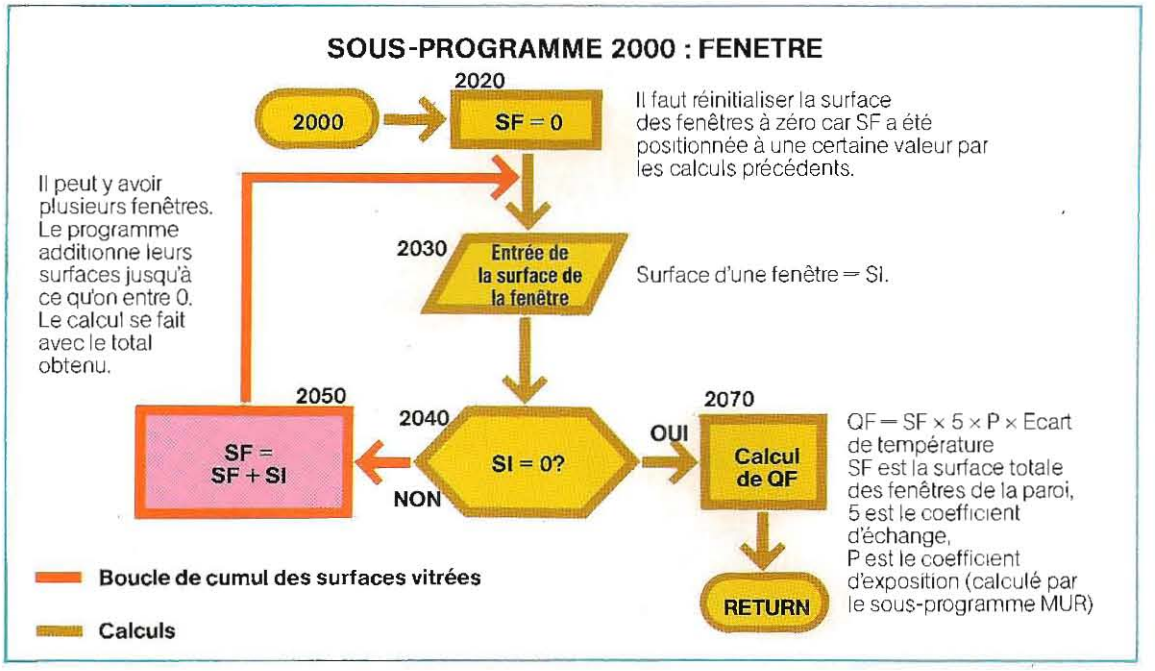
L'organigramme principal présenté page 500 montre l'enchaînement logique des opérations à effectuer. Il faudra traduire ces opérations en Basic pour obtenir le programme proprement dit. Les blocs représentant des calculs seront traités par des sous-programmes (routines) débutant chacun à la ligne indiquée (par exemple, la déperdition par une fenêtre sera calculée par la routine 2000).

Les sous-programmes. Avant d'élaborer l'organigramme des différents sous-programmes, on doit approfondir l'analyse en définissant avec précision les valeurs utilisées par chacun d'eux, les calculs à effectuer et les résultats à dégager.
Voici le détail des différents traitements.
Sous-programme 1000 : MURS
Déroulement :
Saisie des caractéristiques du mur.
Vérification de la concordance de la donnée avec une valeur de la table (symbole de deux lettres, voir page 498).
Lecture du coefficient K correspondant.
Détermination de la correction due a l'exposition.
Sorties :
K = coefficient d'échanges thermiques.
P = coefficient d'exposition (P = 1.2 pour le nord, 1.15 pour l'est, 1.1 pour l'ouest et 1 pour le sud).
LM = longueur du mur.
Tableaux utilisés :
E(3) : épaisseurs de mur possibles.
KB(3) : coefficients thermiques de la brique.
KC(3) : coefficients thermiques du ciment.

Sous-programmes 2000 : FENETRE
Déroulement :
Boucle de saisie de la surface des ouvertures



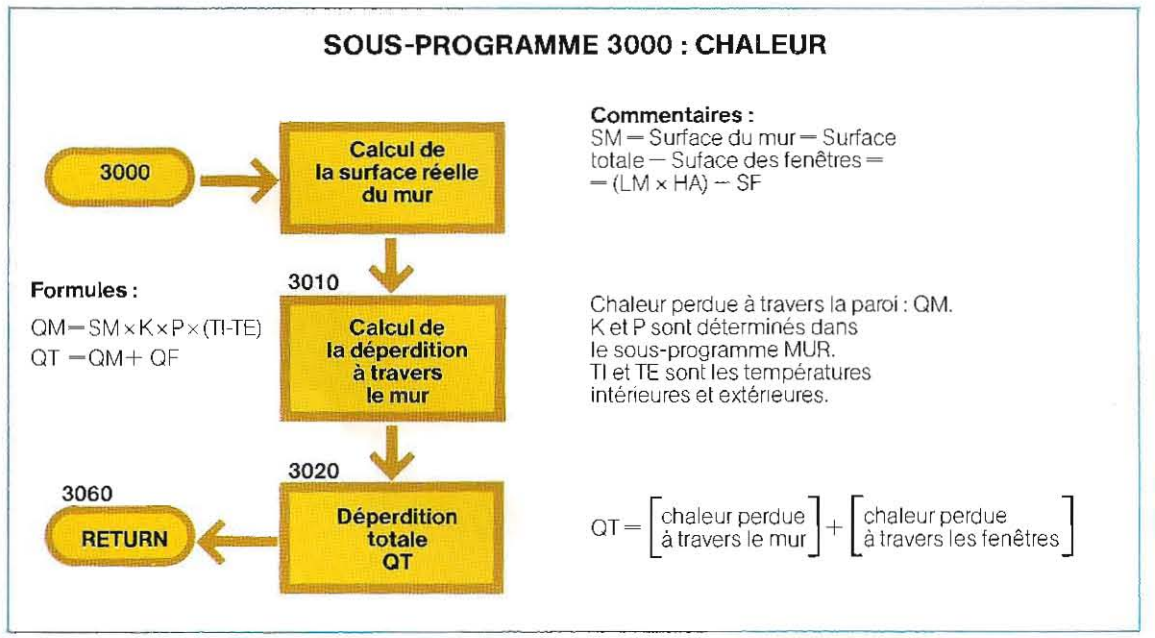

1
1080
I = 1
1090
Epaisseur = E(I)?
OUI
N = I
NON
1100
I = I + 1
I = 3?
NON
OUI
2
Erreur
1130
Type = MB?
OUI
NON
1140
K = KC(N)
K = KB(N)
1150
Exposition et longueur
1180
Exposition = NORD?
OUI
P = 1.2
NON
1190
Exposition = OUEST?
OUI
P = 1.1
NON
1200
Exposition = EST?
OUI
P = 1.15
NON
P = 1
RETURN
L'épaisseur saisie est comparée aux trois valeurs prévues E(1), E(2) et E(3). Si elle est égale à l'une d'elles, on mémorise l'indice (1,2 ou 3) dans N. Ainsi, si on entre 38, cette valeur est égale au contenu de E(2) et on aura donc N = 2.
N est encore égal à zéro après la comparaison avec les trois valeurs de E : l'épaisseur saisie en 1060 n'a pas été prévue
Deux tableaux, KB(3) et KC(3), mémorisent les valeurs de K en fonction de l'épaisseur, respectivement pour les types MB et MC. KB contient les valeurs 1.6, 1.3 et 1 qui correspondent aux épaisseurs 25, 38 et 50. L'épaisseur est mémorisée dans E(N). Pour N = 2, il faut prendre la seconde valeur de KB ou de KC selon le type du mur.
Si le type n'est pas MB, il est nécessairement MC puisque seules ces deux données ont été acceptées en entrée.
A ce niveau, K a été déterminé. Si le mur est de type MB et d'épaisseur 38, on a N = 2 et K = KB(2) = 1.3.
La longueur du mur (LM) permet de calculer sa surface.
Le contrôle de cette entrée n'est pas prévu. On suppose que l'exposition est SUD par défaut.



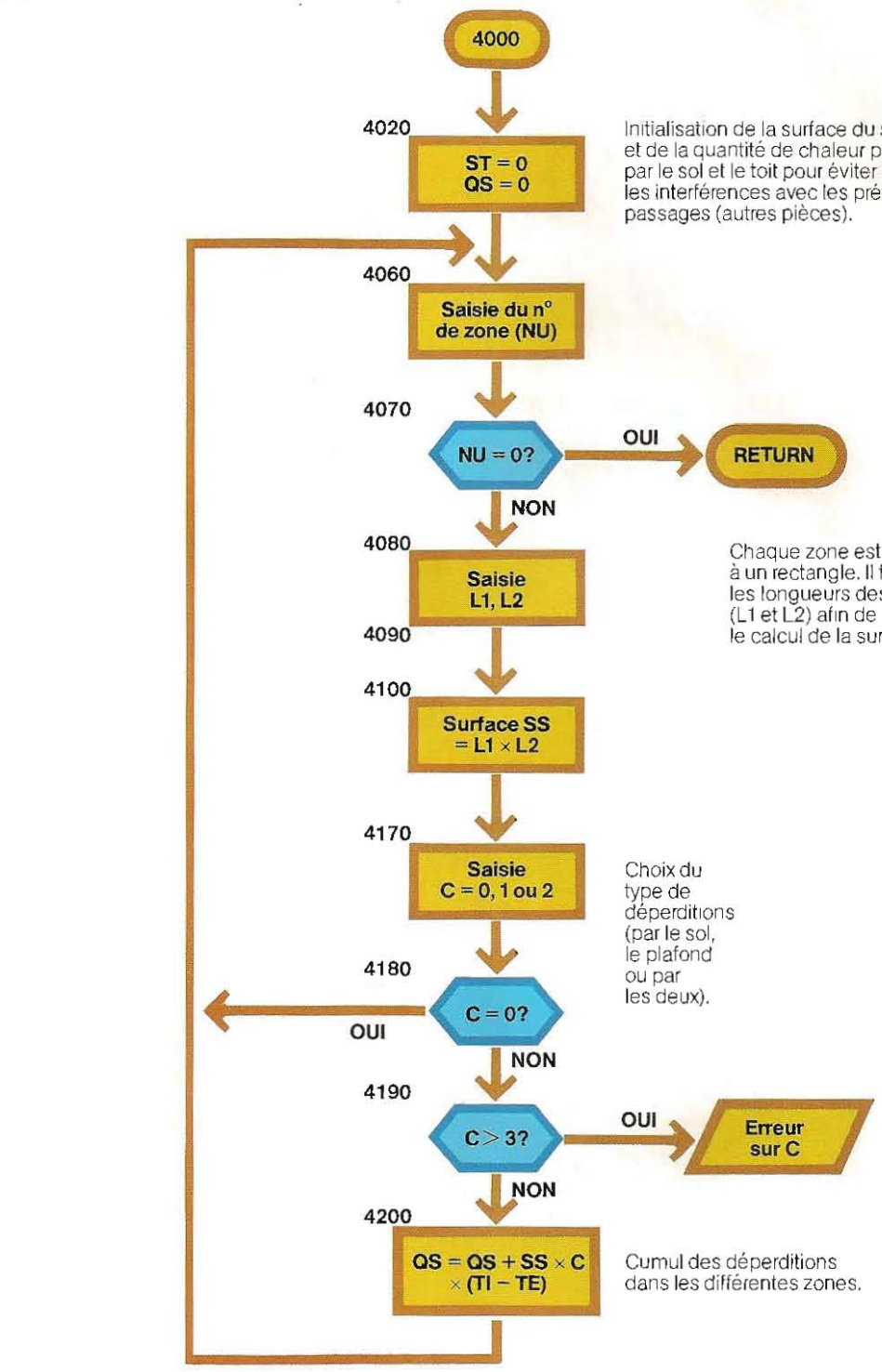
quel que soit leur nombre.
Calcul de la quantité de chaleur qui s'échappe par les fenêtres.
Sorties :
SF = Surface des fenêtres (à soustraire de la surface des murs).
QF = Pertes de chaleur à travers les fenêtres.

Sous-programme 3000 : CHALEUR
Déroulement :
Calcul de la quantité de chaleur perdue par le mur et de la déperdition totale (mur + fenêtres éventuelles).
Sorties :
QM = Déperdition totale de cette façade.

Sous-programme 4000 : SOL
Déroulement :
Calcul des pertes de chaleur par le sol et par le toit.



SOUS-PROGRAMME 4000 : SOL



Sorties :
QS = Total des déperditions de chaleur par le sol et par le plafond.
Il faut effectivement préciser si de la chaleur se dissipe par le sol ou par le plafond : si la pièce est située au-dessus (ou en-dessous) d'une autre pièce chauffée, il n'y a pas de déperditions par le sol (ou par le plafond).

Sous-programme 5000 : AERATION

Opérations exécutées :
Calcul de la chaleur perdue pendant l'aération.

Sorties :
QA = Chaleur perdue pendant l'aération.

Le programme. Le listing complet et commenté du programme de calcul des déperditions thermiques commence ci-dessous et se poursuit pages 506 et 507 ; un exemple d'exécution suit, en pages 508 et 509.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES

```
10 REM ** PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES **
11 REM
12 REM
14 REM exécute en MICROSOFT BASIC
15 REM sur HHC (PANASONIC)
16 REM
18 REM
19 REM
20 REM
21 REM N.B. ici l'instruction standard LPRINT est remplacée
22 REM par PRINT #2 (2=numéro logique de l'imprimante)
23 REM
24 REM   liste des variables utilisées
25 REM   TI  = température intérieure
26 REM   TE  = température extérieure
27 REM   HA  = hauteur des pièces
28 REM   NP$ = nom de la pièce
29 REM   TM$ = type de mur (MB/MC)
30 REM   EM  = épaisseur du mur
31 REM   EX$ = exposition (N,S,E,O) du mur
32 REM   LM  = longueur du mur
33 REM   P   = coefficient d'exposition
34 REM   SI  = surface d'une fenêtre
35 REM   SF  = surface totale fenêtres du mur
36 REM   QF  = déperditions par fenêtres
37 REM   SM  = surface réelle du mur
38 REM   QM  = déperditions par le mur
39 REM   QT  = somme de QM+QF pour le mur
40 REM   QP  = somme des QT pour la pièce
41 REM   NU  = numéro de zone du sol
42 REM   L1  = côté 1 du sol
43 REM   L2  = côté 2 du sol
44 REM   SS  = surface du sol
45 REM   ST  = somme des SS
46 REM   QS  = déperditions par sol + plafond
48 REM   QA  = déperditions par aération
50 REM   TI  = total des déperditions = puissance
51 REM         de l'installation de chauffage
52 REM   T2  = somme des QA
53 REM   T3  = somme des QS
54 REM   T4  = somme des QT
55 REM
56 REM   initialisations
57 REM   tableaux en entrée
60 DIM E(3):REM épaisseur murs
70 DIM KB(3),KC(3):REM coefficients thermiques
80 E(1)=25
90 E(2)=38
100 E(3)=50
110 KB(1)=1.6
120 KB(2)=1.3
130 KB(3)=1
140 KC(1)=2.2
150 KC(2)=1.8
```

```
160 KC(3)=1.6
170 REM   autres initialisations
180 T1=0:T2=0:T3=0:T4=0
181 REM
182 REM   PROGRAMME PRINCIPAL
183 REM
190 INPUT "température intérieure :";TI
200 INPUT "température extérieure :";TE
210 INPUT "hauteur des pièces :";HA
211 REM
212 REM   boucle sur les pièces
220 PRINT "tapez le mot FIN pour sortir"
230 INPUT "nom de la pièce :";NP$
240 IF NP$="FIN"GOTO 390
250 QP=0
251 REM   boucle sur les murs d'une pièce
260 GOSUB 1000:REM routine MUR
270 IF C$=""GOTO 320
280 GOSUB 2000:REM routine FENETRE
290 GOSUB 3000:REM routine CHALEUR
300 QP=QP+QT
310 GOTO 260
311 REM
312 REM   fin du traitement de la pièce
320 GOSUB 4000:REM routine SOL
330 GOSUB 5000:REM routine AERATION
340 PT=QP+QS+QA
350 PRINT #2 "DEPERDITIONS THERMIQUES DANS LA PIECE ";NP$;" : ";PT
360 PRINT #2:PRINT #2
370 T1=T1+PT:REM cumul des pertes
380 GOTO 220
381 REM
382 REM   instruction d'impression
383 REM
390 PRINT #2"temperature extérieure : ";TE
400 PRINT #2"temperature intérieure : ";TI
410 PRINT #2"hauteur des pièces : ";HA
420 PRINT #2
430 PRINT #2"DEPERDITIONS TOTALES (cal/h) : "
440 PRINT #2
450 PRINT #2"murs et fenêtres : ";T4
460 PRINT #2"sols et plafonds : ";T3
470 PRINT #2"aération : ";T2
480 PRINT #2
490 PRINT #2"PUISSANCE DE L'INSTALLATION DE CHAUFFAGE : ";T1
500 END
501 REM
502 REM
503 REM   SOUS-PROGRAMME MUR
504 REM
1000 PRINT "sous-programme MUR":PRINT NP$
1010 PRINT "tapez seulement RETURN quand il n'y a plus de mur à traiter"
1020 INPUT "côté :";C$
1030 IF C$=""THEN GOTO 1300
1040 INPUT "type de mur (MB/MC) :";TM$
1050 IF TM$<>"MB"AND TM$<>"MC"THEN PRINT "erreur de type":GOTO 1040
1060 INPUT "épaisseur du mur (25/38/50) :";EM
1070 N=0
1080 FOR I=1 TO 3
1090 IF EM=E(I) THEN N=1:GOTO 1130
1100 NEXT I
1110 PRINT "erreur sur l'épaisseur"
1120 GOTO 1060
1130 IF TM$="MB"THEN K=KB(N):GOTO 1150
1140 K=KC(N)
1150 INPUT "exposition du mur (NORD,SUD,EST,OUEST) :";EX$
1160 INPUT "longueur du mur (metres) :";LM
1170 P=1
1180 IF EX$="NORD"THEN P=1.2
1190 IF EX$="OUEST"THEN P=1.1
1200 IF EX$="EST"THEN P=1.15
1201 REM
1210 PRINT #2"pièce : ";NP$
```

```
1220 PRINT #2"côté de la pièce : ";C$
1230 PRINT #2"K = ";K
1240 PRINT #2"coefficient d'exposition P = ";P
1250 PRINT #2"longueur du mur en mètres : ";LM
1300 RETURN
1310 REM
1311 REM   SOUS-PROGRAMME FENETRE
1312 REM
2000 PRINT "sous-programme FENETRE"
2010 PRINT NP$
2020 SF=0
2030 INPUT "surface de l'ouverture en m² (0 pour sortir) : ";SI
2040 IF SI=0 THEN GOTO 2070
2050 SF=SF+SI
2060 GOTO 2030
2070 QF=SF*5*P*(TI-TE)
2071 REM
2080 PRINT #2"surface totale fenêtres : ";SF
2090 PRINT #2"déperditions de chaleur par les fenêtres (cal/h): ";QF
2100 RETURN
2110 REM
2111 REM   SOUS-PROGRAMME CHALEUR
2112 REM
3000 SM=(LM*HA)-SF
3010 QM=SM*K*P*(TI-TE)
3020 QT=QM+QF
3021 REM
3030 PRINT #2"déperditions de chaleur par le mur : ";QM
3040 PRINT #2"déperditions totales (mur + fenêtres) : ";QT
3045 PRINT #2
3050 T4=T4+QT
3060 RETURN
3070 REM
3071 REM   SOUS-PROGRAMME SOL
3072 REM
4000 PRINT "sous-programme SOL"
4010 PRINT NP$
4020 ST=0
4030 QS=0
4040 PRINT "donnez un numéro de zone même si"
4050 PRINT "le sol n'a pas été subdivisé"
4060 INPUT "numéro de zone (0 pour sortir) :";NU
4070 IF NU=0 THEN GOTO 4220
4080 INPUT "longueur 1er côté : ";L1
4090 INPUT "longueur 2e côté : ";L2
4100 SS=L1*L2:REM surface de la zone
4110 ST=ST+SS:REM cumul surface de la pièce
4120 PRINT "choisissez le numéro voulu :"
4130 PRINT "0=pas de pertes (ni par le sol ni par le toit)"
4140 PRINT "1=déperditions par le sol"
4150 PRINT "2=déperditions par le toit"
4160 PRINT "3=pertes par le sol et par le toit"
4170 INPUT "votre choix :";C
4180 IF C=0 THEN GOTO 4060
4190 IF C>3 THEN PRINT "erreur":GOTO 4120
4200 QS=QS+SS*C*(TI-TE)
4210 GOTO 4060
4211 REM
4220 PRINT #2"autres déperditions"
4230 PRINT #2"pertes de chaleur par le sol et/ou le toit : ";QS
4240 T3=T3+QS
4250 RETURN
4260 REM
4261 REM   SOUS-PROGRAMME AERATION
4262 REM
4263 REM   chaleur perdue pendant l'aération
4264 REM   = volume pièce * 0.3 * écart températures
5000 QA=ST*HA*0.3*(TI-TE)
5001 PRINT #2"pertes de chaleur en aération : ";QA
5015 PRINT #2
5020 T2=T2+QA
5030 RETURN
```

```
pièce : SALLE DE SEJOUR
côté de la pièce : BC
K = 1
coefficient d'exposition P = 1.15
longueur du mur en mètres : 3.5
surface totale fenêtres : 1.8
déperditions de chaleur par les fenêtres (cal/h) : 134.55
déperditions de chaleur par le mur : 130.065
déperditions totales (mur+fenêtres) : 264.615

pièce : SALLE DE SEJOUR
côté de la pièce : CD
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 4
surface totale fenêtres : 2.52
déperditions de chaleur par les fenêtres (cal/h) : 163.8
déperditions de chaleur par le mur : 123.24
déperditions totales (mur+fenêtres) : 287.04

autres déperditions

pertes de chaleur par le sol et/ou le toit : 396.5
pertes de chaleur en aération : 178.425
DEPERDITIONS THERMIQUES DANS LA PIECE (SALLE DE SEJOUR) : 1126.58

pièce : SALLE DE BAINS
côté de la pièce : DE
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 1.8
surface totale fenêtres : 1.2
déperditions de chaleur par les fenêtres (cal/h) : 78
déperditions de chaleur par le mur : 54.6
déperditions totales (mur+fenêtres) : 132.6

autres déperditions

pertes de chaleur par le sol et/ou le toit : 117
pertes de chaleur en aération : 52.65
DEPERDITIONS THERMIQUES DANS LA PIECE (SALLE DE BAINS) : 302.25

pièce : CUISINE
côté de la pièce : EF
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 2.6
surface totale fenêtres : 2.52
déperditions de chaleur par les fenêtres (cal/h) : 163.8
déperditions de chaleur par le mur : 68.64
déperditions totales (mur+fenêtres) : 232.44

autres déperditions

pertes de chaleur par le sol et/ou le toit : 169
pertes de chaleur en aération : 76.05
DEPERDITIONS THERMIQUES DANS LA PIECE (CUISINE) : 477.49

pièce : CHAMBRE
côté de la pièce : FG
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 3.5
surface totale fenêtres : 1.8
déperditions de chaleur par les fenêtres (cal/h) : 117
déperditions de chaleur par le mur : 113.1
déperditions totales (mur+fenêtres) : 230.1

autres déperditions
```

```
pertes de chaleur par le sol et/ou le toit : 338
pertes de chaleur par aération : 152.1
DEPERDITIONS THERMIQUES DANS LA PIECE (CHAMBRE) : 720.2

température extérieure : 5
température intérieure : 18
hauteur des pièces      : 3

DEPERDITIONS TOTALES (cal/h) :

murs et fenêtres : 1146.785

sols et plafonds : 1020.5

aération         : 459.225

PUISSANCE DE L'INSTALLATION DE CHAUFFAGE : 2626.52
```

Lors de l'impression des résultats, le mot « calorie » désigne la « grande calorie » ou kilocalorie (1 kilocalorie = 1000 calories).
Les reproductions d'écran qui se trouvent pages 510 à 514 illustrent la partie interactive de l'exécution du programme : la phase de saisie des données. De même que ceux du listing, les exemples montrés dans cette séquence se rapportent tous à l'appartement dont le plan détaillé figure page 515.

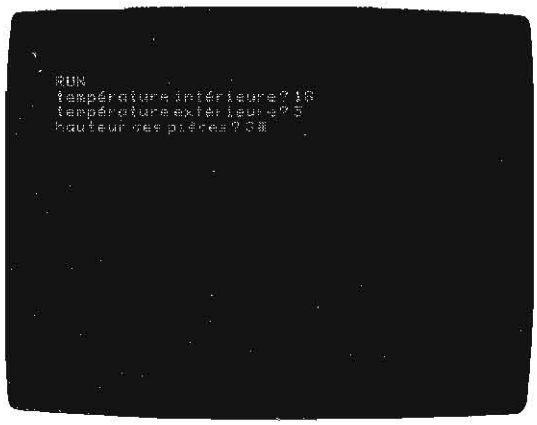
Ce cliché aux infrarouges révèle une isolation thermique insuffisante, voire inexistante.



AGA Infrared Systems/Science Photo Library

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 1)

On lance l'exécution du programme par la commande RUN. L'interpréteur commence alors à traduire et à exécuter les lignes d'instructions, en partant de celle qui porte le plus petit numéro.





L'interpréteur exécute la ligne 190, qui demande une entrée au clavier (l'instruction INPUT est expliquée dans le chapitre consacré aux opérations d'E/S). On voit apparaître sur l'écran le message qui figure entre guillemets dans cette ligne du programme. Le point d'interrogation, ajouté par le système, indique que l'exécution est suspendue en attendant la donnée demandée.
L'opérateur tape 18 sur son clavier et valide ce nombre par la touche RETURN. La valeur entrée est alors affectée à la variable TI, comme le prévoit l'instruction 190.

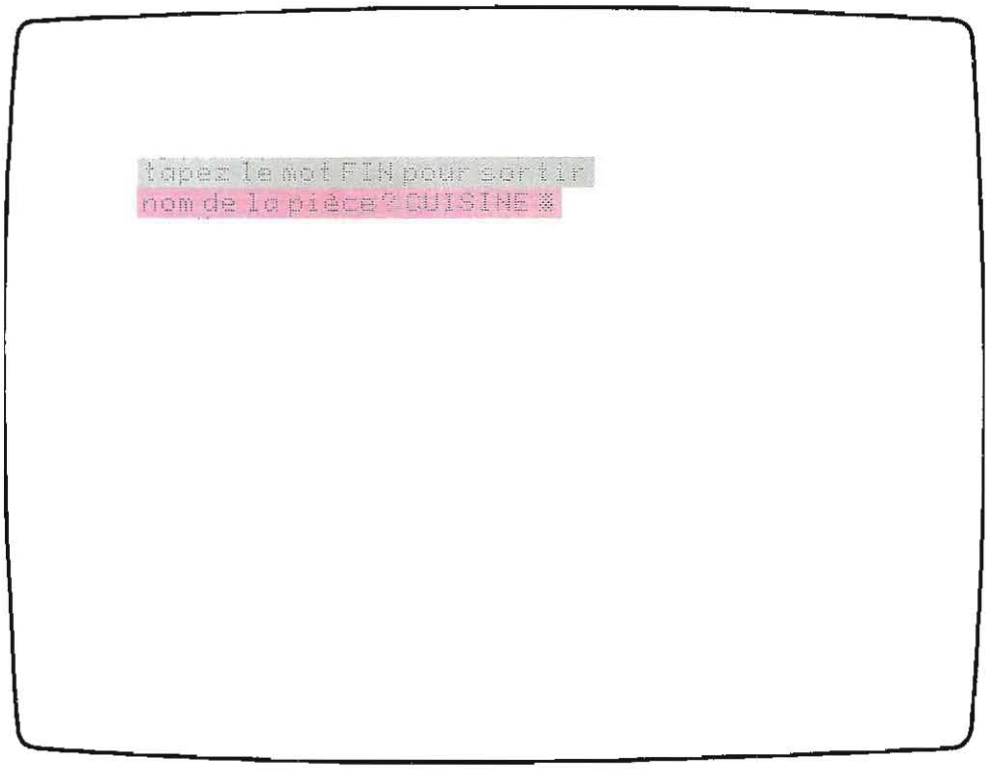
Le programme prévoit la saisie d'une nouvelle donnée (ligne 200). L'opérateur entre la valeur 5, qui est affectée à la variable TE.

Cette fois (ligne 210), c'est la hauteur des pièces qui est demandée (3 m). A l'instant où la photo a été prise, l'opérateur n'avait pas encore appuyé sur la touche RETURN, et le curseur clignotait donc encore à la fin de la dernière ligne.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 2)

Voici maintenant le moment de choisir la pièce dont on veut calculer les déperditions.





L'exécution de la ligne 220 entraîne l'affichage de ce message, qui indique à l'opérateur comment mettre fin à l'exécution (en tapant le mot FIN en guise de nom de pièce). Remarquons qu'il s'agit là de la seule façon logique et correcte d'interrompre la boucle du programme.

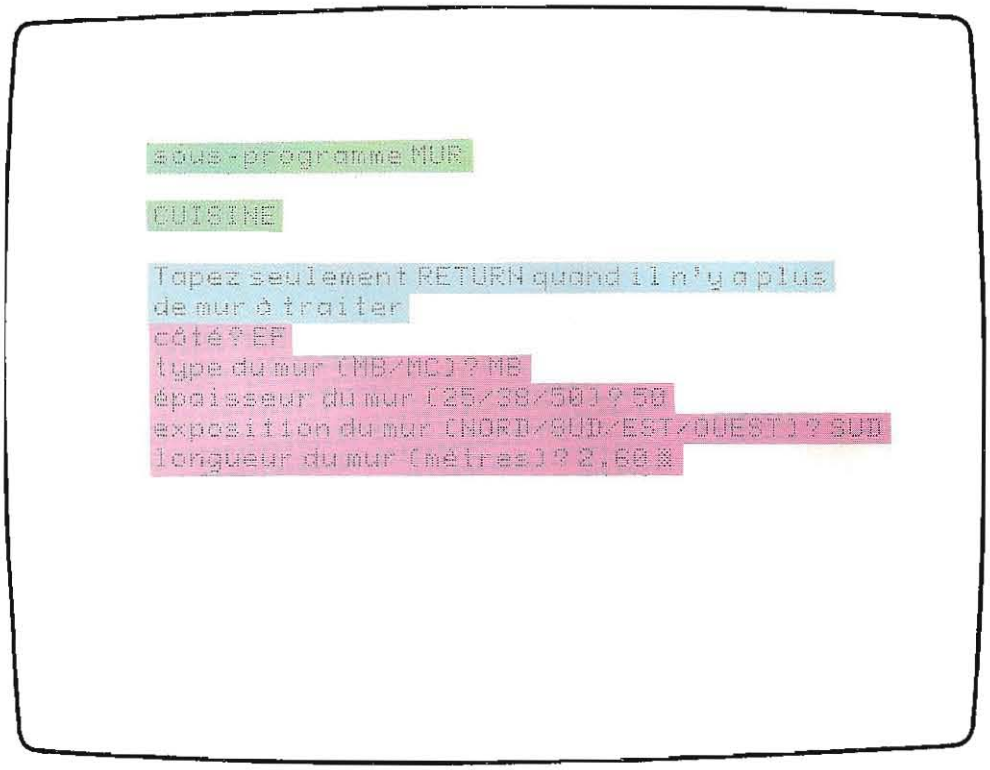
Cette méthode est très fréquemment employée dans les applications en mode interactif (dialogué). L'utilisateur doit suivre à la lettre les indications qui lui sont fournies, s'il veut que l'exécution se poursuive correctement. Il faut donc, bien sûr, que le programmeur pense à guider le mieux possible les opérateurs, par des messages comme celui-ci, et leur accorde le « droit à l'erreur » en prévoyant tous les contrôles nécessaires.
En effet, un programme interactif mal conçu sera très pénible à utiliser, et on risque, de plus, de perdre des données précieuses.

En ligne 230, le programme saisit le nom de la pièce dont les déperditions vont être calculées. L'opérateur tape le mot CUISINE, qui est affecté à la chaîne de caractères NP$.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 3)

Après la saisie des données communes à toutes les pièces (phase 1) et du nom de la première pièce à traiter (phase 2), l'instruction 260 appelle la routine 1000 (SOUS-PROGRAMME MUR).





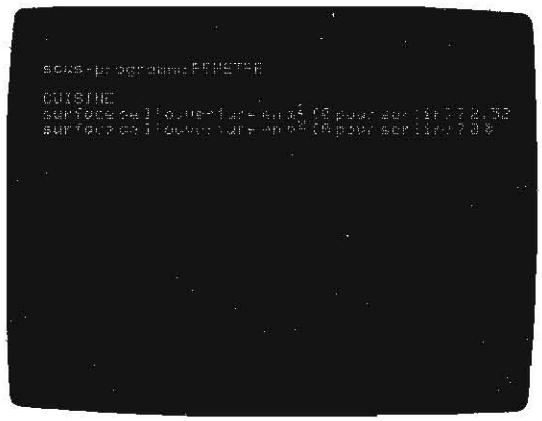
L'instruction 1000 provoque l'apparition de ce message à l'écran, ainsi que l'affichage du mot CUISINE, qui rappelle à l'opérateur quelle est la pièce en cours de traitement.

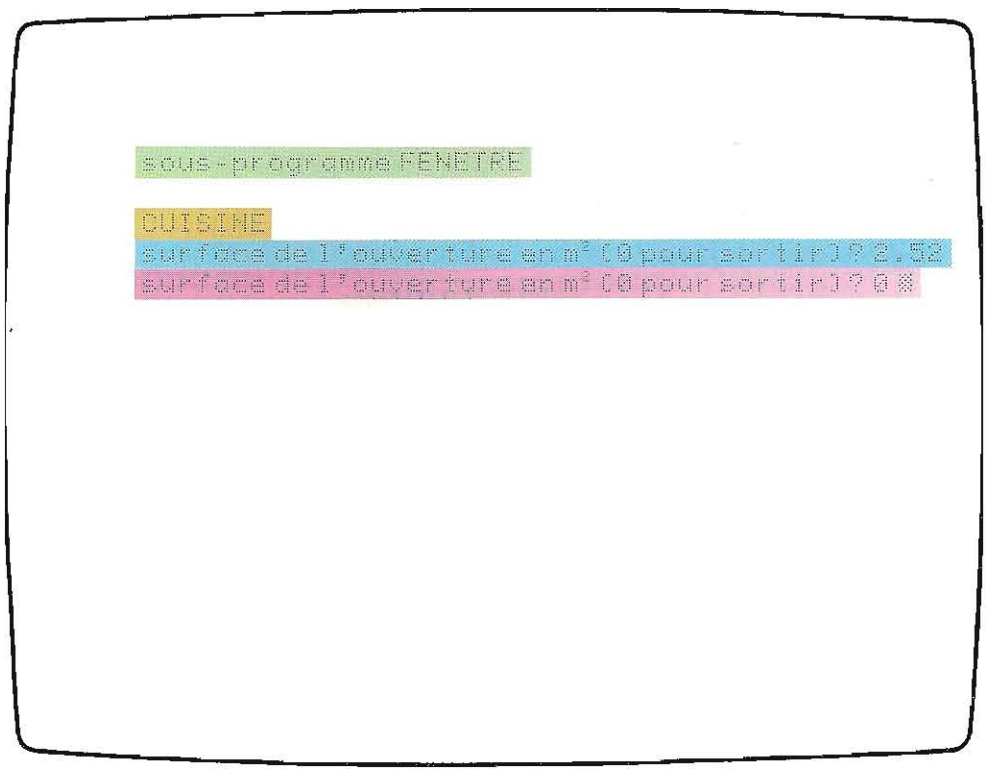
Ce dernier message indique à l'opérateur comment quitter la boucle de traitement des murs de la pièce. En effet, si la chaîne de caractères saisie est vide (l'opérateur a tapé seulement RETURN), on sort du sous-programme MUR, et on passe à la routine SOL.

Ici, l'opérateur entre successivement les informations qui lui sont demandées aux lignes 1020, 1040, 1060, 1150 et 1160. Remarquez que l'ordre de saisie des données est imposé par les messages qui accompagnent les instructions INPUT.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 4)

Lors de l'exécution du sous-programme 1000, la saisie d'une chaîne vide comme nom de mur, rend la main au programme principal, qui appelle alors le sous-programme SOL. Dans le cas contraire, on appelle ensuite la routine FENÊTRE, au cas ou le mur qui vient d'être traité contiendrait une ouverture.





Exécution de l'instruction 2010

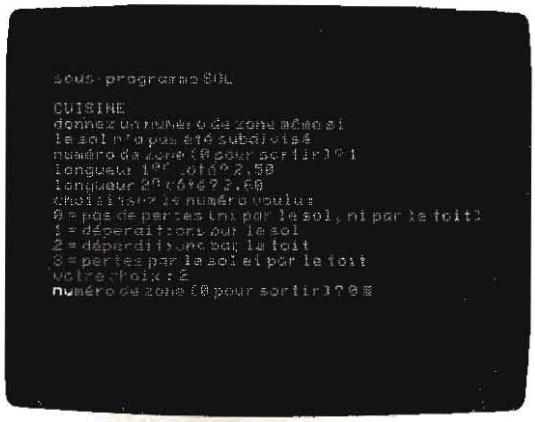
Le nom de la pièce est rappelé à l'opérateur.

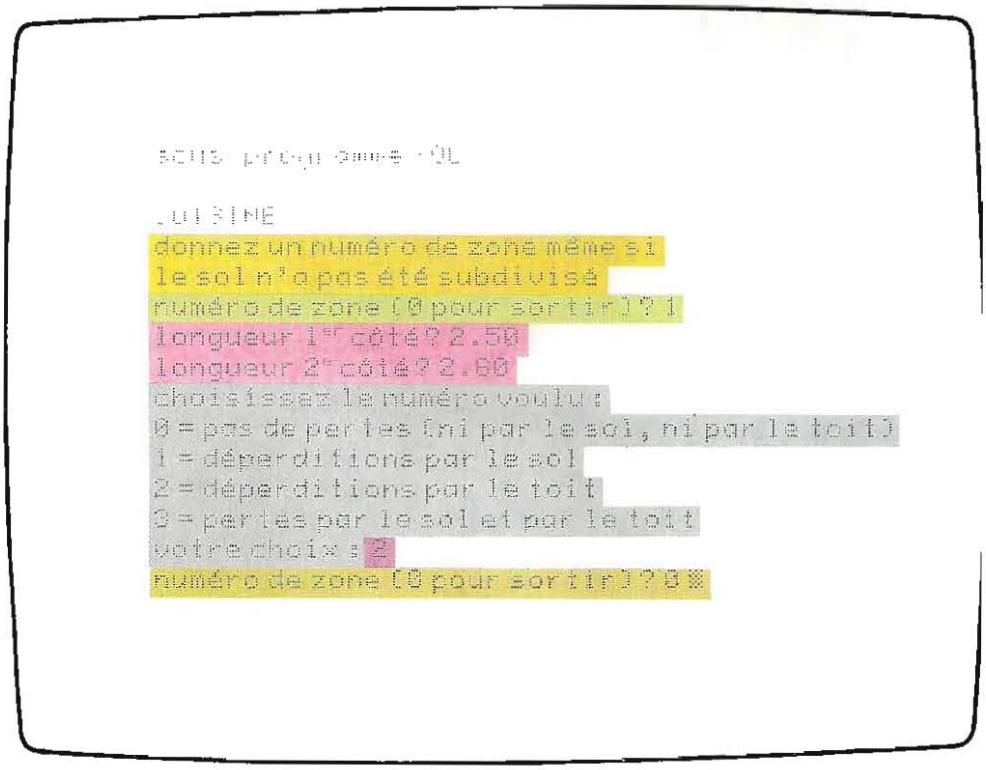
Ce message s'affiche à chaque passage par la boucle (saisies successives des surfaces des différentes fenêtres du mur). Il n'y a ici qu'une fenêtre (voir le plan de l'appartement).

La saisie du nombre 0 permet de quitter la boucle : on passe au calcul des déperditions de chaleur par les fenêtres (ligne 2070).

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 5)

Le sous-programme SOL, appelé à la ligne 320 du programme principal, constitue l'étape suivante. Différents messages s'affichent à l'écran.





Les lignes 4040 et 4050, en affichant ce message, guident l'opérateur afin que l'exécution du programme se déroule correctement.
Quand une pièce a une forme complexe, il est nécessaire de découper sa surface en plusieurs rectangles ou carrés ; les dimensions de chacune de ces parties permettront de calculer séparément leurs surfaces respectives, dont la somme donnera la surface totale

L'opérateur devra attribuer un numéro de zone au sol de la pièce, même lorsqu'elle n'est pas subdivisée.

Saisie du numéro de zone.

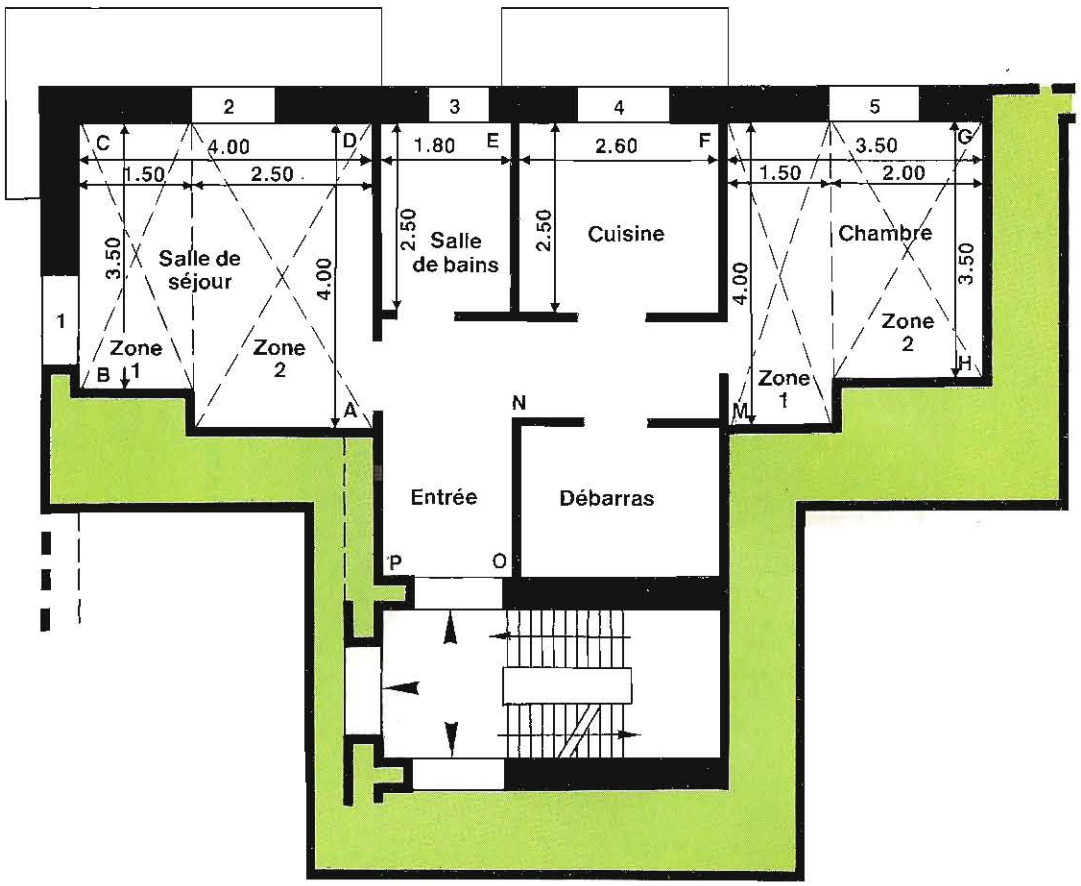
Saisie des longueurs des côtés d'une zone.

On demande maintenant à l'opérateur de préciser la situation de la zone considérée.

La réponse tapée au clavier signifie que, pour cette zone, la chaleur ne se dissipe que par le toit.

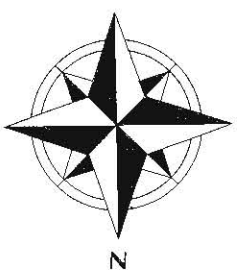
Les déperditions de chaleur sont calculées pour cette zone. L'instruction 4210 renvoie à la ligne 4060, pour traiter éventuellement une nouvelle ligne. La réponse de l'opérateur (0) détermine le retour au programme principal

PLAN DE L'APPARTEMENT



N.B. La chaleur ne se dissipe que par les murs BC, CD, DE, EF et FG, car les autres murs sont mitoyens avec d'autres appartements.

Légende: 1 = Fenêtre de 1.8 m²
2 = Porte-fenêtre de 2.52 m²
3 = Fenêtre de 1.2 m²
4 = Porte-fenêtre de 2 52 m²
5 = Fenêtre de 1.8 m²



Sol de la salle de séjour :	Zone 1 =	1.50 × 3.50
	Zone 2 =	2.50 × 4.00
Sol de la chambre	Zone 1 =	1.50 × 4.00
	Zone 2 =	2.00 × 3.50
Sol de la salle de bains	=	1.80 × 2.50
Sol de la cuisine	=	2.60 × 2.50

L'ordinateur à l'étable

On compte actuellement 210 millions de vaches dans le monde. La production laitière annuelle serait de 427 millions de tonnes, dont 5 millions de tonnes sous forme de lait en poudre, auquel il convient d'ajouter 7 millions de tonnes de beurre et 11 millions de tonnes de fromages. On imagine sans peine les problèmes que posent le transport et la distribution de telles quantités de denrées alimentaires à tous les gouvernements, et notamment à ceux de la Communauté Economique Européenne.

Confrontés aux mutations économiques et à l'augmentation incessante du coût global des aliments destinés au bétail, éleveurs et agriculteurs se voient contraints de planifier rigoureusement leur production. C'est pourquoi, à côté des vaches, apparaît aujourd'hui dans les fermes un nouveau pensionnaire : l'ordinateur.

Il représente, en effet, un outil précieux pour gérer les élevages de plus en plus importants des producteurs de lait. De nombreux autres éleveurs européens ont récemment abandonné ce secteur, encouragés par les subventions (prime à la reconversion) de la CEE, visant à résorber l'excédent de lait. Le cheptel de vaches laitières n'a pas diminué pour autant. Ainsi, en Grande-Bretagne, si les producteurs laitiers ont vu leur nombre baisser de 40 % depuis 1970, et ne sont plus aujourd'hui que 60.000 environ, l'exploitation moyenne compte 52 têtes de bétail contre 25 en 1965, et ce nombre ne cesse de croître.

La ***gestion*** *efficace d'un élevage passe nécessairement par la prévision de toutes les dépenses : frais généraux, investissements, nourriture et entretien. Dans ce but, un projet informatique a été mis au point, qui permet de prévoir un véritable calendrier du troupeau pour une année entière. Il est actuellement expérimenté au Royaume-Uni (où il a vu le jour), ainsi qu'aux Pays-Bas et dans les deux Irlandes. Les exploitants communiquent à un centre informatique situé en Grande-Bretagne, la production mensuelle de chaque vache, ainsi que des renseignements annexes (mises bas, accouplements...). L'ordinateur calculera, à partir de ces données, la production de chaque vache pendant le mois suivant, sa ration alimentaire journalière, ainsi que la date à laquelle cessera sa lactation.*

Des ***logiciels*** *comparables ont été conçus aux Etats-Unis, notamment avec le concours des University Agricultural Extension Services Centers. Dans l'un de ces centres, on détermine les caractéristiques de l'alimentation idéale pour l'hiver à venir. Les agriculteurs intéressés n'ont qu'à téléphoner pour entendre une voix synthétique (féminine !) leur fournir tous les renseignements voulus.*

Toujours aux Etats-Unis, l'Agricultural Economic Departement de l'Université du Michigan a mis en place une structure différente, fondée sur un système de temps partagé. On accède au Telplan par un téléphone un peu particulier, dont le clavier permet de communiquer des données à l'ordinateur en mode conversationnel.

Les producteurs adhérents pourront obtenir du Telplan une cinquantaine de ***plans d'action*** *adaptés à leur cas, qui les aideront, entre autres, à équilibrer les rations alimentaires de leur troupeau, à établir des budgets prévisionnels, ou qui les conseilleront dans leurs investissements. D'autres programmes du Telplan permettent de connaître les caractéristiques des différentes races bovines, et d'évaluer le bilan de production d'une vache.*

*Le gouvernement fédéral du Canada a ressenti depuis longtemps la nécessité de programmes de gestion mieux adaptés aux besoins des agriculteurs ; en 1965, a été créé le Canfarm, office à caractère régional et fédéral, visant le développement de l'agriculture sur l'ensemble du pays, à travers ses antennes régionales, ses établissements de formation et ses agences ouvertes au public. Les quelques 10.000 exploitants agricoles qui ont adhéré au Canfarm disposent aujourd'hui d'une large gamme de logiciels s'appliquant aux sujets les plus variés : les rations de fourrage, l'évolution prévisible des cours de la bourse, ou les derniers projets d'****élevages-modèle.***

Au Royaume-Uni également, plus de 20 % des producteurs laitiers utilisent l'informatique, d'une façon ou d'une autre, pour la gestion de leur exploitation. Ils envoient généralement leurs données par courrier au centre informatique de gestion auquel ils adhèrent. Ces données seront traitées sur de gros ordinateurs, ce qui explique les prix extrêmement élevés que pratiquent ces centres.

Mais une nouvelle tendance se dessine aujourd'hui. On commence à se tourner vers des systèmes de dimensions plus modestes, dont le prix raisonnable favorise la décentralisation. Le micro-ordinateur paraît réellement mieux adapté aux besoins des éleveurs laitiers, d'autant plus que la mise au point de «progiciels» spécifiques, ainsi que la connexion des ordinateurs à des dispositifs d'acquisition des données spécialement conçus, offrent des possibilités surprenantes. Parmi les systèmes informatiques de gestion à base de microprocesseurs, le Fullwood Ellesmere Electronics AFMS/80 est l'un des plus perfectionnés. C'est un logiciel d'application qui saisit et traite un grand nombre d'informations sur la gestion économique et technique d'un troupeau laitier.

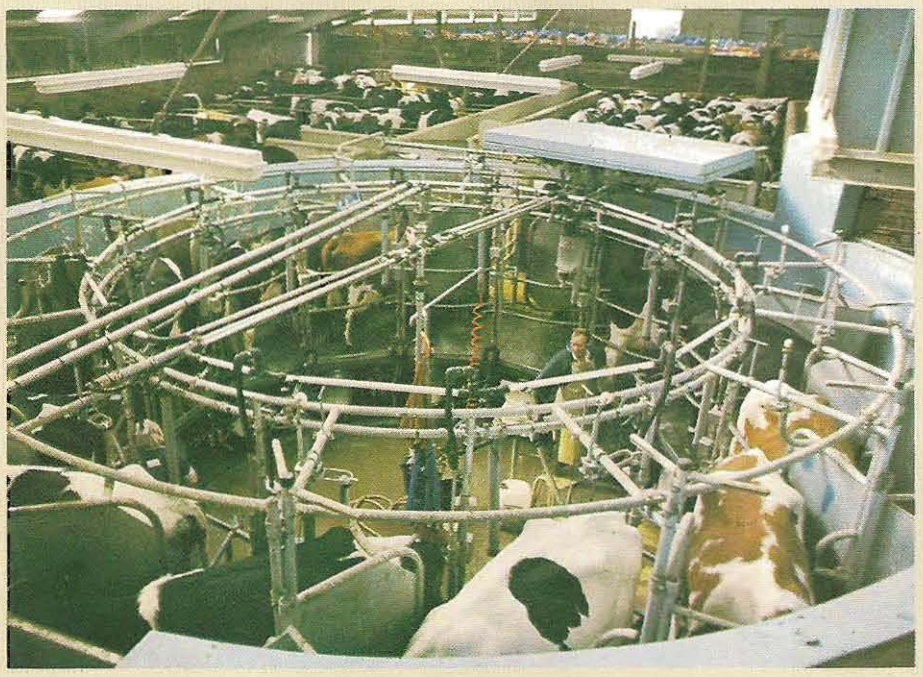
Tous les animaux du troupeau portent un collier à médaille magnétique, qui permet à l'ordinateur de les identifier. Lorsque les vaches sont parquées en stabulation libre (et non attachées), on pourra ainsi repérer les déplacements de chacune d'entre elles, grâce au signal caractéristique de sa médaille: un ***récepteur*** *adéquat captera ce signal (généralement par une antenne), l'interprétera et le transmettra au microprocesseur chargé de l'identification. On peut ainsi savoir à tout moment si une vache a mangé, ou si elle a été traite, et tenir le compte des quantités de nourriture qu'elle ingère et de lait qu'elle produit.*

D'autre part, chaque box possède une fermeture ***magnétique****, qui ne s'ouvrira que devant le collier qui lui correspond. Le système, con-*



G.R. Roberts

En haut: installations d'un établissement de production de lait en poudre.



Alan Hutchison Library

Ci-contre: la salle de traite (en manège) dans une ferme modèle. Eleveurs et producteurs de lait doivent de plus en plus souvent informatiser leur exploitation pour survivre dans le monde moderne.

Les camions-citernes de cette laiterie moderne ont une capacité de 225.000 litres.

Ci-dessous : le système Alfa Laval distribue aux vaches, quatre fois par jour et à heures fixes des quantités précises d'aliments concentrés. Une telle automatisation fait gagner du temps et supprime un travail pénible. De plus, sa régularité se traduit par une augmentation importante de la production de lait.



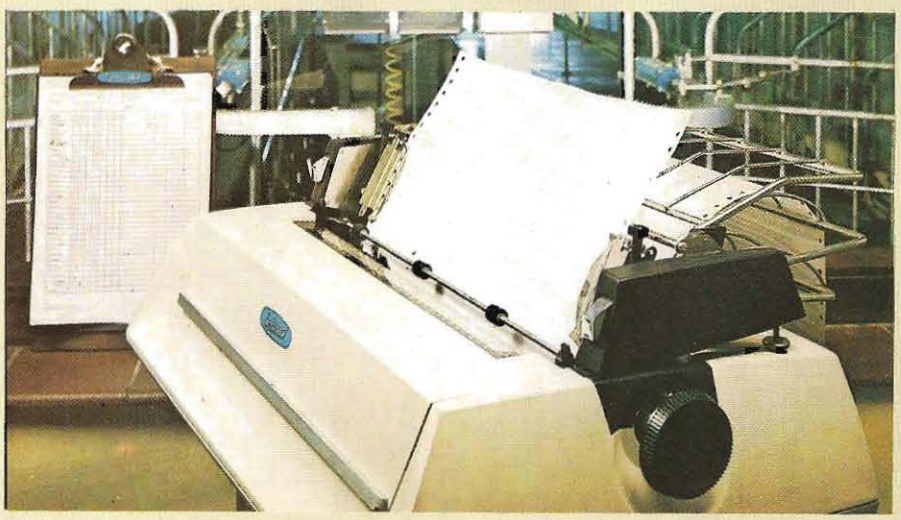
G.R. Roberts



Alfa Laval



R.J. Fullwood & Bland Ltd.



R.J. Fullwood & Bland Ltd.

Ci-dessus : l'ordinateur fournit le calendrier des travaux à accomplir, la production de lait mensuelle et la fiche signalétique de chaque vache.

Ci-contre : l'imprimante édite des listings retraçant les informations reçues et les opérations exécutées.

naissant les besoins alimentaires de la vache, distribue donc automatiquement, cas par cas, la ration de concentré adéquate, en commandant une vis à aliment ou une boîte doseuse. Au cours de ses déplacements, la vache passe sur une balance automatique qui transmet son poids et son identité à l'ordinateur central.

Dans l'avenir, on envisage de remplacer le collier magnétique par un autre dispositif, qui serait implanté en permanence sur l'oreille ou la poitrine de l'animal, et établirait une liaison plus étroite encore avec l'ordinateur.

Le système Fullwood mesure la production effective de lait individuelle et globale au moyen d'un dynamomètre relié aux récipients dans lesquels le lait est recueilli. Le poids est lu par un microprocesseur qui le transmet à l'ordinateur. Ces mesures peuvent s'effectuer simultanément sur 32 postes de traite.

Mais l'assistance informatique à la conduite d'un troupeau ne se limite pas à l'alimentation et à la mesure de la production laitière : les besoins de chaque animal varient en fonction de son état de santé et de plusieurs autres facteurs.

Dans les exploitations équilibrées du Fullwood System, tous les contrôles et soins spécifiques dont doit bénéficier chaque vache sont mémorisés par le système.

Tous les boxes de l'étable sont équipés d'un témoin lumineux, qui s'allume si la vache, identifiée en entrant, est reconnue par le système comme devant subir un traitement particulier. Ainsi, si une vache souffre de mammite et doit recevoir des soins vétérinaires, l'ordinateur rappellera ce fait à l'éleveur, en lui envoyant les instructions appropriées. Plus tard, il demandera le prélèvement d'un échantillon de lait, qui sera analysé pour vérifier qu'il n'est pas contaminé par les médicaments administrés à l'animal.

Bien sûr, le rôle du « vacher » des temps modernes ne se borne pas à exécuter les instructions de l'ordinateur. Il s'agit plutôt d'un **dialogue** permanent, où le système, constamment enrichi par le savoir que lui transmet l'éleveur, jouera le rôle d'un assistant à la mémoire infaillible.

L'ordinateur propose à l'exploitant une liste journalière des travaux à accomplir – tels que les traitements à administrer aux animaux – ainsi qu'un calendrier des événements plus exceptionnels, comme les mises bas, les tarissements ou les dates d'insémination... D'autre part, un panorama hebdomadaire pourra indiquer les examens à faire subir aux vaches pleines et aux veaux nouveaux-nés, et rassemblera éventuellement des renseignements tels que le nombre de vaches qui sont venues s'ajouter au troupeau ou qui en sont sorties, le nombre d'animaux bénéficiant de soins particuliers et la répartition de la production de lait en fonction des différents usages prévus. Enfin, une fois par mois, ou plus souvent si nécessaire, on calculera le **rendement laitier** du troupeau (ou « moyenne d'étable »), en tenant compte de la production totale, de la taille du troupeau et du pourcentage de vaches pleines.

Plus les élevages sont importants, plus il devient difficile pour le vacher de connaître individuellement les bêtes et surtout de se rappeler leurs caractéristiques. Le système fournira à la demande une véritable fiche signalétique de chaque vache : son nom, son ascendance et sa descendance, sa production actuelle et celle des derniers mois, les soins qu'elle a subis, et bien d'autres données encore.

Il ne semble pas raisonnable d'envisager un tel investissement avec un troupeau de moins de 50 vaches. Avec un tel nombre, il faut compter environ 1.500 F par vache (uniquement pour l'alimentation automatique), et 300 à 400 F par vache pour la gestion technico-économique du troupeau. Quels que soient les futurs acheteurs de ces systèmes sophistiqués, la clé de leur réussite résidera dans la façon dont ils auront compris leur utilisation. L'ordinateur aidera précieusement l'éleveur et lui permettra de mieux travailler, de mieux gérer, de mieux comprendre, donc elle sera d'autant plus efficace que l'homme saura s'en servir avec intelligence.

L'ordinateur a déjà fait ses preuves dans de nombreux secteurs de l'économie, et son application à la production laitière constitue un nouvel exemple de réussite. Toutefois, là comme ailleurs, rien ne peut remplacer l'**expérience** acquise par les éleveurs et les agriculteurs au fil de leur pratique quotidienne. Sans le concours des agriculteurs, il aurait d'ailleurs été impossible de mettre au point des logiciels capables de répondre à leurs besoins spécifiques.

Les tableaux

Les variables d'un programme sont des noms symboliques associés par le système d'exploitation à des zones mémoire. Il est souvent pratique de désigner plusieurs zones par un même nom générique. Ainsi, pour définir un tableau (array), comme dans le programme de calcul des déperditions thermiques (page 505), on réserve plusieurs zones mémoire contiguës, représentées par le même nom. On accèdera ensuite à chacun des éléments en complétant ce nom par l'indice (numéro d'ordre) de l'élément dans le tableau. En début de programme, on spécifiera le nom que le système doit affecter à cette zone de mémoire et le nombre d'éléments qu'elle contiendra (taille à réserver). L'instruction de dimensionnement est la suivante :

DIM NOM (N)

DIM (dimension) représente le code opération qui demande au système de réserver une partie de la mémoire pour le tableau, le temps de l'exécution du programme. Il est suivi du nom de la variable (ici, NOM) et du nombre d'éléments (N) qui est de type entier et s'appelle la « dimension » du tableau. L'instruction DIM TABL(20), par exemple, réserve une partie de mémoire pouvant contenir 20 éléments, et lui associe le nom TABL.

Pour accéder à un élément, on le désignera par son indice (c'est-à-dire son rang dans le tableau). Ainsi, l'instruction

R=6.8*TABL(4)

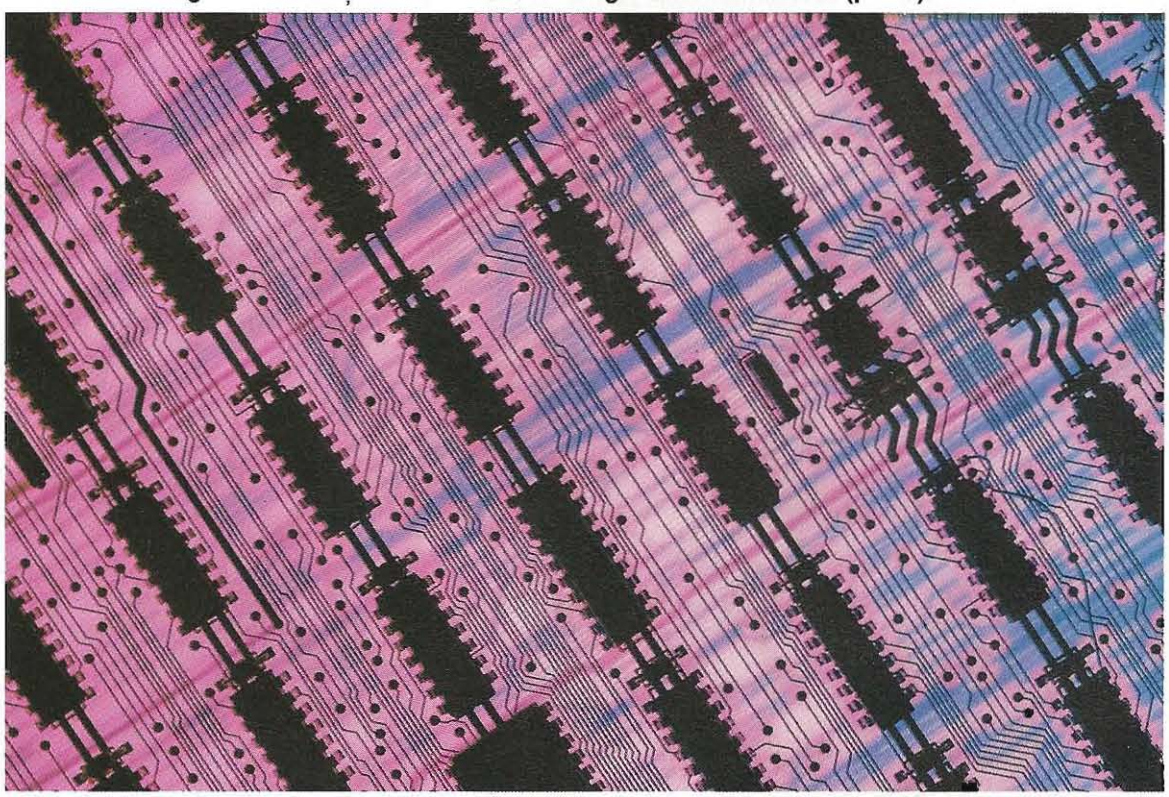
extrait la valeur située dans la quatrième case de la zone de variables TABL, la multiplie par 6.8 et affecte le résultat à R.

Il faut être très attentif à la numérotation des éléments d'un tableau : dans la plupart des Basic, le système « compte » en effet à partir de 0, et le premier indice n'est donc pas 1 (TABL(1)) représente alors le deuxième élément). Certains interpréteurs ou compilateurs Basic offrent au programmeur la possibilité de faire commencer la numérotation à 1, avec l'instruction OPTION BASE 1.

La coexistence de ces deux bases de numérotation des tableaux est expliquée dans le chapitre sur l'Assembleur.

Enfin, insistons sur le fait qu'un tableau est une variable au même titre qu'une autre : elle

Ces circuits intégrés sont conçus selon une technologie bidimensionnelle (plane).



Marka

possède donc un type déterminé.
Tous ses éléments doivent être homogènes : il est impossible, par exemple, de stocker dans un même tableau des entiers et des nombres en simple précision, ou bien des chaînes de caractères et des valeurs numériques.
Les différents types possibles, ainsi que les déclarations implicites (DEFINT, par exemple) seront les mêmes que pour les constantes ou les variables ordinaires, et toucheront alors tous les éléments du tableau.

Organisation des tableaux

Dans tous les programmes qui ont été présentés jusqu'ici, chaque variable possédait un nom propre et était mémorisée dans une zone réservée à elle seule, dont la longueur (nombre d'octets) dépendait du type de la variable elle-même : deux octets pour un entier, un octet pour chacun des caractères d'une chaîne, etc. Pour évaluer la place mémoire occupée par un tableau, il suffit de multiplier le nombre d'éléments qu'il peut contenir (dimension) par la longueur (en octets) d'un élément. Les instructions suivantes :

```
10 OPTION BASE 1
20 DEFINT A-K
30 DIM AR(3)
```

définissent un tableau nommé AR, qui contiendra trois éléments de type entier (l'instruction 20 fixe comme entières toutes les variables d'initiale comprise entre A et K; le tableau AR sera donc entier). La numérotation débute à 1, et le tableau occupera 6 octets en mémoire (2 par élément).
En Basic, il est possible de déclarer implicitement un tableau. Si l'on mentionne, au cours d'un programme, une variable indicée (tableau) sans l'avoir dimensionnée au préalable avec l'instruction DIM, le système lui affectera automatiquement une zone de mémoire. Dans l'exemple précédent, la ligne 30 qui déclare et dimensionne à 3 le tableau AR aurait pu être omise : à l'exécution, dès que le système aurait rencontré une instruction se référant à la variable indicée AR(N), il aurait réservé une zone de mémoire pour le tableau. Par exemple, l'instruction AR(3) = 4.7 aurait automatiquement créé en mémoire un tableau AR, avant d'effectuer l'affectation demandée (le troisième élément prend la valeur 4.7).
Ce mode de déclaration automatique possède cependant ses limites : le Basic Standard réservera systématiquement 10 emplacements pour un tableau dimensionné implicitement. L'instruction précédente [AR(3) = 4.7] sera correcte, mais AR(132) = 4.7 ne pourra pas dimensionner AR à 132 éléments. Dans ce cas, le programmeur devra écrire plus haut l'instruction : DIM AR(132).
Les tableaux de chaînes alphanumériques se déclarent et se dimensionnent de la même manière. La longueur d'une variable définie comme chaîne peut varier entre 0 et 128 caractères (ou 256 sur certains systèmes). Le système attribue automatiquement à la variable la place maximale (128 octets).
L'instruction DIM A$(3) réservera une zone de mémoire pour contenir trois chaînes distinctes, soit un encombrement physique de 128 × 3 = 384 octets au total.
Ainsi, le programme suivant :

```
10 OPTION BASE 1
20 DIM A$(3)
30 A$(1)="Première chaîne"
40 A$(2)="Deuxième"
50 A$(3)="Dernière"
60 END
```

crée en mémoire un tableau de trois chaînes [DIM A$(3)] de 128 octets chacune. La première contient 15 caractères (ligne 30 ; il faut tenir compte de l'espace entre les deux mots), la seconde 8 caractères (ligne 40) et la troisième également. L'occupation effective est donc de 15 + 8 + 8 = 31 octets, alors que 384 sont réservés. Toute variation de longueur de l'une des chaînes modifiera la longueur totale : si l'on pose maintenant A$(3)="Dernière chaîne", 38 octets seront occupés (7 caractères ont été ajoutés).
Il ressort clairement de cet exemple que la méthode utilisée (réservation systématique de 128 octets pour chaque chaîne) occasionne parfois un « gaspillage » de la place mémoire. Sur certains systèmes, il est possible d'imposer aux chaînes une longueur maximale, c'est-à-dire de les dimensionner.
Voici une caractéristique du Basic, qui donne une signification particulière à l'instruction

DIM, lorsqu'elle est appliquée à une variable de type chaîne. En effet, DIM CH$ [8] réservera 8 octets pour la chaîne CH$. La même formulation s'appliquera aux tableaux de chaînes : ainsi, l'instruction DIM A$(3) [8] réserve 24 octets pour trois chaînes, chacune de huit caractères au maximum. Dans ce cas, l'instruction 30 du programme ci-dessus provoquerait une erreur, en cherchant à affecter à A$(1) une chaîne de plus de huit caractères. Précisons toutefois que, comme la notion de « sous-chaînes », le dimensionnement des variables alphanumériques n'est possible que sur les systèmes les plus perfectionnés. Nous reviendrons sur ce point dans le chapitre consacré aux variantes du Basic (par rapport au Basic standard) proposées sur les différents micro-ordinateurs.

Tableaux à une et à N dimensions

L'instruction DIM AR(3) crée un tableau de trois éléments successifs : il n'a qu'une dimension.
Cette notion de dimension devient plus claire si l'on assimile les tableaux à des figures géométriques : une ligne n'a qu'une dimension, une figure plane en a deux (longueur et largeur) et un solide trois (longueur, largeur et hauteur). De même, un tableau sera organisé selon une dimension, que nous pourrons appeler sa « longueur », ou bien deux (sa longueur et sa largeur). S'il possède trois dimensions, on parlera de sa profondeur.
Le langage informatique n'emploie pas ces termes géométriques pour désigner les dimensions d'un tableau. Lorsque ce dernier possède deux dimensions, les informaticiens le découpent en lignes et colonnes, chaque élément se trouvant à l'intersection d'une ligne et d'une colonne. Quant à la troisième dimension (qui correspond à la profondeur), on l'appellera un plan (ou une page). Les deux premières dimensions définissent des pages divisées en lignes et en colonnes, qui se succèderont le long de la troisième dimension.
Pour stocker simplement une série de données, un tableau à une dimension est amplement suffisant. Ainsi, pour mémoriser les dépenses familiales quotidiennes pendant un mois, on dimensionnera un tableau à 31 éléments, le premier contenant le total des dépenses du premier jour du mois, le second celui du deuxième et ainsi de suite. Par contre, un tableau à deux dimensions sera nécessaire pour réaliser une table de multiplication : on stockera le produit I*J à l'intersection de la ligne I et de la colonne J. Ensuite, pour connaître le produit des nombres A et B, on lira le contenu de la case (A,B) du tableau. Les tableaux à trois dimensions seront indispensables pour certaines applications de gestion, mais leur emploi se justifie rarement.
La page 523 représente schématiquement trois tableaux ayant respectivement 1, 2 et 3 dimensions. Voici les déclarations correspondantes :

- DIM A(4)
  Tableau à une dimension, contenant quatre éléments
- DIM B(3,5)
  Tableau à deux dimensions, contenant $3 \times 5 = 15$ éléments, répartis sur 3 lignes et 5 colonnes
- DIM C (3,3,5)
  Tableau à trois dimensions, contenant $3 \times 3 \times 5 = 45$ éléments, répartis sur 3 pages organisées en 3 lignes et 5 colonnes chacune.

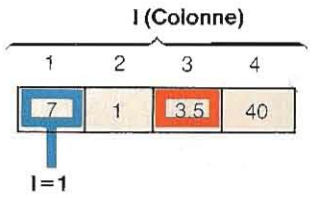
Un tableau peut posséder plus de trois dimensions (il n'existe plus de support géométrique intuitif dans ce cas !) mais il devient alors très lourd à manier. Il est généralement préférable d'organiser ses données selon plusieurs tableaux de trois noms différents. De plus, les opérations que devra effectuer le système d'exploitation pour manipuler les tableaux seront toujours plus simples et plus vite exécutées lorsque le nombre de dimensions est plus réduit.
En outre, beaucoup d'interpréteurs n'acceptent qu'un nombre limité de dimensions.

Instructions d'affectation sur des tableaux

Pour affecter des valeurs aux différents éléments d'un tableau, on procèdera de la même façon que pour les variables ordinaires : soit avec le symbole d'affectation directe =, soit par l'instruction DATA.
Voici, par exemple, un programme qui affecte aux éléments du tableau A les valeurs 7, 1, 3.5 et 40 (voir page 523, schéma du haut) :

QUELQUES EXEMPLES DE VARIABLES STRUCTUREES (TABLEAUX)

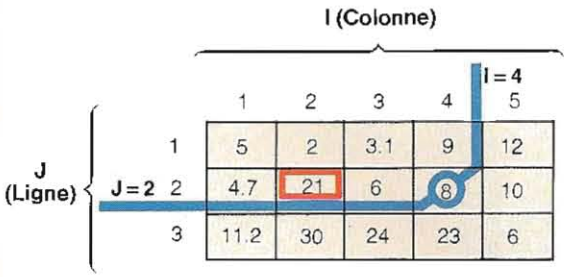
Une dimension



DIM A(4)
Exemple d'affectation :
A(3)=3.5

Exemple de lecture :
Si l'on pose I=1 alors A(I)=7

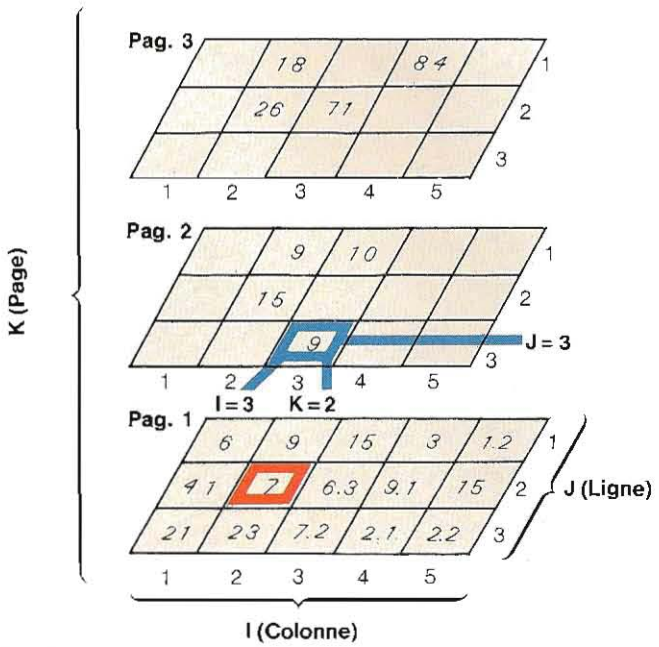
Deux dimensions



DIM B(3,5)
Exemple d'affectation :
A(2,2)=21

Exemple de lecture :
Si l'on pose J=2 et I=4 alors A(J,I)=8

Trois dimensions



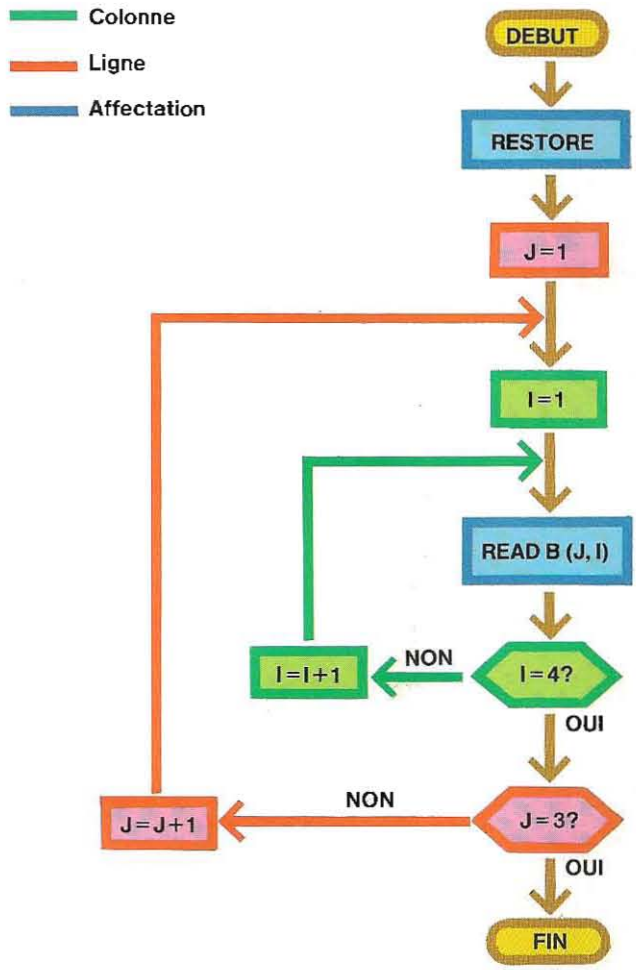
DIM C(3,3,5)

On peut considérer le premier indice comme le numéro de page, et les deux autres comme les numéros de ligne et de colonne dans la page.

Exemple d'affectation :
A(1,2,2,)=7

Exemple de lecture :
Si l'on pose I=3,J=3 et K=2 alors A(K,J,I)=9

REMPLISSAGE D'UN TABLEAU B(3,4) A DEUX DIMENSIONS (ORGANIGRAMME)




```
10 OPTION BASE 1
20 ' La numérotation commence à 1
30 DIM A(4)
40 A(1)=7:A(2)=1:A(3)=3.5:A(4)=40
50 END
```

Le même résultat sera obtenu avec les instructions suivantes :

```
10 OPTION BASE 1
20 ' La numérotation commence à 1
30 RESTORE 70
40 FOR I=1 TO 4
50 READ A(I)
60 NEXT I
70 DATA 7,1,3.5,40
80 END
```

La boucle constituée par les lignes 40, 50 et 60 affecte, l'une après l'autre, les valeurs contenues dans le DATA (ligne 70) à la case d'indice I du tableau A. La première donnée du DATA (7), est lue quand I = 1, et A(1) prend donc la valeur 7. De même, quand I vaut 2, la valeur 1 est affectée au deuxième élément du tableau [A(2) = 1] et ainsi de suite.

Vous trouverez pages 524 et 525 des organigrammes illustrant le déroulement logique du remplissage de tableaux à plusieurs dimensions.

Les listings correspondants se trouvent pages 526 et 527.

De la même manière, on peut également remplir un tableau de chaînes alphanumériques

au moyen de l'instruction DATA :

```
10 OPTION BASE 1
20 DEFINT I-N
30 DIM A$(3)
40 RESTORE
50 FOR I = 1 TO 3
60 READ A$(I)
70 NEXT I
80 DATA "Première ligne"
```

REMPLISSAGE D'UN TABLEAU A(3,3,5) A TROIS DIMENSIONS (ORGANIGRAMME)



REMPLISSAGE D'UN TABLEAU BIDIMENSIONNEL (LISTING)

```
10 ' ** PROGRAMME : TABLEAU BIDIMENSIONNEL **
15 '
20 '              FICHIER = BIDIM
25 '
30 OPTION BASE 1
40 DIM A(3,4) ' Déclaration du tableau
50 RESTORE 110
60 FOR J = 1 TO 3 ' Lecture
70 FOR I = 1 TO 4
80 READ A(J,I)
90 NEXT I
100 NEXT J
110 DATA 11,12,13,14,21,22,23,24,31,32,33,34
120 '
130 FOR J = 1 TO 3 ' Ecriture
140 LPRINT "LIGNE: ";J
150 FOR I = 1 TO 4
160 LPRINT A(J,I);
170 NEXT I
180 LPRINT ' Saut de ligne
190 NEXT J
200 END
```

```
LIGNE: 1
 11  12  13  14
LIGNE: 2
 21  22  23  24
LIGNE: 3
 31  32  33  34
```

REMPLISSAGE D'UN TABLEAU TRIDIMENSIONNEL (LISTING)

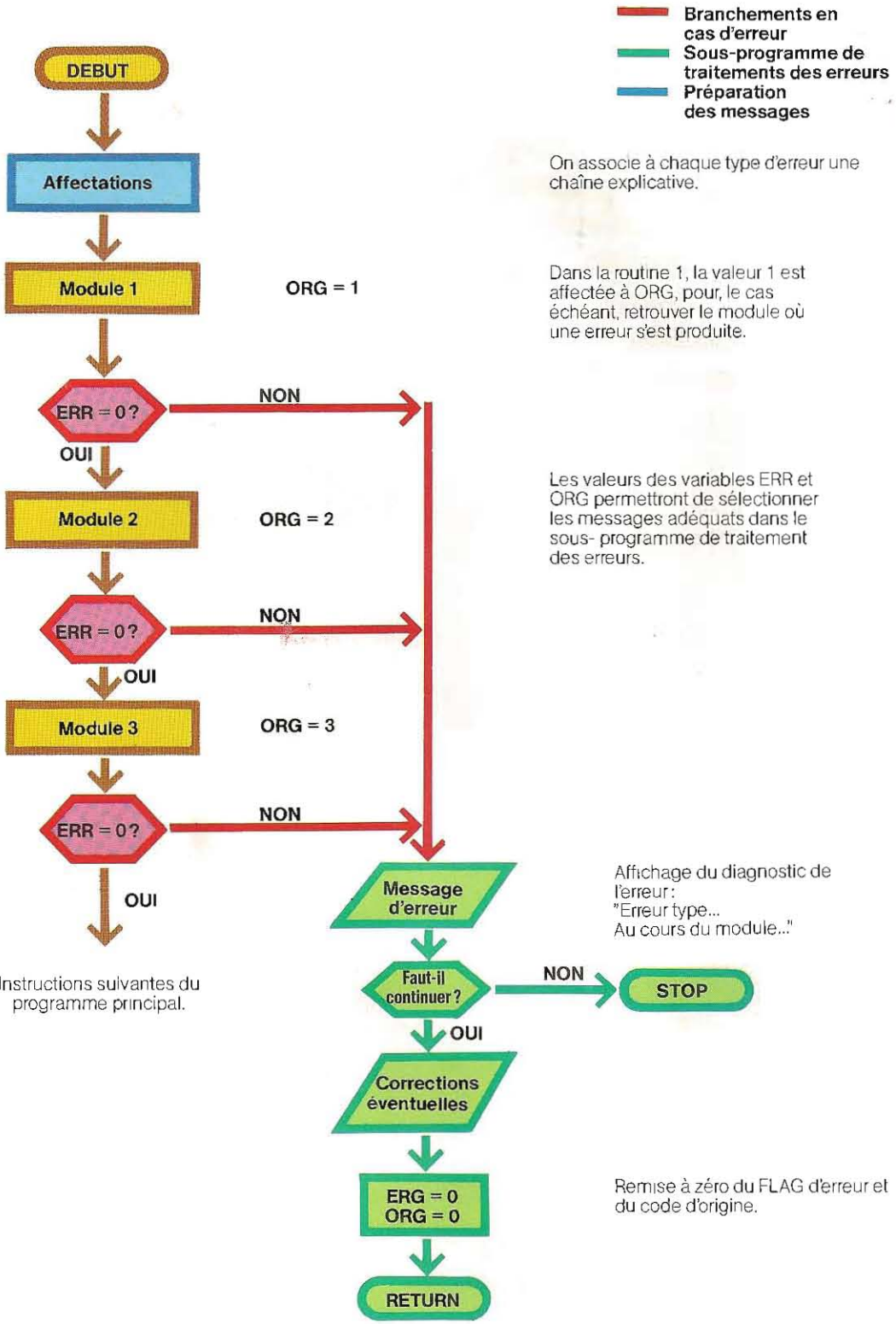
```
10 ' ** PROGRAMME : TABLEAU TRIDIMENSIONNEL **
15 '
20 '              FICHIER = TRIDIM
25 '
30 OPTION BASE 1
40 DIM A(3,3,4) ' Déclaration du tableau
50 RESTORE 140
60 '
70 FOR K = 1 TO 3
80 FOR J = 1 TO 3
90 FOR I = 1 TO 4
100 READ A(K,J,I)
110 NEXT I
120 NEXT J
130 NEXT K
140 DATA 11,12,13,14,21,22,23,24,31,32,33,34
150 DATA 51,52,53,54,61,62,63,64,71,72,73,74
160 DATA 41,42,43,44,81,82,83,84,91,92,93,94
165 '
170 FOR K = 1 TO 3          ' Ecriture
180 LPRINT " PAGE :  ";K
190 FOR J = 1 TO 3
200 LPRINT "LIGNE:  ";J
210 FOR I = 1 TO 4
220 LPRINT A(K,J,I);
230 NEXT I
240 LPRINT
250 NEXT J
260 LPRINT
270 NEXT K
280 END
```

```
PAGE: 1
LIGNE: 1
11  12  13  14
LIGNE: 2
21  22  23  24
LIGNE: 3
31  32  33  34

PAGE: 2
LIGNE: 1
51  52  53  54
LIGNE: 2
61  62  63  64
LIGNE: 3
71  72  73  74

PAGE: 3
LIGNE: 1
41  42  43  44
LIGNE: 2
81  82  83  84
LIGNE: 3
91  92  93  94
```

```
90 DATA "Deuxième"
100 DATA "Dernière"
110 END
```

Ce programme affecte aux éléments du tableau A$ les trois chaînes suivantes :
A$(1) = "Première ligne"
A$(2) = "Deuxième ligne"
A$(3) = "Dernière".
Un tableau de chaînes de caractères sera très utile pour la gestion des messages d'erreur. Au cours de l'exécution d'un programme, différentes erreurs peuvent être diagnostiquées ; certaines sont « rattrapables » par l'opérateur, surtout en phase de saisie de données ou de paramètres qui doivent ensuite servir au traitement.
Nous avons vu précédemment qu'en mode interactif, il est indispensable de signaler les erreurs éventuelles à l'opérateur, en émettant des messages qui apparaîtront à l'écran et lui expliqueront clairement la conduite à suivre pour les corriger.
On peut insérer ces messages d'erreur à n'importe quel endroit du programme, au fur et à mesure des besoins ; mais ceux-ci devront alors être répétés chaque fois qu'une même erreur peut se produire, et le programme principal va s'allonger et devenir illisible.
Il est beaucoup plus pratique de regrouper tous ces messages dans un tableau. Un sous-programme de diagnostic et d'affichage sera appelé dès qu'une erreur quelconque aura été commise.
Le code numérique affecté à l'erreur permet alors de sélectionner le message approprié dans le tableau.
En outre, il peut être intéressant de faire figurer à côté du message (destiné à l'utilisateur), un symbole qui indique le type d'erreur en cause.
Le travail de maintenance du programmeur se trouvera, dès lors, grandement facilité, grâce à une structuration optimisée du logiciel (ensemble des programmes).
En page 528, se trouve l'organigramme simplifié d'un programme construit de cette façon.
Les opérations à exécuter sont réparties en trois modules (routines), qui contiennent chacun des instructions de contrôle et de validation des données à traiter.
Si celles-ci ne répondent pas aux critères requis, une erreur sera diagnostiquée, et le code numérique correspondant sera affecté à la variable ERR, qui, dans le cas contraire, reste initialisée à la valeur 0. D'autre part, le numéro du module où l'erreur a été commise est affecté à la variable ORG.
Ainsi, lorsqu'à un moment donné on entre dans le module de traitement des erreurs avec les valeurs : ERR = 4, ORG = 2, on sait qu'il est survenu une erreur de type 4, au cours de l'exécution du module 2 (c'est le programmeur qui définit lui-même les erreurs et leurs numéros). On pourra alors afficher le message voulu. Les diagnostics d'erreur et les noms des modules sont déterminés dans le bloc « Affectations ».

STRUCTURE D'UN PROGRAMME POUR PREVOIR ET CONTROLER LES ERREURS



Pages 529 et 530, est listé un programme simulant la logique que nous venons d'exposer; les routines 1000, 2000 et 3000 demandent à l'utilisateur l'erreur qu'il veut générer et en produisent les résultats correspondants (pages 530 et 531).

Application des tableaux. Les variables structurées servent très souvent en programmation. L'organisation des données sous forme de tableaux simplifie considérablement l'écriture du programme, et le rend plus lisible.

Dans les pages qui suivent, nous présenterons deux applications typiques.
La première concerne l'utilisation de tableaux pour effectuer des calculs sur des variables numériques possédant un nombre quelconque de chiffres significatifs. La seconde nous donnera l'occasion d'introduire certaines notions de statistique appliquée, qui devraient faire partie du bagage technique de tout programmeur. La connaissance de ces notions n'est toutefois pas indispensable à la poursuite de notre étude.

PROGRAMME DE SIMULATION DES ERREURS

```
10 ' **PROGRAMME DE SIMULATION DES ERREURS **
20 '
30 '            PROGRAMME PRINCIPAL
40 '            FICHIER = SIMERR
50 '
60 ' **DECLARATIONS ET AFFECTATIONS **
70 OPTION BASE 1
80 DEFINT A-R          ' Les variables d'initiales comprises
90 '                   entre A et R sont entieres
100 DIM D$(4)          ' D$ contient quatre chaines
102 '                   de description des diagnostics
110 DIM R$(3)          ' Les trois chaines R$ contiennent, en entier,
112 '                   les noms des sous-programmes
120 RESTORE 190        ' Pointer au premier DATA (diagnostics)
130 FOR I = 1 TO 4     ' Lecture des chaines D$
140 READ D$(I)
150 NEXT I
160 '
170 ' **Les DATA suivants contiennent des diagnostics quelconques
172 '    a titre d'exemple.
190 DATA "Erreur de lecture ou d'ecriture sur disque - "
200 DATA "Choix de donnees non valides - "
210 DATA "Choix d'une option non prevue - "
220 DATA "Le mot de passe n'est pas reconnu - "
230 '
240 RESTORE 320        ' Pointer au deuxième DATA (noms des sous-programmes)
250 FOR I = 1 TO 3     ' Lecture des chaines R$
260 READ R$(I)
270 NEXT I
280 '
290 ' **Les noms sont donnes a titre d'exemple. En realite,
300 '    ces sous-programmes n'executent aucune fonction
310 '
320 DATA "Premier sous-programme"
330 DATA "Calcul interets"
340 DATA "Sous-programme d'essai"
350 '
360 ' **Les instructions RESTORE aux lignes 120 et 240
370 '    ne sont pas absolument necessaires
380 ERR = 0 : ORG = 0          ' Initialisation des flags
390 GOSUB 1000                 ' Appeler le premier sous-programme
400 IF ERR <> 0 GOSUB 9000     ' En cas d'erreur, appeler la ligne 9000
410 GOSUB 2000                 ' Deuxième sous-programme
420 IF ERR <> 0 GOSUB 9000
430 GOSUB 3000                 ' Troisième sous-programme
440 IF ERR <> 0 GOSUB 9000
480 ' **FIN DU PROGRAMME
490 '
500 END
```

```
1000 ' ** ENTREE DANS LE PREMIER SOUS-PROGRAMME
1010 ' Demande l'erreur a simuler et retourne au programme principal
1020 ORG = 1              ' FLAG D'ORIGINE DE L'ERREUR
1030 INPUT "Entrer le code de l'erreur a simuler ";ERR
1040 RETURN
1050 ' Les sous-programmes 2000 et 3000 sont semblables : seul ORG change
1060 '
2000 ' ** DEUXIEME SOUS-PROGRAMME
2010 ORG = 2
2020 INPUT "Entrer le code de l'erreur a simuler ";ERR

2030 RETURN
3000 ' ** TROISIEME SOUS-PROGRAMME
3010 ORG = 3
3020 INPUT "Entrer le code de l'erreur a simuler ";ERR
3030 RETURN
9000 ' ** SOUS-PROGRAMME DE TRAITEMENT DES ERREURS
9010 LPRINT "Erreur n. : ";ERR
9020 LPRINT "Provenance : ";ORG
9030 '
9040 ' ** Les erreurs prevues sont comprises entre 1 et 4 (voir instructions 190-220)
9050 '
9060 IF ERR > 4 OR ERR < 1 GOTO 9240      ' Erreur non prevue
9070 '
9080 ' La valeur de ERR permet de selectionner le message correspondant
9100 '
9110 ' En regle generale, le message est contenu dans D$(ERR)
9120 '
9130 A$ = D$(ERR)         ' Transfere le diagnostic en A$
9140 A$ = A$+" "          ' Ajoute un espace pour separer du prochain mot
9150 '
9160 ' La valeur de ORG est utilisee de maniere analogue a ERR
9170 ' pour selectionner la description du sous-programme
9180 '
9190 IF ORG > 3 OR ORG < 1 GOTO 9250      ' Provenance non prevue
9200 A$ = A$+R$(ORG)                      ' Ajoute a la chaine A$ la provenance
9210 LPRINT A$                            ' Ecrit le diagnostic en entier
9220 ERR = 0 : ORG = 0
9230 RETURN
9240 LPRINT "Erreur non prevue"
9250 LPRINT "Provenance non prevue"
```

```
Erreur n. : 1
Provenance : 1
Erreur de lecture ou d'ecriture sur disque - Premier sous-programme
Erreur n. : 1
Provenance : 2
Erreur de lecture ou d'ecriture sur disque - Calcul interets
Erreur n. : 1
Provenance : 3
Erreur de lecture ou d'ecriture sur disque - Sous-programme d'essai

Erreur n. : 2
Provenance : 1
Saisie de donnees non valides - Premier sous-programme
Erreur n. : 2
Provenance : 2
Saisie de donnees non valides - Calcul interets
Erreur n. : 2
Provenance : 3
Saisie de donnees non valides - Sous-programme d'essai
```

```
Erreur n. : 3
Provenance : 1
Choix d'une option non prevue - Premier sous-programme
Erreur n. : 3
Provenance : 2
Choix d'une option non prevue - Calcul interets
Erreur n. : 3
Provenance : 3
Choix d'une option non prevue - Sous-programme d'essai

Erreur n. : 4
Provenance : 1
Choix d'une option non prevue - Premier sous-programme
Erreur n. : 4
Provenance : 2
Choix d'une option non prevue - Calcul interets
Erreur n. : 4
Provenance : 3
Choix d'une option non prevue - Sous-programme d'essai

Erreur n. : 9
Provenance : 1
Erreur non prevue
```

Exécution de calculs avec un nombre quelconque de chiffres significatifs. Dans le cadre des applications pratiques courantes, l'exactitude des calculs obtenus avec des nombres en double précision est généralement satisfaisante.

Toutefois, il s'avère parfois nécessaire d'exécuter les calculs avec une meilleure précision. Différentes méthodes permettent de résoudre ce problème, mais toutes impliquent des difficultés de programmation parfois considérables.

La question de la précision des résultats doit être envisagée selon deux points de vue différents : d'une part le domaine scientifique, et d'autre part les problèmes de calculs économiques.

Dans le cadre des applications scientifiques, on ne devrait pas rencontrer de difficultés liées à la précision des calculs : on décrit des phénomènes physiques réels (et donc mesurables), et l'on connaît d'avance l'ordre de grandeur des variables qui les représentent. De plus, on sait que les dernières décimales d'un résultat n'ont absolument aucune signification physique.

Ainsi, lorsque l'exécution d'un calcul demande l'utilisation de précisions trop élevées, on doit en rechercher la cause dans la conception même du programme (algorithme), ou dans une mauvaise utilisation, et non dans une trop faible précision de la machine.

Dans le domaine économique, par contre, on raisonnera de façon opposée : pour présenter un bilan, ou tout autre document comptable, les résultats doivent apparaître jusqu'à la dernière décimale, quelque soit la grandeur du nombre (tous les chiffres sont significatifs). Prenons un exemple : d'un point de vue physique, estimer la vitesse d'une voiture à 100 km/heure, au lieu de 100.02 km/heure, se justifie aisément ; en revanche, si le nombre représentait le montant d'une facture participant au calcul d'un bilan, l'arrondi provoquerait une erreur dans la vérification définitive des comptes.

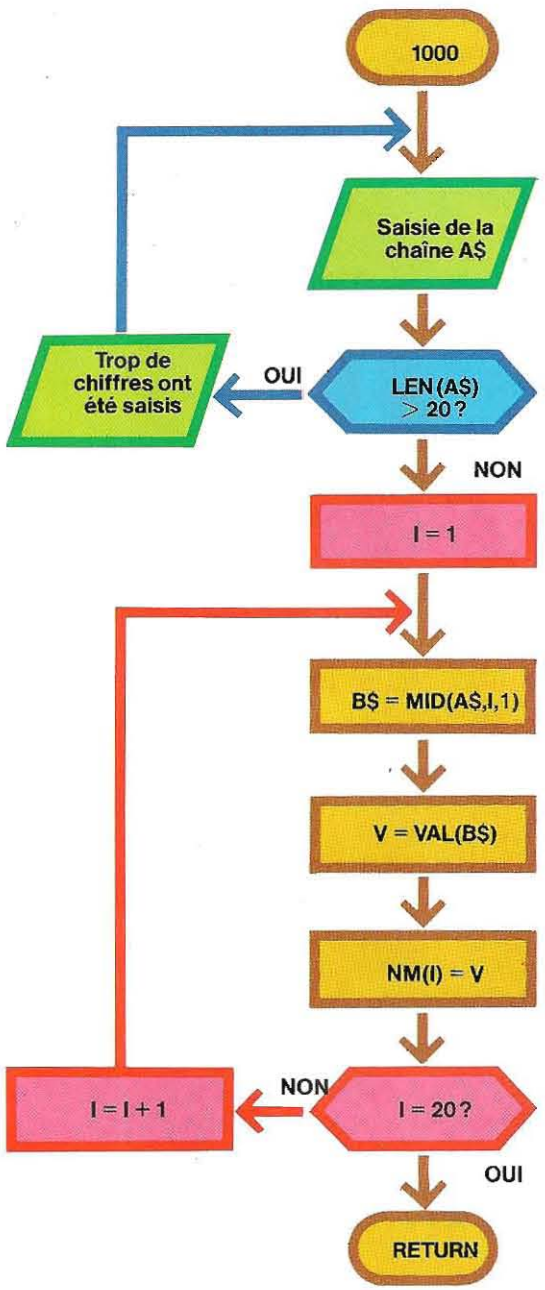
Une méthode simple pour se dégager des limitations de la machine (18 chiffres significatifs au maximum sur les plus gros systèmes), revient à considérer chaque nombre comme une chaîne de caractères. De cette manière, le nombre maximal de chiffres qu'il est possible de traiter augmente, selon les systèmes, jusqu'à 128 ou 256 (longueur maximale d'une chaîne).

Examinons maintenant une méthode permettant d'additionner deux nombres de longueur quelconque, saisis et représentés en mémoire sous forme de chaînes. Si l'on voulait entrer dans l'ordinateur un nombre de 20 chiffres en format numérique, celui-ci serait tronqué pour respecter la précision de la machine.

Ce même nombre, lu en tant que variable

alphanumérique, sera directement et entièrement saisi, comme une quelconque chaîne de 20 caractères.
On extraira successivement chaque caractère ASCII de la chaîne, on le transformera en valeur numérique (instruction VAL), pour le stocker ensuite dans un tableau.
De cette manière, la représentation numérique en mémoire n'est plus une valeur décimale (un nombre unique), mais un ensemble de chiffres séparés, chacun dans une case mémoire distincte. En machine, les problèmes

SOUS-PROGRAMME DE SAISIE D'UN NOMBRE EN « PRECISION ETENDUE »



Boucle de conversion des chiffres

Fonctions E/S

Contrôle des saisies

La chaîne A$ contient le nombre sous forme de caractères.

Ce contrôle est nécessaire pour ne pas déborder du tableau qui devra contenir les valeurs numériques. Pour accroître la précision, il suffit d'augmenter les dimensions du tableau NM.

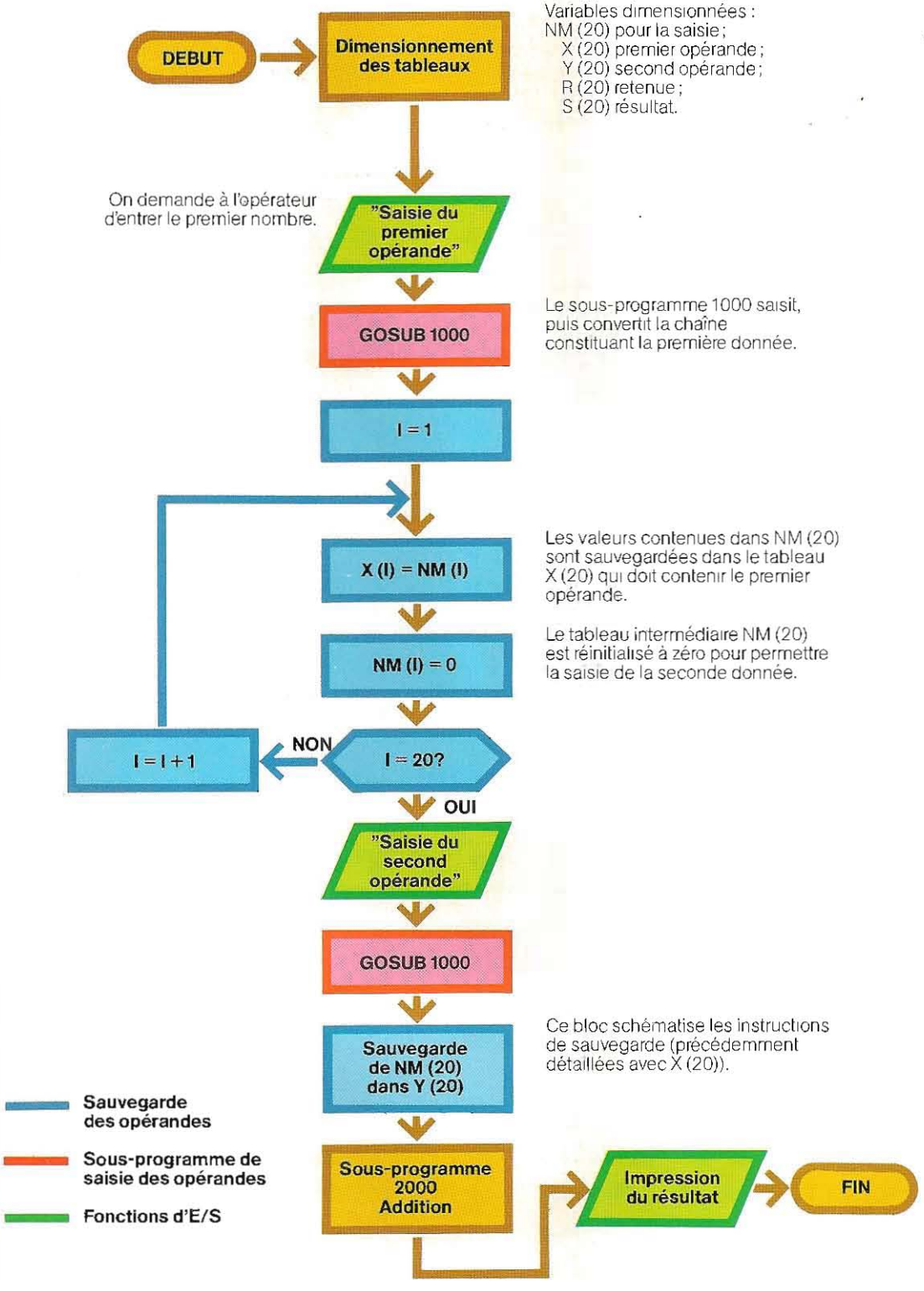
Extrait de A$ le caractère en position I.

Le convertit en valeur numérique.

Chaque chiffre du nombre initial est mémorisé dans un élément du tableau NM (20).

En sortie du sous-programme, le tableau NM (20) contient le nombre saisi (un chiffre par élément).

ADDITION DE DONNEES NUMERIQUES EN « PRECISION ETENDUE »



de précision disparaissent : le nombre est représenté par un tableau, chaque élément contenant un seul chiffre compris entre 0 et 9. L'organigramme page 532 retrace le déroulement du sous-programme 1000, qui saisit, sous forme de chaîne, un nombre allant jusqu'à 20 chiffres, puis convertit chaque caractère en valeur numérique, pour le stocker dans le tableau intermédiaire NM.

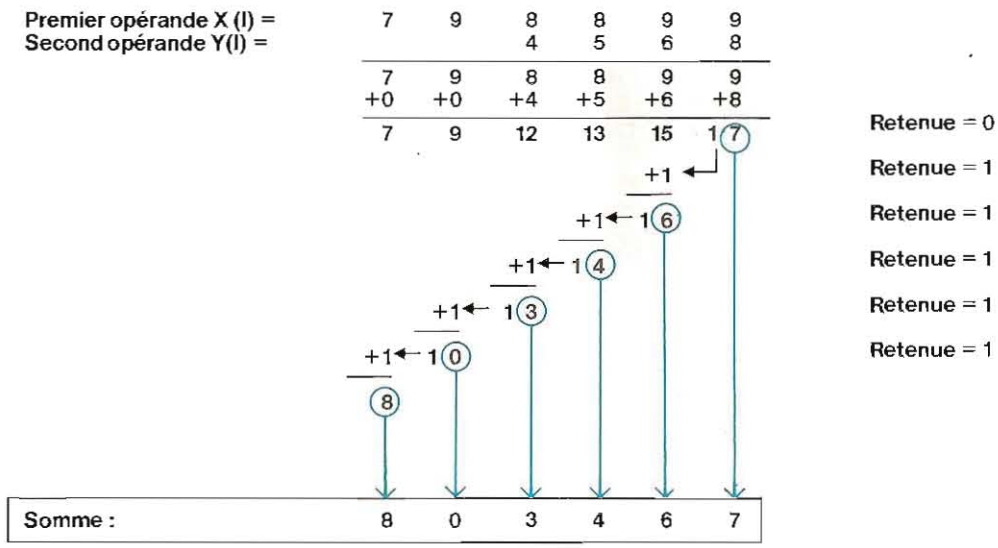
Page 533, est représenté l'organigramme du programme principal, contenant deux appels successifs au sous-programme précédent, puis un appel à la routine d'addition. Le calcul de la somme, élément par élément, s'effectue

EXEMPLE DE CALCUL UTILISANT UN TABLEAU DE CHAINES

```
10 ' FICHIER = ST1
15 DEFINT A-Z
20 DIM NM(20),X(20),Y(20),R(20),S(20)
30 PRINT "Saisie du premier operande"
40 GOSUB 1000
45 LPRINT " VALEUR PREMIER OPERANDE = ";A$
50 FOR I=1 TO 20
60 X(I)=NM(I)
70 NM(I)=0
80 NEXT I
90 PRINT "SAISIE DU SECOND OPERANDE"
100 GOSUB 1000
105 LPRINT " VALEUR SECOND OPERANDE = ";A$
110 FOR I=1 TO 20
120 Y(I)=NM(I)
130 NM(I)=0
140 NEXT I
150 GOSUB 2000
160 PRINT "RESULTAT:" : LPRINT : LPRINT " RESULTAT : "
170 FOR I=1 TO 20
175 PRINT S(I);        ' Le resultat sort a la fois
176 LPRINT S(I);       ' sur l'ecran et l'imprimante
177 NEXT I
180 END
1000 ' ** SOUS-PROGRAMME DE SAISIE **
1010 INPUT "Entrer la valeur";A$
1020 IF LEN(A$)>20 THEN PRINT "Erreur":GOTO 1010
1030 FOR I=1 TO 20
1040 B$=MID$(A$,I,1)
1050 V=VAL(B$)
1060 NM(I)=V
1070 NEXT I
1080 RETURN
2000 ' ** SOUS-PROGRAMME DE CALCUL **
2010 FOR I=20 TO 1 STEP-1
2020 V=X(I)+Y(I)              ' Calculer la somme des deux elements en position I
2030 IF I=20 GOTO 2050        ' Pas de retenue au premier passage
2040 V=V+R(I+1)               ' Ajouter la retenue de l'addition precedente
2050 V1=INT(V/10)
2060 R(I)=V1                  ' Memoriser la nouvelle retenue, qui sera ajoutee
2065 '                        ' au prochain passage
2070 V2=V-V1*10               ' Isoler les unites, qui constituent le chiffre
2075 '                        ' en position I du resultat
2080 S(I)=V2                  ' Memoriser ce chiffre dans le tableau resultat
2090 NEXT I
2095 RETURN
```

```
 VALEUR PREMIER OPERANDE = 01234567899876543210
 VALEUR SECOND  OPERANDE = 09876543210123456789

 RESULTAT :
 1  1  1  1  1  1  1  1  1  0  9  9  9  9  9  9  9  9  9  9

 VALEUR PREMIER OPERANDE = 01234567899876543210
 VALEUR SECOND  OPERANDE = 89764582136987412884

 RESULTAT :
 9  0  9  9  9  1  5  0  0  3  8  8  6  3  9  5  5  5  1  4
```

ADDITION DE DEUX ENTIERS EN « PRECISION ETENDUE »



Test 15

1 / Parmi les suivantes, certaines définitions de fonctions sont erronées : lesquelles et pourquoi ?

a) DEF FNA=CHR$(27)+CHR$(10)
b) DEF FNB=C*2+1
c) DEF FNN$=CHR$(7)+"Valeur"
d) DEF FNA$="Chaîne d'essai"+"="+256

2 / Ecrire un programme réalisant les fonctions suivantes :

- saisir une chaîne et une valeur numérique comprise entre 0 et 99 ;
- vérifier que la valeur numérique saisie représente un caractère alphabétique en ASCII ;
- dans l'affirmative, rechercher le rang de la première apparition de ce caractère dans la chaîne ;
- stopper l'exécution lorsque l'opérateur entre la chaîne "FIN".

3 / Ecrire un programme permettant de :

a) lire une suite de données entrées au clavier, chacune étant constituée par un groupe de nombres :

code = valeur numérique, comprise entre 10 et 20 (11 codes possibles),
coût unitaire = valeur numérique, comprise entre 1250 et 5700,
quantité = valeur numérique quelconque,
nom = chaîne alphanumérique de 20 caractères.

La saisie doit se terminer lorsque le Code lu est nul ;
b) vérifier que le Nom saisi figure parmi 10 noms préétablis dans un Data ;
c) calculer le coût total pour chaque donnée (coût total = Coût unitaire × Quantité) ;
d) effectuer la somme des Quantités saisies et des coûts totaux (calculés précédemment) pour chaque Code ;
e) calculer la somme des coûts totaux pour chaque Nom ;
f) afficher les sommes calculées lors des deux étapes précédentes. Lorsque le Code lu est nul, sortir de la boucle de saisie. Il est conseillé d'établir le programme en définissant les tableaux suivants :
CODE(11) pour les 11 valeurs des totaux par code,
NOM$(10) chaînes contenant les 10 noms prévus,
NMT(10) totaux par nom.

4 / Ecrire un sous-programme s'insérant dans le programme précédent, pour calculer le prix unitaire moyen, toutes données confondues. On se souviendra que la moyenne de N valeurs est donnée par leur somme divisée par N.

5 / La factorielle d'un nombre entier N est notée N! ; elle est obtenue en multipliant entre eux tous les nombres entiers compris entre 1 et N :
Factorielle = N! = $1 \times 2 \times 3 \ldots \times (N-1) \times N$
Par exemple :
$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 \quad 1! = 1$
Lorsque N est nul, on adopte la convention mathématique suivante : 0! = 1.
Ecrire un sous-programme calculant la factorielle d'un nombre entier NUM.

6 / Les « combinaisons de N éléments pris K à K » représentent toutes les manières possibles de choisir un groupe de K éléments parmi N, et sans répétition (un élément ne peut apparaître plus d'une fois dans le même groupe).
Par exemple, en disposant des lettres A, B et C (N = 3), et en extrayant les lettres deux à deux (K = 2), on peut obtenir les groupes : AB, AC et BC.
D'une façon générale, le nombre de choix possibles (nombre de combinaisons) est donné par l'expression :

$$\begin{pmatrix}\text{nombre de combinaisons} \\ \text{de N éléments pris K à K}\end{pmatrix} = \frac{N!}{K! \times (N-K)!}$$

On note cette valeur $\binom{N}{K}$ ou C_N^K

Dans l'exemple précédent (N = 3, K = 2), on a :

$$\text{nombre de combinaisons} = \frac{3!}{2! \times (3-2)!} = \frac{1 \times 2 \times 3}{1 \times 2 \times 1} = 3$$

Ecrire un programme calculant le nombre de combinaisons possibles de K éléments choisis parmi N.

Les solutions du test se trouvent pages 545, 546 et 547.

dans le sous-programme 2000, dont le listing est illustré page 534. La page 535 explicite la logique suivie : l'exemple détaille l'addition de deux nombres entiers (798899 et 4568), selon la méthode utilisée dans le sous-programme 2000. Remarquons que la première donnée (798899) « déborderait » largement du format entier de la machine (16 bits correspondent à l'intervalle [– 32768 ; + 32767]).

Applications à la statistique. Pour décrire des phénomènes de masse tels que la répartition de la population dans un pays déterminé ou les ventes moyennes annuelles par client dans une société, il est nécessaire de disposer d'un grand nombre d'observations individuelles*, supposées représentatives, qui permettront ensuite d'extrapoler les résultats partiels à l'ensemble étudié.

Il serait impossible d'obtenir manuellement une si grande quantité de données. Seul l'ordinateur peut satisfaire les exigences (précision et rapidité) du calcul statistique. C'est donc l'instrument indispensable à l'étude d'un phénomène : des programmes très généraux, acceptant en entrée un nombre quelconque d'observations de types très divers, exécuteront les calculs nécessaires pour obtenir les données statistiques décrivant le phénomène.

A titre d'exemple, sont développés ici certains programmes de calcul des grandeurs statistiques.

Supposons que nous voulions effectuer la synthèse d'un ensemble de données observées. On cherchera à déterminer une caractéristique globale de notre « échantillon », un paramètre qui, substitué aux différentes valeurs observées, n'en altère pas les caractéristiques statistiques. Cette grandeur, que nous appellerons une moyenne, permet des évaluations globales rapides, facilite les comparaisons entre échantillons, et sert de guide dans les situations incertaines.

On distingue des types différents de moyenne, selon la fonction caractéristique des données que l'on veut observer (moyenne

* Le résultat d'une observation individuelle est appelé **unité statistique** ; le résultat d'une opération réalisée sur des unités statistiques est appelé **donnée statistique.**

L'ordinateur est l'instrument indispensable de toute étude statistique. Ici, nous voyons (au fond de la salle et sur le chariot) les tambours de disques permettant le stockage des données.



ISTAT

géométrique, moyennes harmonique, médiane, etc.). La plus fréquente en calcul statistique est la **moyenne arithmétique,** que l'on calcule de la manière suivante :

$$\text{Moyenne arithmétique} = \frac{\text{somme des données}}{\text{nombre de données}}$$

Elle s'exprime toujours dans l'unité de mesure des données prises en compte par le calcul, et se représente mathématiquement par la variable $\overline{X}$.
Pour évaluer la moyenne des revenus annuels de six familles, soit :

$X_1 = 104\,500$ F/an
$X_2 = 109\,500$ F/an
$X_3 = 123\,000$ F/an
$X_4 = 151\,000$ F/an
$X_5 = 176\,500$ F/an
$X_6 = 186\,500$ F/an

on calculera d'abord le revenu annuel total, égal à la somme des six revenus :

$X_1 + X_2 + X_3 + X_4 + X_5 + X_6 = 851\,000.$

Le revenu moyen vaudra :

$\overline{X} = 851\,000 : 6 = 141\,833.33$ F/an.

Ce revenu constituerait celui de chaque famille, si le revenu total de 851 000 F était réparti de façon égale entre les six familles.
Le programme de la page 539 calcule la moyenne arithmétique d'un certain nombre de données (MAX) entrées au clavier. Il utilise un tableau dimensionné à 100 éléments : il pourra, au maximum, calculer la moyenne de 100 données.
Les différents types de moyenne sont obtenus en combinant les données observées de plusieurs façons. Il existe d'ailleurs une moyenne qui ne prend en compte que la position des données. Il s'agit de la **valeur médiane** (ou moyenne probable), qui n'est autre que la moyenne arithmétique des deux termes au centre de la série ordonnée des n valeurs (quand n est pair) ou le terme central (quand n est impair).
Un exemple simple démontrera qu'il est parfois préférable de calculer la médiane d'un ensemble d'observations plutôt que sa moyenne. On doit approvisionner en pain cinq magasins d'alimentation (A, B, C, D et E), chacun situé dans différents quartiers de la ville, mais tous près d'une route qui la traverse. On suppose donc que les magasins se trouvent sur la route, répartis aux kilométrages suivants :

magasin	A	B	C	D	E
distance en km	1	2	3	8	11

On cherche, pour construire un four, le point le plus près possible des cinq magasins. Le meilleur endroit sera celui qui, en additionnant les distances absolues entre le four et chaque magasin, donnera le résultat minimum, et ce lieu sera la valeur de la médiane. En effet, celle-ci correspond à l'élément central (3) et donne une somme de distances égale à 16 km ; en considérant la moyenne arithmétique (5), on obtient 18 km.
Introduisons maintenant une nouvelle grandeur statistique. Il peut s'avérer intéressant de connaître la fluctuation des données autour de la valeur moyenne (on veut, par exemple, savoir si les prix réels observés approchent plus ou moins le prix moyen). L'amplitude de cette variation indique si la moyenne est bien représentative de l'échantillon observé.
On calcule un nouveau paramètre statistique appelé variance, et noté mathématiquement par le symbole σ^2, de la manière suivante :

$$\sigma^2 = \frac{\text{somme des carrés des écarts}}{\text{nombre de données}}$$

où le terme « écart » représente l'écart à la moyenne (différence entre la valeur de chaque donnée et la moyenne). La variance peut aussi s'exprimer sous la forme :

$$\sigma^2 = \frac{\text{somme des carrés des données}}{\text{nombre de données}} - \text{carré de la moyenne}$$

Pour illustrer ces formules, considérons, par exemple, les prix de différentes qualités d'un certain produit :

Qualité	Prix (francs/kg)
A	$X_1 = 2.10$
B	$X_2 = 3.40$
C	$X_3 = 4.50$
D	$X_4 = 5.10$
E	$X_5 = 5.50$
F	$X_6 = 6.00$
G	$X_7 = 6.30$

PROGRAMME DE CALCUL DES MOYENNES

```
10 ' ** PROGRAMME DE CALCUL DES MOYENNES
20 '
30 '   FICHIER : STAT 1
35 B$="£££.£"          ' Format d'impression
40 '
45 DEFDBL A-H
46 DEFINT I-N
47 DEFDBL O-Z
50 DIM A(100)          ' Tableau memorisant les observations
60 '                    au maximum 100 valeurs
70 INPUT " Sur combien de donnees portera le calcul ? ";MAX
80 IF MAX=0 OR MAX>100 THEN PRINT " ** ERREUR ** ":GOTO 70
90 ' * Saisie des valeurs
100 FOR I=1 TO MAX
110 PRINT " Donnee num. :";I
120 INPUT " Valeur      :";A(I)
130 NEXT I
140 ' ** CALCUL DU TOTAL **
150 T=0                        ' Initialisation a zero de la somme
160 FOR I=1 TO MAX
170 T=T+A(I)                   ' Accumulation dans T de la somme des donnees
180 NEXT I
190 ' ** Les lignes 160 a 180 peuvent etre eliminees
200 '    en calculant simplement la somme pendant la saisie
210 '    c.a.d. en ecrivant en ligne 125 l'instruction T=T+A(I)
220 '
230 TMOY=T/MAX                 ' Le nom utilise pour la moyenne doit commencer
231 '                           par une des lettres reservees a la double
232 '                           precision
233 '
234 '
240 '
250 ' ** Impression
260 '
270 LPRINT "Donnees :"
280 ' Les donnees sont imprimees par lignes de 10
290 '
300 FOR I=1 TO MAX STEP 10
305 I1=I+9
310 FOR J=I TO I1
320 LPRINT USING B$;A(J);
330 NEXT J
340 LPRINT
350 NEXT I
360 '
365 LPRINT:LPRINT
370 LPRINT " Moyenne : ";TMOY
380 '
390 INPUT " Voulez-vous continuer (OUI/NON) :";REP$
400 IF REP$="OUI" GOTO 70
410 END
```

```
Donnees :
10.2  15.0  15.2  14.4  13.2  10.0  11.0  10.9  11.6  12.0
12.8  12.4  15.9  16.7  16.1  16.8  11.2  10.0  12.0  13.0
14.0  15.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

Moyenne : 13.1554454545455
```

La somme des données est de 32.90 F.
La moyenne des prix vaut

$$32.90 : 7 = 4.70$$

Pour évaluer la variance (avec la seconde formule), on calcule :

$$X_1^2 = \quad 4.41$$

La compréhension de la parole

D'une certaine manière, un ordinateur est analogue à un cerveau : tous deux assimilent des informations en provenance d'organes sensoriels externes, et tous deux les restituent, après traitement, à d'autres organes externes. Cependant, on constate une différence radicale : le cerveau possède une aptitude supérieure à l'ordinateur pour appréhender les informations, mais en revanche, ce dernier le dépasse en vitesse quand il s'agit de traiter des données numériques. Ceci s'explique probablement par le fait que la saisie des informations par le cerveau s'effectue en parallèle et d'une manière fortement redondante, alors qu'un ordinateur acquiert ses données séquentiellement.

La ***reconnaissance de la parole*** *est à la fois un problème de reconnaissance des formes et d'intelligence artificielle. Son intérêt est évident, la parole constituant le moyen le plus rapide et le plus naturel d'accéder à l'information. Elle jouera certainement un rôle de plus en plus important dans la communication entre l'homme et la machine. Mais le jour où les ordinateurs reconnaîtront et comprendront la parole humaine n'est pas encore arrivé, et cela en raison de multiples problèmes que l'on connaît aujourd'hui, et que l'on s'attache à résoudre.*

En effet, pour comprendre la parole, un homme ne considère pas seulement l'information perçue par l'oreille, mais aussi le ***contexte*** *dans lequel elle a été émise. C'est pour cela que nous comprenons le langage parlé, même lorsque le signal de parole est entaché de bruit. Cela dit, comprendre le contexte du discours suppose une vaste connaissance du monde environnant. Et c'est ici que la reconnaissance de la parole échoue aujourd'hui.*

Il est difficile de réaliser des programmes informatiques suffisamment sophistiqués pour comprendre la parole ***continue*** *(naturelle), d'un locuteur a priori inconnu de la machine. La reconnaissance de la parole par l'ordinateur est possible quand on simplifie le problème : en prononçant les mots séparés par des silences (c'est la reconnaissance de* ***mots isolés****), en limitant la taille du vocabulaire de référence, en restreignant le nombre de locuteurs qui devront être reconnus.*

On peut déjà acheter des jouets et des jeux vidéo qui obéissent à la voix pour moins de 1 000 F. Le prix des produits commercialisés de reconnaissance vocale varie entre 4 000 F et 400 000 F, suivant leurs performances et la taille du vocabulaire traité. La société japonaise Matsushita a présenté au « 1983 Japan Data Show » un module de reconnaissance vocale portable, avec clavier incorporé, affichage sur 20 caractères et haut-parleur intégré, qui reconnaît 62 mots isolés. La société Crouzet utilise un module de reconnaissance de mots enchaînés, en collaboration avec le LIMSI (Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur, CNRS). Ces produits imposent des contraintes à l'utilisateur : le vocabulaire est limité (de 10 à 200 mots), une phase d'apprentissage est nécessaire pour chaque nouveau locuteur, et les mots doivent être prononcés distinctement.

Quels sont les problèmes rencontrés en reconnaissance automatique de la parole ?

Le signal de parole peut être segmenté en une suite de sons élémentaires appelés ***phonèmes****, qui présentent des caractéristiques articulatoires et acoustiques particulières. Bien que l'appareil vocal humain puisse produire un nombre presque infini de positions articulatoires, le nombre de phonèmes reste limité : 36 en français, 22 en espagnol, 44 en anglais. Chaque phonème possède des caractéristiques acoustiques distinctes. Combiné avec d'autres phonèmes, il forme des unités phonétiques plus grandes, telles que les diphonèmes, les demi-syllabes, les syllabes. Une connaissance approfondie des différences acoustiques entre ces unités sonores est nécessaire à la distinction de mots tels que : « cabot », « canot », « cadeau », « cachot »...*

Quand les sons élémentaires sont assemblés pour former des unités phonétiques plus grandes, les caractéristiques acoustiques d'un phonème donné seront modifiées par le phonème qui le précède et celui qui le suit ; cela s'explique par l'interaction physique entre les différentes structures anatomiques (la langue, les lèvres, les cordes vocales, le nez, etc.). Le même phonème de base, « t » par exemple, aura des caractéristiques totale-

ment différentes dans des mots (ou contextes) différents tels que «thé», «trait», «temps», «battu», «astre»... Cet effet de **coarticulation** nous rend particulièrement délicate la reconnaissance d'un phonème à l'intérieur d'un mot.

Une autre difficulté est la **continuité** du langage parlé. En effet, les mots sont séparés dans un texte écrit, alors qu'ils ne le sont pas dans un texte parlé. Au départ, pour l'ordinateur, la parole est uniquement une suite continue de sons. Ainsi, si la phrase prononcée est «J'ai mal au pied», et si l'ordinateur se limite à des connaissances phonétiques, il ne saura pas faire la différence entre «J'aima l'haut pied» et «géai mâle au pied». Dans le cas de cette phrase précise, plus de mille combinaisons sont possibles. L'ordinateur a besoin d'autres connaissances.

On va alors le doter de **connaissances syntaxiques** (règles de grammaire), qui lui permettront d'éliminer les phrases de structure syntaxique incorrecte. Le nombre de combinaisons possibles au niveau syntaxique sera plus faible, mais conservera des phrases qui n'auront aucun sens, comme: «j'aima l'haut pied» (sujet - verbe - article - adjectif - complément).

Pour éliminer les combinaisons qui n'ont pas de sens, on fait appel à des **connaissances sémantiques** qui sont très difficiles à formaliser en langage informatique. Ainsi, après s'être référé successivement aux connaissances phonétiques, lexicales (le dictionnaire de la langue), syntaxiques et sémantiques, deux combinaisons restent candidates: «J'ai mal au pied» et «J'ai mal aux pieds». Et là, seules des considérations sur le contexte pourront nous indiquer la phrase à retenir. Si, dans la phrase précédente, il s'agissait du pied droit, on conservera «J'ai mal au pied». Dans ce cas, on a fait appel à des connaissances au niveau **pragmatique** (lié au contexte).

Un système de reconnaissance de la parole continue doit rassembler une grande quantité de connaissances dans des domaines extrêmement variés: acoustiques (sur le signal de parole), phonétiques (sur la langue concernée), lexicales (dictionnaire de la langue), syntaxiques (structure des phrases), sémantiques (sens des mots), pragmatiques (contexte des mots). Les futurs ordinateurs devront donc accéder à beaucoup de connaissances.

De plus, il faut pouvoir raisonner sur ces connaissances, et de manière astucieuse et non exhaustive: c'est le rôle des stratégies de raisonnement. Celles-ci prennent une place importante dans les systèmes de reconnaissance de la parole continue actuels. En effet, quelles connaissances utiliser, et dans quel ordre?

On distingue, par exemple, les **stratégies ascendantes** (utilisant les connaissances dans l'ordre suivant: acoustique - phonétique - lexical - syntaxique - sémantique - pragmatique), comme dans l'exemple précédent, et des **stratégies descendantes**, dans lesquelles on part d'hypothèses issues du contexte pour redescendre l'échelle des niveaux de compréhension: pragmatique - sémantique - syntaxique - lexical - phonétique - acoustique. Ainsi, dans l'approche descendante, on va émettre des hypothèses, faires de prédictions sur ce qui va être dit, puis les vérifier.

Faut-il utiliser une stratégie ascendante ou descendante? Il semble que la première soit plus performante quand on n'a pas commis trop d'erreurs de reconnaissance phonétique, et que l'univers d'application (lexique, syntaxe, sémantique) est large. Par contre, quand ce dernier est restreint (c'est-à-dire que le contexte est bien connu) et qu'il y a trop d'erreurs de reconnaissance phonétique, la stratégie descendante s'avère plus performante. En fait, les systèmes actuels comportent des stratégies mixtes.

On voit donc apparaître une grande variété de stratégies de reconnaissance, quelquefois très complexes, exploitant des connaissances à des niveaux très différents, et les parcourant de multiples façons selon la phrase à reconnaître.

Où en sommes-nous aujourd'hui?

Aux Etats-Unis, la recherche en reconnaissance de la parole continue fut largement encouragée et coordonnée par le projet ARPA (1971-1976), financé par le Département de la Défense américain. Depuis, elle s'est assoupie, exception faite de l'effort considérable d'IBM pour réaliser une machine à écrire à entrée vocale.

En France, des systèmes de reconnaissance de la parole continue existent au LIMSI (sys-

Phrase orthographique :	J'AI MAL AU PIED
Phrase phonétique :	JÉ MAL O PYÉ
Phrases possibles au niveau lexical :	J'AIMA L'HAUT PIED GEAI MÂLE AU PIED JET MALLE EAU PIEDS ... (PLUS DE 1000 COMBINAISONS POSSIBLES)
Phrases possibles au niveau syntaxique :	J'AIMA L'HAUT PIED J'AI MAL AU PIED GEAI MÂLE AU PIED J'AI MAL AUX PIEDS
Phrases possibles au niveau sémantique :	J'AI MAL AU PIED J'AI MAL AUX PIEDS
Phrase possible au niveau pragmatique :	J'AI MAL AU PIED

Exemple de stratégie ascendante en reconnaissance de la parole continue (d'après J. MARIANI).

tème ESOPE), au CNET (système KEAL), au CRIN (MYRTILLE), au CERFIA (ARIAL), à l'ENSERG, à Marseille-Lumigny et au LEA de Nancy.

Il est intéressant de noter que la machine à écrire à entrée vocale fait partie des applications pilotes envisagées par les Japonais dans leur projet d'ordinateur de la cinquième génération. Mais les Etats-Unis et les pays européens ne doivent pas être effrayés, car la langue japonaise est beaucoup plus simple à reconnaître que les autres. On peut d'ailleurs s'étonner de la modestie de leurs ambitions, étant donné les progrès technologiques considérables que suppose l'aboutissement de leur projet.

Ce qui reste intéressant dans ce projet, c'est l'accent mis sur la réalisation de machines spécialisées dans le traitement des connaissances. L'informatique classique s'appuie sur une structure de contrôle, des ***programmes*** *et des* ***données****. L'évolution récente vers l'intelligence artificielle se fonde au contraire sur la qualité entre les* ***inférences*** *(le raisonnement) et les* ***connaissances****.*

C'est ce que l'on constate déjà dans les techniques de ***« systèmes experts »*** *qui commencent à voir le jour au niveau commercial. Ce type de système est capable de raisonner à partir de connaissances de haut niveau fournies par des experts humains. Déjà, des systèmes experts en diagnostic médical dépassent les performances des spécialistes, puisqu'ils réalisent la synthèse des connaissances acquises dans ce domaine particulier. Leur rôle ne consiste pas à supplanter l'homme, mais plutôt à l'aider dans ses décisions, à lui apprendre et lui expliquer ce que d'autres hommes savent. C'est ainsi que déjà plusieurs sociétés se proposent de construire et de commercialiser des systèmes experts. Mais, à l'heure actuelle, ils ne s'avèrent capables de raisonner que dans des domaines d'expertise très restreints. On entendra parler d'un* ***système expert en*** *prospection minière (PROSPECTOR),* ***en*** *diagnostic d'affections bactériennes et indications thérapeutiques (MYCIN), etc., et non d'un* ***système expert.***

Cette approche a beaucoup séduit les cher-

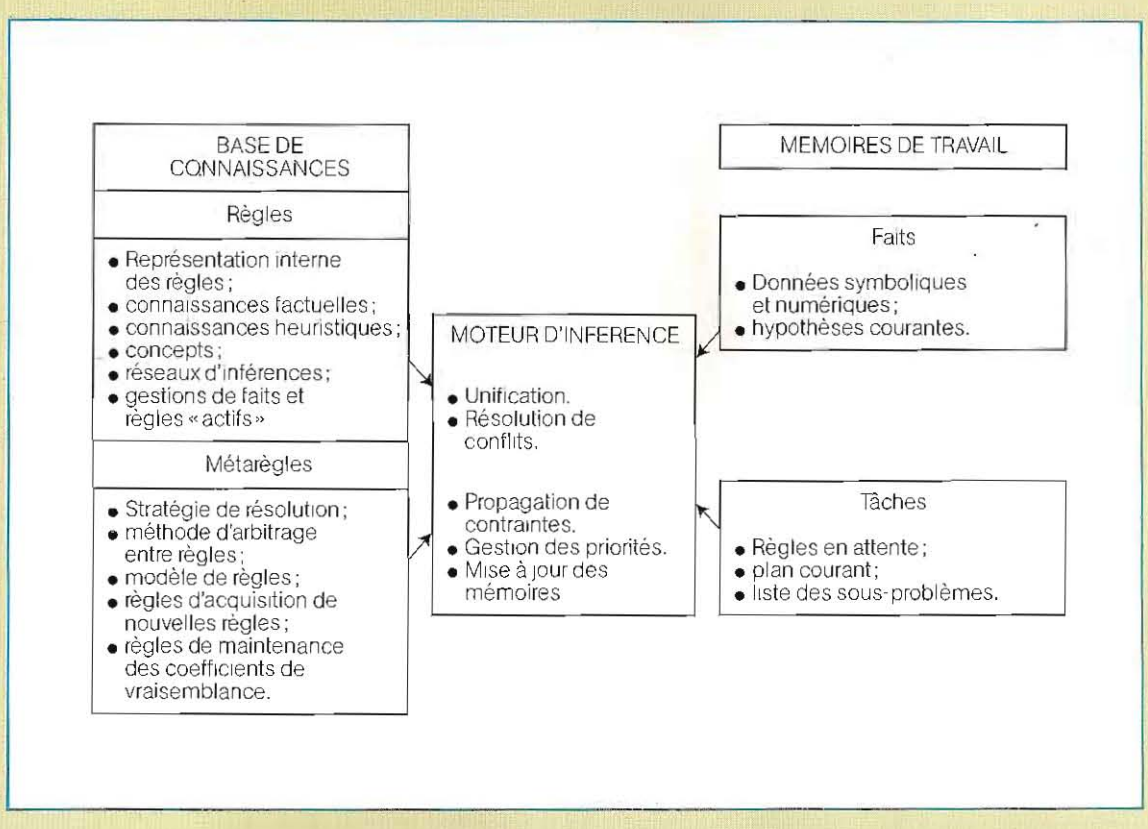


Schéma général d'un système expert.

cheurs en reconnaissance de la parole continue. Les ordinateurs de la cinquième génération y seront d'ailleurs parfaitement adaptés. C'est ainsi qu'aux Etats-Unis, A. Newell et D. McCraken ont récrit les systèmes HEARSAY et HARPY, réalisés à la Carnegie-Mellon University au cours du projet ARPA. De même, les laboratoires de la CGE à Marcoussis ont réalisé une version similaire d'HEARSAY, et le CNET, à Lannion, a récrit son système KEAL sous forme de système expert.

Une orientation récente de quelques laboratoires consiste à réaliser des systèmes experts capables de reconnaître des ***sonagrammes*** *(images d'une émission vocale dans les coordonnées temps - fréquence - amplitude). En effet, un chercheur américain du MIT (V. Zue) arrive à lire des sonagrammes provenant de plusieurs locuteurs, avec moins de 10 % d'erreurs. Ainsi, la réalisation d'un système expert en lecture de sonagrammes devrait permettre d'obtenir des connaissances phonétiques plus fondamentales et mieux exploitables.*

Cette idée est en cours de réalisation en France (CRIN, LIMSI), en Angleterre (Université de Cambridge), et aux Etats-Unis (MIT, VERBEX). Il est clair qu'un tel système serait susceptible d'intéresser au plus haut point les laboratoires qui n'ont pas d'expert en sonagrammes à leur disposition.

Enfin, le groupe d'intelligence artificielle de l'université de Lumigny-Aix-Marseille poursuit des études en traitement automatique de la parole.

Ainsi, il est clair que l'intelligence artificielle, sous ses divers aspects (résolution de problèmes, reconnaissance de la parole continue, compréhension du langage naturel, etc.) permettra à l'homme d'utiliser avec souplesse des systèmes informatiques de plus en plus performants, destinés à jouer un rôle croissant dans sa vie.

• « Les Ordinateurs de la cinquième génération », LA RECHERCHE, n° 154, avril 1984.
• « ESOPE : un système de compréhension de la parole continue », thèse d'Etat, Joseph MARIANI, Paris VI, 1982.
• « Artificial Intelligence », IEEE-SPECTRUM, novembre 1983.

$X_2^2 = 11.56$
$X_3^2 = 20.25$
$X_4^2 = 26.01$
$X_5^2 = 30.25$
$X_6^2 = 36$
$X_7^2 = 39.69$

La somme vaut 132.53 et donc :

$$\frac{\text{somme des carrés des données}}{\text{nombre de données}} - (\text{moyenne})^2 =$$

$= 132.53 : 7 = 22.09 = 18.932857 - 22.09 =$
$= 3.157143$

Comme indice de variation, on considère la racine carrée de la variance. Cette valeur, désignée par σ, est appelée **écart type quadratique moyen.** Dans notre cas, l'écart type étant de 3.157143 F/kg, sur une moyenne de 4.70 F/kg, nous pouvons affirmer que la variation du prix n'est pas négligeable (un peu plus de 15 %). Le programme listé ci-dessous calcule la moyenne et l'écart type d'un certain nombre de valeurs (MAX) saisies au clavier. Le programme mémorise les données dans le tableau A(100) ; (MAX ne peut être supérieur à 100).

PROGRAMME DE CALCUL DE L'ECART TYPE

```
1 REM * programme de calcul de la moyenne et de l'ecart type *
2 REM execute sur HHC en Microsoft BASIC
3 REM
10 DIM A(100)
20 INPUT "nombre d'observations";MAX
30 IF MAX=0 OR MAX>100 THEN PRINT "erreur":GOTO 20
40 T=0
50 FOR I=1 TO MAX
60 PRINT "observation no. ";I
70 INPUT "valeur ";A(I)
80 T=T+A(I):REM calcul de la somme
90 NEXT I
100 TMOY=T/MAX
110 PRINT #2"moyenne = ";TMOY
120 REM calcul de la somme des carres
130 T=0:REM T est reutilise avec une autre signification
140 FOR I=1 TO MAX
150 T=T+A(I)*A(I):REM T contient la somme des donnees au carre
160 NEXT I
170 PRINT #2"somme des carrés = ";T
180 S=SQR (T/MAX-(TMOY*TMOY)):REM S=ecart type
190 PRINT #2"ecart type      = ";S
200 REM
210 REM calcul du nombre de donnees dans l'intervalle
220 REM d'amplitude S et centre sur TMOY
230 INF=TMOY-S
240 SUP=TMOY+S
245 K=0
250 FOR I=1 TO MAX
260 IF A(I)>=INF AND A(I)<=SUP THEN K=K+1
270 NEXT I
280 REM
290 PRINT #2"on compte ";K;" valeurs dans"
300 PRINT #2"l'intervalle (TMOY-S;TMOY+S)"
310 NP=K*100/MAX
320 PRINT #2"soit un pourcentage de ";NP ;" %"
330 REM
340 INPUT "voulez-vous continuer (oui/non) ";REP$
350 IF REP$="oui" GOTO 20
360 END
```

```
Les donnees saisies sont celles de l'exemple precedent

moyenne = 4.7
somme des carres = 132.53
ecart type      = 3.157143
on compte 5 valeurs dans
l'intervalle (TMOY-S;TMOY+S)
```

Solutions du test 15

1 / a) La fonction doit être définie comme une chaîne, car elle utilise CHR$, qui renvoie une chaîne comme résultat. Il faut donc écrire :
DEF FNA$=CHR$(27)+CHR$(10) ;
d) Cette fonction est erronée, car elle contient une valeur numérique, alors qu'elle est définie en tant que chaîne ; la forme correcte est donc la suivante :
DEF FNA$="Chaîne d'essai"+"="+STR$(256).

---

2 / Nous proposons le programme suivant :

```
10  ' ** DEBUT
20  INPUT "CHAINE, NOMBRE ";A$,N
30  IF N<0 OR N>99 GOTO 20          ' CONTROLE
40  IF A$="FIN" GOTO 190
50  ' ** CONTROLER SI N PEUT ETRE UNE LETTRE EN ASCII
60  ' LETTRES MAJUSCULES SEULEMENT
70  K=0
80  IF N>65 AND N<90 THEN K=1
90  IF K=0 GOTO 20
100 '
110 '
120 B$=CHR$(N)
130 '
140 M=INSTR(A$, B$)
150 IF M=0 GOTO 20
160 PRINT "LE NOMBRE ";N;"REPRESENTANT LE CARACTERE ";B$
170 PRINT "SE TROUVE EN POSITION ";M;"DE LA CHAINE ";A$
180 GOTO 20
190 END
```

---

3 / Le programme doit utiliser des tableaux et des DATA. Voici une des formes possibles.

```
10  ' ** DEBUT
20  DIM CODES(11) ,NMT(10) ,NOM$(10)
30  FOR I=1 TO 10:READ NOM$(I) :NEXT I
40  DATA "PREMIER NOM"
50  '
60  '                    NOMS PREVUS
70  '
130 DATA "DIXIEME NOM"
140 ' LECTURE DONNEES
150 INPUT "CODE ";CD
155 IF CD=0 GOTO 310     ' IMPRESSION ET FIN DE PROGRAMME
160 IF CD<10 OR CD>20 GOTO 150
170 INPUT "MONTANT ";MT
180 IF MT<1250 OR MT>5700 GOTO 170
190 INPUT "QUANTITE ";QT
200 INPUT "NOM ";NM$
210 K=0
220 FOR I=1 TO 10
230 IF NM$=NOM$(I) THEN K=1
240 NEXT I
250 IF K=0 GOTO 200      ' K=0 SIGNIFIE QUE LE NOM EST INCORRECT
260 TOT=QT * MT          ' MONTANT TOTAL
270 L=CD-9
```

```
271  ' LE CODE (CD) A UNE VALEUR COMPRISE ENTRE 10 ET 20. LA DIFFERENCE
272  ' (CD-9) VARIE ENTRE 1 ET 11 (10-9=1 , 20-9=11) ET INDIQUE DONC
273    A QUELLE POSITION DU TABLEAU (CODES) CORRESPOND UNE VALEUR
274  ' DONNEE DE (CD) , PAR EXEMPLE, SI CD=12, L'INDICE (L) VAUT
275  ' 12-9=3 : LE MONTANT ASSOCIE AU CODE (CD) DOIT ETRE
276  ' MEMORISE EN POSITION 3 DU TABLEAU (CODES)
280  CODES (L) = CODES (L)+TOT      ' ACCUMULE LE TOTAL PAR CODES
290  NMT (K)=NMT (K)+TOT            ' ACCUMULE LE TOTAL PAR NOMS
291  ' CAR (K) POINTE SUR LE NOM (INSTRUCTIONS 210-240)
300  GOTO 150          ' SAISIE D'UN NOUVEAU GROUPE DE DONNEES
310  ' ** IMPRESSION **
320  FOR I=1 TO 11          ' SELECTIONNE UN DES 11 TOTAUX
330  N =I+9                 ' RESTITUE LE CODE
340  PRINT "CODE: ";N;TOTAL: ";CODES(I)
350  NEXT I
360  '
370  ' ** IMPRESSION PAR NOM **
380  FOR I=1 TO 10          ' SELECTIONNE UN DES 10 NOMS
390  PRINT "NOM: ";NOMS$(I) ; "TOTAL: ",NMT(I)
400  NEXT I
410  END
```

---

4 / On obtient le coût unitaire moyen en additionnant tous les montants lus à la ligne 170, et en divisant ce total par le nombre de saisies effectuées. Le sous-programme doit prévoir un compteur qui s'incrémente à chaque saisie et un totaliseur qui cumulera tous les montants. A la fin du programme, leur quotient donne la valeur moyenne. Ce sous-programme sera appelé en insérant dans le programme précédent la ligne :

```
185  GOSUB 1000
```

et se résume à deux instructions :

```
1000  '*ENTREE
1010  COMPT= COMPT+1 'Compteur
1020  SOMME= SOMME+CT 'Totaliseur
```

A la fin du programme, on calcule la valeur moyenne recherchée et l'on affiche :

```
405  PRINT "Coût unitaire moyen = ";SOMME/COMPT
```

---

5 / L'algorithme consiste à exécuter une boucle de multiplication, de 1 à NUM, avec :
NUM = Valeur transmise au sous-programme, dont on veut calculer la factorielle.

```
1000  'Calcul de la factorielle du nombre NUM
1010  T=1 'Première valeur
1015  IF NUM=0 GOTO 1050 'Calcul de factorielle 0
1020  FOR I=2 TO NUM
1030  T=T*I
1040  NEXT I
1050  RETURN
```

En sortie, la variable T contient la factorielle de NUM.

---

6 / On utilisera ici le sous-programme précédent pour calculer les trois factorielles [N!, K! et (N-K)!] qui apparaissent dans la formule.

```
10  '** DEBUT
15  DEFDBL R-T
```

```
20  INPUT "Nombre d'éléments";N
30  INPUT "Nombre d'éléments par groupe";K
40  NUM=N
50  GOSUB 1000
60  S=T       'Sauvegarde dans S le résultat du sous-programme (factorielle de N)
70  NUM=K
80  GOSUB 1000
90  S1=T      'S1 contient la factorielle de K
100 NUM=N-K
110 GOSUB 1000
120 S2=T      'S2=(N-K)!
130 R=S/S1/S;      'R est le nombre de combinaisons cherché
140 PRINT "Il y a ";R;"manières de choisir"
145 PRINT K;"éléments parmi ";N
150 END
```

On remarquera que la ligne 130, qui exécute le calcul final, aurait pu s'écrire R=S(S1*S2). Cependant, à cause des ordres de grandeur des nombres S1 et S2, leur produit pourrait conduire à une valeur dépassant la précision de la machine. En effectuant d'abord la division S/S1, on réduit les risques d'erreur. De plus, même en déclarant les variables en double longueur (ligne 15), le nombre N que l'on peut traiter est limité à 30 (pour des valeurs supérieures, on obtient des erreurs de précision).

Les fonctions d'entrées/sorties des données

Dans ce chapitre sont exposées les commandes et les fonctions de saisie des données au clavier, d'affichage à l'écran ou d'édition sur imprimante. Les commandes et les fonctions d'E/S s'adressant aux unités disque ou aux bandes magnétiques seront traitées à part.

Dans tous les langages de haut niveau, les données à saisir ou à éditer doivent d'abord être traitées par des programmes système. En entrée, ceux-ci transposent la donnée générée par le clavier dans le format requis par la machine ; en sortie, ils adaptent le format des caractères aux particularités des périphériques. Les modes de conversion dépendent du type de donnée saisie : pour traiter un nombre entier, par exemple, les programmes système le convertissent en binaire et le stockent dans deux octets contigus en mémoire. Dans le cas de nombres réels, le processus est plus complexe. Dans la machine, les réels sont représentés sous forme exponentielle (floating point) et les programmes d'E/S assurent la distinction entre la mantisse et l'exposant. La position automatique du point décimal et le nombre de chiffres significatifs dépendent du type de machine et du système d'exploitation.

Le programme de saisie des données, en plus de sa fonction de « formatage », exécute tous les contrôles nécessaires afin d'assurer l'exactitude formelle des saisies : il vérifiera la cohérence entre la donnée fournie et le type demandé, et, dans le cas d'une saisie de plusieurs valeurs, entre le nombre de données saisies et le nombre attendu.

Les données à imprimer (ou à afficher à l'écran) sont extraites de la mémoire, modifiées éventuellement par le programme d'application, puis envoyées au périphérique. Cette adaptation formelle des données pour respecter une présentation particulière lors de l'édition s'appelle le **formatage** (du terme anglais format). Les formats d'acquisition ou d'impression des données sont définis par le programmeur, avec les instructions et les fonctions appropriées. En Basic, on peut également gérer les contrôles à effectuer, grâce à l'instruction ON ERROR GOTO ... étudiée précédemment.

Fonctions de saisie des données

Chaque phase de saisie s'achève en frappant la touche de validation CR (ou RETURN). Si les données saisies sont correctes, le système les transmet au programme d'application ; dans le cas contraire, un message approprié s'affiche à l'écran.

En Basic, on peut contrôler dans le programme certaines erreurs commises en phase de saisie ; dans tous les autres langages, celles-ci entraînent systématiquement l'arrêt de l'exécution.

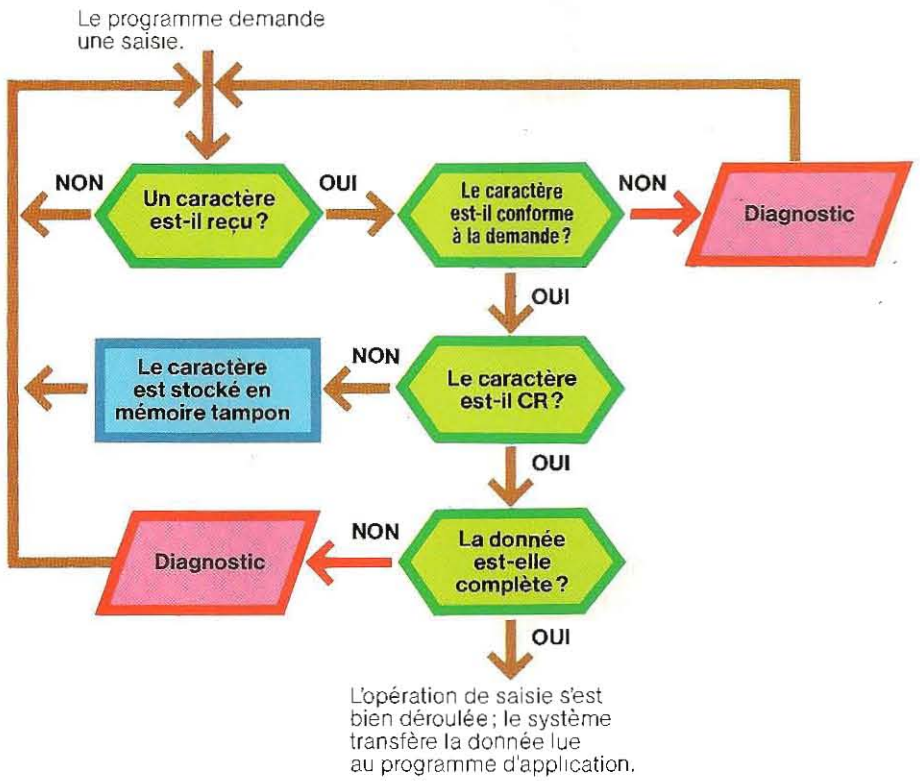
Le schéma ci-dessous illustre les fonctions exécutées en phase d'acquisition des données. Lorsqu'une instruction de lecture s'exécute, le contrôle est passé au programme système correspondant, qui se met en attente d'un caractère. Le temps machine nécessaire à la lecture d'un octet sur le port d'entrée est très court, comparativement au temps que met l'opérateur pour taper un caractère. Le système entre donc dans une boucle d'attente, comme le montre notre schéma. Une fois le caractère reçu, un premier contrôle formel est effectué. Par exemple, lorsqu'on demande une valeur numérique, le système vérifie qu'il ne s'agit pas d'une lettre. Le caractère est alors stocké dans un buffer (mémoire tampon), et le processeur reprend la boucle d'attente. Les caractères émis par l'opérateur sont rangés l'un derrière l'autre, de façon à former la donnée complète, jusqu'à ce que la saisie soit validée par la touche CR (ou RETURN). Le programme système vérifie alors que toutes les données demandées par l'instruction de lecture sont présentes ; dans l'affirmative, l'exécution du programme d'application reprend, sinon le système se met en attente des autres données nécessaires à l'achèvement de la requête.

INPUT. La principale instruction de lecture est la suivante :

INPUT "Commentaires"; VARIABLES

La chaîne "Commentaires", qui est facultative, représente un message que l'on désire affi-

SCHEMA DU PRINCIPE DE SAISIE DES DONNEES



cher au moment de la saisie. Le système écrit à l'écran la chaîne figurant entre guillemets, suivie d'un point d'interrogation, indiquant qu'il attend une saisie. L'opérateur peut alors entrer ses données, qui sont successivement affectées aux variables spécifiées dans l'instruction. Par exemple, l'exécution de :

```
INPUT "Essai"; V1,K$,B
```

provoquera l'affichage de la ligne :

```
Essai ?
```

L'opérateur devra entrer trois données (affectées respectivement à V1, K$ et B) : la première et la troisième seront des valeurs numériques, alors que la deuxième représentera une chaîne. Elles seront séparées par des virgules et, à la fin de la saisie, l'opérateur devra frapper la touche CR.
La saisie 3.4,ABC,100 satisfait l'instruction précédente. En revanche, si l'on intervertit l'ordre des données (par exemple 3.4,100,ABC), la cohérence avec le format demandé (nombre, chaîne, nombre) n'est plus respectée, et le programme système signale une erreur.
En phase de saisie, on peut supprimer le point d'interrogation émis par le système, en remplaçant le symbole ; par , . L'instruction

```
INPUT "Essai", V1,K$,B
```

produit le même effet que la précédente mais avec la suppression du symbole ? après le message. Normalement, le caractère CR (Carriage Return) signale à la machine la fin de la saisie et, en même temps, positionne le curseur au début de la ligne suivante. La dernière option acceptée par l'instruction INPUT permet d'empêcher le déplacement du curseur afin de disposer plusieurs entrées sur une même ligne. Pour cela, on insère le symbole ; tout de suite après le code :

```
INPUT ; "Description";V1,K$,B
```

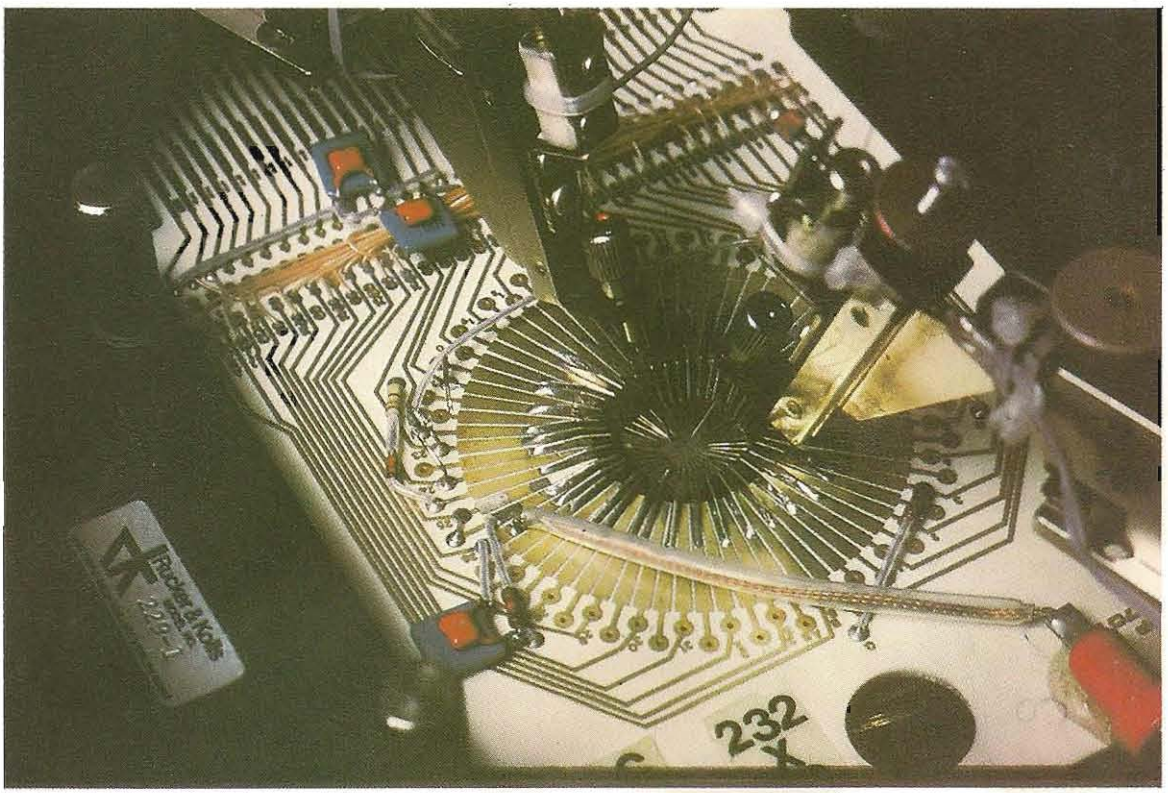
Cette formulation supprime le retour du curseur en début de ligne à la fin de la saisie.
L'instruction INPUT, sous l'une quelconque de ses formes, implique trois limitations :

1 / les caractères saisis sont écrits automatiquement sur l'écran (écho) ;

2 / la longueur de chaque donnée (nombre de chiffres pour les variables numériques ou de caractères pour les chaînes) n'est pas contrôlée par le système (sauf en cas de débordement) ;

3 / les caractères spéciaux (CR, RUBOUT, ESCAPE, etc.) ne peuvent pas être saisis.

Pendant la saisie des données, chaque caractère est acquis directement par le programme, tandis que son « écho » se répercute automatiquement à l'écran. Dans certains cas, par exemple pour saisir un mot de passe de fichier protégé, on préférera un autre type d'instruction de lecture, qui ne provoquera pas d'écho à l'écran.
Le deuxième point (contrôle de la longueur des données) s'avère crucial pour limiter les erreurs de saisie. C'est pourquoi la gestion par **fenêtres d'édition** est souvent utilisée. Avec ce mode de saisie, l'opérateur dispose d'un espace bien défini, dans lequel il peut taper ses valeurs ; ces dernières ne peuvent en aucun cas dépasser la longueur prévue. La simple instruction INPUT n'autorise aucune vérification de ce type. Au contraire, en contrôlant la longueur de la donnée au fur et à mesure de sa saisie, on peut interrompre la lecture au moment voulu.
Le dernier point mentionné est également lié à des utilisations particulières. En phase de lecture, l'instruction INPUT vérifie que les valeurs saisies et leurs variables associées sont cohérentes ; elle élimine d'éventuels caractères spéciaux, qui pourraient cependant s'avérer très utiles dans certaines applications, pour communiquer à l'ordinateur des ordres à la place de données. Prenons l'exemple simple d'un programme destiné à lire, puis à stocker dans un fichier, des renseignements relatifs au personnel d'une entreprise (nom, service, salaire, promotion, absences, primes). Or, nous sommes en présence d'un nouvel employé, et les trois dernières rubriques sont encore inconnues. Si aucun code ne peut imposer la fin de la saisie, l'utilisateur sera toujours contraint de compléter tous les champs demandés, et perdra peut-être un temps considérable.
Si, au contraire, on prévoit la possibilité d'interrompre la phase de saisie, on ne fournira à chaque fois que les données voulues (dans notre exemple, les trois premières). L'instruc-

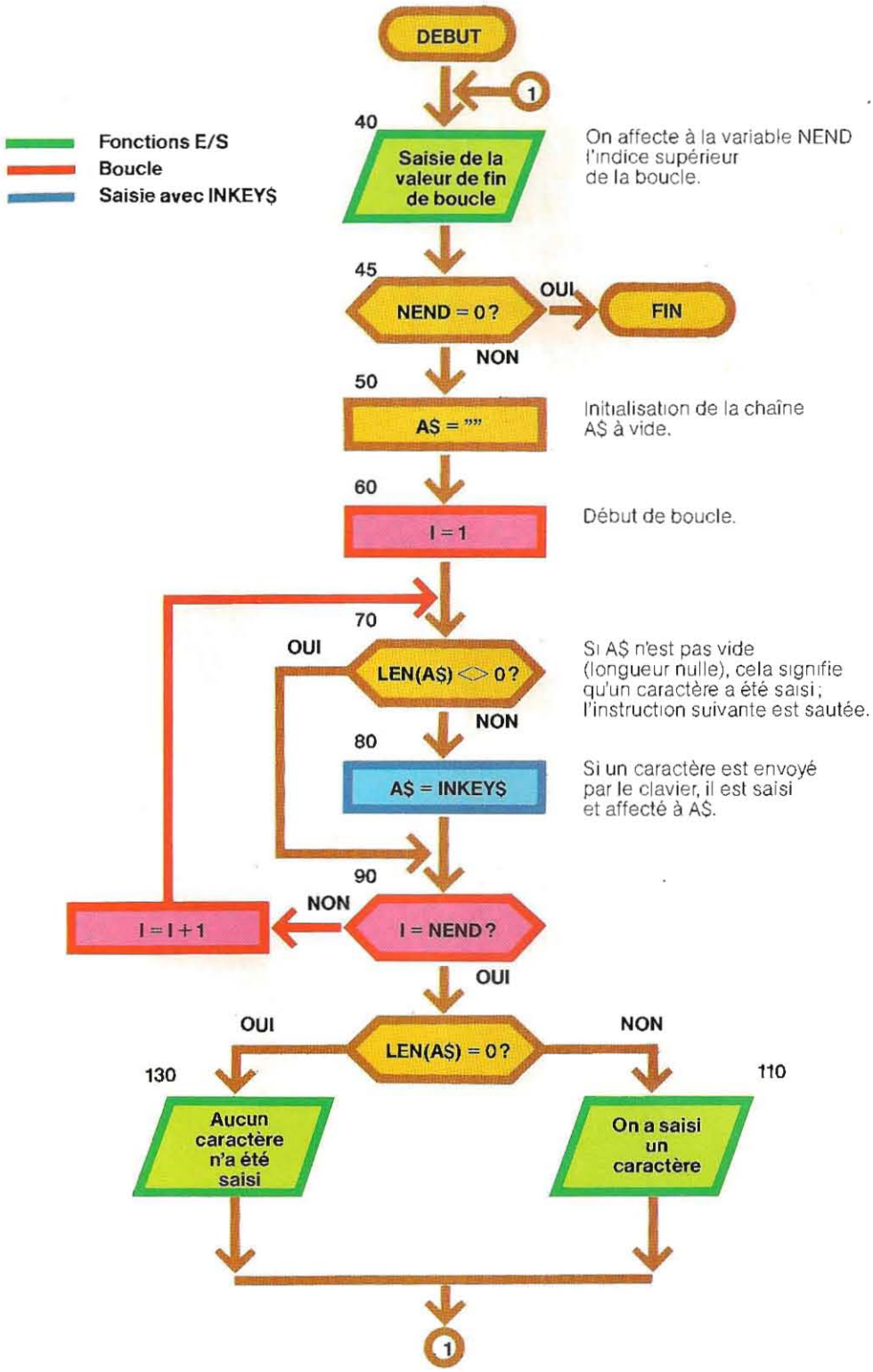


K. Reese/Marka

Contrôle d'un micro-circuit par ordinateur. Les barettes de cuivre rayonnantes établissent les connexions entre le chip (au centre) et l'extérieur.

tion INPUT précédente requiert dans tous les cas six valeurs (deux chaînes, un nombre, une chaîne, deux nombres) et on ne peut omettre aucune donnée.
L'opérateur doit entrer les différentes valeurs conformément à ce qui a été spécifié dans l'instruction de lecture.
Ces difficultés peuvent, dans une certaine mesure, être surmontées avec les instructions INKEY$ et INPUT$(N).
Cependant, de nouvelles complications apparaissent : tous les contrôles (vérifiant par exemple que la donnée est bien du type voulu), qui étaient auparavant effectués par les programmes système avec l'instruction INPUT, doivent dorénavant être assumés par le programme d'application.
Le choix d'une instruction de lecture particulière dépend de l'application envisagée. Dans un programme simple, qui saisit peu de données, l'instruction INPUT suffit. Au contraire, pour un programme d'une certaine complexité, nécessitant une bonne gestion de l'écran et des contrôles spécifiques, l'utilisation des autres formes devient indispensable.

EXEMPLE DE FONCTIONNEMENT DE L'INSTRUCTION INKEY$



EXEMPLE D'UTILISATION DE LA FONCTION INKEY$

```
10 '
20 ' *** EXEMPLE D'UTILISATION DE LA FONCTION INKEY$
30 '       FICHIER : SUNCAR
40 INPUT "Nombre de fois qu'il faut executer la boucle ";NEND
45 IF NEND=0 GOTO 150     ' Sortie
50 A$=""
60 FOR I=1 TO NEND
70 IF LEN(A$)<>0 GOTO 90
80 A$=INKEY$
90 NEXT I
100 IF LEN(A$)=0 GOTO 130
110 PRINT     "Saisie d'un caractere pendant la boucle"
120 GOTO 40
130 PRINT     "ON N'A SAISI AUCUN CARACTERE"
140 GOTO 40
150 END
```

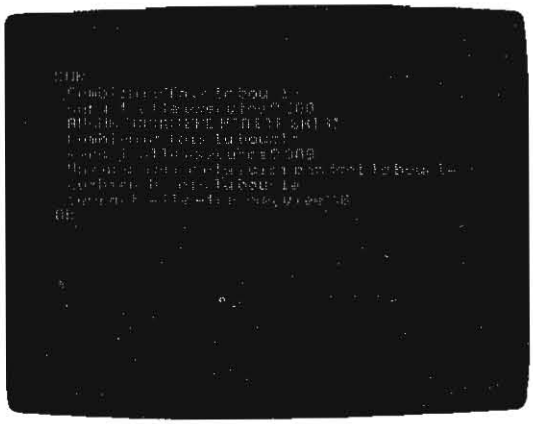
**INKEY$**. Permet la saisie d'un unique caractère depuis le clavier ; le caractère émis n'a pas d'écho et n'apparaît donc pas à l'écran. L'instruction INKEY$ accepte n'importe quel caractère (y compris les touches de contrôle), excepté CTRL + C (touche CONTROL et touche C enfoncées simultanément) qui arrête l'exécution du programme. Notons que certains Basics compilés acceptent même le caractère CTRL + C, qui est rendu temporairement transparent par l'instruction INKEY$. La syntaxe de l'instruction est :

A$ = INKEY$

Le caractère lu depuis le clavier est affecté à la chaîne A$, définie par l'utilisateur. Cette instruction de saisie ne met pas le système en attente d'une donnée : si un caractère est disponible, il est lu et affecté à la chaîne spécifiée dans l'instruction. Sinon, le programme poursuit son exécution. L'organigramme de la page 551 et le listing correspondant retracent un exemple de programme utilisant l'instruction INKEY$ dans une boucle d'attente. A chaque passage dans la boucle, INKEY$ examine l'état de l'unité d'entrée (clavier) et, si elle trouve un caractère prêt (pending), le transfère dans la chaîne A$. A partir de cet instant, la longueur de A$ n'est plus nulle (elle contient le caractère lu) : la condition figurant à l'instruction 70 [LEN(A$)<>0] est vérifiée, et l'instruction INKEY$ est dorénavant sautée. Au terme de la boucle, la chaîne A$ contiendra un caractère, ou bien sera encore vide (si l'opérateur n'a rien tapé).

UTILISATION DE LA FONCTION INKEY$

La fonction d'entrée INKEY$ permet d'acquérir un caractère depuis le périphérique (écran). Cette page retrace l'exécution du programme montré en exemple à la page précédente.



■ Avec la commande RUN, l'exécution démarre en ligne 40 (instruction de lecture).

```
RUN
 Nombre de fois qu'il faut
 executer la boucle? 300
ON N'A SAISI AUCUN CARACTERE
 Nombre de fois qu'il faut
 executer la boucle? 300
 Saisie d'un caractere pendant la boucle
 Nombre de fois qu'il faut
 executer la boucle? 0
Ok
```

L'exécution de cette ligne provoque l'affichage du message que le programmeur a inclus entre guillemets, suivi d'un point d'interrogation (ajouté par le système). L'opérateur tape le nombre 300 qui sera affecté par l'instruction INPUT à la variable NEND (limite supérieure de l'indice de boucle).

Dès que l'opérateur frappe la touche RETURN, après la saisie du nombre 300, l'exécution se poursuit en ligne 50 (initialisation de la chaîne A$ à vide) et se poursuit avec l'entrée dans la boucle (ligne 80) qui contient la fonction INKEY$.

■ Pendant le déroulement de la boucle, aucun caractère en attente n'a été trouvé sur le port d'entrée : la condition de la ligne 100 est vérifiée et on exécute l'instruction d'édition (ligne 130).

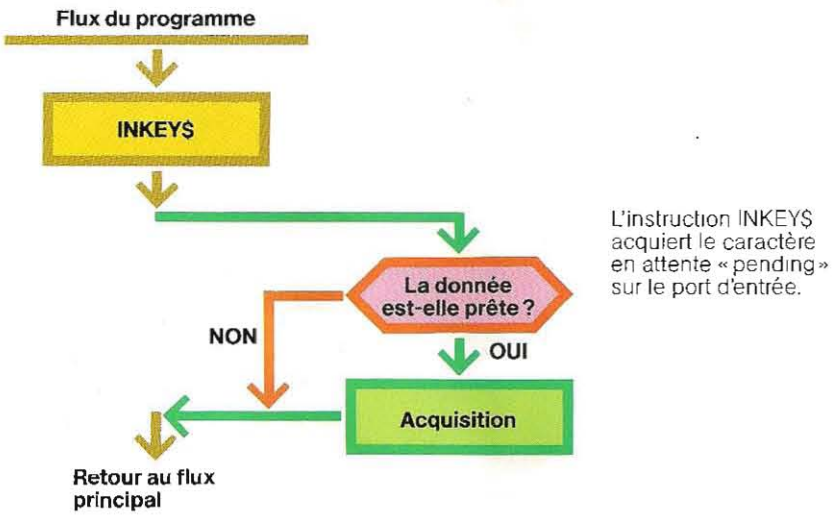
■ L'étape précédemment décrite se répète, mais cette fois l'opérateur a entré un caractère quelconque durant l'exécution de la boucle. Ce caractère n'apparaît pas à l'écran, car la fonction INKEY$ ne provoque pas d'écho, mais est affecté à A$. La condition de la ligne 100 est maintenant fausse et l'exécution passe à la ligne 110, qui provoque l'affichage du message.

■ Pour sortir de l'exécution, l'opérateur entre la valeur 0. Le test prévu en ligne 45 provoque un saut à la fin du programme.

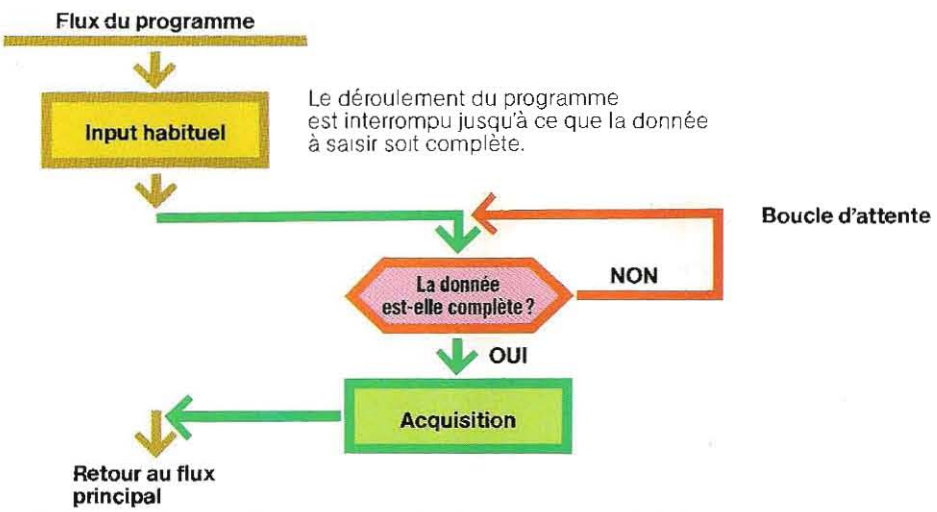
L'instruction INKEY$ fonctionne d'une façon radicalement différente des autres ordres de lecture. Si l'on remplaçait la ligne 80 (A$=INKEY$) par une des autres instructions d'entrée, l'exécution s'arrêterait à cette instruction, dans l'attente de la donnée. Avec l'instruction INKEY$, le système ne se met pas en attente : il poursuit son activité en cours, tout en gardant sous contrôle le port d'entrée, afin d'acquérir la donnée dès son introduction. Les schémas ci-dessous explicitent la logique des deux modes d'acquisition mis en œuvre.

INPUT$(N). Saisit une suite de N caractères depuis le clavier sans produire d'écho. Ainsi, A$=INPUT$(6) transfère six caractères dans la chaîne A$. Dans ce cas, il est inutile de frapper la touche CR pour mettre fin à la saisie : le système comptabilise, au fur et à mesure le nombre de caractères lus ; dès que la donnée atteint la longueur requise, l'exécution passe à l'instruction suivante.

FONCTIONNEMENT DE L'INSTRUCTION INKEY$



FONCTIONNEMENT DES AUTRES INSTRUCTIONS DE SAISIE



EXEMPLES D'UTILISATION DE LA FONCTION INPUT$(N)

```
10 ' ** EXEMPLES D'UTILISATION DE LA FONCTION INPUT$(N)
20 '    FICHIER : INPUT
30 DEFINT A-Z
40 INPUT "Nombre de caracteres a saisir ";N
50 IF N<=0 OR N>80 GOTO 40        ' CONTROLE DU NOMBRE N DE CARACTERES
60 '
70 INPUT "Echo demande (repondre OUI ou NON) ",REP$
80 '
90 ' Le format utilise ligne 70 (symbole "," et non ";") supprime
100 ' le point d'interrogation affiche par le systeme
110 '
120 IF REP$<>"OUI" AND REP$<>"NON" GOTO 70     ' CONTROLE
130 PRINT "Entrez ";N;" caracteres"
140 A$=INPUT$(N)
150 IF REP$="OUI" THEN PRINT A$
160 '
170 ' * SECONDE METHODE
180 '
190 PRINT "Seconde methode : A$ est reecrit caractere par caractere"
200 A$=""
210 FOR I=1 TO N
220 B$=INPUT$(1)     ' LECTURE D'UN SEUL CARACTERE A LA FOIS
230 PRINT B$;
240 A$=A$+B$     ' LE CARACTERE SAISI EST CONCATENE A A$
250 NEXT I
260 '
270 INPUT "VOULEZ-VOUS CONTINUER (OUI/NON)";REP$
280 IF REP$="OUI" GOTO 40
290 END
```

La fonction INPUT$(N) (il ne s'agit pas d'une instruction) ne produit pas d'écho ; il faut donc, pour visualiser les caractères entrés, le spécifier au moyen d'une instruction insérée dans le programme.
L'exemple listé ci-dessus utilise la fonction INPUT$(N), et prévoit les deux formes possibles (avec ou sans écho). Pour obtenir un écho, il faut lire puis envoyer à l'écran un seul caractère à la fois, de façon à ce que la saisie soit immédiatement suivie de son écho (seconde méthode).

LINE INPUT. Permet d'acquérir une ligne entière de caractères (normalement jusqu'à un maximum de 256). Par exemple, l'instruction

LINE INPUT "Commentaire";A$

transfère dans A$ une chaîne saisie au clavier.

Fonctions de sortie des données

Pour l'impression ou l'affichage à l'écran, le système réalise un transcodage des données (par exemple de binaire en ASCII), et les formate selon les instructions de l'utilisateur.
Il y a plusieurs façons d'écrire des données ; les valeurs numériques peuvent comporter un nombre variable de décimales, ou nécessitent parfois des caractères spéciaux. Chacun des aspects sous lesquels une même donnée peut se présenter à l'impression constitue un **format**. L'utilisateur dispose d'instructions appropriées pour spécifier le format qu'il souhaite ; en cas de sortie non formatée (aucune instruction explicite), le système adopte par défaut la représentation exponentielle pour les données numériques, et écrit les chaînes sans les modifier.
Rappelons que le format exponentiel d'une valeur numérique se compose d'une mantisse (chiffres significatifs du nombre en question, séparés par un éventuel point décimal) suivie du symbole E±nn, où E indique qu'il faut multiplier la mantisse par une puissance de 10, nn représentant l'exposant de celle-ci. Si l'exposant est positif, le signe + peut être omis. Ainsi, la notation 4931.3579E2 représente le nombre 493135.79 (4931.3579 × 100, car E2 vaut 100).
Les instructions les plus simples d'émission de données sont PRINT (pour l'affichage à l'écran) et LPRINT (pour la sortie sur imprimante). Pour utiliser un format particulier, l'instruction devient PRINT USING « spécification de format » (« imprime en utilisant... »)

pour l'écran, et LPRINT USING... pour l'imprimante.
Remarquons cependant que l'écran possède seulement 80 colonnes (si nous excluons les consoles graphiques), alors que l'imprimante tolère jusqu'à 132 caractères, et parfois plus. Des problèmes de longueur de ligne peuvent donc se présenter : ils se résoudront en répartissant sur plusieurs lignes de l'écran les données à éditer.

LPRINT. La formulation habituelle de cette instruction est :

LPRINT "Commentaires" ; VARIABLE1 ; ...

Le Basic imprime le message spécifié entre guillemets (symboles "), puis les valeurs des variables énumérées dans l'instruction (VARIABLE1, etc.) Ainsi l'exécution de :

```
10  V1=3.516
20  V2=75
30  LPRINT "Sortie=",V1,V2
```

provoque l'impression de la ligne :

```
Sortie=3.516      75
```

Les valeurs numériques ne sont écrites en format exponentiel (Enn) que si elles dépas-

EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT"

```
10 ' *** EXEMPLES D'UTILISATION DE L'INSTRUCTION (LPRINT) ***
20 '              FICHIER = LPR
30 A1=123
40 A2=456
50 A3=789
60 LPRINT "ESSAI N.1",A1,A2,A3
65 LPRINT                          ' ESPACEMENT
70 LPRINT "ESSAI N.2",A1,A2,A3
75 LPRINT                          ' ESPACEMENT
80 LPRINT "ESSAI N.3",A1,A2,A3
85 LPRINT                          ' ESPACEMENT
90 END
```

```
ESSAI N.1      123           456           789

ESSAI N.2      123  456  789

ESSAI N.3 123  456  789
```

```
10 ' *** EMPLOI DE LPRINT AVEC DES CHAINES **
20 '              FICHIER = LPRC
30 A$="PARIS"
40 B$="EST"
50 C$="GRAND"
60 LPRINT "ESSAI N.1";A$;B$;C$
70 LPRINT
80 LPRINT "*** ATTENTION : L'ESPACEMENT CORRECT EST
90 LPRINT "    PREVU PAR L'INSTRUCTION D'ECRITURE ***"
100 LPRINT "ESSAI N.2",A$,B$,C$
120 END
```

```
ESSAI N.1 PARIS EST GRAND

*** ATTENTION : L'ESPACEMENT CORRECT EST
    PREVU PAR L'INSTRUCTION D'ECRITURE ***

ESSAI N.2      PARIS          EST            GRAND
```

sent le nombre de chiffres maximum autorisé (suivant leur type d'appartenance, 6 chiffres pour les réels, 15 en double précision) ; dans notre exemple, l'exposant est donc omis.
A l'impression, chaque ligne est divisée en champs de 14 caractères. En séparant les variables par une virgule, chacune est imprimée au début de son champ correspondant ; au contraire, en utilisant le point-virgule comme symbole de séparation, toutes les variables seront imprimées à la suite. Si les données sont de type numérique, elles seront cependant espacées par un seul blanc.
Ces deux notations (, ou ;) peuvent également figurer à la fin de l'instruction ; en ce cas, l'exécution du LPRINT suivant continuera l'impression sur la ligne en cours. Tout se passe alors comme si la nouvelle instruction LPRINT était une continuation de la ligne précédente. La page 556 illustre quelques exemples d'édition de variables numériques et de chaînes.

TAB(N). Les fonctions d'émission de données, et surtout de sortie sur l'imprimante, nécessitent souvent une très grande liberté dans le positionnement des champs à éditer. Les instructions LPRINT (ou PRINT), avec leurs options ; et , offrent la possibilité d'éditer les données de manière contigüe ou de les répartir en champs de 14 caractères. Pour obtenir des tabulations plus précises, on emploie la fonction TAB(N), qui déplace de N caractères la zone d'édition de la donnée. Ainsi, l'instruction LPRINT TAB(20);B$ provoque l'impression de la chaîne contenue dans la variable B$ à partir de la position 20. Quelques exemples d'emploi de la fonction TAB(N) seront présentées par la suite, dans le chapitre sur la mise en page des données.

LPRINT USING. Permet l'impression de données numériques ou de chaînes, selon un format défini par l'utilisateur. Voici la syntaxe :

LPRINT USING"Spécification de format";VARIABLES

Les spécifications de format établissent, au moyen de symboles conventionnels, des modes particuliers d'édition.
Les formats prévus par le Basic 80 sont réunis ci-dessous.

■ Variable de chaîne

```
"!"
```

n'imprime que le premier caractère de la chaîne.

```
"\ \"
```

imprime un nombre de caractères égal au nombre d'espaces compris entre les deux symboles \ plus deux.
Par exemple, les lignes

```
10 A$="ESSAI D'EDITION"
20 LPRINT USING "\ \";A$
30 ' 2 blancs entre les deux back slash
```

imprimeront 4 caractères (2 espaces + 2) de la chaîne A$.

```
"&"
```

imprime la chaîne sans contrôle de longueur.

■ Variables numériques

```
"##.#"
```

spécifie* le nombre de chiffres à imprimer, en fixant la longueur de la partie entière et de la partie décimale.
Le nombre de chiffres de la partie entière est égal au nombre des symboles # (dièse) qui précèdent le point décimal.
Par exemple, l'exécution de

```
10 R=35.716
20 LPRINT USING "##.#;";R
```

provoque l'impression du nombre 35.7 (2 chiffres pour la partie entière, puis une décimale).

```
"+"
```

permet l'écriture du signe de la donnée avant ou après sa valeur numérique. Dans certaines applications commerciales, il est d'usage de placer le signe après le nombre ; ainsi, la valeur −519 s'imprimera 519−.
Par exemple, les lignes

```
10 R=125.365
20 PRINT USING "+####.##";R
30 PRINT USING "####.##+";R
```

* Sur certaines imprimantes, le symbole # apparaît sous la forme £.

fournissent deux sorties :
– 125.36 (ligne 20 : seules 2 décimales figurent) 125.36 – (ligne 30).

"—"

Placé obligatoirement à la fin de la chaîne décrivant le format ("##.#— "), il provoque l'impression d'un éventuel signe — à la suite du nombre. Ce symbole est comparable à l'option "+" placée à la fin du champ, mais ici le signe n'est imprimé que s'il est négatif.

"**"

Le double astérisque remplace les éventuels espaces blancs par le symbole *. La partie entière du nombre à éditer peut quelquefois contenir un nombre de chiffres inférieur à celui prévu par le format "#". Si l'on ne spécifie pas d'autre option, il apparaîtra dans le champ d'édition autant d'espaces blancs que de chiffres manquants.
Par exemple, l'impression du nombre 7.1543 avec le format "####.##" donne en sortie 7.15 alors que par le format "**#####.##", elle donne ***7.15.

"$$"

imprime le symbole $ (dollar) juste avant le nombre. Avec ce format, l'éventuel signe négatif doit toujours figurer à la suite de la valeur numérique. Ainsi la spécification "$$—##.#" est erronée (signe — précédant le nombre); la forme correcte est "$$##.#—"

"**$"

Ce format est une combinaison des deux précédents. Il remplit les espaces blancs par des astérisques et place le symbole ($) de devise au début du nombre.

","

La virgule , utilisée pour mettre en évidence les milliers, selon la notation anglo-saxonne, est placée tous les trois chiffres de la partie entière, en partant de la droite. Dans les formats, elle doit toujours apparaître à gauche du point décimal. Ainsi la spécification "####,.##", appliquée au nombre 7563.121, produit en sortie : 7,563.12
L'unique variante autorisée consiste à placer la virgule à la fin du format lui-même (####.##,). Dans ce cas, le symbole , sera imprimé à la fin du nombre. L'impression du nombre précédent sera 7563.12,

"XXXX"

Quatre caractères quelconques, inclus dans un format, servent à spécifier la notation exponentielle : ils permettent de réserver l'espace nécessaire à la notation exponentielle E+nn (où nn représente l'exposant), qui est précisément constituée de 4 caractères.
Le format "##.#AAAA", appliqué au nombre 1200, produit l'impression de 12.0E+02 (E+02=10^2= 100;12.0×100=1200), puisque l'on demande seulement trois chiffres, dont une décimale (##.#), en format exponentiel (AAAA).

"%"

Ce symbole, lorsqu'il fait partie d'un format, apparaît devant la donnée, si elle dépasse la longueur du champ déclaré. Par exemple, le nombre 1470.65, imprimé avec le format "####.#", apparaît sous la forme %1470.6 car sa zone décimale déborde du champ prévu par le format.

Quelques exemples d'utilisation des différents formats sont programmés page 559. Remarquons que le système, en phase d'édition, arrondit automatiquement les données numériques dont la partie décimale excède en longueur celle du champ déclaré. Par exemple, le nombre 412.67, édité avec le format "###.#", devient 412.7 puisqu'on ne demande qu'une seule décimale.
L'arrondi se calcule selon les règles habituelles : si le chiffre à partir duquel s'effectue la troncature est supérieur à 5, on ajoute 1 au chiffre précédent ; sinon il est supprimé. Par exemple, avec un format autorisant une seule décimale, le nombre 56.86 sera transformé en 56.9 (le chiffre retiré étant 6), alors que 56.85 apparaîtra sous la forme de 56.8. Ainsi, une différence de 1/100 seulement entre les valeurs réelles (en mémoire) de deux variables peut devenir en sortie égale à 1/10, du fait des arrondis.

EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT USING"

```
10 ' ** EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT USING"
20 '
30 '              FICHIER = LPRUSI
40 ' *** OPTION "!" (IMPRIME SEULEMENT LE PREMIER CARACTERE DE LA CHAINE)
50 A$="PARIS ET LES PARISIENS"
60 LPRINT USING "!";A$
70 LPRINT                    ' ESPACEMENT
80 '
90 ' *** OPTION "£££.£" (VARIABLES NUMERIQUES)
100 R=123.789        ' N.B. LE NOMBRE POSSEDE UNE PARTIE ENTIERE A 3 CHIFFRES,
105 '                      ET TROIS DECIMALES
110 LPRINT USING "£££.£";R    ' IMPRIME (R) AVEC TROIS CHIFFRES
120 '                         POUR LA PARTIE ENTIERE ET UNE SEULE DECIMALE
130 LPRINT
140 '
150 ' *** OPTION "%"
160 LPRINT USING "%££.££";R    ' LE SYMBOLE % METTRA EN EVIDENCE QUE
170 '                           LE NOMBRE IMPRIME DEBORDE DU FORMAT D'IMPRESSION
180 LPRINT USING "%£££.££";R
190 LPRINT : LPRINT
200 '
210 ' *** OPTION "+£££.££"
220 LPRINT USING "+£££.££";R    ' IMPRIME (R) AVEC TROIS CHIFFRES POUR LA PARTIE
230 '                            ENTIERE, DEUX DECIMALES ET LE SIGNE A GAUCHE
240 ' *** OPTION "£££.££+"
250 LPRINT USING "£££.££+";R    ' COMME CI-DESSUS, MAIS AVEC LE SIGNE A DROITE
260 LPRINT
270 LPRINT USING "+£££.£££";R   ' IMPRIME (R) DANS UN FORMAT TEL QU'AUCUN
280 '                            CHIFFRE N'EST TRONQUE
290 LPRINT USING "£££.£££+";R   ' COMME CI-DESSUS, MAIS AVEC LE SIGNE A DROITE
300 LPRINT
310 '
320 ' OPTION "£££.£££-"         ' N.B. CE FORMAT FAIT APPARAITRE LE SIGNE
330 '                            A DROITE, AVEC OMISSION S'IL EST POSITIF
340 LPRINT Q=-456.789           ' (VALEUR NEGATIVE)
350 LPRINT USING "£££.£££-";Q
360 LPRINT USING "£££.£££-";Q
370 END
```

```
 R

 123.8

 %%123.79
 %123.79

 +123.79
 123.79+

 +123.789
 123.789+

 123.789
 456.789-
```

Si ces formats sont prévus dans la bibliothèque Basic Standard, tous les interpréteurs (ou compilateurs) ne les proposent cependant pas. En Basic Applesoft, par exemple, aucun formatage des données n'est possible : seule existe l'instruction TAB(N).

Les autres langages évolués, comme le Fortran ou le Pascal, offrent de nombreuses possibilités de formatage, à travers un jeu d'instructions très développé.

Test 16

1 / Une **matrice** est une structure mathématique qu'on peut représenter par un tableau à deux dimensions (lignes et colonnes). Par exemple, une matrice 4 × 5 (4 lignes, 5 colonnes) sera mémorisée sous la forme d'un tableau A (4,5).
Ecrire un programme qui saisit au clavier les cinq éléments de la première ligne de la matrice A [A(1,1), A(1,2)..., A(1,5)] et qui calcule les autres éléments, comme le produit du numéro de ligne à laquelle ils appartiennent par l'élément correspondant de la ligne 1. Par exemple, pour la ligne 2, on aura A(2,1)=2*(1,1), A(2,2)=2*A(1,2); en ligne 3, A(3,4)=3*A(1,4).

2 / Ecrire un sous-programme d'impression de la matrice A(4,5) calculée précédemment, soit de façon normale (ligne par ligne), soit de façon inverse (colonne par colonne).

3 / A l'aide de la fonction INPUT$(N), écrire un sous-programme de lecture (avec écho) d'un sigle formé de trois caractères, qui vérifie l'absence de caractère numérique.

4 / Généraliser la routine précédente en l'adaptant à la lecture de N caractères, compris entre 1 et 40.

Les solutions du test se trouvent page 567.

Structuration des impressions

Les sous-programmes d'impression peuvent être conçus de deux manières : en écrivant les instructions selon un format déterminé, ou en paramétrant la ligne à imprimer. La plus courante est la première méthode, qui consiste à écrire pour chaque application une routine spécifique contenant les instructions nécessaires, et qui, en aucun cas, ne s'adaptera à d'autres programmes. La seconde méthode, beaucoup plus complexe, consiste à préparer une variable de chaîne contenant toutes les informations que l'on désire obtenir sur une ligne d'impression. La routine d'impression se limite alors à l'émission de cette variable, appelée **vecteur d'impression**.
La première technique est rigide : pour changer un seul paramètre, il faut modifier le programme. La seconde peut devenir très exhaustive, au point d'obtenir un programme qui demande à l'utilisateur quelles sont les variables à imprimer et leurs positions respectives sur la page. Une telle généralisation demande une expérience d'analyse et de programmation considérable que nous illustrerons par quelques exemples très simplifiés.

Impressions non paramétrées. Voici un premier exemple. On désire imprimer un registre de clients, contenant les coordonnées décrites dans le tableau 1 et les données comptables du tableau 2 (voir page 561). Il faut prévoir un saut de page à chaque nouveau client (changement de code client) ou toutes les 30 lignes de données comptables, même si le client ne change pas. Les pages seront numérotées et, à la fin de l'édition relative à chaque client, on totalisera les montants nets (somme des champs Z2).
La première étape consiste à préparer le schéma d'impression. Sur une feuille quadrillée (il existe des feuilles prévues à cet effet), on reporte à la main les colonnes d'impression, les lignes concernant les coordonnées et les 30 lignes (maximum prévu pour chaque page) de données comptables, en les cadrant aux positions voulues, conformément à la mise en page que l'on désire obtenir.
De cette manière, la position de chaque champ d'impression est déterminée par les coordonnées (ligne et colonne) de la feuille quadrillée. La programmation consistera à traduire en instructions ce qui est représenté sur la feuille.
La page 562 illustre un exemple de disposition des champs sur 80 colonnes, destiné aux

imprimantes les plus petites ; on adaptera aisément cette grille à 132 colonnes, par exemple en espaçant davantage les champs. En haut de la page 563 est schématisé le programme principal, et en bas l'organigramme du sous-programme d'impression. Les fonctions exécutées par ce dernier sont :

1 / reconnaissance du code client ; au changement de code, on doit imprimer le total (Z3) relatif au client précédent, réinitialiser à zéro la variable Z3 pour y cumuler les nouvelles données, et imprimer les coordonnées du nouveau client ;

2 / reconnaissance de la première page ; le schéma précédent ne peut s'appliquer au premier client : en effet, aucun cumul de données précédentes (Z3) ne doit être imprimé. Le programme teste simplement le numéro de la page : si celui-ci est nul, l'instruction d'impression du total précédent est sautée, et l'en-tête seul est imprimé ;

3 / contrôle des lignes ; pour chaque ligne de donnée comptable imprimée, on incrémente un compteur. Dès que la valeur 31 est atteinte, on passe à une nouvelle page, même s'il s'agit toujours du même client.

Sur l'organigramme de la page 563 (en bas), on vérifie en effet que l'impression de l'en-tête et des coordonnées est nécessaire en deux points différents du programme. Il sera donc préférable d'utiliser un sous-programme au lieu de répéter les mêmes instructions. Cette structuration en routines distinctes présente un autre avantage : les différentes fonctions sont nettement séparées, facilitant la compréhension par le lecteur, et, de plus, les éventuels changements pour adapter le programme à de nouvelles applications en sont largement simplifiés.
La page 564 illustre l'organigramme de la routine d'impression de l'en-tête et des coordonnées. Ce sous-programme est activé par la rupture de séquence (changement de code client) ou après l'impression de 30 données comptables. Le programme est listé page 566. Dans cet exemple, la saisie des données s'effectue au clavier, mais dans les applications réelles, les données seront lues à partir de fichiers disque.

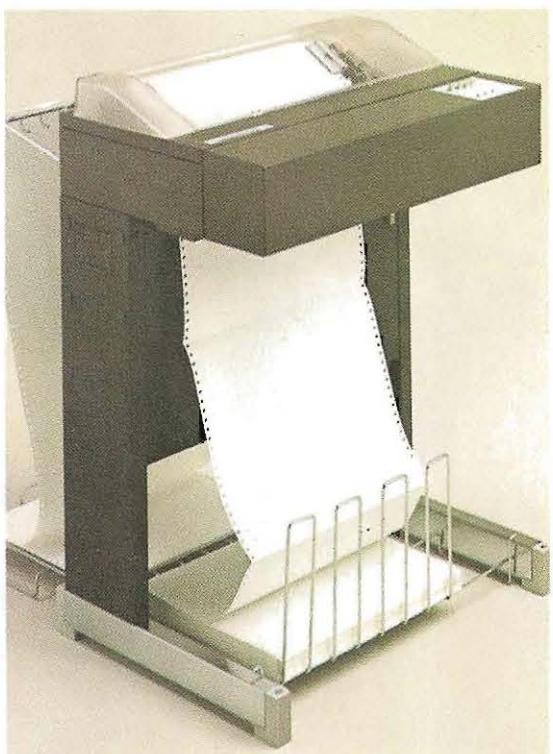
Tableau 1

Nom du champ	Signification	Type	Longueur
CC	Code client	numérique	3
NO$	Nom	alphabétique	15
RU$	Rue	alphabétique	10
NU	Numéro	numérique	5
TL	Téléphone	numérique	12 (indicatif et numéro)
CP	Code postal	numérique	5
VI$	Ville	alphabétique	15
DP$	Département	alphabétique	10

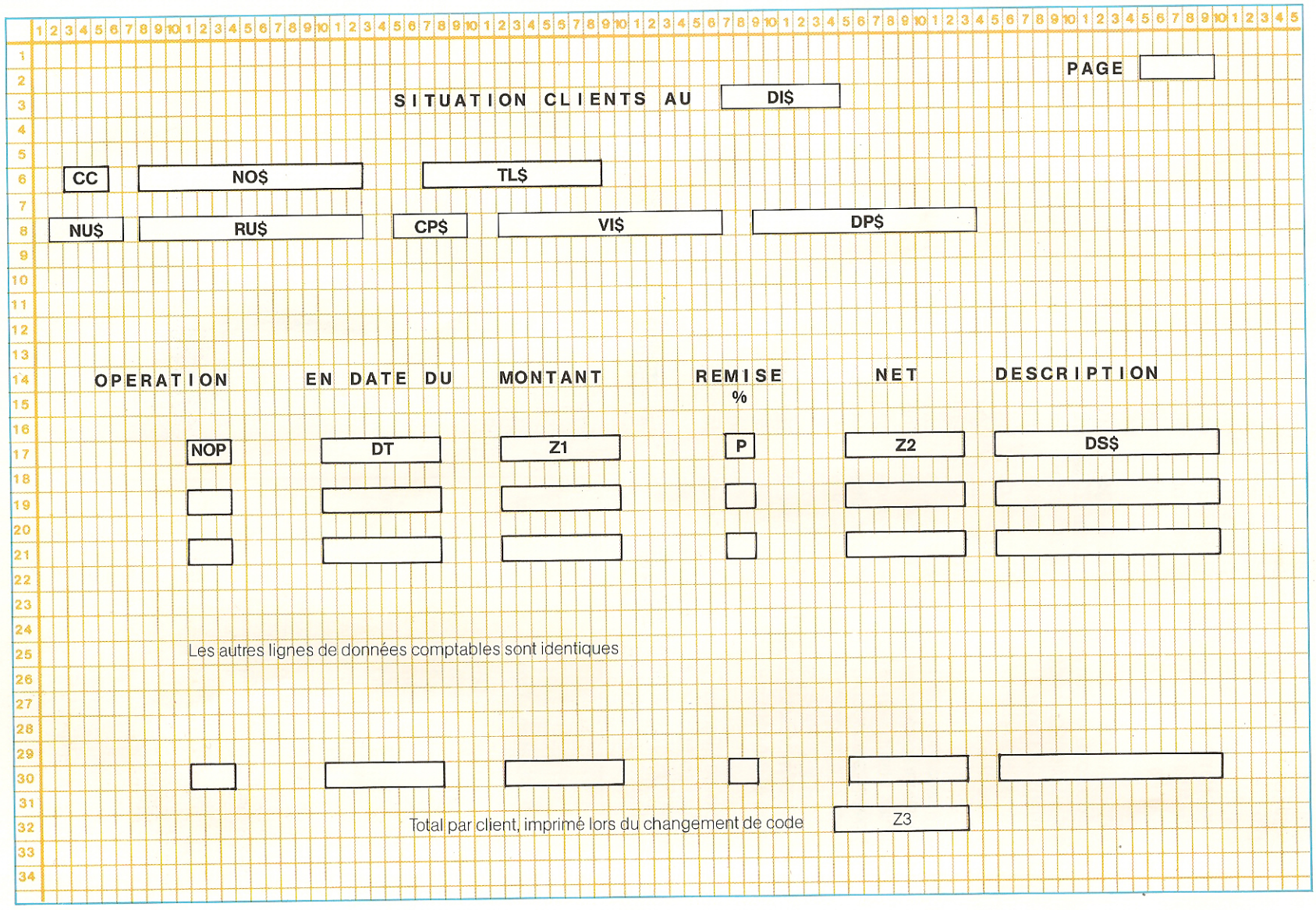
Tableau 2

Nom du champ	Signification	Type	Longueur
NOP	Numéro d'opération	numérique	3
DT	Date	numérique	8
Z1	Montant	numérique	8
P	Remise (%)	numérique	2
Z2	Montant net	numérique	8
DS$	Description	alphabétique	15

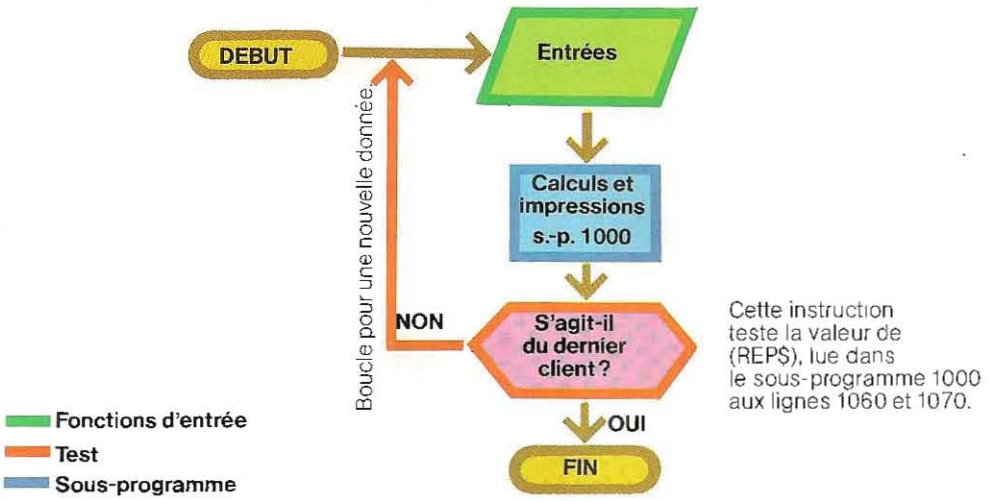
Imprimante bi-directionnelle. L'impression est sous le contrôle d'un microprocesseur.



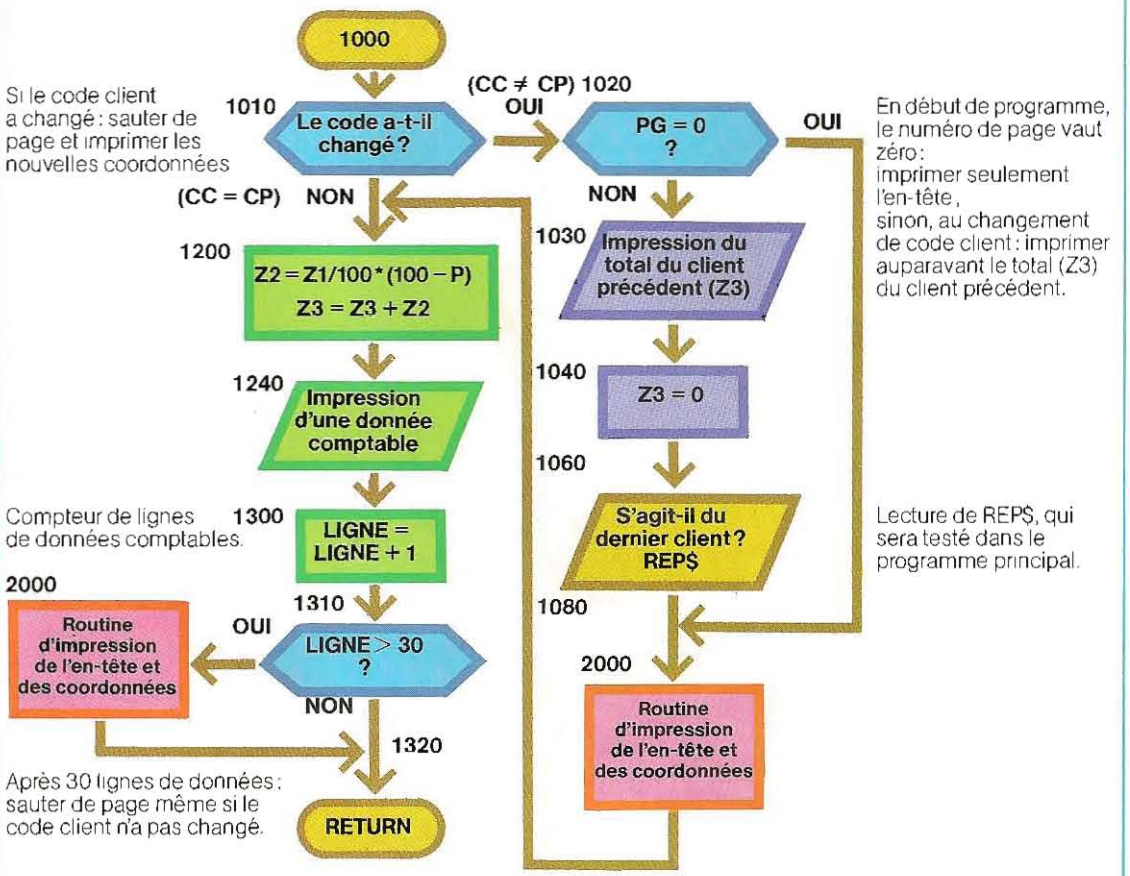
Olivetti


PAGE
SITUATION CLIENTS AU
DI$
CC
NO$
TL$
NU$
RU$
CP$
VI$
DP$
OPERATION
EN DATE DU
MONTANT
REMISE %
NET
DESCRIPTION
NOP
DT
Z1
P
Z2
DS$
Les autres lignes de données comptables sont identiques
Total par client, imprimé lors du changement de code
Z3

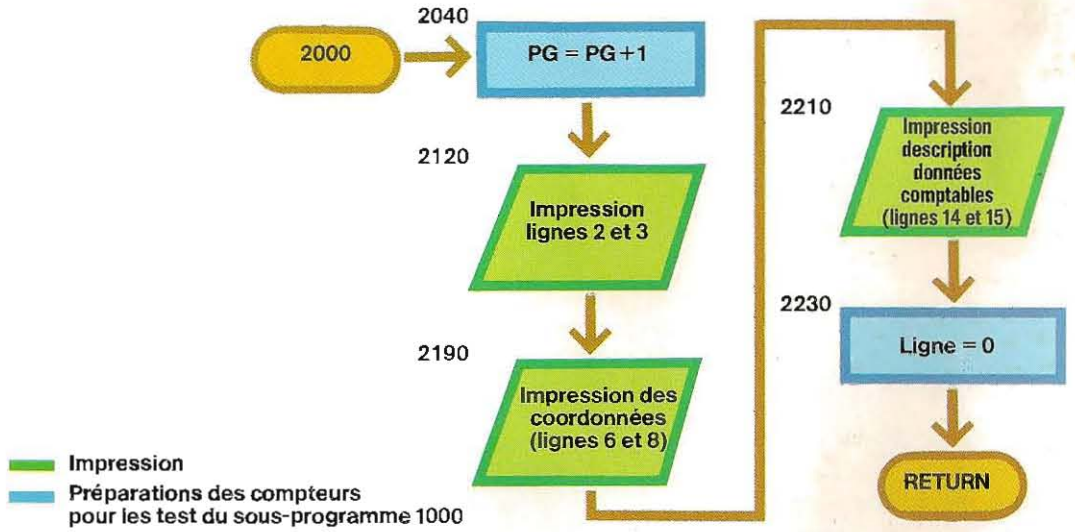
PROGRAMME PRINCIPAL D'IMPRESSION D'UN REGISTRE CLIENTS (sur 80 colonnes)



SOUS-PROGRAMME 1000 : GESTION DES IMPRESSIONS



SOUS-PROGRAMME 2000 : IMPRESSION EN-TETE ET COORDONNEES



PROGRAMME D'EDITION SUR 80 COLONNES

```
5 ' ** PROGRAMME D'EDITION SUR 80 COLONNES **
10 '
15 '              FICHIER = TA80C
20 '
25 ' VARIABLES UTILISEES
30 ' ** coordonnees clients
35 ' NOM DE CHAMP      SIGNIFICATION          TYPE           LONGUEUR
40 ' CC                CODE CLIENT            numerique       3
45 ' NO$               NOM                    alphabetique    13
50 ' RU$               RUE                    alphabetique    15
55 ' NU                NUMERO DE RUE          numerique       5
60 ' TL                TELEPHONE              numerique       12
65 ' CP                CODE POSTAL            numerique       5
70 ' VI$               VILLE                  alphabetique    15
75 ' DP$               DEPARTEMENT            alphabetique    15
80 '       * donnees comptables
85 ' NOP               NUMERO D'OPERATION     numerique       3
90 ' DT                DATE                   numerique       8
95 ' Z1                MONTANT                numerique       8
100 ' P                REMISE                 numerique       2
105 ' Z2               MONTANT NET            numerique       8
110 ' DS               DESCRIPTION            alphabetique    15
115 '
120 ' CC = CODE CLIENT EN COURS
125 ' CP = CODE CLIENT PRECEDENT
130 ' DI = DATE D'IMPRESSION
135 ' PG = NUMERO DE PAGE EN COURS
140 '
145 ' INITIALISATIONS
150 BI$=CHR$(27)+"+"+CHR$(7)    ' Efface l'ecran et emet un "bip"
151 CP=0            ' Initialisation du code client precedent
152 PG=0            ' Initialisation du numero de page
153 Z3=0            ' Initialisation de la somme des montants nets
155 '
160 ' ** PROGRAMME PRINCIPAL **
165 INPUT "DATE D'IMPRESSION ";DI
170 PRINT "INTRODUIRE LE CODE CLIENT (NUMERIQUE, 3 CHIFFRES MAXI)"
175 INPUT CC : IF CC=CP GOTO 215    ' Si le code client n'a pas change
180 INPUT "NOM ";NO$                ' on passe directement a la
185 INPUT "RUE ";RU$                ' saisie des donnees comptables
190 INPUT "NUMERO ";NU
```

```
195 INPUT "TELEPHONE ";TL
200 INPUT "VILLE ";VI$
205 INPUT "CODE POSTAL ";CP
210 INPUT "DEPARTEMENT ";DP$
215 INPUT "NUMERO D'OPERATION ";NOP
220 INPUT "DATE ";DT
225 INPUT "MONTANT ";Z1
230 INPUT "REMISE (%) ";P
235 INPUT "DESCRIPTION (15 CARACTERES MAX) ";DS$
240 GOSUB 1000              ' Calculs et impression
250 CP=CC
255 '
260 PRINT BI$
270 IF REP$<>"OUI" GOTO 170
280 END
285 '
290 ' ** SOUS-PROGRAMME D'IMPRESSION DU FICHIER CLIENTS **
1000 IF CC=CP GOTO 1200     ' Si CC=CP, il n'y a pas eu
1010                        ' de changement de code
1020 IF PG=0 THEN GOSUB 2000 : GOTO 1200
1030 LPRINT TAB(55); : LPRINT USING "£££££.££";Z3
1040 Z3=0
1050 LPRINT CHR$(12)        ' Saut de page
1060 PRINT "EST-CE LE DERNIER CLIENT ? (OUI/NON)
1070 INPUT REP$
1080 GOSUB 2000             ' Impression coordonnees client
1090 '
1100 '
1110 '
1120 ' Calcul du montant net total
1130 '
1200 Z2=Z1/100*(100-P)      ' On deduit de Z1 la remise P
1210 '                      ' pour obtenir le montant net Z2
1220 Z3=Z3+Z2               ' Cumul dans Z3 des montants nets
1230 '
1240 ' Impression d'une ligne de donnees comptables
1245 '
1250 LPRINT TAB(1); : LPRINT USING "£££";NOP; : LPRINT TAB(20);DT;
1260 LPRINT TAB(32); : LPRINT USING "£££££.££";Z1;
1270 LPRINT TAB(47); : LPRINT USING "££";P;
1280 LPRINT TAB(55); : LPRINT USING "£££££.££";Z2; LPRINT TAB(65);DS$
1290 LPRINT
1300 LIGNE = LIGNE+1
1310 IF LIGNE>=30 THEN LPRINT CHR$(12) : GOSUB 2000
1320 RETURN
1330 '
2000 ' ** SOUS-PROGRAMME D'IMPRESSION D'EN-TETE **
2010 '    ET DE COORDONNEES CLIENT
2020 '
2030 LPRINT
2040 PG=PG+1          ' Incrementation du numero de page
2050 LPRINT TAB(70);"PAGE";PG
2060 LPRINT TAB(25);"SITUATION CLIENTS AU ";TAB(47);DI
2070 LPRINT
2080 '
2090 ' Impression des coordonnees
2100 '
2110 LPRINT
2120 LPRINT TAB(3);CC;TAB(8);NO$;TAB(27);TL
2130 LPRINT
2140 LPRINT TAB(2);NU;TAB(8);RU$;TAB(25);CP;TAB(32);VI$;TAB(49);DP$
2150 LPRINT : LPRINT
2160 LPRINT : LPRINT
2170 LPRINT
2180 '
2190 LPRINT TAB(5);"OPERATION";TAB(19);"EN DATE DU";TAB(32);"MONTANT";
2200 LPRINT TAB(45);"REMISE";TAB(57);"NET";TAB(65);"DESCRIPTION"
2210 LPRINT TAB(8);"NUMERO";TAB(47);"%"
2220 LPRINT
2230 LIGNE=0
2240 RETURN
```

RESULTATS DU PROGRAMME TAB80C (PAGE 1)

PAGE 1

SITUATION CLIENTS AU 4/10/84

1 MARTIN [06]25-88-10

123 RUE ST-JACQUES 28000 CHARTRES EURE-ET-LOIR

OPERATION NUMERO	EN DATE DU	MONTANT	REMISE %	NET	DESCRIPTION
936	10/10/83	1230.00	10	1107.00	MARCH. FRAGILE
937	11/10/83	1450.00	15	1232.00	MARCH. VARIEE
938	12/10/83	9000.00	18	7380.00	EXPLOSIFS
939	13/10/83	11920.00	10	10728.00	PDTS CHIMIQUES
				20447.00	

RESULTATS DU PROGRAMME TAB80C (PAGE 2)

PAGE 2

SITUATION CLIENTS AU 4/10/84

2 LEBLANC [06]48-48-00

64 RUE G.-CLEMENCEAU 91405 ORSAY ESSONNE

OPERATION NUMERO	EN DATE DU	MONTANT	REMISE %	NET	DESCRIPTION
800	10/11/83	9700.00	5	9215.00	JEUX
				9215.00	

Impressions paramétrées. Cette application, limitée à un cas très simple, est effectuée dans un but de démonstration. Elle permettra au lecteur de se familiariser avec certaines instructions Basic, et d'acquérir les notions fondamentales de la méthodologie*.

La page 568 schématise l'organigramme du programme principal synthétisant les fonc-

* Nous n'aborderons pas ici l'utilisation et la manipulation des masques d'édition

Solutions du test 16

1 / Le programme exécutant les fonctions demandées peut prendre la forme suivante :

```
10  OPTION BASE 1
20  DIM A(4,5)
25  ' Saisie des éléments de la première ligne
30  FOR I=1 TO 5
40  PRINT "Elément:";I
50  INPUT "Valeur";A(1,I)
60  NEXT I
70  ' Calcul des éléments de la deuxième ligne
80  FOR I=1 TO 5
90  A(2,I)=2*A(1,I)
100 NEXT I
110 ' Calcul des éléments
    de la ligne 3
120 FOR I=1 TO 5
130 A(3,I)=3*A(1,I)
140 NEXT I
150 ' Calcul des éléments
    de la ligne 4
160 FOR I=1 TO 5
170 A(4,I)=4*A(1,I)
180 NEXT I
200 END
```

Les trois boucles calculant les éléments des trois lignes 2, 3, 4 peuvent être réunies en une seule. Les instructions des lignes 70 à 180 sont alors remplacées par les suivantes :

```
70  FOR J=2 TO 4 ' Sélection du numéro de ligne
80  FOR I=1 TO 5
90  A(J,I)=J*A(J,I) ' Calcul de l'élément en Je ligne et Ie colonne
100 NEXT I
110 NEXT J
```

---

2 / La routine se résume à deux boucles imbriquées, la première sélectionnant la ligne, la seconde sélectionnant la colonne. Selon l'indice que l'on fait varier en premier, on obtient l'impression par ligne ou par colonne. On choisira par exemple :

```
1000 FOR J=1 TO 4
1010 FOR I=1 TO 5
1020 PRINT A(J,I)
1030 NEXT I
1040 NEXT J
1050 RETURN
```

---

3 / Le programme répétera trois fois l'instruction A$=INPUT$(1), suivie du contrôle de numéricité à l'aide de la fonction VAL (le code ASCII du caractère doit être compris entre 48 et 57). Si la donnée saisie n'est pas numérique, on affiche A$, sinon le programme doit demander une nouvelle valeur de A$.

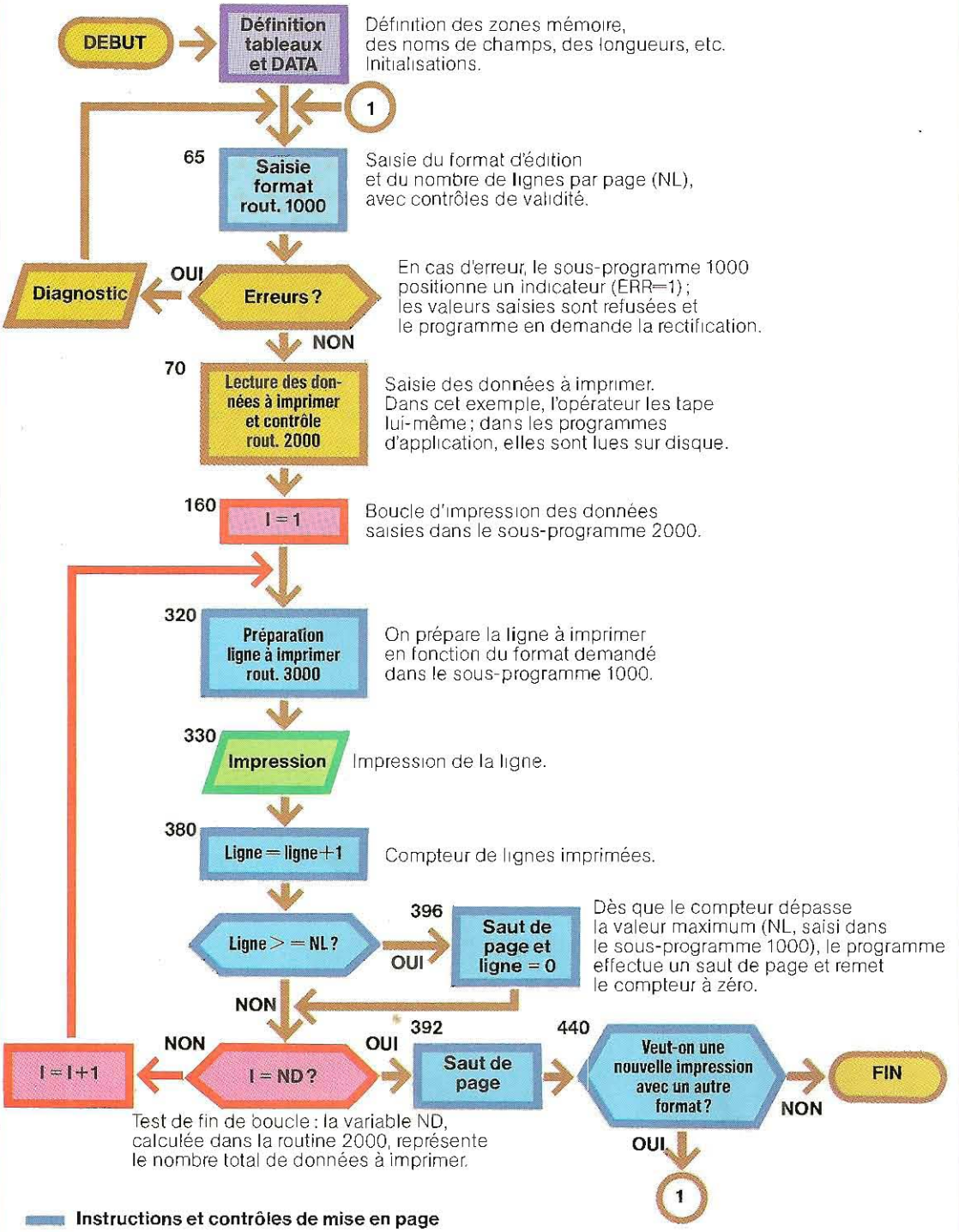
---

4 / On généralise ce programme en effectuant une boucle de 1 à N, réalisant les fonctions suivantes :

lecture d'un caractère [A$=INPUT$(1)] ;
contrôle de numéricité ;
affichage à l'écran si la donnée est acceptée.

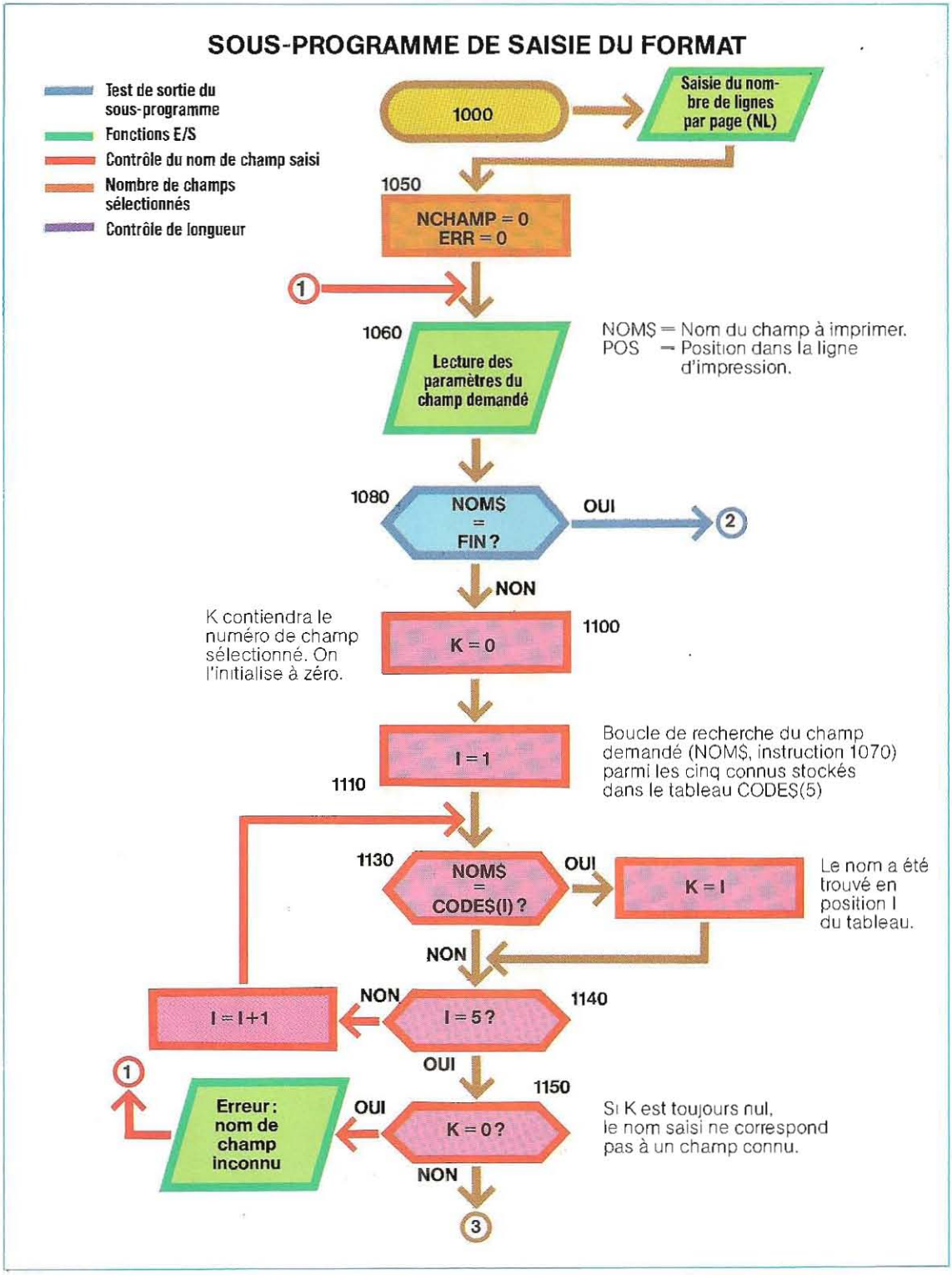
Le programme est très semblable à l'exemple d'utilisation de la fonction INPUT$(N) présenté page 555.

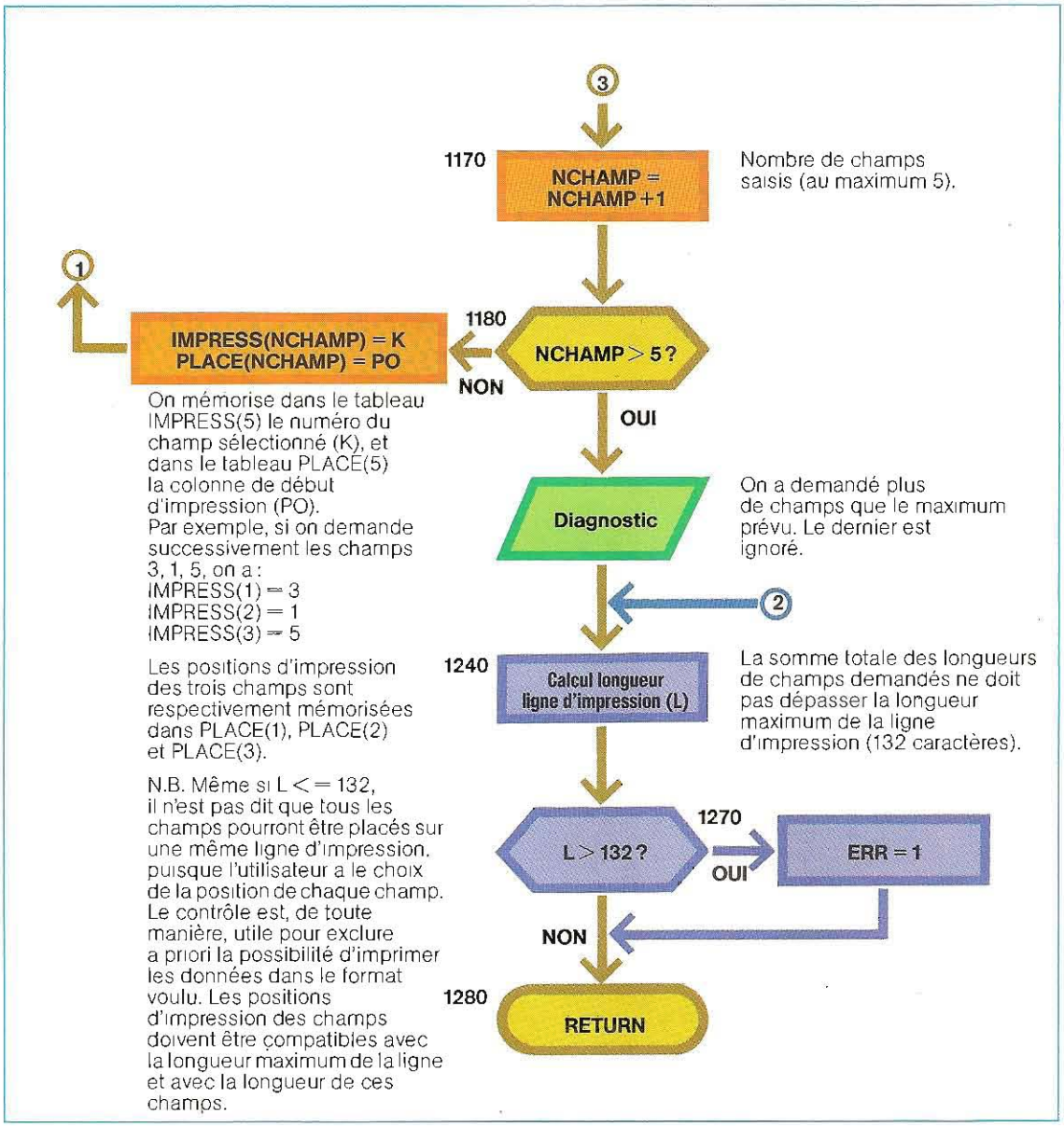
PROGRAMME PRINCIPAL : IMPRESSION PARAMETREE



tions à exécuter ; les organigrammes suivants (pages 569 à 573) détaillent les routines 1000, 2000 et 3000 appelées par le programme principal. Le listing complet et certains résultats sont représentés pages 574 à 576.

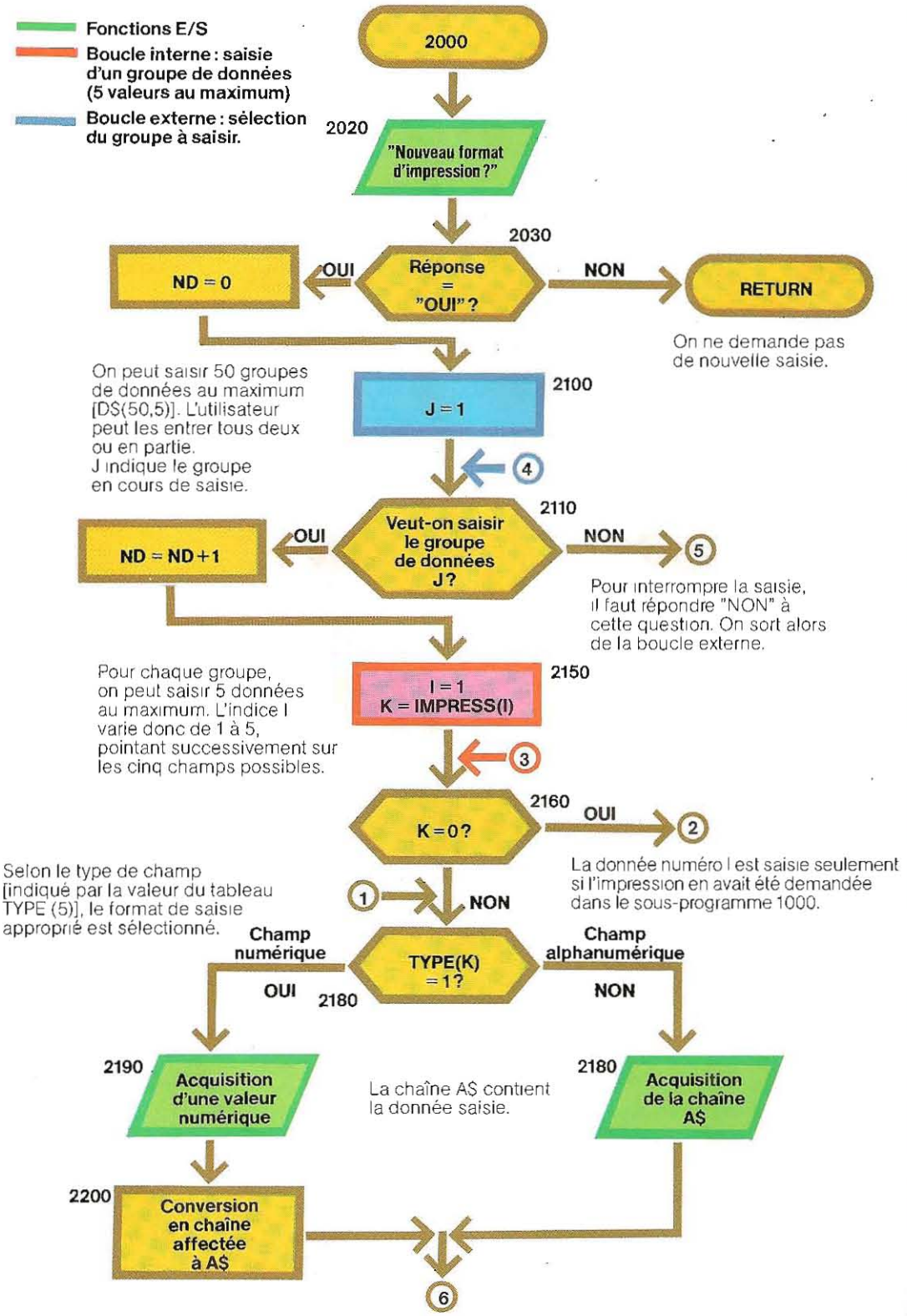
Le programme principal définit un certain nombre de champs (cinq dans notre exemple), dont les noms et longueurs sont fixés une

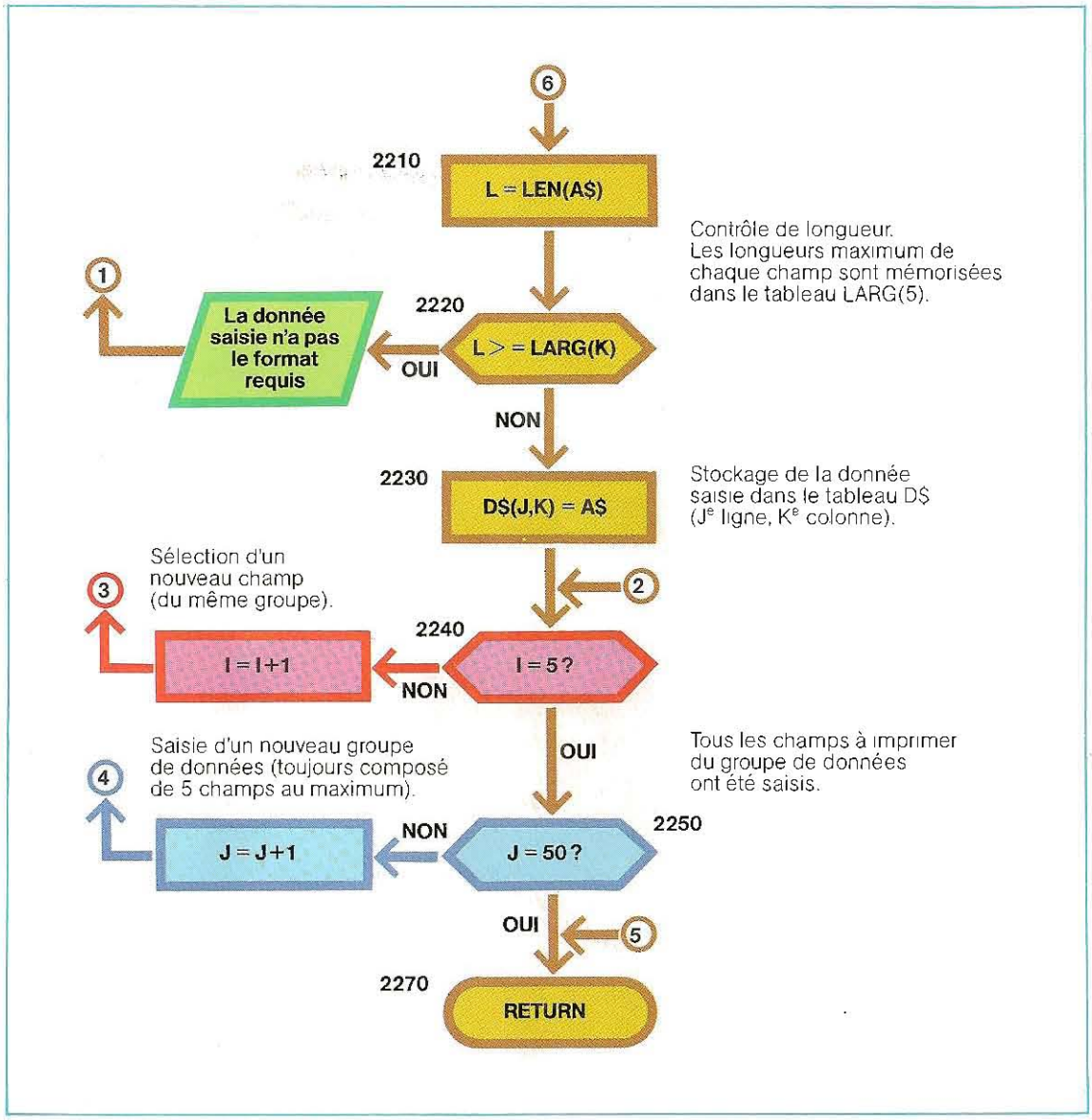




fois pour toutes (par exemple, lors d'une instruction DATA). Le sous-programme 1000 demande à l'utilisateur de spécifier successivement les champs qu'il désire traiter et la position où ils devront être imprimés.
Un certain nombre de contrôles sont prévus : vérification du nom de champ saisi, du nombre total de champs à imprimer, et de la longueur finale de la ligne d'impression (qui ne doit pas excéder 132 caractères). Pour ce dernier test, on cumule les longueurs respectives des différents champs.
La routine 2000 simule une lecture sur disque des groupes de données à imprimer (au maximum 50) : on demande à l'utilisateur de taper les donnés dont il a demandé l'impression dans le sous-programme 1000, et on contrôle la longueur de chaque chaîne saisie, qui sera ensuite stockée dans le tableau D$(50,5). La variable ND comptabilise le nombre total de groupes saisis.
Le sous-programme 3000 chargé de préparer chaque ligne d'impression (correspondant à un groupe de données), concatène successivement dans la chaîne A$ le contenu de chaque champ à imprimer (A2$, lu dans le tableau D$), et la chaîne d'espaces (A1$), qui séparera le champ en cours du précédent.

SOUS-PROGRAMME DE LECTURE ET DE CONTROLE DE DONNES A IMPRIMER



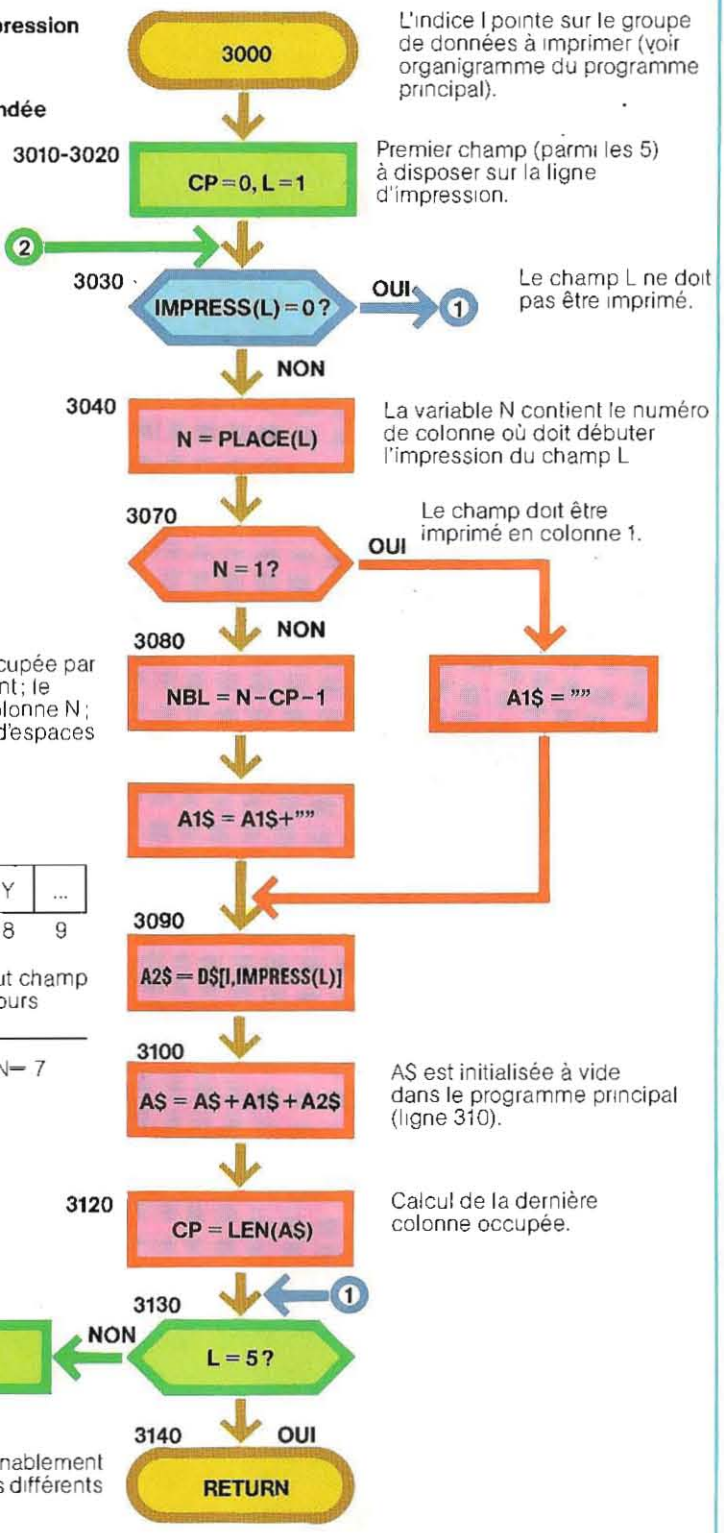


Notre programme d'application pourrait se compléter des fonctions réalisant par exemple des opérations arithmétiques de logiques entre les champs, ou d'autres plus sophistiquées (prévoyant une possibilité de représentation graphique). Avec un interface, permettant d'utiliser des valeurs stockées sur disque comme données d'entrée, on obtiendra un logiciel de portée très générale, offrant à l'utilisateur final un nombre varié de traitements spécifiques sans qu'il ait à entrer au cœur de la programmation.

Le déroulement de ce type de logiciel, s'il est destiné à une application pratique et non, comme dans notre exemple, à une démonstration, présente des difficultés notables et demande l'emploi de langages plus performants et plus proches de la machine que le Basic. Généralement, ces packages d'application sont écrits en Assembleur, ou même directement en langage machine sous forme de code hexadécimal. Cependant, l'utilisation de langages évolués tels que le Pascal ou le LISP se répand progressivement. Ces langages, employés en « intelligence artificielle » (surtout le LISP), permettent, grâce à leur haut niveau de structuration, la conception de programmes beaucoup plus complexes, et avec un gain de temps considérable par rapport à l'Assembleur.

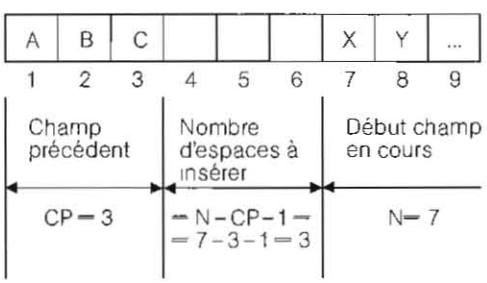
SOUS-PROGRAMME DE PREPARATION DE LA LIGNE D'IMPRESSION

Boucle sur les champs d'impression
Préparation de la ligne
Saut des champs dont l'impression n'est pas demandée



CP est la dernière colonne occupée par la donnée du champ précédent ; le nouveau champ débute en colonne N ; il faut donc insérer un nombre d'espaces égal à NBL = N - CP - 1

Exemple



La chaîne A$ contient, convenablement séparés par des espaces, les différents champs à imprimer.

PROGRAMME D'IMPRESSION PARAMETREE

```
10 REM PROGRAMME D'IMPRESSION PARAMETREE
11 REM ********************************
20 REM
30 REM executable sur APPLE II en basic APPLESOFT
40 REM
45 REM
50 ERR = 0
60 REM Variables
65 REM NL=nombre max de lignes/page (sous-pr. 1000)
70 REM ND=nombre max de donnees a imprimer (sous-pr. 2000)
75 REM
80 REM Tableaux
85 DIM CODE$(5): REM contient les 5 noms de champs prevus
90 DIM LARG(5): REM longueur des champs
95 DIM TYPE(5): REM type de champ (1=numerique, 2=alphanumerique)
100 DIM PLACE(5): REM numero de colonne de debut d'edition de champ
105 DIM IMPRESS(5): REM contient, dans l'ordre voulu, les pointeurs
106 REM                sur les champs a editer
110 DIM D$(50,5): REM tableau de travail contenant les donnees saisies
120 REM
125 REM Initialisations
130 BI$=CHR$(7): REM pour emettre un "bip"
131 SP$=CHR$(12): REM pour un saut de page
135 L$=CHR$(4)
140 REM
150 REM ** PROGRAMME PRINCIPAL **
155 REM
160 FOR I=1 TO 5
170 PRINT "CHAMP N.";I
180 INPUT "NOM DU CHAMP (3 caracteres) ";CODE$(I)
190 INPUT "LONGUEUR EN CARACTERES ";LARG(I)
200 INPUT "TYPE DE CHAMP (1=numerique 2=alphanumerique) ";TYPE(I)
210 PRINT L$;"PR#1"
215 REM Sur l'APPLE, cette instruction transfere le controle
216 REM a l'imprimante : tous les PRINT qui suivent
217 REM correspondront au LPRINT du Basic standard
220 PRINT "CHAMP N.";I; TAB(5);"NOM: ";CODE$(I); TAB(5);
222 PRINT "TYPE: ";TYPE(I); TAB(5);"LONGUEUR: ";LARG(I)
223 REM En Applesoft, l'instruction TAB(N) affiche N espaces,
224 REM au lieu de placer le curseur en position N
225 PRINT L$;"PR#0": REM rend le controle a la console
230 NEXT I
235 REM
240 GOSUB 1000
245 REM Saisie du format d'edition, du nombre de lignes/page,
246 REM validation des donnees saisies
247 REM
250 IF ERR=1 THEN PRINT BI$: PRINT "ERREUR: CORRIGEZ LES VALEURS"
260 IF ERR=1 THEN PRINT "ENTREES": GOTO 240
265 REM
270 GOSUB 2000
275 REM saisie des donnees, tapees au clavier par l'operateur
280 REM (dans les applications reelles, elles sont lues sur disque)
290 REM
295 REM
300 FOR I=1 TO ND: REM boucle d'impression par la routine 3000
310 A$=""
320 GOSUB 3000
325 REM
330 PRINT L$;PR#1": PRINT A$: REM l'imprimante edite la ligne
340 REM selon le format saisi en 1000
360 PRINT L$;"PR#0"
370 REM
380 LIGNE=LIGNE+1: REM compteur de lignes imprimees
```

```
390 IF LIGNE<NL GOTO 400
392 PRINT L$;"PR#1": PRINT SP$: REM saut de page
394 PRINT L$;"PR#0"
396 LIGNE=0
397 REM
400 NEXT I
410 REM
415 REM
420 PRINT L$;"PR#1": PRINT SP$: REM saut de page
425 PRINT L$;"PR#0"
430 PRINT BI$
435 REM
440 PRINT "VEUT-ON UNE IMPRESSION AVEC UN NOUVEAU FORMAT ? (OUI/NON)"
450 INPUT REP$
460 IF REP$="OUI" GOTO 160
470 REM
480 END : REM fin pr. principal
490 REM
500 REM
1000 REM ** SOUS-PROGRAMME DE SAISIE DU FORMAT **
1010 REM
1020 PRINT BI$
1030 PRINT "ENTREZ LE NOMBRE DE LIGNES PAR PAGE"
1040 INPUT NL
1045 REM
1050 NCHAMP=0: REM initialise NCHAMP (nombre a imprimer)
1060 REM lecture des parametres du champ
1070 INPUT "NOM DU CHAMP (tapez FIN pour sortir)";NOM$
1080 IF NOM$="FIN" GOTO 1220
1090 REM
1100 K=0: REM K contiendra un pointeur vers le champ demande
1110 FOR I=1 TO 5: REM boucle de recherche de NOM$
1120 REM              parmi les champs connus (CODE$)
1130 IF NOM$=CODE$(I) THEN K=I: REM le nom se trouve en position I
1140 NEXT I
1145 REM
1150 IF K=0 THEN PRINT "ERREUR, NOM DE CHAMP INCONNU": GOTO 1070
1160 REM si K est encore nul, le nom saisi ne figure pas dans CODE$
1165 REM
1170 NCHAMP=NCHAMP+1
1180 IF NCHAMP>5 THEN PRINT "PLUS DE CHAMPS QUE PREVU": GOTO 1220
1185 REM
1190 IMPRESS(NCHAMP) = K: PLACE(NCHAMP) = PO
1200 GOTO 1070
1210 REM
1220 L=0
1230 FOR I=1 TO NCHAMP
1240 L=L+LARG(IMPRESS(I))
1250 NEXT I
1260 REM
1270 IF L>132 THEN ERR=1
1280 RETURN
1290 REM
1300 REM
2000 REM ** ROUTINE DE LECTURE ET CONTROLE DES DONNES A IMPRIMER **
2010 REM
2020 INPUT "SAISIE DE NOUVELLES DONNEES (OUI/NON) ";REP$
2030 IF REP$="NON" GOTO 2270
2040 PRINT BI$
2050 PRINT "on peut saisir au maximum 50 groupes de donnees"
2060 REM (car D$(50,5) c.f. ligne 110)
2070 PRINT "l'utilisateur peut les entrer tous ou en partie"
2080 PRINT
```

```
2090' ND=/
2100 FOR J=1 TO 50
2110 PRINT "SAISIE DU GROUPE DE DONNEES ";J;"(OUI/NON)?"
2120 INPUT REP$
2130 IF REP$="NON" GOTO 2270
2140 ND=ND+1
2150 FOR I=1 TO 5
2155 K=IMPRESS(I)
2160 IF K=0 GOTO 2240
2170 PRINT "ENTREE DU CHAMP N.";K
2180 IF TYPE(K)<>1 THEN INPUT "chaine ";A$: GOTO 2210
2190 INPUT "Valeur ";IV
2200 A$=STR$(IV)
2210 L=LEN(A$)
2220 IF L>LARG(K) THEN PRINT "DONNEE SAISIE HORS FORMAT": GOTO 2180
2230 D$(J,K)=A$: REM memorisation de la donnee saisie
2240 NEXT I
2250 NEXT J
2260 REM
2270 RETURN
2280 REM
3000 REM ** SOUS-PROGRAMME PREPARANT LA LIGNE A IMPRIMER **
3010 CP=0
3020 FOR L=1 TO 5
3030 IF IMPRESS(L)=0 GOTO 3130
3040 N=PLACE(L): REM N=numero de colonne ou commencera
3050 REM          l'edition du champ L
3060 A1$=""
3070 IF N=1 GOTO 3090
3080 FOR NBL=1 TO N-CP-1:A1$=A1$+" ": NEXT NBL
3090 A2$=D$(I,IMPRESS(L))
3100 A$=A$+A1$+A2$
3110 REM calcul de la dernière colonne occupee
3120 CP=LEN(A$)
3130 NEXT L
3140 RETURN
```

```
CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 20
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 15
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 8
   NOM DU CLIENT       VILLE RESIDENCE    1000
   UNTEL               PARIS              0
   MARTIN              DIJON              1572.12
   DURAND              LYON               -3256.5

CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 3
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 3
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 10
     100   TELEPHONE       VIL      1000
     110   254-90-39       PAR      0
     120   54-90-39        DIJ      1572.12
     130   51-31-48        LYO      -3256.5

CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 10
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 3
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 10
      NOM CLIENT      75000   VIL   10000
      LEBLANC         75018   PAR   0
      MARTIN          22000   DIJ   1572.12
      DURAND          69000   LYO   -3256.5
```

Gestion de l'imprimante

L'instruction LPRINT telle que nous l'avons utilisée jusqu'à présent donne accès à plusieurs formats d'impression. Cette gamme s'avère cependant insuffisante à la majorité des programmes d'application.
On peut, par exemple, être tenu de modifier la taille des caractères ou l'espacement entre deux lignes consécutives. Ces conditions d'impression particulières sont imposées à l'imprimante par des commandes spéciales, généralement un ou deux caractères, qui sont alors interprétés comme des ordres et non des chaînes à imprimer.
Les constructeurs insèrent toujours dans le manuel d'utilisation une liste des commandes et des codes correspondants. Ces codes de l'imprimante sont pour la plupart constitués de caractères spéciaux (CONTROLE ou ESCAPE). On est donc généralement obligé d'avoir recours au code ASCII de ces caractères par l'intermédiaire de la fonction BASIC CHR$(N). Ainsi, le caractère CTRL L ou FF (saut de page) correspond au nombre décimal 12 selon le code ASCII. On exécutera donc cette instruction en tapant :

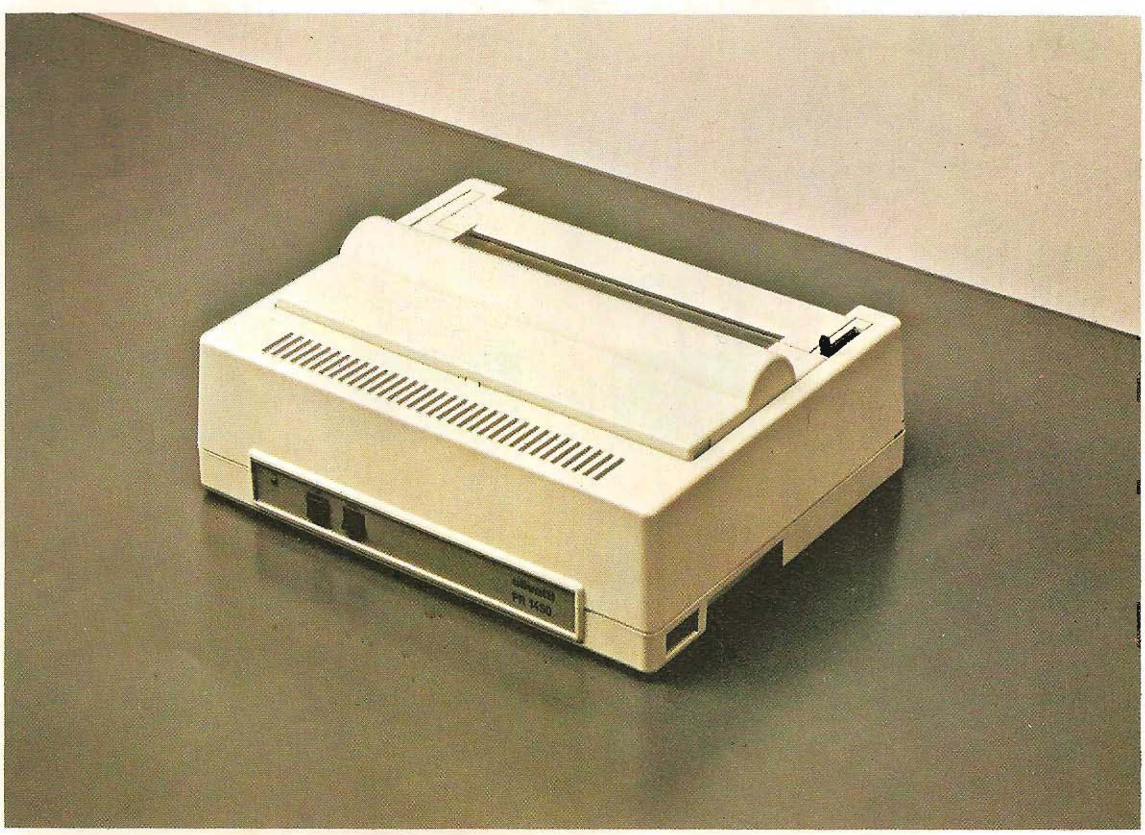
```
LPRINT CHR$(12)
```

Précisons que les commandes de l'imprimante ne font l'objet d'aucune standardisation et n'ont généralement pas les mêmes effets d'une machine à l'autre.
Les programmes cités plus loin en exemple ont été rédigés pour une imprimante EPSON RX 80, mais la mise en œuvre de codes différents reste comparable quel que soit le type de matériel.

Caractéristiques techniques des imprimantes à aiguilles

Avant d'aborder la programmation des fonctions d'édition, nous allons donner un aperçu du fonctionnement des imprimantes et, plus particulièrement, du système à aiguilles. Les imprimantes à marguerite ont en effet un jeu de caractères figé et non programmable.

Une imprimante à aiguilles : le modèle PR1450 d'Olivetti.



Olivetti

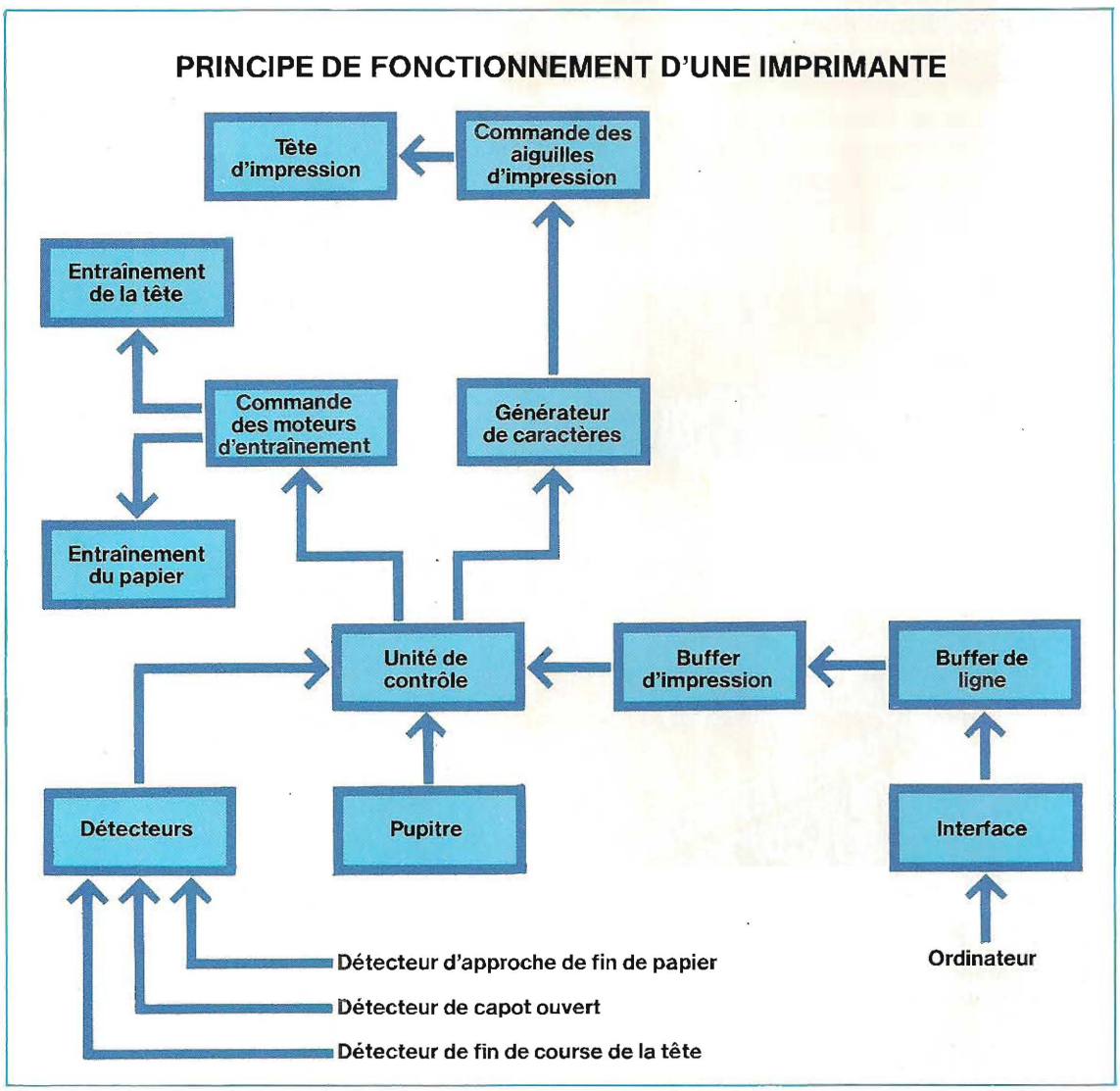
Quant aux modèles à laser ou à jet d'encre, ils sont trop récents et trop coûteux pour être encore très répandus sur les petits systèmes. On trouvera ci-dessous le schéma de principe d'une imprimante. Les données en provenance de l'ordinateur sont stockées dans une première mémoire tampon (optionnelle), le buffer de ligne, puis dans le buffer d'impression. Elles sont ensuite transmises au générateur de caractères, qui les codifie, et les dirige vers la tête d'impression.

Nous allons maintenant décrire le fonctionnement d'une imprimante à aiguilles simple: l'Olivetti PR1450. L'impression est réalisée par l'impact d'une série d'aiguilles qui viennent percuter le ruban encreur et la feuille, les appliquant contre une réglette d'appui. Il en résulte une série de points dont la juxtaposition forme les caractères.

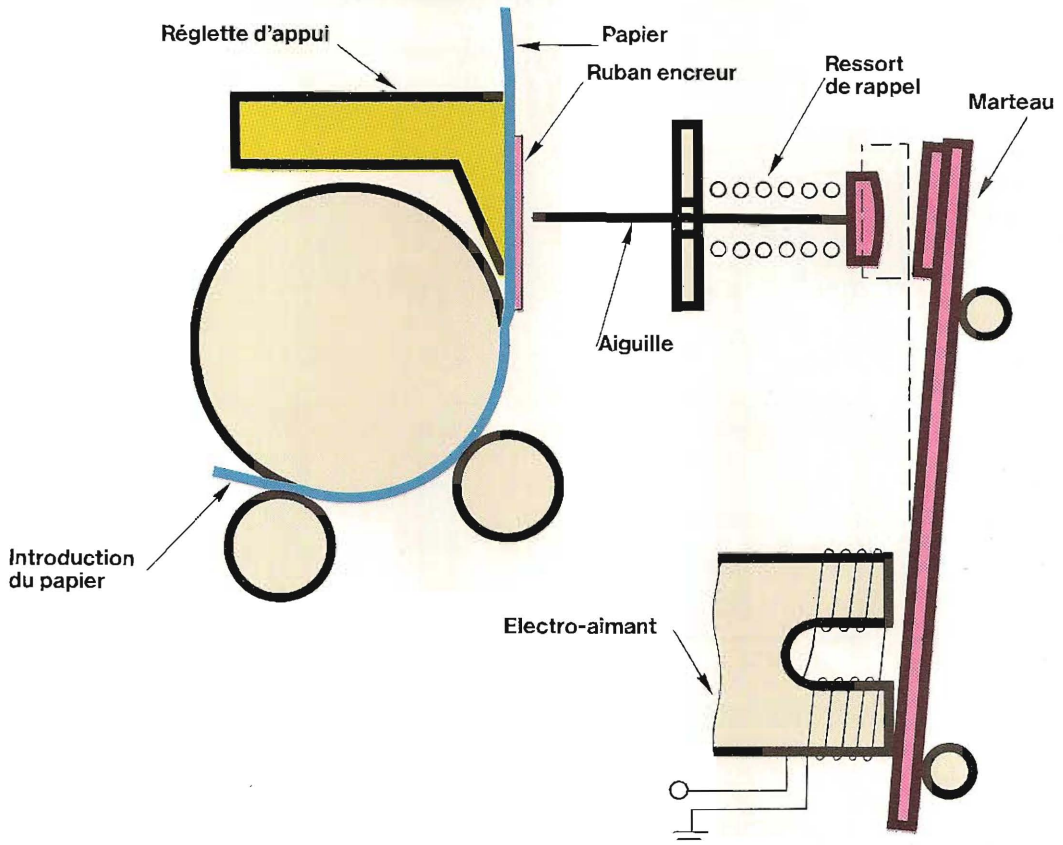
La tête d'impression regroupe neuf aiguilles. Elle est montée sur un chariot qui se déplace sur deux rails, parallèlement à la feuille de papier. Ce chariot est entraîné par un moteur à courant continu agissant sur une courroie crantée. Un levier permet de débloquer la réglette d'appui lors de l'introduction ou de la remise en place du papier.

Les empreintes laissées par les aiguilles au cours d'un déplacement horizontal de la tête composent une ligne d'écriture.

L'impact sur les aiguilles est réalisé par des petits marteaux métalliques actionnés par des électro-aimants (bobines de fil conducteur qui créent un champ magnétique lors-

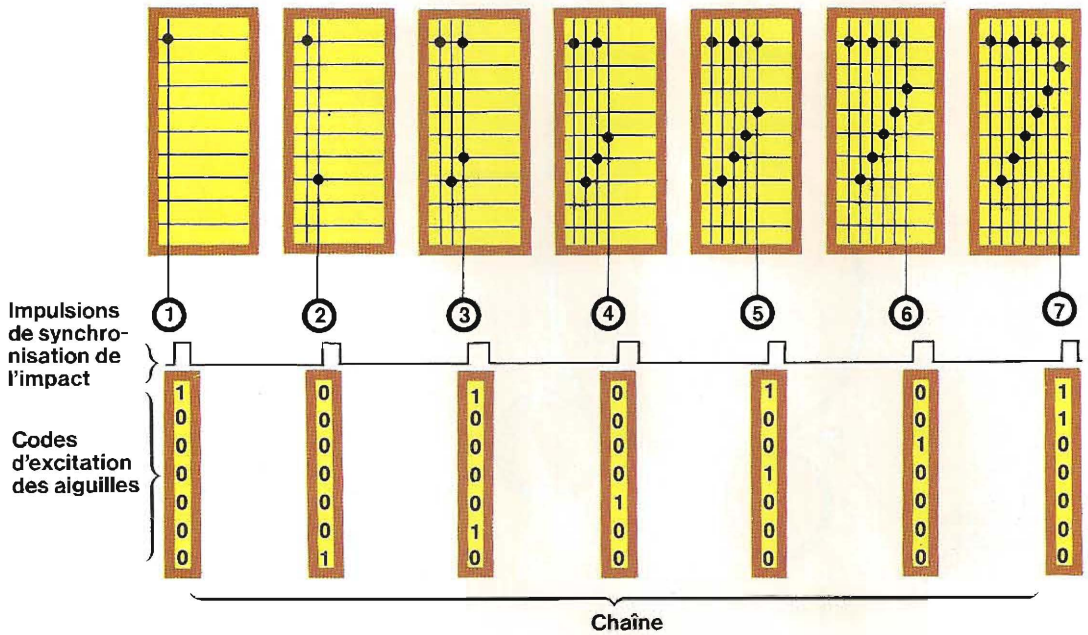


MECANISME D'IMPRESSION DU MODELE PR1450 OLIVETTI

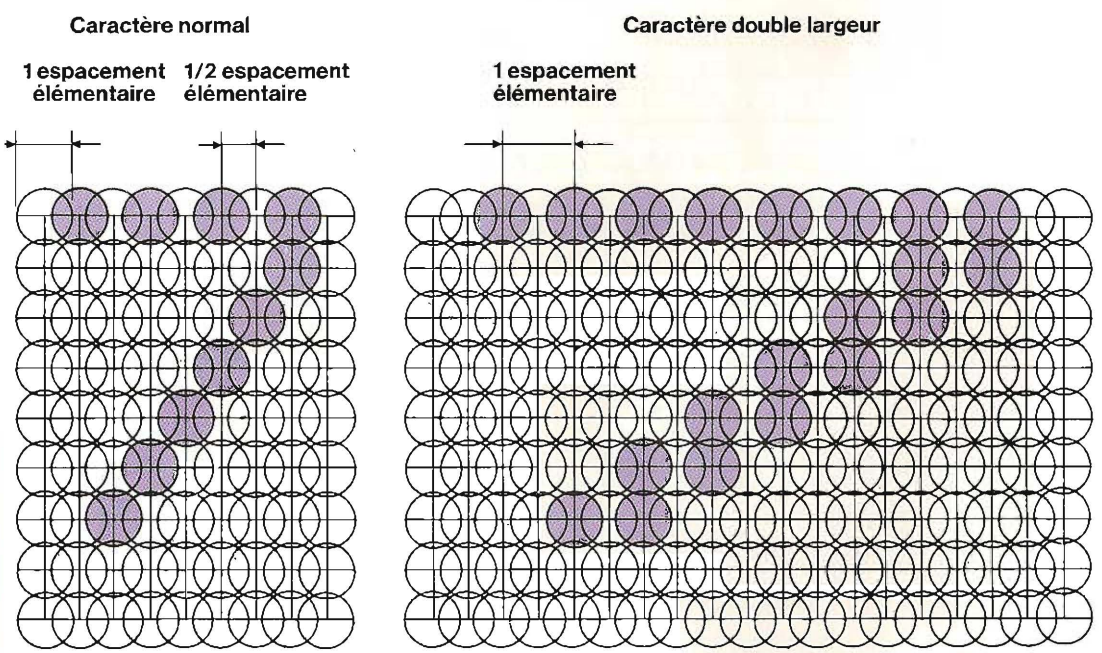


qu'elles sont alimentées par un courant électrique). Lorsqu'un courant est envoyé dans un de ces « solénoïdes », celui-ci attire le marteau qui, dans sa course, percute la tête de l'aiguille correspondante. Ce choc communique à l'aiguille assez d'énergie pour qu'elle poursuive sa course jusqu'à la feuille.
Elle rebondit ensuite et un ressort la ramène à sa position initiale (voir schéma ci-dessus). Il y a donc neuf électro-aimants dont l'excitation, durant le déplacement de la tête d'écriture le long de la ligne, provoque l'impression des combinaisons de points. Il va de soi que, pour former les caractères voulus, l'excitation de chacun doit suivre un ordre précis. Cette synchronisation est assurée par un programme spécialisé et par un générateur de caractères, transcodeur qui associe aux codes ASCII un certain nombre des points de la matrice de façon à former le caractère correspondant. On parle ainsi d'**impression matricielle**. La netteté d'un caractère est pro-

IMPRESSION DU CARACTERE 7 PAR L'IMPRIMANTE PR1450 OLIVETTI



COMPOSITION DES CARACTERES AVEC UNE MATRICE 9 × (4+3)



portionnelle au nombre de points dont il est constitué et il convient de trouver un compromis entre qualité d'impression (nombre de points de la matrice) et vitesse d'écriture en caractères par seconde.

La PR1450 utilise une matrice 9 × (4+3), c'est-à-dire une trame de neuf lignes et sept colonnes. La hauteur maximale d'un caractère est de neuf points (un par aiguille). Les sept colonnes sont en fait réparties en quatre prin-

cipales et trois intercalaires situées à un demi-espacement des principales. Il est impossible d'imprimer, sur la même ligne, deux points distants d'un demi-espacement. Les barres horizontales seront donc constituées de quatre points au maximum, les colonnes intermédiaires restant réservées aux barres obliques. On dispose donc d'une matrice de trente-six points parmi soixante-trois possibles (9 × 7).

Pour les caractères en double largeur, les points ne sont imprimés que sur les colonnes principales. Les huit codes d'excitation sont obtenus par un « et » logique entre le code correspondant et le code précédent du caractère normal.

Voici la liste des différents jeux de quatre-vingt-seize caractères disponibles sur l'imprimante PR1450 :

- INTERNATIONAL
- USASCII
- ALLEMAND
- FRANÇAIS
- ITALIEN
- ANGLAIS
- ESPAGNOL
- PORTUGAIS
- FINNO-SUEDOIS
- DANOIS-NORVEGIEN
- SUISSE

On sélectionne le jeu de caractères désiré soit par mini-interrupteurs (dip-switch), soit dans le programme lui-même.

De l'alphabet ASCII, seuls l'espacement et le caractère DEL (caractère d'oblitération) ne sont pas imprimés. Les caractères non prévus dans un jeu seront remplacés par le symbole « III ». On peut cependant « interdire » cette impression, mais aucun message ne sera alors émis. Sur la plupart des imprimantes, il est possible d'obtenir un format élargi des caractères. Avec l'Olivetti, on utilise pour cela la commande ESC 3 dont l'effet est immédiat. Cet ordre peut être introduit à n'importe quel endroit du texte et reste actif jusqu'à émission de la commande annulant ce mode (ESC 4). Cependant, l'impression redevient normale suite à toute modification de l'espacement et après une réinitialisation générale (Reset).

De même, à la mise sous tension, l'initialisation est automatique et l'impression ne passera en double largeur que si cela est demandé par le programme.

Exemple d'impression

Page 582, se trouve l'organigramme d'un programme qui illustre quelques-unes des possibilités offertes par l'imprimante EPSON RX 80.

Une liste des types d'impression disponibles

CAPACITES D'IMPRESSION D'UNE PR1450

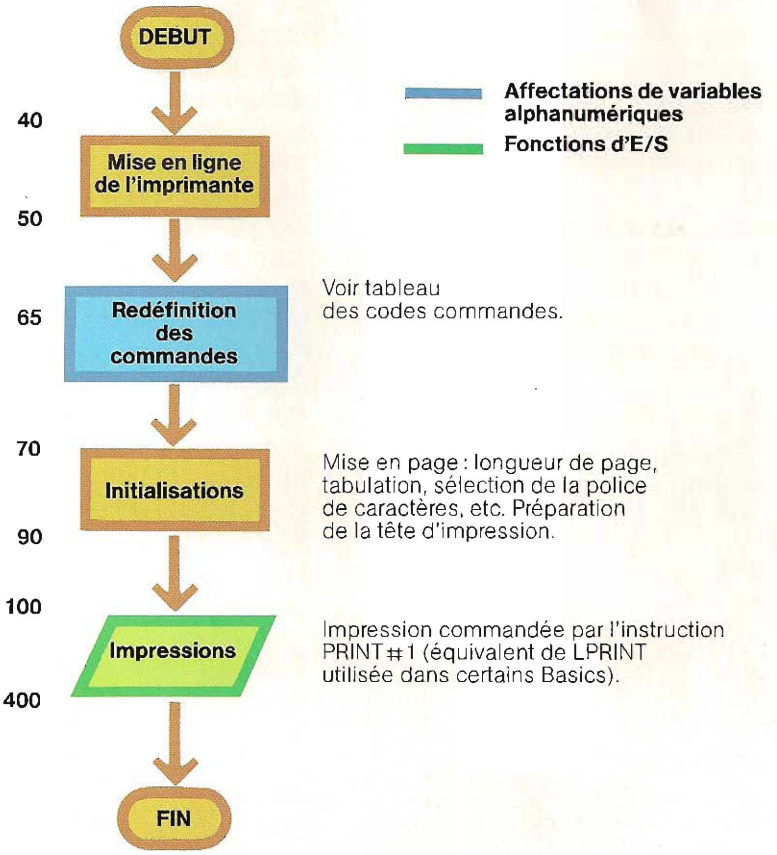
PRINTER CONTROL CODES

PRINT METHOD	Impact Dot Matrix
PRINTING SPEED	100 Char/Sec with fast carriage return
THROUGH PUT	50 lpm full line 86 lpm with 40 char
CHARACTERS COMPOSITION	9x7 (4+3) Dot Matrix
CHARACTERS PER LINE	1) 80 Char/line with 10 Char/inch 2) 132 Char/line with 16,6 Char/inch
LINE FEED	100 msec
COPIES	Up to 1 original and 2 copies

PRINTER CONTROL CODES

PRINT METHOD	Impact Dot Matrix
PRINTING SPEED	100 Char/Sec with fast carriage return
THROUGH PUT	50 lpm full line 86 lpm with 40 char
CHARACTERS COMPOSITION	9x7 (4+3) Dot Matrix
CHARACTERS PER LINE	1) 80 Char/line with 10 Char/inch 2) 132 Char/line with 16,6 Char/inch
LINE FEED.	100 msec
COPIES	Up to 1 original and 2 copies

DEMONSTRATION DES TYPES D'IMPRESSION DE LA RX80 EPSON



sur cette imprimante (à l'exclusion des capacités graphiques) est fournie par le listing de la page 583.

Les caractères sont définis sur une matrice 9 × 9 en mode normal. Le programme provoque l'impression de la dénomination de chaque mode, du nombre maximal de caractères par ligne et de la densité correspondante en CPI (caractères par inch). Le recours fréquent aux codes de commande est dû à l'aspect démonstratif de ce programme, leur emploi étant infiniment plus restreint lors d'une application usuelle. Dans un but de simplification, ces commandes ont toutes été affectées à des variables alphanumériques, ce qui évite une utilisation systématique de l'instruction CHR$.

Tableau des codes de commandes utilisés.
A$ = ESC (changement de code).
C1$ = 50 : validation de l'écriture élonguée.
D1$ = DC4 : invalidation de l'écriture élonguée.
C2$ = SI : validation de l'écriture compressée.
D2$ = DC2 : invalidation de l'écriture compressée.
E$ = ESC E : validation de l'écriture en gras.
F$ = ESC F : invalidation de l'écriture en gras.
G$ = ESC G : validation du double passage.
H$ = ESC M : invalidation du double passage.
M$ = ESC M : validation de l'écriture ELITE.
P$ = ESC P : validation de l'écriture PICA.
T1$ = ESC – SOM : validation du soulignement.
T2$ = ESC – NUL : invalidation du soulignement.
SO$ = ESC S NUL : validation du mode exposant.
S1$ = ESC S SOM : validation du mode indice.
S$ = ESC T : invalidation de ces deux modes.
ESC 4 : validation de l'écriture en italique.
ESC 5 : invalidation de l'écriture en italique.
B$ = ESC 4 + "* ECRITURE "+ESC 5 : impression du mot ECRITURE en italique.

Ce tableau présente les différentes commandes disponibles sur l'EPSON RX80 ainsi que les codes de contrôle correspondants.

EXEMPLE DE TYPES D'IMPRESSIONS

```
10 TITLE"RXIMPMOD":CLEAR500,0:WIDTH20,50
20 PRINT"**********************  EPSON    RX-80  ** MODE  IMPRESSION *";
30 PRINT"**********************"+CHR$(11)+CHR$(23)
40 OPEN"O",#1,"COM0:MODE":WIDTH"COM0:",255
50 A$=CHR$(27):B$=A$+"4* ECRITURE "+A$+"5":C1$=CHR$(14):C2$=CHR$(15):D1$=CHR$(20
):D2$=CHR$(18)
60 E$=A$+"E":F$=A$+"F":H$=A$+"H":G$=A$+"G":P$=A$+"P":M$=A$+"M"
65 T1$=A$+"-"+CHR$(1):T2$=A$+"-"+CHR$(0):S0$=A$+"S"+CHR$(0):S1$=A$+"S"+CHR$(1):S
$=A$+"T"
70 DATA 27,64,27,67,0,11,27,108,6,27,82,1,13,10
80 RESTORE 70
90 READ A:PRINT#1,CHR$(A);:IFA<>10THEN90
100 PRINT#1;E$+G$+A$+CHR$(52)+"       IMPRIMANTE EPSON   RX
80"
110 PRINT#1,A$+CHR$(53)+"            MODE  D'IMPRESSION"
120 PRINT#1,"                    *  *  *  *  *  *":PRINT#1:PRI
NT#1,F$+H$
130 PRINT#1,B$+"NORMALE 80 caractères par lignes          (10 CPI)"
140 PRINT#1,E$+B$+"EN GRAS points doublés horizontalement  (10 CPI)"+F$
150 PRINT#1,G$+B$"DOUBLE PASSAGE décalés verticalement     (10 CPI)"
160 PRINT#1,E$+B$+"GRAS ET DOUBLE PASSAGE                   (10 CPI)"F$+H$
170 PRINT#1,C1$+B$+"ELONGUEE       (5 CPI)"
180 PRINT#1,C1$+E$+B$+"ELONGUEE-GRAS (5 CPI)"+F$
190 PRINT#1,C1$+G$+B$+"ELONGUEE ET    (5 CPI)"
200 PRINT#1,C1$+G$"         DOUBLE PASSAGE"
210 PRINT#1,C1$+E$+B$+"ELONGUEE-GRAS (5 CPI)"
220 PRINT#1,C1$+E$+G$"        ET DOUBLE PASSAGE"+F$+H$                    ex
230 PRINT#1,M$+B$+"ELITE   96 caractères par lignes                  (12 CPI)"
240 PRINT#1,G$+B$+"ELITE DOUBLE PASSAGE                              (12 CPI)"+
H$
250 PRINT#1,C1$+B$+"ELITE ELONGUEE      (6 CPI)"
260 PRINT#1,C1$+B$+"ELITE ELONGUEE ET   (6 CPI)"
270 PRINT#1,C1$+"       DOUBLE PASSAGE"+H$+P$
280 PRINT#1,C2$+B$+"COMPRESSEE 137 caractères par lignes
              (17 CPI)"
290 PRINT#1,G$+B$+"COMPRESSEE ET DOUBLE PASSAGE
              (17 CPI)"+H$
300 PRINT#1,C1$+B$+"COMPRESSEE ELONGUEE              (8.5 CPI)"
310 PRINT#1,C1$+G$+B$+"COMPRESSEE ELONGUEE ET          (8.5 CPI)"
320 PRINT#1,C1$+"       DOUBLE PASSAGE"+H$+D2$
330 PRINT#1,T1$+B$+"SOULIGNE       TECHNOLOGY RESOURCES S.A."+T2$
340 PRINT#1,B$+S0$+"EXPOSANT "+S1$+E$+"INDICE "+F$+S0$+M$+"EN ELITE ";
350 PRINT#1,S1$+P$+C2$+"COMPRESSEE"+S0$+D2$+C1$+"-ELONGUEE"+S$
360 PRINT#1,"     Y=AX"+S0$+"2"+S$+"+BX+C  avec  si B"+S0$+"2"+S$+"-4AC=0     X";
370 PRINT#1,S1$+"1"+S$+"=X"+S1$+"2"+S$+"=-B/2A"
380 PRINT#1,B$+"DE"+E$+G$+" PLUSIEURS MODES "+H$+F$+M$+"D'IMPRESSIONS "+P$;
390 PRINT#1,T1$+C2$+"SUR UNE "+C1$+"MEME "+D2$+"LIGNE"+T2$+CHR$(10)+CHR$(10)
```

```
* ECRITURE NORMALE 80 caractères par lignes          (10 CPI)
* ECRITURE EN GRAS points doublés horizontalement    (10 CPI)
* ECRITURE DOUBLE PASSAGE décalés verticalement      (10 CPI)
* ECRITURE GRAS ET DOUBLE PASSAGE                    (10 CPI)
*  ECRITURE  ELONGUEE                                (5  CPI)
*  ECRITURE  ELONGUEE-GRAS                           (5  CPI)
*  ECRITURE  ELONGUEE  ET                            (5  CPI)
          DOUBLE  PASSAGE
*  ECRITURE  ELONGUEE-GRAS                           (5  CPI)
          ET  DOUBLE  PASSAGE
* ECRITURE ELITE   96 caractères par lignes          (12 CPI)
* ECRITURE ELITE DOUBLE PASSAGE                      (12 CPI)
*  ECRITURE  ELITE  ELONGUEE                         (6  CPI)
*  ECRITURE  ELITE  ELONGUEE  ET                     (6  CPI)
          DOUBLE  PASSAGE
* ECRITURE COMPRESSEE 137 caractères par lignes      (17 CPI)
* ECRITURE COMPRESSEE ET DOUBLE PASSAGE              (17 CPI)
* ECRITURE COMPRESSEE ELONGUEE                       (8.5 CPI)
* ECRITURE COMPRESSEE ELONGUEE ET                    (8.5 CPI)
       DOUBLE PASSAGE
* ECRITURE SOULIGNE        TECHNOLOGY RESOURCES S.A.
* ECRITURE EXPOSANT INDICE EN ELITE COMPRESSEE -ELONGUEE
     Y=AX²+BX+C  avec  si B²-4AC=0     X₁=X₂=-B/2A
* ECRITURE DE PLUSIEURS MODES D'IMPRESSIONS SUR UNE MEME LIGNE
```

L'intelligence artificielle

Dans l'article qui suit, B. Meltzer, qui a travaillé sur un projet d'intelligence artificielle, présente quelques réflexions sur la similitude qu'on peut observer entre les processus cognitifs caractéristiques de l'esprit humain et les possibilités de l'ordinateur. Tout programme est en effet caractérisé par un certain degré de « savoir », dans la mesure où il rassemble une plus ou moins grande quantité de « cognitions » (connaissances) articulées, en fonction de la complexité des opérations qu'il doit exécuter.

L'énorme puissance de calcul de la dernière génération d'ordinateurs a permis de mettre au point des programmes extrêmement complexes et de plus en plus « cognitifs ».

On parle même d'**intelligence artificielle** pour les plus performants.

En intégrant son savoir dans ses programmes (rappelons que le savoir n'est pas seulement la possession de connaissances mais aussi la maîtrise du raisonnement), le programmeur y transfère, sous une représentation symbolique, une part de plus en plus importante de sa pensée.

En d'autres termes, on retrouve, dans les programmes, le matériel de la **psychologie cognitive**, sous une forme plus ou moins explicite.

Nombreux sont les psychologues, philosophes et autres chercheurs qui soutiennent aujourd'hui que, pour toute une série de raisons, il est impossible que des programmes informatiques représentent des structures mentales comparables à celles de l'esprit humain.

Je voudrais contredire ici quelques-unes de leurs objections.

Celle que l'on entend le plus fréquemment est que le fonctionnement du cerveau humain repose sur un support biologique de neurones qui n'ont rien à voir avec les transistors, les microcircuits, les films magnétiques ou autres éléments de l'ordinateur, et que, de ce fait, ses processus cognitifs ne peuvent qu'être totalement différents de ceux des ordinateurs.

Or, il me semble que bien peu d'informaticiens sont sensibles à cet argument dans la mesure où il est évident qu'un même programme tournant sur deux machines de structure différente effectue pourtant la même tâche de façon comparable.

On se heurte souvent à une autre objection concernant les logiciels proprement dits. Elle se fonde sur l'idée que les programmes cognitifs utilisent la vitesse de calcul de l'ordinateur et non l'intuition, l'imagination et l'inspiration qui sont celles du cerveau humain. Or, les exemples qui suivent sont la preuve éclatante que cette opinion est très éloignée de la réalité.

Il existe une profusion de programmes de jeux qui permettent aux hommes de se mesurer, avec plus ou moins de succès, à l'ordinateur. C'est ainsi qu'aux Etats-Unis, le tournoi d'échecs open du Minnesota a été remporté par le programme 4.5 mis au point par Slate et Atkin. Les programmes de jeu d'échecs les plus performants ne pratiquent pas l'analyse systématique de toutes les combinaisons possibles. Ils étudient, au contraire, certains enchaînements de coups, exactement comme le feraient des joueurs, chevronnés ou non.

Toutefois, l'exemple du backgammon (ou jacquet) est peut-être plus intéressant. En juillet 1979, à Montecarlo, l'italien Luigi Villa, champion du monde de backgammon, fut battu 7 contre 1 par le programme d'Hans Berliner, BKG 9.8 (Backgammon 9.8). Chercheur en intelligence artificielle à la Carnegie-Mellon University de Pittsburg, Berliner avait rédigé ce programme en développant ingénieusement le principe de l'un des premiers jeux informatisés, le jeu d'échecs de Samuel : les coups sont décidés au terme d'une analyse globale de la situation, à partir d'un nombre restreint d'éléments caractéristiques du jeu. Berliner a étudié le déroulement de son programme avec Villa et en a tiré les conclusions suivantes : « BKG 9.8 a incontestablement bien joué. La technique de Villa a été irréprochable pendant presque toute la partie alors que 8 fois sur 73, le programme n'a pas trouvé la meilleure réplique. BKG 9.8 n'a d'ailleurs été mis en difficulté que par l'une de ces erreurs, et, surtout, il s'est montré particulièrement brillant quand il s'est agi de faire preuve d'imagination. Et pourtant, sans même savoir ce que sont pions blancs et pions noirs, on aurait sans doute attribué à l'homme le jeu le plus sophistiqué par rapport à la machine ».

S'il est vrai que le langage est la « fenêtre de

l'esprit », de nombreux chercheurs ont tenté de mettre au point des programmes pour essayer de faire comprendre, traduire et résumer des textes en langage naturel par l'ordinateur. Voici un exemple qui permet d'apprécier les résultats obtenus jusqu'ici. Il s'agit d'un programme qui déchiffre les articles d'un journal, retient ce qui l'intéresse et rédige un bref résumé des faits dans la langue désirée (anglais, russe ou espagnol, par exemple).

Baptisé FRUMP (Fast Reading and Understanding Memory Program), ce programme a été conçu par l'unité de recherche sur la compréhension du langage naturel dirigée par Roger Schank à la Yale University.

L'article suivant, extrait d'un quotidien américain, a été communiqué à FRUMP : « Un violent tremblement de terre a secoué hier soir l'Italie septentrionale, détruisant des quartiers entiers de villes situées au nord-est de Venise, non loin de la frontière yougoslave ; d'après un communiqué du ministère de l'Intérieur italien, on compte au moins 95 morts et 1 000 blessés. Selon un porte-parole du gouvernement, on craint qu'il n'y ait au moins 200 morts enterrés sous les décombres, dans la seule ville d'Udine. Située sur le trajet de la principale liaison ferroviaire Rome-Vienne, la population de cette ville est de 90 000 habitants environ. Selon les gendarmes italiens, de graves dégâts se seraient produits dans une demi-douzaine de villes situées au pied des Alpes et des familles entières seraient ensevelies sous leurs maisons écroulées.

Dans de nombreuses localités, les communications n'ont pas encore pu être rétablies. L'intensité du séisme était de 6,3 sur l'échelle de Richter. Dans les régions habitées, une secousse de force 4 provoque généralement des dégâts modérés. La force 6 correspond à un grave séisme et 7 à une véritable catastrophe naturelle ».

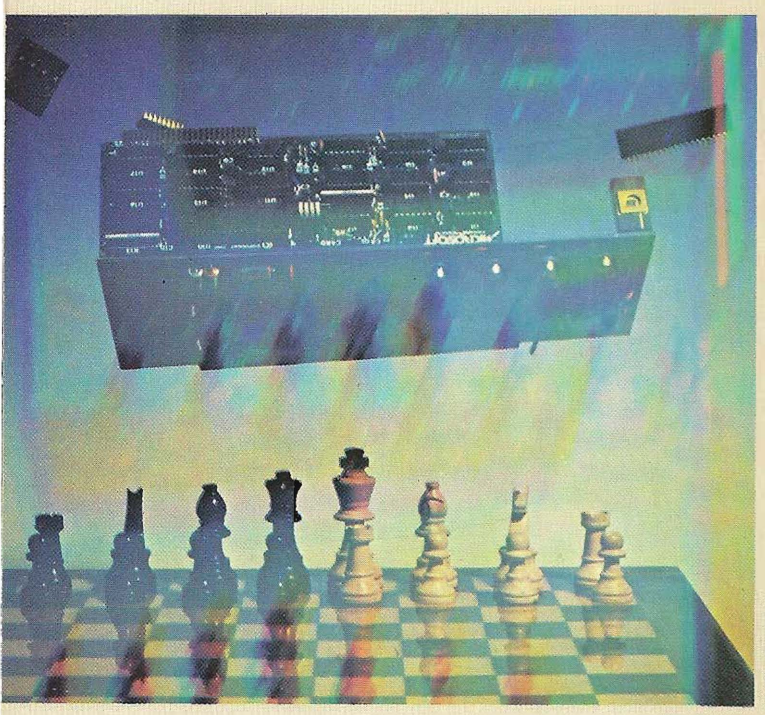
Voici le résumé que FRUMP a été capable de rédiger : « 95 personnes tuées et 1 000 blessées dans un tremblement de terre qui a frappé l'Italie. La secousse enregistrée est de force 6.3 sur l'échelle de Richter ».

L'équipe de Schank construit ses programmes autour de notions de la vie quotidienne. Nous allons maintenant parler d'un programme qui, au contraire, met en œuvre des concepts mathématiques et en découvre de nouveaux, sur lesquels il construit des « raisonnements » plus ou moins pertinents. Ce programme, appelé AM, a été écrit par un jeune américain, Douglas Lenat, qui l'a présenté lors de sa thèse soutenue en 1976 à l'Université de Stanford. Lenat s'est surtout attaché à rechercher des concepts et des modèles mathématiques dignes d'intérêt. En effet, AM n'est pas conçu pour démontrer des théorèmes, mais pour exécuter une série de tâches par ordre d'intérêt, en les abandonnant dès que la puissance de calcul requise se révèle insuffisante.

AM ne « connaissait » pas les notions de base sur les entiers naturels ; il a pourtant redécouvert tout seul de nombreux concepts, même si les notions de fractions et de nombres réels lui ont échappé.

Au départ, la base des connaissances d'AM se composait environ de 100 concepts et de 250 règles heuristiques qui lui permettaient de déterminer la démarche à entreprendre, en fonction du contexte. Elle incluait notamment les notions d'ensemble, de liste (AM est rédigé dans le langage LISP, particulièrement adapté au traitement des listes), de table de vérité, de relations d'appartenance et d'égalité, d'intersection, d'inversion et de composition. Lenat s'appuyait sur les travaux de psychologie infantile de Piaget et de Copeland pour affirmer que ce « savoir » de base correspondait aux connaissances d'un enfant de quatre ans. On peut discuter cette idée ; toujours est-il que le programme trouva rapidement, de façon empirique, qu'il n'existe pas de nombre sans diviseur et qu'un seul n'en possède qu'un. Il découvrit également que certains nombres n'admettent que deux diviseurs, eux-mêmes et l'unité. Il ajouta ainsi à sa base de départ un nouveau concept, celui de nombre premier.

Il faut également signaler que les recherches d'AM aboutirent à la découverte d'un nouveau concept dans la théorie des nombres, inconnu de Lenat lui-même mais qu'un jeune Indien de génie nommé Ramajuan avait étudié au début du siècle. AM découvrit les « nombres les plus composés », qui sont en quelque sorte le contraire des nombres premiers puisqu'ils admettent plus de diviseurs que les nombres qui leur sont inférieurs. Les premiers nombres de ce type sont 1 qui a un diviseur, 2 qui en a deux, 4 et 6 qui sont respectivement les plus petits nombres à avoir



Mallio Falcioni

Les programmes de jeux d'échecs sont parmi les applications les plus révélatrices de l'intelligence de l'ordinateur.

trois et quatre diviseurs, et ainsi de suite. En outre, AM parvint à redémontrer le théorème de la décomposition unique selon lequel tout entier naturel non premier se décompose de façon unique en un produit de facteurs premiers. Enfin, il découvrit l'énoncé de Goldbach qui dit que tout nombre pair est la somme de deux nombres premiers, et que l'on n'a pas encore pu démontrer.

A côté de ces succès remarquables, AM a essuyé des échecs qui sont tout aussi intéressants. J'ai déjà mentionné le fait qu'il fut incapable de découvrir les fractions. Il ne fit pas non plus de grands progrès dans la théorie des ensembles, excepté la découverte des relations les plus simples comme les lois de De Morgan. Il suivit souvent de mauvaises pistes en s'attardant, par exemple, sur l'idée de nombres décomposables de façon unique en une somme de deux nombres premiers. Lenat estimait dans sa thèse que 25 des concepts découverts par AM étaient justes (« bons »), 100 acceptables et les 60 autres erronés (« mauvais »).

Ces trois programme (jeu de backgammon, compréhension du langage naturel et découverte de propriétés mathématiques) ne sont que trois exemples assez récents de ce qu'on peut faire avec un ordinateur. Je les ai présentés pour étayer mon affirmation selon laquelle les procédures mises en œuvre dans les programmes les plus évolués ne sont pas purement « mécaniques », au mauvais sens du terme, mais reproduisent, dans une certaine mesure, les qualités de créativité, d'intuition et d'intelligence caractéristiques de l'esprit humain. On trouverait sans peine d'autres exemples tout aussi convaincants dans les domaines du diagnostic médical, de la perception visuelle, des tests de QI, etc.

L'informatique nous a habitués à utiliser des langages dont le but n'est pas de communiquer des informations mais de les représenter et de les traiter. C'est le cas des langages d'assemblage et des langages machine. D'après une analyse audacieuse proposée il y a peu de temps par le philosophe Aaron Sloman, ce serait même là la fonction première du langage, la fonction de communication n'étant qu'un « sous-produit ». Cela revient à dire que les premiers hommes auraient déjà possédé des langages « intérieurs » de communication et de représentation, qui auraient évolué en signes vocaux et imagés.

Pourquoi alors – si l'on admet l'existence de langages intérieurs se réduisant à une activité neuronique – ne pas établir une nette distinction entre ces langages intérieurs et les langages que nous parlons, comme le français ou l'anglais, et que j'appellerai, pour l'instant, les langages « extérieurs » ? Je crois cependant que le problème gagne en clarté à être abordé comme un seul phénomène.

Le processus de l'apprentissage humain – notamment lorsqu'il implique le maniement de symboles assez formels – constitue un solide argument en faveur d'une étroite continuité entre les deux types de langage. Prenons l'exemple des changements qui interviennent chez les enfants quand ils commencent à apprendre l'arithmétique. Lorsque leur maître leur explique l'addition pour la première fois, ils l'exécutent en suivant ses indications et en utilisant consciemment des mots et des phrases du langage naturel comme « je retiens » ou « 3 plus 3 égal 6 ». Mais quand ils savent faire les additions, ils les effectuent automatiquement, sans réfléchir, et il ne subsiste pratiquement plus rien dans

leur conscience. Pourtant, même au stade inconscient, il est évident que cette activité implique toujours une élaboration symbolique : simplement celle-ci passe désormais par le langage intérieur.

Il faut souligner combien la transition entre les deux langages se fait aisément et « naturellement ». Ainsi, face à une addition particulièrement longue et difficile, il nous arrive, alors même que nous avons bien intégré les mécanismes de calcul, d'effectuer la transition inverse et de revenir au langage extérieur en nous disant clairement : « il ne faut pas que j'oublie d'ajouter la retenue au total de cette colonne ».

Cela m'amène à formuler l'hypothèse suivante : « les processus mentaux de tout système biologique ne sont que des transformations qui s'opèrent au sein du langage propre à ce système, langage qui peut être sollicité à n'importe quel niveau ».

Cette notion de niveau nous est familière pour ce qui est des systèmes de traitement de l'information. Un programme source écrit dans un langage évolué ne peut être exécuté qu'après avoir été traduit d'abord en langage d'assemblage, puis en langage machine. Cette structure hiérarchisée constitue le schéma normal d'exécution des programmes. Toutefois, des variations peuvent modifier ce schéma.

Ainsi, sur certains systèmes, on a la possibilité d'insérer des passages en langage machine dans un programme écrit en langage évolué, afin de le rendre plus performant. A partir du moment où l'on fait ce genre de choses, la notion des différents niveaux de langage de la machine commence à s'estomper et il vaut mieux alors considérer que le programme est écrit dans un unique langage, constitué du langage source, de l'assembleur, du code machine ainsi que de toutes leurs variantes.

On peut faire un rapprochement entre ce langage et le langage unique que j'ai imaginé pour les êtres vivants.

J'ai évoqué les différents « niveaux » de mise en œuvre de ce système linguistique complexe : il faut bien trouver une explication au fait que nous puissions imaginer des objets sans les voir, rien qu'en les entendant nommer. Cela devient possible si l'on admet que le fonctionnement de nos différentes facultés cognitives repose sur un langage commun et qu'on peut donc accéder aux mêmes représentations en empruntant différents parcours linguistiques.

La poésie constitue un cas particulièrement intéressant car, tout en utilisant exclusivement le langage naturel, elle semble chercher à atteindre des couches plus profondes de la partie intérieure du langage. Elle éveille des résonances que l'usage « normal » du langage ne semble pas en mesure de susciter. Selon moi, la poésie s'emploie à recréer, à refléter et à représenter, par le langage naturel, des aspects extrêmement signifiants de notre langage intérieur.

La musique n'utilise pas non plus le discours habituel. Elle n'agit pas par son canal mais s'adresse plus directement à notre langage intérieur, ce qui lui confère une puissance d'évocation plus forte.

Si l'on considère les arts figuratifs, mon hypothèse semble impliquer que nos ancêtres reproduisirent sur les parois de leurs cavernes certains éléments de leur représentation intérieure de l'univers. Peut-être est-ce là ce qu'ont fait, après eux, tous les artistes ? A cet égard, il est révélateur que, dans ses travaux sur les modèles mathématiques de la perception visuelle des objets, Davis Marr ait défini, parmi les différents stades d'élaboration, celui des « stick figures », dont il désigne le graphisme caractéristique de l'une des périodes de Picasso.

Si, comme nous l'avons vu, les méthodes employées en intelligence artificielle sont bien adaptées à la représentation d'objectifs, de suppositions et d'interprétations, il ne devrait pas être très difficile de reconstituer des modèles satisfaisants pour les émotions. Le travail de pionnier entrepris par Freud sur les processus inconscients qui interviennent dans les rêves, les mots d'esprit et les lapsus ne semble pas avoir eu les répercussions scientifiques qu'on aurait pu attendre.

Les transformations qui interviennent dans les rêves ou dans les jeux de mots pourraient trouver une place, sous une formulation tout à fait rigoureuse, au sein des transformations informatiques déjà connues telles que l'interprétation ou la compilation. On devrait alors pouvoir écrire sans trop de peine un programme qui, à partir d'une certaine somme de connaissances, saurait nous raconter des histoires drôles avec beaucoup d'humour.

Présentation des diagrammes à barres

Lors de la préparation d'un travail quel qu'il soit, scientifique ou de gestion, il est souvent appréciable de disposer d'une représentation graphique des données. Un examen approfondi et des calculs précis s'appuient sur les tableaux de chiffres qui abondent dans rapports et comptes rendus. En revanche, pour visualiser l'aspect général d'une situation ou mettre en relief les traits caractéristiques d'un phénomène, il est préférable de transcrire ces tableaux en graphiques. Nous reviendrons plus en détail sur ces techniques, mais voyons dès à présent comment présenter des histogrammes par l'intermédiaire de l'imprimante.

Un **histogramme** est un type de représentation graphique souvent employé pour illustrer les tableaux simples. Supposons que l'on veuille rendre compte du chiffre d'affaires réalisé par une équipe de vendeurs. On pourra reporter dans un tableau le nombre d'articles vendus par chacun d'eux :

Vendeur	Quantité vendue
A	12
B	7
C	9
D	10
E	2
F	8

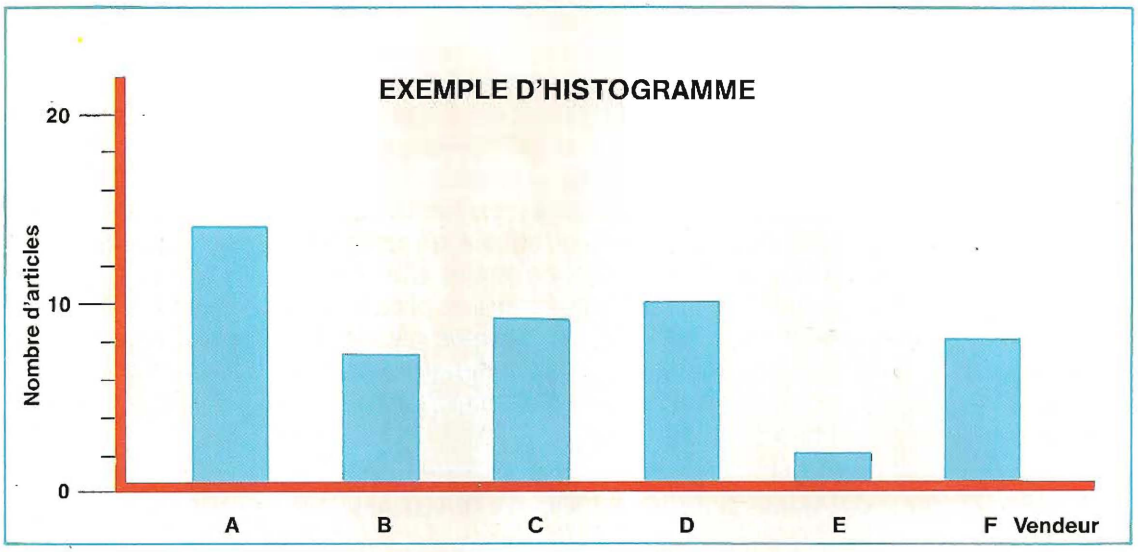
La représentation graphique du haut de la page 589 illustre ce tableau, permettant de mieux visualiser la disparité des résultats et l'hétérogénéité du groupe. On a choisi un histogramme car les vendeurs sont en « quantité discrète » (il n'y a pas d'intermédiaires entre les deux valeurs A et B, par exemple), contrairement aux fonctions que l'on peut représenter par une courbe continue.

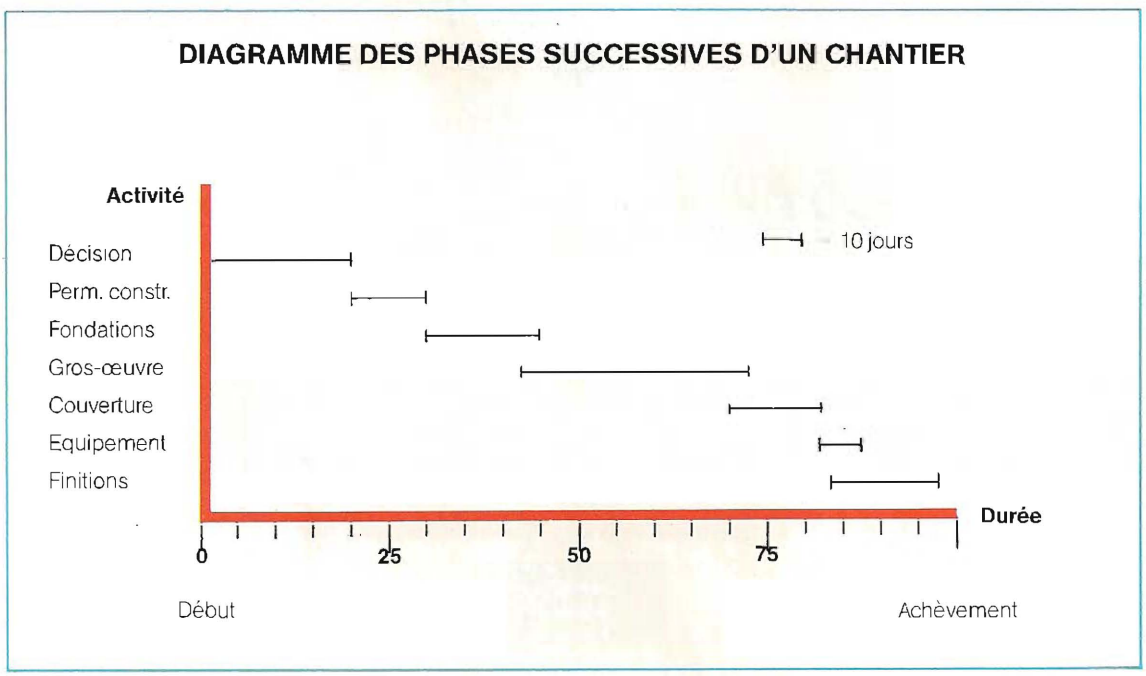
Il est évident que dans un cas comme celui-ci, il est impossible de tracer une courbe. Sauf exception, une courbe est en effet définie pour toutes les valeurs comprises entre deux points. Un histogramme ne concerne, au contraire, qu'un certain nombre de valeurs ponctuelles. Nous allons maintenant développer une application concernant un type particulier d'histogramme : **le diagramme à barres.**

Pendant la phase préparatoire d'un projet industriel, on doit rendre compte des temps de réalisation. Le projet le plus simple comporte plusieurs étapes de durée différente,

Imprimante FX 80 Epson.







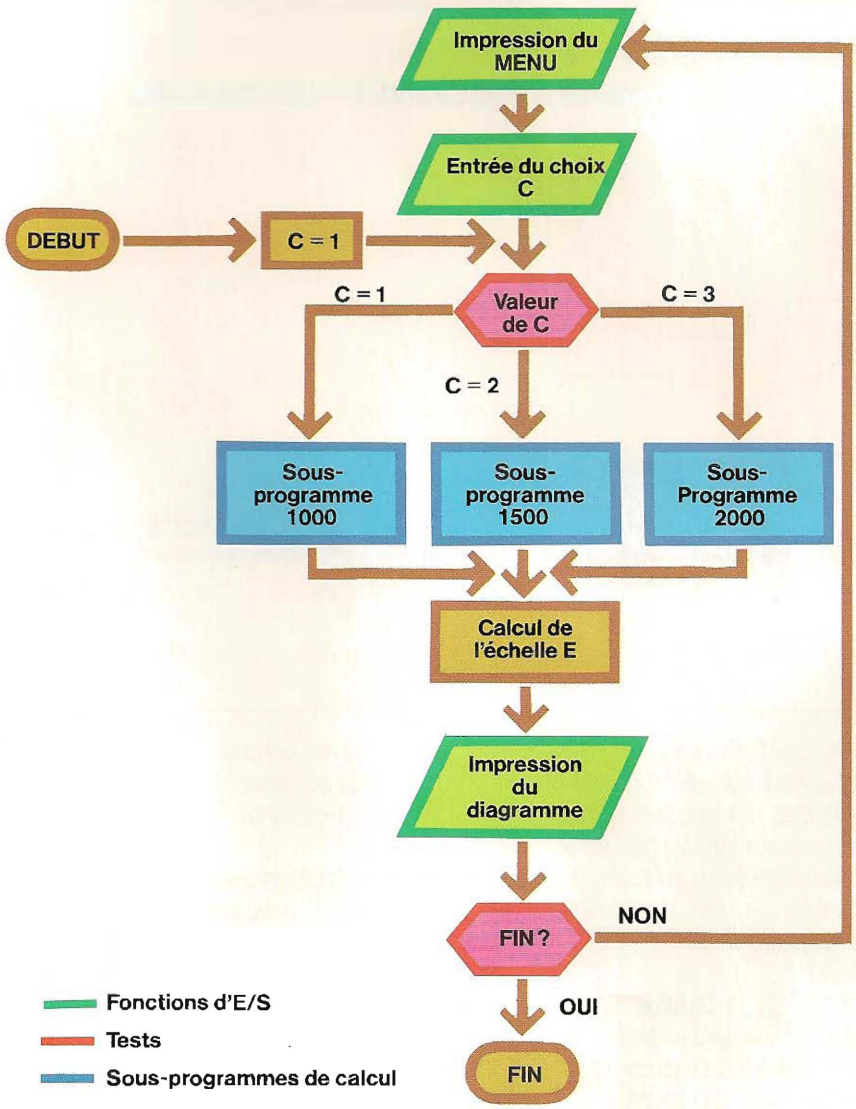
et dont la plupart sont subordonnées aux précédentes. La durée totale de réalisation dépend donc de la succession des différentes étapes, que l'on peut symboliser sur un diagramme. Ainsi, pour la construction d'une maison, on recensera les différentes phases de la réalisation et leur durée respective :

Activité	Durée
Décision	40 jours
Permis de construire	20 jours
Fondations	30 jours
Gros-œuvre	60 jours
Couverture	24 jours
Equipement	16 jours
Finitions	32 jours

En reportant les données de ce tableau sur un diagramme à barres, on obtient le graphique présenté ci-dessus. On remarque que, si certaines phases ne peuvent commencer qu'une fois la précédente terminée, d'autres autorisent un certain chevauchement.
La durée totale de la réalisation n'est donc pas égale au cumul des différentes phases mais à une valeur résultante qui apparaît clairement

sur le schéma de la page 589.
Le recours à de tels diagrammes est assez fréquent dans les programmes d'organisation et de planification disponibles sur les micro-ordinateurs ou les mini-ordinateurs de gestion. Ce type de logiciel permet de prévoir quels seront les besoins en main-d'œuvre à telle ou telle période, ou encore les répercussions sur l'ensemble du projet d'un retard éventuel dans l'une ou l'autre des phases. On peut alors, en simulant diverses causes de retard possibles, évaluer leur retentissement sur la durée des travaux et leurs conséquences économiques.

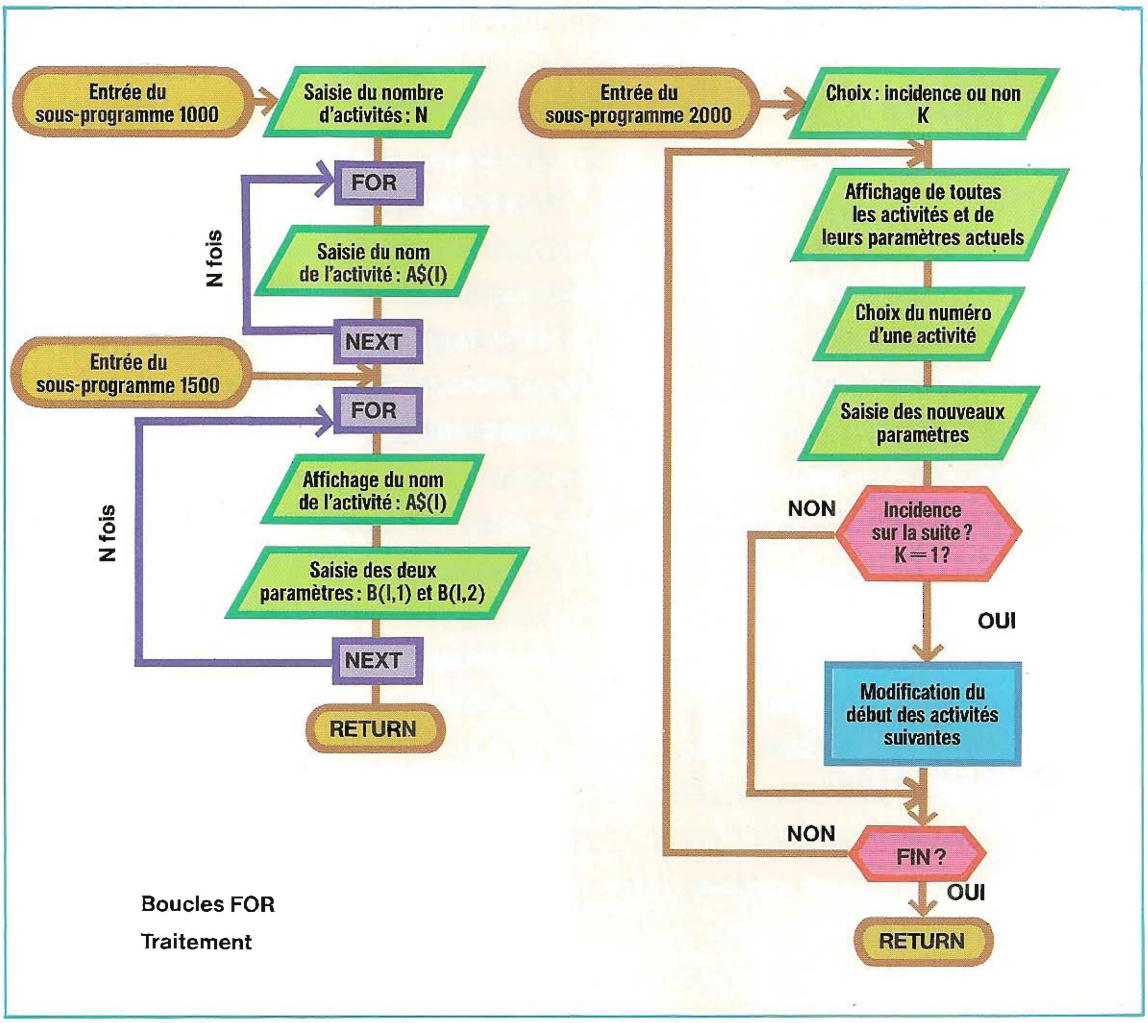
ORGANIGRAMME D'IMPRESSION DE DIAGRAMMES A BARRES



Le MENU est la liste des choix proposés à l'utilisateur. En réponse, on n'a souvent qu'une touche : numéro du choix ou initiale du choix.

Ce test est effectué par l'instruction : ON X GOSUB... qui permet de rajouter des options au menu et d'aiguiller le traitement vers d'autres sous-programmes adaptés.

Le calcul de l'échelle et le tracé du diagramme dépendent de la longueur d'une ligne sur l'imprimante : on a choisi 80 caractères par ligne mais on pourrait très bien augmenter la précision en employant le mode condensé ou une imprimante plus large.



Un programme simple peut permettre d'élaborer des diagrammes à barres et d'obtenir leur tracé par l'intermédiaire de l'imprimante. Voyons comment.

Les diverses activités seront tout d'abord définies puis associées à deux paramètres : leur début et leur durée. Le segment affecté à chacune d'elles sera symbolisé par un certain nombre de caractères unité.

Ce caractère (en l'occurence : *) représentera l'échelle du diagramme en jours, échelle qui sera déterminée de façon à ce que la durée totale du projet tienne dans la largeur de la feuille (quatre-vingts caractères en mode normal d'impression).

De manière à étendre les possibilités de ce programme, on aura recours à un Menu qui proposera à l'utilisateur plusieurs options :

– définition d'une nouvelle série d'activités ;
– conservation des activités précédemment définies, mais modification des paramètres de chacune d'entre elles ;
– modification des paramètres de certaines activités seulement.

Cette dernière possibilité permet de corriger une éventuelle erreur lors de l'entrée des données, mais autorise aussi, si on le désire, une simulation de retard (par exemple). La modification du début ou de la durée d'une des phases est alors répercutée sur toutes les activités ultérieures.

Pour faciliter ces traitements différents selon le choix, les affectations et les calculs sont gérés par des sous-programmes adaptés. Les retours de sous-programme donnent alors lieu à l'impression après calcul de l'échelle E adaptée.

D'autres types de représentations graphiques seront abordés à la fin de l'étude consacrée à la console vidéo.

PROGRAMME DE TRACE DE DIAGRAMMES A BARRES

```
10 ' *** TRACE DE DIAGRAMMES A BARRES
20 '     PROGRAMME : DIAG/BARRES
30 OPTION BASE 1
40 DEFINT A-Z
50 S$="*"
60 I=1 : GOTO 150
70 '
80 ' *** MENU ***
90 PRINT TAB(10);"MENU" : PRINT
100 PRINT " 1 - CHANGEMENT D'ACTIVITES"
110 PRINT " 2 - MODIFICATIONS SUR TOUTES LES ACTIVITES"
120 PRINT " 3 - MODIFICATIONS SUR QUELQUES ACTIVITES"
130 PRINT
140 INPUT "Quel est votre choix ? ";I
150 ON I GOSUB 1000, 1500, 2000
160 '
170 ' CALCUL DE L'ECHELLE ET IMPRESSION
180 E=1
190 IF (B(N,1)+B(N,2) / E > 64 THEN E=E+1 : GOTO 190
200 INPUT "TITRE DU DIAGRAMME : ";T$
210 LPRINT
220 LPRINT T$ : LPRINT
230 FOR I=1 TO N
240 C = ABS(B(I,1)/E)
250 D = ABS(B(I,2)/E)
260 LPRINT A$(I); TAB(15); "!";
270 LPRINT SPACE$(C)
280 LPRINT STRING$(D,S$)
290 NEXT I
300 FOR I=1 TO 80 : LPRINT "-"; : NEXT I
310 LPRINT TAB(16); "Echelle : ";S$;" = ";E
320 LPRINT
330 INPUT "Voulez-vous continuer ? "; REP$
340 IF REP$="OUI" GOTO 90
350 END
1000 '
1010 ' ** SOUS-PROGRAMME 1000 **
1020 INPUT "Nombre d'activités ? ";N
1030 DIM A$(N),B(N,2)
1040 FOR I=1 TO N
1050 PRINT "ACTIVITE "; I;" : (14 caractères maximum)"
1060 INPUT A$(I)
1070 A$(I) = LEFT$(A$,14)
1080 NEXT I
1500 '
1510 ' ENTREE DU SOUS-PROGRAMME 1500
1520 FOR I=1 TO N
1530 PRINT "ACTIVITE "; I; " : "; A$
1540 INPUT "Début des travaux :"; B(I,1)
1550 INPUT "Durée prévue : "; B(I,2)
1560 NEXT I : RETURN
2000 '
2010 ' ** SOUS-PROGRAMME 2000 **
2020 FOR I=1 TO N
2030 PRINT I, A$(I), "DEBUT : "; B(I,1), "DUREE : "; B(I,2)
2040 NEXT I
2050 INPUT "Numéro de l'activité à modifier ? "; J
2060 PRINT A$(J)
2070 INPUT "DEBUT DES TRAVAUX : "; K
2080 INPUT "DUREE PREVUE : "; K
2090 INPUT "Cette modification aura-t-elle une incidence sur la suite ? "; REP$
2100 IF REP$<>"OUI" GOTO 2070
2120 M = K+L-(B(J,1)+B(J,2)
2130 FOR I=J TO N
2140 B(I,1) = B(I,1)+M
2150 NEXT I
2160 B(J,1) = K : B(J,2) = L
2170 INPUT "Désirez-vous modifier une autre activité ? "; REP$
2180 IF REP$="OUI" GOTO 2020
2200 RETURN
```

```
DIAGRAMME A BARRES : ESSAI 1

PROJET          !  **********
REVISIONS       !            *****
CALCULS         !                 ********
TERRASSEMENT    !                         *****
FONDATIONS      !                              **********
MURS            !                                        *****
TOITURE         !                                             **********

          Echelle : * = 2 jours

DIAGRAMME A BARRES : RETARD  EN DEBUT DE CHANTIER

PROJET          !  **********
REVISIONS       !            *****
CALCULS         !                 ********
TERRASSEMENT    !                         ********
FONDATIONS      !                                  **********
MURS            !                                            *****
TOITURE         !                                                  **********

          Echelle : * = 2 jours
```

Test 17

1 / Une imprimante présente les caractéristiques suivantes :
- tête d'impression à aiguilles
- bi-directionnelle
- 132 colonnes
- 120 cps
- buffer 4K

Que signifient ces indications ?

---

2 / Quels caractères seront imprimés si on transmet à l'imprimante les commandes suivantes ?

CHR$(33) CHR$(110) CHR$(27) CHR$(10)

---

3 / Qu'imprime le programme suivant ?

```
10  A$="ESSAI" :B=1270 :C=10
20  PRINT A$,C
30  PRINT USING "####.#";A$;B
```

---

4 / On demande l'édition de la table A(5) à l'aide des instructions suivantes :

```
10  FOR I=1 TO 5 : LPRINT A(I) : NEXT I
20  FOR I=1 TO 5 : LPRINT A(I) ; : NEXT I
30  FOR I=1 TO 5 : LPRINT TAB(10) ; A(I) ; : NEXT I
```

Que va-t-on obtenir pour chacune d'elles ?

Les solutions du test se trouvent page 599.

Gestion de la console vidéo

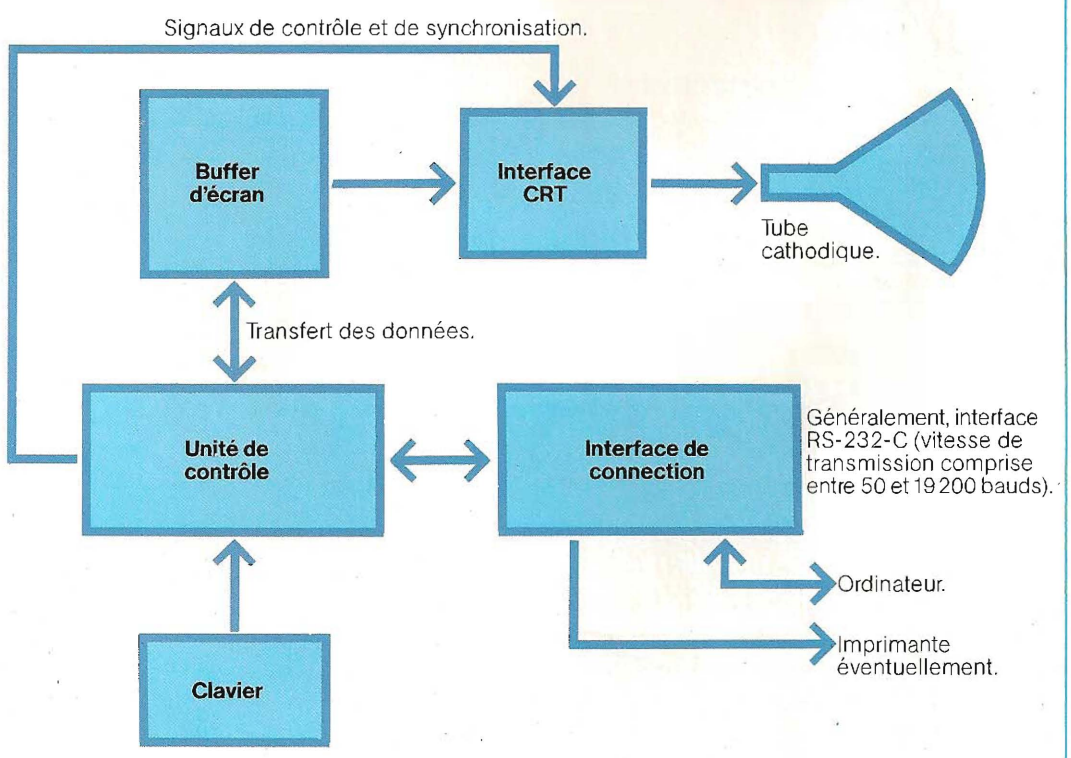
L'évolution des micro-ordinateurs a conduit à la configuration de base actuelle qui adopte la console vidéo (clavier et écran) comme principal organe d'entrée/sortie.
La commodité et la fonctionnalité de ce périphérique sont aujourd'hui universellement reconnues et son domaine d'utilisation s'est même étendu aux terminaux des gros systèmes.
Bien connaître les possibilités de la console vidéo permet d'en tirer meilleur parti et notamment de travailler en mode conversationnel, établissant ainsi un véritable dialogue avec l'unité centrale.
Toutefois, c'est surtout au programmeur qu'il incombe de déterminer le mode de gestion qui exploitera au mieux les capacités du matériel.
En effet, lors des applications spécifiques comme, entre autres, la saisie ou la présentation de données, une bonne maîtrise de la console vidéo constitue toujours un avantage appréciable.

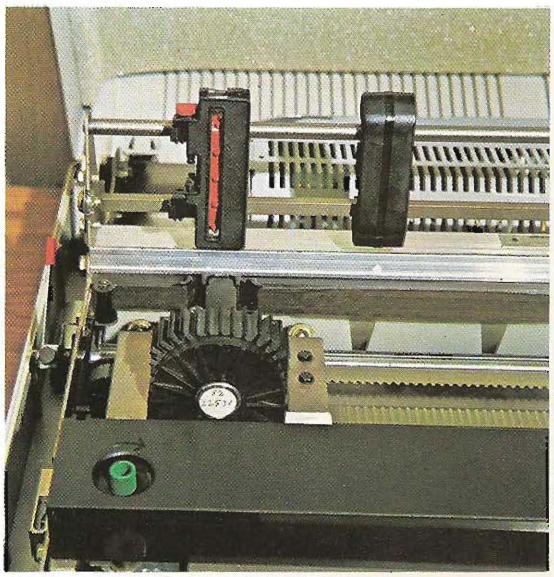
Structure et caractéristiques d'une console vidéo

Le principe de fonctionnement d'une telle console est schématisé au bas de cette page. Nous allons reprendre un à un les principaux éléments de cet ensemble pour décrire brièvement leur utilité.

- **Le clavier** est l'organe d'entrée. A la frappe d'une touche, le code ASCII correspondant est généré et transmis à l'unité de contrôle ;
- **L'unité de contrôle** a un rôle de coordination et de synchronisation des autres organes. Elle assure l'interprétation et l'exécution des commandes, et contrôle les communications avec l'ordinateur ;
- **L'interface de connection** gère le transfert des données de l'ordinateur au périphérique et réciproquement. Certaines consoles sophistiquées disposent d'une interface permettant de les connecter directement à une imprimante. Ce système décharge l'unité centrale et autorise une transcription fidèle du texte affiché : la copie d'écran.

PRINCIPE DE FONCTIONNEMENT D'UNE CONSOLE VIDEO





V. Pirozzi/Archives Curcio

Sous le couvercle d'une imprimante.

- **Le buffer d'écran** est un bloc de mémoire RAM où est stockée, sous forme codée, l'image de ce qui doit apparaître sur l'écran. La capacité de cette mémoire tampon dépend du nombre de caractères affichables. Ainsi, pour un écran de 80 colonnes × 24 lignes, qui peut donc contenir 1920 caractères, le buffer doit correspondre à 1920 emplacements.

C'est volontairement qu'on a employé le terme d'emplacement et non celui d'octet, car, dans ce cas précis, un emplacement n'occupe pas forcément un octet. En effet, pour chaque caractère à afficher, on doit mémoriser non seulement les 8 bits du code ASCII mais éventuellement aussi des informations complémentaires ou **attributs** modifiant les caractéristiques de présentation (vidéo normale ou inverse, clignotement, soulignement, etc.). Cela peut nécessiter jusqu'à 4 bits supplémentaires.
La longueur d'un emplacement de mémoire varie donc d'un écran à l'autre selon les possibilités offertes.
Ces indications ont une valeur générale, mais certains matériels adoptent des caractéristiques très différentes, notamment lors d'utilisation orientée vers le mode graphique.
Ces écrans à vocation particulière feront ultérieurement l'objet d'une présentation spécifique.

- **L'interface CRT** (Cathode Ray Tube) transforme le code de chaque emplacement par l'intermédiaire d'un générateur de caractères. Les signaux vidéo ainsi obtenus sont alors dirigés vers le tube cathodique.

La console autorise plusieurs modes de fonctionnement adaptés aux besoins des différentes applications et basés sur le protocole de transmission des commandes ou données de la console vers l'unité centrale.

CONVERSATIONNEL. Les commandes sont envoyées directement à l'unité centrale dès l'entrée au clavier. Il s'agit le plus souvent d'une transmission asynchrone et en série.
Ce mode comporte deux variantes :

le semi-duplex : liaison bi-directionnelle non simultanée où le terminal exécute les commandes provenant soit de l'unité centrale, soit du clavier ;

le duplex intégral : liaison bi-directionnelle simultanée où les commandes entrées au clavier sont toutes transmises à l'ordinateur avant d'être exécutées. La console est donc sous le contrôle intégral de l'ordinateur.

PAGE. Ce mode très particulier permet de transmettre les données à l'ordinateur non plus une à une ou ligne par ligne, mais page par page. Une page correspond à la totalité de l'écran. Ce mode n'est généralement pas accessible à partir du Basic, ni des autres langages évolués. Il peut cependant être mis en place par des routines spéciales rédigées en langage Assembleur.

SCROLL. Ce mode est très proche du mode PAGE mais n'est pas limité au contenu de l'écran. Alors que, dans le mode précédent, le curseur se bloque en bas et à droite après la frappe du dernier caractère, le SCROLL permet de continuer en faisant défiler les lignes. La première disparaît et tout le texte est décalé vers le haut, libérant une nouvelle ligne en bas de l'écran.

PROTECT. Ce mode permet d'assurer la protection de caractères ou groupes de caractères en leur accordant un attribut particulier. Il sera alors impossible à l'opérateur de modifier ces caractères car le curseur ne s'arrête pas sur eux.

Ces différents modes de fonctionnement sont commutables, à partir du programme ou en direct, par l'emploi de commandes spécifiques. Comme celles destinées à l'imprimante, ces commandes sont normalement introduites par un caractère de contrôle: ESCape, de code ASCII 27. Ce caractère informe le terminal que les caractères qui le suivent doivent être interprétés comme des commandes et non comme des données.

Gestion du curseur

Des terminaux vidéo très sophistiqués vont jusqu'à permettre des modifications de la forme du curseur. Les commandes que nous allons citer en exemple sont celles du terminal Seletron 3410. Bien que leur gamme soit particulièrement fournie, leur mise en œuvre est généralement représentative des solutions les plus classiques. Voici cependant quelques cas particuliers. Par exemple, un curseur rectangulaire constitué d'une matrice 9 × 11. On peut tout d'abord intervenir sur le mode d'affichage de ce curseur et opter pour :
- l'affichage fixe ;
- l'effacement du curseur ;
- le clignotement (un réglage de la fréquence est possible sur certains matériels).

Pour modifier la forme du curseur, on précisera simplement lesquelles des neuf lignes et des onze colonnes de la matrice doivent être visualisées (toutes à l'origine). Les commandes sont transmises sous la forme habituelle :

ESC + "paramètres"

Ainsi, sur cette machine, "c0" est le code de l'affichage fixe et "c3" le code du clignotement rapide. L'instruction prend alors la forme :

```
PRINT CHR$(27)+"c0"
```

Le curseur sera donc fixe et le restera jusqu'à l'envoi d'une commande contraire telle que :

```
PRINT CHR$(27)+"c3"
```

Cette dernière déclenche le clignotement rapide du curseur qui reste actif jusqu'à la modification suivante. .
Un peu plus complexe est l'instruction produisant une modification de la forme du curseur. Elle se présente ainsi :

```
PRINT CHR$(27)+"c4SE"
```

S est le numéro de la première ligne de la matrice du curseur devant être visualisée. E est le numéro équivalent pour la première colonne.
Page 597 se trouve le listing d'un programme de démonstration des diverses possibilités d'affichage. Comme nous l'avons vu, celles-ci sont sélectionnées par des commandes qui sont explicitées dans les lignes 44 à 58 du listing. La ligne 100 définit une fonction qui associe le caractère ESCape et une chaîne F$(N) elle-même définie grâce aux différents codes commande mémorisés en DATA (lignes 140 à 210).
Pour obtenir le mode d'affichage désiré, on donne à N la valeur voulue et on demande l'impression de FNC$(N). Si, par exemple, on affecte 3 à N, on prélèvera la chaîne F$(3), c'est-à-dire "G1", qui est le code du soulignement.
Voici donc les instructions qu'il convient d'utiliser pour qu'un texte apparaisse souligné à l'écran :

```
N = 3
PRINT FNC$(N);"Essai de soulignement"
```

Le programme présente également une fonction de positionnement du curseur qui permet d'écrire le texte à une position précise de l'écran. Soit les instructions suivantes :

```
10  A$="Essai"
20  X=5 :Y=10
30  PRINT FNP$(X,Y);A$
```

Elles déterminent l'affichage de la chaîne "Essai" à partir de la 5e colonne (X=5) de la 10e ligne (Y=10).

Les masques de saisie

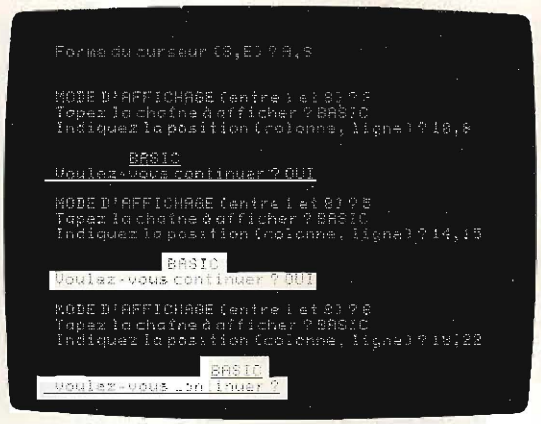
Ces nombreuses possibilités de contrôle de l'affichage sont à l'origine des « masques de saisie ».
Ces masques confèrent une grande souplesse aux programmes d'acquisition de

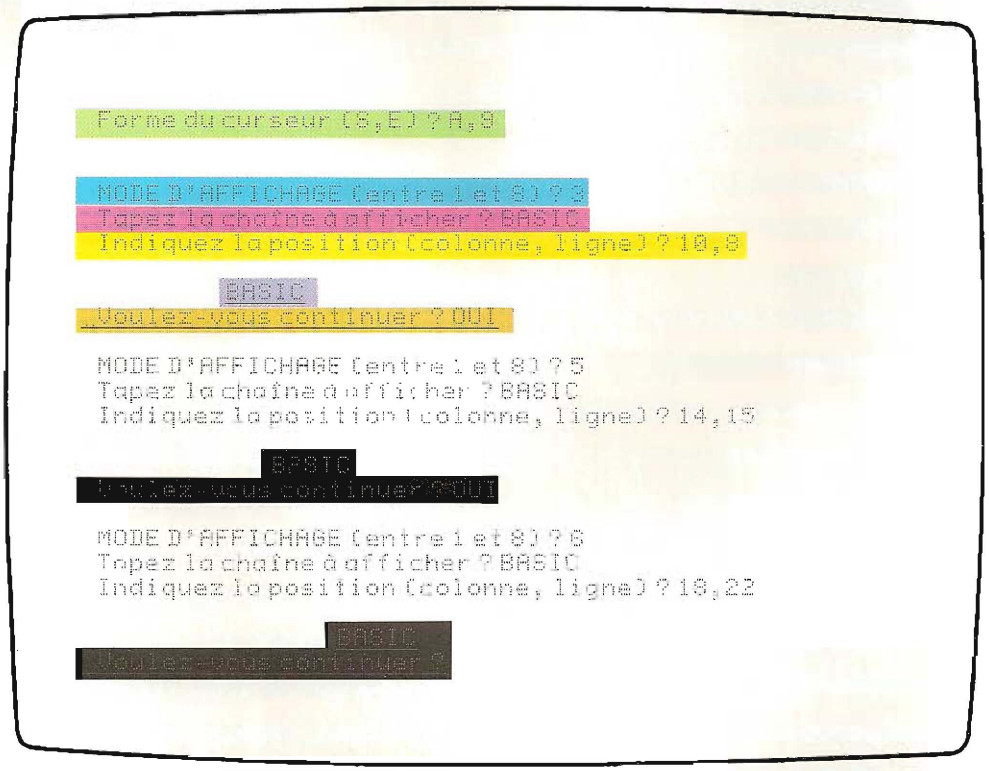
EXEMPLES DE GESTION DE L'AFFICHAGE

```
10 ' *** Exemples de gestion de l'affichage
20 '     PROGRAMME : AFFICH.DEMO
25 OPTION BASE 1
30 DEFINT A-Z
42 '   COMMANDES                          EFFET
44 '   CHR$(27)+")"          Affichage en demi-intensité
46 '   CHR$(27)+"("          Retour à la luminosité normale
48 '   CHR$(27)+"G1"         Soulignement
50 '   CHR$(27)+"G2"         Clignotement
52 '   CHR$(27)+"G4"         Vidéo inverse (caractères noirs sur blanc)
54 '   CHR$(27)+"G5"         Soulignement et vidéo inverse
56 '   CHR$(27)+"G8"         Clignotement et vidéo inverse
58 '   CHR$(27)+"G0"         Retour à la normale
60 '
70 ' * FORME DU CURSEUR : CHR$(27)+"c4"+"S"+"E"
80 '
90 '
100 DEF FNC$(N)=CHR$(27)+F$(N)    ' Mode d'affichage (lignes 44-58)
110 DEF FNF$(S$,E$)=CHR$(27)+"c4"+S$+E$    ' Forme (ligne 70)
120 ' VALEURS DES CHAINES F$ POUR LESQUELLES LA FONCTION EST DEFINIE
125 RESTORE 140
130 FOR I=1 TO 8 : READ F$(I) : NEXT I
140 DATA")"
150 DATA"("
160 DATA"G1"
170 DATA"G2"
180 DATA"G4"
190 DATA"G5"
200 DATA"G8"
210 DATA"G0"
220 '
230 ' *** POSITIONNEMENT DU CURSEUR : X=COLONNE Y=LIGNE
240 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
250 '
260 ' ** FONCTIONS SPECIALES
270 '
280 BLK$=CHR$(27)+"+"          ' Effacement de l'écran
290 BEL$=CHR$(7)               ' Emission d'un bip
300 '
310 ' ** CHOIX DE LA FORME DU CURSEUR ET DE SON POSITIONNEMENT
320 '
325 X=1:Y=1: PRINT FNP$(X,Y);
330 INPUT "Forme du curseur (S,E) ";S$,E$
360 PRINT FNF$(S$,E$)          ' On donne au curseur la forme voulue
370 '
380 ' * MODE D'AFFICHAGE
390 PRINT FNC$(8)          ' RETOUR A LA NORMALE
400 INPUT " MODE D'AFFICHAGE (entre 1 et 8) ";N    ' Voir les DATA
410 IF N>=1 AND N<=8 THEN GOTO 500    ' Valeur correcte
420 ' ** ERREUR **
430 '
440 X=35:Y=12 PRINT FNP$(X,Y)    ' Curseur au centre de l'écran
450 N=6: PRINT FNC$(N);" ** ERREUR **"
460 FOR I=1 TO 800: NEXT I        ' Boucle de "temporisation"
461 G$=INPUT$(1)
470 GOTO 260
480 '
500 ' ** ENTREE D'UN TEXTE **
510 '
520 INPUT " Tapez la chaîne à afficher ";A$
530 INPUT " Indiquez la position (colonne, ligne) ";X,Y
540 K=LEN(A$)          ' Longueur de la chaîne à afficher
550 M=80-K             ' On vérifie que la chaîne peut tenir sur la ligne
560 IF X<0 OR X>M THEN PRINT " ** ERREUR **": GOTO 520
570 IF Y<0 OR Y>24 THEN PRINT " ** ERREUR **": GOTO 530
580 '
590 ' **** AFFICHAGE ****
610 PRINT FNP$(X,Y);FNC$(N);A$
640 INPUT " Voulez-vous continuer ? ";REP$
650 IF REP$="OUI" THEN GOTO 390
660 END
```

EXEMPLE DE GESTION D'ECRAN

Ces illustrations montrent quelques modes d'affichage obtenus grâce au programme de la page précédente.





A l'exécution de la ligne 330, les paramètres qui définissent la forme du curseur sont demandés à l'opérateur. Celui-ci a tapé les chiffres hexadécimaux A et 9 (10x9), qui donnent au curseur une forme rectangulaire.

A la ligne 400, le programme demande l'entrée du code du mode d'affichage. La valeur entrée (3) correspond au soulignement (ligne 48).

L'instruction 520 demande la frappe de la chaîne à afficher. Ici, la chaîne « BASIC » est affectée à la variable A$.

Le programme demande maintenant à l'opérateur les coordonnées du point de l'écran où devra commencer la chaîne A$ (ligne 530). L'opérateur choisit l'intersection de la colonne 10 et de la ligne 8.

L'instruction PRINT de la ligne 610 est exécutée. Le mot BASIC, entré auparavant par l'opérateur, apparaît souligné sur l'écran.

A la demande du programme (ligne 640), l'opérateur exprime son désir de continuer. La ligne 650 provoque le branchement à la ligne 390.
La suite montre deux autres modalités d'affichage de la même chaîne : d'abord, en vidéo inverse, puis toujours en vidéo inverse mais avec soulignement.

données. Leur emploi représente un excellent moyen de contrôle lors de la saisie. Côté utilisateur, ils permettent de corriger à tout moment les données déjà tapées, ce qui s'avère impossible avec l'instruction INPUT qui provoque l'acquisition immédiate des données par l'ordinateur. L'écriture de programmes ayant recours aux masques de saisie présente incontestablement des difficultés. Nous aborderons donc cette étude progressivement et nous commencerons par l'exemple simple d'un sous-programme d'entrée et de contrôle de la date.

L'émission de la date est indispensable dans de nombreuses applications, telles que le classement d'états ou l'édition de documents comptables. Tout logiciel sérieux se doit de vérifier la cohérence de la date entrée par l'opérateur. Nous nous bornerons, dans notre exemple, à contrôler la conformité du mois et du jour entrés de façon à ce que le dernier ne soit pas supérieur au nombre de jours du mois concerné. L'organigramme de ce sous-programme se trouve page 600.

Il faut prévoir, parallèlement à cette routine, deux tables à une dimension, l'une pour les noms ou sigles des douze mois et l'autre pour les nombres de jours correspondants.

Voici comment s'effectuera alors le contrôle. Avant d'appeler le sous-programme, on aura réparti la date entrée dans trois chaînes de deux caractères : JOUR$, MOIS$ et ANNEE$. On pourra donc, désormais, comparer le contenu de la chaîne MOIS$ aux douze valeurs prévues. S'il n'est égal à aucune d'elles, c'est que le mois aura été mal tapé. Si, au contraire, on trouve MOIS$ dans la table, son numéro permettra de trouver dans la seconde table le nombre de jours qu'il contient. Le contrôle de la validité du jour consistera alors à vérifier, après l'avoir converti en numérique, qu'il est inférieur ou égal au nombre de jours du mois indiqué. Seul le mois de février fait exception. Pour ce mois, on est obligé de regarder si l'année est, ou non, bissextile. Ce contrôle est effectué dans un autre sous-programme.

Solutions du test 17

1 / Sur cette imprimante, les caractères sont formés par une juxtaposition de points parmi ceux d'une matrice déterminée. Ces points sont obtenus par l'impact d'aiguilles réunies dans une tête d'impression qui est active dans les deux sens de son déplacement (aller et retour). Une ligne est constituée de 132 caractères maximum imprimés à une vitesse de 120 par seconde. Une mémoire tampon de 4 K (4 096 octets) permet de libérer plus rapidement l'ordinateur lors des travaux d'impression.

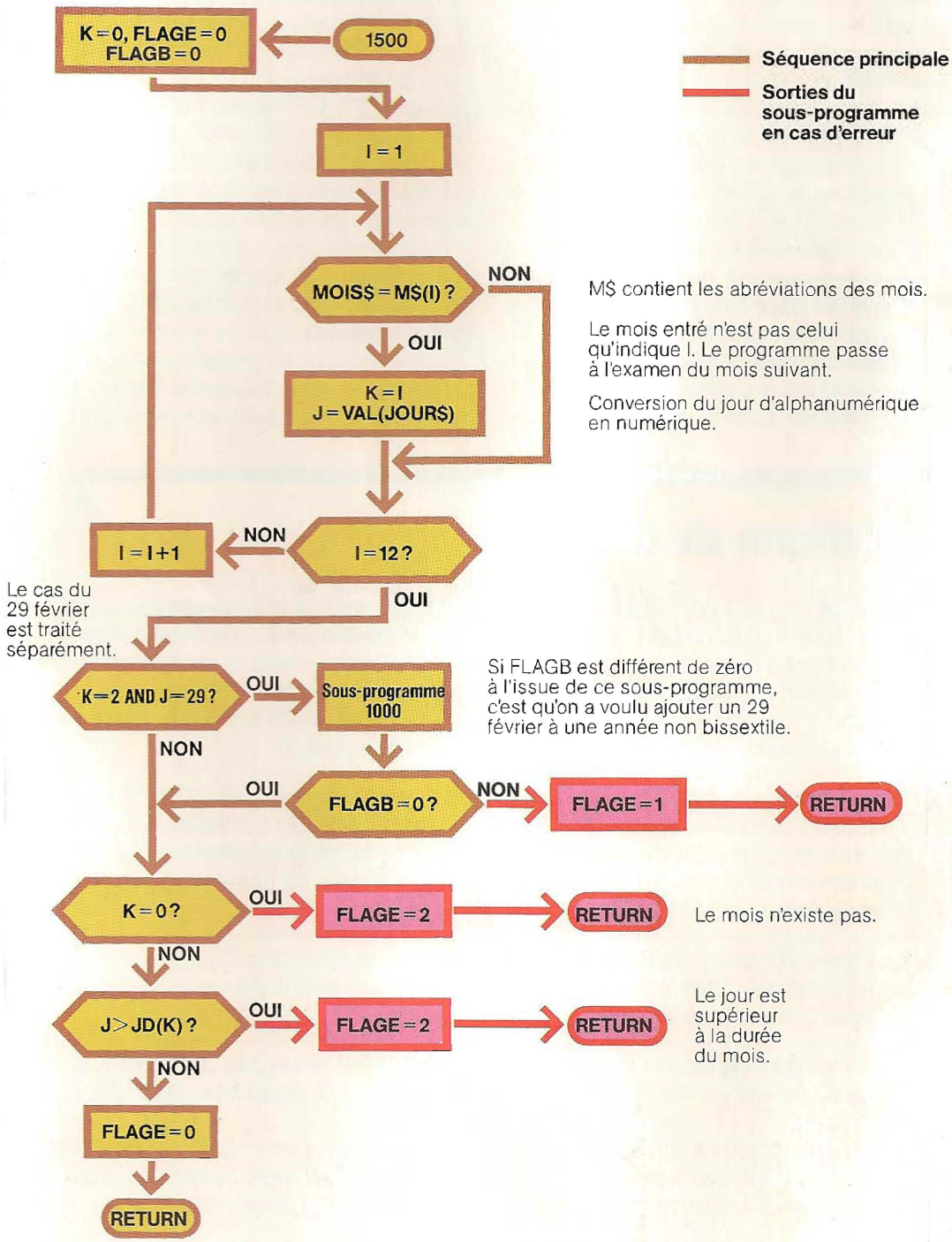
2 / 33 et 110 sont les codes respectifs des caractères ! et n. En revanche, 27 et 10 sont les codes de « caractères spéciaux » ou caractères de contrôle. 27 correspond à ESCAPE. Il avertit un périphérique (imprimante ou terminal vidéo) que les données qui suivent sont des ordres et non des caractères à imprimer. Quant à 10, c'est le code réservé au saut de ligne (LINE FEED).

3 / La ligne 20 provoque l'affichage de la chaîne A$ et de la variable C. L'instruction de la ligne 30 est par contre incorrecte car elle emploie pour l'édition d'une chaîne de caractères un format réservé aux valeurs numériques.

4 / Ligne 10 : les cinq valeurs sont imprimées au début de cinq lignes consécutives.
Ligne 20 : les cinq valeurs sont imprimées à la suite les unes des autres sans séparation (symbole ;).
Ligne 30 : les cinq valeurs seront imprimées à partir de la dixième colonne. Cependant, le ; après A(I) empêche le retour du chariot (passage à la ligne suivante). Les cinq valeurs seraient donc superposées si l'imprimante pouvait revenir en arrière.

SOUS-PROGRAMME DE CONTROLE MOIS-JOUR

Entrées : JOUR$ = chaîne contenant le jour (nombres de 1 à 31) ;
MOIS$ = chaîne contenant le mois (abréviation) ;
ANNEE$ = chaîne contenant l'année (pour les années bissextiles).
Sortie : FLAGE > 0 en cas d'erreur, FLAGE = 0 pour une date correcte.



On y vérifie que l'année est divisible par quatre (voir schéma ci-dessous). N'oublions pas, cependant, que les années séculaires ne sont bissextiles que si elles sont divisibles par quatre cents (1900 ne l'était pas, 2000 le sera). On a prévu, dans ces sous-programmes, plusieurs flags qui indiqueront le type d'erreur éventuellement commis.
Les pages 602 et 603 retracent l'organigramme d'un sous-programme de lecture et de contrôle d'une date entrée au clavier, qui réunit les différentes routines que nous venons de présenter, et illustre un exemple d'acquisition des données à l'aide d'un masque de saisie. L'instruction INPUT constitue le moyen le plus simple et le plus direct de communiquer au programme les données en entrée. Ainsi, l'instruction

INPUT JOUR$, MOIS$, ANNEE$

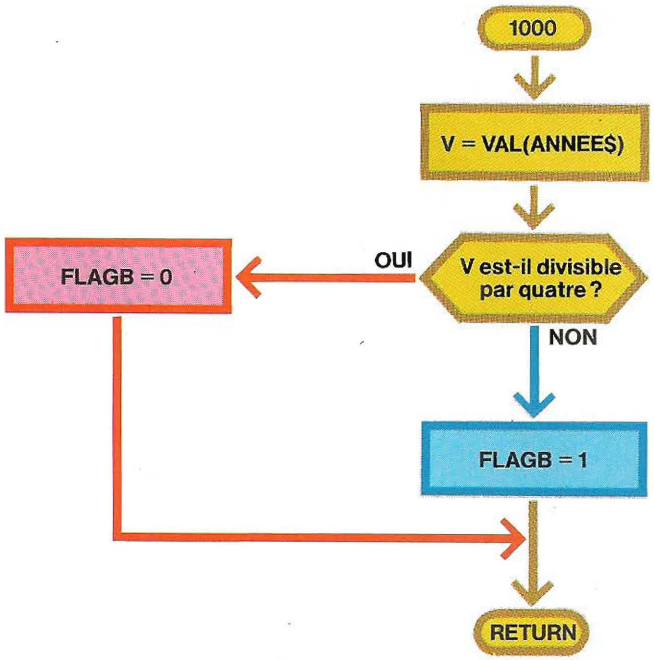
permet l'acquisition des trois éléments (chaînes de caractères) de la date.
Toutefois, ce mode de saisie présente deux inconvénients majeurs. Tout d'abord, rien n'indique à l'utilisateur sous quelle forme il doit taper les données. Il ignorera donc, par exemple, combien de caractères représentent le mois (deux caractères numériques, ou jusqu'à neuf caractères alphabétiques). De plus, la ligne d'affichage de la date est déterminée par le système et l'entrée d'autres données peut la faire disparaître.
Si l'on suit, au contraire, la méthode présentée en page 602 (voir le listing correspondant pages 604 et 605), l'inscription

../../..

apparaîtra à l'écran, avec le curseur placé sous le premier point.
Entrer les données devient alors aussi simple que remplir un formulaire : les points et les séparateurs (symboles /) indiquent à l'utilisateur la longueur de chaque élément. De plus, l'inscription apparaît à l'endroit voulu de l'écran, et y demeure tant qu'une instruction spéciale du programme ne demande pas son

SOUS-PROGRAMME ANNEE BISSEXTILE

Entrée : ANNEE$ = chaîne contenant l'année.
Sortie : FLAGB = 0 si l'année est bissextile, sinon FLAGB = 1.
Calculs : l'année est bissextile si elle est divisible par quatre.



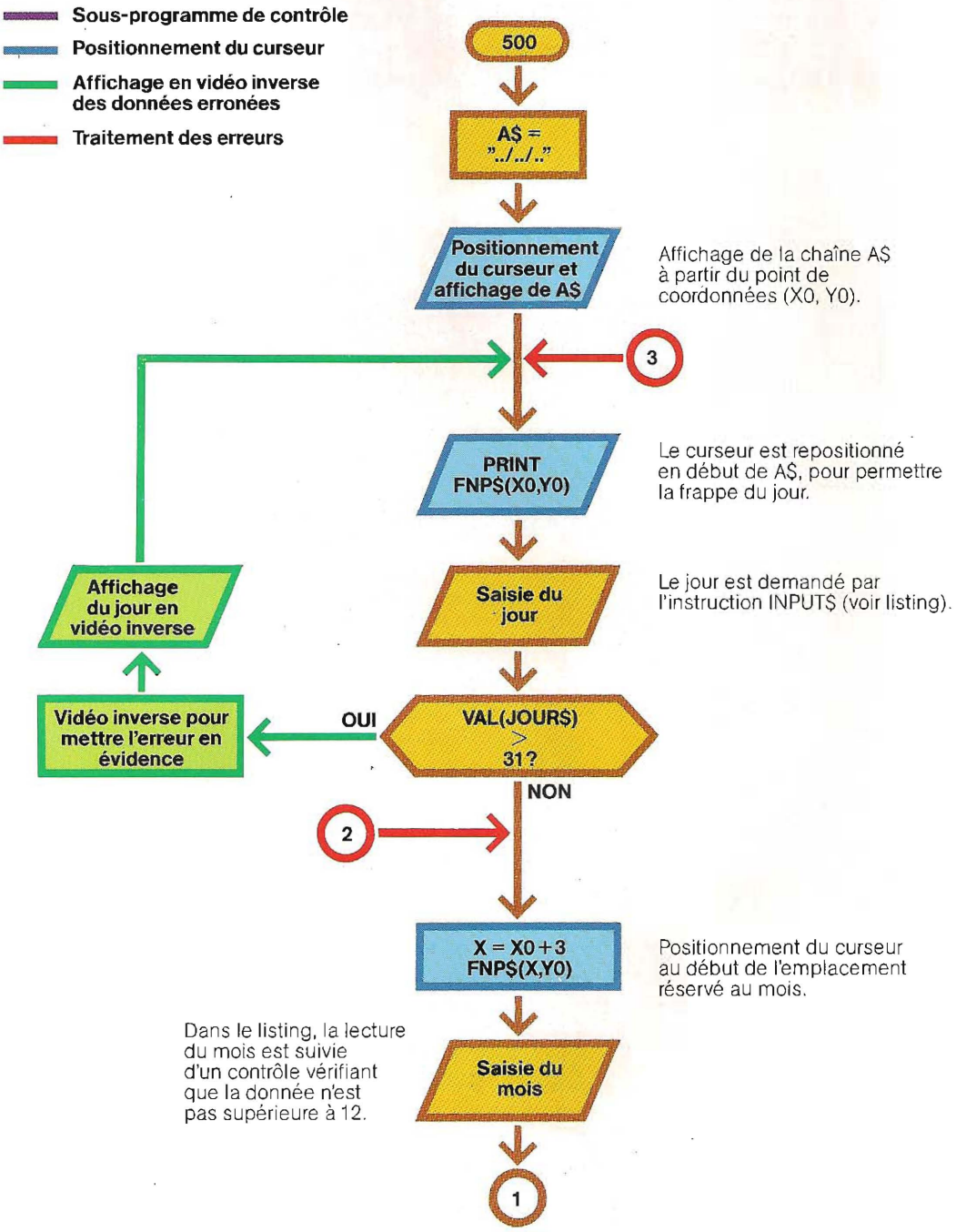
Les années bissextiles sont divisibles par quatre. Toutefois, les années séculaires (1900, 2000, etc.) ne sont bissextiles que si elles sont divisibles par quatre cents.

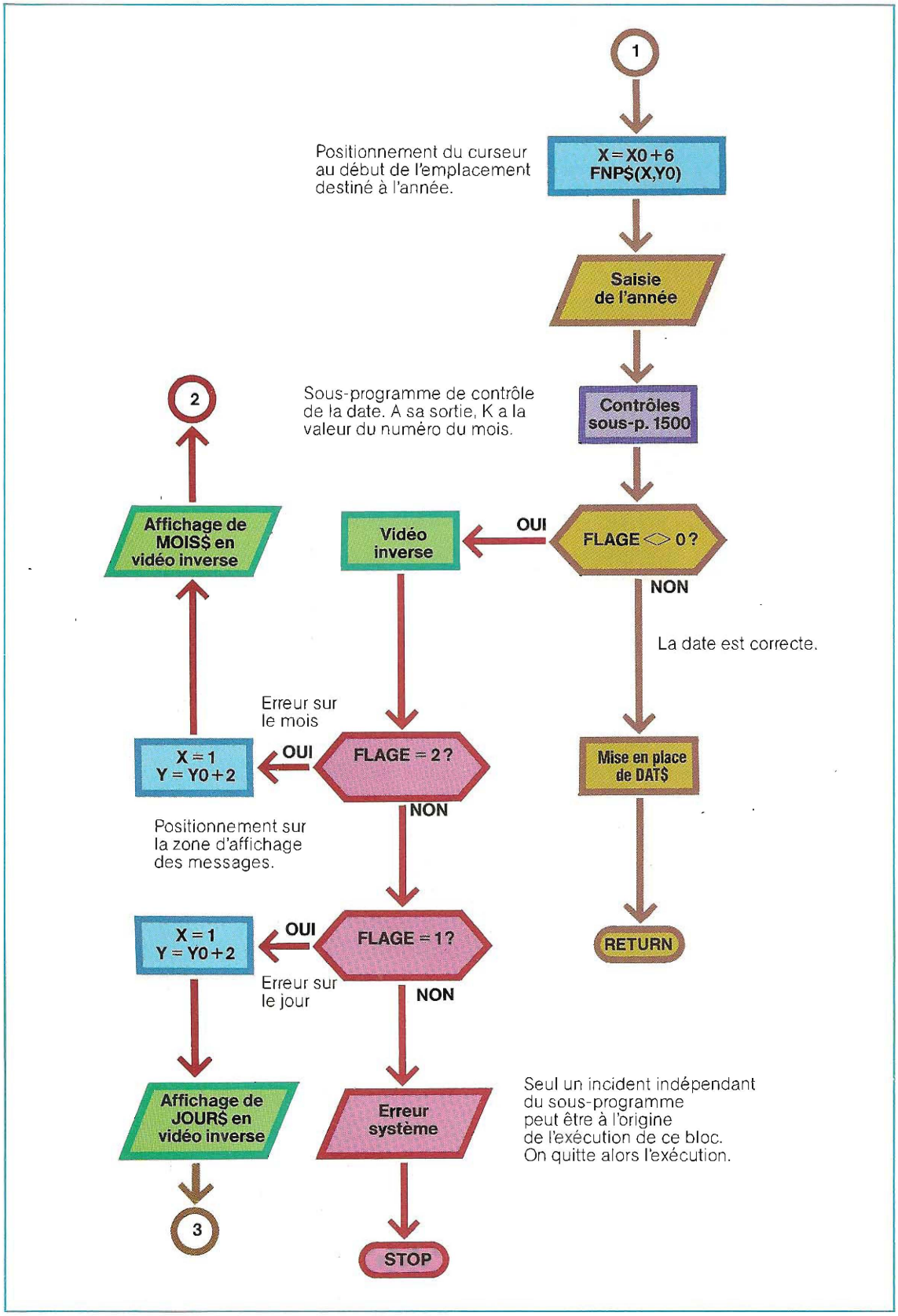
SOUS-PROGRAMME DE SAISIE ET DE CONTROLE DE LA DATE

On utilise, dans ce sous-programme, deux tableaux remplis par des instructions DATA :
M$(12) contient les numéros des mois (1, 2, 3, etc.) ;
NJ(12) mémorise le nombre de jours correspondants (31, 28, 31, etc.) classés dans le même ordre.

Entrées : (X0, Y0) = coordonnées du point de l'écran à partir duquel s'affiche la date.

Sortie : DAT$ = chaîne de caractères contenant le jour, le mois et l'année (tous exprimés par deux chiffres).




1
Positionnement du curseur au début de l'emplacement destiné à l'année.
X=X0+6
FNP$(X,Y0)
Saisie de l'année
Sous-programme de contrôle de la date. A sa sortie, K a la valeur du numéro du mois.
Contrôles sous-p. 1500
2
Affichage de MOIS$ en vidéo inverse
Vidéo inverse
OUI
FLAGE <> 0 ?
NON
La date est correcte.
Erreur sur le mois
X=1
Y=Y0+2
OUI
FLAGE = 2 ?
NON
Mise en place de DAT$
Positionnement sur la zone d'affichage des messages.
X=1
Y=Y0+2
OUI
FLAGE = 1 ?
RETURN
Erreur sur le jour
NON
Affichage de JOUR$ en vidéo inverse
Erreur système
Seul un incident indépendant du sous-programme peut être à l'origine de l'exécution de ce bloc. On quitte alors l'exécution.
3
STOP

PROGRAMME DE SAISIE ET DE CONTROLE DE LA DATE

```
10 ' **** PROGRAMME TEST ****
15 ' FICHIER = DATE
20 DEFINT A-Z
30 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
40 DEF FNN$=CHR$(27)+"G0"
50 DEF FNI$=CHR$(27)+"G4"
60 DIM M$(12),NJ(12)
62 FOR I=1 TO 12 : READ M$(I) : NEXT I
64 DATA"01"
66 DATA"02"
68 DATA"03"
70 DATA"04"
72 DATA"05"
74 DATA"06"
76 DATA"07"
78 DATA"08"
80 DATA"09"
82 DATA"10"
84 DATA"11"
86 DATA"12"
87 X0=5 : Y0=10
88 FOR I=1 TO 12 : READ NJ(I) : NEXT I
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31
91 'PRINT CHR$(27)+"*"          ' EFFACE L'ECRAN
92 GOSUB 500          ' SAISIE DE LA DATE ET CONTROLE
94 PRINT "La date ";DAT$;" est correcte"
100 INPUT "VOULEZ-VOUS CONTINUER (OUI/NON) ";REP$
110 IF REP$="NON" GOTO 120
115 Y0=Y0+2 : GOTO 92
120 END
500 ' * Sous-programme de saisie de la date *
501 X=1 : Y=Y0-2 : PRINT FNP$(X,Y);      ' Positionne le curseur en (1,8)
505 PRINT "TAPEZ LA DATE SOUS LA FORME ../../.."
510 A$="../../.."
520 PRINT FNP$(X0,Y0);A$          ' Affiche A$ à partir de (5,10)
530 PRINT FNN$                    ' Affichage normal
540 PRINT FNP$(X0,Y0);            ' Positionnement du curseur sur le jour : (5,10)
550 C$=INPUT$(1) : PRINT C$;      ' PREMIER CARACTERE
560 D$=INPUT$(1) : PRINT D$;      ' SECOND CARACTERE
570 JOUR$=C$+D$
580 IF VAL(JOUR$)<=31 THEN GOTO 640     ' On peut continuer si le jour n'est pas
590 ' **** ERREUR : JOUR > 31 ****                    supérieur à 31
600 PRINT FNI$                ' Passage en vidéo inverse
610 PRINT FNP$(X0,Y0);        ' Positionnement au début du jour
620 PRINT JOUR$               ' Le jour est réaffiché en vidéo inverse
630 GOTO 530
640 ' LE JOUR EST A PRIORI CORRECT
650 X=X0+3
655 PRINT FNP$(X,Y0);         ' Positionnement sur le mois : (8,10)
660 C$=INPUT$(1) : PRINT C$;
670 D$=INPUT$(1) : PRINT D$;
680 MOIS$=C$+D$ : IF VAL(MOIS$)<=12 THEN GOTO 690
685 PRINT FNI$ : PRINT FNP$(X,Y0);MOIS$ : PRINT FNN$ : GOTO 655    ' Correction
690 X=X0+6 : PRINT FNP$(X,Y0);    ' Positionnement sur l'année : (11,10)
700 C$=INPUT$(1) : PRINT C$;
710 D$=INPUT$(1) : PRINT D$;
720 ANNEE$=C$+D$
730 ' ** CONTROLES **
740 GOSUB 1500
750 IF FLAGE=0 GOTO 960       ' Date correcte
760 '
770 ' **** ERREURS ****
780 PRINT FNI$
790 IF FLAGE<>2 GOTO 840
795 '
```

```
800 ' ** ERREUR SUR LA VALEUR DU MOIS **
810 X=1 : Y=Y0+2
820 PRINT FNP$(X,Y); : PRINT MOIS$ : PRINT FNN$ : GOTO 650    ' Relecture du mois
830 '
840 IF FLAGE=1 GOTO 880
850 PRINT "****** ERREUR SYSTEME ******"
860 STOP
870 '
880 ' ** ERREUR SUR LA VALEUR DU JOUR **
890 X=1 : Y=Y0+2
900 PRINT FNI$ : PRINT FNP$(X,Y); : PRINT JOUR$
910 GOTO 530        ' Relecture du jour et du mois
920 '
930 ' **** DATE CORRECTE ****
940 DAT$=JOUR$ + " " + MOIS$ + " " + ANNEE$
950 RETURN
960 '
1500 ' * CONTROLE DE COMPATIBILITE ENTRE LE JOUR ET LE MOIS *
1510 K=0 : FLAGE=0 : FLAGB=0
1520 FOR I=1 TO 12
1530 IF MOIS$=M$(I) THEN K=1
1540 NEXT I
1550 '
1560 J=VAL(JOUR$)
1570 IF K<>2 OR J<>29 GOTO 1630
1580 ' * Dans ce cas, J=29 et K=2 *
1590 GOSUB 2000     ' Sous-programme année bissextile
1600 IF FLAGB=1 GOTO 1650
1610 GOTO 1660
1620 ' * Dans ce cas, K<>2 ou J<>29 *
1630 IF K=0 THEN FLAGE=2 : GOTO 1660     ' Erreur de mois
1640 IF J<=NJCK(I) GOTO 1660
1650 FLAGE=1        ' Erreur de jour
1660 RETURN
1670 '
2000 ' * SOUS-PROGRAMME ANNEE BISSEXTILE *
2010 V=VAL(ANNEE$)
2020 V1=V MOD 4
2030 '
2040 IF V1=0 GOTO 2060     ' Année bissextile
2050 FLAGB=1               ' Année non bissextile
2060 RETURN
```

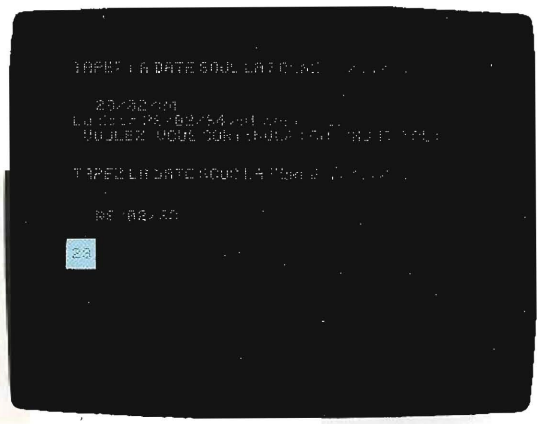
effacement. On peut très bien prévoir l'entrée d'autres données dans une autre partie de l'écran.
On a complété le sous-programme par des instructions qui attirent l'attention de l'opérateur sur les données incorrectes, en les affichant en vidéo inverse.
Cette méthode est très souvent employée dans les programmes qui demandent la saisie d'un grand nombre de données, car elle permet à l'opérateur de voir son erreur du premier coup d'œil.

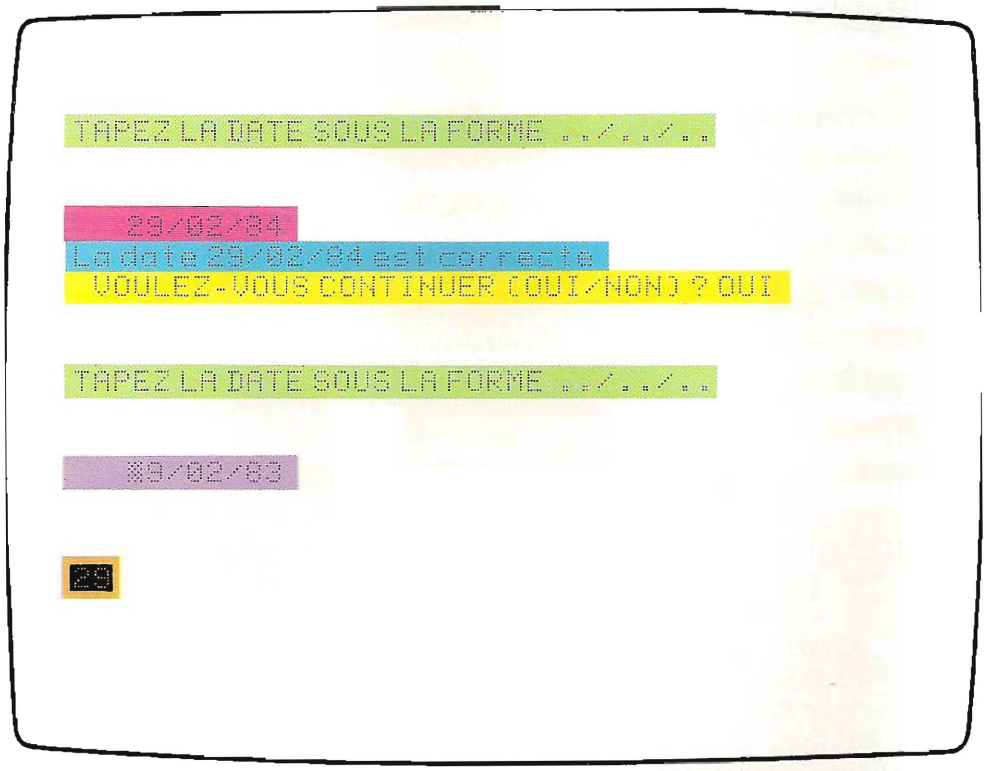
Les touches de fonction

De nombreux ordinateurs possèdent des touches programmables. On les appelle « touches de fonction » car leur frappe déclenche l'exécution de fonctions programmées.
La frappe d'une touche génère le code ASCII correspondant. La touche marquée A est ainsi associée à 65 (nombre décimal), celle marquée B à 66 et ainsi de suite. Aucun code n'est, au contraire, attribué a priori aux touches de fonction. Le programmeur dispose d'instructions pour leur associer lui-même telle ou telle valeur, en fonction de ses besoins.
Les touches de fonction sont réservées aux commandes. On leur affecte donc des codes qui ne sont pas utilisés en ASCII.
On peut ainsi attribuer à une touche de fonction le nombre 18, qui, en ASCII, ne correspond ni à une lettre, ni à un chiffre. Le système reconnaîtra alors ce code et pourra interrompre une opération en cours pour exécuter la commande associée. On pourrait, par exemple, faire correspondre la sauvegarde sur disque au code 18, l'impression au code 19, etc. L'organigramme ci-dessous illustre la méthode à employer : il utilise trois touches de fonction, qui ont été associées aux valeurs 18, 19 et 20.
Il faut, bien entendu, attribuer explicitement à

SAISIE ET CONTROLE DE LA DATE

Cette page retrace l'exécution du programme DATE (pages 604 et 605).





L'instruction 505 commande l'affichage de cette phrase, destinée à guider l'utilisateur.

L'exécution de la ligne 520 fait apparaître à l'écran le masque de saisie ../../.. La photographie a été prise alors que l'opérateur avait déjà tapé la date.

L'entrée de la date est suivie du contrôle de validité prévu dans le programme. 1984 étant bissextile, la subroutine vérifie que la date 29/02/84 est correcte, et en avertit l'opérateur (ligne 94).

Le programme (ligne 100) demande à l'opérateur s'il désire tester une nouvelle date. Celui-ci répond par l'affirmative, et l'exécution reprend en ligne 92.

Cette fois, la date entrée est erronée. Il ne peut, en effet, y avoir de 29 février 1983 puisque l'année 1983 n'est pas bissextile.

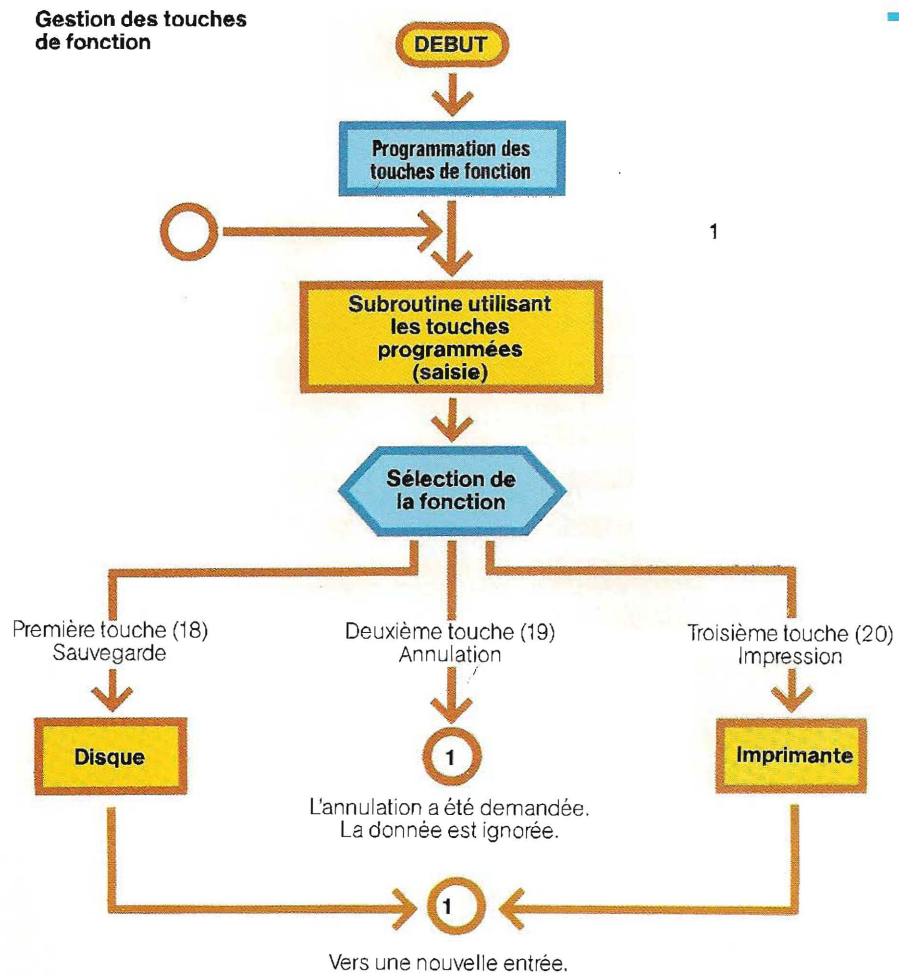
Cette erreur est détectée par le contrôle de compatibilité jour/mois. Le numéro du jour apparaît donc en vidéo inverse (instructions 890 et 900) tandis que le curseur se repositionne sur le premier chiffre de ce numéro (lignes 530 et 540).

chaque touche la valeur numérique voulue, avant de pouvoir l'utiliser. Les programmes se servant de touches de fonction doivent donc inclure un module d'initialisation des codes.
La programmation d'une touche de fonction conserve son effet jusqu'à l'extinction de la machine, à moins qu'on ne la modifie par une nouvelle affectation.
L'identification des touches de fonction n'obéit pas aux mêmes critères sur toutes les machines. Dans l'exemple que nous allons développer, les touches sont désignées par des lettres minuscules.
Sur un certain nombre de machines, le module d'initialisation est constitué de la séquence suivante :

- Escape = CHR$(27)
- Fonction d'initialisation = CHR$(46)
- Identification de la touche (pour la touche c, par exemple, CHR$(99) puisque c = 99 en ASCII).
- Valeur hexadécimale à associer à la touche, sous la forme de deux caractères en représentation ASCII.

Ainsi, le code ASCII de la lettre A étant 41 (en hexadécimal), il faut, pour associer la lettre A à une touche de fonction, transmettre séparément les caractères ASCII 4 et 1.
En décimal, le code de 4 est 52 et celui de 1 est 49. On représentera donc la lettre A par les caractères CHR$(52) et CHR$(49).

METHODE D'UTILISATION DES TOUCHES DE FONCTION



Voici la chaîne complète nécessaire à l'initialisation de la touche c :

A$=CHR$(27) + CHR$(46) + CHR$(99) +
Escape — Code d'initialisation — Identification de la touche (c)

CHR$(52) + CHR$(49)
Caractères représentant la valeur (A) à associer à la touche.

Ensuite, une simple instruction PRINT A$ permettra de rendre la touche opérationnelle. Remarquons que dans ce cas, le mot « PRINT » ne provoque aucune impression (de même que PRINT CHR$(7) émet un « bip »).
Le listing d'un programme qui affecte les valeurs 17, 18 et 19 à trois touches de fonctions (c, d et e) est reproduit ci-dessous, à titre d'exemple.
L'acquisition des données avec masques de saisie écran constitue l'une des applications des touches de fonctions programmables. Les pages 609 et 610 reproduisent l'organigramme général d'un module de saisie type. Le détail de ce module sera développé un peu plus loin. Pour l'instant, notons simplement qu'il effectue les opérations suivantes :

1 / affichage, en un point quelconque de l'écran, du libellé de la donnée à saisir. Visualisation de la longueur de cette donnée par des pointillés ;

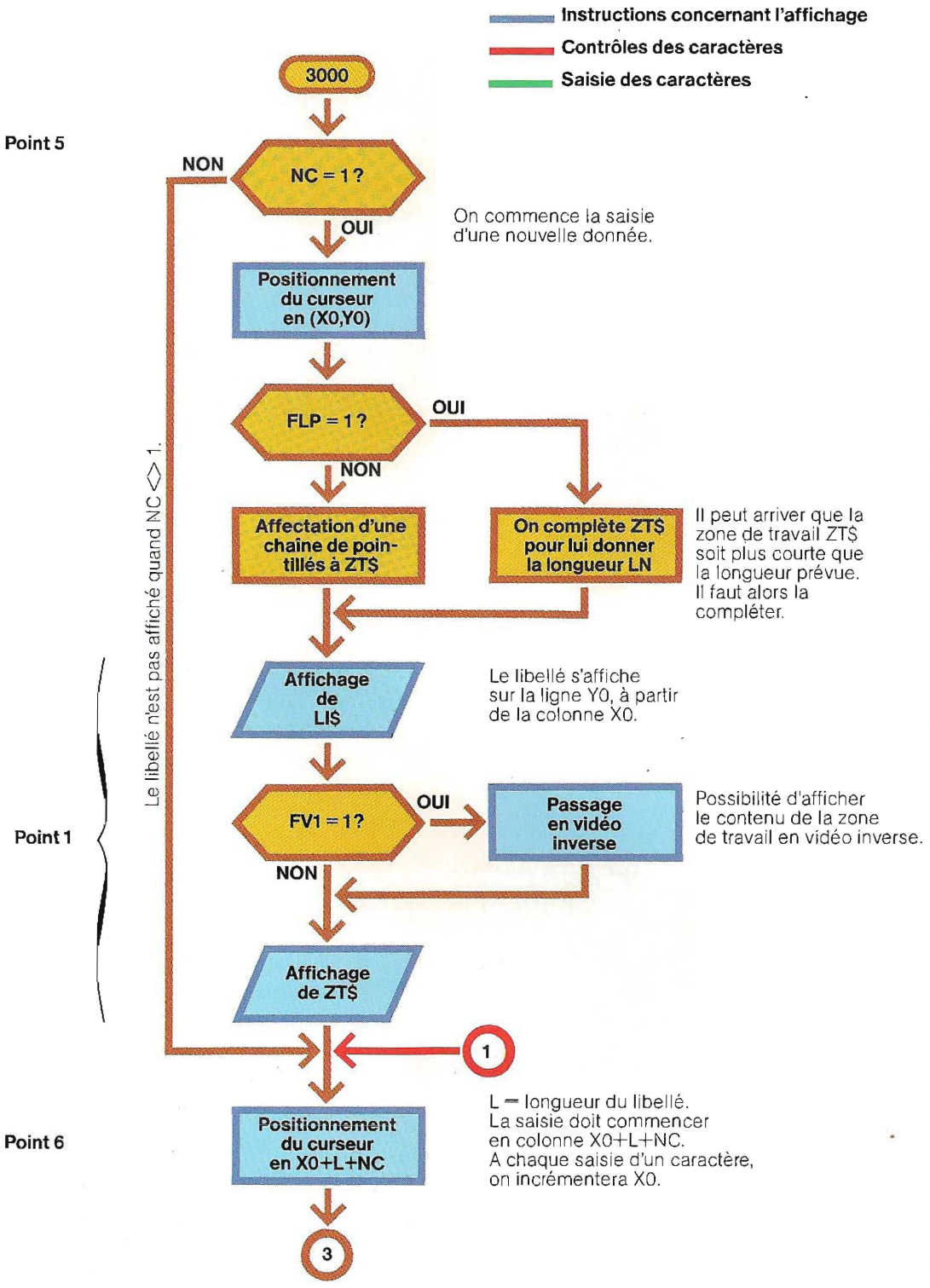
2 / contrôle du type de chaque caractère entré. Si TP vaut 1, la donnée attendue doit être

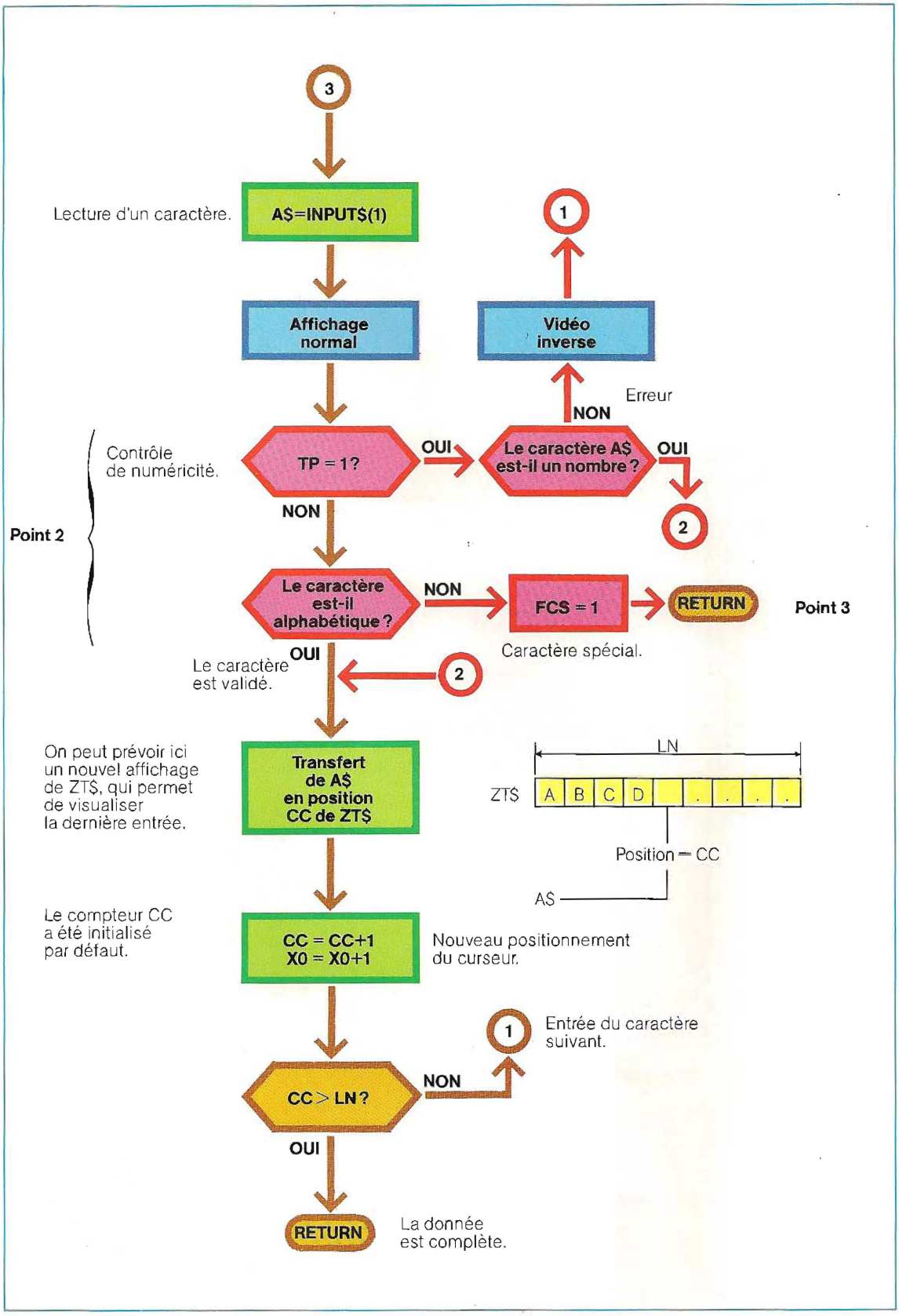
EXEMPLE DE PROGRAMMATION DES TOUCHES DE FONCTION

```
10 ' *** EXEMPLE DE PROGRAMMATION DES TOUCHES DE FONCTION ***
15 '     FICHIER TFN
30 DEFINT A-Z
40 A$=CHR$(27)+CHR$(46)      ' PARTIE COMMUNE DE LA CHAINE
50 B3$=CHR$(99)              ' TOUCHE N.2 = c
60 B4$=CHR$(100)             ' TOUCHE N.3 = d
70 B5$=CHR$(101)             ' TOUCHE N.4 = e
71 ' ** ATTRIBUTION DE LEUR VALEUR AUX TOUCHES DE FONCTION **
72 '    Les valeurs doivent etre exprimees en hexadecimal
73 '    Exemple : 11 hexadecimal equivaut a 17 decimal
74 '    Si l'on attribue cette valeur a une touche,
75 '    on obtiendra 17 par la fonction ASC
80 C1$=CHR$(49)+CHR$(49)       ' TOUCHE N.2 = 17
90 C2$=CHR$(49)+CHR$(50)       ' TOUCHE N.3 = 18
92 C3$=CHR$(51)+CHR$(51)       ' TOUCHE N.4 = 19
110 PRINT A$+B3$+C1$                ' PROGRAMMATION
120 PRINT A$+B4$+C2$                '      DES
130 PRINT A$+B5$+C3$                '    TOUCHES
140 '
150 PRINT "Enfoncez une touche quelconque"
160 I$=INPUT$(1)
162 ' Pour connaitre la valeur correspondant a la touche frappee,
163 ' on utilise la fonction ASC(I$). On obtient l'equivalent
164 ' decimal de la valeur hexadecimale attribuee a
165 ' la touche
170 IF ASC(I$)<17 OR ASC(I$)>19 THEN GOTO 260    ' On n'a programme que
172 '                                              les valeurs 17, 18 et 19
180 K=ASC(I$)-16       ' K vaut 1, 2 ou 3 selon la touche enfoncee
190 ON K GOTO 200, 220, 240
200 NU=2 : GOTO 250
220 NU=3 : GOTO 250
240 NU=4
250 PRINT "VOUS AVEZ APPUYE SUR LA TOUCHE DE FONCTION N.";NU
255 GOTO 270
260 PRINT "VOUS N'AVEZ PAS APPUYE SUR UNE TOUCHE DE FONCTION PROGRAMMEE"
270 INPUT "VOULEZ-VOUS CONTINUER ";REP$
280 IF REP$="OUI" THEN GOTO 150
300 END
```

SUBROUTINE D'ACQUISITION DES DONNEES AVEC MASQUE DE SAISIE

L'alignement à droite ou à gauche (point 4) ne fait pas partie de ce module : il vaut mieux le traiter séparément.




3
Lecture d'un caractère.
A$=INPUT$(1)
Affichage normal
1
Vidéo inverse
Erreur
NON
Point 2
Contrôle de numéricité.
TP = 1?
OUI
Le caractère A$ est-il un nombre ?
OUI
2
NON
Le caractère est-il alphabétique ?
NON
FCS = 1
RETURN
Point 3
Caractère spécial.
OUI
Le caractère est validé.
2
On peut prévoir ici un nouvel affichage de ZT$, qui permet de visualiser la dernière entrée.
Transfert de A$ en position CC de ZT$
LN
ZT$
A B C D
Position = CC
A$
Le compteur CC a été initialisé par défaut.
CC = CC+1
X0 = X0+1
Nouveau positionnement du curseur.
1
Entrée du caractère suivant.
CC > LN ?
NON
OUI
RETURN
La donnée est complète.

numérique ; en cas d'erreur, le curseur n'avance pas ; le caractère erroné s'affiche en vidéo inverse et on redemande une saisie. Si TP est différent de 1, le caractère doit être alphabétique ;

3 / lorsque le caractère entré n'est ni une lettre, ni un chiffre, son code numérique est mémorisé dans une variable, on positionne un flag (FCS) à 1 et on sort du sous-programme. Cette méthode permet de faire intervenir des touches de fonction programmables, gérées dans une autre routine ;

4 / quand une donnée ne remplit pas la zone qui lui est allouée, elle est alignée à droite ou à gauche, selon qu'elle est numérique ou alphanumérique, puis réaffichée. Ce point, traité séparément, n'apparaît pas dans notre organigramme ;

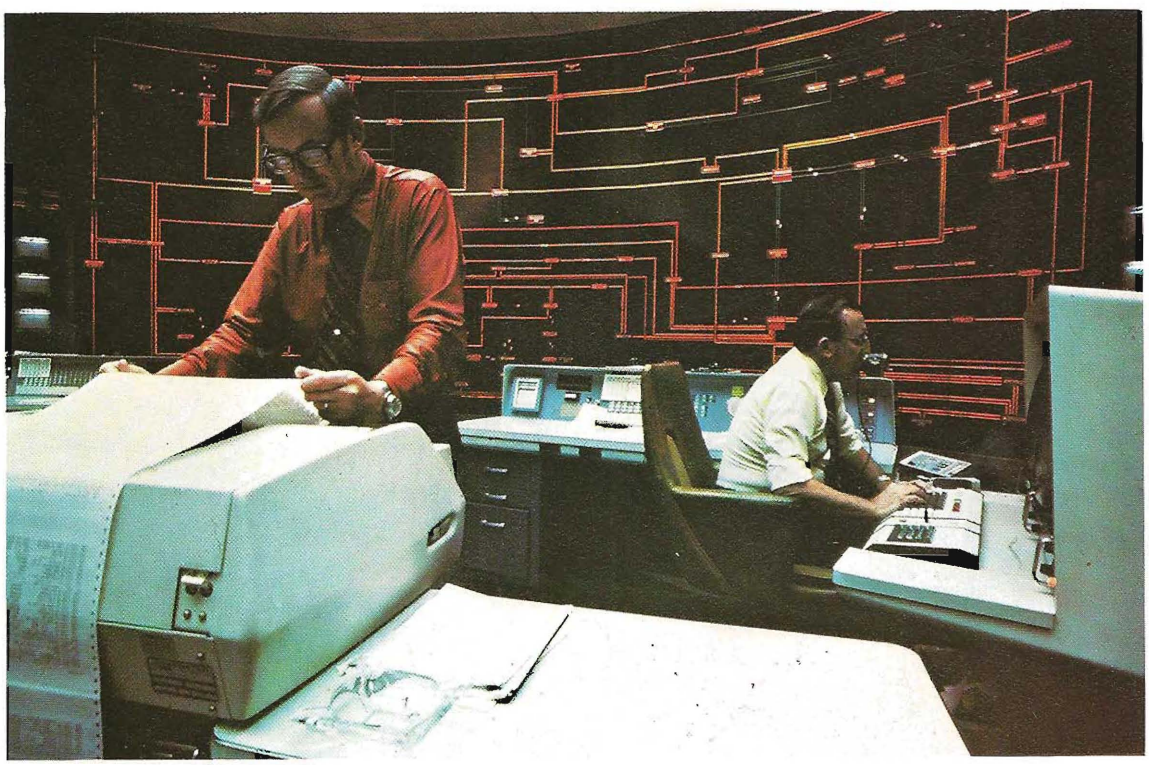
5 / les données transitent par une zone de travail, dont l'affichage est subordonné à la valeur d'un flag (NC). Lorsqu'on appelle ce sous-programme, on peut donc afficher une donnée à compléter (NC + 1) ou alors (NC = 1), afficher un libellé (indiquant qu'on attend la saisie d'une nouvelle donnée), suivi de pointillés ;

6 / la position initiale du curseur (où commencera la saisie) doit être déterminée dans le programme d'appel si on veut utiliser cette routine pour une saisie partielle.

Le rôle des points 5 et 6 sera éclairci par un exposé plus approfondi des masques de saisie.
Voici la liste des variables utilisées dans ce sous-programme :

ZT$ = Zone de travail où les données sont stockées temporairement ;
LN = longueur de la zone allouée à la donnée (en caractères) ;
TP = type de la donnée attendue (1 = numérique, 2 = alphanumérique) ;
LI$ = chaîne contenant le libellé de la donnée ;
X0,Y0 = position du curseur (coordonnées) sur l'écran ;
NC = numéro du caractère à partir duquel doit commencer la saisie ;

La salle de contrôle d'une centrale électrique moderne.



K. Reese/Marka

Informatique et vie privée

Banques de données : *voilà l'un des termes les plus répandus du jargon informatique. Il faut dire qu'elles représentent aujourd'hui un organe essentiel de la direction des institutions gouvernementales et des établissements publics ou privés. Dans les pays industrialisés, aucun citoyen ne peut désormais se vanter de n'être recensé dans aucun fichier informatisé. Toutefois, cette « mise en fiches » ne mérite pas d'être systématiquement dénigrée. Sans elle, les administrations seraient impuissantes, les services d'action sanitaire et sociale ne pourraient remplir leur rôle ; comment planifier les impôts, déterminer l'utilité des travaux publics et mener une politique d'urbanisme cohérente sans des recensements nationaux ?*

Il s'avère cependant légitime de s'inquiéter de certains abus. C'est ainsi qu'aux Etats-Unis, on ne compte pas moins de deux mille établissements de crédit qui recueillent d'énormes quantités d'informations confidentielles. Pour ne citer qu'un exemple, 100 millions de dossiers sont rassemblés dans les fichiers de l'un d'eux.

Pourtant, bien plus que la quantité des informations recueillies, la nouveauté réside dans la façon dont elles sont mémorisées.

Il y a bien longtemps, en effet, que l'on rassemble des informations, mais on les entassait autrefois dans des archives réparties sur d'interminables rangées d'étagères, où il n'était pas toujours facile de les retrouver. La disponibilité des informations s'est considérablement accrue avec l'apparition des mémoires de masse à support magnétique, de capacité bien supérieure et d'accès beaucoup plus aisé.

De plus, la notion de distance n'a plus guère de sens puisqu'on peut les interroger depuis des terminaux très éloignés. C'est ainsi qu'il n'y a pas si longtemps, les dossiers de sécurité sociale du Canada étaient gérés dans un centre situé en Corée.

Il faut reconnaître qu'il est pratiquement impossible pour le particulier de connaître l'existence et la finalité des données enregistrées sur son compte et de savoir qui y a accès. C'est pourquoi on cherche, en dépit de nombreux obstacles, à mettre en place des conventions internationales. De nombreux défenseurs des libertés ont cherché à protéger la vie privée du citoyen, en faisant reconnaître le droit de chacun au contrôle de la divulgation des informations qui le concernent.

Tout citoyen est désormais contraint de fournir des renseignements personnels à différents ***services administratifs*** *qui les engrangent dans leurs banques de données. Il est impossible d'ouvrir un compte ou de conduire une automobile sans se plier à cette obligation. Il faudrait cependant limiter le volume d'informations conservées dans ces banques et leur dissémination. Il semble, en outre, qu'à l'époque de l'ordinateur, le sérieux et l'exactitude des informations nominatives ne soient pas garantis comme on pourrait le souhaiter. Il arrive, en effet, que des renseignements soient inutiles, incomplets, périmés, voire complètement erronés. Plus grave encore, des informations fournies dans un but précis pourraient être communiquées à d'autres services à des fins totalement différentes. Enfin, sans l'institution d'un « droit de regard », comment s'assurer que des renseignements ne sont pas recueillis illégalement et à l'insu des citoyens !*

Les fichiers des ***services de police*** *contiennent, d'ores et déjà, un nombre d'informations impressionnant. Même dans les pays les plus démocratiques, les ordinateurs de la police ingurgitent et classent les informations les plus diverses, que ce soit des listes de véhicules volés, d'automobilistes indisciplinés, de propriétaires de véhicule, ou des renseignements sur des personnes faisant l'objet d'un avis de recherche, en passant par le signalement anthropométrique et les empreintes digitales des criminels. On constitue aussi parfois des fichiers concernant les personnes simplement suspectées d'activités politiques clandestines ou terroristes. En Grande-Bretagne, toute personne qui acquiert un véhicule doit le faire immatriculer au centre de Swansea. Transmis à l'ordinateur central de la police, les renseignements ainsi réunis vont alimenter un fichier où sont déjà recensés 26 millions de citoyens, dont la plupart n'ont vraiment rien à se reprocher.*

Comme nous l'avons vu, il arrive que des informations fournies à une administration

pour satisfaire une obligation légale soient divulguées pour servir à tout autre chose. Enfin, il est de notoriété publique que le secret des informations détenues par la police est particulièrement mal gardé, puisque les détectives des agences privées – souvent dirigées par d'anciens membres de la police – conservent leurs facilités d'accéder aux fichiers qui les intéressent.
Des criminels, qui connaissent un numéro de téléphone confidentiel, ont même réussi à obtenir des renseignements, en se faisant passer pour des policiers.

Il est très difficile de préserver des indiscrétions les informations conservées dans les banques de données.
Pour chercher une information dans les archives traditionnelles, il fallait qu'une personne ouvre une pièce ou une armoire. Il était donc possible de protéger les fichiers par des serrures de sécurité. De plus, si quelqu'un parvenait, malgré tout, à déjouer cette protection, il ne pouvait recopier, photographier ou emporter qu'un nombre limité de documents. Au contraire, et malgré l'extrême sophistication de l'informatique, on n'a pas encore inventé l'ordinateur qui saurait empêcher un escroc, décidé et... possédant une solide expérience d'informaticien, d'accéder à un fichier et de s'emparer de son contenu en une seule fois.
Certains de ces délits ont été portés à la connaissance du public et ont fait beaucoup parler d'eux. Ainsi, des étudiants du Massachusetts Institute of Technology ont réussi à se procurer des informations de la plus haute importance pour la défense nationale de leur pays. A l'université du Michigan, d'autres étudiants ont détruit plusieurs fichiers après en avoir décrypté les mots de passe. Un passionné d'informatique est parvenu à obtenir, dans les fichiers du FBI, des renseignements sur des personnes suspectes ou arrêtées. Ces « professionnels » de haut vol sont d'ailleurs très recherchés. On en a arrêté quelques-uns, mais ils ont presque tous été engagés à prix d'or par leurs victimes, en tant que conseillers en matière de sécurité !
Par ailleurs, la connection des ordinateurs à des terminaux très éloignés accroît les risques en tous genres. Il suffit d'une interférence entre deux lignes téléphoniques pour qu'une information soit divulguée ou même détruite. Le problème se pose donc en ces termes : que peut-on faire pour ***protéger la vie privée du citoyen****, tout en permettant aux gouvernements et aux administrations de continuer à collecter les informations nécessaires, dans l'intérêt de ce même citoyen ?*
En Grande-Bretagne, les suspects arrêtés dans le cadre des mesures anti-terroristes sont systématiquement photographiés et leurs empreintes digitales sont relevées. Ces informations demeurent ensuite dans les fichiers même s'il n'y a pas de suites judiciaires (et c'est le cas le plus fréquent). On peut ficher les personnes qui ont un lien quelconque avec des criminels connus, ainsi que les personnes qui sont placées sous surveillance parce qu'on les suspecte de s'apprêter à commettre un délit. Tout cela s'insère dans une politique de prévention légitime, dont le but consiste à empêcher les délits ou d'accélérer l'arrestation des coupables.

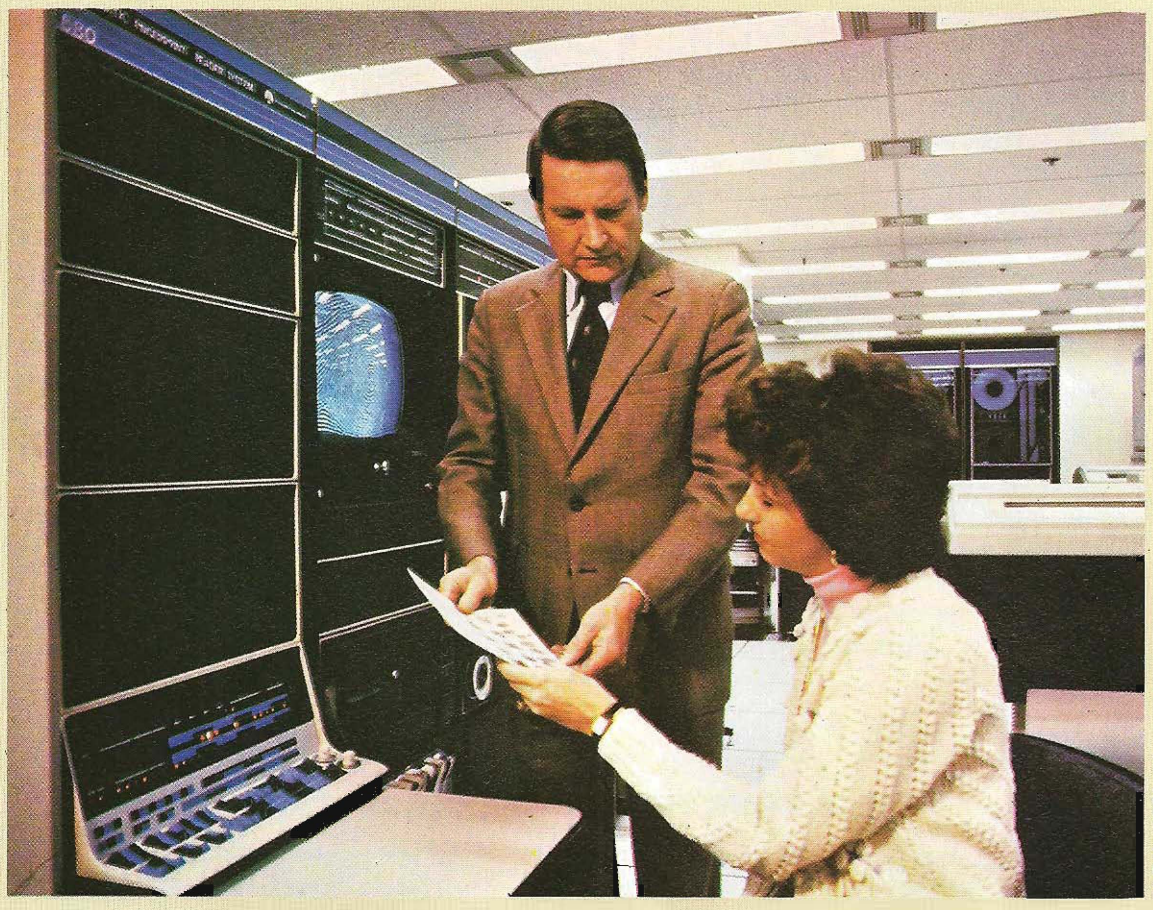
La technologie permet aujourd'hui de corréler, de confronter et de transmettre d'énormes quantités de renseignements que le simple particulier jugerait sans doute tout à fait personnels. Des agences spécialisées mettent des « registres de crédit » à la disposition des responsables d'entreprise qui désirent se renseigner sur la situation financière de leurs clients potentiels. La plus grande banque de données du monde appartient à l'une de ces agences, la TRW Credit Data. Elle rassemble des informations nominatives sur plus de 50 millions de citoyens américains.
Les principales banques d'escompte suédoises ont récemment créé un institut, l'Agence d'Informations pour le Crédit, qui fournit le nom, l'adresse, les revenus, l'état-civil, les activités imposables et la situation judiciaire de tous les citoyens suédois âgés de plus de 16 ans, ainsi que les contrats de location ou de vente qu'ils ont conclus. Cependant, cet institut remet à l'intéressé une copie de tous les documents communiqués à des tiers.
Malheureusement, il est rare que les ***agences de crédit*** *vérifient l'exactitude des informations recueillies. Leur rôle se limite à collecter des renseignements sur les dettes et les créances irrécouvrables et à les communiquer aux organismes qui ont besoin de s'informer avant d'accorder un prêt.*

L'existence de ces fichiers se justifie, parce qu'ils permettent de prévenir l'endettement excessif. Cependant, ils sont parfois à l'origine de graves erreurs. Citons le cas de cet homme d'affaires, à la tête d'une entreprise prospère, qui, désireux de développer son activité, sollicita un prêt et essuya un refus. Bouleversé, il demande une étude sur la situation de son entreprise et il lui fut conseillé de ne pas trop insister. Il approfondit pourtant la question et finit par découvrir qu'une plainte avait été déposée peu auparavant contre l'un de ses homonymes.

Personne ne songe à contester l'importance des ***services sociaux****. Toutefois, on peut regretter que leur organisation fasse peser de lourdes menaces sur les libertés individuelles. En effet, l'inquiétude et le désarroi conduisent de nombreuses personnes à se confier au personnel de ces services.*
Il peut très bien arriver que les informations enregistrées dans une banque de données n'aient pas d'autre fondement que le jugement subjectif d'un fonctionnaire. Dans les rapports des services sociaux, «paranoïaque» et «névrotique» sont deux mots qui qualifient trop souvent les personnes venues demander des secours. Il va de soi, en effet, que la plupart de ces personnes ne viendraient pas demander une aide si elles ne traversaient pas une période d'épuisement nerveux. Le point crucial c'est que ces informations sont mémorisées par l'ordinateur : le simple fait qu'elles soient enregistrées leur donne un caractère de constance et de véracité. Il ne faut pas oublier que les témoignages des services sociaux ont souvent valeur d'expertise auprès des tribunaux et qu'un individu risque d'être étiqueté pour le restant de ses jours, à cause d'un jugement hâtif.
Les informations recueillies au sujet des habitants par les autorités locales sont ventilées dans plusieurs sections, selon le service concerné : services sociaux et financiers, services du logement, de l'éducation et de l'emploi. Les informations à caractère financier abondent dans les fichiers des administrations locales, où sont détenus les registres comptables et où l'on doit connaître la situation financière des particuliers pour accorder, à bon escient, des dizaines d'aides en tout genre.

Ces fichiers sont touchés, en nombre de plus en plus grand, par l'informatisation. Or, s'il demeure théoriquement possible de coder les informations pour en interdire l'accès aux services non compétents, il n'existe ni loi, ni règlement pour empêcher l'échange d'informations entre un bureau et un autre ou même un organisme extérieur comme la police. En fait, avant qu'une personne puisse bénéficier d'une aide, son dossier doit souvent transiter par de multiples bureaux. Mais les gens qui font l'objet de ces enquêtes ne sont pas autorisées à connaître le contenu de leur propre dossier. Vrai ou faux, celui-ci est archivé et parfois étalé au grand jour, au moment le moins propice, alors qu'il est impossible de faire rectifier les erreurs éventuelles ou de demander la justification d'un jugement défavorable.

De nombreux pays sont conscients de la nécessité d'instaurer une législation pour protéger la vie privée, mais cela n'a encore été réalisé que dans quelques-uns. Aux Etats-Unis, de nombreuses mesures ont établi le libre accès aux fichiers nominatifs de l'administration fédérale, ainsi que le droit de contester les informations qu'ils contiennent et de les faire rectifier.
La Suède, où la centralisation et l'informatisation des fichiers sont importantes, a reconnu le droit d'accès de chacun aux dossiers qui le concernent, et une loi réglemente la création et la gestion des banques de données : il faut obtenir une autorisation et se soumettre à des contrôles. En Allemagne fédérale, toute information nominative doit légalement être détruite (ou, au moins, définitivement remisée) au bout de cinq ans. En France et en Hollande, il est interdit de réunir certains types d'informations. C'est ainsi que dans notre pays, il est totalement illégal de faire figurer dans les fichiers des renseignements concernant les origines ethniques, les opinions politiques et les croyances religieuses de qui que ce soit. Au Danemark, la divulgation d'informations concernant la vie privée des individus n'est pas autorisée.
De nombreuses personnes souhaitent l'instauration de contrôles officiels sur le volume, la pertinence et les sources des informations recueillies. Il est certain qu'on ne devrait enregistrer que les données réellement nécessai-



J. Pickerell/Marka

Un exemple parmi d'autres de l'efficacité de l'informatique : les banques de données du FBI, gérées par un PDP 15 de DIGITAL.

res. De plus, les renseignements fournis dans un but précis ne devraient pouvoir être utilisés à d'autres fins sans l'accord de la personne concernée. Il ne devrait s'établir aucun lien entre les différents fichiers nominatifs ; les personnes chargées de la collecte et de la conservation des informations devraient être tenues de respecter un code déontologique, sous le contrôle d'une autorité compétente.

En Grande-Bretagne, la promulgation en 1974 du Consumer Credit Act a mis fin à un certain nombre d'abus. Depuis, il est obligatoire de transmettre aux gens qui déposent une demande de prêt le nom de l'agence auprès de laquelle on se procure des renseignements sur leur situation. Ces personnes peuvent alors se faire communiquer les renseignements fournis, demander, le cas échéant, leur rectification, et même donner leur propre version des faits dans un texte de moins de 200 mots. Cette mesure présente toutefois certaines lacunes. Ainsi, elle n'oblige pas les agences à faire part des corrections aux autres établissements d'où pourraient émaner de nouvelles demandes de renseignements, et elle ne réglemente pas la façon dont elles se procurent leurs informations. Elle n'interdit pas non plus la création et la vente de fichiers injustifiables, contenant, par exemple, des informations sur les idées politiques ou la vie sexuelle des personnes. De surcroît, elle ne limite pas la durée de validité des informations et les dossiers peuvent donc très bien mentionner des dettes remboursées depuis longtemps.

Le même objectif est donc partagé un peu partout dans le monde : protéger les individus contre la divulgation d'informations confidentielles. Toutefois, la rapidité de l'évolution technologique et, notamment, la percée des micro-ordinateurs risquent de donner du fil à retordre au législateur.

CC = compteur du nombre de caractères entrés ;
FCS = ce flag prend la valeur 1 après la saisie d'un caractère spécial (ni alphabétique, ni numérique) ;
FLP = flag d'initialisation. Si FLP vaut 1, le contenu précédent de ZT$ est conservé ; sinon, on lui affecte une chaîne de pointillés ;
FVI = flag d'inversion vidéo ; si FVI est mis à 1 avant l'appel du module, ZT$ sera affiché en vidéo inverse.

On sort du sous-programme dans deux cas :

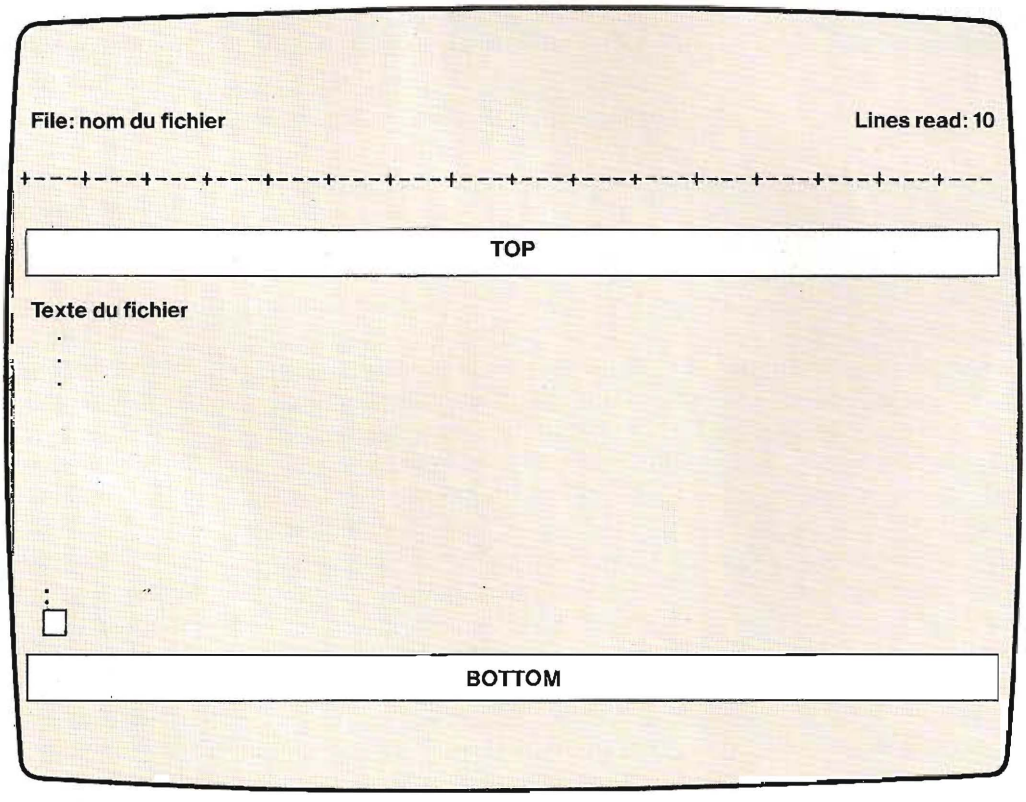
– la donnée est complète, c'est-à-dire que la zone allouée a été remplie (CC = LN) ;
– le caractère saisi n'est ni numérique, ni alphabétique (point 3).
Dans ce dernier cas, le code ASCII du caractère est transmis au programme d'appel ainsi qu'un flag (FCS).

Les éditeurs de texte

On peut tirer parti des multiples possibilités du visuel pour concevoir des programmes spécialement destinés à la préparation des textes ou à la représentation graphique des données. Nous commencerons par l'exemple du Video File Editor*, programme d'application développé par Olivetti pour faciliter l'écriture et la modification des fichiers. Comme la plupart des éditeurs, il s'applique aussi bien aux programmes (sauvegardés en ASCII) qu'aux fichiers de données.
Nous allons décrire les fonctions prévues par ce système, afin de bien montrer l'intérêt qu'il représente.
Tout éditeur permet la création et la modification de fichiers texte, dont les enregistrements sont composés de caractères ASCII imprimables, séparés par les caractères CR,

* Video File Editor est une marque déposée.

STRUCTURE TYPE D'UN FICHIER EN EDITION



LF (Carriage Return = retour chariot et Line Feed = avancement d'un interligne) ou par le caractère RS (Record Separator).
On passe sous éditeur à l'aide de la commande EDIT, suivie d'un nom de fichier. Ce dernier est alors chargé en mémoire, ou créé s'il n'existait pas.
L'éditeur affiche alors dans une « fenêtre » la première partie du fichier. Les dimensions de cette fenêtre varient selon les systèmes : sur le Video File Editor, elle comporte vingt-quatre lignes de quatre-vingts caractères, dont vingt-et-une sont réservées au texte du fichier et les autres à des indications diverses. L'utilisateur peut modifier en mémoire les lignes affichées à l'écran, au moyen des fonctions et des commandes de l'éditeur.
Il peut également remplacer la portion de fichier présente dans la fenêtre par n'importe quelle autre. Quand il a terminé, il sauvegarde sur disque la version modifiée du fichier.
Sous Video File Editor, l'écran se présente comme le montre le schéma de la page 616. Le nom du fichier et sa longueur (en nombre de lignes) s'affichent sur la première ligne.
Rien n'apparaît sur la seconde ligne : elle servira par la suite à l'entrée de commandes ou à la recherche de chaînes.
La largeur de l'écran est découpée en zones de quatre caractères. Les symboles affichés sur la troisième ligne permettent de repérer la position de chaque zone et de chaque colonne. Les vingt-et-une lignes suivantes constituent la fenêtre réservée au texte (text window). La vingt-cinquième ligne de l'écran reste vide. Le début et la fin du texte sont toujours signalés par deux lignes virtuelles, appelées respectivement TOP et BOTTOM (ces mots sont affichés en vidéo inverse). Elles ne font pas partie du texte mais en indiquent simplement les limites.
Quand on passe sous Video File Editor, la ligne TOP apparaît au-dessus de la première ligne du fichier et c'est sur elle que le curseur est positionné. La ligne BOTTOM apparaîtra après la dernière ligne du texte. A sa création, un fichier est vide : on ne voit que les lignes TOP et BOTTOM.
Quant on passe en mode Insertion, le curseur change de forme : il n'apparaît plus comme un tiret placé sous le caractère mais comme un petit triangle situé à sa gauche. Cependant, il clignote dans un cas comme dans l'autre. Le clavier fonctionne différemment quand il est sous le contrôle de l'éditeur de texte. C'est ainsi qu'on peut commander les différentes fonctions d'édition au moyen d'un ensemble de touches programmées automatiquement. Le clavier du M 20 Olivetti est reproduit page 618 et les principales fonctions du Video File Editor sont illustrées pages 619 à 623.
Pour passer en mode commande, l'utilisateur doit enfoncer simultanément la touche COMMAND et la touche 1 (fonction COMMAND MODE). Le curseur se place alors au début de la ligne numéro 2.
A partir de ce moment, toutes les fonctions d'édition de lignes telles que INSERT MODE, BACKSPACE, DELETE CHAR, RECALL LINE s'appliquent à la ligne de commande. La fonction RECALL LINE rétablit, par exemple, son précédent contenu. En revanche, les autres fonctions agissent uniquement sur le fichier texte.
En mode commande, la touche → correspond à la fonction EXECUTE COMMAND et non à la fonction INSERT LINE.
Pour faire revenir le curseur sur le fichier texte et poursuivre la modification de celui-ci, il faut entrer à nouveau la fonction COMMAND MODE.

Les fonctions d'édition et de recherche des chaînes de caractères. Les fonctions de recherche de chaînes permettent de balayer le fichier afin d'y retrouver une suite donnée de caractères.
Avant de commencer la recherche, l'utilisateur doit se mettre en mode commande. Il tape ensuite la chaîne à rechercher sur la seconde ligne de l'écran. Ensuite, il entre la fonction SEARCH DOWN ou la fonction SEARCH UP selon qu'il désire parcourir le fichier vers l'avant ou vers l'arrière par rapport à la position du curseur.
Pour sortir du Video File Editor et revenir sous système, il faut transmettre la commande ABORT (COMMAND + 6) puis CR, ou encore EXIT AND SAVE (CTRL + 6) – qui détermine également la sauvegarde du fichier modifié – puis CR. Si l'on veut sauvegarder le fichier modifié tout en restant sous le contrôle de l'éditeur, il faut entrer SAVE TEXT (CTRL + 5).

Les menus

On peut toujours découper une procédure, aussi simple soit-elle, en un certain nombre

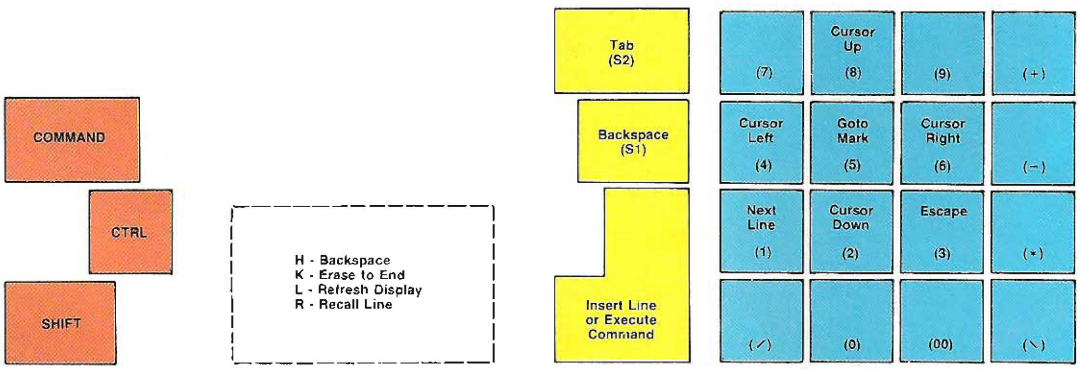
LES TOUCHES DE FONCTION DU VIDEO FILE EDITOR



Ces touches numériques sont validées en tant que touches de fonction, ici par la touche COMMAND.

Command Mode (1)	Delete Line (2)	Split Line (3)	Start of Line (4)	Restore Lines (5)	Abort (6)	Insert Mark (7)	Search Up (8)	Line Up (9)	Half Screen Up (0)	Full Screen Up (−)	Top (^)
Insert Mode (1)	Delete Char (2)	Join Lines (3)	End of Line (4)	Save Text (5)	Exit and Save (6)	Reverse Tab (7)	Search Down (8)	Line Down (9)	Half Screen Down (0)	Full Screen Down (−)	Bottom (^)

Et ici, au contraire, par la touche CTRL.



Commandes alphabétiques transmissibles en frappant simultanément la touche CTRL.

Commandes transmissibles en frappant simultanément la touche SHIFT (touche des majuscules).

FONCTIONS D'EDITION ET DE RECHERCHE DE CHAINES DE CARACTERES

Cette page et les suivantes détaillent l'action des fonctions d'édition et de recherche de chaînes du Video File Editor.
Le fichier texte à modifier est reproduit ci-contre tel qu'il apparaît sur l'écran. Vous trouverez dans la suite deux illustrations pour chaque fonction, celle de gauche indiquant les touches à enfoncer et celle de droite montrant son effet sur le texte.



DELETE LINE
Suppression de la ligne où se trouve le curseur.



CURSOR UP
Déplacement du curseur d'une ligne vers le haut.



INSERT MODE This is
Insertion de la chaîne indiquée à partir de la position du curseur.

```
The purpose of this text is to act
as an example of how to use
the editing functions of
the Video File Editor
```

```
as an example of how to use
the editing functions of
the Video File Editor
```

```
as an example of how to use
the editing functions of
the Video File Editor
```

```
This is
as an example of how to use
the editing functions of
the Video File Editor
```



JOIN LINES
Concaténation de la ligne où se trouve le curseur avec la suivante.



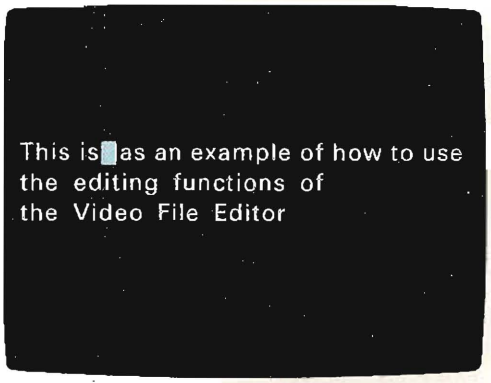
CURSOR RIGHT
Déplacement du curseur d'une colonne vers la droite.

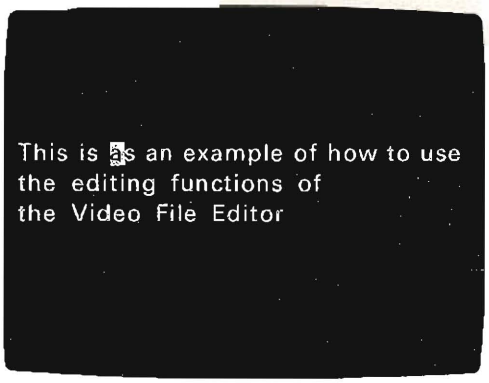


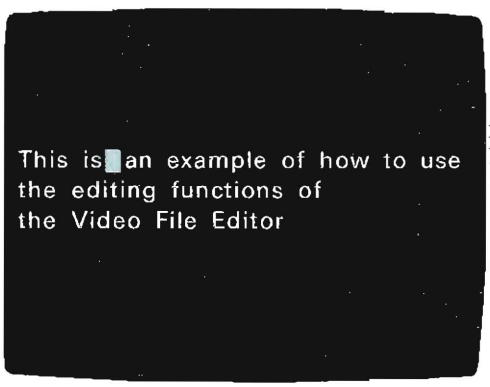
DELETE CHAR
Suppression du caractère sur lequel se trouve le curseur et déplacement d'une colonne vers la gauche de la suite de la ligne.
Ici, la fonction a été utilisée trois fois.

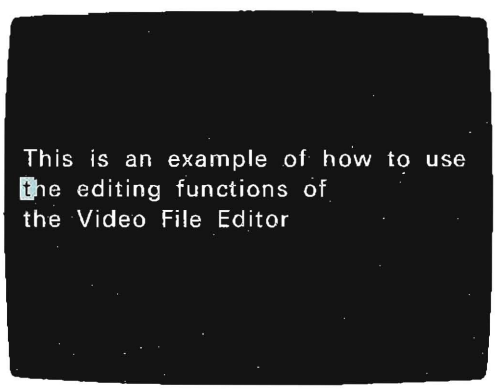


NEXT LINE
Le texte monte d'une ligne et le curseur se place au début de la ligne suivante.











DELETE LINE
Suppression de la ligne où se trouve le curseur.

```
This is an example of how to use
the Video File Editor
```



RESTORE LINES
Restitue la ligne précédemment supprimée, sous celle où se trouve le curseur.

NEXT LINE
Visualisation de la ligne suivante du texte.

```
This is an example of how to use
the Video File Editor
the editing functions of
```



INSERT MODE
Passage en mode insertion pour ajouter le point et la majuscule.

```
This is an example of how to use
the Video File Editor.
The editing functions of
```



END OF LINE
Positionnement du curseur sur la colonne qui suit le dernier caractère non blanc de la ligne.

```
This is an example of how to use
the Video File Editor.
The editing functions of
```



BACKSPACE
Déplacement du curseur d'une colonne vers la gauche et suppression du caractère ainsi pointé. Ici, la fonction a été utilisée à deux reprises.

```
This is an example of how to use
the Video File Editor.
The editing functions
```



RECALL LINE
Restitution du précédent contenu de la ligne. Cependant, le curseur ne retrouve pas son ancienne position.

```
This is an example of how to use
the Video File Editor.
The editing functions of
```



SPLIT LINE
Renvoi à la ligne suivante des caractères situés à droite du curseur (y compris celui sur lequel il se trouve).

```
This is an example of how to use
the Video File Editor.
The editing functions
of
```



CURSOR UP
Déplacement du curseur d'une ligne vers le haut.

```
This is an example of how to use
the Video File Editor
The editing functions
of
```



INSERT LINE
Insertion d'une ligne « blanche » derrière celle où se trouve le curseur et positionnement de ce dernier au début de la nouvelle ligne.

```
This is an example of how to use
the Video File Editor.

The editing functions
of
```



Les deux exemples suivants ont trait à la recherche de chaînes de caractères. La recherche sera effectuée dans le texte reproduit ci-contre et portera d'abord sur la chaîne « func », puis sur la chaîne « e of ».

```
This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```



Si l'on entre la commande SEARCH DOWN (CTRL + 8) après avoir écrit « func » sur la ligne de commande, la recherche commence à l'endroit où se trouve le curseur et se poursuit vers la fin du texte.

```
func

This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```



Si, au contraire, on entre SEARCH UP (COMMAND + 8), la recherche de la chaîne (ici « e of ») va s'effectuer en sens inverse, en remontant de la position du curseur vers le début du texte.

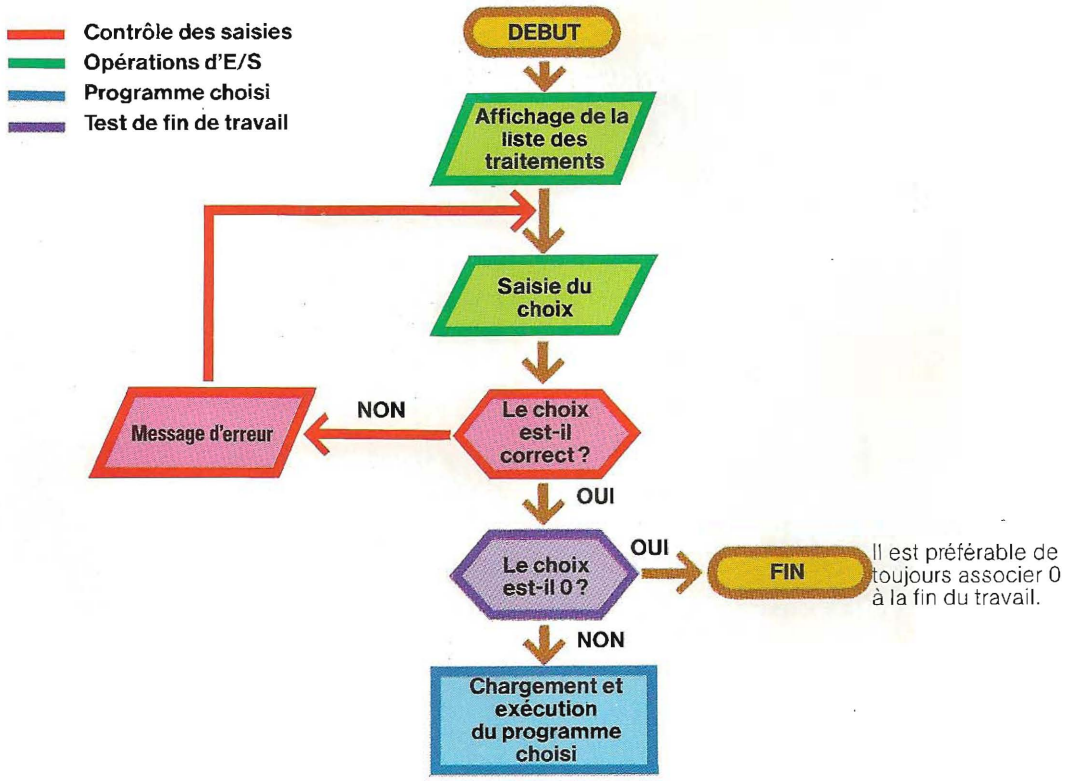
```
e of

This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```

de modules indépendants, dont chacun forme un tout.
Certes, ces différentes parties Une fois ces modules définis, il y a deux façons de procéder : soit on subordonne leur exécution à un ordre de priorité fixé dans le programme (il s'agit alors de routines classiques), soit on les choisit au fur et à mesure des besoins. Dans ce dernier cas, les modules doivent être indépendants, puisque l'utilisateur lui-même établit librement l'ordre qui lui convient le mieux.
Si le traitement requiert le respect d'un enchaînement précis des tâches, l'exécution doit être pilotée par le système et l'on préférera généralement la première méthode.
De toute façon, un bon logiciel doit toujours prévoir

- de présenter à l'utilisateur la liste des différents traitements possibles ;
- de lui demander son choix et de le contrôler ;
- de lui exposer, sur sa demande, les caractéristiques et la logique générale des programmes qui constituent cette application.

Le programmeur a d'ailleurs tout à gagner en s'astreignant à cette discipline. Il doit structurer son logiciel : la rédaction s'en trouve facilitée et le découpage en plusieurs modules évite bien des problèmes (dépassement de capacité, par exemple).
Les trois points énumérés plus haut sont généralement traités dans un module de **menu**, qui offre à l'utilisateur la possibilité de travailler, en quelque sorte, « à la carte ».
Les menus ont une structure très simple : ils affichent la liste des différents traitements proposés, et demandent l'entrée du code associé au traitement désiré.
Des instructions d'appel commandent ensuite le chargement et l'exécution du module choisi.
Ci-dessous se trouve l'organigramme d'un menu type. Le programme choisi par l'utilisateur est chargé et prend la place du menu en mémoire centrale.

ORGANIGRAMME D'UN MENU TYPE



Tous les programmes ainsi appelés doivent se terminer par une instruction permettant de recharger le menu en mémoire, en vue de la sélection du traitement suivant.

En fin d'exécution, le programme appelé doit, à son tour, recharger le menu en mémoire afin de permettre un choix ultérieur. L'instruction réalisant ces chargements est la suivante :

RUN "X:NOM"

X désigne l'unité disque où réside le programme à appeler (A/B ou 0/1) et NOM le nom par lequel le programme à exécuter est identifié sur disque.
On peut concevoir plusieurs menus imbriqués, afin de multiplier les possibilités de choix et faciliter la recherche de la fonction désirée. Par exemple, un menu principal peut appeler un certain nombre de menus secondaires, chacun étant consacré à une certaine catégorie de fonctions ; à leur tour, les menus secondaires appellent des programmes spécifiques. A la fin de chaque programme, on devra recharger le menu secondaire à partir duquel il a été appelé, et ce dernier appellera le menu principal. De cette manière, le choix sera possible dans les deux sens : d'une notion générale, arriver au détail et, à l'opposé, de l'application particulière, remonter au cas général. Cette page et la suivante illustrent le listing d'un menu principal appelant d'autres menus. Ce programme constituant le point d'entrée de toute la procédure, on doit y prévoir les fonctions communes à tous les programmes, comme l'initialisation des touches de fonction ou la définition d'une forme particulière du curseur.
On remarquera, à cette occasion, que les touches de fonction sont seulement initialisées, c'est-à-dire qu'un code numérique leur est affecté. La fonction à exécuter sous ce code sera, elle, établie par chaque programme. Par exemple, dans le menu principal, la valeur numérique 2 peut être attribuée à une touche donnée : ce code, dans un programme déterminé, pourra faire débuter les fonctions d'impression, dans un autre programme il pourra

MENU PRINCIPAL

```
4540 ' PROGRAMME MENU
4541 ' *****FICHIERS*****
4542 '
4543 '    NOM       DRV    DESCRIPTION                OCTETS     CHAMPS
4544 '
4545 ' REPCLI         A     Repertoire clients          128         7
4546 ' REPFOU         A     Repertoire fournisseurs     128         7
4547 ' MARGE H        A     Marge hebdomadaire           91         9
4548 ' MARGE G        A     Marge generale              100        10
4549 ' FACTUR         B     Factures                     70         9
4550 ' RECETT         B     Recettes generales           70         9
4551 ' PAIES          B     Paiements generaux           70         9
4552 ' NCRED          B     Notes de credit              70         9
4553 ' TTVA           A     Taux TVA                    128         7
4554 ' TCRED          A     Taux credit                 128         7
4555 '
4556 LPRINT CHR$(27)+CHR$(82)    ' definition de l'espacement
4560 GOSUB 6040          ' Initialisation des touches de fonction
4580 BI$=CHR$(27)+"+"+CHR$(7)
4600 GOSUB 4960
4620 ON V GOTO 4640, 4660, 4680, 4700, 4720, 4740, 4760, 4780, 4800, 4820
4640 RUN "A:MENU1"          ' STOCK
4660 RUN "A:MENU2"          ' REPERTOIRE CLIENTS ET FOURNISSEURS
4680 RUN "A:MENU3"          ' RECETTES
4700 RUN "A:MENU4"          ' FACTURATION
4720 RUN "A:MENU5"          ' MARCHANDISES
4740 RUN "A:MENU6"          ' NOTES DE CREDIT
4760 RUN "A:RECLI"          ' RECUS CLIENTS
4780 RUN "A:REFOUR"         ' RECUS FOURNISSEURS
4800 RUN "A:HLP1"
4820 PRINT BI$
4840 X=30 : Y=10 : GOSUB 5700
4860 PRINT CHR$(27)+"G1"+"FIN DE TRAVAIL"     ' message souligne
4880 PRINT CHR$(27)+"G0"                      ' video normale
4900 STOP
4920 END
```

```
4940          ***** AFFICHAGE DU MENU *****
4960 PRINT BI$
4980 X=2 : Y=1 : GOSUB 5700
4990 PRINT "COMPTABILITE GENERALE"
5000 X=15 : Y=6 : GOSUB 5700
5020 PRINT " 1 - STOCK "
5040 Y=Y+1 : GOSUB 5700
5060 PRINT " 2 - REPERTOIRE CLIENTS ET FOURNISSEURS "
5080 Y=Y+1 : GOSUB 5700
5100 PRINT " 3 - RECETTES ET PAIEMENTS "
5120 Y=Y+1 : GOSUB 5700
5140 PRINT " 4 - FACTURATION "
5160 Y=Y+1 : GOSUB 5700
5180 PRINT " 5 - MARCHANDISES "
5200 Y=Y+1 : GOSUB 5700
5220 PRINT " 6 - NOTES DE CREDIT "
5240 Y=Y+1 : GOSUB 5700
5260 PRINT " 7 - RECUS CLIENTS "
5280 Y=Y+1 : GOSUB 5700
5300 PRINT " 8 - RECUS FOURNISSEURS "
5320 Y=Y+1 : GOSUB 5700
5340 PRINT " 9 - Explications sur l'utilisation du programme "
5360 Y=Y+1 : GOSUB 5700
5380 PRINT " 0 - FIN DE TRAVAIL "
5400 X=1 : Y=Y+2
5410 GOSUB 5700
5420 PRINT " ENTRER L'OPTION CHOISIE : "
5440 X=26 : GOSUB 5700
5450 A$=INPUT$(1)
5460 PRINT A$
5470 V=VAL(A$)
5480 IF V>9 GOTO 5600        ' ERREUR DE SAISIE
5500 PRINT BI$
5520 IF V=0 THEN V=10 : GOTO 5680        ' SORTIE
5540 X=5 : Y=6 : GOSUB 5700
5560 PRINT "PATIENCE ! On charge le programme"
5580 GOTO 5680
5600 PRINT CHR$(7), CHR$(7)        ' TRAITEMENT D'ERREUR
5620 X=42 : GOSUB 5700
5640 PRINT "CODE NON VALIDE"
5660 GOTO 5440
5680 RETURN
5685 '
5690 '
5695 '
5700 ' **** POSITIONNEMENT DU CURSEUR ****
5720 '       EN COORDONNEES PARAMETREES (X,Y)
5740 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X) ;
5760 RETURN
5780 ' On ajoute 31 aux coordonnees pour obtenir la
5800 ' position en hexadecimal
5820 '
5840 ' **** INITIALISATION DES TOUCHES DE FONCTION *****
5860 PRINT CHR$(27)+"c4AA"        ' Forme du curseur
5880 A$=CHR$(27)+CHR$(48)        ' Partie commune de la chaine
5900 '
5920 B=98 : I1=49 : I2=48
5940 FOR I = 1 TO 8
5960 T$=A$+CHR$(B)+CHR$(I1)+CHR$(I2)
5980 PRINT T$;        ' On affecte aux touches b a i (designees
6000 B=B+1            ' par la variable B)
6020 I2=I2+1          ' Les fonctions codees en hexa
6040 NEXT I           ' par la chaine I1 I2 (de 10 a 17)
6060 B=106 : I1=48 : I2=52
6080 FOR I=1 TO 4
6100 T$=A$+CHR$(B)+CHR$(I1)+CHR$(I2)
6120 PRINT T$;        ' On affecte aux touches j à m
6140 B=B+1            ' les fonctions codees en hexa de 04 a 07
6160 I2=I2+1
6180 NEXT I
6200 RETURN
```

servir de clé de sortie du traitement, etc. Notre listing contient les instructions nécessaires à la gestion du curseur et des touches fonction d'une machine bien déterminée ; si le menu devait être utilisé sur un système différent, ces instructions devraient être modifiées (comme on l'a déjà dit, les codes utilisables sont spécifiques à chaque machine).

Pour faciliter l'adaptation du logiciel présenté (on parle alors de « portabilité »), les fonctions spécifiques étroitement liées au matériel hardware sont développées séparément, dans des routines appropriées.

Si l'on se réfère toujours au listing de la page 625 (menu principal), on remarque une partie initiale de commentaires, qui énumèrent les fichiers utilisés dans la procédure (lignes 4545 à 4555) : le nom de chaque fichier, l'unité disque (driver) où doit être chargée la disquette, le contenant (colonne DRV), la description du contenu, la longueur des enregistrements en octets et le nombre de champs constituant les enregistrements. Cette description initiale des fichiers de données nécessaires au programme se révélera très utile quand des modifications seront nécessaires.

La ligne 4560 appelle le sous-programme 6040, qui définit la forme du curseur et initialise les touches de fonction.

La chaîne BI$ (ligne 4580) contient les caractères ESCAPE [CHR$(27)], + et CHR$(7). ESCAPE provoque le passage en mode commande. Le deuxième caractère pour l'unité d'écran considérée commande l'effacement de l'écran. Le dernier émet un son afin d'appeler l'attention de l'opérateur ; il s'agit d'un « bip » qui, sur certaines machines, est généré par l'instruction spéciale BEEP. On remarquera que le code CHR$(7) permettant d'obtenir le bip est conforme aux tables ASCII (CTRL + G) et est donc présent sur presque toutes les consoles.

La présentation des différentes fonctions disponibles (affichage du menu) s'effectue dans le sous-programme 4960, qui commence par initialiser l'écran (PRINT BI$). La position sur l'écran des différentes lignes est paramétrée par le sous-programme 5700. En effet, la ligne 4980 comporte deux paramètres (X et Y) qui seront transmis à la routine. Les valeurs de X et Y indiquent respectivement la colonne et la ligne où doit commencer l'impression.

Le micro-ordinateur simplifie considérablement la gestion d'un stock.

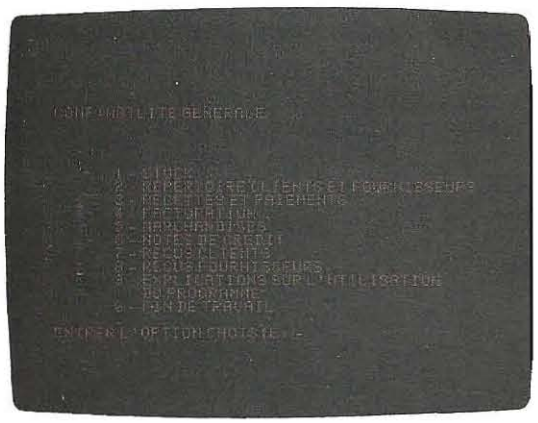


Olivetti

MENU DU STOCK

```
4500 '
4505 ' ** MENU DU STOCK **
4510 '    PROGRAMME MENU1
4512 '
4514 '
4516 ' Les touches fonction sont initialisees dans le programme MENU
4518 '
4520 BI$=CHR$(27)+"+"+CHR$(7)
4525 GOSUB 4630
4526 '
4530 ON V GOTO 4535, 4540, 4545, 4550, 4555, 4560, 4565, 4570, 4575, 4580
4535 RUN "A:CREART"          ' Creation d'articles
4540 RUN "A:MOUHEB"          ' Mouvements hebdomadaires
4545 RUN "A:PRMOUH"          ' Impression mouv. hebdo.
4550 RUN "A:TRANSF"          ' Transfert vers fichier archives
4555 RUN "A:IMPRES"          ' Impression fichier archives
4560 GOTO 4510               ' Ces 3 lignes sont
4565 GOTO 4510               ' prevues pour l'elargissement
4570 GOTO 4510               ' ulterieur de MENU1
4575 RUN "A:HLP2"
4576 '
4580 PRINT BI$
4585 X=30 : Y=10
4590 GOSUB 4870              ' Deplacement du curseur
4595 PRINT CHR$(27)+"G1"+"fin de travail"    ' message souligne
4600 PRINT CHR$(27)+"G0"                     ' video normale
4605 RUN "A:MENU"
4610 END
4615 '
4620 '
4625 ' ** AFFICHAGE DU MENU **
4630 '
4635 PRINT BI$
4640 X=2 : Y=1 : GOSUB 4870
4645 PRINT "Gestion du stock"
4650 X=15 : Y=6 : GOSUB 4870
4655 PRINT "1 - Creation et modification d'articles"
4660 Y=Y+1 : GOSUB 4870
4665 PRINT "2 - Mouvements chargement/dechargement hebdomadaires"
4670 Y=Y+1 : GOSUB 4870
4675 PRINT "3 - Impression mouvements hebdomadaires avec montants"
4680 Y=Y+1 : GOSUB 4870
4685 PRINT "4 - Transfert des mouvements vers fichier archives"
4690 Y=Y+1 : GOSUB 4870
4695 PRINT "5 - Impression du fichier archives avec montants"
4700 Y=Y+1 : GOSUB 4870
4705 PRINT "6 - ................"
4710 Y=Y+1 : GOSUB 4870
4715 PRINT "7 - ................"
4720 Y=Y+1 : GOSUB 4870
4725 PRINT "8 - ................"
4730 Y=Y+1 : GOSUB 4870
4735 PRINT "9 - Explications sur l'utilisation du programme"
4740 Y=Y+1 : GOSUB 4870
4745 PRINT "0 - Fin de travail"
4750 Y=1 : Y=Y+2
4755 GOSUB 4870
4760 PRINT "ENTRER L'OPTION CHOISIE:"
4770 A$=INPUT$(1)
4775 PRINT A$
4780 V=VAL(A$)
4785 IF V>9 GOTO 4820        ' ERREUR
4790 PRINT BI$
4795 IF V=0 THEN V=10 : GOTO 4840        ' Sortie
4800 X=5 : Y=6
4805 GOSUB 4870
4810 PRINT "PATIENCE ! On charge le programme"
4815 GOTO 4840
4820 PRINT CHR$(7), CHR$(7)      ' Traitement d'erreur
4825 X=42 : GOSUB 4870
```

```
4830 PRINT "CODE NON VALIDE"
4835 GOTO 4765
4840 RETURN
4845 '
4850 '
4855 ' ** POSITIONNEMENT DU CURSEUR **
4860 '    en coordonnees (X,Y)
4870 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X);
4875 RETURN
```



Menu principal : selon l'option choisie, l'un des menus secondaires est chargé et présenté.



On a choisi l'option 1 (magasin). La sélection suivante permet l'accès aux différentes procédures.

Pour préparer l'écriture à partir des coordonnées X, Y (colonne, ligne), on commencera donc par un appel au sous-programme 5700 de déplacement du curseur ; l'écriture de la chaîne d'en-tête s'effectuera ensuite par une instruction PRINT classique. Le déplacement du curseur se programme simplement en envoyant à l'écran la chaîne de commande (ligne 5740), terminée par le symbole ; qui empêche le retour à la ligne.

A la ligne 5450 figure l'instruction A$= INPUT$(1), permettant la saisie d'un unique caractère (le choix peut se faire seulement entre 0 et 9) ; le caractère ainsi acquis est transformé en numérique (ligne 5470) et contrôlé (ligne 5480). Si la valeur est correcte, on efface l'écran (ligne 5500), un message s'affiche (ligne 5560) et le contrôle est rendu au programme principal (instruction 4620).

La sélection du programme à charger est effectuée en fonction du choix de l'opérateur. Si, par exemple, la réponse est 1 (V=1), le programme MENU1 est chargé et exécuté.

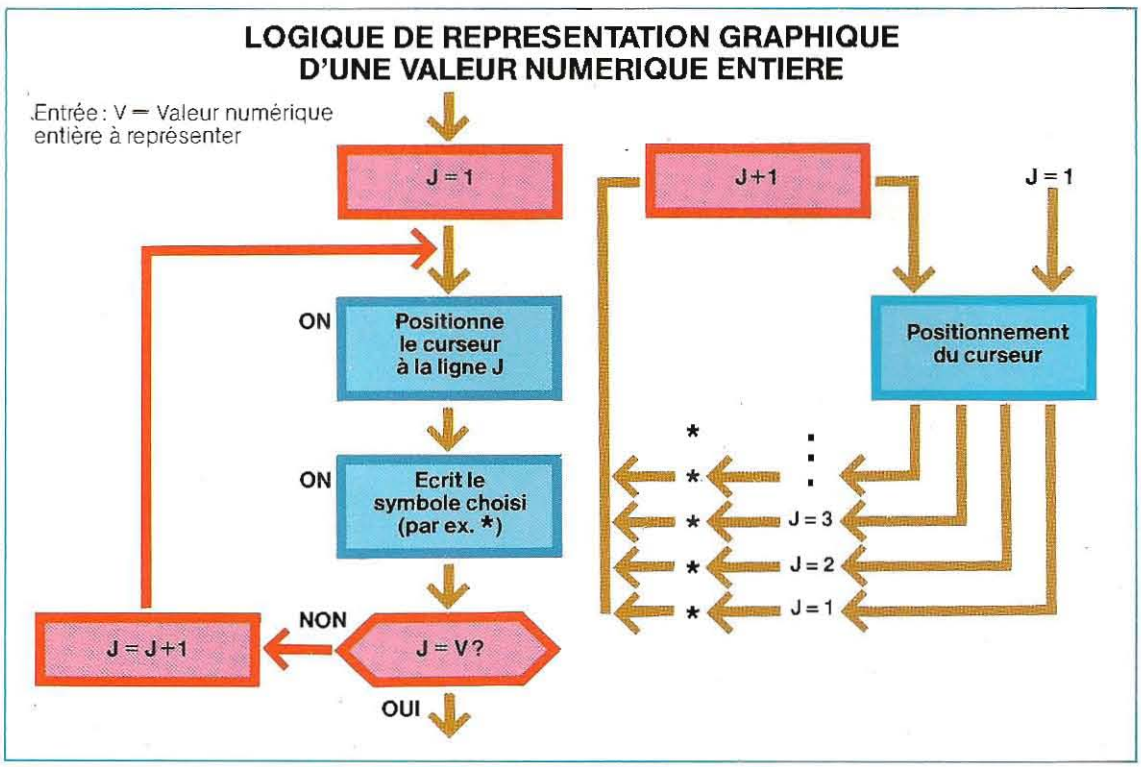
Ce programme (dont le listing est illustré sur la page 628 et ci-dessus) permet d'effectuer de nouveaux choix (lignes 4535 à 4575) et, à son terme, rappelle le menu précédent (ligne 4605) : il s'agit donc d'un menu secondaire. Il ne figure pas ici d'instructions pour obtenir une forme particulière du curseur, ni d'initialisation des touches fonction (opérations déjà exécutées dans le menu principal) ; par contre, le sous-programme de déplacement du curseur (ligne 4870) doit toujours être présent. On remarque que les deux menus (principal et secondaire) possèdent la même numérotation : l'un recouvre l'autre et il n'est pas nécessaire d'utiliser des numérotations différentes. Au contraire, en partant du même numéro de ligne, la recherche des erreurs est facilitée ainsi que les modifications apportées aux programmes.

Génération d'histogrammes

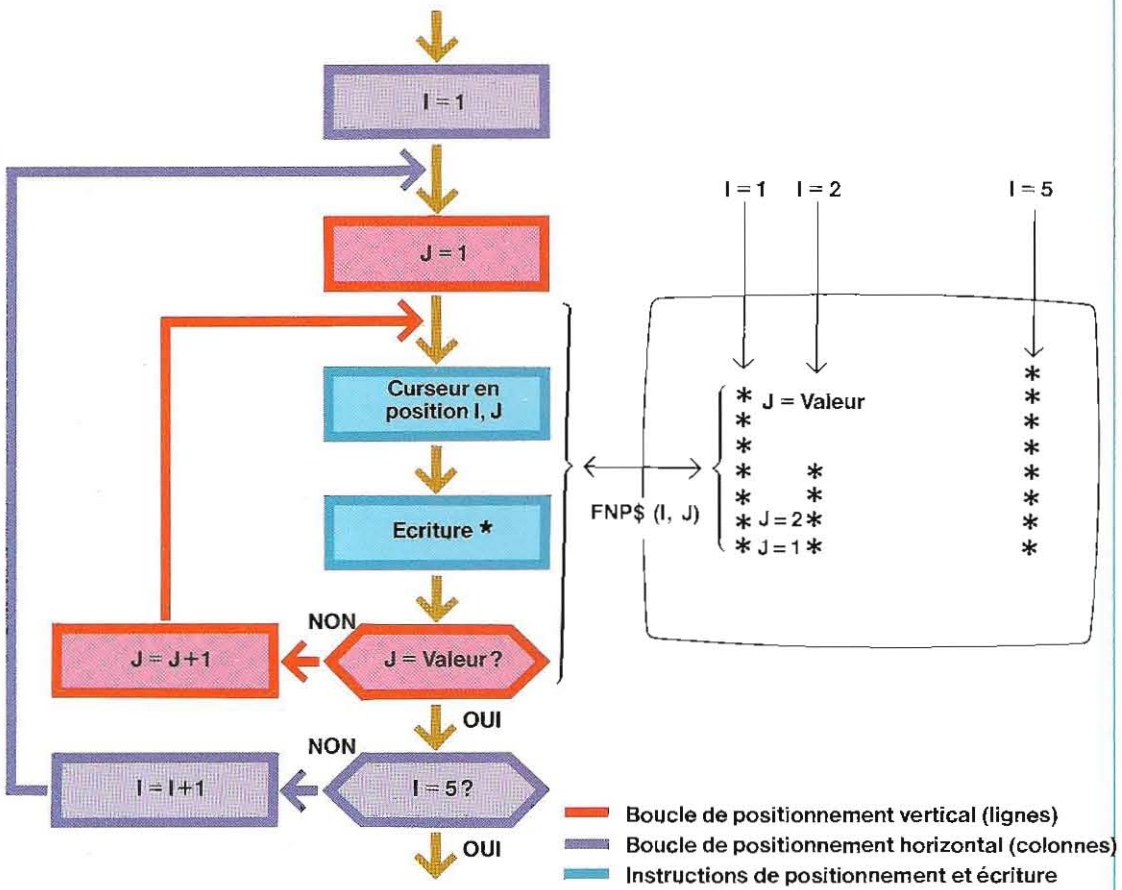
En utilisant la fonction d'adressage du curseur, il est possible de dessiner des histogrammes, même sur des écrans uniquement alphanumériques.
La procédure consiste à représenter sur l'écran une série de symboles, en nombre proportionnel (selon un facteur convenu) à la grandeur à représenter. Par exemple, en choisissant un facteur 10, la valeur 150 sera représentée par une colonne de 15 symboles (150/10).
Pour l'afficher à l'écran, il faudra écrire 15 fois le même symbole, en décrémentant chaque fois d'une unité le numéro de la ligne, tout en gardant inchangé le numéro de colonne. De cette manière, on « dessine » une petite colonne de hauteur proportionnelle à la valeur numérique à représenter.
Le schéma ci-dessous illustre le principe de cette logique de représentation.
La position du curseur peut se calculer en définissant une fonction contenant les codes d'adresage. Par exemple, la fonction :

```
FNP$(A,B)=CHR$(27)+CHR$(61)
          +CHR$(31+B)+CHR$(31+A)
```

positionne le curseur à la ligne B, colonne A. En effet, en ajoutant 31 (code hexa de 1) à la valeur décimale A (ou B), on obtient la valeur hexadécimale exacte de la position du curseur.
Comme l'écran alphanumérique possède généralement 80 colonnes et 24 lignes, les valeurs maximales envisageables sont A=80 et B=24, alors que le déplacement minimal sera pour l'axe X (indiqué par I) d'un caractère et pour l'axe Y (indiqué par J) d'une ligne. En d'autres termes, on ne peut afficher des symboles plus rapprochés que la distance existant entre deux lignes (axe Y) ou entre deux caractères (axe X) : sur un écran alphanumérique (non graphique) la possibilité d'adressage se limite à un nombre entier de lignes et de colonnes (caractères) : les positions intermédiaires n'existent pas.
Sur la page 631, la première colonne de l'histogramme, représentant la valeur 150, sera obtenue en positionnant le curseur sur 15 lignes successives (1, 2, 3, etc.) et en écrivant le symbole choisi, la valeur de la colonne (1) restant inchangée. Pour représenter, à côté de la précédente, une nouvelle valeur, il faut changer l'abscisse (colonne I) et répéter la boucle sur les lignes (J). En réunissant les



REPRESENTATION GRAPHIQUE SUR PLUSIEURS COLONNES



deux boucles, c'est-à-dire en les imbriquant l'une dans l'autre, on peut obtenir des histogrammes comprenant jusqu'à 80 colonnes disposées côte à côte.

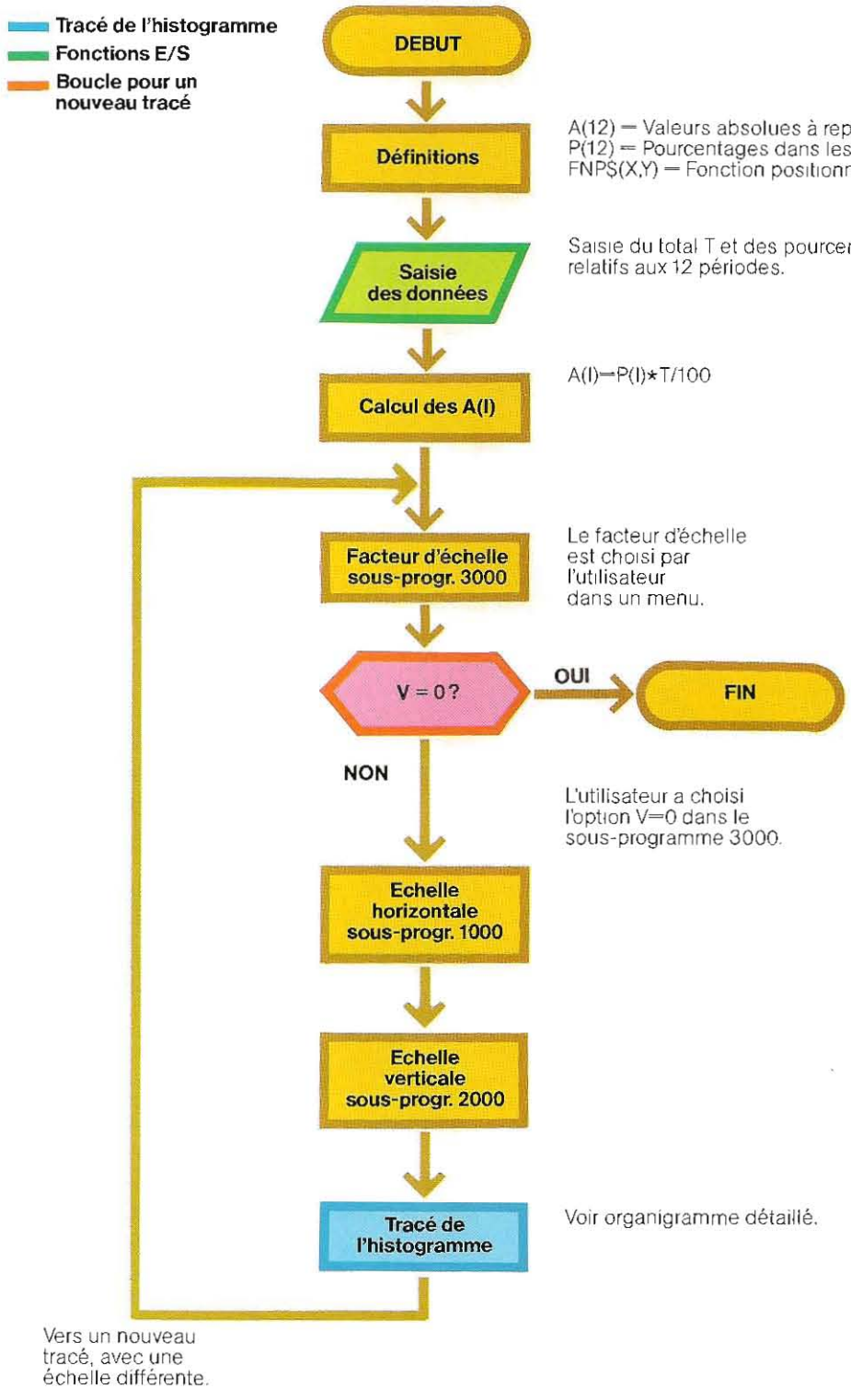
Le schéma de cette page illustre la préparation d'un histogramme représentant consécutivement cinq valeurs numériques. La première boucle, la plus externe, sélectionne la colonne de l'écran sur laquelle se fera la représentation graphique de la valeur. La boucle la plus interne détermine la hauteur de chacune des cinq petites colonnes, en répétant sur un nombre de lignes proportionnel à la valeur qui doit être représentée le symbole choisi (dans l'exemple le symbole *).

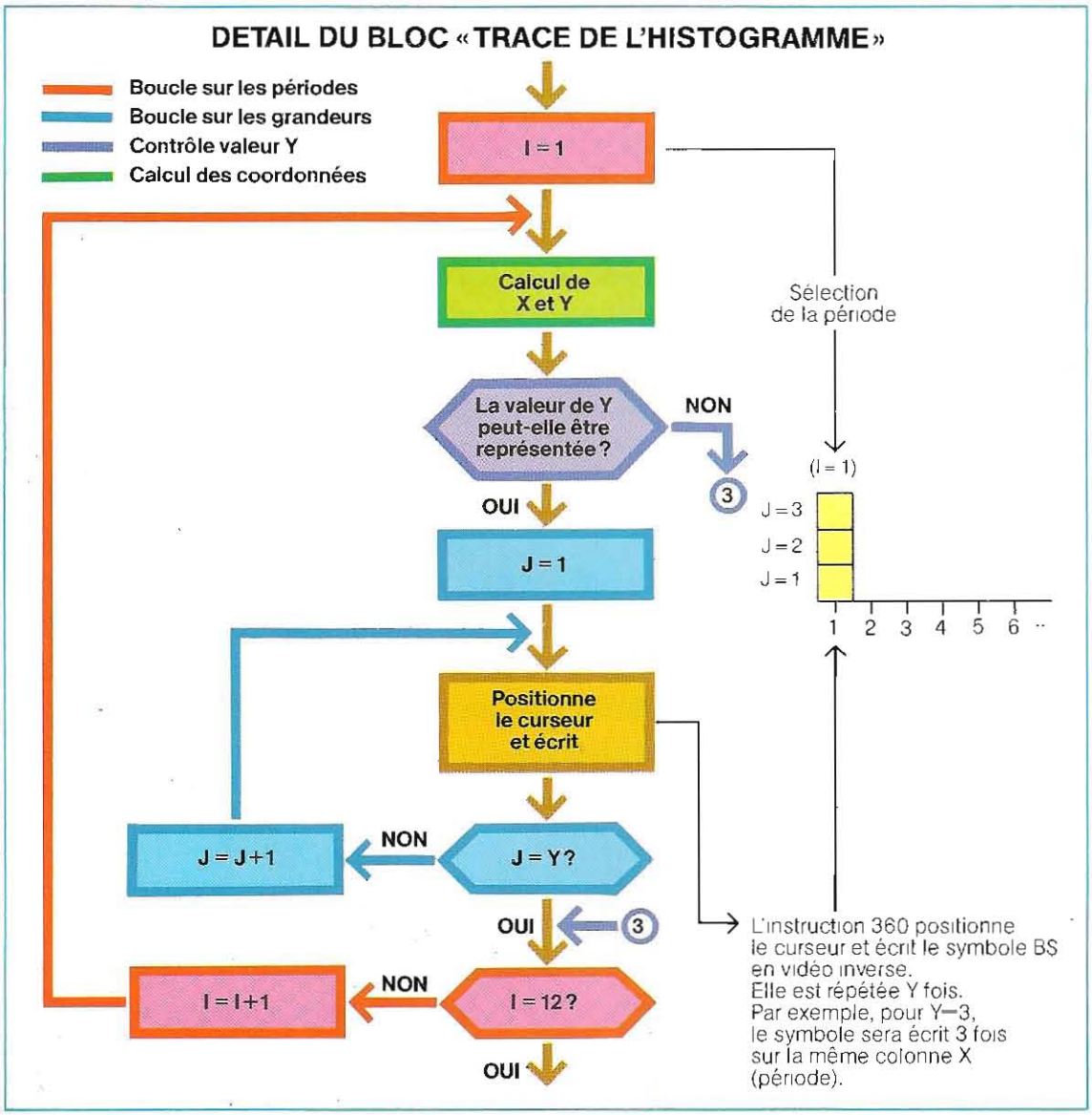
Sur le schéma, les coordonnées X et Y du curseur sont désignées par les symboles I et J, pour distinguer ce cas particulier d'adressage discret du cas général d'adressage par coordonnées.

Sur les écrans alphanumériques classiques, les coordonnées du curseur peuvent prendre comme valeurs des nombres entiers compris entre 1 et 24 (pour X), et entre 1 et 80 (pour Y), représentant effectivement à l'écran des lignes et des colonnes de caractères. En d'autres termes, on ne peut pas positionner le curseur au point 2.75 (lignes) : ou bien on choisit la position (ligne) 2 ou bien la position 3. En ce cas, on dit que l'adressage est discret (limité à un certain nombre de valeurs).

Sur les écrans graphiques, si les coordonnées restent entières, on n'est cependant plus lié aux lignes et aux colonnes ; on peut adresser les points de l'écran, caractérisés par un couple de coordonnées, comme les points d'une feuille de papier millimétré. Même dans ce cas, toutes les valeurs possibles de X et Y ne peuvent être représentées, car les points de l'écran, même si ce dernier est très grand,

PROGRAMME DE GENERATION D'HISTOGRAMMES





sont en nombre limité.

Les consoles graphiques professionnelles possèdent un écran à « haute résolution », dont la définition atteint, pour les modèles haut de gamme en noir et blanc, 4096 × 4096 points (ou pixels) adressables. Certaines consoles ouvrent des perspectives entièrement nouvelles grâce à l'apport de la couleur : si la définition de l'écran s'abaisse alors à 1024 × 1024, l'utilisateur peut afficher simultanément jusqu'à 4096 couleurs, choisies dans une palette de base de 4 millions de nuances.

Ces deux pages illustrent l'organigramme d'un programme de préparation d'histogrammes de 12 valeurs numériques. Cet exemple, de portée générale, peut s'appliquer à des cas pratiques de représentation graphique de la distribution d'une grandeur quelconque en 12 périodes. Il sera utilisé, par exemple, pour représenter des budgets, des objectifs de travail, etc. ; le nombre des périodes (12) permet l'écriture d'une représentation mensuelle sur l'étendue d'une année.

Les données requises par le programme en entrée (instructions 100 à 150) concernent la valeur totale (montant de la dépense, heures de travail, etc.) et le pourcentage de ce total prévu pour chacune des périodes.

Le programme calcule la valeur absolue (A(I))

EXEMPLE DE PREPARATION D'HISTOGRAMMES

```
30 ' PROGRAMME HISTO
40 DIM A(12), P(12)
50 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
60 C$=CHR$(27)+"+"          ' pour effacer l'ecran
70 B$=""                    ' motif composant l'histogramme
80 FI$=CHR$(27)+"G4"        ' passage en video inverse
90 FN$=CHR$(27)+"G0"        ' retour en video normale
100 INPUT "MONTANT TOTAL ";T
110 PRINT "ENTREZ LE POURCENTAGE DE CHAQUE PERIODE"
120 FOR I=1 TO 12
130 PRINT "Periode ";I
140 INPUT " Pourcentage: ";P(I)
150 NEXT I
160 EH=5        ' Facteur d'echelle horizontale (1 periode = 5 positions)
170 ' ** Calcul de la valeur de chaque periode **
180 FOR I=1 TO 12
190 A(I)=P(I)*T/100
200 NEXT I
205 PRINT C$          ' efface l'ecran
210 '
220 GOSUB 3000        ' choix de l'echelle verticale
230 IF V=0 GOTO 440   ' sortie du programme
240 '
250 EV=20/MX          ' calcul du facteur d'echelle verticale
255 '
260 GOSUB 1000          ' trace echelle horizontale
270 GOSUB 2000          ' trace echelle verticale
275 '
280 ' ** Boucle de trace de l'histogramme
290 FOR I=1 TO 12
300 X=1*EH
310 Y=A(I)*EV
320 IF Y<1 GOTO 390      ' Valeur non representable
330 FOR J=1 TO Y
340 L=22-J          ' La position J=1 se trouve en haut a gauche.
345 '                 Pour partir du bas, on se place en 22-J (la ligne
350 '                 22 est occupee par l'echelle horizontale)
360 PRINT FNP$(X,L); FI$; B$; FN$
370 NEXT J
380 X1=X-2 : PRINT FNP$(X1,L); P(I)
390 NEXT I
400 X=40 : Y=1 : PRINT FNP$(X,Y);
410 PRINT "Pour continuer, taper un caractere ":S$=INPUT$(1)
420 PRINT C$
425 '
430 GOTO 220
440 END
1000 ' ** Ecriture de l'echelle horizontale (12 periodes)
1010 NL=EH*12+1          ' Longueur de l'echelle
1020 G$=STRING$ (NL,"-")        ' Une ligne de - symbolise l'echelle horizontale
1030 X=1 : Y=22 ; PRINT FNP$(X,Y); G$
1040 Y=23
1050 FOR I=1 TO 12
1060 X=1*EH
1070 PRINT FNP$(X,Y); I;
1080 NEXT I
1090 RETURN
2000 ' ** Ecriture de l'echelle verticale
2010 X=1
2020 FOR I=1 TO 20          ' 20 = hauteur de l'ecran
2030 L=22-I
2040 PRINT FNP$(X,L); "I"        ' Une colonne de I symbolise l'echelle verticale
2050 NEXT I
2060 PRINT FNP$(X,1); "Total = ";T
2070 RETURN
3000 ' ** Choix de l'echelle verticale **
3010 X=2 : Y=1 : PRINT FNP$(X,Y);
3020 PRINT "CHOIX DE L'ECHELLE VERTICALE"
3030 X=15 : Y=6 : PRINT FNP$(X,Y);
```

```
3040 PRINT "1 - relative a la valeur maximale des pourcentages"
3050 Y=Y+1 : PRINT FNP$(X,Y);
3060 PRINT "2 - relative a la valeur totale"
3070 Y=Y+1 : PRINT FNP$(X,Y);
3080 PRINT "3 - autre echelle"
3090 Y=Y+1 : PRINT FNP$(X,Y);
3100 PRINT "4 - fin de travail"
3110 X=1 : Y=Y+2 : PRINT FNP$(X,Y);
3120 PRINT "ENTREZ L'OPTION CHOISIE";
3130 A$=INPUT$(1) : PRINT A$
3140 V=VAL(A$)
3150 ON V GOTO 3155, 3210, 3220, 3230
3155 ' *V=1 : valeur relative au pourcentage maximal *
3160 MX=0
3170 FOR I=1 TO 12
3180 IF A(I)>MX THEN MX=A(I)
3190 NEXT I
3200 GOTO 3230
3210 MX=T : GOTO 3230    ' *V=2 : valeur relative au total T
3220 INPUT "Valeur d'echelle choisie"; MX    ' *V=3 : valeur choisie
3230 RETURN
```

correspondant à chaque période (lignes 170 à 200), et fixe l'échelle horizontale à 5 (EH) : les colonnes de l'histogramme seront espacées de cinq unités.

Trois sous-programmes sont prévus. La routine 3000 met en évidence l'effet du changement d'échelle, en proposant à l'utilisateur trois types de représentation. Dans le premier cas, la hauteur maximale (vingt lignes en Y) est utilisée pour représenter la valeur maximale des A(I). Dans le second cas, c'est la valeur totale (T) qui est dessinée sur vingt lignes. Les valeurs seront ensuite représentées par un pourcentage de la hauteur de la colonne échelle.

Remarquons que cette routine 3000 représente un nouvel exemple de menu, qui permet la visualisation successive de différents modes de représentation.

Les lignes 400 et 410 constituent un exemple type de message suivi d'un temps d'attente (temporisation). Pour continuer le traitement qui suit l'affichage, il faut attendre que l'utilisateur ait lu les résultats montrés à l'écran. On réserve généralement deux ou trois lignes au dialogue avec l'utilisateur, et l'on fait figurer sur une de ces lignes le message choisi (ligne 410). L'attente se réalise par l'instruction S$= INPUT$(1) qui interrompt l'exécution du programme, jusqu'à l'entrée d'un caractère quelconque. Comme l'instruction supprime l'écho, le caractère n'apparaît pas à l'écran et la chaîne S$ ne sert qu'à réaliser l'attente.

Notre programme utilise comme symbole pour le tracé des histogrammes un espace, affiché en vidéo inverse (il se présente sous la forme d'un rectangle lumineux) ; le symbole est affecté à la variable B$ (ligne 70) et pour le remplacer, il suffit de changer l'affectation.

On définit la position du curseur et l'écriture du symbole par l'instruction 360. La fonction FNP$(X,L) positionne le curseur à la colonne X, ligne L ; le numéro de ligne est déterminé par l'expression 22-Y, de façon à tracer l'histogramme à partir de la ligne 22, dans la partie inférieure de l'écran. La variable FI$ contient les codes de commande pour passer en vidéo inverse (ligne 80) ; B$ est le symbole à écrire et FN$ contient les codes chargés de remettre l'écran en mode d'affichage normal (ligne 90). A l'extrémité d'une colonne représentant une valeur donnée, on écrit le pourcentage P(I) correspondant (ligne 380). On obtient l'écriture de cette valeur en laissant le curseur positionné sur la ligne L de sortie de la boucle, mais déplacé de deux colonnes à gauche (X1=X–2) afin d'aligner le nombre avec le sommet de la colonne.

Les codes employés ici se réfèrent à une machine particulière : pour d'autres modèles, il faut se conformer aux valeurs spécifiées dans les manuels d'utilisation.

Les sous-programmes 1000 et 2000 tracent les échelles horizontales et verticales. En ligne 1020, on a utilisé l'instruction STRING$(NL,"–"), qui génère la chaîne G$ de longueur NL, formée de symboles – concaténés ; l'écriture de cette chaîne donne une ligne horizontale pointillée. De plus, sur l'échelle horizontale sont reportées les valeurs des intervalles (de 1 à 12).

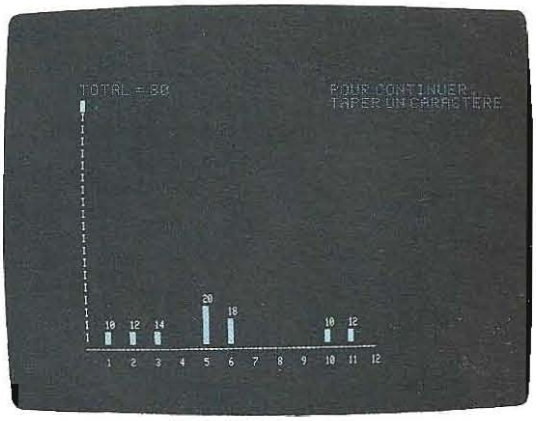
L'échelle verticale est tracée à l'aide d'une boucle (lignes 2020-2050) qui affiche le caractère « I ».

GENERATION D'HISTOGRAMMES A L'ECRAN

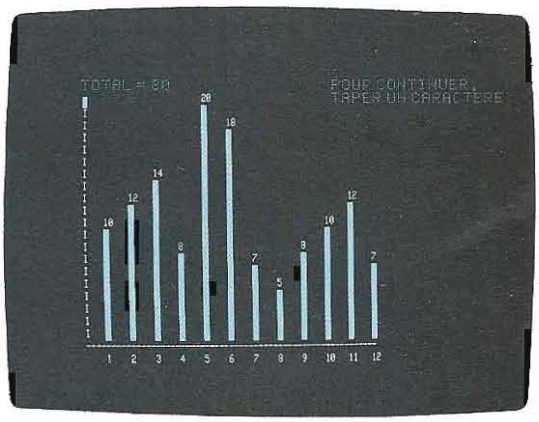
Sur cette page sont illustrés les écrans de la phase de saisie de données (ci-contre) et d'affichage des histogrammes (au-dessous), avec référence au programme illustré à la page 634. La phase de saisie est guidée par le programme lui-même, qui demande tour à tour les données nécessaires (lignes 100 à 150). Après la phase de calcul, le contrôle passe au sous-programme 3000 qui détermine l'échelle verticale à adopter.



Dans le premier cas, on a choisi l'échelle relative à la valeur totale Celle-ci occupe donc une hauteur de 20 lignes.
Chaque donnée numérique est représentée par une colonne de hauteur égale à un pourcentage (c'est la donnée elle-même) de la hauteur maximale disponible.
Notons que la différence entre les valeurs 10, 12 et 14, par exemple, n'est pas représentable, et que la période 12 doit avoir une valeur inférieure à 5.



Dans le deuxième cas, on a choisi l'échelle relative au pourcentage le plus fort, cette fois, la valeur maximale saisie (20) est représentée sur toute la hauteur disponible.
Les autres données sont représentées proportionnellement.
On obtient alors un étalement optimal des données sur l'écran.



Test 18

1 / Qu'appelle-t-on attributs d'un caractère ?

2 / Parmi ces instructions, lesquelles sont erronées ?

```
10 PRINT A,B,C
20 FOR I=1 TO 20 : PRINT "ESSAI" ; : NEXT I
30 PRINT A,TAB(60),B,TAB(100),C
```

3 / Quel est le résultat des instructions suivantes ?

```
10 FOR I=1 TO 30
20 PRINT I,"Données",A$
30 NEXT I
```

4 / En supposant que la fonction FNP$(X,Y) positionne le curseur à la colonne X et la ligne Y, quelles sont les instructions qui permettent d'écrire une chaîne quelconque de caractères en un point quelconque de l'écran ? Les coordonnées (ligne, colonne) et la chaîne seront saisies au clavier et le programme se terminera en lisant colonne = 0 ou bien ligne = 0.

5 / En affectant à la chaîne INV$ les codes de génération de caractères en vidéo inverse et à NOR$ les codes de retour à l'affichage normal, comme doit-on modifier l'instruction de la question précédente pour obtenir l'édition de la chaîne en affichage inversé ?

Les solutions du test se trouvent en pages 644 et 645.

Gestion des fichiers

Les fichiers reconnus par le Basic sont de deux types : séquentiels et en accès direct. Dans les fichiers séquentiels, les données sont écrites (et lues) les unes après les autres ; en d'autres termes, on ne peut accéder à une donnée sans avoir d'abord balayé toutes celles qui la précèdent. Ces types de fichiers constituent généralement le support de données entrées au clavier.
Les fichiers en accès direct (ou « Random »), par contre, permettent la lecture et l'écriture de données en adressant directement l'enregistrement intéressé.
La logique d'accès aux données d'un fichier est la même dans les deux cas et comprend la série suivante de fonctions :

- ouverture du fichier ;
- lecture et/ou écriture des données ;
- fermeture du fichier (à la fin de son utilisation).

Chaque fonction possède une syntaxe différente selon qu'elle s'adresse à un fichier séquentiel ou en accès direct. Elles seront donc examinées par la suite séparément avec une référence aux deux cas mentionnés ci-dessus.

Fonctions d'accès aux fichiers séquentiels

Un fichier séquentiel peut être ouvert de deux manières : en sortie (écriture par le programme), indiqué par le symbole O (Output) ou en entrée (lecture par le programme), indiqué par le symbole I (Input). Le type d'accès (I/O) sera spécifié dans l'instruction d'ouverture, qui doit être exécutée avant tout accès aux données.
Cette instruction est :

OPEN"TYPE",N,"NOM"

où
TYPE prend une des deux valeurs O ou I.
N représente une valeur numérique, appelée "numéro d'unité logique", que l'instruction d'ouverture associe au fichier.
Par la suite, toutes les opérations d'accès aux données se réfèreront à ce numéro pour désigner le fichier.
NOM est le nom avec lequel le fichier a été créé.

Par exemple, l'instruction

OPEN"I",2,"A:ESSAI"

ouvre le fichier nommé ESSAI, résidant sur la première unité disque (A:), en mode lecture (I), et lui associe le numéro d'unité logique 2. Dans les opérations de lecture de données, ce fichier sera ensuite désigné par le numéro 2. Sous certains systèmes, le numéro du fichier doit être précédé du symbole #. En ce cas, on modifie l'instruction d'ouverture de la manière suivante :

OPEN"I",#2,"A:ESSAI"

La modalité d'ouverture du fichier dépend des fonctions que le programme doit exécuter. Pour acquérir des données à parti du fichier, il faut l'ouvrir en mode INPUT : en effet, il s'agit d'une saisie de données par le programme. A l'opposé, pour écrire dans un fichier, on l'ouvrira en mode OUTPUT, car le fichier, pour le programme, est équivalent à un dispositif de sortie.
Après avoir accédé aux données, le fichier doit être fermé. L'instruction à utiliser est

CLOSE N

N étant la valeur numérique associée au fichier en phase d'ouverture. Par exemple, pour fermer le fichier précédent, on écrit l'instruction CLOSE 2.
Entre les instructions OPEN et CLOSE, on peut exécuter tooutes sortes de fonctions d'accès aux données ou de contrôle de l'état du fichier.

Voici la liste des instructions Basic opérant sur des fichiers séquentiels :

INPUT, LINE INPUT, WRITE, PRINT, PRINT USING, EOF, LOC, KILL, NAME.

INPUT. Cette instruction,dont la syntaxe est :

INPUT#N, VARIABLE

lit une donnée du fichier numéro N et l'affecte à la variable spécifiée. Par exemple, à l'exécution de la ligne INPUT #1,B$, le contenu de l'enregistrement « courant » (dans les fichiers séquentiels, la lecture et l'écriture débutent toujours à l'enregistrement 1 et continuent en séquence) du fichier 1 est lu et affecté à B$. Dans cette instruction, le symbole # ne peut pas être omis.

LINE INPUT. La syntaxe complète de l'instruction est :

LINE INPUT#N,VARIABLE

Elle lit une ligne entière de caractères (jusqu'à 254). Cette instruction trouve une utilisation particulière lors de la lecture d'un programme (mémorisé sur disque en format ASCII) par un autre programme. Le programme qui s'exécute voit le premier (celui sur disque) comme un fichier quelconque de données et peut donc le lire. Le seul obstacle réside dans la longueur variable des lignes du programme mémorisé en ASCII : chacune tient un nombre de caractères dépendant de l'instruction écrite ; on reconnaît le point où la ligne prend fin en lisant séquentiellement les enregistrements (caractères) jusqu'à rencontrer le code CR (Carriage Return) qui termine la ligne Basic.

WRITE. La fonction WRITE, dont la syntaxe est

WRITE#N,VARIABLE

écrit, dans le fichier identifié par le numéro N, la valeur contenue dans la variable spécifiée. Pour utiliser l'instruction, le fichier doit être ouvert en mode « O » (il s'agit d'une sortie du programme). L'instruction WRITE insère automatiquement une virgule entre les valeurs s'il y a plus d'une variable à transférer.

PRINT ET PRINT USING. La syntaxe de ces instructions est la suivante :

PRINT#N, VARIABLE
PRINT#N,USING...

Elles ont la même signification que les fonctions d'impression du même nom. Le fichier N est ici traité comme un périphérique.

EOF(N). Il s'agit d'une fonction de contrôle du fichier séquentiel numéro N, qui fournit la valeur – 1 (vrai) quand, lors de la lecture, on rencontre le code particulier signalant la fin du fichier. La fonction EOF sera utilisée en particulier dans la boucle de lecture d'un fichier séquentiel quelconque : elle permet le déroulement de la boucle, en lisant tous les enregistrements écrits, sans qu'il soit nécessaire de connaître au préalable la longueur du fichier. Par exemple, le programme

```
10 OPEN"I",1,"A:ESSAI"
20 INPUT#1,A$,B$,C$
30 IF EOF(1)GOTO100
40 GOTO 20
100 PRINT"FIN DE LECTURE DU FICHIER"
110 END
```

exécute la boucle constituée des instructions 20, 30 et 40 jusqu'à la rencontre du code End Of File. La fonction EOF est alors vraie et l'exécution se poursuit avec l'instruction 100. Si la ligne 30 était omise, la boucle ne serait plus contrôlée et tenterait de lire des renseignements au-delà des limites du fichier, provoquant l'apparition d'erreurs système et l'arrêt de l'exécution.

LOC(N). Fournit le nombre de secteurs lus ou écrits dans le fichier N à partir de son ouverture (rappelons qu'un secteur est constitué de 128 ou 256 octets).

KILL. L'instruction KILL"NOM" efface le fichier NOM, qui doit avoir été préalablement fermé. Elle est la même pour tous les types de fichiers (séquentiel, en accès direct au programme). Le fichier n'est pas physiquement « effacé » mais l'espace qu'il occupait est rendu disponible, et il pourra être écrasé par un nouveau fichier.

NAME. Cette fonction est utilisée pour renommer un fichier. L'instruction

NAME"NOM1" AS "NOM2"

remplace l'ancien nom (NOM 1) par le nouveau (NOM 2). Par exemple, la ligne NAME"OLDFIL" AS "NEUF" renomme le fichier OLDFIL sous le nom NEUF.

La page 640 illustre l'organigramme d'un programme qui utilise les instructions mentionnées afin d'écrire une série de données saisies au clavier sur un fichier séquentiel. Au terme de la saisie, le programme donne la longueur du fichier (en secteurs), puis relit et imprime les données saisies.

L'organigramme, qui peut paraître succinct par rapport à ceux présentés jusqu'ici, reflète la forme couramment utilisée dans la pratique. Pour être lisibles, les organigrammes d'un programme doivent synthétiser les procédures à accomplir, même sans détailler les moyens pour arriver au résultat : ce niveau de détail sera atteint dans la phase d'écriture des instructions proprement dites.

Par exemple, le pavé 500 de l'organigramme page 640 teste la fin de la phase de saisie. Les modalités de ce contrôle ne sont pas spécifiées de manière à s'appliquer à n'importe quel fichier. Dans le cas présent, chaque donnée se compose de quatre valeurs numériques et d'une chaîne de caractères ; le test portera sur cette dernière.

De même, pour relire le fichier qui vient d'être créé (et est donc ouvert en mode écriture), il faut tout d'abord le refermer, puis l'ouvrir à nouveau en mode lecture.

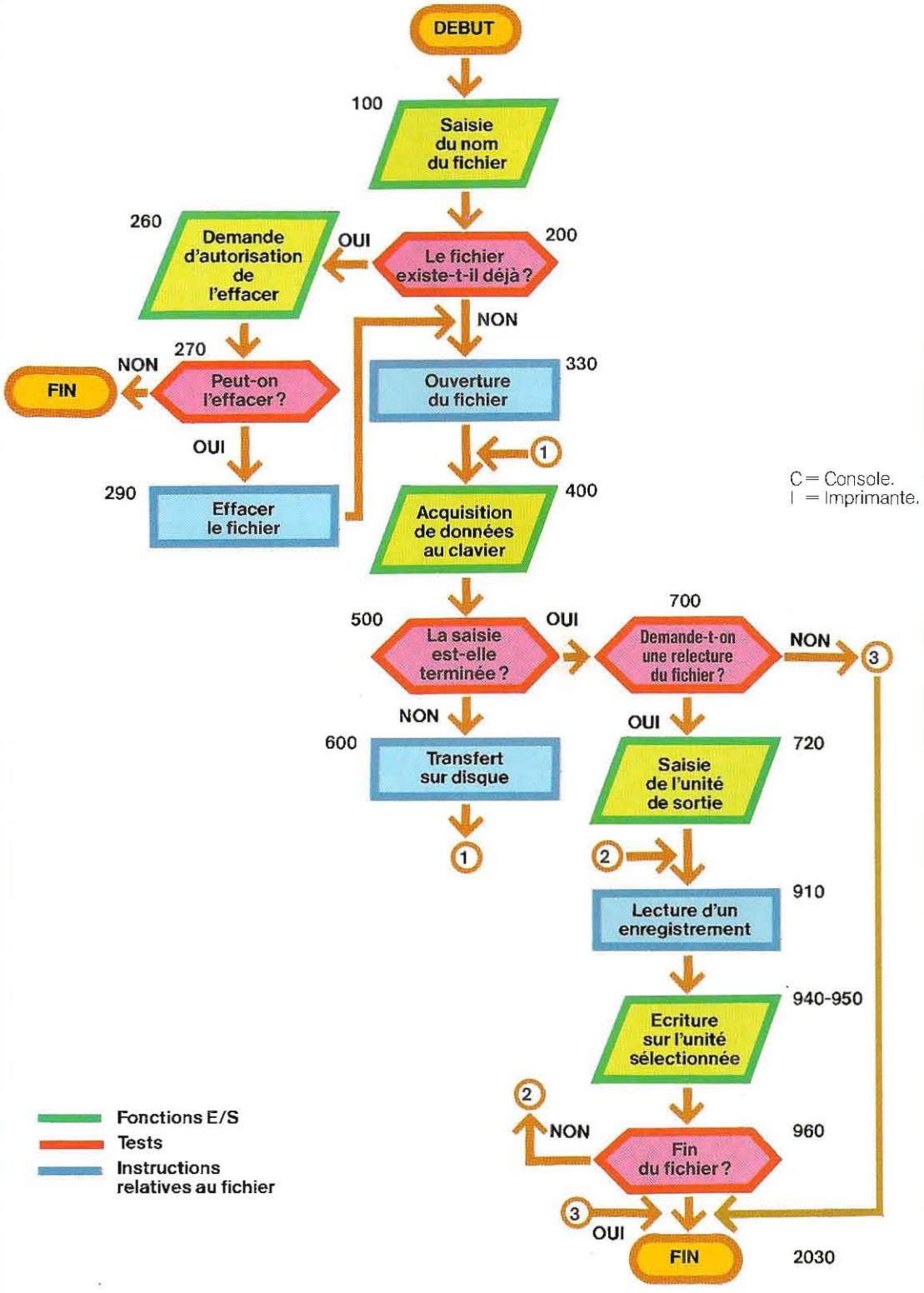
Pour contrôler si le fichier existe déjà, il suffit de l'ouvrir en lecture ; s'il n'est pas présent sur le disque, le système détecte une erreur, que le programme se charge de gérer.

Le programme est listé à la page 641. L'organigramme peu détaillé de la page 640 laisse une grande liberté de choix quant au moyen de codifier le programme. Le listing présenté propose une des solutions possibles, commentée et simplifiée intentionnellement. On peut toutefois obtenir les mêmes résultats avec des programmes plus compacts, mais moins compréhensibles.

Insertion de données dans un fichier séquentiel

Les enregistrements d'un fichier séquentiel ne sont pas directement adressables, on ne

EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL



EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL

```
10 ' *EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL *
20 '   PROGRAMME SEQ
30 '
40 ON ERROR GOTO 1000
50 '
100 INPUT "NOM DE FICHIER";NM$
110 '
200 ' *Verification de son existence *
210 OPEN "I",1,NM$ 'On associe le numero 1 au fichier
220 '  Si le fichier n'existe pas, le systeme diagnostique une erreur
230 '  et l'execution se poursuit en 1000 (cf. ligne 40)
240 '
250 PRINT "LE FICHIER EXISTE DEJA SUR DISQUE"
260 INPUT "DOIT-ON L'EFFACER (OUI/NON)"; REP$
270 IF REP$<>"OUI" GOTO 2020
280 CLOSE 1
290 KILL NM$          'On ferme et on efface le fichier
300 OPEN "O",1,NM$ 'On ouvre le nouveau fichier en ecriture
310 PRINT
320 PRINT "LE FICHIER EST OUVERT EN ECRITURE"
330 PRINT "Pour terminer la saisie, taper FIN"
340 PRINT
400 INPUT "DONNEE (1 chaine et 4 nombres)",A$,N1,N2,N3,N4
500 IF A$="FIN" GOTO 700
600 WRITE £1,N1,N2,N3,N4,A$ 'Ici, le symbole £ equivaut à #
610 GOTO 400
620
630 '*fin de la saisie *
700 INPUT "VOULEZ-VOUS UNE RELECTURE DU FICHIER (OUI/NON)";REP$
710 IF REP$="NON" GOTO 2020
720 PRINT "Ecriture sur quel peripherique (C = console, I = imprimante)"
730 INPUT REP$
740 IF REP$ <> "C" AND REP$ <> "I" THEN PRINT "ERREUR": GOTO 720
800 CLOSE 1                    'Fermeture du fichier en ecriture
810 OPEN "I",1,NM$             'Reouverture en lecture
900 K=0                        'Initialisation du compteur d'enregistrements
910 INPUT £1,N1,N2,N3,N4,A$ 'Lecture d'un enregistrement
920 K=K+1                      'Incrementation du compteur
930 PRINT " ENREGISTREMENT No ";K
940 IF REP$="C" THEN PRINT N1,N2,N3,N4,A$
950 IF REP$="I" THEN LPRINT "ENREGISTREMENT No";K: LPRINT N1,N2,N3,N4,A$
960 IF EOF(1) THEN 2000 'Fin du fichier
970 GOTO 910
980 '
1000 ' *Gestion des erreurs
1010 PRINT " LE FICHIER N'EXISTE PAS SUR DISQUE, IL DOIT ETRE CREE "
1020 RESUME 300
1030 '
2000 ' *SORTIE DU PROGRAMME *
2010 PRINT " LE FICHIER CONTIENT ";K;" ENREGISTREMENTS "
2020 PRINT " ***** FIN D'EXECUTION ***** "
2030 END
```

```
ENREGISTREMENT No 1
 12            45            78            23            MICHEL
ENREGISTREMENT No 2
 55            70            5             789           CHARLOTTE
ENREGISTREMENT No 3
 11            22            33            44            GEORGES
ENREGISTREMENT No 4
 99            88            77            66            YANICK
LE FICHIER CONTIENT 4 ENREGISTREMENTS
***** FIN D'EXECUTION *****
```

peut donc pas ajouter de données en les écrivant simplement à la fin du fichier. Il faut alors effectuer une copie du fichier à modifier, en ajoutant les nouveaux enregistrements en phase d'écriture de la copie. Plus précisément, on procède comme suit :

1 / Ouverture du fichier d'origine en mode « I » ; (il sera lu par le programme.
2 / Ouverture d'un second fichier en mode « O » ; il contiendra les enregistrements recopiés sur le premier, plus les nouvelles données.
3 / Transfert du fichier d'origine vers le second ; on crée ainsi une copie des données dans le nouveau fichier, qui est en phase d'écriture.
4 / Ajout des nouveaux enregistrements au second fichier.
5 / Le premier fichier, n'étant plus nécessaire, est d'abord fermé puis effacé.
6 / On attribue le nom du premier fichier au second ; on obtient ainsi un fichier ayant le même nom qu'à l'origine, mais de longueur supérieure.

Fonctions d'accès aux fichiers en accès direct

Les fichiers en accès direct sont les plus utilisés, car ils permettent d'adresser librement l'enregistrement auquel on s'intéresse. De plus, ils sont mémorisés en binaire, et occupent donc, sur le disque, moins d'espace que les fichiers séquentiels, mémorisés en ASCII.
Lors de leur ouverture, on ne déclarera pas le type d'accès, car ils peuvent être adressés indifféremment en lecture ou en écriture.
Pour l'échange de données avec ces fichiers, le système d'exploitation utilise une mémoire tampon. Avant d'exécuter l'écriture, les valeurs à transférer doivent transiter d'abord par cette mémoire tampon, qui sera ensuite déchargée sur disque.
Inversement, les données à lire sont extraites du disque et stockées dans le buffer, puis transmises au programme qui les demande par des instructions appropriées. Il est important de remarquer que cette mémoire tampon, qui constitue une variable particulière dont le nom est défini dans le programme d'application, ne peut être utilisée que pour échanger des données avec le disque. Avant toute autre utilisation, les données qui y sont contenues doivent être transférées dans un autre espace mémoire identifié par un nom différent.
Les fonctions et instructions utilisées dans la gestion des fichiers en accès direct dont les suivantes :

OPEN, FIELD, CLOSE, PUT, GET, LOC, KILL, NAME.

Analysons dans le détail la syntaxe et l'effet de chaque instruction. Pour les instructions KILL et NAME, on se référera à ce qui a été dit au sujet des fichiers séquentiels.

OPEN. En voici la syntaxe :

```
OPEN"R",#N,"NOM",L
```

où
R indique qu'il s'agit d'un fichier en accès direct ;
N représente le numéro d'unité logique attribué au fichier par le programme (le symbole # n'est pas toujours nécessaire) ;
NOM est le nom du fichier, et peut indiquer aussi l'unité disque sur laquelle il se trouve (par exemple A:ESSAI) ; quand l'unité disque n'est pas spécifiée, le système prend par défaut la dernière unité disque (driver) sélectionnée.
L indique la longueur en octets des enregistrements logiques constituant le fichier.

Par exemple, l'instruction :

```
OPEN"R",3,"B:TEST,64
```

ouvre un fichier en accès direct appelé TEST, sur l'unité-disque B. Le numéro logique 3 est associé au fichier, qui est structuré en enregistrements logiques de 64 octets.
L'instruction OPEN peut aussi servir à créer des fichiers.
Ce système, avantageux par certains côtés, peut poser quelques problèmes. Les programmes les plus complexes manipulent souvent des fichiers stockés sur plusieurs disques différents. Si l'utilisateur oublie de remplacer le disque, l'instruction OPEN, qui sur le bon disque aurait seulement ouvert le fichier, en créera alors un nouveau, avec le même nom mais sur le mauvais disque.
Le programme de la page 641 illustrait une

méthode pour éliminer tout risque d'erreur : on ouvre tout d'abord le fichier comme un fichier séquentiel, en mode INPUT. S'il n'existe pas, cette instruction provoque une erreur système et arrête l'exécution du programme. Si le fichier existe, le système ne signale aucune erreur. Il faut alors fermer le fichier en annulant l'ouverture en mode séquentiel (uniquement faite pour cette vérification), et l'ouvrir à nouveau, en mode « R » (accès direct).
Voici un programme simple, illustrant la technique exposée :

```
10  ' Ouverture d'un fichier en accès direct
20  ON ERROR GOTO 1000
30  OPEN"I",1,"ESSAI"
40  CLOSE1
50  PRINT "Le fichier existe"
60  OPEN"R",1,"ESSAI",64
70  '
80  ' Programme d'application
90  '
100  END
1000  ' ERREUR
1010  PRINT "Le fichier n'existe pas"
1020  INPUT "Doit-il être créé ?";R$
1030  IF R$<>"OUI" THEN STOP
1040  RESUME 60
```

Dans les programmes d'application, les instructions relatives au fichier (lignes 1010, 1020, 1030 et 1040) seront conditionnées par le type d'erreur. A l'apparition d'une erreur système, la variable ERR contiendra le code relatif à l'erreur, et ERL le numéro de ligne qui l'a engendré (ERR et ERL sont des noms réservés du système).
Par exemple, en supposant que le système d'exploitation réponde ERR=53 (c'est le cas en CP/M), le programme précédent doit être modifié de la manière suivante :

```
1000  ' ERREUR
1010  IF ERR=53 AND ERL=30 GOTO 1030
1020  ON ERROR GOTO 0
1030  PRINT "Le fichier n'existe pas"
1035  INPUT "Doit-il être créé ?";R$
1040  ' IF R$<>"OUI" THEN STOP
1050  ' RESUME 60
```

La ligne 1010 contrôle le type d'erreur et son origine ; si la cause de l'erreur n'est pas l'absence du fichier, le contrôle passe à la ligne 1020, qui réactive le mécanisme de gestion des erreurs dans le système ; le diagnostic habituel est émis et le programme s'arrête.

FIELD. Attribue à un fichier (préalablement ouvert) la mémoire tampon à utiliser dans les opérations de lecture et d'écriture. La syntaxe est la suivante :

```
FIELD 1, 64 AS B$
```

Cette instruction réserve la variable B$ au fichier numéro N, qui se compose d'enregistrements de 64 octets.
La mémoire tampon ainsi définie sera le « véhicule » d'échange de données avec le disque ; B$ ne peut plus servir à d'autres fins et, en particulier, ne peut pas être utilisée par des instructions d'INPUT et d'affectations classiques. Le transfert successif des données depuis le buffer vers le programme s'effectuera par les instructions LSET et RSET.
L'enregistrement logique peut être subdivisé entre plusieurs mémoires tampon.
Par exemple :

```
FIELD 1,12 AS N$, 30 AS C$, 20 AS V$, 2 AS F$
```

L'enregistrement (64 octets) sera réparti dans quatre buffers : N$ de 12 octets, C$ de 30 octets, V$ de 20 octets et F$ de 2 octets.

CLOSE. Fonctionne comme dans le cas des fichiers séquentiels : en exécutant l'instruction CLOSE#N, le fichier numéro N (précédemment défini par une instruction OPEN) est déclaré fermé et son contenu n'est plus accessible (jusqu'à une nouvelle ouverture du même fichier). L'instruction CLOSE libère les mémoires tampon intermédiaires attribuées au fichier ; elles pourront donc être utilisées de nouveau pour un autre fichier.

PUT. Instruction de transfert du contenu des mémoires tampon vers le disque. Il faut déclarer le fichier sur lequel on souhaite écrire, en spécifiant son numéro d'unité logique (défini par l'instruction OPEN) et l'emplacement (le numéro d'enregistrement logique). Par exemple, l'instruction

```
PUT#1,75
```

écrit sur l'enregistrement 75 du fichier 1 (dans certains cas le symbole # peut être omis).

Solutions du test 18

1 / Les attributs sont des indicateurs, stockés en mémoire en même temps qu'un caractère, et qui définissent certaines particularités de ce caractère. Pour cette raison, la mémoire associée à l'écran est plus étendue que celle qui serait strictement nécessaire à la définition en code ASCII des 24 × 80 caractères qui le composent.

2 / Les lignes 20 et 30 sont incorrectes. La ligne 20 affiche vingt fois une chaîne longue de 5 caractères (ESSAI), donc un total de 100 caractères sur la même ligne de l'écran. Ce dernier ne peut normalement contenir que 80 caractères par ligne. La ligne 30 contient le même type d'erreur : TAB(100) fait débuter l'affichage de la variable C en colonne 100, qui n'existe pas à l'écran (l'instruction serait valide s'il s'agissait de l'imprimante, LPRINT remplaçant alors PRINT).

3 / La boucle affiche une suite de données sur chacune des lignes de l'écran, jusqu'à la ligne 30. Puisque l'écran ne possède normalement que 24 lignes, les premières valeurs seront perdues puisqu'elles seront repoussées vers le haut (en dehors de l'écran) pour laisser place aux dernières lignes.

4 / Voici l'illustration du programme :

```
10 ' Solution question 4
20 ' Lecture des données au clavier
30 INPUT " Chaîne à écrire ";A$
40 INPUT " Numéro de colonne ";X
50 IF X=0 GOTO 160
60 INPUT " Numéro de ligne ";Y
70 IF Y=0 GOTO 160
80 '
90 ' Contrôles
100 '
110 K=LEN(A$) ' Longueur de la chaîne à écrire
120 IF (K+X)>80 THEN PRINT "ERREUR": GOTO 40
130 '
140 PRINT FNP$(X,Y);A$
150 GOTO 30
160 END
```

5 / L'instruction 140 devient :

```
140 PRINT   FNP$(X,Y);   INV$;   A$;   NOR$
            (Position)   (Vidéo inverse)   (Donnée)   (Retour affichage normal)
```

L'instruction pourrait être aussi écrite sur plusieurs lignes (le symbole ; supprime le retour en début de ligne). Elle pourrait, par exemple, être remplacée par :

```
140 PRINT FNP$(X,Y); 'Position
142 PRINT INV$;      'Affichage inversé
144 PRINT A$;        'Chaîne à écrire (donnée)
146 PRINT NOR$       ' Affichage normal
```

De cette façon, on peut conditionner certains ordres d'éditions, par la valeur d'un indicateur, afin d'obtenir des présentations différentes selon la valeur du flag. Par exemple, les instructions qui suivent génèrent l'édition en vidéo inverse dans le cas où FLAG=1 :

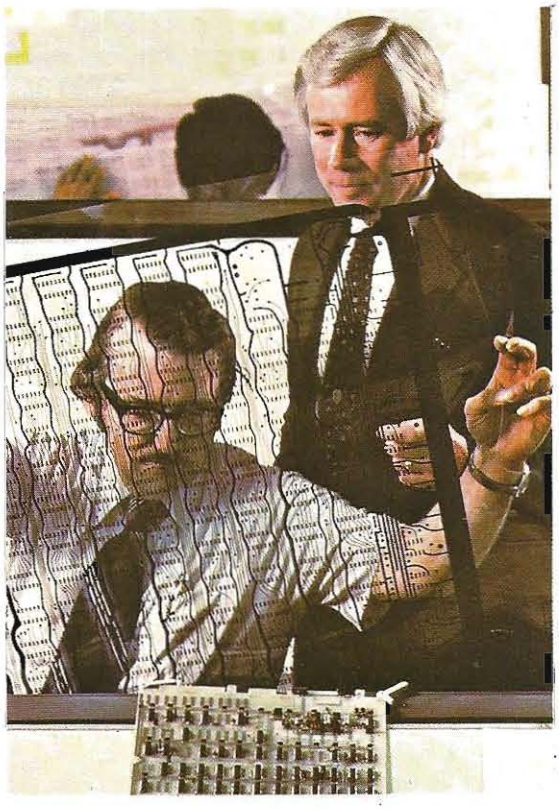
```
140 PRINT FNP$(X,Y);
142 IF FLAG=1 THEN PRINT INV$;
144 PRINT A$;
146 IF FLAG=1 THEN PRINT NOR$
148 IF FLAG<>1 THEN PRINT
```

La ligne 146 ramène en affichage normal et effectue le retour en début de ligne (pas de ;) dans le cas où FLAG=1. Dans le cas contraire (FLAG<>1), une ligne ne contenant que l'instruction PRINT est nécessaire pour obtenir le retour, qui n'avait pas été effectué par la ligne 144.

La taille d'un fichier en accès direct n'a pas besoin d'être définie a priori ; le système prend en compte les nouveaux enregistrements au fur et à mesure de leur écriture, et met à jour les pointeurs du fichier.
En général, ce procédé ne pose aucun problème, sauf si l'on modifie la taille des fichiers plusieurs fois. Au début, les fichiers sont ouverts et occupent physiquement des zones contiguës du disque. Si l'on désire rallonger le premier au-delà d'une certaine limite, les nouveaux enregistrements logiques ne pourront se placer de façon contiguë à la suite des précédents. Aussi, l'extension du premier fichier doit-elle se faire en un emplacement séparé. Les différentes parties de chaque fichier seront retrouvées par le système d'exploitation grâce à une table des extensions.
Entièrement gérée par le système d'exploitation, cette méthode s'avère parfaitement satisfaisante sur les micro-ordinateurs. En revanche, lorsqu'on s'adresse à de grosses machines, et pour des applications manipulant de très longs fichiers, il vaut mieux limiter cette dispersion des données.
On peut envisager deux solutions :

- copier périodiquement le disque contenant les fichiers sur un nouveau disque. A l'occasion du transfert, le système assure la contiguïté physique de tous les enregistrements d'un même fichier (extensions comprises) ;
- créer dès le début des fichiers de longueur égale à la taille maximum prévue ; ceci implique d'écrire d'une valeur de fin de fichier dans le dernier enregistrement prévu.

Projet de circuit imprimé.



Marka

Quelques kilo-octets pour jouer

Au début des années soixante-dix, alors que l'intérêt pour les OVNI tournait à la psychose et que l'on parlait tant des petits hommes verts, de leur présence présumée et de rencontres du troisième type, le premier « alien » véritable arriva sur terre.
Il s'appelait « Pong » et c'était un ***jeu vidéo*** *qui, grâce à un petit ordinateur, simulait pour la première fois une partie de ping-pong, en la représentant sur un moniteur de télévision.*
Ses apparences n'avaient rien d'anthropomorphique, mais il s'entendit immédiatement avec les habitants de notre planète, instaurant avec eux un rapport de paix et d'amitié. Il attira d'abord l'attention de cette faune humaine bariolée qui fréquente les salles de jeux, celle des « arcades » aux Etats-Unis, au point que les machines à sous furent en permanence saturées par un flot continu de pièces de monnaie : il connut un incroyable succès et sa progéniture, depuis l'Amérique, colonisa le globe entier.

Le premier à constater cet engouement fut précisément l'inventeur de Pong : Nolan Bushnell. Il ne perdit pas de temps et, poussé par un esprit d'entreprise authentique et génial, fonda Atari, première grande société de jeux vidéo de taille industrielle.
L'expansion extraordinaire des jeux électroniques conduisit les dirigeants d'Atari à lancer sur le marché une nouvelle version de Pong, destinée aux familles, qui utiliseraient justement leur téléviseur habituel comme moniteur.
On était en 1975, et Pong fut le premier jeu vidéo de l'Histoire.
Cette nouvelle présence quotidienne dans la maison bouleversa les rapports établis entre les téléspectateurs et la télévision : pour la première fois, on voyait à l'écran autre chose que la simple reproduction intangible de la réalité.

A partir de ce moment, le lien immatériel reliant l'émetteur à chaque poste de télévision était pour la première fois rompu, et une partie du monopole de l'information, qui créait un lien unidirectionnel entre la télévision et le spectateur, fut entamée.
Ce dernier, grâce au jeu vidéo, put ouvrir largement la fenêtre de l'écran sur une nouvelle réalité.
Le joueur interagit avec son partenaire électronique, échange avec lui des informations de façon continue et, à travers une représentation symbolique, ressent des émotions bien réelles : il devient acteur, propulsé sur la scène de l'écran.
Pour la première fois, s'établit ainsi un lien fonctionnel, unissant étroitement les deux biens de consommation les plus caractéristiques de notre époque : la ***télévision****, déjà détentrice d'une solide tradition auprès de millions de familles, et le* ***micro-ordinateur****. A l'époque, ce dernier effectuait ses premiers pas sur un marché encore vierge, mais prometteur, en particulier dans le domaine des jeux vidéo.*
Ce terme (néologisme dérivé de l'expression anglaise « video-game ») désigne un système électronique apte à reproduire à l'écran l'atmosphère et les circonstances d'un jeu donné. A partir de certaines règles de décision et des stimuli externes, (instructions du joueur), l'ordinateur simule, en temps réel, une modification de l'environnement du jeu, et la traduit à l'écran. Dans le jeu vidéo, on retrouve les caractéristiques fonctionnelles typiques d'un logiciel interactif.

Habituellement, un jeu vidéo se compose d'une console, de quelques dispositifs de commande, et d'un poste de télévision normal pour l'affichage. On sait qu'entre l'ordinateur et le joueur s'établit un dialogue, sous la forme d'un échange continuel d'informations. Les caractéristiques fonctionnelles des différents composants d'un jeu vidéo découlent logiquement de cette ***interactivité****.*
Une fois effectuée sa connection, l'ordinateur envoie au joueur la première information. Un peu comme pour les dessins animés, il génère un paysage de fond, sur lequel se déplacent les ***opérateurs actifs*** *(personnages) du jeu ; l'image ainsi synthétisée est envoyée sur l'écran de télévision.*
Réagissant à cette stimulation, le joueur entre en relation avec l'ordinateur, en accord avec la dynamique du jeu, et lui transmet ses décisions à travers les mécanismes de contrôle (joystik, trackball), exactement comme un conducteur tourne le volant, écrase la pédale d'accélérateur ou de frein. Les commandes,

saisies sous forme de déplacements mécaniques, sont converties par une interface en impulsions électriques, facilement interprétées par l'ordinateur en quelques millisecondes. Celui-ci vérifie leur compatibilité avec le déroulement correct du jeu et, se fondant sur les instructions de son programme de gestion, propose au joueur une image vidéo représentant une nouvelle situation de jeu. Dans une succession continuelle d'actions et de réactions, le logiciel déplace les opérateurs actifs, décrit de nouvelles scènes et de nouveaux paysages, modifie les couleurs, émet des sons ou de la musique, afin de rendre le jeu plus attractif.
Selon le type d'image proposée, on distingue deux catégories de jeux.
En reprenant l'exemple du conducteur, on appelle **subjectif** le jeu où le mécanisme de contrôle ne déplace pas des opérateurs actifs, mais tout le panorama de l'image télévisée, exactement comme si le joueur se trouvait à bord d'une voiture ou d'un avion (la célèbre simulation de vol) ; dans le jeu **objectif**, l'opérateur actif représente à l'écran le joueur, celui-ci s'identifiant en le faisant bouger selon ses commandes.

Tout ordinateur de jeu vidéo est capable de gérer les deux catégories, moyennant le simple remplacement de la **cartouche** (cassette) où est mémorisé le logiciel approprié.
La cartouche a constitué, pour le marché du jeu vidéo, un élément révolutionnaire, clé de son succès commercial. Grâce à elle, l'utilisateur n'est plus contraint à changer de console pour changer de jeu : dans le cas de Pong, toutes les données du programme résidaient dans une mémoire indissociable de la console. Dans les jeux vidéo des générations suivantes (le VCS 2600 d'Atari, par exemple), le logiciel réside sur une mémoire dissociée de la console : un connecteur permet de la remplacer par d'autres cartouches, et d'accéder ainsi à de nouveaux jeux.
Cette interchangeabilité, comparable à celle d'un magnétophone à cassettes, a contribué à développer un nouveau marché, celui de la production de cartouches pour jeux vidéo. Accessible aux sociétés qui ne fabriquaient pas de consoles, ce marché a assuré la naissance et la prospérité de nouvelles industries, qui aujourd'hui peuvent se vanter de chiffres d'affaires annuels de centaines de millions de dollars. La profession de « game designer » (créateur de jeux vidéo) s'installe dans la notoriété.
Comme ses collègues artistes et musiciens, celui-ci signe ses propres créations, participe aux bénéfices des ventes, possède son public d'admirateurs et devient la vedette des années quatre-vingts.
Pour lui, concevoir un jeu vidéo à partir d'un sujet original constitue un métier très lucratif et, pour la société qui produit la cartouche, une affaire de millions de dollars : un nouvel Eldorado !

Mais les routes qui conduisent à cette mine d'or commencent en 1983 à être encombrées. L'envie de s'approprier une petite tranche des 75 milliards de francs qui représentent le chiffre d'affaires de l'industrie de jeux vidéo a poussé une multitude de petites et grandes sociétés à se lancer dans l'aventure.
Ce fut inévitablement le conflit. Piégées dans une course frénétique pour la conquête d'un marché, jusque-là considéré comme facile et prometteur, mais qui s'avéra bientôt plus complexe que prévu, de nombreuses sociétés se sont retrouvées au bord de la catastrophe.
Au début de 1983, l'avènement de la troisième génération de jeux vidéo a provoqué une reconfiguration draconienne des parts du marché, rendant difficile la survie de ceux qui ne pouvaient offrir un produit technologiquement à jour.
Parallèlement, s'est ouvert une polémique violente au sujet du danger présumé des jeux vidéo sur le plan psychologique. Impitoyablement, certains mass média menèrent une campagne de scandales, afin de discréditer le fétiche moderne qui, quelques mois auparavant, était l'objet d'un tel engouement. Ainsi s'éteignit un phénomène caractéristique de cette culture des excès, où les choix dédaignent parfois l'intelligence et la capacité de réflexion reconnues essentiellement humaines, et dérivent d'un sentiment d'angoisse collective, à laquelle la société croit échapper par une consommation effrénée d'artifices fascinants, colorés, séduisants, mais terriblement éphémères.
Au cours de l'année 1983, à la Bourse de New York, on a respiré, pendant plusieurs jours, un

air de tempête : certaines grosses sociétés chancelaient sous les coups infligés par les pertes de centaines de millions de dollars, d'autres perdaient pied, suivies du glas funèbre de la cloche des Lloyds. En 1983, le jeu vidéo nous a montré ce sinistre paysage, et de nombreux personnages furent désintégrés pour disparaître de la scène ; cependant, la partie ne s'est pas arrêtée sur un catégorique « game-over ». Les ventes de consoles et de cartouches ont subi un certain fléchissement, signe d'un plus grand discernement dans les achats ; mais le public n'a pas déserté massivement.

Aux Etats-Unis, la recette des arcades a dépassé les 5 milliards de dollars, franchissant de manière significative le chiffre d'affaires des industries du cinéma et du disque. Nous sommes en présence d'un phénomène ancré désormais dans le quotidien des habitants d'une bonne partie du monde, mais enfin reconnu pour ce qu'il est : une source d'émotions à bon marché.

Les aventures offertes au joueur varient à l'infini et satisfont tous les goûts. Les statistiques affirment qu'aux Etats-Unis le rapport entre la vente de cartouches et de consoles s'établit, pour l'utilisateur moyen, aux alentours de sept pour un : sept jeux vidéo différents par joueur. On lui propose d'évoluer entre des guerres stellaires (application typique de l'idée : faire feu et s'enfuir), d'échapper à des animaux et fantômes féroces, de trouver sa route dans des labyrinthes inextricables, de sauver la terre d'envahisseurs inter-galactiques. Le joueur peut s'asseoir à la cabine de pilotage d'un B 17, et, depuis son fauteuil, envoyer tranquillement des bombes sur l'Allemagne nazie ; en fait, pas vraiment tranquillement, car un synthétiseur vocal ou un générateur de sons reproduit au mieux les sensations et les peurs qu'un pilote est susceptible d'éprouver dans un bombardier en mission. Des hurlements stridents, des appels au secours, des instructions et des ordres donnés sèchement se succèdent à un rythme serré. Le pilote joueur doit prendre des décisions immédiates, au risque d'être abattu et de tomber dans les eaux de la Manche. Il est à la fois amusant et inquiétant de voir de paisibles fonctionnaires dans le rôle d'as de la RAF.

La recherche et l'expérimentation de nouveaux jeux vidéo se développent en mêlant des connaissances psycho - physico-sensorielles toujours croissantes. Les produits commercialisés aujourd'hui impliquent une technologie sophistiquée de synthèse et de traitement de l'image vidéo, alliant la micro-électronique à la technique du laser. Au moyen de dispositifs électroniques recourant largement aux composants modernes (microprocesseurs, etc.), l'**image analogique** est numérisée et rendue accessible à l'ordinateur, qui se charge de la transcrire par un rayon laser sur un disque spécial de métal et de plastique. Sur une face du disque, environ 250 000 images synthétiques distinctes peuvent être mémorisées. A la différence des bandes magnétiques, qui ne fournissent les images que dans un ordre strictement séquentiel, il est possible d'aller « lire » sur le disque, très rapidement, deux images différentes, stockées à des adresses très éloignées. Reprenons l'exemple du bombardier.

Les fabricants de jeux vidéo organisent régulièrement des expositions : ici, stand Mattel au Computer Play 83.



R. Lonardi/AICA



Imprimatur

Jouer avec un ordinateur signifie aussi apprendre.

Comme les images produites par les jeux vidéo sont très schématiques, on peut envisager, pour rendre la simulation plus vraisemblable, de projeter sur l'écran des séquences tirées d'un film de guerre, montrant une ou plusieurs scènes de bombardement. En manipulant les commandes du jeu vidéo, on aura alors l'impression d'être à bord d'un vrai B 17, mais qu'on ne pourra pas guider, puisqu'aucun rapport de cause à effet (ou mieux de commande à mouvement) ne peut s'établir entre les commandes et le film séquentiel : quoi qu'on fasse, le film se déroule toujours à son propre rythme.

Avec l'emploi du disque vidéo, au contraire, le logiciel choisit parmi toutes les quatre images, celle correspondant aux manœuvres et aux décisions du joueur pilote.
On crée de cette façon des parcours très réalistes fondés sur un véritable rapport commande-mouvement.
En manœuvrant les commandes, on verra défiler ***en temps réel*** *une suite d'images télévisées correspondant au style de pilotage : en tournant à gauche, on verra les maisons, la campagne, l'horizon (avec une précision cinématographique) se déplacer vers la droite.*
On pourra tirer sur les ennemis, viser en suivant la trace des projectiles et les voir exploser véritablement, comme si on était réellement aux commandes d'un avion de guerre ou dans le rôle d'un héros de film.
Le vieux rêve des instructeurs de vol de posséder un simulateur parfait est ainsi réalisé.

Ces systèmes sophistiqués ne demeurent cependant pas dans le domaine professionnel : le jeu vidéo du futur est maintenant à notre portée ; aux Etats-Unis, certains exemplaires de ce type de jeu sont déjà commercialisés.
« Dragon's Lair », le précurseur de cette nouvelle vague, a été réalisé avec les ***photogrammes*** *d'un dessin animé style Disney, et se distingue par un graphisme absolument remarquable.*
Il propose au joueur quarante deux scénarios différents, où se déroule une histoire digne de

la meilleure poésie épique médiévale : sauver la belle princesse prisonnière d'un dragon ; il y a deux cents décisions à prendre d'une manière correcte, sinon c'est la mort.

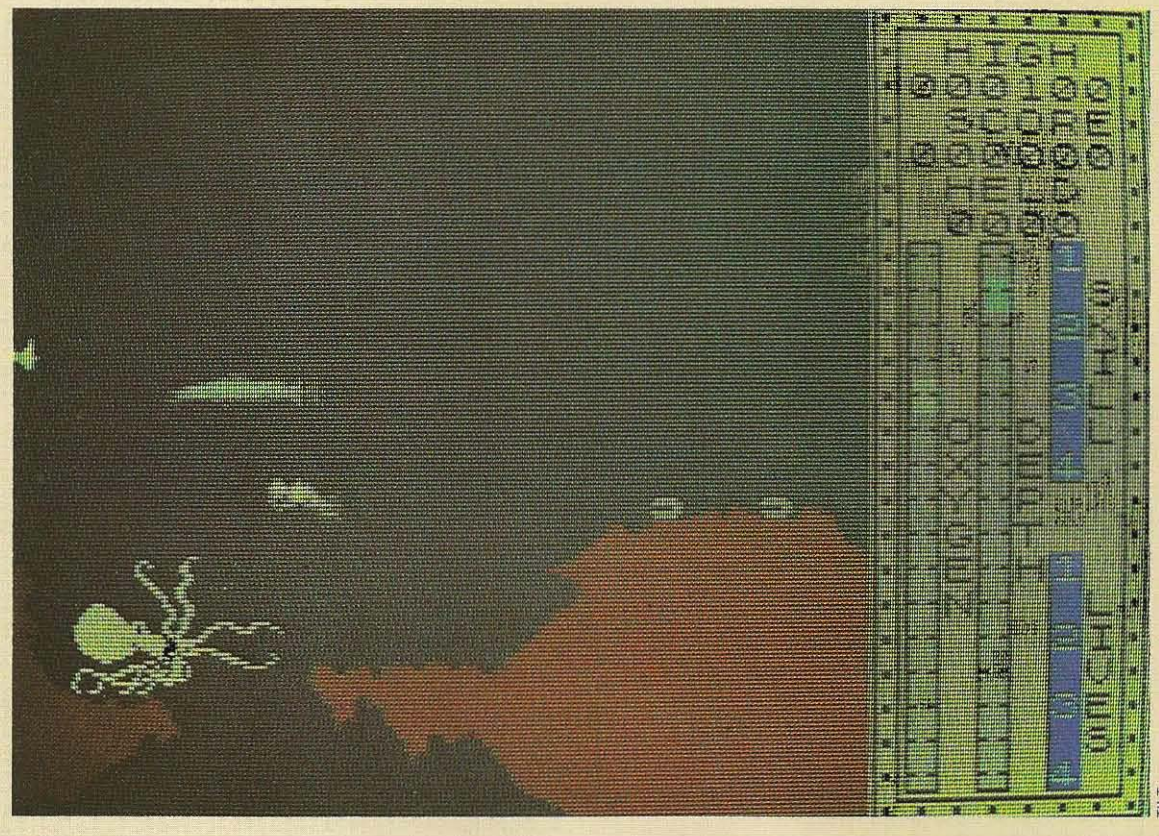
Moins chevaleresque mais plus dramatique, le scénario proposé à certains étudiants en médecine américaine remplace la belle princesse par un patient atteint de péritonite, et qui doit être opéré d'urgence.
Un contexte réel, qui nécessite la plus grande attention et des prises de décision rapides, apprend aux futurs médecins à opérer sans erreur, à travers une simulation d'une vraisemblance absolue.
La première application de ce nouveau jeu est donc ***didactique****, et gagnera certainement nos écrans dans un avenir proche.*

En 1984, des choix courageux s'imposent : on assiste à la naissance d'un phénomène qui, dans les années à venir, va marquer nos temps de loisirs.
Inutile de se leurrer quant à l'indépendance présumée du jeu vidéo par rapport aux modèles mentaux érigés par la société actuelle : il en constitue un fidèle reflet. Le milieu culturel engendre ses propres loisirs, et les jeux vidéo en témoignent.
Inversement, les schémas proposés par les jeux vidéo influent, dans une certaine mesure, sur le ***comportement*** *du joueur.*
Tant qu'on y verra presqu'exclusivement des messages d'agressivité et de guerre, nul ne s'étonnera de la violence latente qui caractérise notre société.
Remarquons toutefois qu'une alternative à l'agressivité engendrée par les jeux vidéo (guerres stellaires, etc.) est déjà proposée avec des jeux qui requièrent des qualités de rapidité de vision et d'action, et qui proposent au joueur de déjouer des séries de difficultés qui font de ces jeux des aventures passionnantes.

Pour certains, le jeu vidéo menace dangereusement l'activité intellectuelle : il est le produit de modèles mentaux utilisés d'une manière non critique, et proposés continuellement au joueur sous une forme obsédante et privée de sensibilité, car masquée par les aspects extérieurs les plus séduisants (couleurs, musiques, sons).
Il ne faut pas oublier que ces jeux s'adressent justement aux enfants et aux jeunes, dont la personnalité plus fragile risque de subir certaines distorsions dangereuses.
Dans les pays anglo-saxons, la vente des jouets obéit à une réglementation sévère : l'usage des jouets doit être précisé lisiblement sur l'emballage, accompagné d'une référence à l'âge des enfants ou des adolescents auxquels ils sont destinés.
Il s'agit là d'une forme très répandue de ***protection*** *et de tutelle vis à vis des plus faibles, au sein des pays les plus évolués socialement.*
Une formule analogue, concernant l'aspect pédagogique et le type de public auquel ils s'adressent, devrait être mise en place également dans un avenir proche pour les jeux vidéo.

Le jeu vidéo existe en tant que phénomène social et peut jouer un rôle culturel positif, surtout dans la mesure où il participe au développement de la propre ***créativité*** *du joueur.*
Jean Piaget affirmait qu'on apprend davantage lorsqu'on invente quelque chose ; en ce sens, le jeu vidéo et l'ordinateur domestique peuvent constituer deux instruments actifs de connaissance.
Les différences entre le jeu vidéo et l'ordinateur domestique ne sont pas si grandes ; ce dernier représente l'évolution logique du jeu vers des applications plus larges.
L'ordinateur possède la même structure de base interne qu'une console de jeu vidéo, améliorée par l'introduction d'autres dispositifs : un clavier, plus d'espace mémoire et un lecteur de cassettes pour stocker les programmes.
Certains constructeurs de jeux vidéo proposent déjà des « extensions informatiques » compatibles avec le système de base, accompagnées d'un manuel d'apprentissage du langage Basic.
Le jeu vidéo constitue ainsi une approche remarquable de l'informatique, surtout chez les enfants, en habituant l'utilisateur au dialogue avec l'ordinateur.

Depuis quelques années, on assiste dans toute l'Europe, à une véritable invasion d'ordinateurs domestiques.
Dans de nombreux pays, on offre un ordina-



TILT

Un exemple de jeu vidéo : ces aventures sous-marines sont mises en valeur par un excellent graphisme et une très bonne animation (Durell Software pour Spectrum 48 K).

teur comme cadeau de Noël à des jeunes dont l'âge se situe entre dix et quatorze ans. Parmi les programmes destinés aux ordinateurs domestiques, il est possible d'établir une classification des plus demandés : à travers celle-ci, on a mis aussi en évidence les critères d'application de l'ordinateur dans le domaine familial.

Les jeux de société et les jeux éducatifs prédominent toujours : ils couvrent 54 % des ventes. L'importance de ce phénomène n'est pas à sous-évaluer : le jeu est un choix qui persiste même quand le joueur peut créer ses programmes de manière autonome. Ceci signifie qu'on est en présence de ***stimulations*** *fondamentales, de désirs profonds qui s'expriment chez l'homme et qu'il veut satisfaire de toute façon, et pas seulement dans son plus jeune âge.*

Dans toute l'Europe, un certain nombre d'études ont été faites dans ce domaine, débouchant souvent sur des congrès et des expositions.

Ces démarches avaient en général pour but d'établir un débat sur les problèmes liés à la diffusion des jeux vidéo, et de proposer, de la part d'experts qualifiés, une analyse des nombreuses questions que l'avènement des jeux informatiques a soulevées : qu'est-ce qu'un jeu, comment le conçoit-on, comment l'évalue-t-on, quels sont les aspects éducatifs du jeu informatique, quel marché couvre-t-il, à travers quel jeu, par quelle méthode peut-on introduire le jeu dans l'enseignement, quel jeu proposer pour un âge donné.

Durant ces congrès, les participants ont affronté ces questions, en partant d'analyses différentes. On a, toutefois, pu mettre en évidence une tendance toujours plus prononcée vers une plus vaste application des jeux éducatifs à ***haut niveau conceptuel****.*

Cette nouvelle culture du jeu s'impose aux goûts des gens, et on doit lui prêter toute l'attention qu'elle mérite. Le but des promoteurs a été d'observer sur le terrain ce qui se fait, au niveau de l'initiative privée, dans le secteur des jeux pour ordinateur.

GET. La syntaxe de cette instruction est la suivante :

GET # N,R

Elle lit les données de l'enregistrement R du fichier N et les transfère dans les mémoires tampon précédemment associées au fichier N. Par exemple,

GET # 1,75

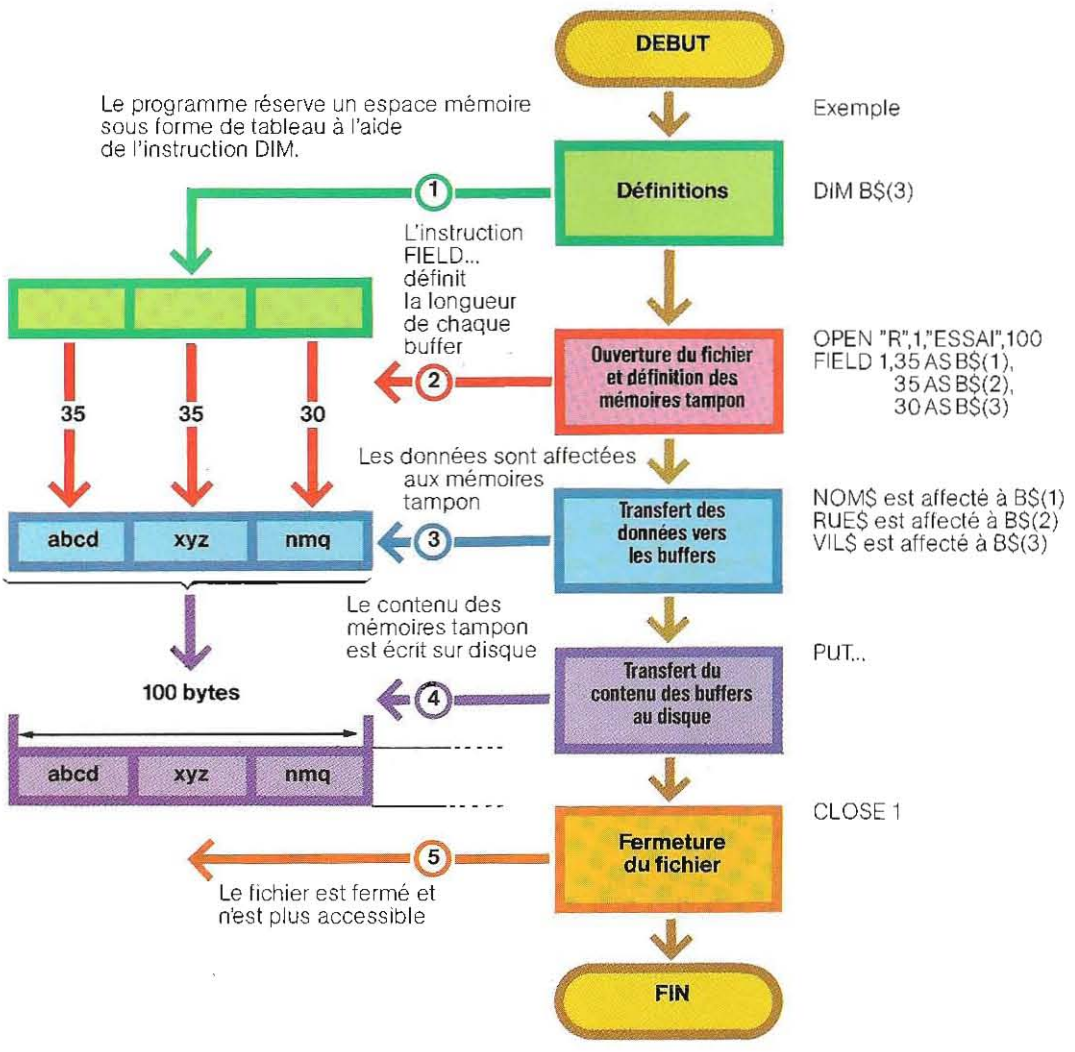
lit les données contenues dans l'enregistrement 75 du fichier 1.

LOC(N). La fonction LOC(N), où N représente le numéro d'unité logique associé au fichier, fournit la valeur « courante » du pointeur, c'est-à-dire le numéro du dernier enregistrement utilisé (dans une instruction PUT ou GET), augmenté de 1. Par exemple, l'exécution de la suite d'instructions :

```
10  GET 1,7
20  PUT 1,9
30  A=LOC(1)
```

affecte à A la valeur 10, puisque le dernier numéro d'enregistrement utilisé vaut 9 (instruction 20).
Le schéma ci-dessous illustre un transfert de

SCHEMA D'ECHANGE DE DONNEES AVEC LE DISQUE



PROGRAMME D'ECRITURE SUR UN FICHIER SEQUENTIEL

```
10  ' ** PROGRAMME D'ECRITURE SUR UN FICHIER SEQUENTIEL **
20  '
30  INPUT "NOM DU FICHIER ";NN$
40  NN$="A:" + NN$                  ' NN$ contient le nom complet du fichier
50  '                                (A est l'unite disque ou il est stocke)
60  '
70  OPEN "O",1,NN$                  ' Ouverture en mode ecriture du fichier de nom NN$
80  '                                et de numero logique 1
90  ' GOSUB 500                     ' Routine de saisie des donnees au clavier
100 CLOSE 1
110 '
120 OPEN "I",1,NN$                  ' Ouverture en mode lecture
130 GOSUB 1000                      ' Routine de relecture et impression
140 CLOSE 1
150 '
160 '
170 INPUT "Voulez-vous ajouter de nouvelles donnees (oui/non) ";REP$
180 IF REP$="NON" GOTO 400          ' Sortie du programme
190 '
200 OPEN "I",2,NN$                  ' Ouverture en lecture du fichier original NN$
210 '                                avec le numero logique 1
220 OPEN "O",1,"A:COPY"             ' Ouverture en ecriture d'un nouveau fichier
230 '                                avec le numero logique 1
240 INPUT £2,V,A$                   ' Lecture d'un enregistrement
250 PRINT £1,V,A$                   ' Copie sur le nouveau fichier
260 IF EOF(2) GOTO 280
270 GOTO 240                        ' Vers une nouvelle recopie
280 CLOSE 2                         ' Le fichier original (2) a ete entierement recopie
290 KILL NN$                        ' On l'efface
300 '
310 GOSUB 500                       ' Saisie des donnes a rajouter sur le fichier 1
320 CLOSE 1
330 NAME "A:COPY" AS NN$            ' On renomme le nouveau fichier avec
340                                  le nom original
350 OPEN "I",1,NN$
360 GOSUB 1000                      ' Relecture
370 CLOSE 1
380 GOTO 170                        ' Vers une nouvelle modification eventuelle
390 '
400 END
410 '
500 ' ** ROUTINE DE SAISIE DES DONNEES AU CLAVIER **
510 PRINT "POUR TERMINER LA SAISIE, TAPER 0"
520 PRINT
530 INPUT "Valeur numerique a ecrire ";V
540 IF V=0 GOTO 580          ' Sortie
550 INPUT "Chaine alphanumerique ";A$
560 PRINT £1,V,A$            ' Ecrit sur le fichier 1 les valeurs V et A$
570 GOTO 530                 ' Vers une nouvelle saisie
580 PRINT "**** FIN DE SAISIE ****"
590 RETURN
600 '
1000 ' ** ROUTINE DE RELECTURE ET IMPRESSION DU FICHIER **
1010 INPUT £1,V,A$           ' Lecture des donnees d'un enregistrement
1020 LPRINT                  ' et affectation aux variables V et A$
1030 LPRINT "Valeur numerique lue = ";V
1040 LPRINT "Chaine alphanumerique = ";A$
1050 IF EOF(1) GOTO 1070
1060 GOTO 1010               ' Vers une nouvelle lecture
1070 PRINT "**** FIN DE RELECTURE ****"
1080 RETURN
```

```
Valeur numerique lue = 1965
Chaine alphanumerique = ISABELLE

Valeur numerique lue = 1959
Chaine alphanumerique = LAURE

Valeur numerique lue = 1925
Chaine alphanumerique = ANNE
```

données sur disque pour la mémorisation d'un carnet d'adresses. Les champs prévus sont les suivants :

nom et prénom = 20 + 15 caractères ;
numéro et rue = 5 + 30 caractères ;
code postal et ville = 30 caractères.

On utilise 3 buffers, une pour chaque champ de données, qui peuvent être appelés par trois noms quelconques de chaîne. Il est cependant préférable d'employer toujours le même nom générique, différencié par un indice : on peut utiliser un tableau de chaînes, dimentionné dans ce cas particulier à trois valeurs, de façon à n'avoir qu'un seul nom réservé à la mémoire tampon, dans les opérations d'entrées/sorties.

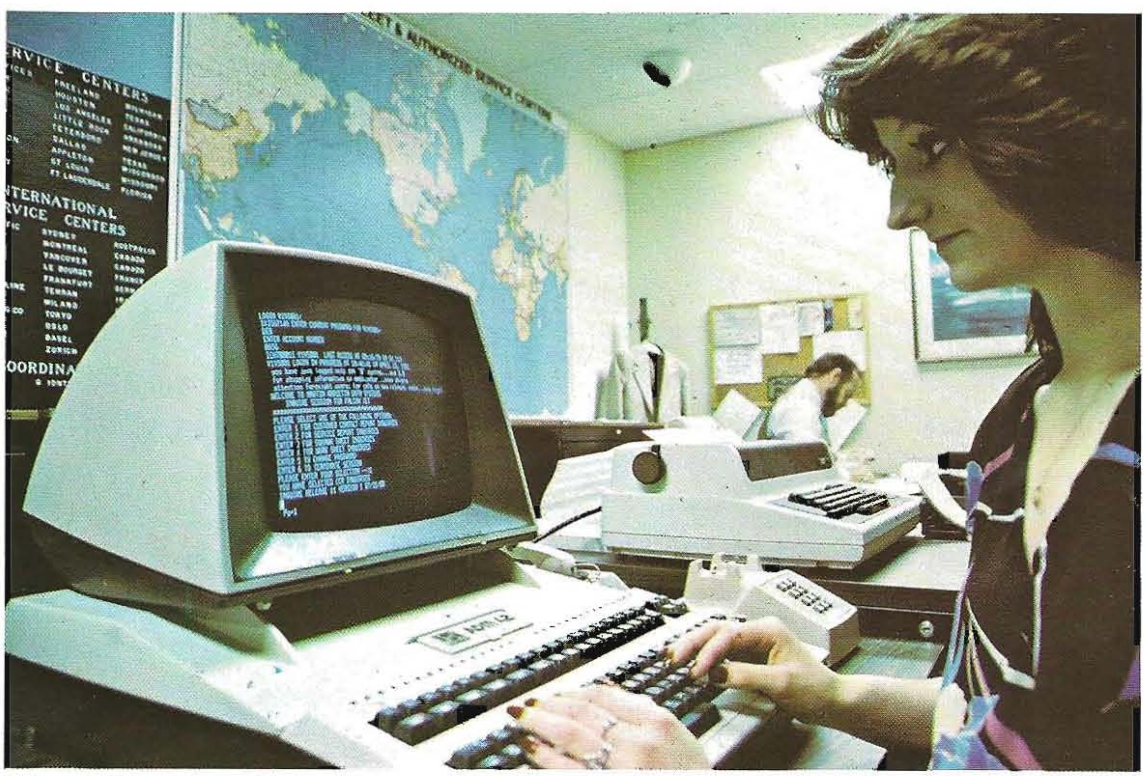
Formats et codifications des données

Dans les fichiers séquentiels, les données sont mémorisées en format ASCII, mais elles peuvent cependant être lues par le programme sous forme de valeurs numériques. Le programme listé à la page 653 écrit sur un fichier séquentiel (ouvert ligne 70 sous le numéro 1) un certain nombre de données saisies au clavier dans la routine 500, constituées alternativement d'une valeur numérique et d'une chaîne. La saisie se termine en entrant 0 comme valeur numérique (ligne 540). Pour relire le fichier et l'éditer sur l'imprimante (routine 1000), il faut d'abord le fermer puis l'ouvrir en mode « I ». On remarquera que les instructions PRINT et INPUT des lignes 560 et 1010 contiennent le symbole £1 (qui remplace ici le #1 classique), pour indiquer à l'interpréteur Basic qu'il s'agit d'un échange de données avec l'unité logique 1 et non avec la console.

Le programme propose ensuite de rajouter des données au fichier séquentiel ainsi créé. Dans les fichiers en accès direct, les données sont mémorisées sous forme binaire, mais elles doivent toujours être fournies en format ASCII, puisqu'on utilise comme support une mémoire tampon de type chaîne. Cela constitue la seule difficulté survenant dans les opérations d'E/S avec ce type de fichier ; le jeu d'instructions du Basic permet d'ailleurs de la surmonter facilement.

Les fonctions principales utilisées pour la

Un ordinateur ADM utilisé à la Falcon Jet Company.



Leo de Wys/Marka

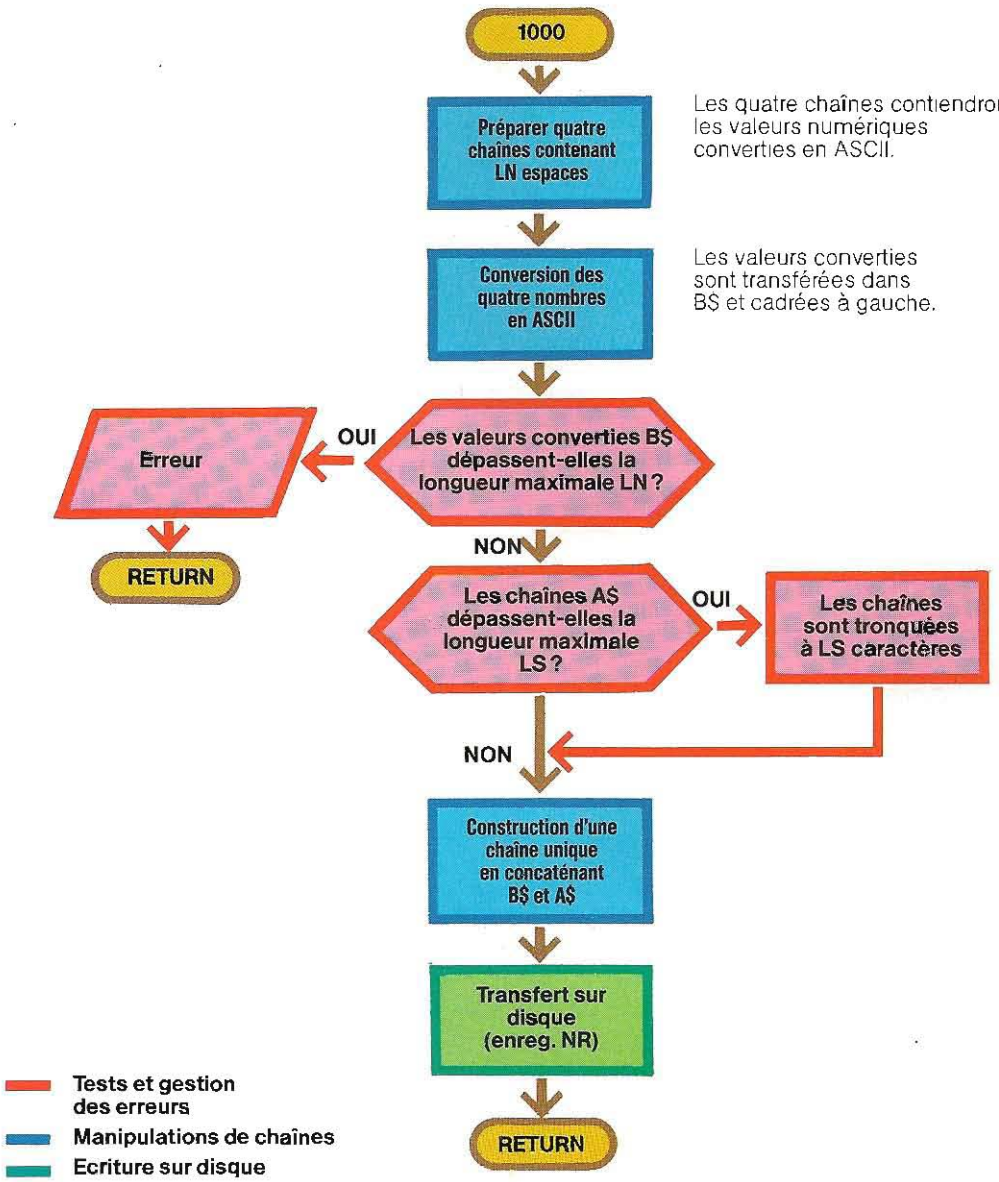
PROGRAMME D'ECRITURE SUR UN FICHIER EN ACCES DIRECT

```
5 ' ** PROGRAMME D'ECRITURE SUR UN FICHIER EN ACCES DIRECT **
10 ' Exemples de conversion des valeurs numeriques pour les operations d'E/S
20 ' avec le disque (fichiers en acces direct)
30 '
40 OPTION BASE 1
50 DEFINT I          ' Les noms d'initiale I se referent a des variables
60 '                   entieres
70 DEFSNG R          ' Les noms d'initiale R se referent a des variables
80 '                   en simple precision
90 DEFDBL D          ' Les noms d'initiale D se referent a des variables
100 '                  en double precision
110 DIM BF$(3)       ' Buffer d'entrees/sorties (un pour chaque type de variable)
120 OPEN "R",1,"TEST",30      ' Chaque enregistrement a une longueur de 30 octets
130 FIELD 1,10 AS BF$(1), 10 AS BF$(2), 10 AS BF$(3)
140 '
150 '
160 ' ** Lecture des valeurs numeriques a ecrire sur disque
170 '
180 INPUT "Valeur entiere"; INTEG
190 INPUT "Valeur en simple precision"; REEL
200 INPUT "Valeur en double precision"; DOUBLE
210 '
220 ' ** Les variables INTEG, REEL et DOUBLE contiennent les donnees saisies,
230 '    a convertir en ASCII
240 '
250 A1$=MKI$(INTEG)          ' Les fonctions MKI$, MKS$ et MKD$
260 A2$=MKS$(REEL)           ' transforment les valeurs numeriques
270 A3$=MKD$(DOUBLE)         ' en leur equivalent ASCII
280 '
290 ' ** Saisie du numero d'enregistrement a remplir
300 INPUT "Dans quel enregistrement doit-on ecrire"; IR
310 '                  le numero IR est un entier (initiale I)
320 '
330 ' ** Transfert dans les buffers d'E/S
340 '
350 BF$(1)=A1$          ' Rappelons que les variables intermediaires
360 BF$(2)=A2$          ' A1$, A2$ et A3$ sont necessaires, car aucune
370 BF$(3)=A3$          ' operation ne peut etre effectuee sur les buffers
380 '
390 ' ** Ecriture sur le fichier
400 ' Le contenu des buffers BF$ est ecrit dans l'enregistrement
405 ' numero IR du fichier 1
410 PUT #1,IR
420 INPUT "Voulez-vous continuer (oui/non)"; REP$
430 IF REP$="oui" GOTO 180        ' Nouvelle ecriture
440 '
450 ' ** Relecture du fichier
460 '
470 INPUT "Quel enregistrement voulez-vous relire"; IR
480 GET #1, IR          ' Lecture de l'enregistrement numero IR
485 '                     Les donnees sont transferees dans les buffers BF$
490 A1$=BF$(1)
500 A2$=BF$(2)
510 A3$=BF$(3)
520 INTEG=CVI(A1$)          ' Les fonctions CVI, CVS et CVD
530 REEL=CVS(A2$)           ' transforment les codes ASCII
540 DOUBLE=CVD(A3$)         ' en valeurs numeriques
550 LPRINT "Donnees lues : "; INTEG, REEL, DOUBLE
560 INPUT "Voulez-vous continuer (oui/non)"; REP$
570 IF REP$="oui" GOTO 470
580 CLOSE #1
590 END
```

transformation des données sont :
MKI$, MKS$, MKD$ en écriture (conversion de valeur numérique en chaîne) ;
CVI, CVS, CVD en lecture (conversion de chaîne en valeur numérique).
Le programme listé ci-dessus utilise ces fonctions pour l'écriture et la lecture de données sur un fichier en accès direct.

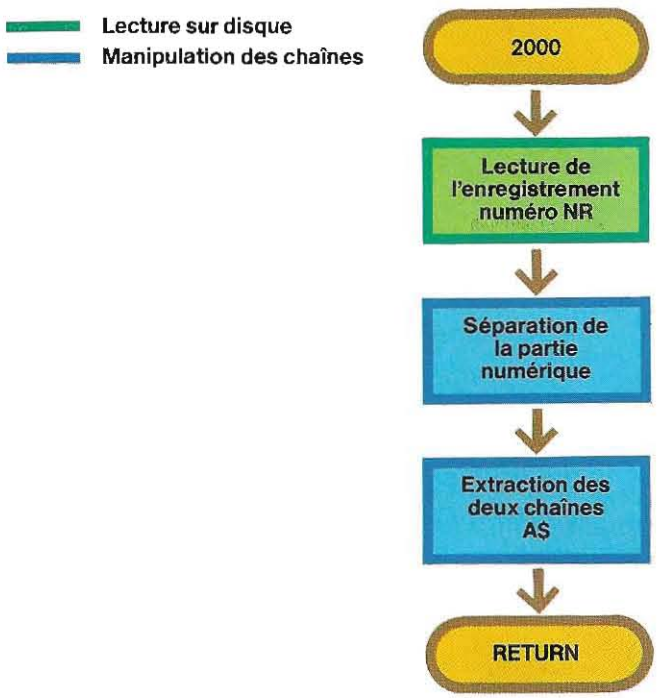
PREPARATION ET ECRITURE D'UN ENREGISTREMENT

Entrées du sous-programme : NR = Numéro de l'enregistrement.
V(4) = Valeurs numériques à écrire dans l'enregistrement NR.
LN = Longueur maximale (en nombre de chiffres) des valeurs numériques.
A$(2) = Chaînes à écrire dans l'enregistrement NR.
LS = Longueur maximale de chaque chaîne (en caractères).



La conversion des données peut aussi s'effectuer grâce aux fonctions STR$(R) et VAL(A$).
L'organigramme ci-dessus correspond à un sous-programme qui utilise ces fonctions pour préparer l'enregistrement à écrire sur disque. La page 657 illustre l'organigramme de la routine exécutant la lecture.
Les deux sous-programmes sont appelés par le programme principal, listé en pages 657 et 658. Pour faciliter la compréhension du sous-programme 2000, le schéma de la page 659 explique la logique mise en œuvre pour l'extraction des données.

LECTURE D'UN ENREGISTREMENT ET PREPARATION DES VALEURS NUMERIQUES



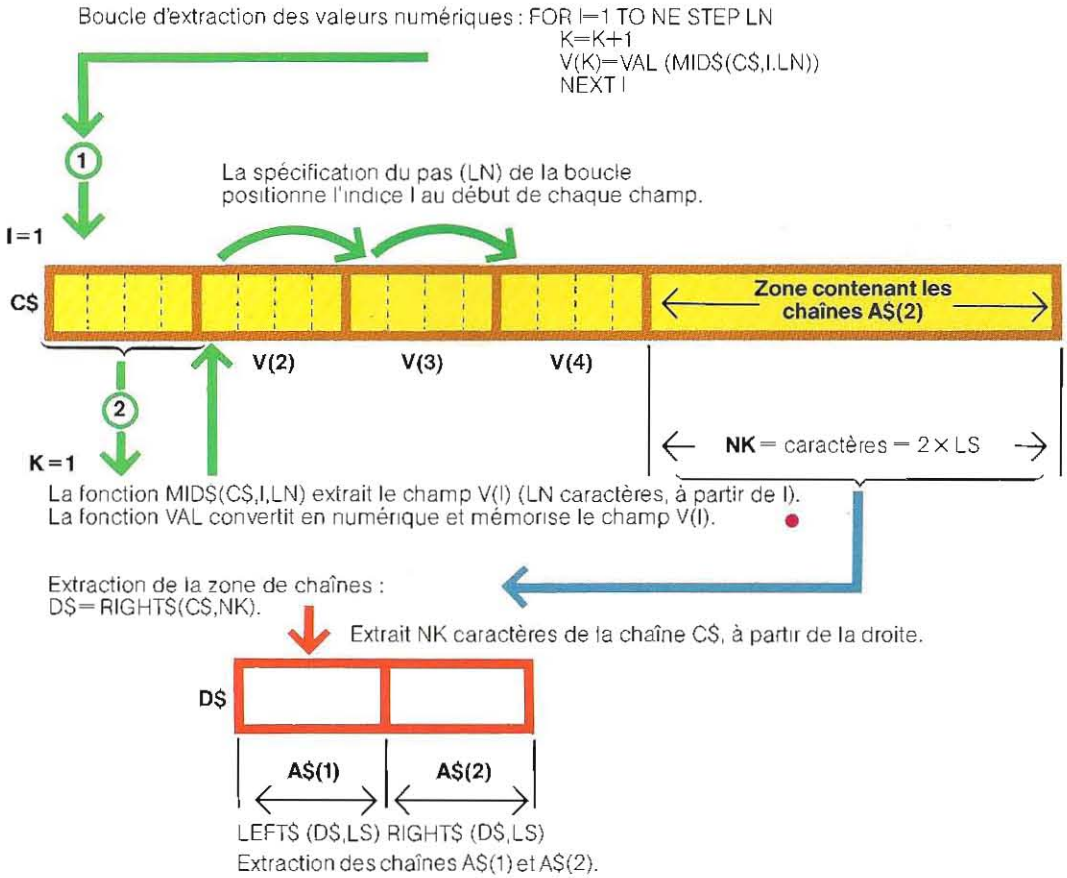
PREPARATION, LECTURE ET ECRITURE D'UN ENREGISTREMENT

```
10 '
20 '
30 '    PROGRAMME =LEC/EC
35 DEFINT I-N
40 ' **
50 '     NR = numero d'enregistrement a remplir
60 '     V(4) = valeurs numeriques a ecrire
70 '     LN = longueur max. des valeurs numeriques (nombre de chiffres)
80 '     A$(2) = chaines a ecrire a la suite des valeurs numeriques
90 '     LS = longueur max. des chaines (nombre de caracteres)
100 '
110 DIM B$(4), A$(2)
112 OPEN "R",1, "TEST2",50        ' Les enregistrements ont 50 octets de longueur
114 FIELD #1,50 AS BUF$           ' BUF$ est la memoire tampon associee au fichier TEST2
120 LS=10
130 LN=5
140 INPUT "Numero d'enregistrement a ecrire"; NR
150 FOR I=1 TO 4
160 PRINT "Donnee numerique"; I
170 INPUT "Valeur (max. 5 chiffres)"; V(I)
180 NEXT I
190 INPUT "Premiere chaine"; P$
200 INPUT "Seconde chaine"; S$
202 A$(1)=SPACE$(LS) : A$(2)=SPACE$(LS)
204 LSET A$(1)=P$          ' Transfert et cadrage a gauche des chaines
206 LSET A$(2)=S$          ' lues
210 '
220 GOSUB 1000           ' Preparation et ecriture de l'enregistrement
230 '
240 A$(1)="" : A$(2)=""       ' Initialisations
250 FOR I=1 TO 5
252 V(I)=0
254 NEXT I
```

```
260 '
270 GOSUB 2000          ' Relecture et ecriture de l'enregistrement
280 '
290 LPRINT "Valeurs numeriques:"
300 FOR I=1 TO 4
310 LPRINT V(I)
320 NEXT I
330 LPRINT
340 LPRINT "Chaines:"
350 LPRINT A$(1)
360 LPRINT A$(2)
370 '
380 END
390 '
1000 ' ** SUBROUTINE D'ECRITURE SUR DISQUE **
1010 IF NR=0 THEN PRINT "Erreur sur le numero d'enregistrement" : GOTO 1240
1020 FOR I=1 TO 4
1030 B$(I)=SPACE$(LN)          ' Preparation des chaines d'espaces
1040 NEXT I
1050 FOR I=1 TO 4
1060 C$=STR$(V(I))             ' Conversion de V(I) en chaine
1070 IF LEN(C$)>LN THEN PRINT "Erreur sur la valeur numerique";I : GOTO 1240
1080 RSET B$(I)=C$             ' Transfert et cadrage a droite
1090 NEXT I
1100 C1$=A$(1)                 ' Transfert de la premiere chaine
1110 IF LEN(A$(1))>LS THEN C1$=LEFT$(A$(1),LS)
1120 C2$=A$(2)                 ' Transfert de la seconde chaine
1130 IF LEN(A$(2))>LS THEN C2$=LEFT$(A$(2),LS)
1140 ' *C1$ et C2$ contiennent les memes chaines que A$(1) et A$(2)
1150 '  mais eventuellement tronquees si elles etaient trop longues
1160 ' ** Preparation de la chaine a ecrire
1170 C$=""
1180 FOR I=1 TO 4
1190 C$=C$+B$(I)               ' Concatenation des chaines representant
1200 NEXT I                    ' les 4 valeurs numeriques
1210 C$=C$+C1$+C2$
1212 PRINT "Les donnees sont stockees dans l'enregistrement, sous la forme:"
1214 PRINT C$
1220 BUF$=C$                   ' Transfert dans la memoire tampon
1230 PUT £1,NR                 ' Ecriture sur disque
1240 RETURN
1250 '
2000 ' ** SUBROUTINE DE LECTURE SUR DISQUE **
2010 GET £1,NR                 ' Lecture
2020 C$=BUF$                   ' C$ = buffer intermediaire, contenant les donnees
2025 '                          telles qu'elles sont lues
2030 PRINT "Enregistrement lu";C$
2040 NE=4*LN                   ' Longueur en octets de la zone numerique,
2050 '                          soit 4 champs de longueur LN
2060 K=0
2070 FOR I=1 TO NE STEP LN
2080 K=K+1
2090 V(K)=VAL(MID$(C$,I,LN))
2100 NEXT I
2110 NK=2*LS                   ' Longueur de la zone alphanumerique
2120 ZA=RIGHT$(C$,NK)          ' Extraction de cette zone qui contient
2130 '                          les deux chaines
2140 A$(1)=LEFT(ZA$,LS)        ' Separation des deux chaines
2150 A$(2)=RIGHT(ZA$,LS)
2160 RETURN
```

```
Valeurs numeriques:
 1111
 2222
 3333
 4444

Chaines:
AAAAAAAAAA
BBBBBBBBBB
```

SCHEMAS LOGIQUES D'EXTRACTION DES DONNEES



Sélection des données

Au cours de la gestion des fichiers, il arrive rarement d'avoir à traiter la totalité des enregistrements existants. En règle générale, les mises à jour, recherches ou autres traitements n'intéressent que certains enregistrements, reconnaissables par les valeurs de certains champs déterminés.

Ces champs constituent la **clé** d'accès à l'enregistrement et, lorsque le système le permet, le programmeur préfèrera adopter une structure de fichier indexé. De cette manière, toute la gestion des pointeurs est assurée par le système, et l'utilisateur n'a à sa charge, dans son programme, que quelques instructions spécifiant la clé qu'il entend utiliser pour sélectionner l'enregistrement.

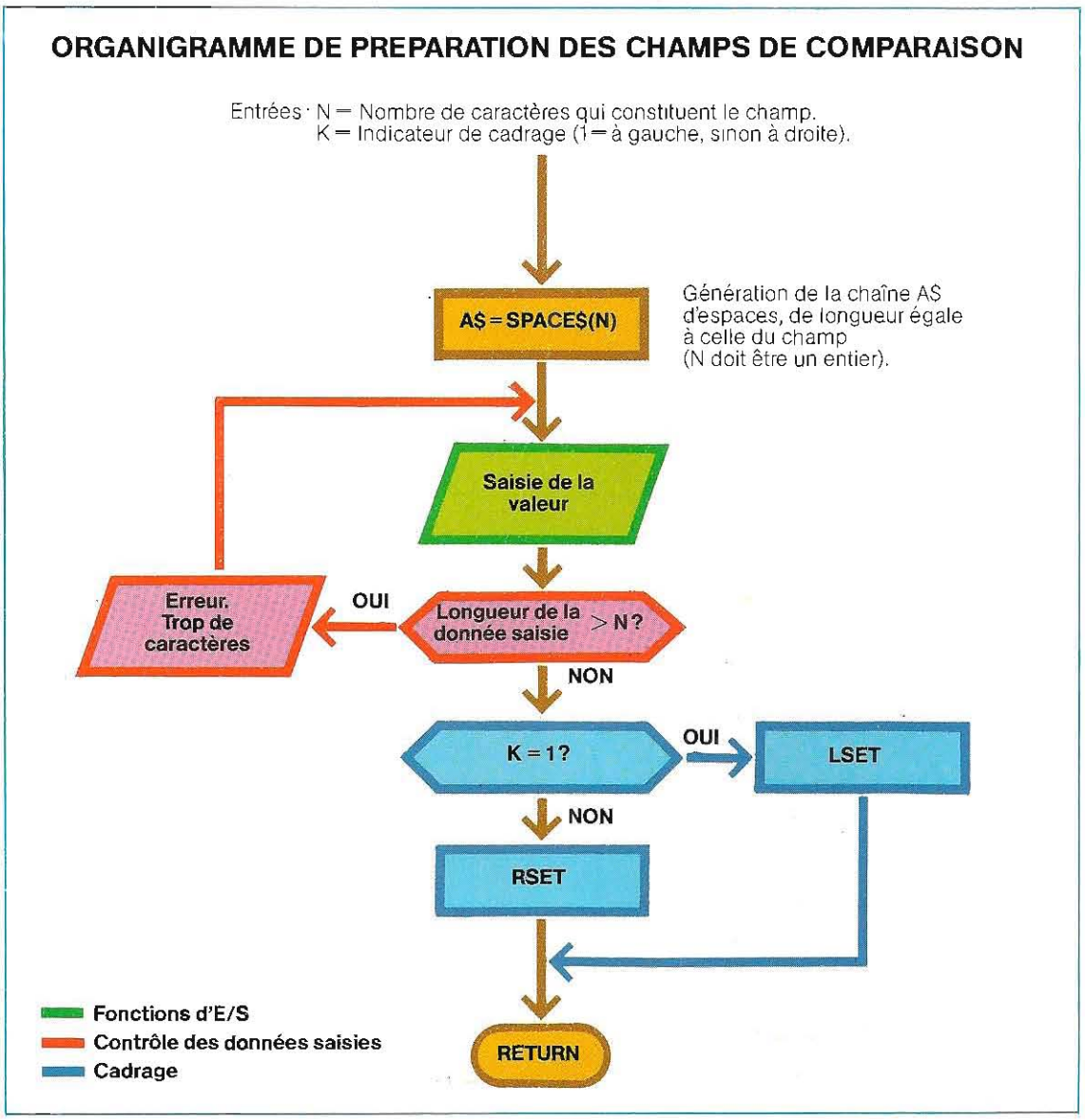
Le logiciel de base (bibliothèque de programmes « utilitaires ») permettant ce type de gestion n'existe pas sur tous les systèmes, et très souvent, le programmeur devra écrire lui-même ses « utilitaires ».

Prenons l'exemple d'une routine de recherche d'un enregistrement selon une valeur de champ donnée. Une programmation de ce genre ne présente pas de difficultés particulières : il s'agit uniquement d'utiliser les opérateurs logiques à bon escient. La seule précaution à prendre est de justifier (cadrer) tous les champs de la même manière.

En général, lors de l'écriture sur disque, les champs ne sont pas complètement remplis de caractères, et il demeure des blancs. Lorsque le programme lit sur disque en enregistrement, tous les caractères sont chargés en mémoire, y compris les blancs. Ainsi, un champ extrait d'un fichier et un autre saisi au clavier seront peut-être cadrés différemment, et une comparaison entre les deux donnera un résultat négatif, même s'ils contiennent les

mêmes caractères (exceptés les blancs).
Supposons, par exemple, qu'un fichier mémorisé sur disque comporte un champ pour la saisie d'un nom de ville, d'une longueur de 10 caractères : la donnée tapée au clavier en phase de saisie est LYON. En ce cas, le champ contient, à la suite des quatre caractères de la donnée, six blancs. Afin de rechercher dans le fichier l'enregistrement contenant la donnée LYON, il faut préparer une chaîne de comparaison possédant les mêmes caractéristiques : la donnée doit être cadrée à gauche, et l'espace restant doit être laissé en blanc.
La convention habituelle consiste à cadrer les chaînes de caractères à gauche et les chaînes représentant des valeurs numériques à droite ; on peut bien sûr adopter une convention différente, à condition de la respecter pour toutes les opérations de fichier.
Dans l'exemple précédent, il suffit de préparer une chaîne d'espaces, de la longueur du champ, et d'y transférer, correctement cadrée selon la valeur de l'indicateur K, la donnée saisie au clavier. La chaîne ainsi construite pourra être utilisée comme valeur de comparaison, dans une boucle de lecture et de sélection des enregistrements.
L'organigramme ci-dessous illustre schématiquement la préparation de cette chaîne.

ORGANIGRAMME DE PREPARATION DES CHAMPS DE COMPARAISON



Commandes et fonctions spéciales

Les systèmes d'exploitation limitent toujours le nombre de fichiers qui peuvent être ouverts simultanément (par exemple, trois sous CP/M) ; d'éventuelles modifications de cette limite doivent être explicitement communiquées au système lors du chargement de l'interpréteur BASIC.
Dans ce cas, le format de commande du chargement de l'interpréteur est habituellement le suivant :

MBASIC/F:N

Le symbole / sert de délimiteur entre la commande MBASIC et les options choisies ; la lettre F indique qu'on veut ouvrir plus de fichiers que la limite standard, et la valeur N représente le nombre de fichiers désiré.
Ainsi, la commande

MBASIC/F:5

prépare l'interpréteur à manipuler simultanément cinq fichiers. Sous le système d'exploitation CP/M, l'espace mémoire représente 166 octets pour chaque fichier. Dans l'exemple précédemment considéré, la zone mémoire réservée aux blocs de description (FDB : File Description Block) des cinq fichiers serait de : $5 \times 166 = 830$ octets.
Une autre option est possible :

/ S:N

Le nombre N indique la longueur maximale des enregistrements, exprimée en octets.
Si elle n'est pas spécifiée, cette longueur vaut, par défaut, 128 octets (la spécification d'une longueur maximale n'empêche pas d'attribuer des longueurs plus petites aux buffers dans l'instruction FIELD).
Rappelons que, sous CP/M, le nombre maximum d'enregistrements logiques adressables se limite à 32767, sauf si l'on définit, au moyen de l'option S, une longueur d'enregistrements de 256 octets ; en ce cas, l'adressage maximal est de 8 Méga-octets.
Sur certains interpréteurs BASIC, une fonction spéciale permet de connaître la taille des fichiers (en secteurs).
En Basic 80, cette fonction est :

LOF(N)

qui délivre le nombre d'enregistrements de la dernière extension du fichier de numéro logique N. Si ce fichier ne possède qu'une seule extension, ce nombre coïncide avec sa taille réelle.

Fichiers multivolumes

La capacité de stockage des disquettes varie de 160 Kilo-octets à 1 Méga-octet, selon leurs dimensions et leurs types. Pour certaines applications, il s'avère parfois nécessaire de disposer d'une capacité plus grande que celle offerte par la disquette.
La meilleure solution consiste évidemment à utiliser des supports mieux adaptés, comme, par exemple, les Winchester (qui sont des disques durs). Sur les micro-ordinateurs, certains permettent de stocker jusqu'à 36 Méga-octets, les modèles les plus usuels restant entre 5 et 10 Méga-octets. En outre, les disques durs offrent l'avantage de temps d'accès très inférieurs lors des fonctions d'E/S.
Cependant, des questions de coût ne permettent pas toujours d'adopter cette solution, optimale d'un point de vue technique. Il faut alors, par programme, suppléer aux limites du matériel.
La méthode classique consiste à partager le fichier en plusieurs sections, qui seront stockées séparément sur disquettes.
Le programme d'application devra, bien sûr, tenir compte de cette répartition, de manière à guider l'utilisateur dans le choix de la disquette appropriée. Au moment du traitement, le programme déterminera la disquette sur laquelle sont stockées les données en cours, en informera l'utilisateur, et attendra que le montage de la disquette requise soit achevé.
Un tel fichier, caractérisé par sa subdivision en plusieurs parties, est dit **multivolume**, et chacune de ses parties est appelée **volume**.
Cette répartition des données qui, sur les micro-ordinateurs ou les ordinateurs personnels, permet souvent d'accroître les capacités de stockage, peut aussi se justifier par un choix logique. Il n'est pas toujours souhaitable de mémoriser des données dans un fichier unique ; parfois, il s'avère plus avantageux de diviser le fichier en plusieurs volumes, même si le disque est capable de le contenir en entier. Dans ce cas, on opère une division de

type logique : la répartition en volumes existe, mais tous les volumes résident sur la même disquette. Le fichier étant structuré de cette façon, le programme d'application qui l'utilise n'a pas à attendre le montage des différents disques.
Un programme de gestion de fichiers multivolumes sera extrêmement complexe s'il opère sur des fichiers indexés, alors qu'il s'avère relativement simple dans le cas de fichiers en accès direct. La logique de gestion d'un fichier en accès direct repose sur l'emploi du numéro d'enregistrement comme clé d'accès aux données : la première donnée saisie portera le numéro d'enregistrement 1, la deuxième sur le numéro 2 et ainsi de suite. Rappelons, toutefois, qu'assimiler le numéro d'enregistrement à une clé d'accès est un abus de langage ; l'adresse physique de chaque enregistrement se calcule, en réalité, en soumettant les données à des traitements relativement complexes. La gestion d'un fichier multivolume consiste à mémoriser dans un fichier à part, les noms des volumes (c'est-à-dire des disquettes) et pour chacun, les numéros de premier et de dernier enregistrement stockés. Ce fichier, qui retrace l'évolution du fichier multivolume, constitue une sorte de répertoire, et ne comporte que peu d'enregistrements (un par disquette) constamment mis à jour.

Gestion des fichiers sur les grands systèmes

Sur les systèmes plus complexes, à partir des mini-ordinateurs, sont implantés des programmes « utilitaires » de gestion des bases de données, qui déchargent le programmeur d'une grande partie des tâches de détail.
Ces programmes de gestion se rangent en deux catégories : programmes spécialisés autonomes et programmes nécessitant un langage hôte.
L'ensemble des programmes spécialisés constitue un véritable langage de haut niveau, comportant des « instructions » comparables à celles d'un langage de programmation, ainsi que des commandes spécifiques à la gestion des fichiers. Les programmes rattachés à un langage hôte sont de véritables routines formées d'une série d'instructions, et qui accroissent les possibilités du langage.

Sur les gros systèmes, les principaux langages hôtes (toujours compilés) sont le Cobol et le Fortran ; les commandes inhérentes à la gestion des fichiers sont traduites sous une forme compatible avec le langage hôte (lors de la compilation), puis exécutées.
Les commandes fondamentales prévues dans tout système d'exploitation sont :

STORE écrit un enregistrement, et crée toutes les chaînes éventuelles de pointeurs vers les autres données ;
FIND recherche un ensemble de données en fonction de paramètres fournis par l'utilisateur ;
MODIFY modifie une ou plusieurs données ;
ERASE efface un enregistrement, et modifie en conséquence les chaînes de pointeurs qui lui sont associées.

De nombreux langages élaborés spécifiquement pour la gestion des bases de données sont pourvus de « sous-ensembles » d'interrogation, véritables procédures de recherche de données, accessibles par un personnel qui n'a pas nécessairement des connaissances de programmation. La syntaxe des instructions est très simple et tente de se rapprocher des phrases en langage naturel. Par exemple, l'interrogation d'un fichier de produits, en utilisant le système MDQS (Management Data Query System) ressemblera presque à un dialogue :

```
DISPLAY PRODUIT TYPE QUANTITE
IF CODE = "A"
LET QUANTITE = ENTREE-SORTIE
END
```

L'emploi d'un tel langage de commandes ne requiert que la connaissance de quelques mots anglais, faciles à retenir : DISPLAY = montre, affiche ; IF = si ; LET = soit, etc.

La gestion d'une entreprise

La gestion d'une petite entreprise constitue l'une des applications typiques d'un ordinateur, dans la mesure où les fonctions à exécuter sont répétitives et impliquent de nombreuses données.
La conception d'un logiciel de gestion présente des difficultés non négligeables, dues à la nécessité de relier les données de nom-

breuses procédures successives et à l'exécution de fonctions spécifiques d'ordre économique ou fiscal.
Pour les applications d'une certaine envergure, il n'est pas avantageux d'écrire son propre logiciel : on choisira l'un des nombreux systèmes disponibles sur le marché, en l'adaptant éventuellement selon ses besoins.
Etant donné la complexité du sujet, on ne va exposer ici que les problèmes majeurs associés à la gestion d'une entreprise.
Les principales questions qu'un logiciel de gestion doit résoudre peuvent s'énumérer de la façon suivante :

TEST 19

1 / Parmi ces affirmations, lesquelles sont vraies, lesquelles sont fausses ?

Un fichier séquentiel :
a) Ne permet pas l'ajout d'enregistrements.
b) Possède une vitesse d'accès élevée.
c) Occupe moins de place qu'un fichier en accès direct.
Un fichier en accès direct :
d) Ne peut posséder d'extensions.
e) Peut être utilisé indifféremment en lecture ou en écriture.
f) Dispose de la fonction EOF(N).

2 / Que provoque l'exécution des instructions suivantes, si le fichier ESSAI n'existe pas sur l'unité de disque A ?

a) OPEN"I",1,"A:ESSAI"
b) OPEN"R",1,"A:ESSAI",120

3 / Au moyen de quelle instruction ou fonction peut-on déterminer le numéro du dernier enregistrement utilisé après une instruction PUT ou GET ?

4 / Ecrire une routine de saisie des données suivantes :

C% = code numérique (entier compris de 1 à 99).
D$ = description (20 caractères).
QT% = quantité (nombre entier).
C = coût.

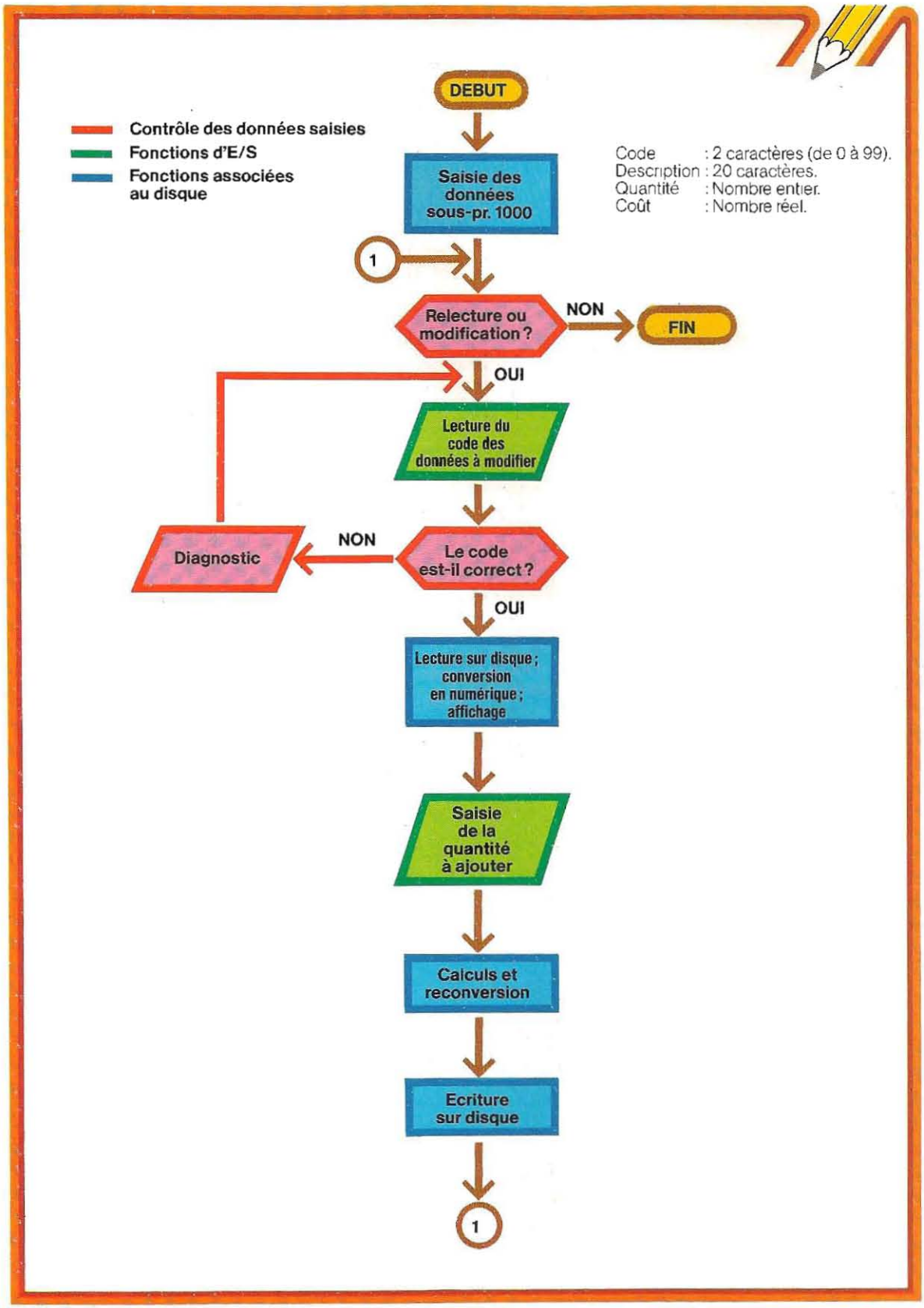
Puis d'écriture dans un fichier en accès direct nommé ESSAI, résidant sur le disque monté sur l'unité A.

5 / Compléter la routine précédente par un programme principal contenant les instructions pour :

- relire une donnée quelconque du fichier ESSAI ;
- ajouter à la quantité existante QT%, une nouvelle quantité, saisie au clavier ;
- enregistrer la modification sur disque (à la même position).

L'organigramme de la routine est illustré page 664.

Les solutions du test se trouvent page 671.


Contrôle des données saisies
Fonctions d'E/S
Fonctions associées au disque
Code : 2 caractères (de 0 à 99).
Description : 20 caractères.
Quantité : Nombre entier.
Coût : Nombre réel.
DEBUT
Saisie des données sous-pr. 1000
1
Relecture ou modification ?
NON
FIN
OUI
Lecture du code des données à modifier
Le code est-il correct ?
NON
Diagnostic
OUI
Lecture sur disque ; conversion en numérique ; affichage
Saisie de la quantité à ajouter
Calculs et reconversion
Ecriture sur disque
1

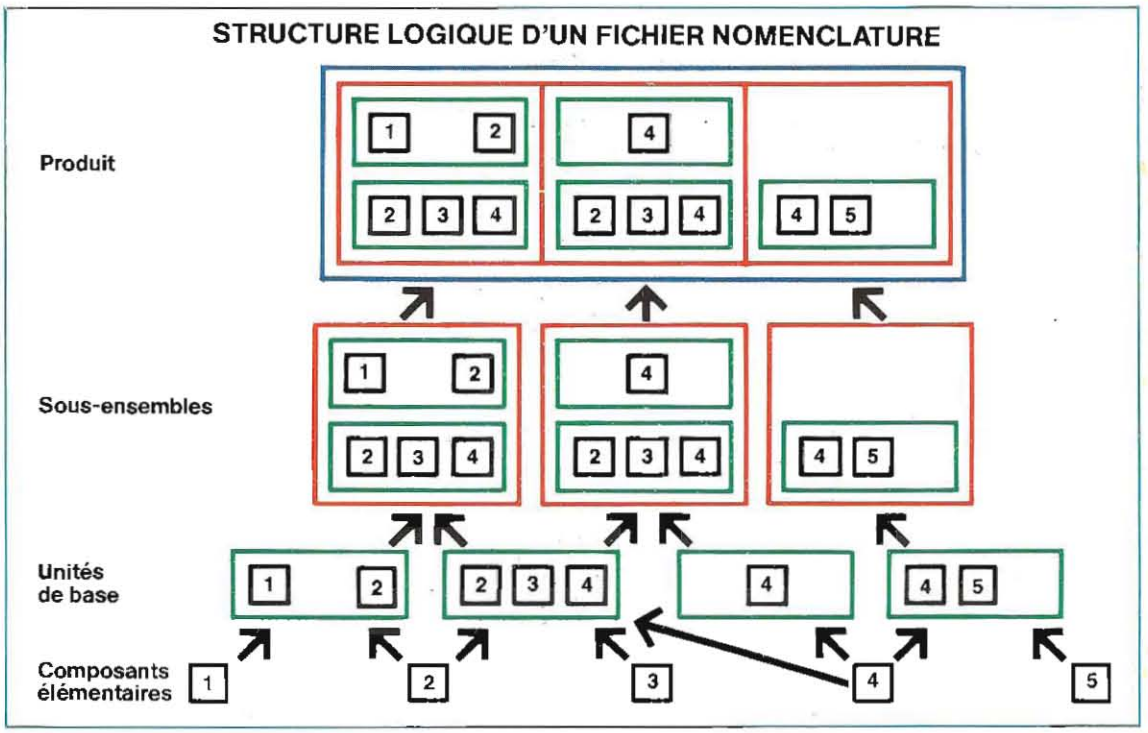
Réévaluation des stocks et inventaires
Fichier Nomenclature
Simulation
Relations entre procédures.

Réévaluation des stocks et inventaires. L'entreprise reçoit ses marchandises (matières premières) selon un calendrier préétabli ; le coût d'une marchandise donnée varie donc en fonction de sa date de réception. Au moment de la vente d'un produit fini, il s'agit de trouver un moyen d'établir son prix de revient, qui tiendra compte, entre autres, du coût de la matière première, et donc de sa date d'arrivée. Il faut donc réserver de l'espace mémoire pour stocker les dates de réception de toutes les marchandises. En outre, si le logiciel de gestion doit satisfaire à des normes d'ordre fiscal, la logique d'attribution de la valeur du produit en fonction de sa date d'arrivée ne peut être arbitraire.

Fichier Nomenclature. La gestion d'une entreprise de production s'appuie généralement sur un fichier Nomenclature, c'est-à-dire le répertoire des composants nécessaires à la fabrication du produit.
En général, un produit est formé de plusieurs sous-ensembles, chacun d'eux constitué également de plusieurs unités et chacune d'elles de plusieurs composants. On crée ainsi une échelle hiérarchisée spécifiant, pour chaque produit, le nombre et le type d'éléments nécessaires au niveau inférieur.
On comprendra aisément que les liens d'interdépendance ainsi créés engendrent des difficultés notables en ce qui concerne l'écriture des programmes. Le schéma ci-dessous représente une structure à quatre niveaux. En observant les différentes couleurs, on peut se faire une idée de la complexité des pointeurs à gérer pour maintenir l'intégrité de cette structure.

Simulation. A partir du fichier Nomenclature, on peut réaliser des traitements particuliers, qui vont simuler l'effet produit par des variations de coûts, ou par le manque de quantité en stock d'un ou plusieurs composants. En suivant les liens de dépendance logiques, le logiciel de simulation est en mesure de déterminer sur quels produits, et dans quelles proportions ces différentes causes vont se répercuter.
La simulation d'événements préétablis, et l'étude de l'effet de leur propagation sur la structure hiérarchisée constituent un excellent outil de choix stratégiques d'une entreprise. L'importance de ces méthodes est telle

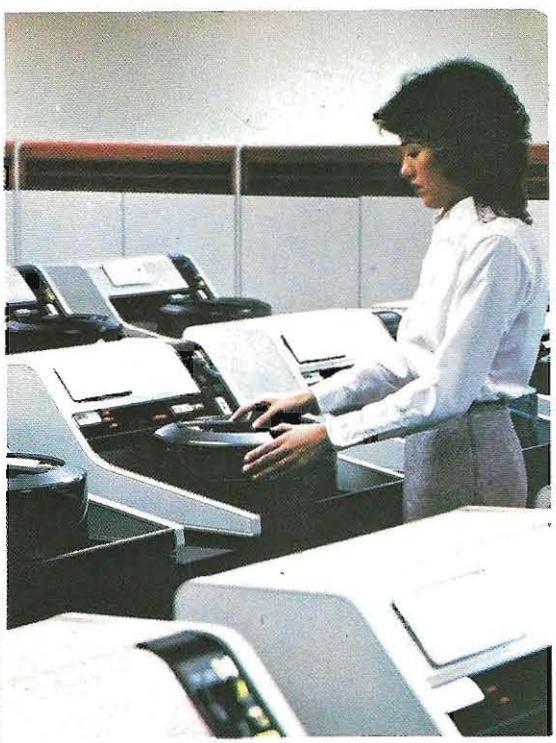
STRUCTURE LOGIQUE D'UN FICHIER NOMENCLATURE



que plusieurs créateurs de logiciels proposent actuellement des programmes de simulation de portée générale et pouvant être implantés sur micro-ordinateurs.

Relations entre procédures. La gestion d'une entreprise, même de dimensions modestes, implique un échange constant de données de provenances diverses. Plus cet échange est automatisé et rapide, plus la gestion de la structure dans son entier s'en trouve facilitée.
On comprend aisément que, pour une bonne gestion, tous les niveaux doivent interagir (la détermination des coûts, les modes de livraison, les bilans, etc.). En d'autres termes, les différents services doivent pouvoir accéder à des données communes. Dans les entreprises de grande dimension, cet aspect de la question est résolu par l'installation d'un site central qui contient les fichiers généraux. Même dans ce cas, quelques problèmes subsistent parfois : l'utilisateur final ne possède pas toujours les connaissances spécialisées qui lui offriraient un accès aisé aux procédures générales et, très souvent, il préfère développer ses propres traitements particuliers, qui ne s'intègrent pas dans la bibliothèque de programmes généraux.
Les micro-ordinateurs peuvent résoudre aisément ces problèmes, en offrant une solution facilement intégrable au travail de bureau.
L'emploi d'un logiciel « clés en main » par l'utilisateur final ne requiert pas de connaissances particulières et peut être appris en quelques jours. En outre, grâce à des réseaux appropriés, les différents postes de travail peuvent dialoguer entre eux ou avec le site central, et échanger des informations en temps réel.
L'évolution prévisible de la fonction de programmeur sur ces machines configurées en réseau sera, dans un futur proche, celle d'ingénieur-système : maîtrisant la connaissance des capacités du matériel et des possibilités des différents logiciels d'application, il sera en mesure de conseiller les utilisateurs dans leur choix et leurs installations.

Unité de disque dans un centre de calcul.



J. Pickerell/Marka

Exemple d'application : la gestion d'un répertoire

Afin de résumer les concepts exposés au sujet de la gestion des fichiers, nous voudrions maintenant décrire la méthode de création d'un programme de gestion d'une rubrique. Quel que soit le fichier à gérer, et quelle que soit l'application, les fonctions à exécuter sont les suivantes :

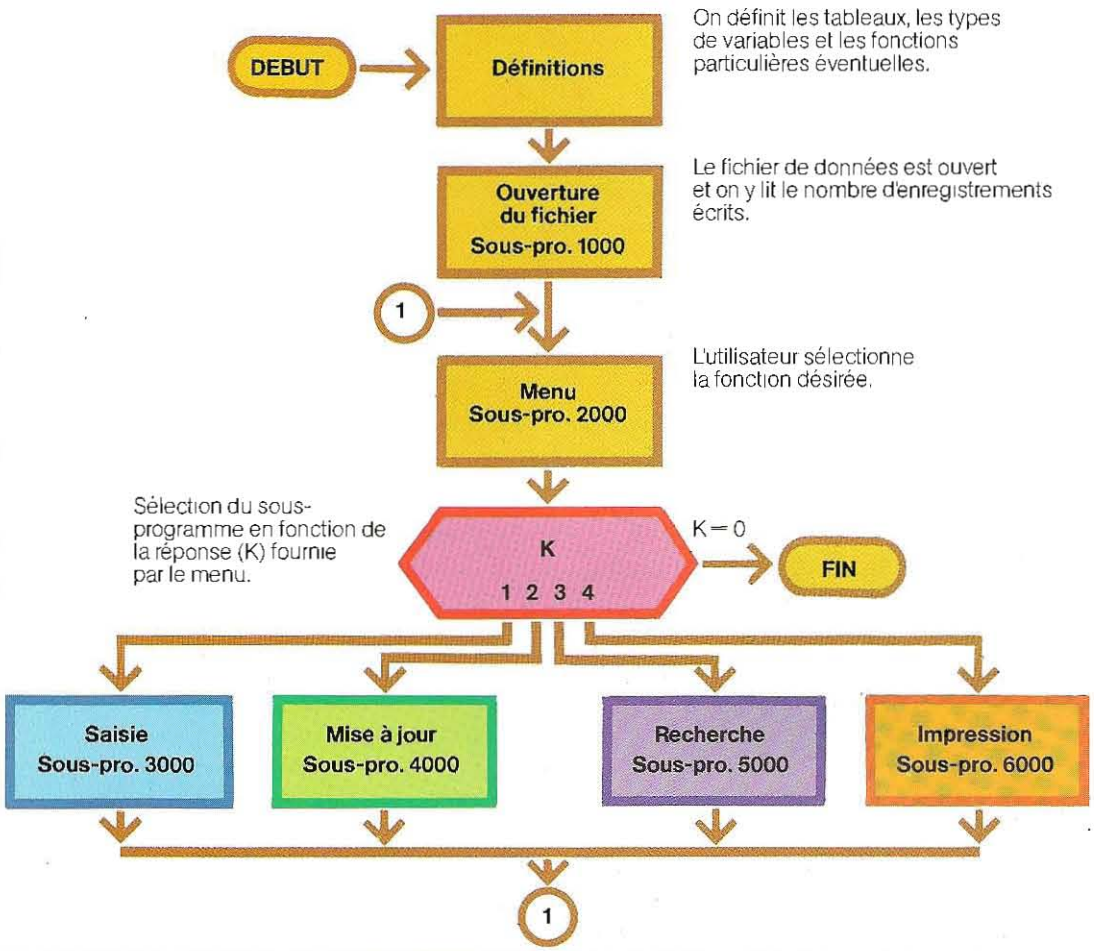
- saisie des nouvelles données ;
- modification des données existantes ;
- recherche d'une donnée en fonction de certains paramètres ;
- impression.

Ces quatre fonctions seront appelées par un menu, dans le programme principal de gestion du fichier, et seront exécutées par autant de sous-programmes.
La page 667 illustre l'organigramme général : le programme principal ne contient que des définitions et l'appel aux sous-programmes ; toutes les fonctions du programme sont exécutées dans les sous-programmes.

Ouverture du fichier (sous-programme 1000)

Ce sous-programme ouvre le fichier, définit les champs et lit le numéro du dernier enregistrement rempli. Le fichier de taille quelconque peut en effet n'être occupé qu'en partie.

GESTION D'UN FICHIER SUR DISQUE. ORGANIGRAMME GENERAL



Pour écrire un nouvel enregistrement, il faut connaître le numéro du dernier enregistrement rempli ; on incrémente de 1 cette valeur. On crée un fichier de la taille maximale prévue et on utilise son dernier enregistrement pour y stocker le nombre d'enregistrements déjà remplis. Ce dernier enregistrement devra être mis à jour au fur et à mesure que de nouvelles données seront entrées (voir p. 668).
Dans notre application, le fichier contiendra les champs suivants :

Nom	= 20 caractères
Rue	= 20 caractères
Téléphone	= 12 caractères
Ville	= 20 caractères
Longueur d'un enregistrement	= 72 caractères

La création du fichier s'effectue de façon immédiate, en entrant au clavier l'instruction :

```
OPEN"R",1,"A:DATA",72
```

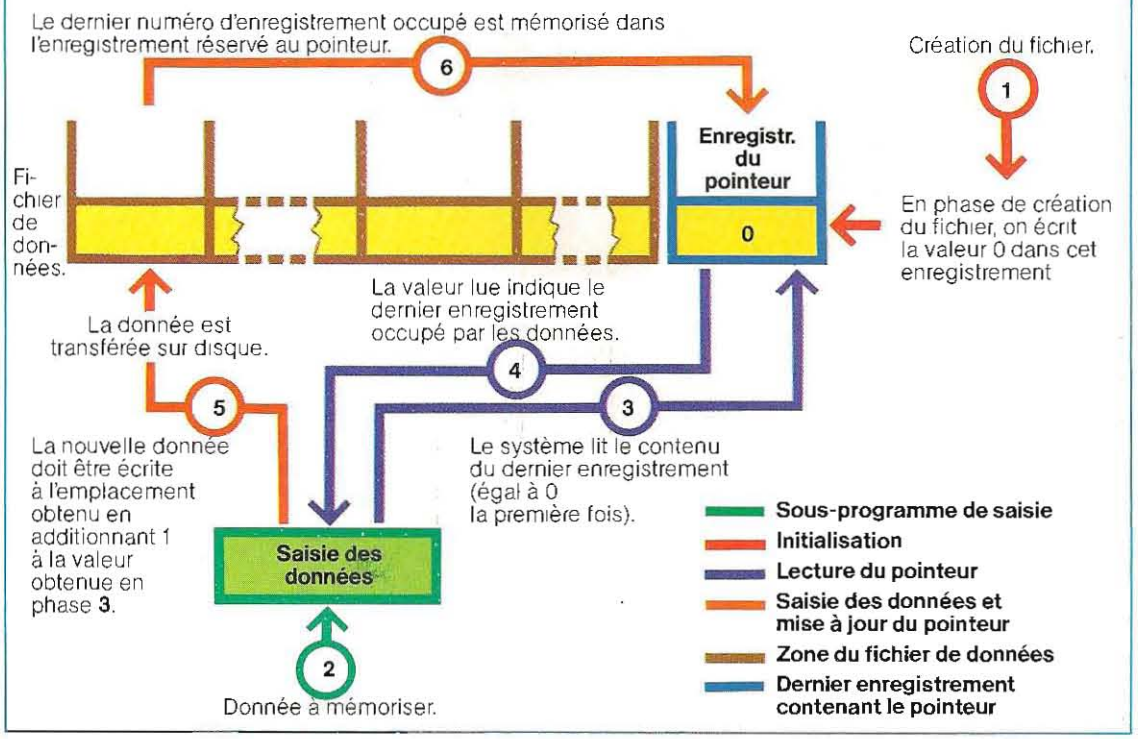
Le système mémorise alors le nom du fichier dans le répertoire du disque A et lui attribue une taille initiale. Si nous avons au maximum 100 groupes de données à stocker, il faut écrire dans l'enregistrement 101 (virtuellement le dernier) la valeur 0 : au début, le fichier est vide et le premier enregistrement disponible porte le numéro 1 (0+1=1).
La création du fichier et l'initialisation du dernier enregistrement peuvent être réalisées par un programme utilitaire, de portée générale, dont le listing est illustré à la page 668, en bas.

Une fois que la procédure de création et d'initialisation est réalisée, on peut passer à l'écriture du sous-programme 1000, qui devra exécuter les fonctions suivantes :

- ouverture du fichier ;
- lecture du dernier enregistrement.

La structure de ce sous-programme est très simple ; le listing correspondant est reproduit en haut de la page 669.
Comme on le voit, cette routine est paramétrée, et tous les paramètres lui sont transmis par le programme principal. De cette façon, elle peut être utilisée par n'importe quelle application.

GESTION DES FICHIERS. UTILISATION DU POINTEUR SUR LE DERNIER ENREGISTR.



CREATION DU FICHIER ET INITIALISATION DU DERNIER ENREGISTREMENT

```
10 ' ** Programme utilitaire de creation du fichier
20 '    et d'initialisation du pointeur (dernier enregistrement) **
30 '
40 '       PROGRAMME CREAT
50 INPUT "NOM DU FICHIER ";NOM$
60 INPUT "SUR QUELLE UNITE DISQUE (A:/B:) ";D$
70 IF D$<>"A:" AND D$<>"B:" GOTO 60
80 NM$=D$+NOM$
90 INPUT "Longueur des enregistrements (Nb de caracteres) ";L
100 IF L>128 GOTO 90
110 OPEN "R",1,NM$,L
120 FIELD 1,L AS BF$
130 INPUT "Nombre d'enregistrements maximum ";NE
140 MX=NE+1          ' Numero du dernier enregistrement
150 B$="0" : C$=STRING$(L,B$)       ' C$="000...0"
160 BF$=C$
170 PUT 1,MX         ' Ecriture pointeur
180 PRINT "Le fichier ";NM$;" est créé"
190 PRINT "Longueur=";NE
200 PRINT "Pointeur en position=";MX
210 END
```

ROUTINE D'OUVERTURE DU FICHIER

```
1000 ' ** Subroutine d'ouverture du fichier **
1010 '
1020 '   Entrees =
1030 '      NM$ = nom complet du fichier (A : DONNEE par exemple)
1040 '        L = longueur des enregistrements
1050 '      NE = longueur du fichier (nombre d'enregistrements)
1060 '      BF$ = buffer associe au fichier
1070 '
1080 OPEN "I",1,NM$ : CLOSE 1       ' Verification de l'existence du fichier
1090 OPEN "R",1,NM$,L
1100 FIELD 1,L AS BF$
1110 MX=NE+1
1120 GET 1,MX              ' Lecture du pointeur
1130 A$=BF$                ' La chaine est affectee a A$
1140 MAX=VAL(A$)           ' Conversion en numerique
1150 '
1160 ' En sortie, MAX contient la valeur stockee dans le dernier
1170 ' enregistrement du fichier.
1180 ' C'est le numero du dernier enregistrement occupe
1190 ' Il faut ajouter 1 pour obtenir le numero du premier enregistrement libre
1200 '
1210 RETURN
```

ROUTINE DE SAISIE DES DONNEES

```
3000 ' ** SAISIE DES DONNEES **
3010 '    Entrees =
3020 '       MAX = numero du dernier enregistrement occupe
3030 '        NE = longueur du fichier
3035 '      BF$ = buffer associe au fichier
3040 ' Les instructions de saisie sont propres a cette application
3042 ' Il faudra eventuellement les modifier en fonction
3044 ' des exigences particulieres du fichier traite
3050 ' * Saisie des donnees au clavier
3060 INPUT "NOM ";N$
3070 INPUT "RUE ";R$
3080 INPUT "TELEPHONE ";T$
3090 INPUT "VILLE ";V$
3100 ' * Preparation de l'enregistrement
3110 A1$=SPACE$(20)        ' Creation des quatre chaines d'espaces
3120 A2$=SPACE$(20)
3130 A3$=SPACE$(12)
3140 A4$=SPACE$(20)
3150 LSET A1$=N$           ' Cadrage a gauche des donnees
3160 LSET A2$=R$           ' dans les champs d'espaces,
3170 LSET A3$=T$           ' afin de faciliter des comparaisons ulterieures
3180 LSET A4$=V$
3190 ' L'instruction LSET tronque les caracteres excedentaires
3192 '
3200 A$=A1$+A2$+A3$+A4$       ' Concatenation
3210 BF$=A$
3220 M=MAX+1                  ' Premier enregistrement libre
3230 PUT 1,M                  ' Ecriture
3240 IF MODE=1 GOTO 3310      ' Si MODE=1, on vient de la routine 4000
3250 '                          donc on ne modifie pas le pointeur
3255 '
3260 MAX=MAX+1                ' Incrementation du pointeur
3270 B$=STR$(MAX)             ' Conversion en chaine
3280 A$=SPACE$(L-LEN(B$))     ' Creation d'une chaine d'espaces
3290 BF$=A$+B$                ' pour completer l'enregistrement
3295 MX=NE+1                  ' Numero du dernier enregistrement
3300 PUT 1,MX
3310 RETURN
```

Insistons sur le fait que l'instruction 1090 (ouverture en mode « I » et fermeture du fichier) a seulement pour but de vérifier l'existence du fichier : si l'on ouvre en mode « R » un fichier non créé, on en crée un nouveau non initialisé.

Menu (sous-programme 2000)

En ce qui concerne ce sous-programme, le lecteur se référera à l'un de ceux déjà présentés lorsque nous avons abordé le sujet d'un point de vue général. L'unique changement réside dans les messages à prévoir.

Saisie (sous-programme 3000)

Les fonctions à exécuter sont les suivantes :
– lecture des données depuis la console ;
– leur écriture sur le premier enregistrement disponible ;
– mise à jour du dernier enregistrement afin de prendre en compte la nouvelle donnée saisie.
Ce sous-programme est listé en bas de la page 669.

Mise à jour (sous-programme 4000)

Elle doit exécuter les fonctions suivantes :
– lecture, à partir du clavier, du numéro d'enregistrement à mettre à jour ;
– lecture des données concernées, depuis le disque ;
– affichage de ces données ;
– lecture des nouvelles valeurs (modification des données existantes) ;
– écriture des nouvelles données dans le même enregistrement.
Le listing correspondant est illustré ci-dessous. Pour le programmer, on peut utiliser en partie la routine de saisie.
La différence entre les phases de saisie et de mise à jour réside dans l'emplacement différent où sont écrites les données : durant la saisie, chaque donnée est placée dans un nouvel enregistrement, alors qu'en mise à jour,la donnée doit être récrite sur le même enregistrement, et le pointeur de fin de fichier (numéro du dernier enregistrement) ne doit pas être incrémenté.
Le sous-programme 4000 pourrait donc direc-tement appeler la routine 3000, après avoir positionné un indicateur (sur le listing, KS) indiquant au sous-programme 3000 de ne pas mettre à jour le pointeur sur les données.
Il faut également modifier le pointeur (MAX) qui désigne l'enregistrement à remplir.
On pourrait également récrire directement dans le sous-programme 4000 les instructions nécessaires (lignes 3050 à 3230). L'accroissement de longueur du programme est négligeable et on y gagne en clarté.

ROUTINE DE MISE A JOUR

```
4000 ' ** MISE A JOUR **
4005 '
4010 INPUT "Numero d'enregistrement a modifier";N
4020 IF N>NE GOTO 4010      ' Erreur
4030 GET 1,N
4040 A$=BF$
4050 ' * Extraction des champs
4060 N$=LEFT$(A$,20)
4070 R$=MID$(A$,21,20)
4080 T$=MID$(A$,41,12)
4090 V$=RIGHT$(A$,20)
4100 PRINT "DONNEES LUES DANS L'ENREGISTREMENT ";N
4110 PRINT C$:PRINT R$
4120 PRINT T$:PRINT V$
4130 ' * Pour la saisie des nouvelles donnees,
4140 '   on appelle la subroutine 3000 de saisie
4150 '   on positionne un Flag (MODE) a 1 pour que
4160 '   les modifications ne soit pas considerees comme une nouvelle donnee
4170 MS=MAX          ' Sauvegarde du nombre d'enregistrements
4180 MAX=N-1         ' Prepare la position d'ecriture
4190 MODE=1
4200 GOSUB 3000
4210 MAX=MS          ' On restaure la valeur du pointeur
4220 RETURN
```

Solutions du test 19

1 / a) est vrai en précisant ceci : les ajouts sont possibles au moyen d'un fichier intermédiaire (et non directement) ;
b), c), d) sont faux ; e) vrai ; f) faux : la fonction EOF(N) est employée pour indiquer la fin d'un fichier séquentiel.

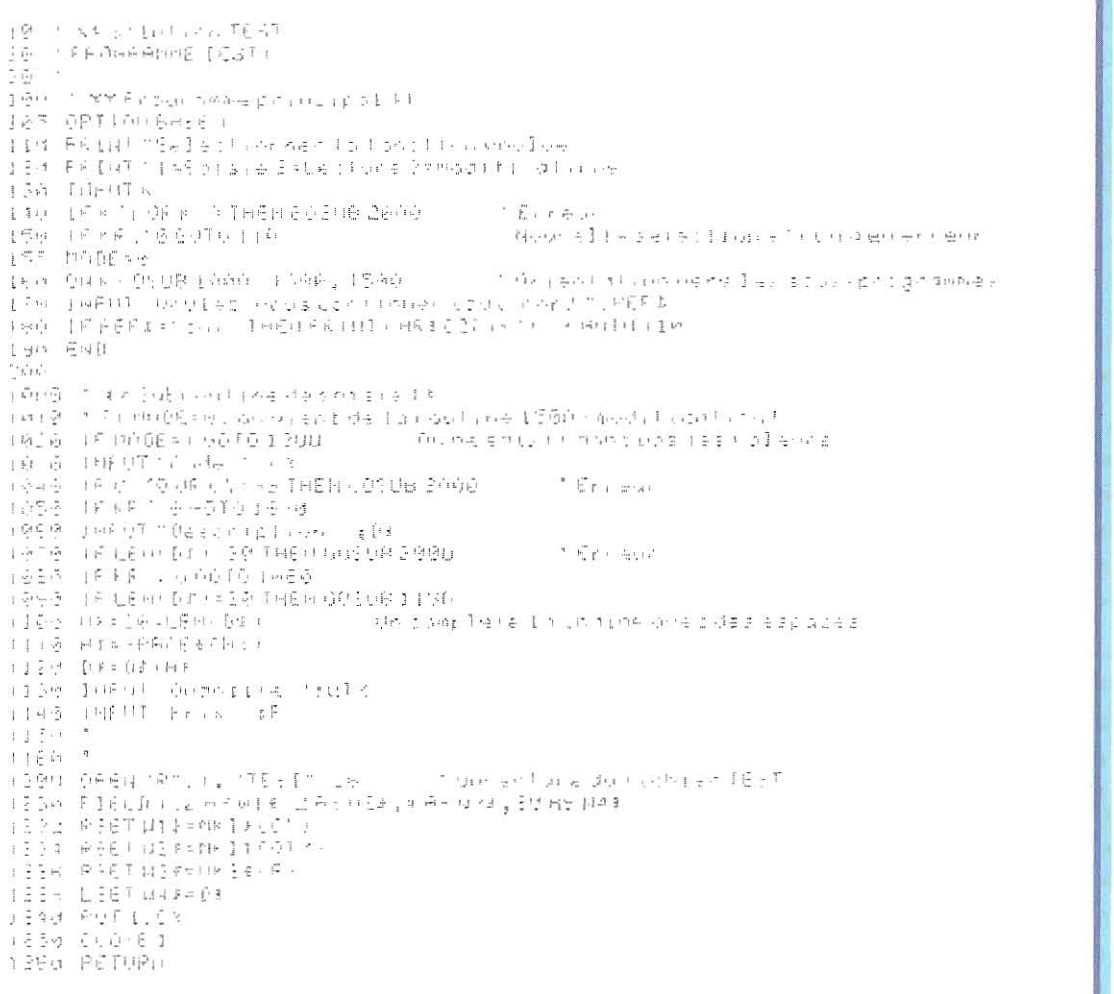
---

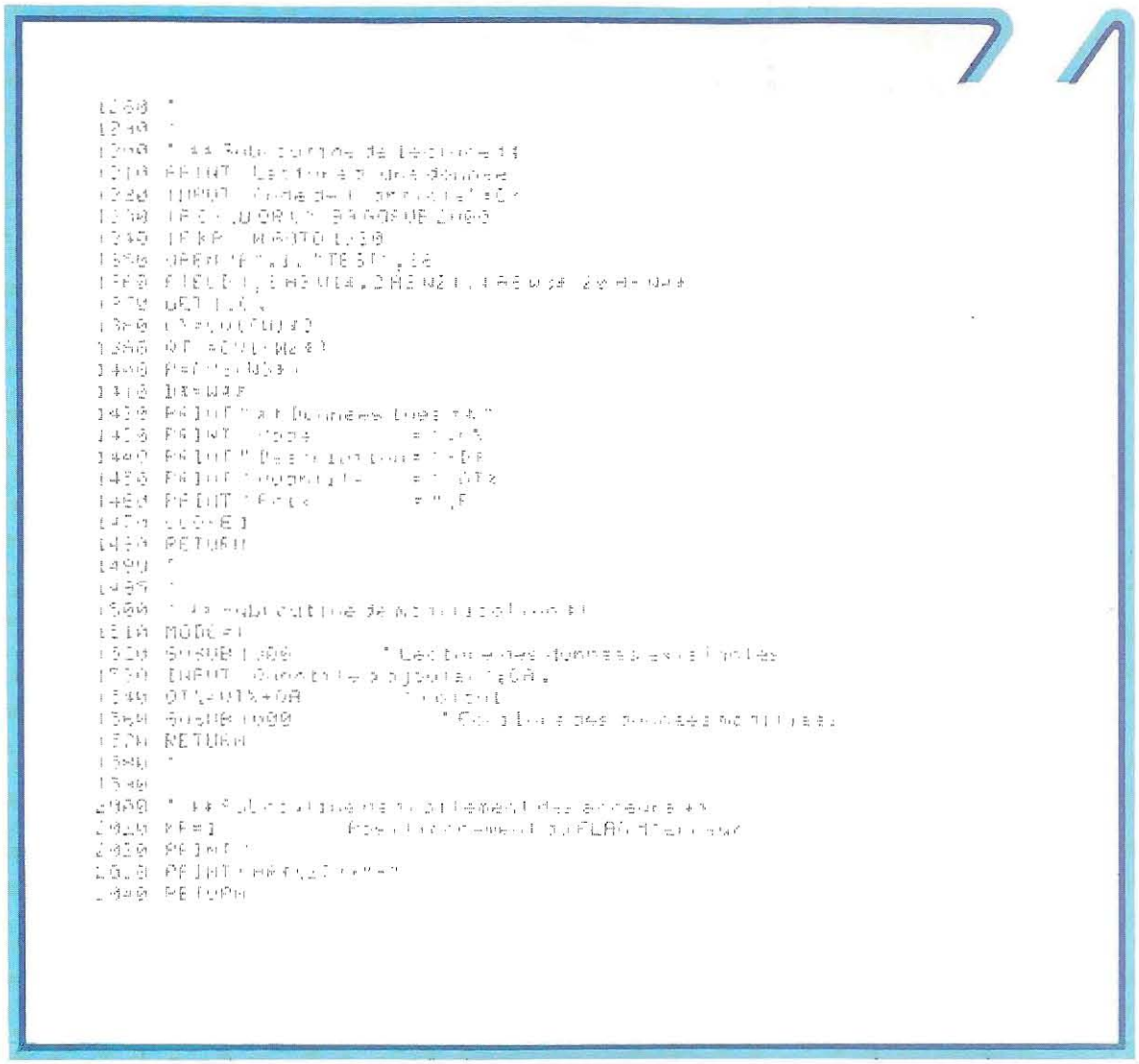
2 / a) génère une erreur système et stoppe l'exécution du programme ;
b) créé le fichier.

---

3 / On doit utiliser la fonction LOC(N).

4-5 / Le listing du programme répondant aux questions 4 et 5 est illustré ci-dessous et page suivante.
Rappelons que cette solution n'est donnée qu'à titre indicatif. L'écriture dans le détail des instructions d'un programme est généralement très subjective. La forme d'un programme, les instructions ou les fonctions employées dépendent beaucoup des habitudes du programmeur.





Recherche (sous-programme 5000)

En général, la complexité des sous-programmes de recherche dépend étroitement du volume des données à traiter. Si le fichier de données est de grande dimension, on aura intérêt à l'ordonner au préalable, pour faciliter ensuite la recherche. S'il s'agit d'un répertoire, par exemple, on classera les noms par ordre alphabétique ; la recherche d'un nom particulier sera ensuite effectuée très rapidement.
Dans les applications sur micro-ordinateur, le volume des données constitue rarement un obstacle ; dans ce cas, il sera plus simple d'examiner le fichier en entier, sans le trier au préalable, et d'extraire les enregistrements répondant aux critères recherchés, au fur et à mesure qu'ils se présentent. Par cette technique, on dépense plus de temps que nécessaire, puisque tous les enregistrements sont lus (même ceux qui auraient été éliminés avec un fichier ordonné), mais la différence en temps d'exécution ne justifie pas toujours de se lancer dans les difficultés d'une programmation de haut niveau. Pour avoir un ordre de grandeur, sachant que le système lit environ une centaine d'enregistrements à la minute, on peut considérer que le gain de temps éventuel est de l'ordre de la minute.
La complexité des sous-programmes de recherche dépend en fait des degrés de liberté nécessaires à l'application, c'est-à-dire du nombre de paramètres que l'on veut utiliser comme clés d'accès aux données.

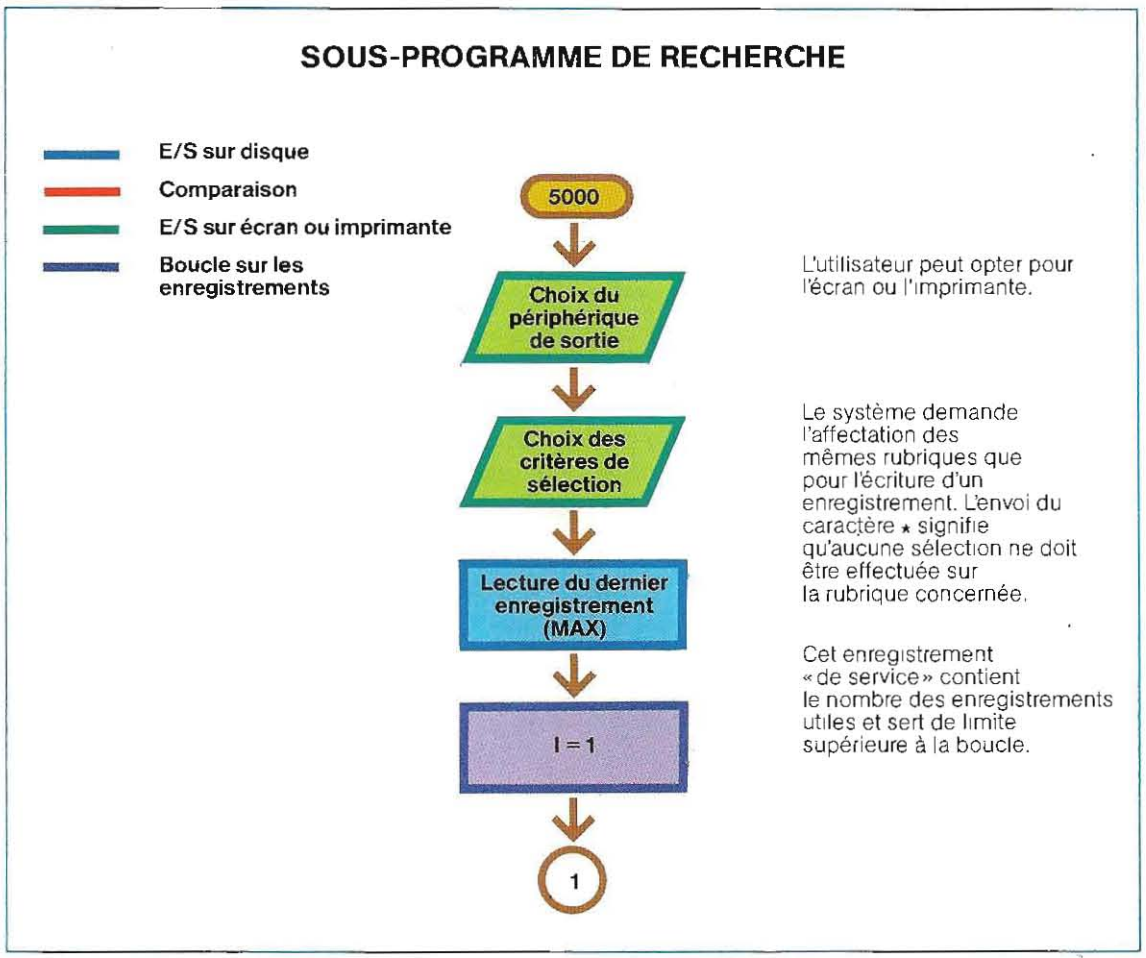
Par exemple, dans notre fichier d'adresses, nous pouvons choisir comme élément de sélection le nom seul, ou sélectionner également sur la ville. Une telle méthode impliquerait l'écriture d'un sous-programme spécifique pour chaque critère de recherche.
On aura plutôt intérêt à paramétrer le sous-programme de recherche, puis à rendre ceci transparent pour l'utilisateur en affichant la liste des rubriques de chaque enregistrement. L'utilisateur peut alors choisir le critère de sélection, en entrant la rubrique concernée.
Cette méthode a été utilisée dans le sous-programme 5000 (voir organigramme ci-dessous) où toutes les rubriques d'un enregistrement sont passées en revue. L'utilisateur peut alors soit exclure la rubrique des critères de choix (caractère *), soit la valider en entrant la valeur à rechercher pour ce critère.

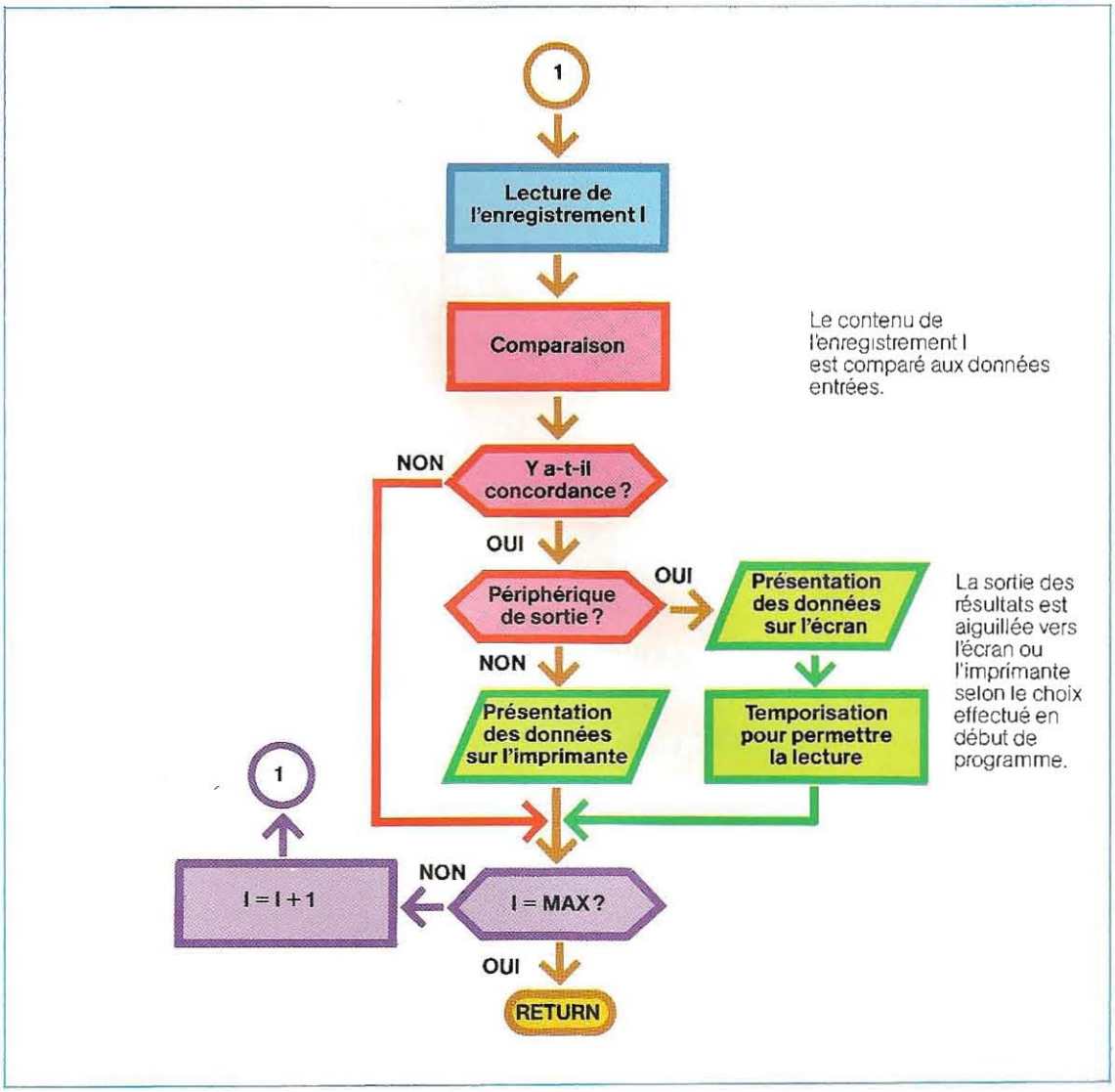
Impression (sous-programme 6000)

Le sous-programme d'impression doit permettre l'édition sur papier de tous les enregistrements du fichier. Il sera donc constitué d'une boucle allant du premier au dernier enregistrement écrit. Le numéro de ce dernier enregistrement est mémorisé dans une zone supplémentaire du fichier. Cette procédure ne présentant pas de difficultés particulières, nous laissons au lecteur le soin de tracer lui-même l'organigramme selon la structure qui lui semble la mieux adaptée à l'édition. On pourrait également utiliser une routine d'impression paramétrée dont nous présenterons les modalités par la suite.

Opérations d'impression particulières

Le programme que nous venons de décrire est adaptable à de multiples applications et ne nécessite pour cela que des modifications mineures (nombre, signification et longueur des variables). Par contre, la routine d'impression est difficilement aménageable pour certains types d'édition. Les fichiers d'adresses,





par exemple, sont souvent utilisés pour des expéditions en nombre de courrier ou d'imprimés. Les sorties du programme, c'est-à-dire les noms et adresses, doivent alors être réalisées sur un support directement utilisable.
A cet effet on trouve dans le commerce des bandes de papier sulfurisé, adaptées au système d'entraînement par picots et sur lesquelles sont disposées des rangées d'étiquettes autocollantes. Ce système est beaucoup plus rapide et universel que l'impression d'enveloppes sur support spécifique (enveloppes, bandes de routage, etc.).
Le sous-programme d'impression doit toutefois être modifié en conséquence car on doit tenir compte d'un format très particulier dû à la disposition des étiquettes. Il faut, en effet, structurer les sorties de façon à faire imprimer plusieurs noms sur une même ligne – un sur chaque étiquette, puis les adresses correspondantes sur la ligne suivante et enfin les villes sur une troisième ligne. La routine d'impression doit donc opérer sur des groupes d'enregistrements équivalents au nombre d'étiquettes présentes sur la largeur de la feuille. Par conséquent, il faudra prévoir deux boucles imbriquées : une principale assurant la saisie de tous les enregistrements à partir du disque et une secondaire qui regroupera les données par une ligne d'impression et effectuera les sorties rubrique par rubrique.
Ce principe d'impression des étiquettes est illustré par le schéma de la page 675 pour une rangée de quatre étiquettes. Dans cet

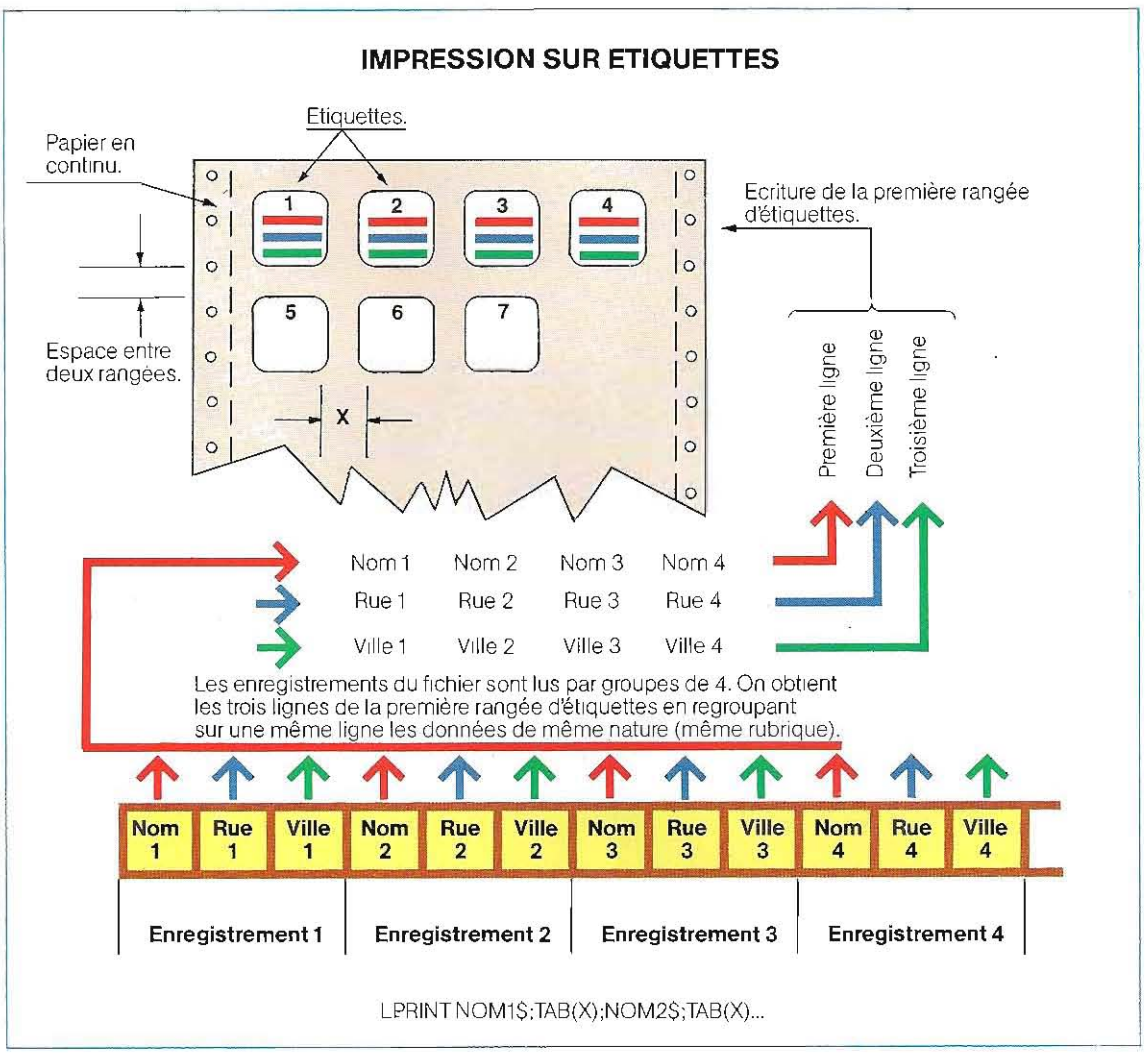
exemple, l'incrément de la bouche principale est 4 et les trois rubriques correspondant aux noms et prénoms, aux rues et aux villes sont traitées par une boucle intérieure. Le module d'impression doit contenir des instructions de tabulation permettant le centrage de l'adresse sur les étiquettes.

L'agencement des données

Pour gérer des fichiers, il est indispensable de disposer de routines spécialisées dans le tri des données à partir d'une ou plusieurs rubriques.

Ces routines d'usage général, appelées programmes de tri, sont souvent fournies systématiquement avec le matériel et sont, dans tous les cas, très faciles à se procurer. Il peut cependant s'avérer utile de disposer de sous-programmes personnels mieux adaptés à un usage particulier, notamment quand les données à traiter sont en nombre limité. De même, il est souvent intéressant de pouvoir ordonner des tables présentes dans la mémoire centrale. Or, les utilitaires de tri travaillent sur des fichiers conservés dans des mémoires de masse et leur emploi en RAM est beaucoup moins performant. Signalons toutefois qu'un algorithme de tri écrit en Basic n'est envisageable qu'avec un Basic compilé. L'interprétation demande en effet beaucoup trop de temps pour ce type d'applications, la seule alternative étant alors le langage Assembleur.

Nous allons maintenant présenter deux programmes de tri. Le premier travaille en mémoire et ne peut traiter un fichier externe que dans la mesure où celui-ci peut être transféré intégralement dans la mémoire. Le

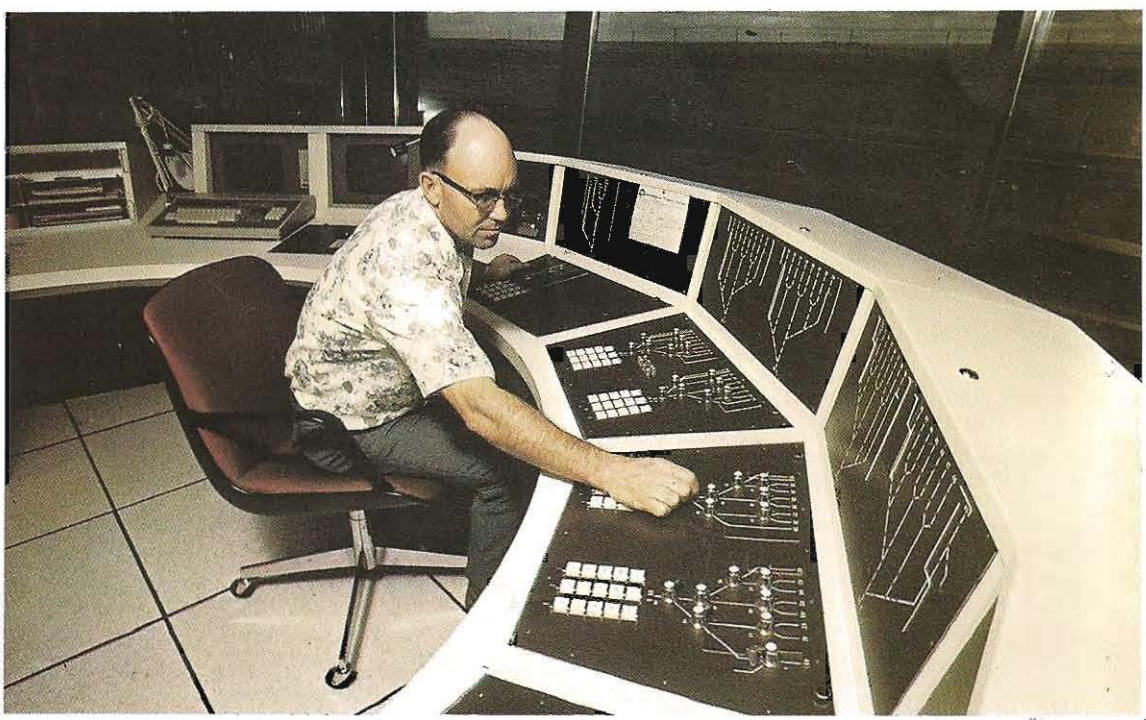


second est par contre conçu pour des fichiers sur disque et utilise une méthode de recherche dichotomique. C'est un sous-programme performant adapté à la gestion de fichiers de taille moyenne.

Tri en mémoire

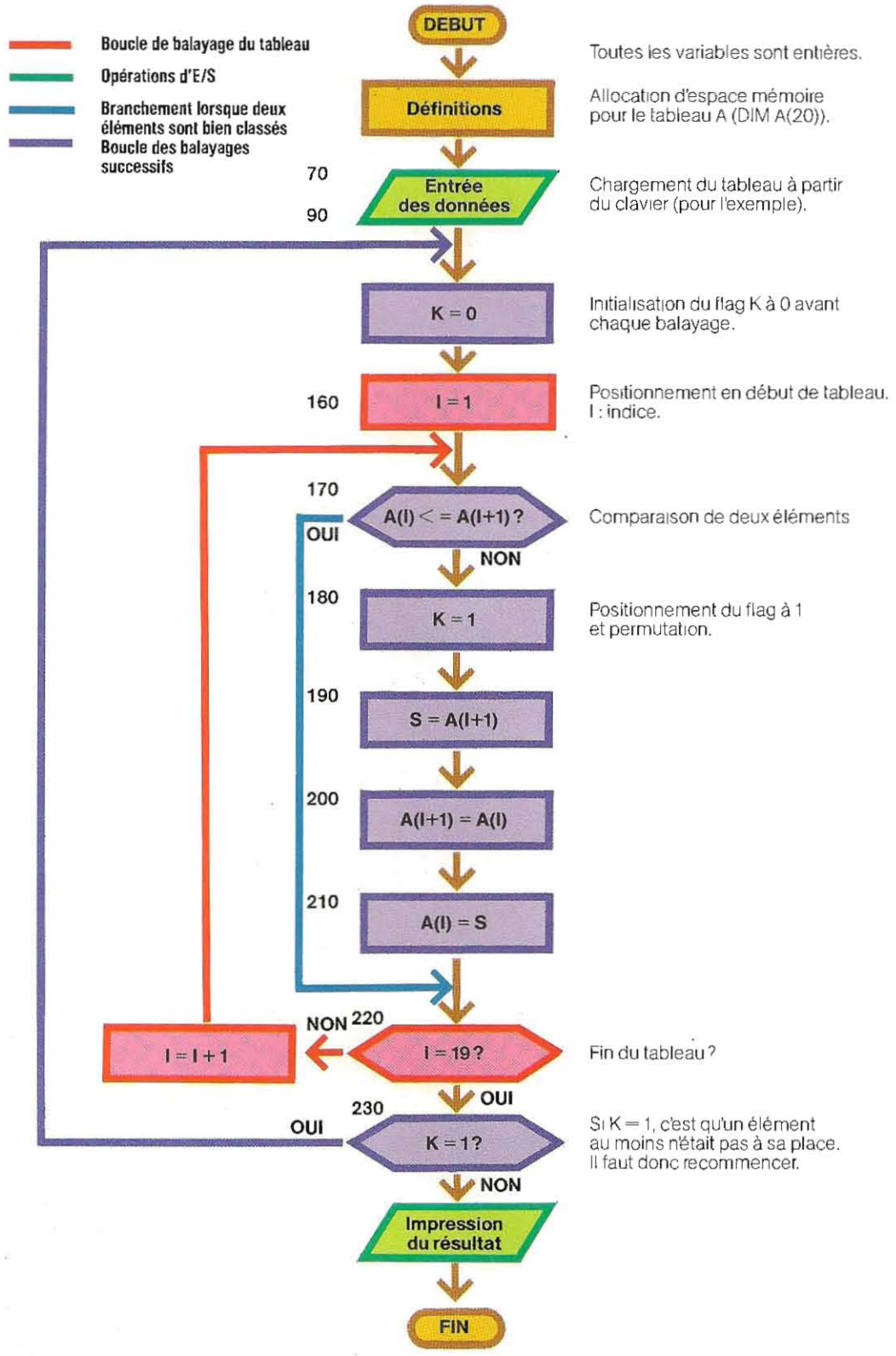
L'organigramme représenté page 677 est celui d'un programme qui classe par ordre croissant le contenu d'un tableau à 20 éléments : A(20).
Le classement s'effectue par comparaisons successives de chaque élément du tableau avec l'élément suivant (d'indice immédiatement supérieur) : si le premier est le plus grand, les deux éléments sont intervertis. Les valeurs les plus importantes du tableau sont ainsi peu à peu acheminés vers la fin de celui-ci. Ensuite un nouveau balayage a lieu, et ainsi de suite jusqu'à ce qu'un passage n'ait donné lieu à aucune inversion. Chaque élément du tableau est alors inférieur au suivant et le tri est terminé. Pratiquement, on utilise un flag K qui est positionné à 1 dès qu'une inversion a lieu. La fin d'un balayage (comparaison des deux derniers éléments) assortie d'un flag K=0 indique donc qu'aucune permutation n'a eu lieu et qu'on peut donc sortir de la boucle pour terminer le programme.
Cette routine, dont le listage se trouve pages 678 et 679, n'a qu'une simple valeur d'illustration. Elle pourrait, en effet, être simplifiée par une utilisation plus rationnelle du flag K.
Cet indicateur ne peut prendre que deux valeurs : 1 et 0, associées à Vrai et Faux puisque son positionnement est lié à une condition (la réalisation d'une permutation au cours d'un passage ou l'achèvement du classement). Une boucle, devant être répétée tant qu'une condition n'est pas remplie, est un cas typique d'application de l'instruction WHILE... WEND.
Vous trouverez donc dans la seconde partie de la page 679 une autre version du programme où l'on a utilisé cette instruction. Une seconde modification a été introduite dans ce programme avec l'emploi de la fonction SWAP. Dans la première version, on avait dû prévoir une mémoire de travail (S) pour permuter le contenu de deux éléments successifs du tableau A(I) et A(I+1). Avec l'instruction SWAP, la gestion de cette mémoire est prise en charge par le système et l'écriture s'en trouve allégée.
La permutation du contenu de deux variables nécessite, en effet, le transfert momentané de

Salle de contrôle d'une importante gare.



K. Reese/Marka

ORGANIGRAMME DE CLASSEMENT D'UN TABLEAU



CLASSEMENT D'UN TABLEAU (PREMIERE VERSION)

```
10 ' ** PROGRAMME CLASS1
20 OPTION BASE 1
30 DIM A(20)
40 ' ** ENTREE DES DONNEES A CLASSER
50 '
60 FOR I=1 TO 20
70 PRINT " Donnee N.:";I,
80 INPUT "VALEUR     ";A(I)
90 NEXT I
100 '
110 ' ** IMPRESSION DES VALEURS ENTREES
120 '
130 LPRINT " TABLEAU ENTRE "
140 LPRINT
150 FOR I=1 TO 20
160 LPRINT " I = 1 ";I," Valeur = ";A(I)
170 NEXT I
180 '
190 ' *** CLASSEMENT
200 '
210 K=0
220 FOR I=1 TO 19
230 IF A(I)<=A(I+1) GOTO 270
240 K=1
250 S=A(I+1)
255 A(I+1)=A(I)
260 A(I)=S
270 NEXT I
280 IF K=1 GOTO 210
290 '
300 ' ** IMPRESSION DES DONNEES CLASSEES
310 LPRINT " TABLEAU CLASSE "
320 LPRINT
330 FOR I=1 TO 20
340 LPRINT " I =";I," Valeur = ";A(I)
350 NEXT I
360 END
```

```
 TABLEAU ENTRE

I = 1           Valeur = 32
I = 2           Valeur = 25
I = 3           Valeur = 18
I = 4           Valeur = 0
I = 5           Valeur = 5
I = 6           Valeur = 48
I = 7           Valeur = 98
I = 8           Valeur = 74
I = 9           Valeur = 125
I = 10          Valeur = 2
I = 11          Valeur = 56
I = 12          Valeur = 36
I = 13          Valeur = 14
I = 14          Valeur = 25
I = 15          Valeur = 12
I = 16          Valeur = 10
I = 17          Valeur = 91
I = 18          Valeur = 58
I = 19          Valeur = 33
I = 20          Valeur = 51
```

```
TABLEAU CLASSE

I = 1          Valeur = 0
I = 2          Valeur = 2
I = 3          Valeur = 5
I = 4          Valeur = 10
I = 5          Valeur = 12
I = 6          Valeur = 14
I = 7          Valeur = 18
I = 8          Valeur = 25
I = 9          Valeur = 25
I = 10         Valeur = 32
I = 11         Valeur = 36
I = 12         Valeur = 48
I = 13         Valeur = 56
I = 14         Valeur = 58
I = 15         Valeur = 61
I = 16         Valeur = 74
I = 17         Valeur = 77
I = 18         Valeur = 91
I = 19         Valeur = 98
I = 20         Valeur = 125
```

CLASSEMENT D'UN TABLEAU (SECONDE VERSION)

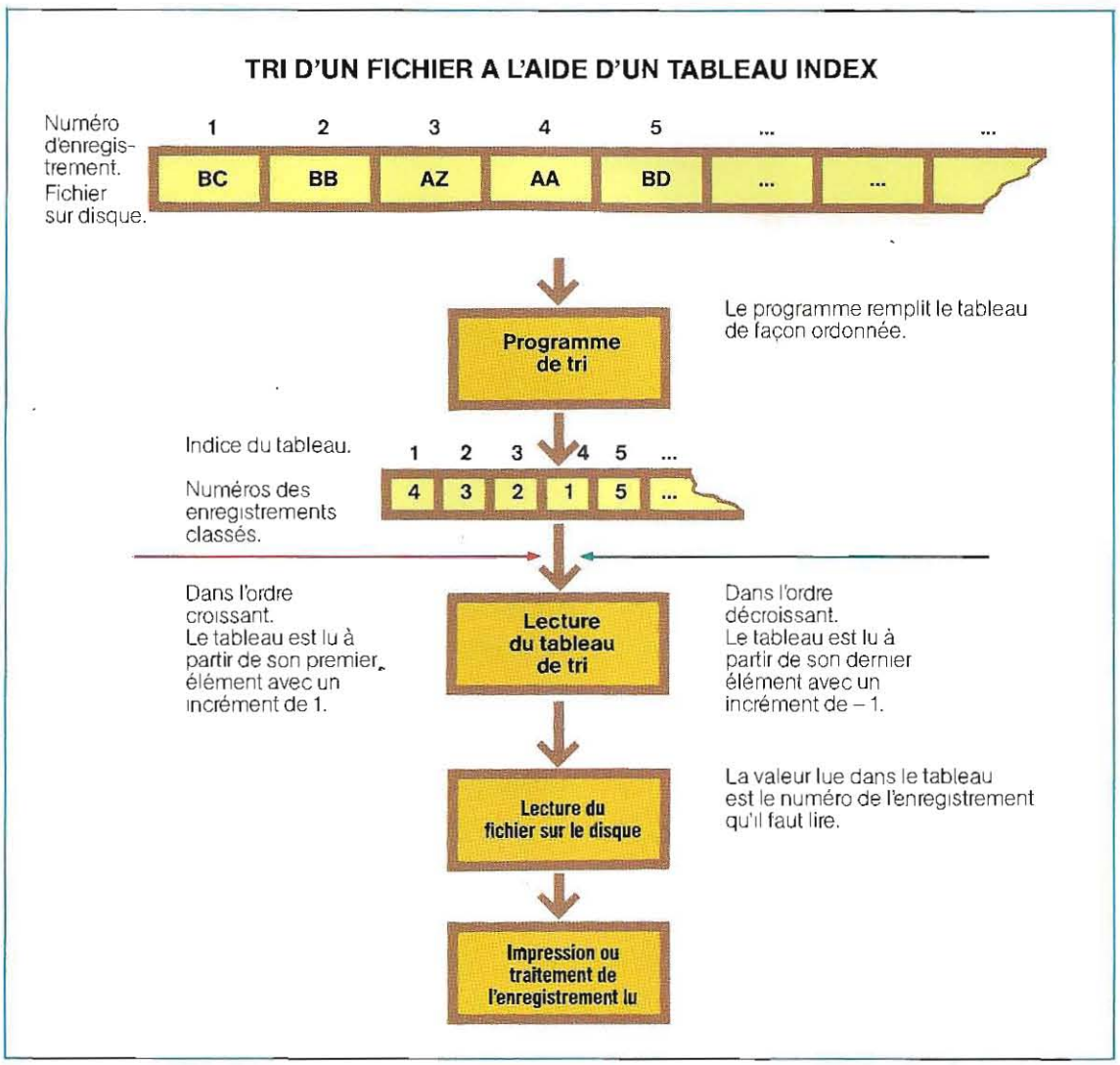
```
10 ' ** PROGRAMME CLASSE
20 OPTION BASE 1
30 DIM A(20)
40 ' ** ENTREE DES DONNEES A CLASSER
50
60 FOR I=1 TO 20
70 PRINT " Donnee N. : ";I,
80 INPUT "VALEUR      ";A(I)
90 NEXT I
100 '
110 ' ** IMPRESSION DES VALEURS ENTREES
120 '
130 LPRINT " TABLEAU ENTRE "
140 LPRINT
150 FOR I=1 TO 20
160 LPRINT " I = ";I," Valeur = ";A(I)
170 NEXT I
180 '
190 ' *** CLASSEMENT
200 '
210 K=1 ' On place K a 1 pour que la boucle s'execute au moins une fois
220 WHILE K
230 K=0
240 FOR I=1 TO 19
250 IF A(I)>A(I+1) THEN SWAP A(I),A(I+1):K=1
260 NEXT I
270 WEND
290 '
300 ' ** IMPRESSION DES DONNEES CLASSEES
310 LPRINT " TABLEAU CLASSE "
320 LPRINT
330 FOR I=1 TO 20
340 LPRINT " I = ";I," Valeur = ";A(I)
350 NEXT I
360 END
```

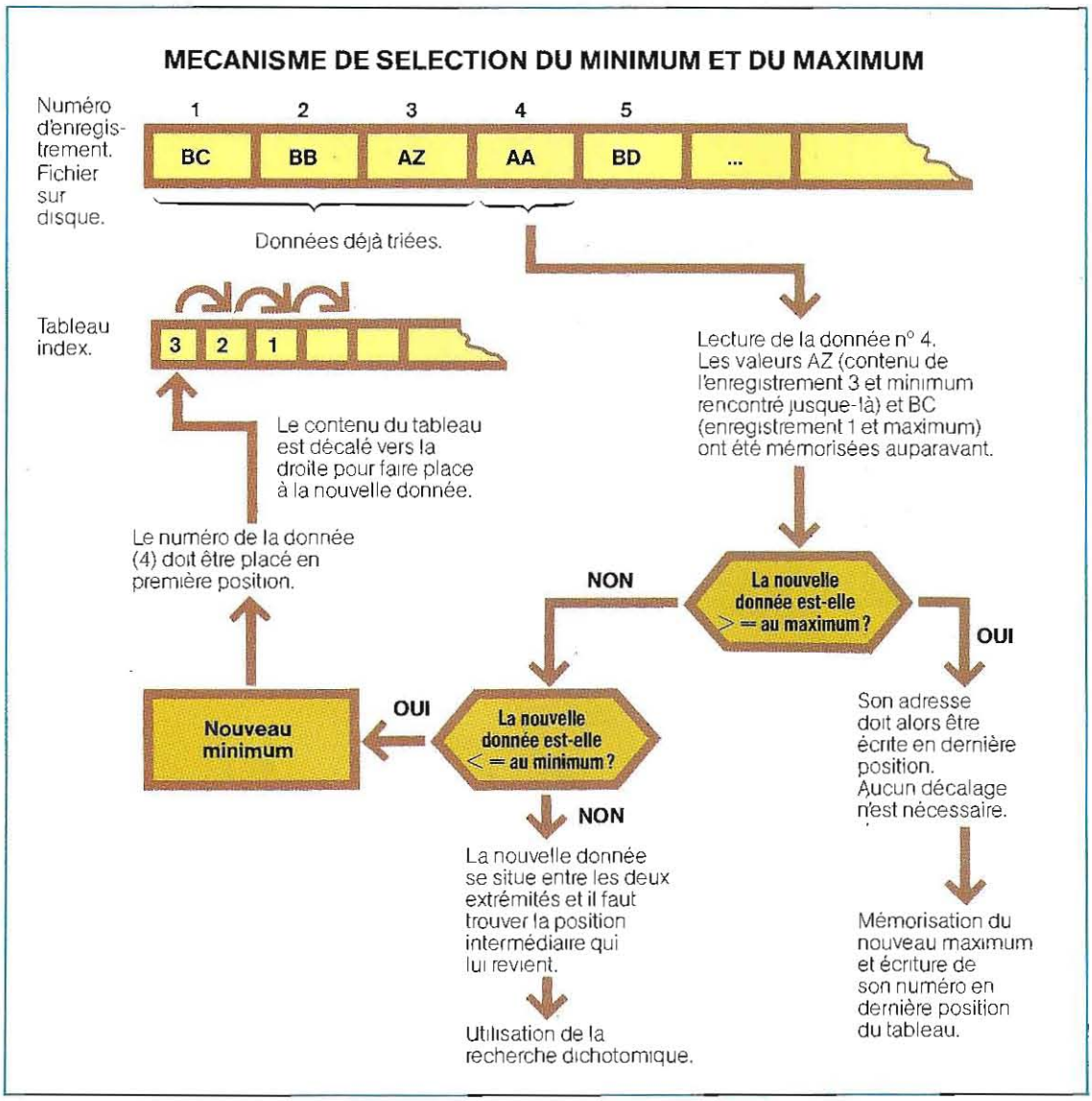
l'une des valeurs à échanger dans une troisième variable, dite mémoire de travail. L'instruction SWAP permet d'obtenir le même résultat par simple interversion du nom des deux variables. La zone mémoire de départ prend ainsi le nom de la zone d'arrivée et inversement. Le but initial – faire passer les données d'une variable à une autre – est donc également atteint, mais inverser les adresses de deux variables est beaucoup plus rapide que d'en réaffecter trois.

On peut adapter ce programme à tout tableau, quel que soit le nombre de ses éléments – dans les limites de l'espace mémoire disponible. Il suffit pour cela de remplacer les nombres 20 et 19 (lignes 30, 150, 240 et 330) par les valeurs appropriées ou un paramètre défini au préalable. Enfin, le programme tel qu'il est rédigé ne s'applique qu'à des valeurs numériques mais son adaptation à des chaînes de caractères est aisément réalisable en définissant comme telles (symbole $) la variable indicée A(N) et la mémoire de travail S. Rappelons en effet que les symboles > et < employés sur des chaînes de caractères opèrent selon l'ordre alphabétique.

Tri d'un fichier sur disque

Dans le programme qui va être présenté, on utilise comme mémoire de travail un tableau qui doit être dimensionné de façon à contenir autant d'éléments que le fichier comporte d'articles. En effet, chaque fois qu'un enregistrement sera lu, son numéro sera transféré dans ce tableau à la place qui lui revient en fonction du contenu alphabétique de cet





enregistrement. Donc, si, par exemple, le contenu de l'enregistrement n° 15 est inférieur (selon l'ordre alphabétique) à ceux du n° 1 et du n° 2, la valeur 15 sera positionnée dans le tableau de tri avant les valeurs 1 et 2.

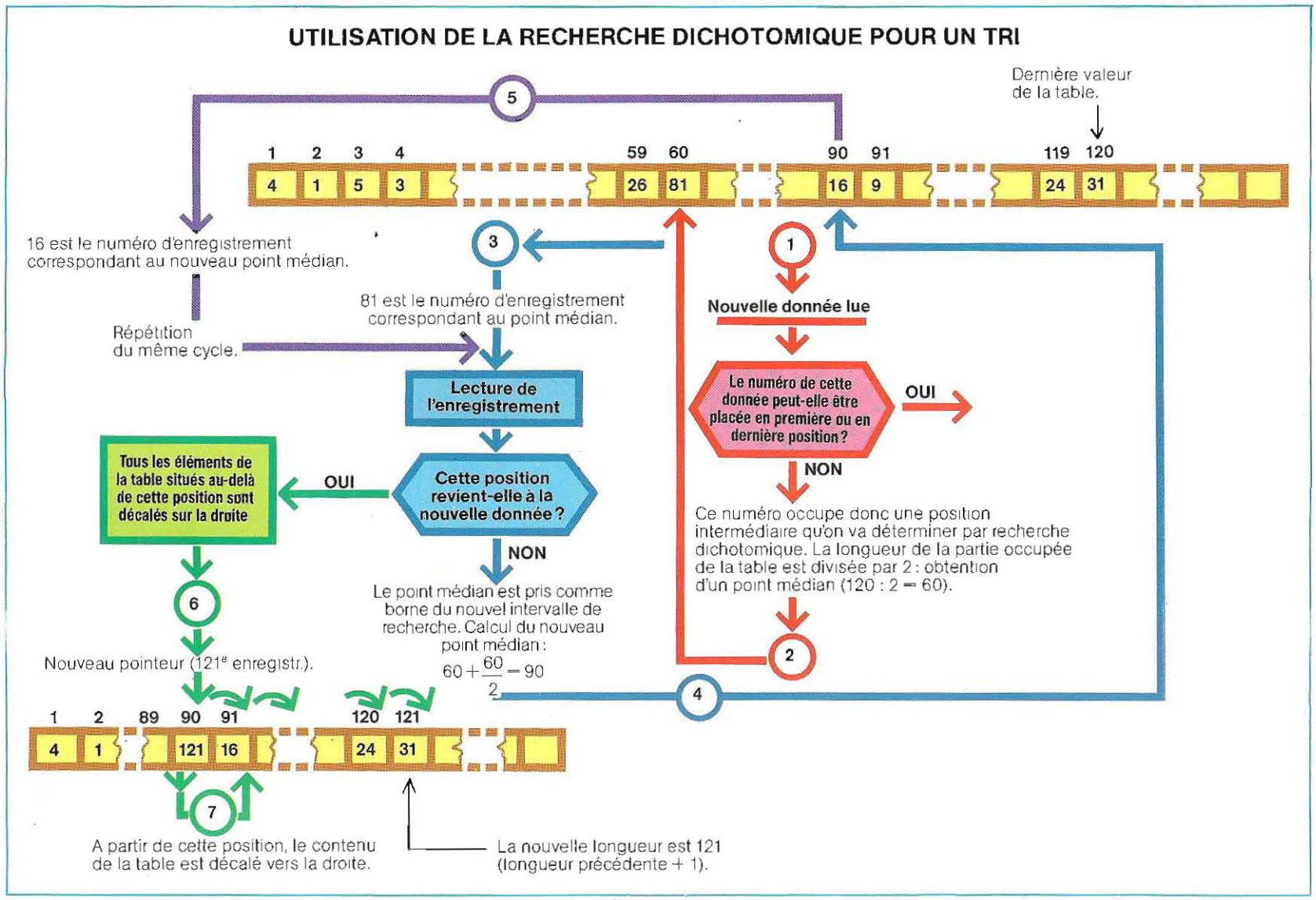
On peut ensuite utiliser les données de ce tableau comme des pointeurs. Selon qu'on remontera ou qu'on descendra le tableau, on aura accès aux enregistrements classés en ordre alphabétique croissant ou décroissant. On dit que ce tableau constitue un index. La logique du traitement est illustrée ci-dessous. Le remplissage du tableau de tri se fait de la façon suivante.

Supposons que plusieurs enregistrements aient déjà été traités, les trois premiers par exemple. Le tableau contient dorénavant trois valeurs, qui sont les numéros de ces enregistrements classés dans l'ordre croissant de leur contenu. Le premier élément du tableau contient donc le numéro de la plus petite donnée déjà triée, le second élément celui de la donnée immédiatement supérieure et ainsi de suite.

On extrait ensuite la donnée suivante (enregistrement numéro 4). On peut, bien sûr, déterminer sa place dans le tableau en relisant tous les enregistrements déjà traités et en comparant leur contenu à cette nouvelle donnée.

Cependant, il existe deux méthodes dont l'utilisation conjointe accélère la recherche. La

UTILISATION DE LA RECHERCHE DICHOTOMIQUE POUR UN TRI



première consiste à mémoriser la plus grande et la plus petite des données déjà triées et à comparer systématiquement chaque nouvelle donnée à ces deux extrêmes (voir schéma page 681).
Le numéro d'un enregistrement inférieur ou égal au minimum sera placé en première position et celui d'un enregistrement supérieur ou égal au maximum en dernière position.
Si ces deux comparaisons n'ont rien donné, c'est que l'enregistrement en question correspond à une position intermédiaire.
On emploiera alors la seconde méthode dont nous avons parlé : la recherche dichotomique (divisions successives par deux). Après avoir extrait la valeur médiane du tableau, on lit l'enregistrement du fichier correspondant à ce numéro (BB) et on le compare à la nouvelle donnée. On décide ensuite de continuer la recherche dans la partie droite ou gauche du tableau, selon le résultat de la comparaison.
Un schéma général du programme est représenté page 682.
Des possibilités complémentaires sont généralement offertes dans les logiciels de tri proposés dans le commerce. La description du logiciel utilitaire OLISORT* élaboré par Olivetti donnera une idée des performances de tels outils.

Le programme utilitaire OLISORT

Conçu pour l'ordinateur individuel Olivetti M20, Olisort est un progiciel de tri/fusion performant et d'une grande souplesse.
Une vaste gamme de fonctions le rend apte à la plupart des applications de gestion. Voici quelques-unes des possibilités d'Olisort :

- tri d'un fichier ;
- fusion de deux fichiers ;
- sélection d'un enregistrement dans un fichier ;
- concaténation d'un fichier à la fin d'un autre ;
- réalisation simultanée d'un tri et d'une sélection.

Les commandes d'Olisort sont constituées de paramètres qui sont transférés dans une chaîne de caractères. Cette chaîne peut être communiquée au logiciel par le biais d'instructions Basic ou par une séquence de

* OLISORT est une marque déposée.

Le système de calcul MOT 53385 sur IBM.



Grazia Neri/Photo Media

Communiquer à l'aide d'un micro-ordinateur

Remarquable outil de communication, le micro-ordinateur peut servir à pallier les handicaps des personnes qui souffrent de troubles moteurs, ou de troubles de la compréhension ou de la phonation, comme le montrent certaines expériences menées tant en Europe qu'aux Etats-Unis et au Japon. Ces programmes sont l'aboutissement de travaux poursuivis en collaboration entre les spécialistes de la rééducation fonctionnelle et les chercheurs en électronique.

Aide à l'expression ou à la compréhension, la machine est dans tous les cas un précieux moyen pour permettre aux handicapés de communiquer avec leur entourage et d'évoluer dans leur environnement.

Pour un aveugle, ce sera par exemple l'intérêt de l'ascenseur (ISI) ou du feu rouge (SILEC) parlants, des jeux électroniques parlants (ISI), de la calculatrice dotée de la parole (ISI, Kurzweil).

Il s'agit là de ***synthèse acoustique,*** *tandis que, grâce à la* ***synthèse phonétique,*** *on réalise des machines à écrire parlantes qui disent le mot qui vient d'être tapé (IBM-France), autorisant les aveugles à les utiliser normalement pour tous les travaux spécifiques du traitement de texte.*

Dans le cas des infirmes moteurs cérébraux, la synthèse joue le rôle d'une ***prothèse vocale.*** *C'est un moyen très efficace pour l'apprentissage de la lecture et de l'écriture.*

Selon les pays et selon le matériel utilisé, le système peut fonctionner différemment mais il comporte toujours une base fondamentale assez similaire et nous nous bornerons à décire quelques modèles ou programmes français et italiens.

En ce qui concerne la France, citons le système à clavier complet sur lequel on forme les mots (Handivoice-Votrax, Sahara du CNET qui fonctionne avec un langage Bliss) à l'aide de touches correspondant à des lettres, des mots ou des symboles, et le système à touches de sélection (en nombre très réduit : deux en général ou même une seule), qui offre une manipulation très aisée, particulièrement adapté aux besoins des grands handicapés moteurs (Vidéocom du professeur Eschstruth, et fonctionnant avec le synthétiseur Icolog développé par le LIMSI-CNRS).

Le Vidéocom permet la sortie d'un texte par affichage sur un écran vidéo ou sur imprimante. Les entrées peuvent se faire de deux manières différentes. Pour l'utilisateur dont la motricité n'est pas trop gravement atteinte, l'entrée se fait par clavier, sur lequel le texte est composé assez rapidement. Mais si l'utilisateur souffre d'une motricité très perturbée ou très limitée, la commande par une ou deux touches permet de procéder à une sélection progressive parmi les soixante lettres ou symboles disponibles, en partant du choix d'une des douze colonnes possibles. Le système, bien que moins rapide, est une première amélioration des conditions de la communication des grands handicapés. Grâce au synthétiseur Icolog qui accepte directement le texte en français, l'utilisation peut avoir recours à une orthographe simplifiée, ce qui lui fera gagner du temps. Il peut écrire, par exemple : « je veu alé ché moi ».

Le système italien Logos 4, développé par l'université de Gênes, est assez proche dans son principe.

Des études statistiques faites sur le langage, et particulièrement sur les textes destinés aux enfants de l'école primaire ou écrits par eux, ont permis d'établir des listes de mots et de groupes de lettres rassemblées selon leur fréquence dans le langage. A partir d'un premier choix fait par l'utilisateur, ce dernier peut visualiser à l'écran les lettres qui pourraient venir ensuite en ordre de probabilité décroissante. Il dispose alors, pour choisir le caractère adéquat, soit de deux touches – une touche de rejet et une touche d'acceptation –, soit d'une seule touche de sélection (dans ce cas, les lettres proposées par la machine se succèdent à un certain rythme, modifiable). La séquence de lettres affichée varie sans cesse puisque l'étude statistique du vocabulaire permet de déterminer quelle lettre a la plus grande probabilité de venir compléter un mot. On évite ainsi de parcourir inutilement l'alphabet et on parvient à communiquer assez rapidement dès qu'on a acquis une certaine habitude.

Deux autres touches sont utilisées : la première pour inverser l'ordre des caractères et la seconde pour passer du balayage automatique au balayage manuel. Les lettres sont alors affichées en différentes couleurs. Leur taille

diminue au fur et à mesure qu'elles se rapprochent (graphisme « en entonnoir »). L'utilisateur peut ainsi préparer son choix, en s'aidant de ce qu'il voit sur l'écran. En outre, il est possible d'ajouter à cette représentation une amplification sonore et de ralentir la vitesse de balayage pour laisser aux infirmes moteurs les plus handicapés le temps de transmettre leur choix.

La différenciation des lettres par une couleur présente un intérêt pour les utilisateurs ayant également des problèmes de vision.

Les lettres déjà choisies pour un même mot sont visualisées en couleur, à gauche de l'écran. Il est ainsi plus facile de terminer le mot et éventuellement de le corriger. Une fois composé, le mot apparaît à la suite des lignes précédentes du texte, qui sont affichées au bas de l'écran en plus petits caractères.

En plus des vingt-six lettres de l'alphabet, plusieurs symboles ont été ajoutés pour désigner les fonctions d'espacement, d'alinéa, de retour en arrière, de correction, d'impression du texte (qui vient d'être écrit et qui n'est pas mémorisé par la machine) et de retour aux fonctions initiales du programme.

Un programme comme le Logos 4 (mis au point pour un matériel IBM) ne peut être appliqué que si la configuration de l'ordinateur utilisé comporte au moins les éléments suivants :

Une unité centrale disposant de 128 K de RAM et d'un connecteur d'entrée analogique (manettes de jeux) ;

un lecteur de disquettes 180 K ;

un écran grahique ;

un clavier simplifié à larges touches pour handicapés ;

un clavier ;

une imprimante.

L'écran utilisé peut être celui d'un poste de télévision connecté au système par un modulateur.

On pourra augmenter la taille de la configuration minimale, en lui ajoutant une seconde unité de disque de 180 K ou bien en remplaçant la première par une unité de 360 K.

Par ailleurs, il faut prévoir des claviers spéciaux différents (très grands ou pneumatiques, par exemple) afin de les adapter à chaque cas particulier.

Chaque utilisateur a la possibilité d'obtenir une disquette programmée en fonction de ses propres capacités motrices. Il existe, en effet, deux masques de saisie qui permettent d'opter pour telle ou telle caractéristique d'affichage ou d'impression.

L'un de ces marques offre le choix entre différentes modalités de contrôle du mouvement et de présentation des caractères.

Les lettres peuvent se présenter de deux façons : en balayage automatique, elles se déplacent automatiquement de droite à gauche à une vitesse programmable (un déplacement par seconde, et jusqu'à un toutes les cinq secondes), alors qu'en balayage manuel, l'utilisateur doit exprimer son choix pour chaque lettre – acceptation ou refus – en appuyant sur la touche « oui » ou sur la touche « non » : il doit donc être en mesure d'actionner deux touches.

En revanche, avec le balayage automatique, la sélection des lettres ne demande à l'utilisateur qu'un effort minimal (un seul mouvement suffit généralement).

Pourtant, le balayage manuel permet d'aller plus vite et de commander directement la machine. Il est possible de faciliter la perception des informations visuelles en modifiant la couleur du fond (seize possibilités), des lettres de l'« entonnoir » (quatre couleurs), de la lettre en cours de sélection (quatre couleurs) et des lettres déjà acquises (quatre couleurs). On peut ainsi adapter la présentation aux facultés visuelles de l'utilisateur.

En outre, on peut faire varier la combinaison de couleurs pour augmenter ou diminuer la difficulté, en fonction des capacités de perception et d'apprentissage de l'utilisateur.

Enfin, on peut faire disparaître les lettres de l'entonnoir ou les lettres déjà choisies en les écrivant de la même couleur que le fond.

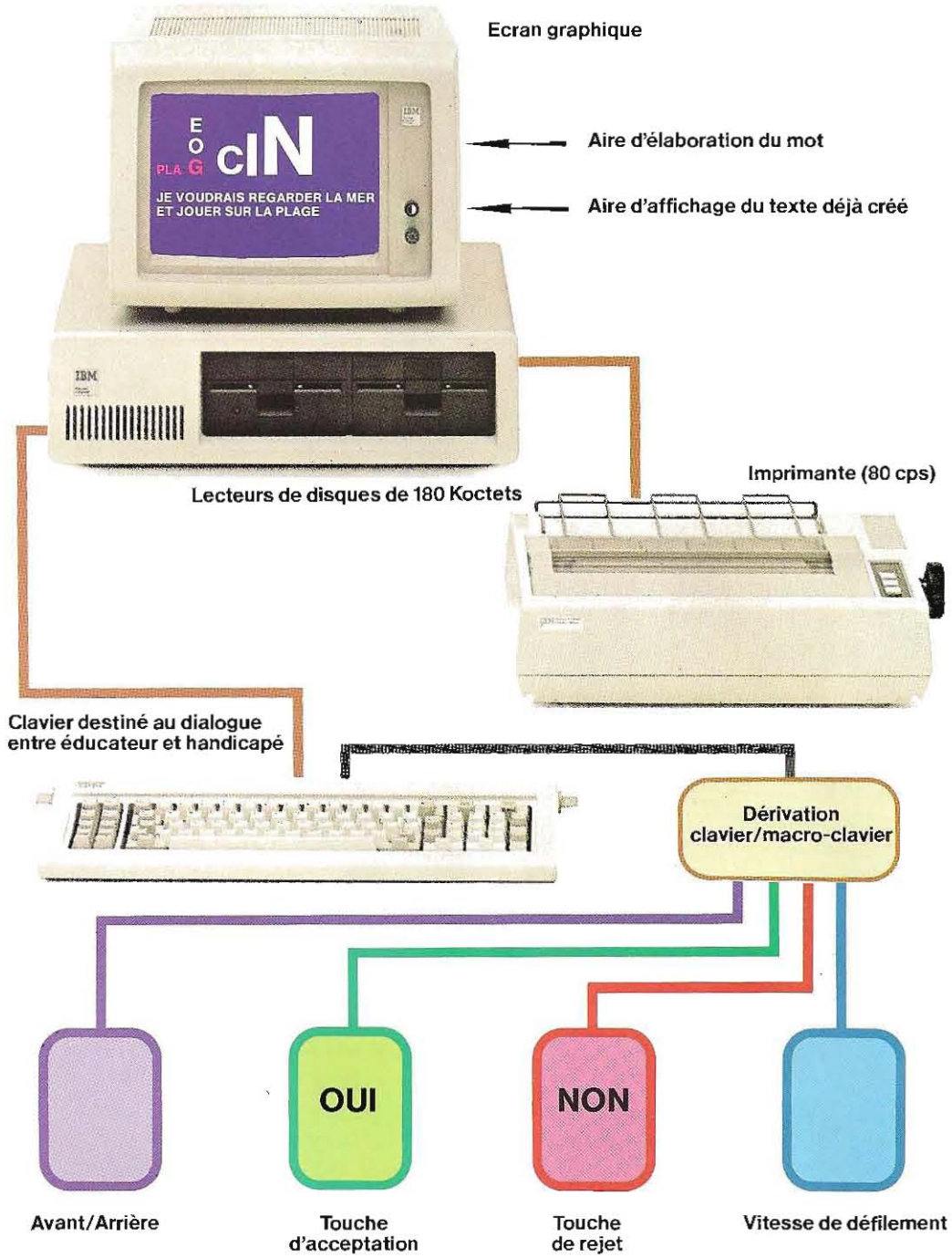
Quant à l'impression, on peut programmer sur l'imprimante la taille des caractères et des interlignes, ainsi que l'intensité du trait.

Au début de l'exécution du programme, le choix est donné à l'utilisateur entre sept fonctions qui permettent une gestion autonome du programme lui-même.

On peut commencer un nouveau texte, le mémoriser éventuellement, reprendre un travail interrompu, rappeler un texte sauvegardé sur disque ou imprimer la totalité d'un texte.

La première fonction de communication est la fonction « d'appel ». Elle commande un signal acoustique intermittent qui avertit l'édu-

CONFIGURATION DU MATERIEL NECESSAIRE AUX PROGRAMMES D'AIDE PAR L'ORDINATEUR A LA COMMUNICATION DES HANDICAPES



Macro-clavier ou autre périphérique d'entrée adapté aux aptitudes de l'handicapé.



IBM

L'élaboration d'un message à l'aide du programme Logos 4. Les espoirs fondés sur le micro-ordinateur pour la rééducation des handicapés se concrétisent aujourd'hui. Les expériences menées tant en Europe qu'aux Etats-Unis ou au Japon, révèlent d'immenses possibilités pour abattre le mur d'incommunicabilité qui isole les handicapés.

cateur quand le sujet demande à être aidé. De plus, éducateur et handicapé peuvent dialoguer, le premier à l'aide de son clavier et le second au moyen de ses touches spéciales. L'échange de questions et de réponses affichées à l'écran aide beaucoup l'élève à se familiariser avec le programme.

Il est clair, par ailleurs, que le travail sur l'ordinateur apprend progressivement à l'handicapé à se réinsérer dans son milieu de vie, par la communication avec les personnes mais aussi par sa possibilité de commander à son environnement.

Des systèmes comme Mozart du LIMSI-CNRS, Seraphine du CNET, pour la France ; ou Logica en Angleterre ; Nec au Japon ; ou encore Verbex aux Etats-Unis, permettent à ces handicapés (tétraplégiques par exemple), de commander l'ouverture des portes, le déplacement d'un lit, la marche d'une voiture, mais aussi les mouvements de leurs propres prothèses, sur ordre oral. L'individu est alors en situation de reconquérir une autonomie proche de la normale et peut envisager d'apprendre un métier qu'il pourra pratiquer. Il ne faut pas s'étonner que les métiers de l'informatique figurent en bonne place dans cette perspective.

Les systèmes dont nous venons de parler sont fondés sur le principe de la ***reconnaissance de la parole*** *par la machine, et certaines expériences sont appliquées au service des malentendants. La machine reconnaît la voix de l'interlocuteur du malentendant et l'affiche sur son écran, où le malentendant la lit, ce qui lui permet un dialogue normal.*

De même, un tel appareil, relié à un téléviseur, permettrait le sous-titrage simultané à l'écran de ce que disent speaker ou personnages du film (étude LIMSI-CNRS en cours).

La machine sert donc d'intermédiaire dans la transmission des messages par affichage ou par impression, mais elle peut aussi fournir de véritables exercices de rééducation vocale, remplaçant le professeur traditionnel.

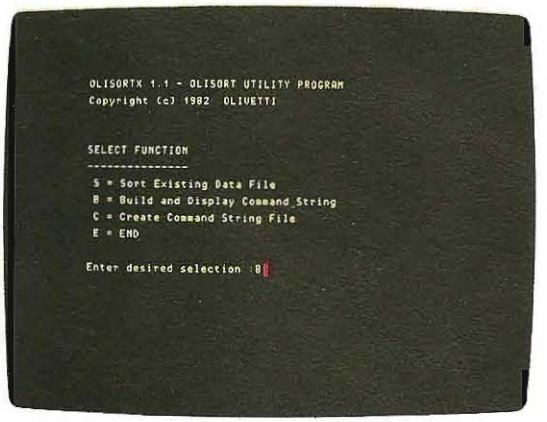
Elle servira pour ceux qui entendent mais présentent des troubles particuliers relevant de l'orthophonie (retard de la parole chez l'enfant, dyslexie, dysorthographie, ou encore laryngectomie, entre autres). Le micro-ordinateur sert alors d'aide aux enfants sans langage écrit ou sans langage oral. En favorisant la communication autrement que par la parole avec un adulte, il contribuera à débloquer certains obstacles (c'est le cas pour les bègues) et constituera une rééducation efficace.

Dans ce domaine, des expériences très positives ont été tentées en France, par exemple avec Sparte, objet de type automate qui a toujours du succès auprès des enfants.

Il est donc tout à fait regrettable que les grands producteurs en informatique (matériel et logiciel) ne manifestent pas un très grand intérêt dans ce domaine.

LE PROGRAMME OLISORT

Les fonctions du logiciel Olisort sont accessibles soit à partir d'instructions regroupées dans un programme en Basic, soit directement par appel de l'utilitaire Olisortx. Dans ce dernier cas apparaît à l'écran le texte reproduit sur cette page.




```
OLISORTX 1.1 - OLISORT UTILITY PROGRAM
Copyright (c) 1982  OLIVETTI


SELECT FUNCTION
---------------
 S = Sort Existing Data File
 B = Build and Display Command String
 C = Create Command String File
 E = END

Enter desired selection : B
```

Les deux lignes affichées en haut de l'écran informent l'utilisateur du chargement d'Olisort et, plus précisément, de l'utilitaire Olisortx.
L'exécution commence par l'affichage du menu, c'est-à-dire des fonctions disponibles sur cet utilitaire et sélectionnables par la frappe d'une touche

Tri d'un fichier préexistant : touche S.

Définition d'une chaîne constituée de commandes : touche B.

Création d'un fichier regroupant des chaînes commandes : touche C.

Sortie de l'Olisortx : touche E.

A la suite de cette liste, l'opérateur doit frapper la touche correspondant à la fonction choisie. Sur la photo, c'est la touche B qui a été enfoncée. On va donc pouvoir définir une chaîne composée des paramètres et des valeurs conditionnant le tri de données

codes plus succints affectés à des commandes particulières dans un fichier paramètres. Les utilitaires permettent de définir en mode interactif aussi bien le fichier paramètres que les chaînes de commande.
Olisort peut être appelé depuis un programme en Basic ou par l'intermédiaire de l'utilitaire Olisortx en mode direct. On pourra donc, soit créer un programme qui réalise le tri d'un fichier – voire de plusieurs par adjonction de sous-programmes spécifiques –, soit procéder « manuellement » au classement d'un fichier déterminé.
Voici les différentes modalités de tri-sélection autorisées par l'Olisort :

- spécification des critères de tri dans la limite de 10 par enregistrement ;
- sélection ou rejet d'un enregistrement grâce aux clés Select/Exclude (4 au maximum) ;
- deux types d'entrée pour chaque clé : le caractère * qui valide tous les caractères suivants en temps que clé de sélection ou rejet, et le caractère ? qui ne prend en compte que le premier caractère suivant ;
- indication pour chaque clé Select/Exclude de l'opérateur de comparaison (inférieur, égal ou supérieur) ;
- combinaison des différentes clés au moyen des opérateurs logiques OR et AND ;
- possibilité, lors de la comparaison d'une donnée à une clé, d'attribuer la même valeur aux minuscules et aux majuscules ;
- le tri ou la fusion ne débutent pas obligatoirement en début de fichier (on peut sauter jusqu'à 32.767 enregistrements) ;
- disque de travail et disque de sortie peuvent être intervertis sans risque, le programme le signalant avant toute possibilité d'erreur ;
- un traitement des erreurs est programmable car le logiciel mémorise systématiquement tout code d'erreur ;
- l'appel d'Olisort à partir d'un programme Basic est très simple et ne réclame aucune connaissance particulière.

Ajoutons toutefois que ce logiciel est spécifiquement conçu pour l'interpréteur Basic du Olivetti M20 et qu'il n'opère que sur des enregistrements de longueur donnée.

Les prestations. Dans la mesure où l'on ne peut pas effectuer un traitement multivolume, la taille maximale des fichiers auxquels on peut appliquer Olisort est fonction de la capacité du disque. En effet, en cours de tri, le fichier du disque de travail peut atteindre 1,7 fois la dimension du fichier traité. C'est le facteur limitatif. La longueur des enregistrements ne doit pas excéder 256 caractères. Une commande du système PCOS permet toutefois de modifier ce maximum si l'on doit traiter des enregistrements d'une longueur supérieure. Dans les exemples qui suivent, on a systématiquement adopté cette longueur qui est la valeur prise par défaut.
Les clés de tri et les clés Select/Exclude peuvent appartenir aux types de données suivantes :

- chaînes de caractères ;
- valeurs hexadécimales ;
- nombres entiers ;
- nombres en simple précision ;
- nombres en double précision.

Nous avons vu qu'avec une clé alphanumérique, il était possible de supprimer la différenciation entre minuscules et majuscules. Notons que cela n'est pas valable avec une clé de type hexadécimal. La comparaison aura alors lieu sur les codes ASCII des caractères qui sont différents pour les majuscules et les minuscules.
Le tri se fait en fonction des critères définis et en ordre croissant ou décroissant selon la valeur affectée au paramètre correspondant (A pour « ascendant »/D pour « descendant »).
Voici l'exemple du classement d'un fichier clients dans l'ordre croissant des départements (clé majeure) et dans l'ordre décroissant des soldes à régler (clé mineure). L'édition après traitement donne le résultat suivant :

Liste des soldes à régler

Département	Client	Solde (en F)
Ain	M & M Sarl	2596,00
	Boquet Frères	1976,55
	ABC (SA)	568,88
Aisne	NDT (Ets)	8870,77
	Zig et Puce	8345,98
	Pallas (SA)	5689,00
	SOPAC (Sté Nlle)	67,43

Le fichier trié peut être mémorisé sur un autre disque ou, éventuellement, sur le même à la place du fichier initial. Cependant, on risque alors de perdre une partie du fichier, dans le cas d'une défaillance matérielle au cours de l'opération. Il est donc conseillé, par mesure de sécurité, de réaliser cette opération sur une copie de la disquette et de préserver l'original.
Olisort permet également la sélection de tous les enregistrements du fichier qui répondent à certains critères. Cette sélection peut éventuellement être effectuée à l'occasion d'un tri. Nous pourrions, dans notre exemple du fichier clients ordonné, reprendre le même travail, mais en ne retenant que les clients dont le solde est supérieur à 1.000 F. La partie correspondante du fichier obtenu serait alors :

Liste des soldes des clients après sélection

Département	Client	Solde (en F)
Ain	M & M Sarl	2596,00
	Boquet Frères	1976,55
Aisne	NDT (Ets)	8870,77
	Zig et Puce	8345,98
	Pallas (SA)	5689,00

Le nombre maximal de critères de sélection/exclusion employés simultanément est de 4. Ces critères sont totalement indépendants de ceux de tri qui peuvent être effectués en même temps. La sélection peut donc porter sur des rubriques différentes de celles conditionnant le classement. Si seules sont définies des clés de sélection/exclusion, le fichier obtenu en sortie n'aura fait l'objet d'aucun classement et suivra donc l'ordre du fichier d'origine. Quand une chaîne de caractères est employée comme clé de sélection, le programme compare d'abord sa longueur à celle de la rubrique correspondante avant de confronter les contenus des deux chaînes.
Olisort procède de la façon suivante :

1 / il compare les longueurs de la clé et de la zone afin de déterminer, éventuellement, laquelle des deux est la plus longue ;
2 / il tronque alors celle-ci pour obtenir deux chaînes de taille égale ;
3 / enfin, ces deux chaînes sont comparées, caractère par caractère.

Ainsi, avec des enregistrements contenant une rubrique de la forme suivante :

Tuyau de plomb 12 mm × 3 m

On pourra utiliser « Tuyau de plomb » comme clé de sélection.
On peut introduire dans une clé de sélection un ou plusieurs symboles à la place de caractères. Ces symboles sont les trois **signes de comparaison** :

< indique que le caractère situé à la même place dans la rubrique ne doit pas être testé, mais considéré comme supérieur à ce à quoi il aurait dû être comparé ;
> provoque l'effet inverse ;
= signifie que quel que soit ce caractère dans la rubrique, il doit être accepté (considéré comme égal à celui de la clé).

L'emploi de ces caractères est lié à la possibilité de sélectionner (ou d'exclure) un enregistrement, selon que la rubrique testée est inférieure, égale ou supérieure (LEG) à la clé introduite. Le fait d'utiliser ces symboles permet donc d'ignorer les caractères correspondants de la rubrique, en les rendant automatiquement inférieurs, égaux ou supérieurs à la clé, selon qu'on sélectionne suivant l'une ou l'autre de ces caractéristiques. Supposons, par exemple, que des composants soient répertoriés dans un fichier par un code général constitué en fait de codes différents :

Rubrique « code » du fichier des composants

Code de montage	Code du composant	Classe du composant
94	543678	B
67	553478	A
80	887690	A
88	113425	D
80	984400	C
80	000789	A
56	722990	B

On veut effectuer la sélection des composants de classe A et dont le code de montage est 80. Pour cela, il suffit d'utiliser la clé de sélection suivante :

80 = = = = = = A

Deux composants seront alors sélectionnés dans la partie du fichier reproduite :

80-887690-A
80-000789-A

Quand on utilise plusieurs clés de sélection/exclusion, on doit les relier par un opérateur logique AND ou OR.
Avec deux clés reliées par AND, un enregistrement doit remplir simultanément les deux conditions pour être sélectionné, tandis qu'avec OR, il suffit que l'une des deux soit satisfaite. Ainsi, en définissant pour le fichier de composants la combinaison suivante :

80 ═══ A OR 80 ═══ C

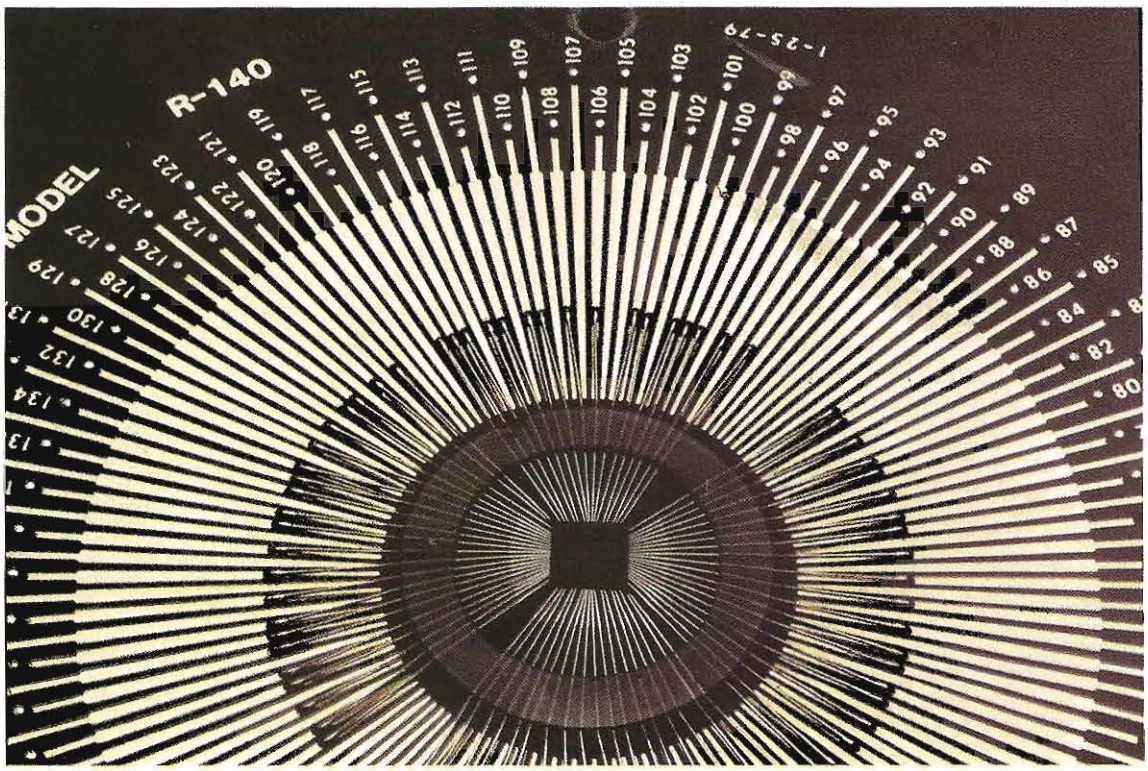
nous obtiendrons la sélection de trois composants :

80-887690-A
80-984400-C
80-000789-A

Notons que l'opérateur AND ne peut relier que des clés portant sur des rubriques différentes. Il serait en effet impossible de sélectionner un composant qui appartiendrait à la fois à classe A et à classe C.
Olisort permet également de regrouper deux fichiers en un seul. Deux types de combinaisons sont réalisables : la fusion (merge) qui opère sur deux fichiers classés et renvoie le fichier résultant ordonné, et la concaténation (append) simple qui n'effectue aucun tri.
La fusion nécessite donc une spécification des clés pour le tri.
Par contre, la concaténation de deux fichiers ne demande aucune précision particulière.
Olisort peut reconnaître jusqu'à cinq fichiers simultanément. Cependant, trois d'entre eux sont réservés au fonctionnement :

— INPUT 1 : fichier général du progiciel ;
— INPUT 2 : fichier utilisé lors d'une fusion ou d'une concaténation ;
— OUTPUT : fichier résultant de toute opération de tri, de fusion, de sélection ou de concaténation. Si on donne à ce fichier de sortie le nom du fichier original, il sera écrit sur le disque à la place de celui-ci.

Un testeur de circuits intégrés.



© Dunbar/Marika

Les modalités d'exécution. Nous avons vu que la mise en œuvre du programme Olisort peut se faire :

- en fixant la valeur des paramètres de commande dans un programme, pour une exécution différée ;
- en passant par l'utilitaire Olisortx pour sélectionner en temps réel ces mêmes paramètres.

Les paramètres sont d'un format identique quelle que soit la procédure.
Ils peuvent être transmis à Olisort de deux façons :

- à l'aide d'un fichier paramètres sauvegardé sur disque (ce qui est avantageux dans le cas d'opérations d'usage fréquent). Un utilitaire Olisortp facilite la création de ce fichier ;
- par l'intermédiaire d'une suite de commandes regroupées au sein d'une chaîne alphanumérique. Cette chaîne s'appelle OLISORT.CMD$. et peut être créée à partir du Basic (programme) ou de l'Olisortx (direct). Dans ce second cas, elle sera mémorisable sur disque.

Notons que le recours à un fichier paramètres n'exclut pas l'insertion de chaînes de commande dans un programme. Ces chaînes seront cependant très brèves (un paramètre et une référence au fichier commandes).
L'un des paramètres permet de préciser le mode suivant lequel doit fonctionner le progiciel.
Le choix porte alors sur quatre modes.

Le **Mode 0** permet d'ordonner un fichier, de réaliser une sélection d'enregistrements ou d'effectuer conjointement ces deux opérations. Toutes les commandes nécessaires doivent avoir été préalablement mémorisées dans un fichier paramètres.
Supposons qu'on dispose d'un fichier permanent de stock classé selon les codes articles et de deux fichiers temporaires où sont répertoriés de nouveaux produits. On pourra procéder au tri de ces deux derniers fichiers de la façon suivante :

1 / création d'un fichier paramètres à l'aide de l'utilitaire Olisortp ;

2 / écriture d'un court programme Basic qui appellera deux fois Olisort afin d'exécuter le tri des deux fichiers selon le même principe (même fichier paramètres). Ce programme précisera le mode d'exécution d'Olisort (Mode 0) et la liste des commandes qui ne peuvent être intégrées au fichier paramètres.

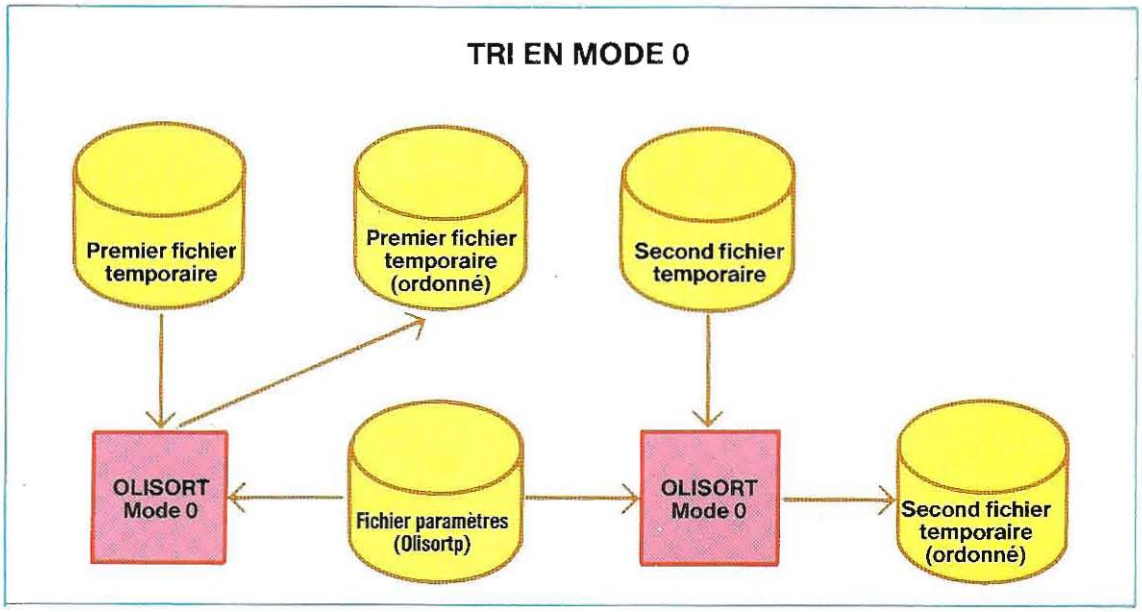
Le **Mode 1** gère la fusion et la concaténation des fichiers. Ce mode se réfère lui aussi à un fichier paramètres. Si les deux fichiers sont ordonnés et que le fichier paramètres contient une clé de tri, Olisort procédera alors à une fusion. Dans le cas contraire, les deux fichiers seront simplement concaténés.
Reprenons les trois fichiers cités plus haut, on peut désormais effectuer la mise à jour du fichier permanent de la façon suivante :

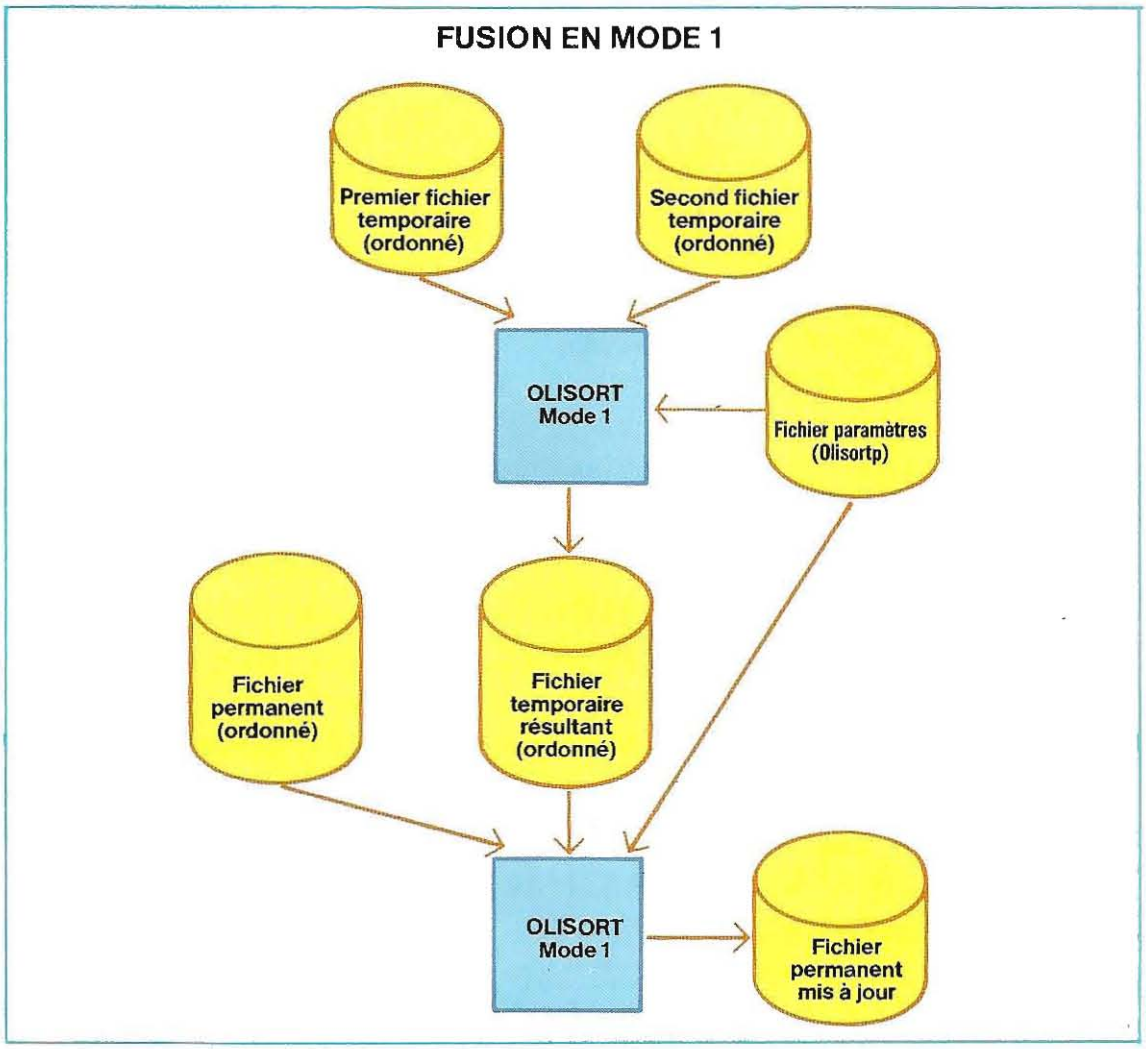
1 / on réutilisera le fichier paramètres créé pour le tri des fichiers temporaires mais on modifiera le programme Basic de façon à obtenir une exécution du mode 1. Les fichiers INPUT 1 et INPUT 2 seront les deux fichiers temporaires ordonnés ;
2 / l'exécution du programme Basic aboutira donc à la fusion de ces deux fichiers en un unique fichier temporaire ordonné lui aussi ;
3 / on reprendra alors le programme Basic précédent et on l'appliquera au nouveau fichier temporaire et au fichier permanent ;
4 / ce programme aura pour effet la fusion de ces deux fichiers réalisant ainsi une mise à jour du fichier permanent.

Le **Mode 2** sert à exécuter les mêmes opérations que le Mode 0 mais sans recours à un fichier paramètres. Les commandes devront donc être définies une à une et in extenso dans le programme Basic.

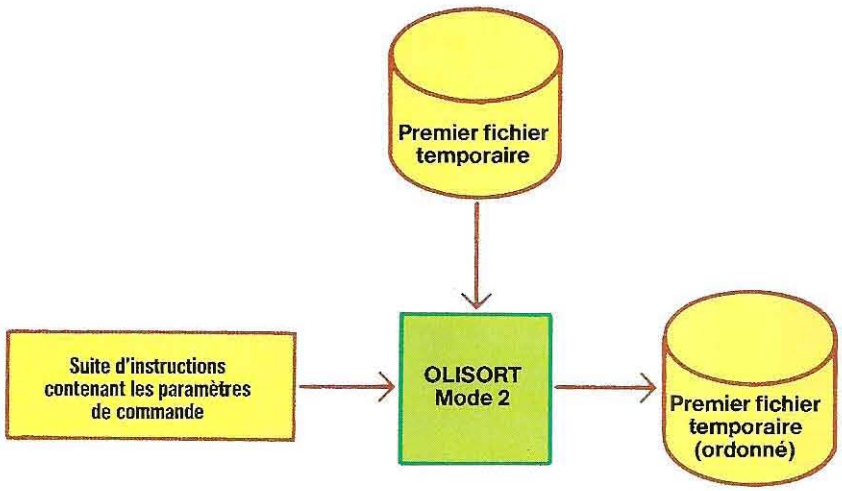
Le **Mode 3** est identique au Mode 2. Cependant, un fichier paramètres est créé à partir des commandes successives insérées dans le programme Basic et pourra donc être réutilisé par la suite.
Nous aurions ainsi pu réaliser le classement de nos deux fichiers temporaires de la façon suivante :

1 / en écrivant un programme Basic appelant Olisort pour le tri en Mode 3 du premier fichier temporaire ;

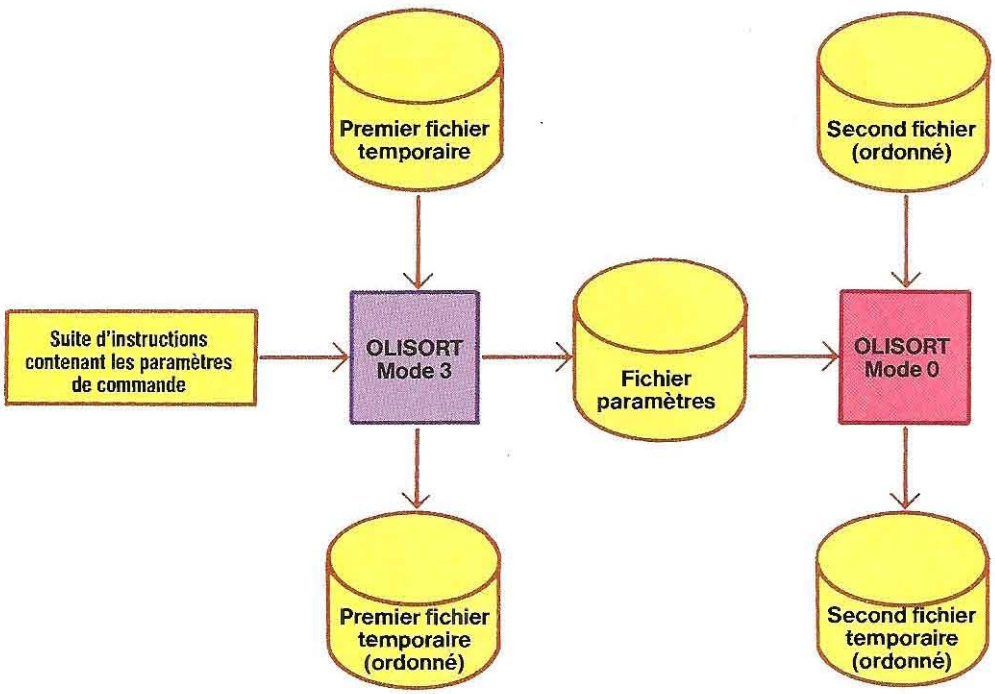

TRI EN MODE 0
Premier fichier temporaire
Premier fichier temporaire (ordonné)
Second fichier temporaire
OLISORT Mode 0
Fichier paramètres (Olisortp)
OLISORT Mode 0
Second fichier temporaire (ordonné)


FUSION EN MODE 1
Premier fichier temporaire (ordonné)
Second fichier temporaire (ordonné)
OLISORT Mode 1
Fichier paramètres (Olisortp)
Fichier permanent (ordonné)
Fichier temporaire résultant (ordonné)
OLISORT Mode 1
Fichier permanent mis à jour

TRI EN MODE 2



TRI EN MODE 3



2 / l'exécution de ce programme aurait alors conduit non seulement au classement du fichier temporaire mais également à la sauvegarde des paramètres de commande dans un fichier paramètres ;

3 / ce dernier aurait ensuite pu être utilisé par un autre programme Basic pour effectuer en Mode 0 le classement du second fichier temporaire.

Conception d'un programme de gestion de données

Les ordinateurs sont généralement acquis pour mémoriser et traiter des données en quantité importante.

Sur les gros et les moyens systèmes, on dispose de programmes de gestion des données spécifiques, capables d'assurer la création et la maintenance des fichiers les plus complexes.

En revanche, les logiciels de gestion de données adaptés aux micro-ordinateurs se veulent universels et leur emploi impliquerait parfois des aménagements conséquents.

Il est alors préférable de rédiger soi-même un programme moins performant peut-être mais mieux adapté à un travail particulier.

Pour éviter d'écrire un programme spécifique à chaque application, on aura avantage à concevoir un logiciel bien structuré regroupant plusieurs modules paramétrés.

La modification de la valeur des paramètres offrira ainsi une plus large gamme d'utilisation. En règle générale, un programme de gestion de données est articulé sur les opérations suivantes :

- acquisition des données ;
- traitement ;
- mémorisation sur disque.

Il est préférable d'établir des sous-programmes gérant chacun une opération.

La rédaction d'un programme général en sera grandement simplifiée puisqu'il ne comportera guère que des renvois aux routines adaptées.

Il faudra, avant tout, spécifier le type et la longueur des variables à traiter.

Ces informations sont en effet indispensables aux sous-programmes de saisie et de mémorisation.

L'acquisition des données

Supposons qu'on veuille créer des enregistrements de la structure suivante :

Nom		9 caractères
Code		4 caractères
	Jour	2 caractères
Date	Mois	2 caractères
	Année	2 caractères

On devra définir un masque de saisie écran pour contrôler le nombre de caractères et le type de chaque donnée (ainsi la date doit être constituée exclusivement de caractères numériques). Toutefois, si une autre acquisition portant sur des données d'un autre type ou d'un autre format est nécessaire plus loin dans le programme, un masque différent devra être défini.

Préparation du masque de saisie

Les caractéristiques d'un masque de saisie sont définies par un certain nombre de données :

- ligne et colonne de début (position sur l'écran) ;
- nombre de lignes (chaque ligne correspondant à une entrée) ;
- libellé de chaque variable ;
- nombre de caractères de chaque variable ;
- type des variables.

Ainsi, dans l'exemple mentionné plus haut, nous aurions :

- position de début : ligne 4, colonne 6 (valeurs arbitraires) ;
- nombre de lignes ;
- libellés : 1 - Nom
  2 - Code
  3 - Jour
  4 - Mois
  5 - Année
- longueur de chaque variable :
  Variable 1 = 9 caractères
  Variable 2 = 4 caractères
  etc.
- type des variables
  Variable 1 = alphabétique
  Variable 2 = numérique
  etc.

Le rôle d'un masque de saisie est d'afficher le libellé suivi d'autant de points que de caractères à entrer, puis de positionner le curseur à l'emplacement du premier de ces caractères (coordonnées 6 et 4).
Les libellés peuvent être stockés dans une variable alphanumérique indicée. Dans notre exemple, cette variable devra être dimensionnée à cinq éléments. Si nous l'appelons LIB$(5), il suffira de demander l'affichage de LIB$(1) pour faire apparaître le premier libellé, de LIB$(2) pour le second et ainsi de suite. Pour les pointillés, le caractère « . » sera défini en temps que variable alphanumérique et utilisé pour former des chaînes de la longueur désirée. Ainsi, en posant A$=« . » et B$= STRING$(9,A$), on crée une chaîne B$ constituée de 9 points. On obtiendra la première ligne du masque de saisie par l'instruction :

```
PRINT LIB$(1)+" "+B$
```

L'espace sert à séparer le libellé des pointillés pour améliorer la présentation. L'emploi des instructions DATA et READ est le plus indiqué pour affecter les valeurs voulues aux paramètres du masque de saisie. On procèdera donc au remplissage de la table des libellés de la façon suivante :

```
10 RESTORE 30
20 FOR I=1 TO 5:READ LIB$(I):NEXT I
30 DATA "1 - Nom"
40 DATA "2 - Code"
50 DATA "3 - Jour"
60 DATA "4 - Mois"
70 DATA "5 - Année"
```

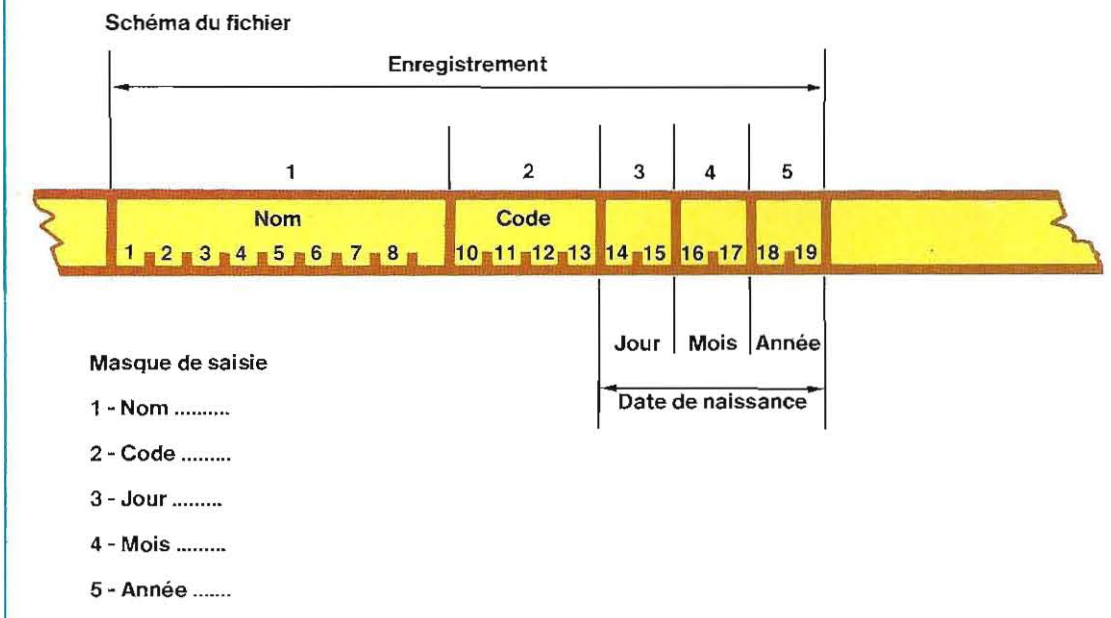
Dans le sous-programme de mémorisation, il faudra définir la longueur de chaque zone mais aussi sa position dans l'enregistrement : la zone « Nom » de l'exemple occupe les octets 1 à 9, la zone « Code » les octets 10 à 13, etc. Voici les noms qui ont été adoptés dans les programmes listés plus loin :

PO(I) = Premier octet de la variable I
DO(I) = Dernier octet de la variable I
TC(I) = Type de la variable I

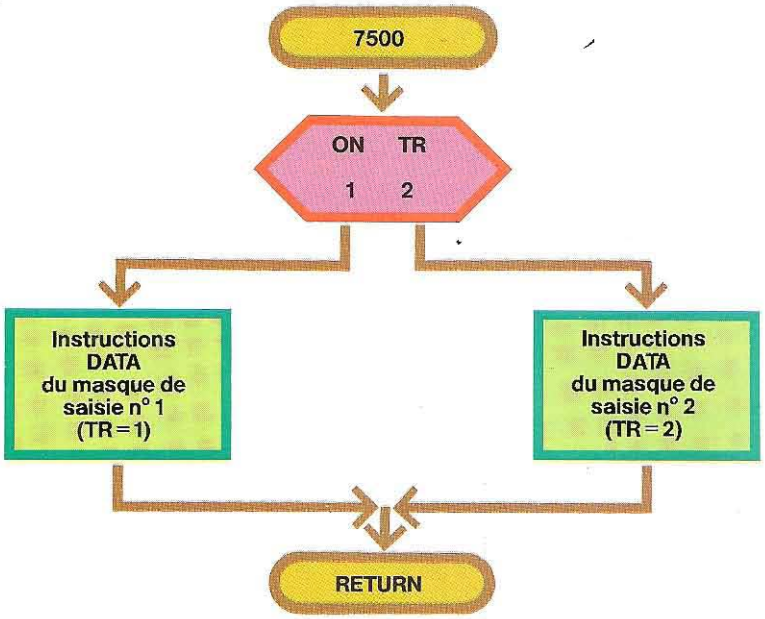
Dans le cas présent, il faudrait donner à ces variables les valeurs suivantes :

PO(1) = 1 DO(1) = 9 (première zone, octets 1 à 9) ;
PO(2) = 10 DO(2) = 13 (seconde zone, octets 10 à 13) ;
PO(3) = 14 DO(3) = 15 (troisième zone, octets 14 à 15).

STRUCTURE D'UN ENREGISTREMENT ET MASQUE DE SAISIE CORRESPONDANT



SOUS-PROG. D'AFFECTATION DES LIBELLES ET DES LONGUEURS DE CHAMPS



On définira de la même façon le type de chaque champ. A cet effet, on adoptera, par exemple, la convention suivante :

TC(I) = 1 pour une variable numérique ;
TC(I) = 2 pour une variable alphanumérique ;
TC(I) = 3 pour une variable faisant l'objet d'une simple présentation.

La première zone de notre exemple sera alors de type 2, tandis que toutes les autres seront de type 1. On utilise le type 3 pour définir des champs dont le contenu doit être visualisé sans que l'opérateur puisse le modifier.
Certaines applications nécessitent l'affichage de plusieurs masques de saisie successifs : c'est le cas du programme que nous allons maintenant développer. Il sera judicieux de réutiliser les mêmes variables pour chacun des masques. Il suffit alors de prévoir plusieurs groupes d'instructions DATA et d'activer sélectivement celui qui contient les données voulues. On peut, au niveau du programme d'appel, mémoriser le numéro du masque à visualiser dans une variable (appelée par exemple NM), dont la valeur déterminera, dans le sous-programme, le choix des instructions d'affectation correspondantes. Le sous-programme définit également le nom et le numéro du fichier dans lequel les données doivent être écrites (représentés respectivement par les variables NF$ et NO).
L'organigramme du haut de cette page correspond à un sous-programme dans lequel on n'a prévu que deux masques de saisie (NM peut prendre la valeur 1 ou 2). Pour lui conférer un domaine d'application plus large, il suffirait d'ajouter des groupes de DATA, des ordres de lecture et, bien sûr, de modifier en conséquence l'instruction de branchement multiple. Cette routine est listée page 698.
La structure de chaque nouveau groupe de DATA devra se calquer sur celle du bloc compris entre les lignes 7800 et 7824.
La première instruction (ligne 7530) sert à définir la position du masque à visualiser (KC=2, KL=3) ; l'exécution se poursuit ensuite en fonction du numéro de masque choisi (ligne 7540).
Dans chaque cas, on initialise tout d'abord les variables caractéristiques du masque : son nombre de champs (NC) et le numéro logique du fichier associé (NO). On positionne ensuite le pointeur de DATA, et on appelle la routine 7730, qui effectuera toutes les lectures et affectations (ces instructions sont en effet communes à tous les masques). Au retour, on mémorise le nom du fichier dans

AFFECTATION DES LIBELLES ET DES LONGUEURS DE CHAMPS

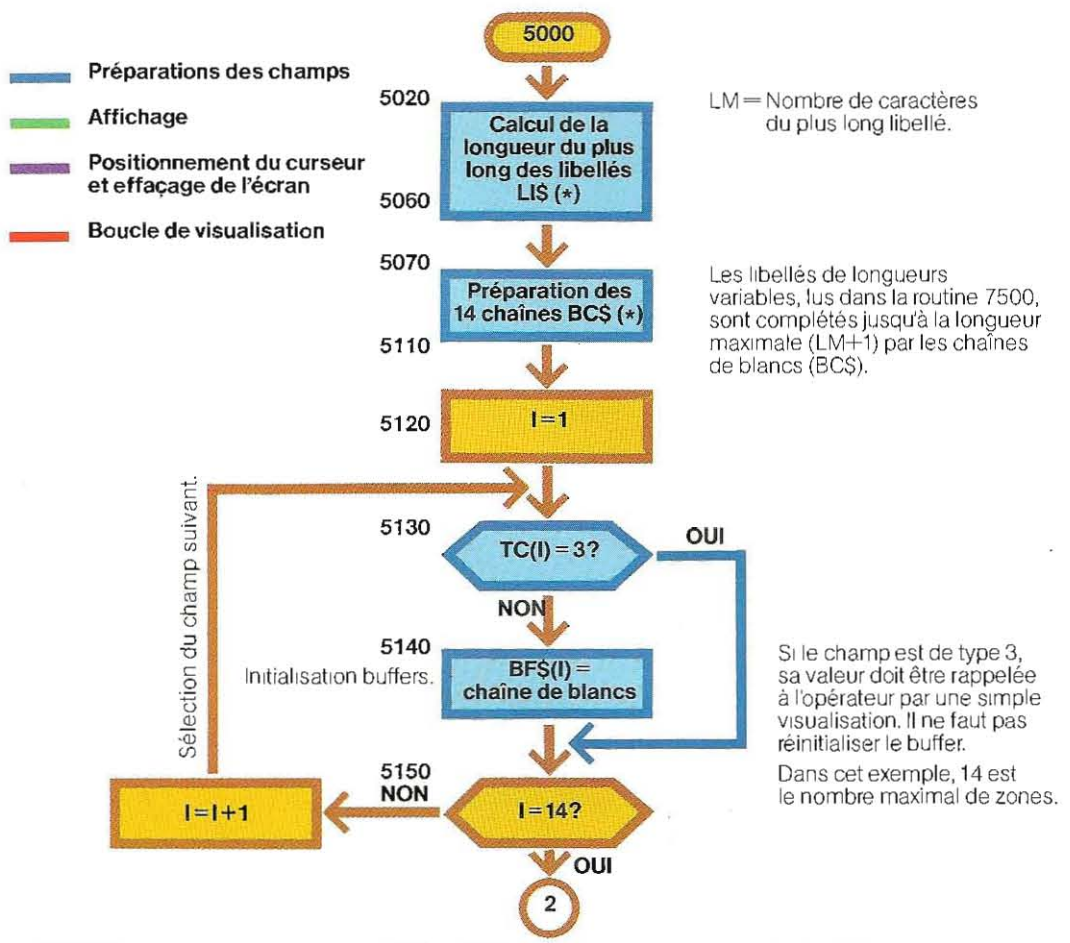
```
7500 '**SOUS-PROGRAMME DATA**
7502 '   FICHIER = ENCYCL
7504 '
7506 'ENTREE :    NM = numero de masque
7508 'SORTIES:    KC = position de la premiere colonne sur l'ecran
7510 '            KL = position de la premiere ligne
7512 '         NC(*) = nombre de champs
7514 '       LI$(*) = libelle des lignes
7516 '         TC(*) = type de champ
7518 '         PO(*) = pointeur vers le premier octet du champ
7520 '         DO(*) = pointeur vers le dernier octet
7522 '         LO : NO = numero logique du fichier
7524 '           NF$ = nom du fichier
7526 '           LN = nombre de caracteres d'un enregistrement du fichier
7528 '
7530 KC = 2 : KL = 0    'Position du curseur
7540 ON NM GOTO 7570, 7640    'Selection selon le numero de masque
7550 '
7560 '* SI NM = 1
7570 NC(NM) = 10 : N = 10 : NO = 1    'Initialisations
7580 RESTORE 7800                     'Positionnement dans les DATA
7590 GOSUB 7740                       'Lecture des DATA
7600 NF$ = "A:ACHVEN"                 'Nom du fichier
7610 GOTO 7690
7620 '
7630 '* SI NM = 2
7640 NC(NM) = 9 : N = 9 : NO = 2
7650 RESTORE 7830
7660 GOSUB 7740
7670 NF$ = "A:MVTMAR"
7680 '
7690 N = N+1                          'Reinitialisation des champs excedentaires
7700 FOR I = N TO 14 : LI$(I) = "" : TC(I) = 0 : PO(I) = 0 : DO(I) = 0 : NEXT I
7710 RETURN                           'Sortie du sous-programme DATA
7720 '
7730 '**LECTURE DES DATA
7740 FOR I = 1 TO N : READ LI$(I) : NEXT I
7750 FOR I = 1 TO N : READ TC(I) : NEXT I
7760 FOR I = 1 TO N : READ PO(I) : NEXT I
7770 FOR I = 1 TO N : READ DO(I) : NEXT I
7780 LN = DO(N)
7790 RETURN                           'Retour au sous-programme DATA
7795 '
7796 '**DATA
7798 '* cas ou NM = 1
7800 DATA "  1 - Type marchandise"
7802 DATA "  2 - Taux T.V.A."
7804 DATA "  3 - Unite de mesure"
7806 DATA "  4 - Cout Achat"
7808 DATA "  5 - Prix Vente"
7810 DATA "  6 - Cumul des Entrees"
7812 DATA "  7 - Valeur totale"
7814 DATA "  8 - Cumul des Sorties"
7816 DATA "  9 - Valeur totale"
7818 DATA "10 - ............"
7820 DATA 2!, 1!, 2!, 2!, 2!, 3!, 3!, 3!, 3!, 3!
7822 DATA 1!, 21!, 23!, 31!, 38!, 45!, 53!, 63!, 72!, 82!
7824 DATA 20!, 22!, 30!, 37!, 44!, 52!, 62!, 71!, 81!, 100!
7826 '* cas ou NM = 2
7830 DATA " 1 - CODE MARCH."
7832 DATA "2 - DATE MOUVEMENT"
7834 DATA " 3 - QTE ENTREE"
7836 DATA "4 - COUT ACHAT"
7838 DATA "5 - QTE SORTIE"
7840 DATA "6 - PRIX VENTE"
7842 DATA "7 - ..........."
7844 DATA "8 - ..........."
7846 DATA "9 - transfert Mvt."
7848 DATA 1!, 2!, 2!, 2!, 2!, 2!, 2!, 2!, 3!
7850 DATA 1!, 4!, 9!, 17!, 24!, 32!, 38!, 65!, 91!
7852 DATA 3!, 8!, 16!, 23!, 31!, 38!, 64!, 80!, 91!
```

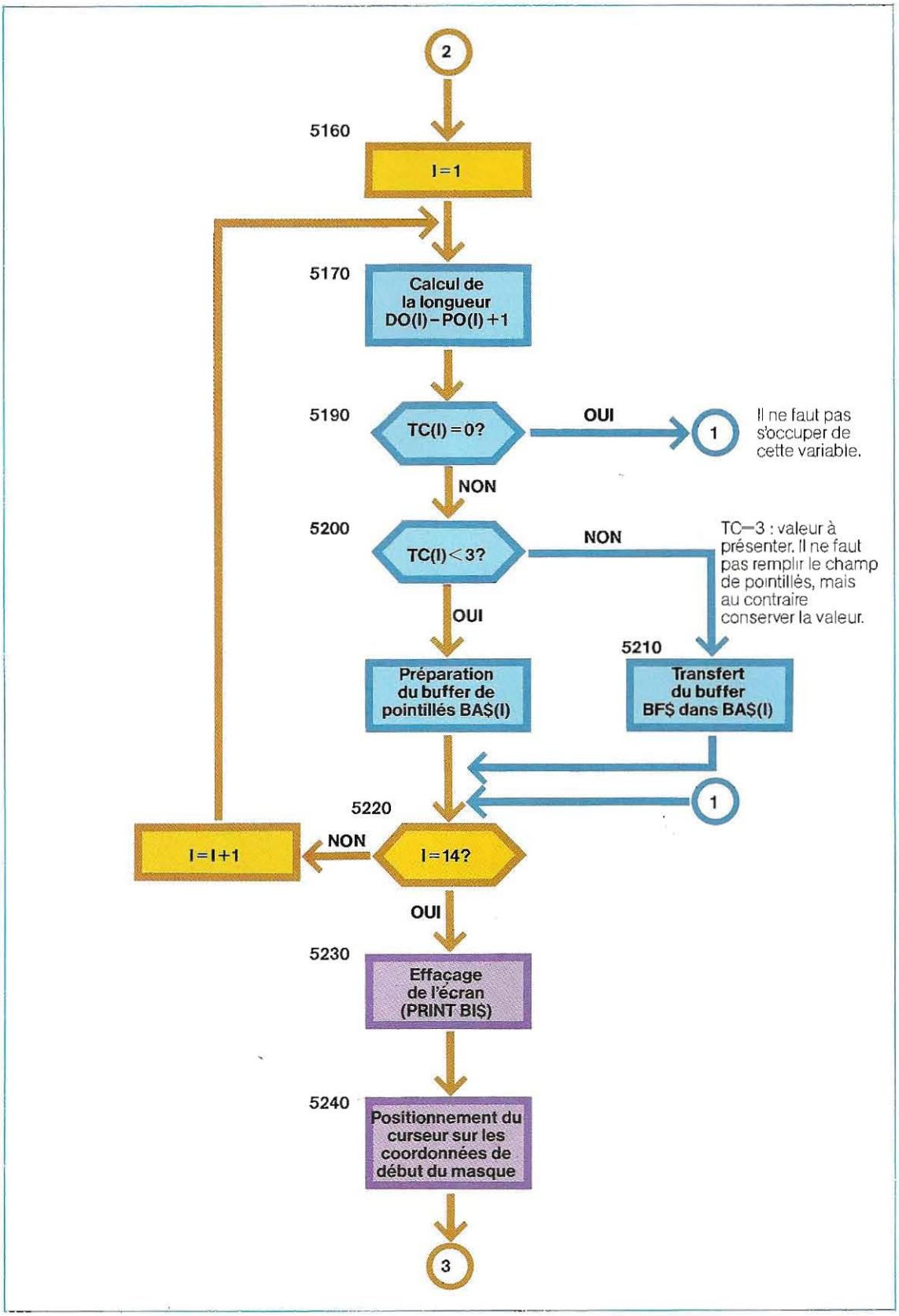
la chaîne NF$. Dans ce programme, le fichier correspondant au premier masque s'appelle ACHVEN et se trouve sur la disquette A.
Enfin, les dernières lignes (7690-7700) réinitialisent les éléments inutilisés des tableaux. Ainsi, avec le masque 1, on n'affiche que 10 zones, alors que le maximum prévu est 14. Les variables excédentaires sont mises à zéro ou à vide, selon leur type, dans une boucle qui commence à N+1, N étant le nombre de lectures effectivement exécutées (ligne 7570 pour le masque 1).

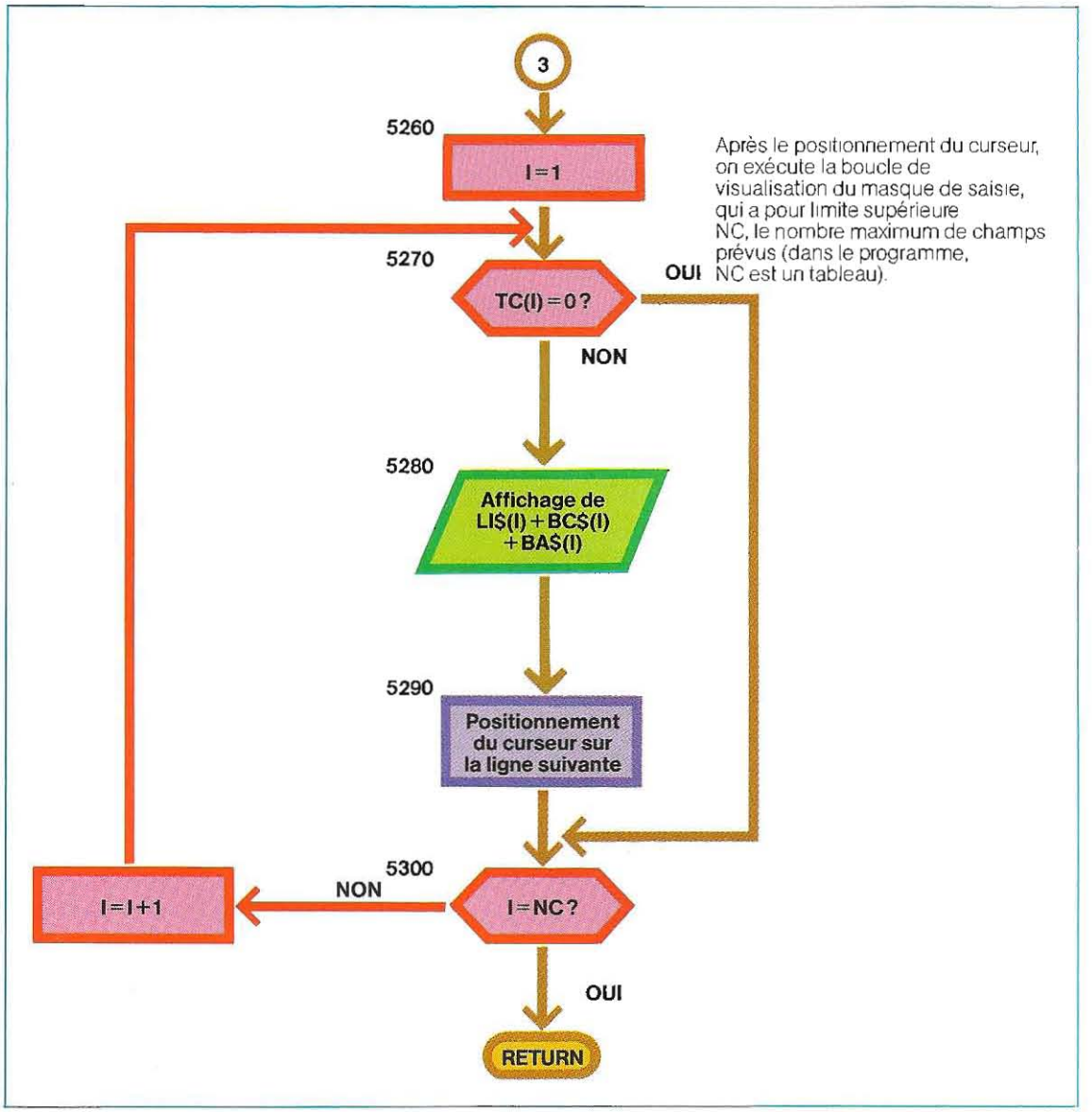
Affichage du masque de saisie. Une fois qu'on a appelé le sous-programme précédent, il faut visualiser le masque de saisie. Cette opération présente deux difficultés : le cadrage des zones et, éventuellement, la présence de variables de type 3.

Les différents libellés ne possèdent pas forcément la même longueur. Il ne faut donc pas afficher systématiquement les pointillés à la suite du libellé correspondant, car ils ne seraient alors pas alignés les uns avec les autres. Pour éviter cette mauvaise présentation, on procédera de la façon suivante : on cherchera la longueur du plus grand libellé, et on complètera chaque libellé par un nombre de blancs égal à la différence entre la longueur maximale et celle du libellé lui-même. En fait, il vaudra mieux prévoir chaque fois un espace supplémentaire, afin que les pointillés ne soient pas accolés au libellé le plus long.
L'organigramme du sous-programme de visualisation des masques de saisie commence page 699 et se poursuit pages 700 et 701. Le listing correspondant se trouve page 702.

SOUS-PROGRAMME D'AFFICHAGE DES MASQUES DE SAISIE




2
5160
I=1
5170
Calcul de la longueur DO(I)-PO(I)+1
5190
TC(I)=0?
OUI
1
Il ne faut pas s'occuper de cette variable.
NON
5200
TC(I)<3?
NON
TC=3 : valeur à présenter. Il ne faut pas remplir le champ de pointillés, mais au contraire conserver la valeur.
OUI
5210
Préparation du buffer de pointillés BA$(I)
Transfert du buffer BF$ dans BA$(I)
1
5220
I=I+1
NON
I=14?
OUI
5230
Effacage de l'écran (PRINT BI$)
5240
Positionnement du curseur sur les coordonnées de début du masque
3



Après le positionnement du curseur, on exécute la boucle de visualisation du masque de saisie, qui a pour limite supérieure NC, le nombre maximum de champs prévus (dans le programme, NC est un tableau).

Certaines des variables mentionnées dans ce sous-programme sont initialisées dans le programme principal, et notamment :

BL$ = chaîne d'espaces blancs
BP$ = chaîne de pointillés

La recherche du nombre de caractères du plus long libellé s'effectue par les instructions 5020 à 5060, tandis que la boucle suivante (5070 à 5110) construit les chaînes BC$ de blancs, qui viendront compléter les différents libellés LI$. Les chaînes BC$ sont générées par prélèvement de N caractères dans BL$ (instruction 5100).

Deux formes de présentation sont prévues dans le sous-programme : l'une pour les valeurs à saisir au clavier et l'autre pour celles qui doivent seulement être affichées. En effet, le type de champ (TC), lu dans le sous-programme 7500 prend les valeurs 1, 2 ou 3. Dans les deux premiers cas, on affichera une chaîne de pointillés (BA$). Pour le type 3, au contraire, la chaîne BA$ à afficher devra contenir la valeur lue et affectée au buffer BF$. Ainsi, pour obtenir la visualisation d'un masque de saisie, on appellera successivement le module 7500 qui contient les DATA, puis le module 5000. Pour visualiser le premier masque, on exécutera les instructions suivantes :

AFFICHAGE DES MASQUES DE SAISIE

```
5000 ' AFFICHAGE DES MASQUES DE SAISIE
5002 ' ** sous-programme visuel **
5004 '       FICHIER = ENCYCL
5006 '
5008 ' ENTREES : LI$(*),TC(*),PO(*),DO(*),NC(*),KC,KL de la routine 7500
5010 '           BF$(*)                                de la routine 1200
5012 '           NM                                    numero de masque
5014 ' SORTIE  : LO(*)=longueur des champs
5016 '           BF$(*)=buffer des champs, corrige a la bonne longueur
5020 LM=0
5030 FOR I=1 TO 14                  'Recherche de la longueur maximale
5040 N=LEN(LI$(I))                  ' des libelles
5050 IF N>LM THEN LM=N
5060 NEXT I
5070 '
5080 FOR I=1 TO 14                  ' Chaines de blancs pour completer
5090 N=LEN(LI$(I))                  ' les libelles
5100 NB=LM-N+1
5110 BL$(I)=LEFT$(BL$,NB)
5114 NEXT I
5115 '
5120 FOR I=1 TO 14
5130 IF TC(I)=0 OR TC(I)=3 GOTO 5150
5140 BF$(I)=BL$                     'Reinitialisation du buffer
5150 NEXT I
5155 '
5160 FOR I=1 TO 14
5170 N=PO(I)-PO(I)+1                ' Longueur du champ
5180 LO(I)=N
5190 IF TC(I)=0 GOTO 5220
5200 IF TC(I)<3 THEN BA$(I)=LEFT$(BF$,N) : BF$(I)=LEFT$(BF$(I),N) : GOTO 5220
5210 BA$(I)=BF$(I)                  ' Transfert de la donnee a afficher
5220 NEXT I
5225 '
5230 PRINT BI$          ' Efface l'ecran et emet un bip
5235 '
5240 X=KC : Y=KL : GOSUB 6900       ' Position curseur
5250 NE=NC(NM)                  ' Nombre de lignes a afficher
5260 FOR I=1 TO NE
5270 IF TC(I)=0 GOTO 5300
5280 PRINT LI$(I)+BC$(I)+BA$(I)
5290 Y=Y+1 : GOSUB 6900             Incrementation d'une ligne
5300 NEXT I
5310 RETURN
```

NM=1
GOSUB 7500
GOSUB 5000

Dans le module 7500, les cinq dernières zones du premier masque sont définies en type 3. Or, ces champs n'ont pas encore fait l'objet d'une affectation. Ils peuvent donc être vides, ou contenir des valeurs erronées provenant de traitements antérieurs. Si l'on veut afficher des valeurs correctes, il faut les initialiser avant d'appeler le sous-programme 5000. Ces zones portent les numéros 6, 7, 8, 9 et 10. Leurs libellés figurent aux lignes 7810 à 7818, et voici leurs longueurs respectives :

champ 6	de 45 à 52 = 8 caractères
champ 7	de 53 à 62 = 10 caractères
champ 8	de 63 à 71 = 9 caractères
champ 9	de 72 à 81 = 10 caractères
champ 10	de 82 à 100 = 19 caractères

Si l'on transfère des chaînes de caractères dans les buffers BF$ correspondants, celles-ci seront visualisées en même temps que le reste du masque de saisie. Ecrivons, par exemple :

NM=1
GOSUB 7500
BF$(9)="0123456789"
GOSUB 5000

L'exécution de ces instructions entraîne l'affichage de la chaîne "0123456789" sur la neuvième ligne du masque.

Entrée des données. Après le module 5000, on parvient en phase de saisie des données. Les opérations à exécuter sont les suivantes :

— positionnement du curseur ;
— lecture d'un caractère (sans écho) ;
— contrôle du caractère lu.

La suite des opérations dépend de l'issue de ce contrôle. Quatre cas peuvent se présenter :

1 - le caractère entré est un caractère de commande ;
2 - il n'est pas homogène avec le type de variable attendu ;
3 - il est homogène et donc accepté ;
4 - il n'est pas reconnu.

Dans le deuxième et le quatrième cas, il faut repositionner le curseur sur le même point et demander une nouvelle entrée (correction). Dans le troisième cas, le caractère est transféré dans le buffer de la variable ; si, par exemple, le curseur se trouve sur la ligne 4, le caractère est mis dans le quatrième buffer. Enfin, dans le premier cas, la machine doit exécuter les fonctions associées au caractère de commande.
Quand on conçoit un masque de saisie, il faut toujours offrir à l'opérateur la possibilité d'effectuer des corrections. Les déplacements du curseur sont commandés par quatre touches programmées dont la frappe déclenche l'envoi d'un code numérique spécial. Le tableau TF, qui regroupe les différents codes de commande, est initialisé dans le programme principal. Pour déterminer si la touche frappée est associée à une fonction, il suffit de balayer ce tableau. Si le nombre généré s'y trouve, on va se brancher à un endroit du programme défini par sa valeur (lignes 6450-6462). Ici, les codes associés aux touches programmées sont 10, 8, 9 et 11 (pour les déplacements), ainsi que 16, 17 et 13. Ces touches de fonction se répartissent en deux groupes : le premier commande des fonctions internes au masque de saisie et le second permet de sortir du sous-programme. La gestion des touches du premier groupe présente quelques difficultés. Prenons, par exemple, le cas des deux touches de déplacement vertical. Voici les opérations qu'elles réalisent et les codes correspondants :

— déplacement d'une ligne vers le haut, code 11 (ligne 6590) ;
— déplacement d'une ligne vers le bas, code 10 (ligne 6470).

On peut résumer ainsi les opérations effectuées par ces touches :

— incrémentation (ou, selon le sens du déplacement, décrémentation) du compteur de lignes CL ;
— incrémentation (ou décrémentation) de la coordonnée verticale Y ;
— remise à zéro du compteur de caractères (CC) ;
— positionnement horizontal en début de zone (X=X0).

Supposons, par exemple, que le curseur se trouve sur la ligne 3 (CL=3). Les données qu'on entre à ce moment sont transférées dans le buffer numéro 3 (BF$(3)). L'activation du code 11 devra entraîner :

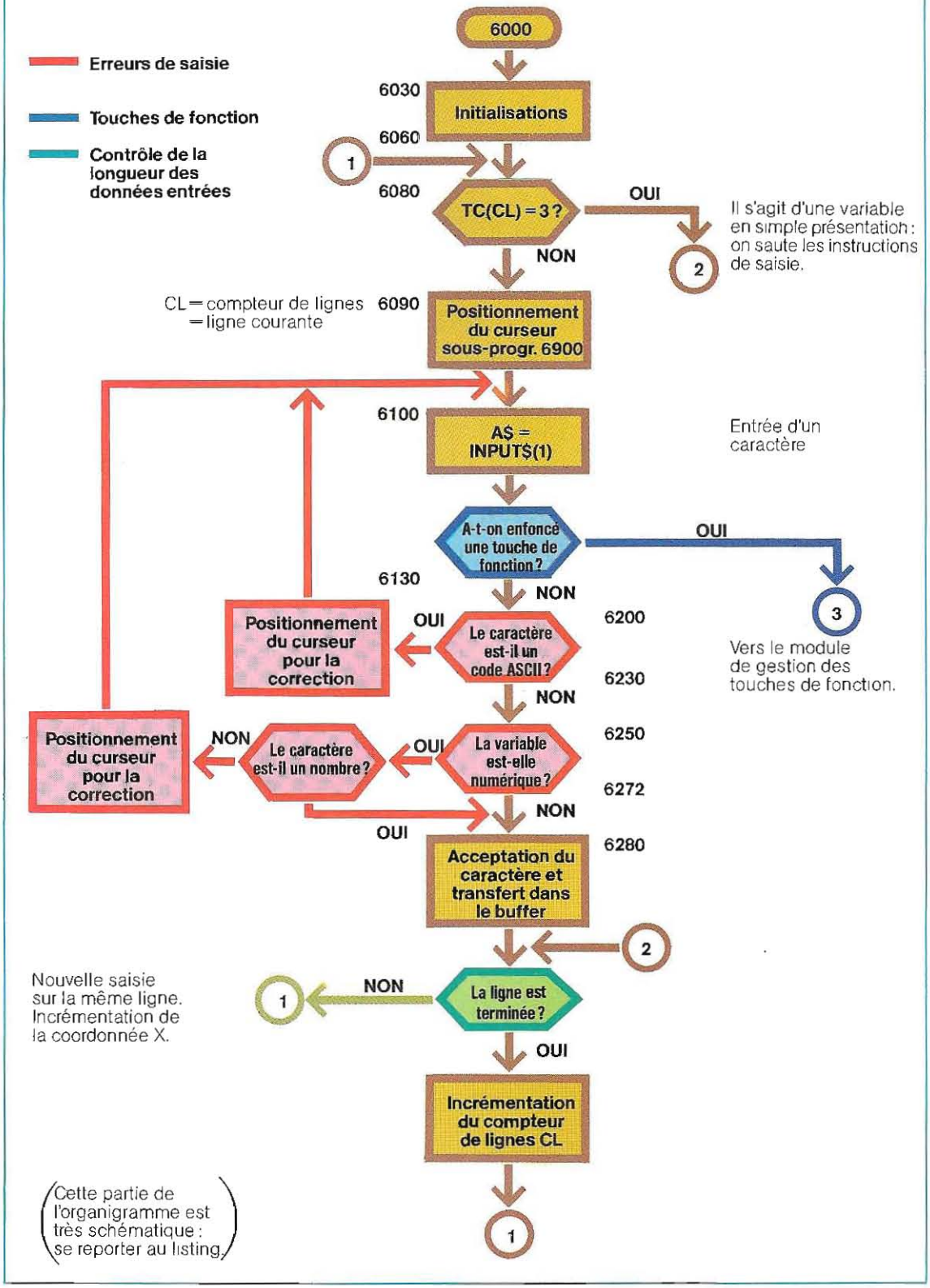
— la décrémentation du compteur de lignes (CL=3-1=2) qui doit désormais pointer vers le buffer numéro 2 ;
— la décrémentation de la position (Y=Y-1) pour que le curseur revienne sur la ligne précédente ;
— la remise à zéro du compteur de caractères (CC=0). Ainsi, le curseur ira se placer au début de la zone réservée à la donnée, et tout se passera comme si aucun caractère n'avait encore été transféré dans le buffer numéro 2.

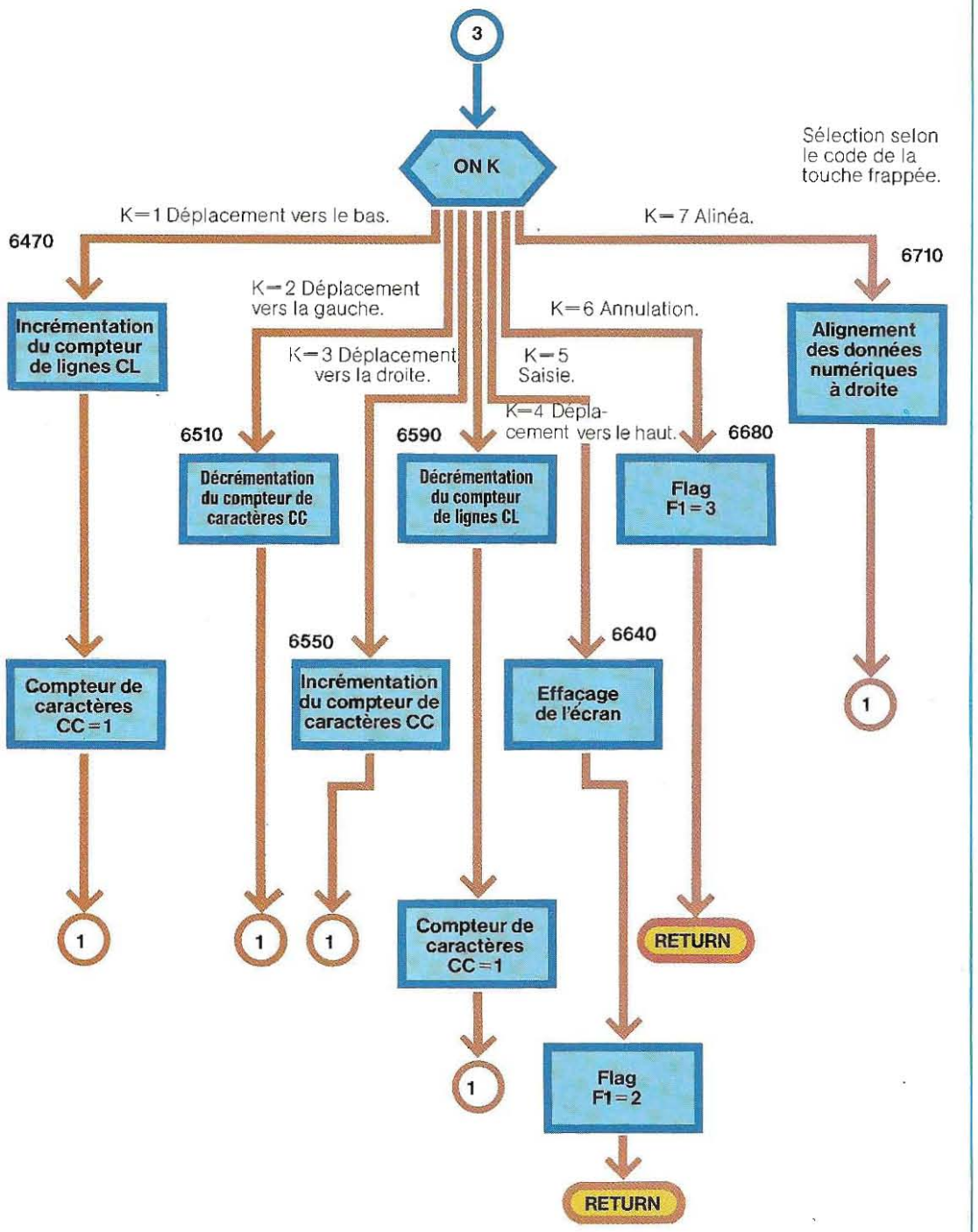
Les autres touches de fonction internes commandent les opérations suivantes :

— déplacement d'une position vers la droite, code 9, ligne 6550 ;
— déplacement d'une position vers la gauche, code 8, ligne 6510 ;
— justification, code 13, ligne 6710 (touche RT).

Le mécanisme des fonctions de déplacement horizontal est semblable à celui des fonctions de déplacement vertical, mais le compteur de lignes reste inchangé, alors que le compteur de caractères est incrémenté (ou décrémenté).

SOUS-PROGRAMME D'UTILISATION DES MASQUES DE SAISIE





On remarquera qu'il n'est possible de sortir du sous-programme qu'en frappant les touches de fonction associées à la saisie ou à l'annulation. Selon le cas, le flag est positionné à 2 ou à 3. Sa valeur est ensuite testée dans le programme principal, où elle sert d'aiguillage vers la saisie (2) ou l'annulation suivie d'une nouvelle saisie (3).

UTILISATION DES MASQUES DE SAISIE

```
5000 '
5001 ' UTILISATION DES MASQUES DE SAISIE
5002 '
5003 ' **SOUS PROGRAMME SAISIE**
5004 '    FICHIER = ENCYCL
5005 '
5006 ' ENTREES :      TC(*),KC,KL          de la routine 7500
5007 '               LM,LC(*),BF$(*)      de la routine 5000
5008 '               NM = numero du masque
5009 '               TF(20) = codes des touches programmées (pr. principal)
5010 '    code        ASCII      Signification standard     fonction programmée
5011 '      10        CTRL+J     line feed                  curseur bas
5012 '       8        CTRL+H     backspace                  curseur gauche
5013 '       9        CTRL+I     horizontal tabulation      curseur droite
5014 '      11        CTRL+K     vertical tabulation        curseur haut
5015 '      16        CTRL+P     DEL                        saisie des donnees
5016 '      17        CTRL+Q     DC1                        annulation des donnees
5017 '      13        CTRL+M     carriage return            justification
5018 ' SORTIES :   FI=flag  saisie/annulation
5019 '
5020 ' CL = compteur de lignes = ligne courante
5021 ' CC = compteur de caractères
5022 '
5023 '
5024 '
5025 '
5030 X0=KC+LM+1 : Y0=KL
5040 CL=1 : CC=0
5050 FI=0                    ' Flag de saisie/annulation
5060 X=X0 : Y=Y0
5065 '
5070 IF TC(CL)=0 THEN GOSUB 6850 ' Ligne de masque inexistante : on cherche la suivante
5080 IF TC(CL)=3 GOTO 6400       ' Valeur affichee : pas de saisie
5090 GOSUB 6900                  ' Position du curseur en (X,Y)
5095 '
5100 A$=INPUT$(1)                ' Saisie d'un caractere sans echo
5110 GOSUB 6500
5120 V=ASC(A$)                   ' Conversion
5125 '
5130 K=0                         ' La touche frappée est-elle programmee?
5140 FOR I=1 TO 20
5150 IF TF(I)=0 GOTO 6170
5160 IF V=TF(I) THEN K=I
5170 NEXT I
5180 IF K<>0 GOTO 6440           ' Gestion des touches de fonction
5190 PRINT A$
5195 '
5200 ' Controle du caractere frappé
5210 IF V<123 AND V>31 GOTO 6240     ' Caractere ASCII
5220 PRINT CHR$(7) : GOSUB 6850      ' On signale l'erreur a l'operateur
5230 GOTO 6090                       ' Vers une nouvelle saisie
5235 '
5240 ' Controle du type de caractere
5250 IF TC(CL)=2 GOTO 6280           ' Caractere alpha
5260 IF V>47 AND V<58 GOTO 6280      ' Caractere numerique correct
5270 PRINT CHR$(7)
5272 GOTO 6090
5275 '
5280 ' Caractere accepte : insertion
5290 N=LC(CL)                        ' Longueur totale du champ
5300 ND=N-LC                         ' Nombre de caracteres deja memorises
5310 C$=A$(+)                        ' Caractere en cours de saisie
5320 BF$=LEFT$(BF$(CL),N-1)
5330 ND=N-LC                         ' Nombre de caracteres encore a saisir
5340 IF ND=0 THEN BD$="" : GOTO 6360
6350 BD$=RIGHT$(BF$(CL),ND)
6360 BF$(CL)=BD$+A$+BF$
```

```
6370 IF CC=N THEN CL=CL+1 : CC=0 : Y=Y+1 : X=X0-1      ' La ligne est complete
6380 IF CL>NC(NM) THEN CL=1 : Y=Y0                     ' Retour en debut de masque
6390 Y=Y+1                                            ' Deplacement vers la droite
6392 GOTO 6070
6395 '
6400 ' Cas d'une valeur affichee a ne pas modifier (TC(CL)=3)
6405 '
6410 CL=CL+1 : Y=Y+1              ' On passe simplement a la ligne
6420 IF CL>NC(NM) THEN CL=1 : Y=Y0 : CC=0 : X=X0      ' Retour en debut de masque
6430 GOTO 6070
6434 '
6438 '
6440 ' ** Gestion des touches programmees **
6445 '
6450 ON K GOTO 6470, 6510, 6550, 6590, 6640, 6680, 6710
6455 '
6456 ' Erreur : code fonction non attribue
6460 ER=5 : PU=5
6462 RETURN          ' Sortie du sous-programme
6465 '
6470 ' * U=10        deplacement vers le bas
6480 CL=CL+1 : CC=0
6485 X=X0 : Y=Y+1
6490 IF CL>NC(NM) THEN CL=1 : Y=1        ' Retour en debut de masque
6500 GOTO 6070
6505 '
6510 ' * U=8         deplacement vers la gauche
6520 IF CC=0 THEN GOSUB 6950 : GOTO 6080      ' On ne fait rien
6530 CC=CC-1 : X=X-1
6540 GOTO 6090
6545 '
6550 ' * U=9         deplacement vers la droite
6560 IF CC>LC(CL) THEN GOSUB 6950 : GOTO 6080
6570 CC=CC+1 : X=X+1
6580 GOTO 6090
6585 '
6590 ' * U=11        deplacement vers le haut
6600 IF CL=1 THEN GOSUB 6950 : GOTO 6080
6610 CL=CL-1 : Y=Y-1
6615 CC=0 : X=X0
6620 IF TC(CL)=0 GOTO 6600
6625 ' Si la ligne du masque n'est pas valide : on remonte encore d'une ligne
6626 ' Lorsqu'on se deplace en descendant, la recherche d'une ligne valide
6627 ' s'effectue dans la subroutine 6850
6630 GOTO 6080
6635 '
6640 ' * U=16        touche de saisie
6650 PRINT BI$
6660 F1=2                    ' Flag de saisie
6670 RETURN                  ' Sortie du sous-programme
6675 '
6680 ' * U=17        touche d'annulation
6690 F1=2                    ' Flag d'annulation
6700 RETURN                  ' Sortie
6705 '
6710 ' * U=13        justification a droite des donnees numeriques
6720 IF TC(CL)=2 GOTO 6810          ' Donnee alpha
6730 A$=""
6740 N=LC(CL)-CC             ' Nombre de zeros a ajouter a gauche
6750 IF N=0 GOTO 6810
6760 FOR I=1 TO N
6764 A$=A$+"0"
6767 NEXT I
6770 X=X0 : GOSUB 6900
6780 IF CC<>0 THEN A$=A$+LEFT$(BF$(CL),CC)
6790 PRINT A$
6800 BF$(CL)=A$
6805 '
```

```
6810  CL=CL+1 : CC=0
6815  X=X0 : Y=Y+1
6820  IF CL>NC(CL) THEN CL=1 : Y=Y0
6830  GOTO 6870
6835  '
6840  '
6845  '
6850  ' ** Routine de recherche d'une ligne de masque valide **
6855  '
6860  CC=0 : Y=X0
6870  CL=CL+1
6880  IF CL>NC(CL) THEN CL=1 : Y=Y0        ' Retour en début de masque
6890  IF TC(CL)=0 GOTO 6870                ' On continue la recherche
6895  RETURN
6896  '
6897  '
6898  '
6900  ' ** Routine de positionnement du curseur **
6905  '
6910  PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X);
6920  RETURN
6925  '
6930  '
6940  '
6950  ' ** Routine de signalement des erreurs **
6960  '
6970  PRINT CHR$(7)
6980  PRINT CHR$(27)+CHR$(61)+CHR$(51)+CHR$(51);    ' Curseur en 20,20
6990  INPUT "ERREUR : tapez un caractere";S$
6995  PRINT CHR$(27)+CHR$(61)+CHR$(51)+CHR$(51); : PRINT " 30 "
6996  RETURN
```

Quant à la touche RT, on l'utilise pour aligner les données. En effet, lors de la saisie, le curseur se place au début de chaque zone, puis se décale d'une position vers la droite chaque fois qu'un caractère est saisi. Les données s'alignent donc toutes à gauche, quel que soit leur type, alors que les valeurs numériques doivent être cadrées à droite.

Les instructions 6710 à 6830 testent la nature de la variable (numérique ou alphanumérique selon la valeur de TC) et, éventuellement, rectifient l'alignement en ajoutant une chaîne de zéros à gauche du champ numérique. Elles sont activées par la frappe de la touche de justification.

Enfin, la fonction de positionnement du curseur est traitée séparément, dans un sous-programme d'une seule ligne (6900).

Quant aux fonctions du deuxième groupe, nous avons vu qu'elles permettent de sortir du module de visualisation. Au nombre de deux, elles se différencient par la valeur que prend le flag F1, selon qu'on frappe la touche associée à l'une ou à l'autre. On peut ainsi savoir, dans le programme principal, si les données sont correctes (touche 1, F1=2) ou si l'opérateur a décidé leur annulation (touche 2, F1=3).

Pour définir d'autres fonctions en programmant de nouvelles touches, il suffit de faire figurer leur code dans le tableau TF et d'ajouter au sous-programme de saisie les instructions nécessaires.

La phase de mémorisation sur disque

Lorsque la condition F1=2 est vérifiée (données correctes), les données doivent être écrites sur le disque. On peut encore concevoir un sous-programme d'emploi général.

Voici les quatre opérations qu'on peut effectuer sur un fichier :

- ouverture et définition de la zone tampon pour E/S ;
- lecture des enregistrements ; ,
- écriture des enregistrements ;
- fermeture du fichier.

Une instruction non paramétrée (CLOSE n) suffit à fermer un fichier. Il vaut donc mieux la placer dans le programme principal, plutôt que l'inclure dans la routine.

Les trois autres fonctions sont, au contraire, paramétrées. On devra donc, avant d'appeler

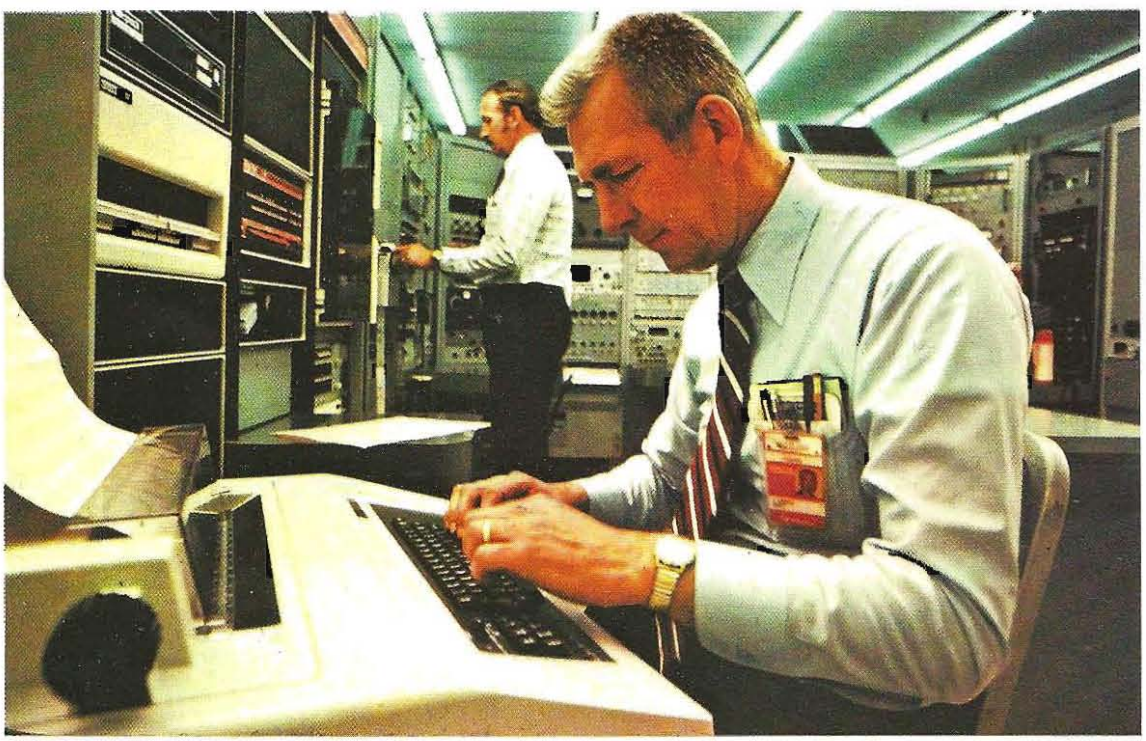
le sous-programme 1200 de gestion du disque, définir les paramètres suivants :

OP = flag indiquant l'opération désirée (ouverture, lecture, écriture) ;
NF$ = nom du fichier à traiter ;
NO = numéro d'unité logique de ce fichier ;
NE = numéro d'enregistrement, en lecture ou en écriture ;
LN = longueur de cet enregistrement (tableau calculé dans la routine 7500).

En outre, il faut dimensionner un tableau alphanumérique (DX$ par exemple), qui jouera le rôle de buffer d'entrées/sorties, et dont chaque élément sera associé au fichier de même numéro (voir ligne 1226). Ainsi, on posera NO=3, LN=100, NF$="A:ESSAI" et OP=1 pour ouvrir, sur le disque A, un fichier de nom ESSAI, de mémoire tampon DX$(3) et contenant des enregistrements de 100 caractères. Les opérations de lecture et d'écriture présentent une certaine complexité, et nous allons les étudier de façon plus détaillée.

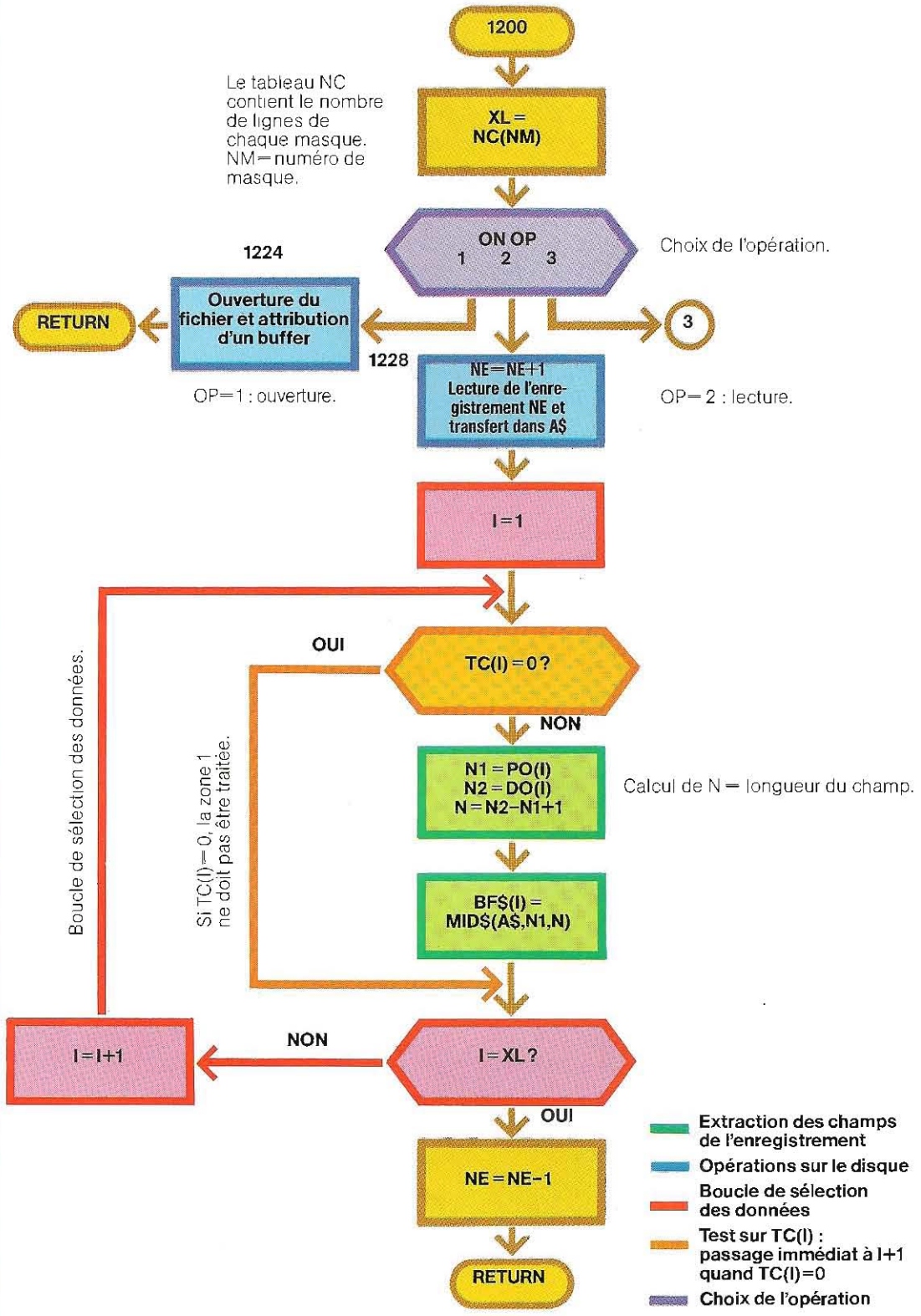
L'écriture (instructions 1242 à 1272). Conformément à la méthode que nous avons plusieurs fois mise en œuvre, le numéro du dernier enregistrement écrit est mémorisé dans le premier enregistrement, qui ne représente donc pas une donnée.
Ce décalage risquerait de provoquer des erreurs si l'utilisateur devait le contrôler lui-même, en ajoutant 1 avant de lire ou d'écrire une donnée et en s'en abstenant pour le premier enregistrement. On préférera donc le gérer automatiquement à l'intérieur même du sous-programme. Pour cela, on donnera le numéro 0 au premier enregistrement (NE=0), ce qui permettra d'ajouter systématiquement 1 pour accéder à n'importe quel enregistrement. Il faut naturellement soustraire 1 à NE avant de sortir du sous-programme, afin de lui rendre sa valeur au moment de l'appel.
Avant d'écrire un enregistrement, on doit regrouper les zones BF$ qui contiennent les données saisies à l'aide du masque.
On prépare les enregistrements de la façon suivante : après avoir cadré le contenu des buffers de saisie (BF$), on les concatène dans un ordre déterminé de manière à créer une chaîne A$, comme le montre le schéma de la page 713 et les instructions 1246 à 1263. On transfère ensuite cette chaîne A$ dans le

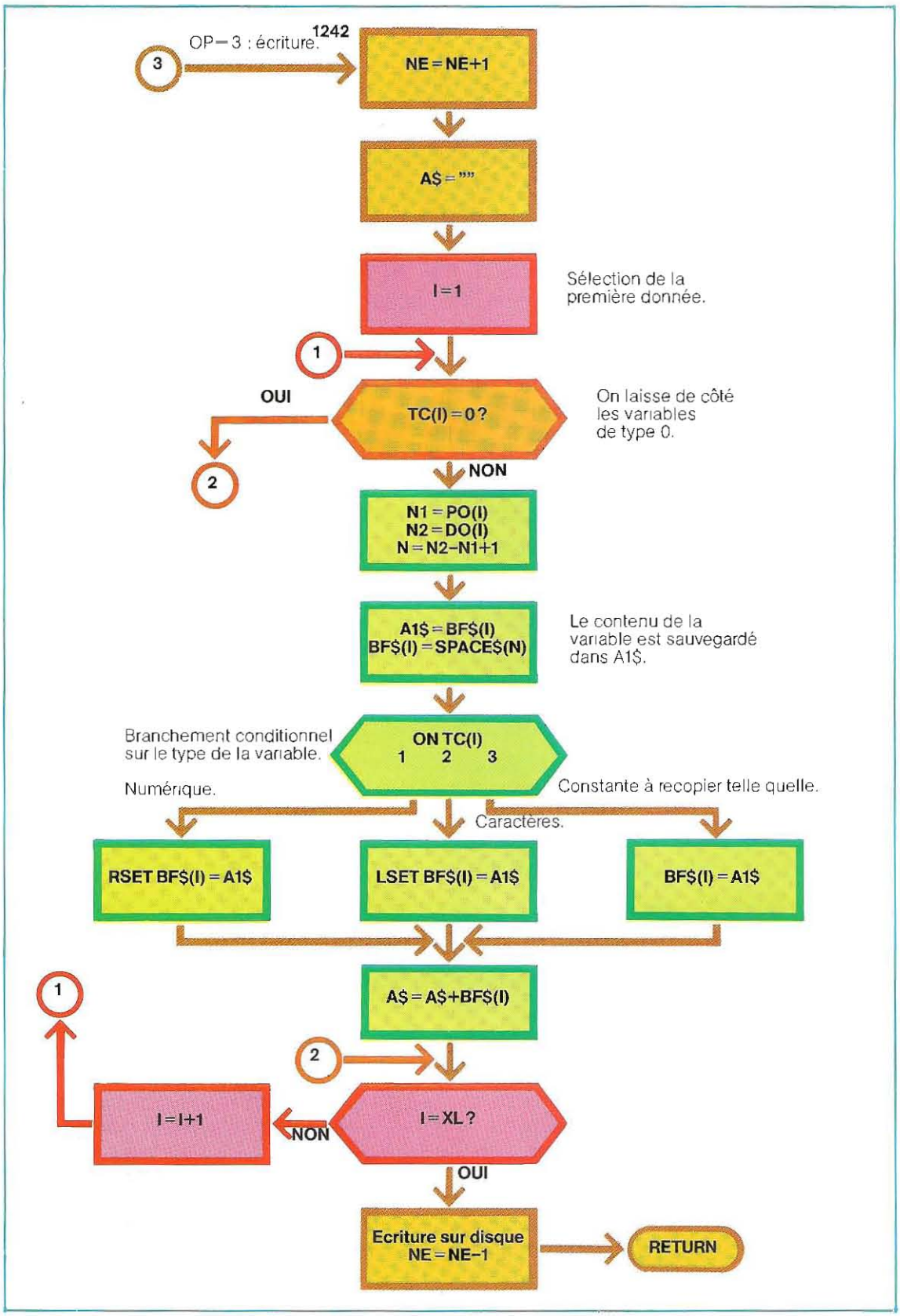
Techniciens vérifiant le fonctionnement du matériel dans un centre informatique.



K. Reese/Marka

SOUS-PROGRAMME PARAMETRE DE GESTION DU DISQUE




OP=3 : écriture.
1242
3
NE=NE+1
A$=""
I=1
Sélection de la première donnée.
1
OUI
TC(I)=0?
On laisse de côté les variables de type 0.
2
NON
N1=PO(I)
N2=DO(I)
N=N2-N1+1
A1$=BF$(I)
BF$(I)=SPACE$(N)
Le contenu de la variable est sauvegardé dans A1$.
Branchement conditionnel sur le type de la variable.
ON TC(I)
1 2 3
Numérique.
Caractères.
Constante à recopier telle quelle.
RSET BF$(I)=A1$
LSET BF$(I)=A1$
BF$(I)=A1$
A$=A$+BF$(I)
1
2
I=I+1
NON
I=XL?
OUI
Ecriture sur disque
NE=NE-1
RETURN

SOUS-PROGRAMME PARAMETRE DE GESTION DU DISQUE

```
1200 ' ** SOUS-PROGRAMME DISQUE **
1201 '    Lecture/ecriture sur disque
1202 '    FICHIER = ENCYCL
1203 '    ENTREES :
1204 ' NE = numero de l'enregistrement
1205 ' NO = numero du fichier = numero du buffer ; NF$ = nom du fichier
1206 ' OP = code operation
1207 '     1 : ouverture
1208 '     2 : lecture
1209 '     3 : ecriture
1210 ' LN,PO(*),DO(*),NC(*),TC(*)    parametres de la routine 7500
1218 '
1219 '
1220 XL=NC(NM)          ' Nombre de lignes du masque
1221 '
1222 ON OP GOTO 1224,1228,1242
1223 '
1224 ' **** OUVERTURE ****
1226 OPEN "R",NO,NF$,LN : FIELD NO,LN AS DX$(NO) : RETURN
1227 '
1228 ' **** LECTURE ****
1229 ' NE=NE+1     ' Incrementation pour obtenir le veritable numero d'enregistrement
1230 GET NO,NE : A$=DX$(NO)
1232 FOR I=1 TO XL : IF TC(I)=0 GOTO 1236
1234 N1=PO(I) : N2=DO(I) : N=N2-N1+1 : BF$(I)=MID$(A$,N1,N)      ' Extraction
1236 NEXT I
1237 NE=NE-1          ' Decrementation pour retrouver la valeur initiale
1238 RETURN
1240 '
1242 ' **** ECRITURE ****
1243 NE=NE+1
1244 A$=""
1246 FOR I=1 TO XL : IF TC(I)=0 THEN GOTO 1264
1248 N1=PO(I) : N2=DO(I) : N=N2-N1+1
1250 A1$=BF$(I)
1252 BF$(I)=SPACE$(N)
1254 ON TC(I) GOTO 1256,1258,1260
1256 RSET BF$(I)=A1$ : GOTO 1262
1258 RSET BF$(I)=A1$ : GOTO 1262
1260 BF$(I)=A1$
1262 A$=A$+BF$(I)
1263 NEXT I
1264 '
1265 PRINT "A=";A$ : S$=INPUT$(1)
1268 LSET DX$(NO)=A$
1270 PUT NO,NE
1271 NE=NE-1
1272 RETURN
```

buffer d'entrées/sorties associé au fichier (ligne 1268), et ce dernier sera recopié sur disque à la place qui lui revient (ligne 1270).

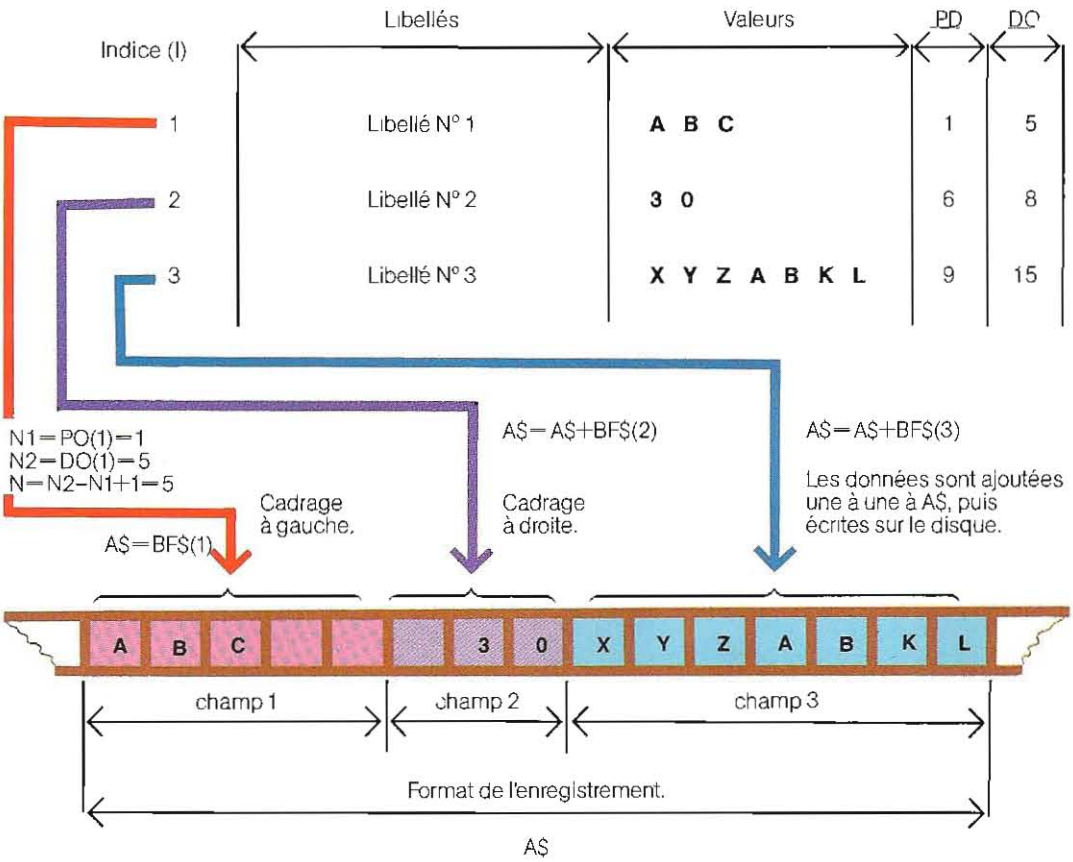
La lecture (lignes 1228 à 1238). Son mécanisme est exactement l'inverse du précédent. Après avoir extrait du disque l'enregistrement cherché (sans oublier d'ajouter 1), on transfère le contenu du buffer d'E/S du fichier dans la chaîne A$, puis on répartit les différents champs dans leurs buffers respectifs (BF$). Ce processus d'extraction est illustré par le schéma du bas de la page 713.

Le programme principal

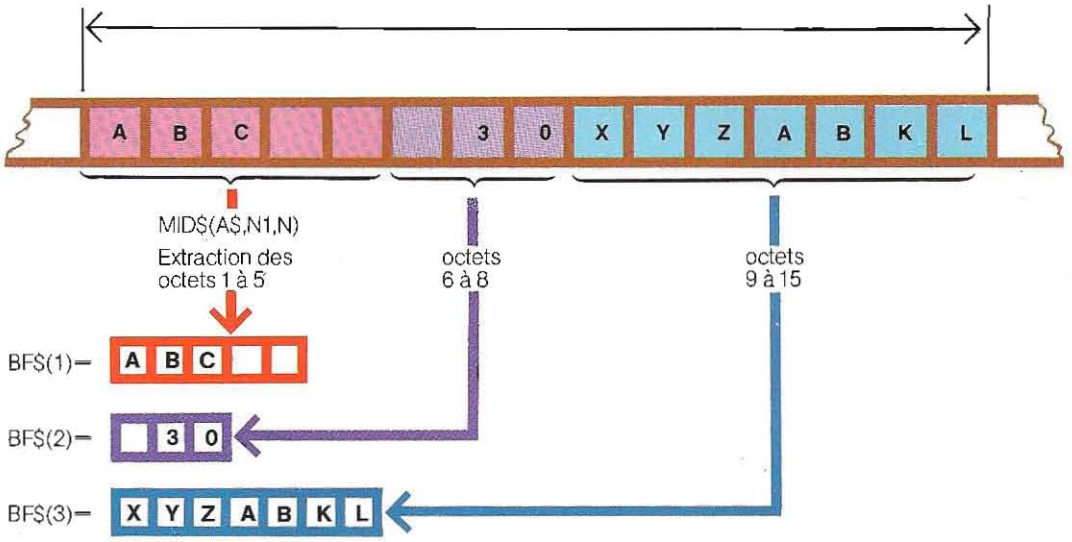
Nous avons conçu un programme principal afin de tester nos différentes routines. Son organigramme se trouve page 704 et son listing pages 715 et 716, suivi de celui du sous-programme 9000 de programmation des touches de fonction. Les codes correspondants à ces touches sont initialisés par les instructions 240 et 250 de DATA. On en a défini vingt, en prévision de développements futurs.

PREPARATION D'UN ENREGISTREMENT AVANT ECRITURE

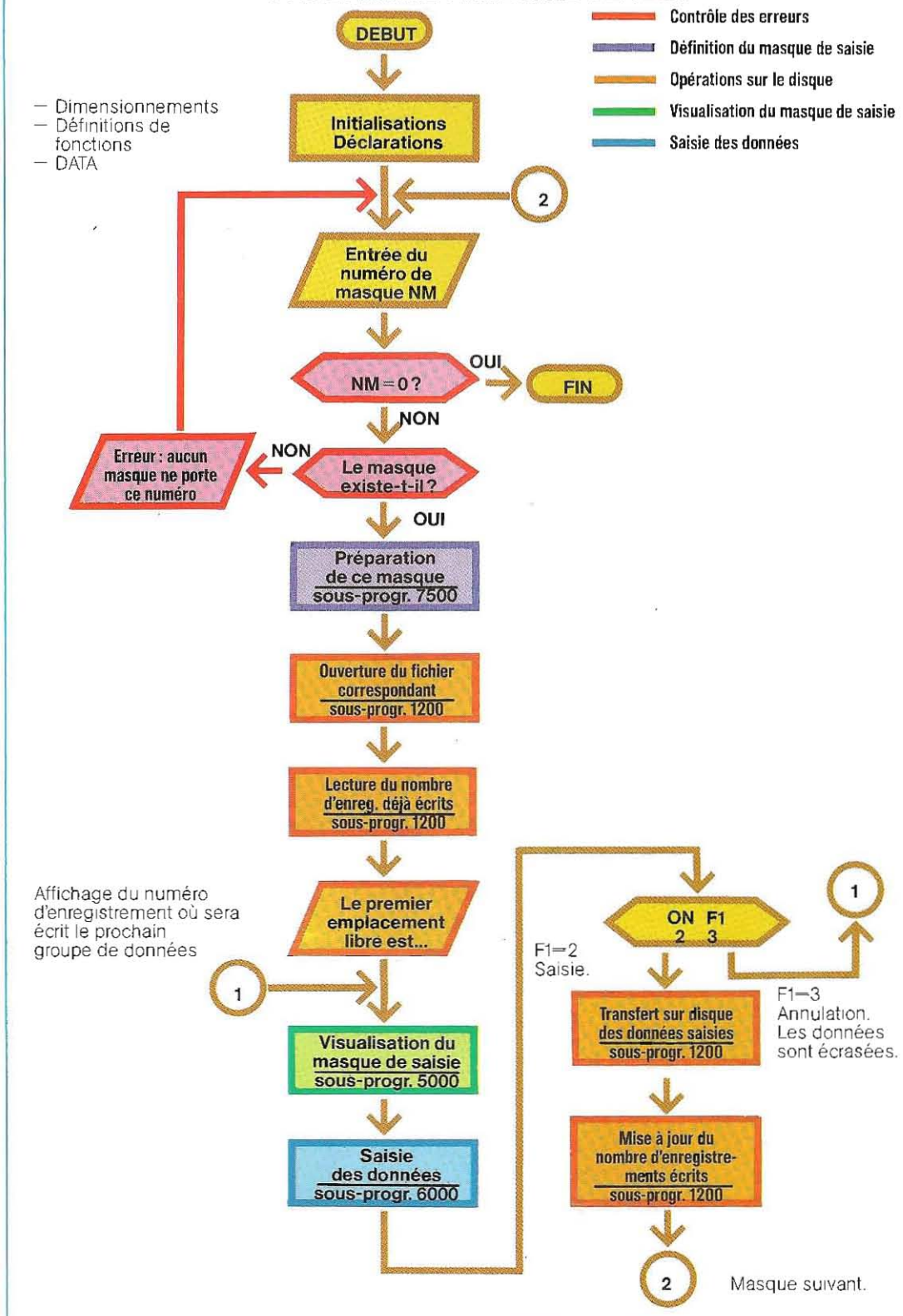
On construit un enregistrement en concaténant les différents champs par l'instruction A$=A$+BF$(I).



EXTRACTION DES CHAMPS DE DONNEES APRES LECTURE



PROGRAMME PRINCIPAL DE TEST



PROGRAMME PRINCIPAL DE TEST

```
10 ' ** PROGRAMME PRINCIPAL **
20 '    FICHIER = ENCYCL
30 OPTION BASE 1
40 '
50 ' **** DEFINITIONS ****
60 ' TF(*)  = codes des touches de fonction (8000)
70 ' DX$(*) = buffer d'E/S du fichier disque N° * (1200)
80 ' TC(*)  = type du champ * (7500)
90 ' DO(*)  = dernier octet du champ * (7500)
100 ' PO(*) = premier octet du champ * (7500)
110 ' LI$(*) = libelle du champ * (7500)
120 ' NC(*)  = nombre de champs du masque * (7500)
130 ' BF$(*) = buffers de saisie (6000)
140 ' BC$(*) = chaine de blancs pour completer les libelles (5000)
150 ' LC(*)  = longueur du champ * (5000)
160 ' BA$(*) = buffer de pointilles ou de donnee a presenter (5000)
170 '
180 '
190 '
200 '
210 DEFINT A-Z
230 DIM TF(20)          ' Tableau de 20 codes de fonctions
240 FOR I=1 TO 20 : READ TF(I) : NEXT I
250 DATA 10,8,9,11,16,17,13,0,0,0,0,0,0,0,0,0,0,0,0,0,
255 ' seules 7 touches sont utilisées. Pour en rajouter, il faut modifier le DATA
260 DIM DX$(4)
270 DIM TC(14),DO(14),PO(14),LC(14),NC(14)
280 DIM LI$(14),BF$(14),BC$(14),BA$(14),
300 '
310 '
320 '
330 ' * FONCTIONS *
340 '
350 BL$="                                        "
360 BP$="........................................"
370 BI$=CHR$(27)+"+"+CHR$
390 '
400 GOSUB 9000        ' Programmation des touches de fonction
410 PRINT BI$         ' Efface l'ecran et emet un bip
420 '
500 ' ** SELECTION DU MASQUE DE SAISIE **
510 INPUT "Taper le numero du masque desire ";NM
520 IF NM=0 THEN GOTO 900          ' Sortie
530 IF NM<1 OR NM>2 THEN PRINT "ERREUR : ce masque n'existe pas"
540 IF NM<1 OR NM>2 THEN S$=INPUT$(1) : GOTO 510
550 '
560 ' ** PREPARATION DU MASQUE **
570 GOSUB 7500          ' Chargement des DATA
575 '
580 ' ** OUVERTURE DU FICHIER **
590 OP=1 : GOSUB 1200
600 ' Lecture de l'enregistrement 1
610 NE=0 : OP=2
620 GOSUB 1200
630 DE=VAL(BF$(1))      ' Dernier enregistrement ecrit
640 NE=DE+1             ' Premier emplacement libre
650 PRINT "Le premier emplacement libre est : ";NE
660 PRINT "Appuyer sur une touche pour continuer" : S$=INPUT$(1)
665 '
670 ' ** VISUALISATION DU MASQUE DE SAISIE **
680 GOSUB 5000
685 '
690 ' ** SAISIE DES DONNEES **
700 GOSUB 6000
```

```
710 ' ** BRANCHEMENT CONDITIONNEL (VALEUR DE F1)
720 K=F1-1          ' On soustrait 1 : valeurs possibles 1 ou 2 au lieu de 2 et 3
725 '
730 ON K GOTO 740,870
735 '
740 ' * F1=2 : SAISIE *
750 OP=3            ' operation=ecriture
760 GOSUB           ' Ecriture des donnees dans l'enregistrement NE
770 '
780 ' Remise a jour du premier enregistrement
790 A$=STR$(NE)             ' Conversion en ASCII du nombre d'enregistrements
800 NE=0                    ' Le premier enregistrement est designe par 0
810 '                         On utilise le buffer BF$(1) pour transferer NE
820 N=DO(1)-PO(1)+1         ' Nombre de caracteres de BF$(1)
830 BF$(1)=SPACE$(N)        ' Mise a blanc de BF$(1)
840 RSET BF$(1)=A$          ' Cadrage a droite du contenu de BF$(1)
850 OP=3 : GOSUB 1200       ' Ecriture
860 GOTO 640                ' Entree suivante
865 '
870 ' ** ANNULATION DES DONNEES **
880 PRINT BI$
885 GOTO 640
890 '
900 ' ** SORTIE **
910 PRINT BI$
920 PRINT "*** AU REVOIR ***"
930 END

9000 ' ** PROGRAMMATION DES TOUCHES DE FONCTION **
9010 '
9020 '    FORME DU CURSEUR
9030 PRINT CHR$(27)+"c4AA"
9040 '
9050 '
9060 ' Les touches de deplacement du curseur (codes 10, 8, 12 et 11)
9065 ' sont gerees dans la routine 6000, ainsi que celle de justification (code 13)
9066 ' On code ici 8 touches (b a i) en leur attribuant les valeurs 16 a 23.
9067 ' Les deux premieres sont utilisees dans le programme
9068 ' et les suivantes laissees a la disposition de l'utilisateur
9069 ' qui devra bien sur les ajouter au DATA et definir leurs fonctions.
9070 A$=CHR$(27)+CHR$(48)    ' Premiere etape de la programmation des
9080 '                         touches de fonction
9090 B=38                   ' Code de la premiere touche de fonction (38 = code ASCII de b)
9100 I1=49                  ' 49 et 48 ASCII=10 HEXA car 49=1 et 48=0
9110 I2=48                  ' et 10 HEXA equivaut a 16 decimal
9120 FOR I=1 TO 8           ' Boucle de programmation des touches b a i
9122 ' La valeur decimale 16 est associee a la premiere touche de fonction (n° 98)
9130 C$=CHR$(I1)+CHR$(I2)   ' Code a associer a la touche I (1 a 8)
9140 D$=CHR$(B)             ' Selection de la touche
9150 E$=A$+D$+C$            ' Chaine contenant les codes de commande
9160 PRINT E$               ' La touche est programmee
9170 B=B+1                  ' Touche suivante
9180 I2=I2+1                ' Code suivant (incremente de 1)
9190 NEXT I
9200 RETURN
```

La compilation

Il est beaucoup plus commode de travailler en mode interprété pendant les phases de conception et de mise au point des programmes Basic. Par contre, l'exécution d'un programme sous le contrôle de l'Interpréteur est très longue pour trois raisons fondamentales :

- à chaque instruction d'appel (GOTO ou GOSUB), l'Interpréteur doit lire entièrement le programme jusqu'à la ligne où il doit poursuivre l'exécution ;
- quand un nom de variable est mentionné, l'Intepréteur doit chercher dans la table des variables l'emplacement de mémoire où se trouve sa valeur ;
- l'Interpréteur doit traduire chaque instruction en langage machine avant de l'exécuter. En particulier, dans une boucle, il décodera les instructions Basic à chaque itération. Or, si la traduction d'une instruction s'effectue très rapidement, on finit par atteindre des temps d'exécution assez longs quand on exécute de nombreux passages.

Tous ces inconvénients sont liés à la manière dont fonctionne l'Interpréteur.
On les évitera en **compilant** les programmes après leur mise au point. Le Compilateur se charge de traduire les instructions en binaire et de convertir les noms de variables en adresses absolues. L'exécution d'un programme compilé est donc bien plus rapide (4 à 10 fois). De plus, le Compilateur effectue un diagnostic global et relève éventuellement des erreurs qui ont pu échapper à l'Interpréteur.
C'est notamment le cas pour les instructions de branchement conditionnel. Si la ligne à laquelle renvoie l'instruction de saut n'existe pas, l'Interpréteur ne s'en apercevra (et, donc, ne le signalera) qu'au moment d'effectuer le saut, c'est-à-dire seulement si la condition est vérifiée.
L'erreur pourra donc passer inaperçue pendant la mise au point, et ne se révéler qu'après la mise en service du programme, ce qui risquera de fausser les données en cours de traitement, ou même d'entraîner leur perte.
En revanche, la compilation inclut un contrôle systématique des adresses, qui permet de déceler les erreurs éventuelles de ce type.

Le schéma de la page 718 retrace la succession des opérations d'écriture, de compilation et d'exécution d'un programme Basic.
Nous allons maintenant décrire en détail ces différentes étapes.

La préparation

Le Compilateur ne peut opérer que sur des programmes sauvegardés en ASCII. Il faut donc, avant tout, s'assurer que le programme source a bien été mémorisé sous cette forme : il doit être sauvegardé sur disquette (depuis la mémoire centrale) avec l'option "A". Ainsi, pour un programme baptisé ESSAI, on procédera de la façon suivante :

```
LOAD "A:ESSAI"
SAVE "A:ESSAI",A
```

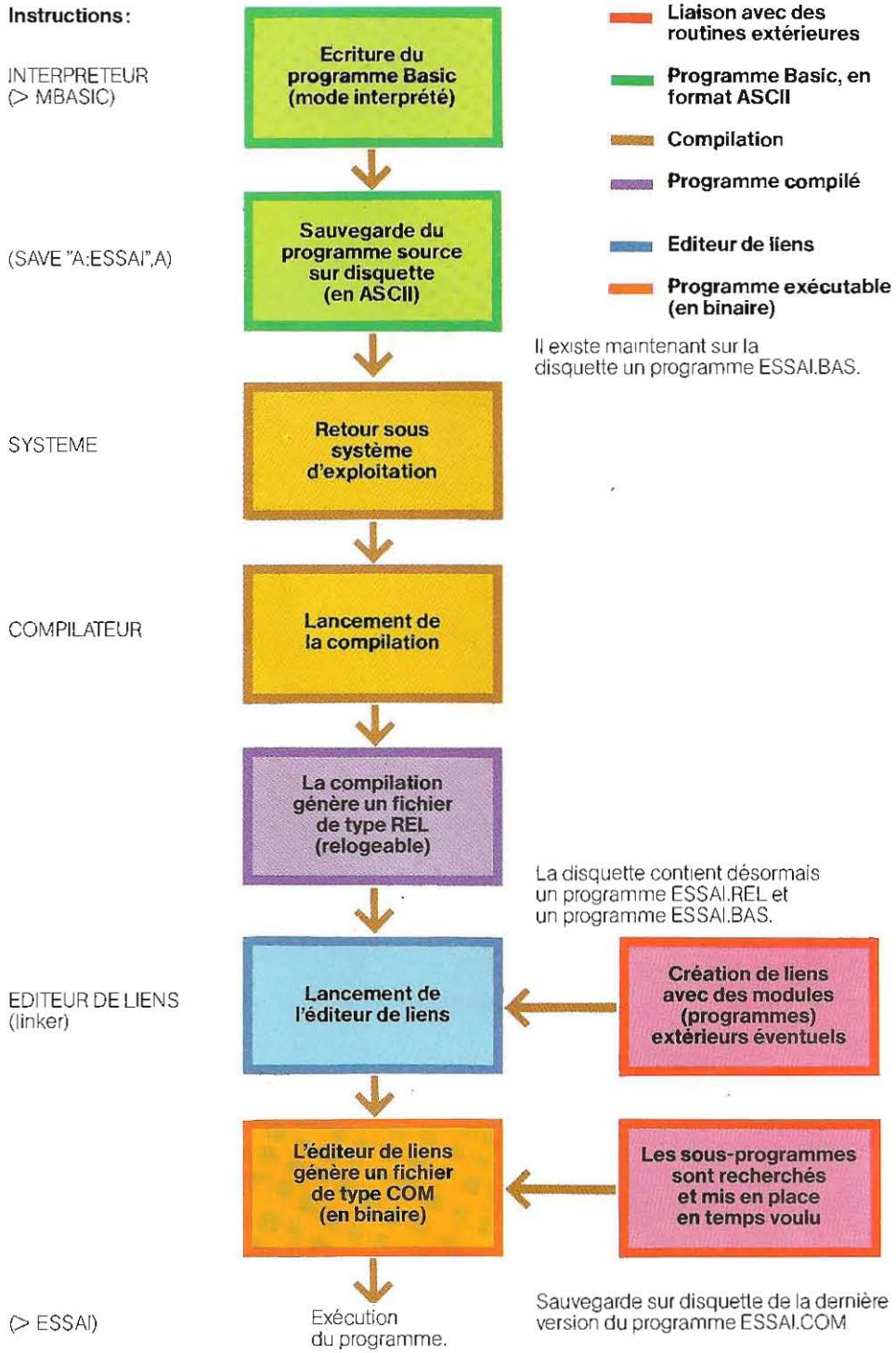
En outre, il est préférable – toujours pour raccourcir le temps d'exécution – de vérifier que les indices de toutes les boucles sont définis comme entiers, grâce à l'instruction DEFINT ou en spécifiant l'attribut % (écrire DEFINT I revient à faire suivre I du symbole %).
Enfin, il est nécessaire que tous les programmes (utilitaires du système ou non) qui seront appelés pendant la compilation, puis pendant l'« édition de liens »* (linkage) se trouvent sur la disquette A. C'est, en effet, sur l'unité A que travaille le Compilateur ; s'il est possible d'effectuer certaines opérations sur l'autre unité, il vaut mieux que tout se passe sur la même disquette. De toute façon, rien n'empêche de transférer le programme après sa compilation.
Les programmes indispensables à la compilation sont énumérés ci-dessous.

BASCOM.COM. C'est un programme système qui traduit le Basic en Assembleur. Il fournit une version compilée du programme source.

BRUN.COM. Il contient la plupart des routines utilitaires appelées par le programme d'application. Utilisé pendant l'exécution de celui-ci, il doit être présent dès la première étape afin que les liens nécessaires puissent être mis en place.

* Nous expliquerons un peu plus loin en quoi consiste l'édition de liens entre des programmes ou des parties de programmes. Considérons pour l'instant que ce terme désigne une série d'opérations qui relient des modules compilés séparément.

AVANT, PENDANT ET APRES LA COMPILATION



La disquette contient donc au moins trois versions du programme :
ESSAI.BAS = Programme source en Basic ;
ESSAI.REL = Programme compilé ;
ESSAI.COM = Programme exécutable en binaire.

Sammie et l'automobile

La conception d'une automobile doit réaliser le compromis le plus satisfaisant entre des contraintes d'ordre technico-économiques souvent antagonistes. L'équilibre est très difficile à trouver, car il faut respecter des normes de plus en plus nombreuses, répondre aux exigences d'un marché en constante évolution, tout en assurant aux véhicules d'excellentes performances.

La complexité du travail de conception ne se pose plus en termes de technologie, mais plutôt d'une multiplicité d'éléments à concilier pour offrir au consommateur un éventail de services de plus en plus diversifiés.

En effet, le véhicule familial ne représente plus aujourd'hui un simple moyen de transport utilitaire. Il doit répondre à une foule de besoins plus ou moins subjectifs.

Une bonne voiture aura une charge utile élevée et un habitacle spacieux. De ligne séduisante, elle consommera peu et sera facile à conduire pour les automobilistes peu expérimentés. Sa production, son entretien et sa réparation seront simples et peu coûteux.

Sur le plan commercial, ses ventes devront assurer rapidement les bénéfices nécessaires au financement de nouveaux modèles.

Beaucoup de temps s'écoule entre la conception d'un nouveau modèle et sa commercialisation, et il n'est pas rare que les ingénieurs et les responsables du marketing doivent imaginer les caractéristiques techniques de produits qui ne seront pas mis sur le marché avant une dizaine d'années. Ces prévisions sont suivies d'un ensemble d'études complexes et interdépendantes, depuis la réalisation de maquettes, de modèles, de prototypes, de moules et de machines-outils, jusqu'à la mise en place des équipements de production. Ce n'est qu'au terme de ce long processus que la production en série peut commencer.

On comprend donc l'intérêt des ***systèmes de CAO*** *(Conception Assistée par Ordinateur), qui réduisent considérablement les délais de conception et facilitent le recours aux techniques d'analyse structurelle, dynamique et vibratoire, qui permettent d'améliorer le produit et, par conséquent, de donner plus grande satisfaction à l'utilisateur.*

Avec les systèmes de CAO, on peut notamment rejeter tout de suite de mauvaises solutions, dont les défauts n'apparaissaient autrefois qu'après la construction des prototypes, alors qu'on manquait de temps ou d'argent pour résoudre les ultimes problèmes découverts. Les dernières étapes de la conception et les systèmes d'expérimentation non informatisés absorbaient d'énormes investissements.

De nombreux logiciels disponibles aujourd'hui rendent possible la conception de ***maquettes virtuelles tridimensionnelles*** *(3D). Ils permettent de définir simplement des formes géométriques à l'aide d'un langage évolué.*

On peut combiner à volonté ces éléments géométriques dans l'espace, pour représenter les objets les plus complexes, qu'il s'agisse d'un bâtiment, d'une pièce mécanique de moteur, d'une chaîne de production, du fuselage d'un avion, de la carcasse d'une automobile ou de n'importe quelle autre structure. Citons pour mémoire les systèmes français EUCLID (Matra-Datavision), CATIA (Dassault-Systèmes), UNISURF (Renault-Automatismes) et SISTRID (SNIAS).

Le logiciel de CAO américain SAMMIE offre la particularité intéressante de pouvoir représenter un personnage humain, et de simuler ses déplacements, facilitant ainsi toutes les études d'ergonomie en vogue actuellement.

Nous avons choisi l'exemple concret de la conception d'un nouveau modèle de voiture pour mieux illustrer cette possibilité.

Le dessinateur commence par résumer les études préliminaires dans un croquis où figurent les mesures les plus significatives (longueur totale, largeur, hauteur) et les formes proposées pour chaque surface.

A partir de ces quelques données disponibles immédiatement, on peut matérialiser l'idée initiale par un modèle à trois dimensions. On se contente, dans un premier temps, de représenter les surfaces courbes de la carosserie par un assemblage de carreaux plans. Cette opération, relativement rapide, demande environ 30 % de la durée totale de la phase de conception. Ainsi, même s'il ne dispose encore que de données extrêmement sommaires, le concepteur est déjà en mesure de se faire une idée relativement précises du nouveau véhicule.

On fait ensuite « tourner » la maquette sur elle-même, afin de pouvoir l'examiner en perspective sous tous ses angles ; cette étude, avec les techniques traditionnelles, était un luxe exigeant des heures de travail sur la planche à dessin.

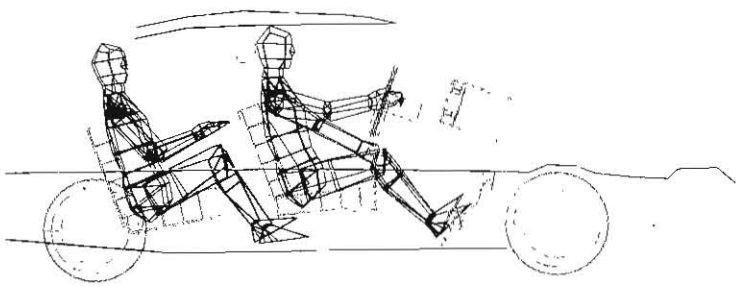
Après avoir défini la carosserie du véhicule, on commence à s'occuper de l'habitacle. Le concepteur doit alors tenter de réutiliser le plus grand nombre possible d'éléments ayant déjà servi sur les précédents modèles. On pourra, dans certains cas, reprendre les sièges d'une voiture antérieure et les adapter aux dimensions de la nouvelle carrosserie. Il suffira, pour cela, de rechercher dans la ***base de données*** *de SAMMIE les maquettes créées auparavant. On réduit non seulement la durée mais le coût des études.*

Il n'est évidemment pas possible d'appliquer cette méthode à tous les éléments de l'habitacle. La disposition ergonomique des appareils du tableau de bord soulève des difficultés particulières, car le conducteur doit pouvoir les consulter, à tout moment, d'un seul coup d'œil et sans effort.

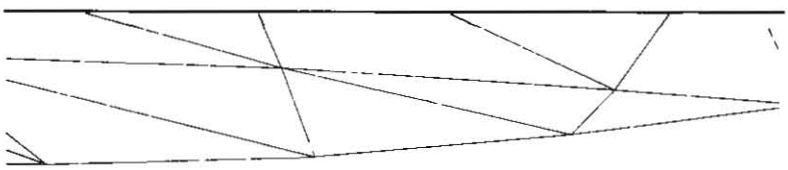
Il faut donc les placer à la fois le plus haut et le plus loin possible du conducteur. Cette nécessité est, bien sûr, en contradiction avec le besoin de visibilité et les contraintes de la géométrie extérieure du véhicule.

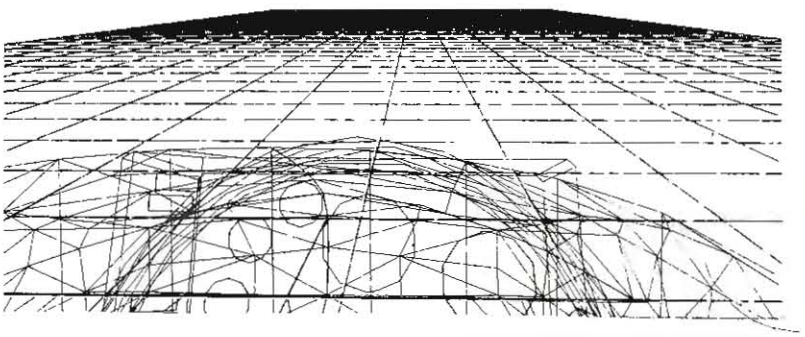
On peut résoudre ce genre de problèmes en recourant aux méthodes classiques mais la CAO offre une toute autre efficacité.

SAMMIE autorise une véritable ***« modélisation sous contraintes »,*** *qui prend en compte des données comme l'occupation de l'espace, les intersections et tangences autorisées, etc.*

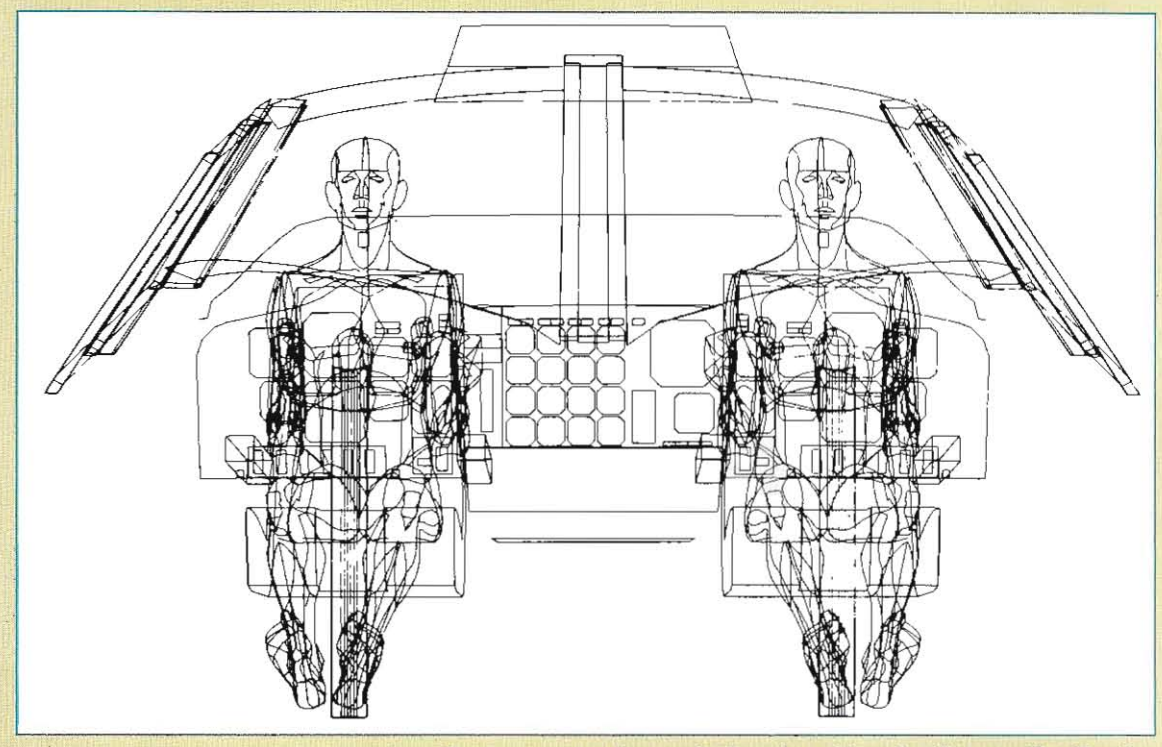


Les logiciels de CAO permettent aujourd'hui la conception rationnelle et rapide des structures les plus complexes.





Ces images ont été réalisées par une table traçante contrôlée par le logiciel SAMMIE. Elles illustrent deux étapes de la conception de l'habitacle d'un véhicule de tourisme. L'ingénieur peut s'assurer que les sièges, le volant et le tableau de bord se placent correctement les uns par rapport aux autres (en haut) et que la visibilité est satisfaisante (ci-contre). On aperçoit le tableau de bord au premier plan.



IBM

Deux mannequins assis dans la cabine de pilotage d'un Boeing. On notera la précision de la position du corps.

En pratique, le concepteur effectue une douzaine de tentatives qui, avec un système manuel, auraient duré plusieurs semaines. L'aménagement de l'habitacle prend environ 25 % du temps de conception.

SAMMIE peut représenter un personnage humain, opérateur ou homme-type, et le situer dans son milieu de travail. Le mannequin reproduit les caractéristiques cinématiques du corps humain : une suite de joints d'articulation, reliés à des membres rigides, schématise l'allure et la silhouette d'un personnage réel.

A première vue, on pourrait peut-être penser qu'une modélisation aussi rigide est inutile, et que le même résultat s'obtiendrait par des équations mathématiques. Mais on perdrait alors cette ***communication instantanée*** *par le biais de l'image. Le mannequin donne en fait une idée immédiate de l'orientation et de la position prise par le corps humain, et le concepteur peut ainsi vérifier facilement les relations qui s'établissent entre l'homme et son milieu de travail.*

La base de données utilisée par Sammie pour la création de l'homme-type peut en outre se modifier facilement, afin d'analyser la diversité des comportements due à la variété anatomique présente dans le monde réel.

Le mannequin est ensuite articulé dans tous ses joints, mais ses mouvements sont contrôlés par une table qui l'empêche de prendre des positions anormales. Le concepteur peut donc intervenir sur cette table pour simuler, par exemple, l'effet d'une contrainte quelconque, anatomique ou imposée par le milieu.

A la flexibilité du modèle, correspond la souplesse d'emploi des commandes qui feront bouger les articulations. Lorsqu'une commande est émise, le système répond à différents niveaux, par des rotations ***explicites*** *(mouvement demandé) et* ***implicites*** *(mouvement induits par les premiers) ; la précision de la réponse dépend du type particulier de mannequin que l'on a choisi.*

En retournant maintenant à notre maquette d'automobile, constituée de surfaces externes et d'objets internes, nous remarquons que le concepteur peut réellement se mettre à la place du conducteur. Il lui suffira pour cela de faire coïncider le champ de vision du mannequin (qui normalement doit regarder devant lui), avec l'angle sous lequel il veut voir le modèle en perspective.

Il pourra ainsi jeter un œil au tableau de bord, se faire une idée exacte de l'amplitude du déplacement que l'œil doit effectuer pour cette opération, et déterminer la partie de route qui sera alors exclue de son champ de vision. Afin de rendre cette vérification encore plus réaliste, on peut disposer des cylindres sur le tableau de bord, de façon à simuler la présence d'instruments.

Grâce au module SIGHT (repère) et un modèle de volant, on peut ensuite étudier différentes alternatives, afin de minimiser l'encombrement du tableau de bord et la réduction du champ de vision. Des améliorations ultérieures seront pratiquées, par une étude paramétrée de ces effets en fonction de la taille du conducteur.

Après avoir résolu le problème de la visibilité, il faut vérifier les possibilités de manœuvre. Le module REACH (atteindre) permet au concepteur de demander au mannequin d'essayer de toucher un point quelconque autour de lui, avec les doigts, la paume d'une main, le talon ou la semelle de sa chaussure. Ce module fonctionne avec un positionnement approximatif obtenu grâce au curseur, ou bien avec une précision géométrique fondée sur les coordonnées tridimensionnelles. En pratique, il permet de vérifier si le conducteur peut atteindre les instruments et si les manettes de commandes sont placées correctement les unes par rapport aux autres.

Grâce à la nature **hiérarchisée** du système, les fonctions cinématiques peuvent s'appliquer également aux composants mécaniques. Par exemple, les pédales et le levier de vitesses peuvent entrer en rotation autour d'un système de coordonnées locales, afin de simuler les positions limites et de vérifier l'accessibilité dans tous les cas, par une simple itération du calcul. On évitera ainsi, par exemple, qu'en quatrième, le levier de vitesses donne des crampes à 95 % des conducteurs, ou que 5 % d'entre eux n'aient à accomplir un effort excessif pour engager la première ou la marche arrière. Cette possibilité de modification rapide des perspectives, joint à une vaste bibliothèque et à l'**interactivité** entre la maquette et l'homme-type, permet une réduction notable du temps de conception, par rapport aux méthodes traditionnelles de type manuel.

Après avoir modélisé l'intérieur, il faut revenir plus attentivement aux questions d'esthétique et d'aérodynamisme qui, dans notre cas, impliquent des modifications de la carrosserie du véhicule. Ces **surfaces curvilignes** avaient été schématisées dans un premier temps, au moyen d'interpolations triangulées. La maquette extérieure doit maintenant être affinée, et prendre en compte de complexes descriptions de surfaces, à partir de modèles mathématiques très précis. Une fois définies, ces surfaces courbes sont intégrées à la structure hiérarchique de la maquette. Disposant d'un modèle mathématique rigoureux, le concepteur peut effectuer tous les calculs classiques d'aérodynamisme, de résistance des matériaux... comme s'il disposait d'un prototype classique.

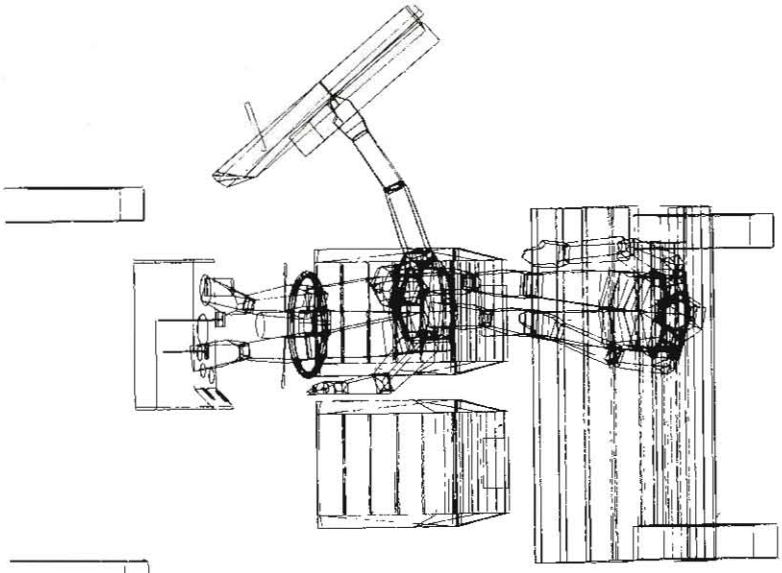
Après ce **lissage** de la surface externe de la carrosserie, on peut réévaluer la visibilité des instruments et de la route. Par exemple, on ajoutera un quadrillage représentant le terrain, et on y placera des modèles d'objets (des piliers, des feux de signalisation, etc.) pour mesurer la réduction de visibilité due à la présence d'un passager sur le siège avant, ou au rétroviseur. On peut ensuite dessiner une carte complète de la visibilité et obtenir des informations très utiles, avant même de disposer de maquettes de bois ou de prototypes.

SAMMIE propose également un module appelé MIRROR (miroir), qui transforme certaines surfaces internes du modèle en miroirs, pourvus des caractéristiques optiques nécessaires pour renvoyer en perspective les images réfléchies.

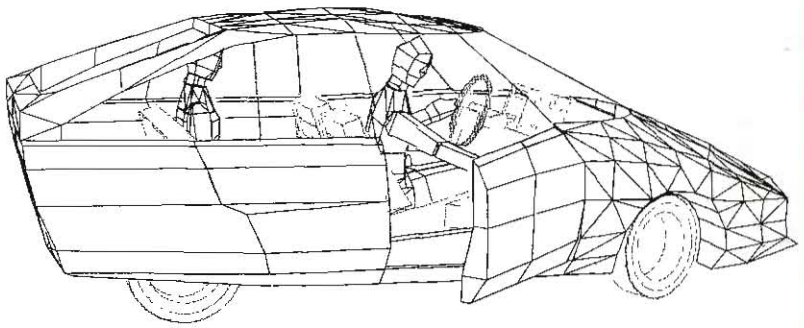
L'utilisateur peut aussi se servir de la commande ADJUST FOR OBJECT (vise un objet), qui, se fondant sur la dernière perspective appelée, modifie la position du miroir de façon que la bissectrice des deux vecteurs (allant de l'œil du conducteur au miroir et du miroir à l'objet à viser) soit perpendiculaire à la surface du miroir. Par cette opération, l'image de l'objet en question apparaît en perspective dans le miroir, avec des zones d'ombre ou d'interférences éventuelles.

L'élimination des parties cachées constitue la pierre d'achoppement des petits systèmes de CAO, pour la quantité de temps machine qu'elle demande.

SAMMIE offre la possibilité d'éliminer les parties cachées, mais son mode d'utilisation normale se limite à construire un modèle



L'intégration de commandes appropriées permet de contrôler les possibilités de mouvement et de manœuvre des personnages assis dans l'habitacle. Ici, Sammie vérifie que la poignée de la portière est à portée de main.



Un sérieux problème, qui apparaît dans la conception assistée par ordinateur, est le manque de lisibilité des images reproduites, dû à la présence de lignes internes qui, dans la réalité, seraient masquées par les surfaces externes. Avec Sammie, on peut les éliminer par une simple commande.

*représenté par un ensemble de lignes (**structure filaire** ou « fil de fer »), à le faire tourner et à mémoriser les différentes perspectives. A la fin de sa journée, le concepteur peut « mettre au propre » les perspectives qu'il a réalisées grâce à une fonction de HIDDEN LINES et les conserver comme documentation.*

La conception d'un projet suit donc le même cheminement qu'avec les techniques traditionnelles, mais se réduit à des temps beaucoup plus courts, et le système fournit toutes les informations nécessaires pour construire une maquette satisfaisante du premier coup. Toutes les évaluations dont nous avons parlé s'effectuent avec une précision et une vitesse supérieure à celle qu'on obtiendrait avec un crayon, du papier et une machine à dessiner. Par ce moyen, le concepteur se libère des tâches répétitives et ennuyeuses, pour se concentrer sur l'aspect le plus créatif de son travail.

Actuellement, les systèmes de CAO permettent de réduire les coûts des projets dans un rapport variant de 5 à 1 jusqu'à 100 à 1 et davantage. Généralement, ces évaluations n'intègrent pas le coût de réalisation de la maquette réelle, mais même en en tenant compte, le bilan demeure tout de même très positif.

Les systèmes de CAO actuels, extrêmement puissants, sont implantés sur de grosses machines (IBM, VAX...) et accessibles seulement à quelques entreprises très importantes. A l'opposé, seuls des logiciels graphiques plus ou moins rudimentaires (2D et non volumique) peuvent tourner sur micro-ordinateur. Il est regrettable que rien ne soit proposé entre ces deux extrêmes, et l'avenir appartient peut-être aux systèmes de CAO tournant sur micro-ordinateur, destinés aux PMI et PME, aux bureaux de stylisme ou aux cabinets d'architectes.

BASLIB.REL. Contient d'autres programmes qui complètent la bibliothèque BRUN.COM et qui peuvent être appelés par les programmes d'application. Le programmeur accède aux programmes système (par exemple, la fonction SQR) sans spécifier où ils résident. Tout le travail de traduction et d'édition de liens est pris en charge par le système, au moyen du Compilateur et du Linker (plus précisément Linking Loader). Il analyse chaque ligne du programme source, reconnaît l'appel à un programme système, recherche ce dernier parmi les différentes bibliothèques du système d'exploitation et établit les adressages corrects qui en permettront l'appel pendant l'exécution du programme d'application.

OBSLIB.REL. Contient des programmes et fonctions similaires à ceux de BRUN et BASLIB. La distinction réside dans un mode de gestion différent, conséquence d'une utilisation différente du programme d'application.
Pour certaines applications, le logiciel une fois mis au point ne sera plus jamais modifié, et l'utilisateur voudra l'implanter définitivement sur la machine. Dans ce cas, il sera stocké sur une mémoire ROM (accessible en lecture seulement) ; on le qualifie alors de « software permanent » (firmware). De cette manière, on obtient l'activation automatique du programme dès la mise sous tension de la machine ; l'utilisateur final n'a pas à se préoccuper de la phase de chargement. On adopte cette solution dans des cas particuliers qui répondent à deux critères fondamentaux :

- La machine et le programme constituent ensemble un système figé, consacré à une application précise immuable (sinon par le remplacement des mémoires).
- Ce système doit être produit en série, car il ne serait pas intéressant de préparer une mémoire ROM pour une application ponctuelle.

L'écriture d'un programme sur une mémoire ROM implique un mode de travail différent, puisqu'il n'est pas possible d'écrire des valeurs dans l'espace mémoire réservé à celui-ci. Les variables seront stockées dans une mémoire RAM, séparée de la mémoire ROM qui contient le programme, ce qui nécessitera des procédés plus complexes que d'habitude. Dans la pratique, cette structuration contraint à utiliser deux espaces mémoires distincts : l'espace programme (mémoire ROM) et l'espace données (mémoire RAM). Ces options particulières sont spécifiées à l'Editeur de liens, à l'aide d'indicateurs spéciaux qui demandent le chargement de la bibliothèque OBSLIB à la place des bibliothèques BRUN et BASLIB.

BCLOAD. Ce fichier contient les liens, c'est-à-dire les références aux emplacements mémoire où se trouvent les différentes parties du programme, ainsi que les programmes système appelés par ce dernier.

L80.COM. Ce programme, exécuté après BASCOM, établit les liens nécessaires au chaînage des différentes parties, et génère ainsi un programme final exécutable. Ce module Linking Loader (Editeur de liens) tient son nom des fonctions même qu'il exécute. Certains systèmes offrent d'autres fonctions spécifiques dont certaines seront décrites par la suite.

La compilation

Le compilateur (BASCOM) lit en entrée un fichier source (programme en Basic), et produit en sortie un fichier compilé (programme traduit en Assembleur), avec toutes les références à des adresses relatives.
Lors de l'écriture d'un programme, pour affecter la valeur 2.5 à une variable V, on utilisera l'instruction V=2.5, qui code sous une forme simple et synthétique l'opération à effectuer. Pour la machine, cette instruction n'a pas de sens ; il est nécessaire de la décoder, c'est-à-dire de spécifier l'adresse mémoire attribuée à la variable V et de traduire l'instruction en langage machine (transfert de la valeur 2.5 à l'adresse de V).
Pendant la compilation, toutes les instructions du programme sont traduites à tour de rôle, et à chaque variable se voit attribuer une adresse d'emplacement mémoire. Toutes les adresses utilisées sont **relatives** à une valeur d'index, déterminée par le système : en changeant cette unique valeur, on modifie l'implantation en mémoire de toutes les variables. Ainsi, le programme compilé est dit **relogeable**.
Le calcul qui, à partir des valeurs relatives,

permet d'obtenir les adresses absolues (physiques) dans la mémoire, est exécuté par l'**éditeur de liens** (Linking Loader). Le programme compilé relogeable est mémorisé dans un fichier du type .REL. La synthaxe de la commande de compilation est

BASCOM Objet, Liste = Source.

Voici la signification de ses différentes paramètres :

Objet. Il s'agit du nom donné au fichier de sortie (de type .REL) dans lequel sera mémorisée la version relogeable du programme. Ce type de fichier est appelé « object » (Objet).

Liste. Il s'agit du fichier « liste » qui contient le listing du programme source et sa traduction. Les pages 726 et 727 représentent le contenu de ce fichier dans un exemple de compilation. Le listing contient l'expansion de chaque instruction Basic en instructions Assembleur correspondantes et retrace la compilation. Comme on peut le voir, une instruction Basic génère une série d'instructions Assembleur, contenant aussi des appels aux programmes systèmes, par exemple CALL $CINA (l'appel à un programme Assembleur utilise le code CALL et non GOSUB).
Dans le listing, chaque instruction du fichier source est précédée de deux nombres hexadécimaux. Le premier représente l'adresse de la ligne relative à l'adresse de début du programme (nulle par définition), le second représente l'espace mémoire réservé aux variables de l'instruction. Par exemple, la traduction de la ligne Basic 50 se trouve à l'adresse relative 1F (31 en décimal). Lorsque le programme est chargé en mémoire, son adresse initiale ne vaut plus zéro, mais perd une certaine valeur (origine) déterminée par le système ; ainsi l'instruction 50, avec 1F pour adresse relative, sera positionnée à une adresse absolue égale à 1F plus l'origine (adresse à laquelle le programme a été chargé).

Source. Il s'agit du fichier ASCII où réside le programme à compiler.

Appareillages d'un centre d'instruction aéronavale. Les consoles reproduisent les situations des centres opérationnels de combat.



B.A. Liotta/Il Dagherrotipo

EXEMPLE DE COMPILATION

```
BASCOM 5.30 - Copyright 1979,80,81 (C) by MICROSOFT - 25298 Bytes Free
 0014 0007       10 ' ** ESSAI DE COMPILATION **
 0014 0007       20 OPTION BASE 1
 0014 0007       30 DIM A(5)
 0014 0007       40 FOR I=1 TO 5
        ** 0014' I00000: CALL    :$530
        ** 0017' L00010: L00020: L00030: L00040:
        ** 0017'         CALL    $LFMA
        ** 001A'         DW      <const>
        ** 001C'         JMP     I00001
        ** 001F' I00002:
 001F 001B       50 B=I*A(I)
        ** 001F' L00050: LXI     H,I!
        ** 0022'         CALL    $CINA
        ** 0025'         DAD     H
        ** 0026'         DAD     H
        ** 0027'         LXI     D,A!+FFFC
        ** 002A'         DAD     D
        ** 002B'         CALL    $SLHA
        ** 002E'         CALL    $FMUC
        ** 0031'         DW      I!
        ** 0033'         CALL    $FASD
        ** 0036'         DW      B!
 J038 0023       60 NEXT I
        ** 003B' L00060: CALL    $FADA
        ** 003B'         DW      I!
        ** 003D'         DW      <const>
        ** 003F' I00001:
        ** 003F'         CALL    $FASD
        ** 0042'         DW      I!
        ** 0044'         CALL    $LEJA
        ** 0047'         DW      I!
        ** 0049'         DW      <const>
        ** 004B'         DW      I00002
 004D 0023       70 FOR I=1 TO 5
        ** 004D' L00070: CALL    $LFMA
        ** 0050'         DW      <const>
        ** 0052'         JMP     I00003
        ** 0055' I00004:
 0055 0023       80 PRINT "I = ";I," A(I) = ";A(I)
        ** 0055' L000080: CALL   $PROA
        ** 0058'         LXI     H,<const>
        ** 005B'         CALL    $PVID
        ** 005E'         LXI     H,I!
        ** 0061'         CALL    $PVOA
        ** 0064'         LXI     H,<const>
        ** 0067'         CALL    $PVID
        ** 006A'         LXI     H,I!
        ** 006D'         CALL    $CINA
        ** 0070'         DAD     H
        ** 0071'         DAD     H
        ** 0072'         LXI     D,A!+FFFC
        ** 0075'         DAD     D
        ** 0076'         LXI     $LFHA
        ** 0079'         LXI     H,$AC%
        ** 007C'         CALL    $PV2A
 007F 0023       90 NEXT I
        ** 007F' L00090: CALL    $FADA
        ** 0082'         DW      I!
        ** 0084'         DW      <const>
        ** 0086' I00003:
        ** 0086'         CALL    $FASD
        ** 0089'         DW      I!
```

```
        **008B'          CALL    $LEJA
        **008E'          DW      I!
        **0090'          DW      <const>
        **0092'          DW      I00004
  0094 0023      100 END
        **0094' L00100: CALL    $END
  0097 0023
        **0097'          CALL    $END
  00B6 002D

  00000 Fatal Error(s)
  24892 Bytes Free
```

Pendant la compilation, le fichier objet est toujours généré, même si son nom n'est pas déclaré expressément dans l'appel du Compilateur. En ce cas, il porte le même nom que le fichier source. Par exemple, la commande

BASCOM = ESSAI1

déclenche la compilation et génère le fichier ESSAI1.REL, nom par lequel il devra être appelé dans la phase d'édition de liens.
Le fichier contenant le listing, par contre, n'est pas généré automatiquement ; en outre, on peut l'obtenir directement à l'écran ou sur imprimante sans demander la création d'un fichier. Dans ce cas, l'appel sera ainsi formulé :

BASCOM ESSAI2, LST: =ESSAI

La machine crée alors le fichier relogeable ESSAI2.REL et imprime le listing de la compilation au lieu d'en faire un fichier. Le code LST: est interprété comme une demande d'impression (sous CP/M, l'unité LST est l'imprimante) ; le code TTY: dirigera le listing sur l'écran. Les fichiers de provenance (Source) ou ceux de destination (Relogeable, Liste) peuvent résider sur des unités disques distinctes spécifiées au moment de l'appel. Par exemple, BASCOM B:ESSAI2, LST: =ESSAI exécute les mêmes fonctions que la commande précédente, en transférant toutefois la sortie (ESSAI2.REL) vers l'unité de disque B.
Le compilateur, tout en traduisant les instructions, détecte les erreurs de syntaxe que pourrait recéler le fichier source. Il s'avère parfois intéressant de compiler sans générer de fichier objet. En ce cas, la commande devient :

BASCOM,=Source

On obtiendra uniquement, en sortie, l'affichage de la liste des erreurs et l'espace mémoire occupé par le programme.
Pour finir, la compilation peut s'exécuter en mode « pas-à-pas » (instruction par instruction). La commande

BASCOM,TTY: =TTY:

n'utilise aucun fichier (même pas la source) et accepte les instructions à l'écran. Chaque ligne est traduite au moment de la saisie et présentée à l'écran en même temps que les diagnostics éventuels. Cette option sert, en règle générale, à la vérification formelle (syntaxique) des instructions ; elle peut s'avérer très utile à l'apprenti-programmeur, puisqu'elle jouera un rôle de guide, en signalant les erreurs, instruction par instruction.

Les fonctions particulières du Compilateur

La commande qui appelle la compilation peut se compléter par certains indicateurs (switches), qui activent à leur tour des fonctions particulières. Le même Compilateur travaillera sur différents types de Basic de manière différente ; ces indicateurs spécifient le type de Basic et la manière de traiter certaines informations particulières. Les switches se divisent en trois catégories :

- spécification de conventions ;
- fonctions de détection des erreurs ;
- codes spéciaux.

Ces indicateurs seront décrits par la suite, d'un point de vue qualitatif, car le mode d'emploi correct et les codes à adopter dépendent du Compilateur particulier qu'on utilise.

EXEMPLE DE COMPILATION (1)

Cette page présente le listing d'un programme simple contenant trois erreurs.
On va tout d'abord lancer l'exécution du programme sans le compiler, en utilisant l'interpréteur Basic (> MBASIC).



```
LIST
10 ' ** ESSAI DE COMPILATION **
20 DIM A(5),AS(6)
30 PRINT "ENTREZ LES VALEURS DU TABLEAU (A):"
40 FOR I=1 TO 5
50 PRINT "I=";
60 INPUT "VALEUR ";A(I)
70 NEXT I
80 FOR I=1 TO 6:PRINT A(I):NEXT I
90 '
100 PRINT "ENTREZ LES VALEURS DU TABLEAU (AS):"
110 FOR I=1 TO 6
120 PRINT "I=";
130 INPUT "VALEUR ";AS(I)
140 NEXT I
150 FOR I=1 TO 6:PRINT AS(I):NEXT I
160 '
170 IF A(3)=1743 GOTO 2000
180 GOTO 30
190 END
OK
RUN
ENTREZ LES VALEURS DU TABLEAU (A):
I= VALEUR ? %
```

En lignes 20, 130 et 150, on référence un tableau nommé AS. Or, ce mot est réservé au système, comme nous l'avons vu lors de l'attribution d'une mémoire tampon E/S pour la gestion d'un fichier disque (par exemple FIELD 1,20 AS BXS...) ; il ne peut donc pas s'appliquer à une variable. Dans certaines versions de Basic interprété, le système est capable de déterminer si un mot réservé est employé indûment.

La deuxième erreur est due à l'absence de l'instruction OPTION BASE 1. La numérotation des éléments des tableaux A et AS commence donc à 0 ; les tableaux étant dimensionnés respectivement à 5 et 6, il sera impossible d'affecter une valeur aux éléments A(5) et AS(6).
Dans certains cas, le Basic corrige automatiquement ce type d'erreurs, sans les signaler.

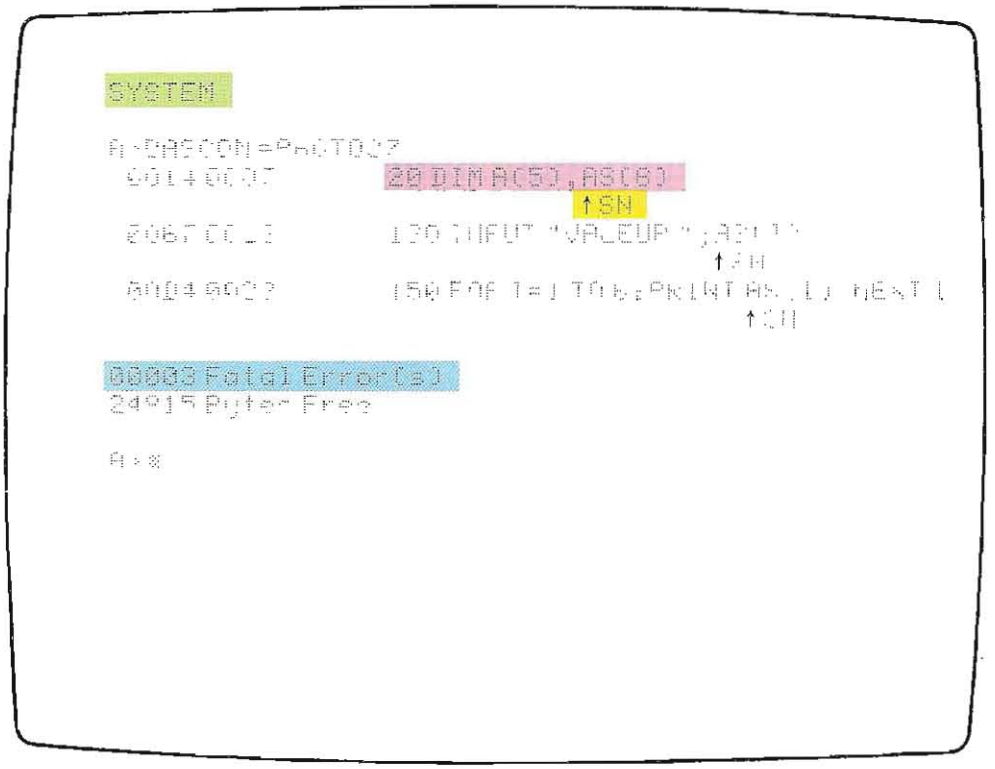
La dernière erreur réside dans l'instruction 170, qui se réfère à une ligne 2000 inexistante. L'interpréteur ne pourra détecter l'erreur qu'au moment de l'exécution de la ligne 170. De par la structure du programme, il n'y aura erreur que si l'opérateur entre A(3)=1743.

Les trois dernières lignes de l'écran montrent que le programme, bien que contenant les erreurs citées, a été lancé et que son exécution en mode interprété se déroule d'une manière apparemment normale pour l'instant.

EXEMPLE DE COMPILATION (2)

Cette page retrace l'étape de compilation du programme pris en exemple à la page précédente.





La première ligne représente la commande tapée par l'opérateur, qui demande le passage sous système d'exploitation ; la deuxième appelle le Compilateur.
La compilation, vérifiant la syntaxe des instructions, débute à partir de la première ligne (10) et se poursuit en séquence.

Le compilateur a détecté en ligne 20 un emploi incorrect du mot réservé AS ; il affiche la ligne à l'écran.

A la ligne suivante, le Compilateur désigne d'une flèche le point où l'erreur a été relevée et affiche un symbole indiquant le type d'erreur détectée : SN signifie Syntax Error (erreur de syntaxe).
Le même type d'erreur est détecté aux lignes 130 et 150. L'erreur de la ligne 170 n'a pas été diagnostiquée car elle est masquée par les erreurs précédentes. elle apparaîtra à la première compilation qui suivra la correction des erreurs détectées.

A la fin de la compilation, le nombre d'erreurs relevées s'affiche à l'écran.

Spécification des conventions. Selon la version du Basic dont on dispose, la syntaxe de certaines instructions sera différente.
On peut rencontrer des différences substancielles, en fonction de la convention sur les mots réservés.
Les noms des instructions ou des fonctions Basic sont réservés au système, et ne peuvent servir de noms de variables. Dans certaines formes de Basic, tous les mots doivent être séparés par des blancs, alors que d'autres versions ne l'imposent pas (le compilateur se charge alors de les insérer). Si on n'indique pas au Compilateur la notation en usage, des erreurs surviendront inévitablement. Par exemple, l'instruction FOR I=A TO C, en enlevant les blancs, devient FORI=ATOC ; si le Compilateur utilise une option erronée, l'instruction prendra une signification tout à fait différente que celle de début de boucle, et affectera à la variable FORI la valeur contenue dans la variable ATOC.
En parlant de ce type d'erreurs, signalons une anecdote amusante. En langage Fortran (comparable au Basic) les boucles sont définies par une instruction du genre

DO 100 I=3,10

Le code DO remplace le code FOR et I=3,10 remplace I=3 TO 10. Le nombre 100 se réfère au numéro de la ligne à laquelle la boucle se termine.
En comparant les deux formes de boucle (Fortran et Basic), on obtient :

Fortran	Basic
10 DO 100 I=3,10	10 FOR I=3 TO 10
100 CONTINUE	100 NEXT I

Le compilateur Fortran enlève tous les blancs contenus dans l'instruction ; la ligne 10 devient donc DO100I=3,10 et l'exactitude de son interprétation est liée uniquement à la présence de la virgule entre les indices (3 et 10). Dans un programme chargé de piloter le lancement d'une sonde spatiale, la virgule fut remplacée par un point dans le fichier source et l'instruction se transforma en DO100I = 3.10.
A l'exécution, au lieu d'effectuer la boucle, on affecta la valeur 3.10 à la variable DO100I : ce fut désastreux pour le lancement.*
En Basic, une erreur de ce genre serait impossible à cause de la syntaxe utilisée (présence de l'instruction NEXT qui ferme la boucle). Si le début (FOR...) est interprété de manière erronée, la compilation sera fausse à la ligne NEXT, car le FOR correspondant manquera. Par contre, en Fortran, l'instruction de fermeture de boucle n'existe pas (le code CONTINUE est fictif) ; si l'interprétation de la ligne initiale est inexacte, la boucle disparaît et aucun diagnostic n'est émis.
D'autres options du Compilateur Basic concernent les boucles, les arrondis et la numérotation des lignes.
Lors du déroulement d'une boucle paramétrée, il est possible de trouver une situation particulière, dans laquelle les deux limites coïncident ; par exemple, la boucle suivante débute et prend fin avec l'indice 3 :

```
10 A=3:B=3
20 FOR I=A TO B
```

L'incertitude quant au déroulement probable ou non de la boucle peut être levée au moyen d'une option qui force l'exécution de chaque boucle au moins une fois.
En règle générale, la même option permet aussi de demander une troncature au lieu d'un arrondi pour toutes les conversions en entier de nombres en double précision.
Une autre option très utile permet d'ignorer les numéros de ligne. Avec cette procédure particulière, les lignes peuvent être numérotées dans un ordre quelconque, ou ne pas être numérotées du tout, exception faite de celles auxquelles on se réfère explicitement dans les instructions GOTO et GOSUB.
De cette manière, un programme Basic devient très semblable à son homologue Fortran, il est plus compact et plus lisible.

Fonctions de traitement des erreurs. L'utilisation de l'instruction ON ERROR GOTO implique un système d'édition de liens du programme plus complexe et donc une forme différente de compilation.
L'option de traitement des erreurs doit être spécifiée par des indicateurs particuliers ; le

* D'après « Software Reliability », Wyley & Sons, New York 1976.

Compilateur produit alors un tableau d'adressage, contenant les numéros de ligne du source Basic. De cette manière, si une erreur apparaît pendant l'exécution du programme, le système peut retrouver le numéro de la ligne incriminée.
Dans les programmes compilés normalement, le numéro de ligne disparaît et chaque instruction est identifiée par son emplacement mémoire (adresse). Lors de la détection d'une erreur, si le système n'a pas conservé en mémoire la table de conversion adresses – numéro de ligne, l'adresse physique où s'est produit cette erreur apparaît à l'écran, en notation hexadécimale. Elle ne peut servir à rechercher la ligne source qui a engendré l'erreur, sinon au moyen de calculs fastidieux.
En phase de test des programmes (debugging), on ne saurait trop recommander l'emploi de cette spécification, même si le programme compilé nécessite un espace mémoire plus grand ; une fois la phase de test achevée, on peut de nouveau recompiler le programme sans les indicateurs d'erreurs.
La table associant à chaque numéro de ligne son adresse hexadécimale possède un point d'entrée (entry point, moyen par lequel on accède à une partie du programme ou à un tableau) pour chaque élément. Généralement, les tableaux possèdent un seul point d'entrée, l'adresse en mémoire du premier élément et les autres sont repérés en fonction de leur position par rapport au point d'entrée. Cela n'est pas possible pour les tables de conversion des numéros de ligne, et un point d'entrée pour chaque instruction du programme s'avère nécessaire. Une gestion aussi étendue des adresses mémoire alourdit considérablement le programme et augmente la taille du fichier objet (binaire) ; d'où la nécessité de l'utiliser uniquement dans la phase de mise au point.

Codes spéciaux. Les fonctions principales des indicateurs concernent le choix du type d'Assembleur, l'activation des instructions TRON et TROFF et le choix du type de compilation particulière, lors du transfert du programme sur ROM.
Les instructions TRON et TROFF, exécutées sans problème dans le cas d'un Basic interprété, sont par contre ignorées dans un programme compilé, à moins qu'on ne les ait déclarées expressément.
Un indicateur spécial permet de spécifier sous quelle forme d'Assembleur on désire le listing. L'Assembleur, exception faite de ses caractéristiques principales, est un langage fondamentalement lié au microprocesseur, et en dépend dans sa syntaxe ; ainsi, une même instruction sera traduite sous des formes différentes selon le microprocesseur.

L'édition de liens

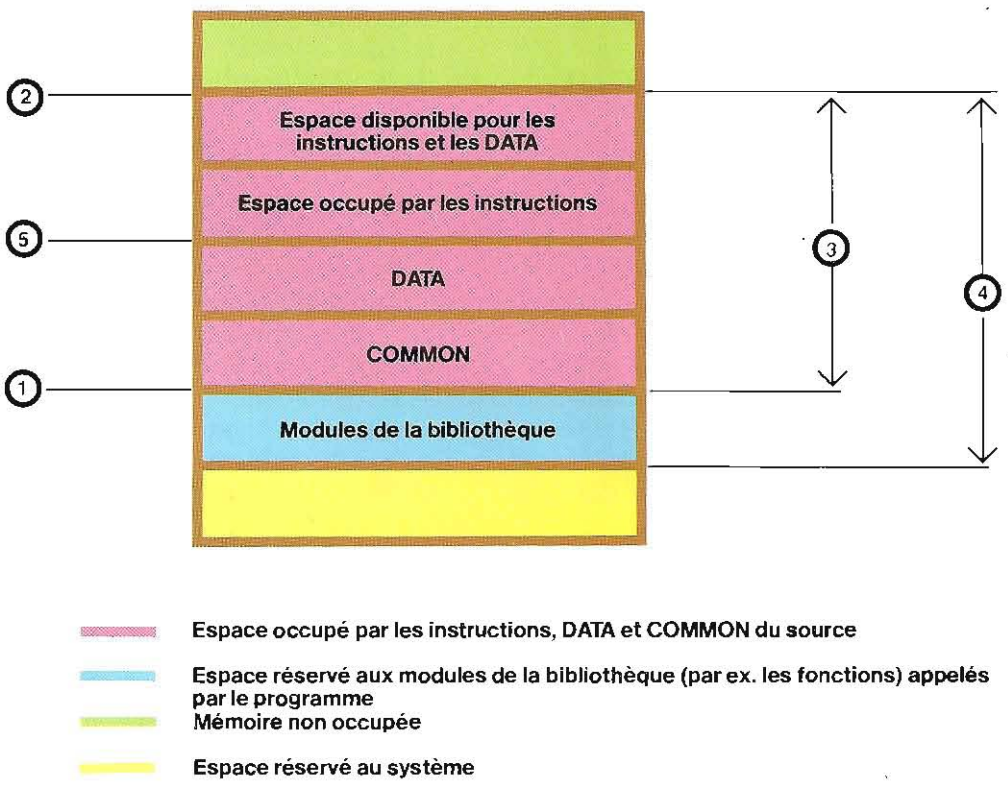
La dernière phase nécessaire à l'obtention d'un programme compilé exécutable est l'édition de liens, ou chaînage des différentes parties. Cette fonction est réalisée par un programme appelé Linking Loader, qui demande en entrée le nom du fichier relogeable (engendré en sortie du compilateur) et génère en sortie le programme exécutable.
Le Linking Loader présente à l'écran une série d'informations concernant l'implantation en mémoire, informations dont les principales sont les suivantes (le numéro se réfère au schéma de la page 732) :

- début du programme (program-start) ; adresse initiale en hexadécimal (**1**) ;
- fin du programme (program-end) ; adresse finale en hexadécimal (**2**) ;
- taille ; nombre d'octets occupés par le programme (**3**) ;
- espace restant (mémoire non utilisée) ;
- adresse de début d'exécution (start-address), exprimée en hexadécimal. On remarquera que l'adresse de début d'exécution du programme ne coïncide pas obligatoirement avec l'adresse de début du programme (**5**) ;
- nombre de pages ; nombre hexadécimal indiquant le nombre de « pages » mémoire de 256 octets utilisées par le programme (**4**).

Le schéma de la page 732 illustre la carte (map) de répartition de la mémoire. On constate que le début du programme comprend l'espace COMMON et l'espace réservé aux valeurs déterminées par l'instruction DATA, alors que l'adresse de début d'exécution coïncide avec la première instruction exécutable. L'espace COMMON est une zone de mémoire accessible à différents programmes, utilisée pour le passage de données et de paramètres de l'un à l'autre.

Des programmes différents peuvent s'appeler tour à tour, mais le chargement en mémoire de l'un recouvre l'espace occupé par l'autre ; ainsi toutes les variables sont réinitialisées, et les valeurs calculées par un programme sont perdues pour le suivant.
Afin de réaliser le transfert des données d'un programme à l'autre, il faut déclarer ces variables comme « communes » (COMMON). Elles seront stockées dans un espace mémoire séparé, qui ne sera pas réinitialisé lors des chargements successifs des programmes.
La zone réservée au système contient tous les pointeurs et autres informations nécessaires au dialogue avec le système d'exploitation.
L'espace dénommé « modules de la bibliothèque » regroupe les fonctions particulières du Basic appelées par le programme d'application. En réalité, on y stocke plus de fonctions que nécessaire, afin de gagner du temps, en évitant une longue opération de recherche et d'édition de liens.
La méthode choisie a été de partager les programmes système en deux bibliothèques : d'une part les fonctions classiques, d'autre part celles qu'on utilise rarement.
Durant la phase d'édition de liens, on charge systématiquement la bibliothèque courante (BRUN.COM), et éventuellement la seconde (BASLIB.REL), seulement si nécessaire.
Ainsi, un programme, même très petit, occupe au moins 16 Koctets de mémoire, qui représentent l'espace minimum nécessaire au module contenant les programmes système.

CARTE DE REPARTITION DE LA MEMOIRE



Test 20

1 / Ce sous-programme de gestion du disque est appelé depuis différents points du programme principal, qui lui transmet la donnée à mémoriser (BF$) et le numéro de l'enregistrement (REC).

```
110 OPEN "R",1,"A:ESSAI",128
120 FIELD 1,128 AS A$
130 LSET A$=BF$
140 PUT 1, REC
```

Quelles erreurs contient-il ?

2 / Les affirmations suivantes sont-elles vraies ou fausses ?
a) Un programme compilé occupe un espace mémoire supérieur au programme source en Basic.
b) Le Compilateur détecte des erreurs qui peuvent échapper à l'Interpréteur.
c) Un programme compilé ne peut pas être copié.

3 / Quelles opérations doit-on exécuter afin d'obtenir un programme exécutable ?

4 / Un Compilateur est-il lié au type d'ordinateur, ou bien peut-il s'utiliser sur n'importe quelle machine ?

Les solutions du test se trouvent page 737.

Instructions particulières complétant le Basic 80

Nous présentons dans ce paragraphe certaines instructions et commandes spécifiques, dont la compréhension demande une très bonne connaissance du langage Basic et de l'architecture de la machine.
Nous allons commencer par quatre instructions d'usage général et nous poursuivrons par l'étude des problèmes liés à la gestion de fonctions graphiques et de certains périphériques spéciaux.

Instructions de portée générale

Dans l'exposé des instructions du Basic, nous avons déjà introduit certaines notions concernant la segmentation, l'enchaînement des programmes et l'appel de sous-programmes. Afin de proposer un tableau complet, il est nécessaire d'évoquer trois autres instructions :

CALL
COMMON
USR

L'instruction STOP sera étudiée par la suite.

CALL. Permet d'appeler une routine écrite en langage Assembleur (sous-programme externe). La syntaxe de l'instruction est la suivante :

CALL Adresse (Arguments)

où Adresse est la variable qui contient l'adresse de départ du sous-programme à appeler, et Arguments représente une liste de variables à lui transmettre afin qu'il puisse les utiliser.
Par exemple, les instructions :

```
10 SUBA=&HA000
20 CALL SUBA(I,X)
```

appellent un sous-programme écrit en Assembleur, qui débute à l'adresse mémoire A000 (le symbole &H indique que la valeur numérique qui suit est hexadécimale), en lui transmettant les valeurs contenues dans les variables I et X. Les données à transmettre doivent avoir été affectées précédemment à

autant de variables. En règle générale, il n'est pas possible de transmettre une valeur constante comme telle : il faut d'abord l'affecter à une variable, puis transférer cette dernière. Par exemple, les instructions permettant le transfert de la valeur 157 à un sous-programme qui débute à l'adresse B010 sont les suivantes :

```
10  V=157
20  ADR=&HB010
30  CALL ADR(V)
```

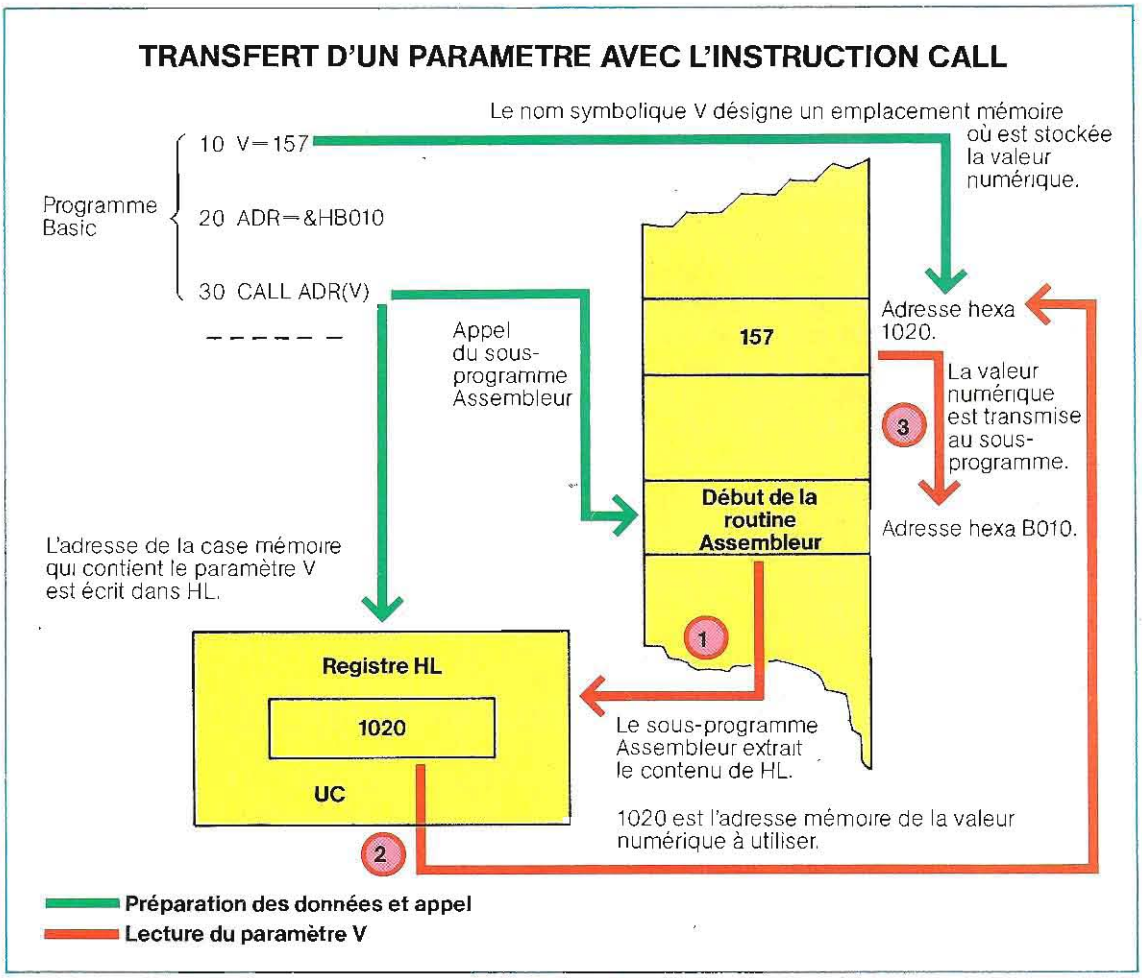
Dans d'autres langages, en particulier le Fortran, le code CALL est similaire au GOSUB du Basic : il appelle un sous-programme interne, écrit dans le même langage que le programme principal. L'instruction CALL ADR(157) donne alors le contrôle à un sous-programme nommé ADR(V), et lui transmet la valeur numérique 157 (argument) qu'il affectera à la variable V.

En Basic compilé, il n'est pas absolument nécessaire de définir numériquement l'adresse de départ du sous-programme appelé : on la désigne par un nom, et l'Editeur de liens, lors du chargement, la recherche dans une table d'adresses.

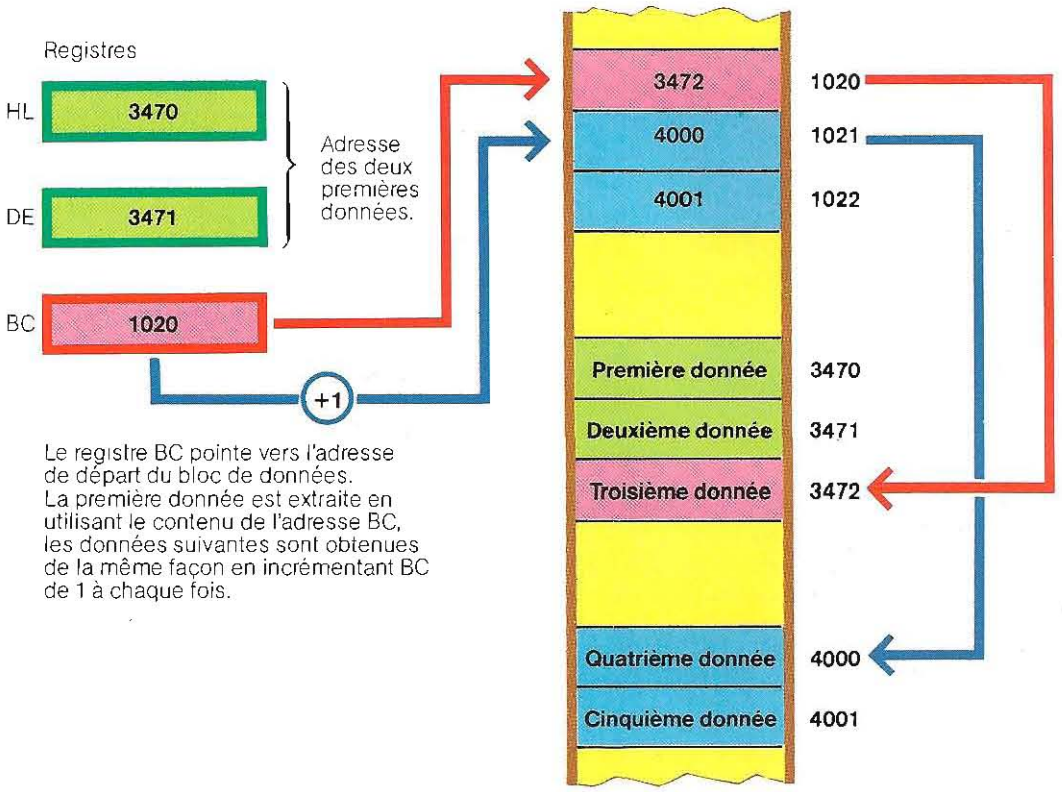
L'instruction CALL permet également d'accéder à des zones de mémoire qui ne peuvent être adressées en Basic : certains Interpréteurs et Compilateurs limitent leur champ d'action à une portion de mémoire bien définie. Dans ce cas, on peut accéder aux deux zones restantes en écrivant un ou plusieurs sous-programmes Assembleur, et en les appelant en Basic par l'instruction CALL.

La transmission des paramètres, qui se formule aisément en langage de haut niveau, présente en Assembleur certaines difficultés, dues au mécanisme complexe de transfert. Les arguments représentent en fait des

TRANSFERT D'UN PARAMETRE AVEC L'INSTRUCTION CALL



SCHEMA LOGIQUE DU TRANSFERT DES PARAMETRES



adresses en mémoire, où se trouvent les valeurs considérées. Chaque adresse est codée sur deux octets ; les paramètres occupent donc toujours deux octets, indépendamment du type de donnée (entier, réel, etc.).

Le transfert des paramètres du Basic à l'Assembleur se déroule selon des modes différents selon leur nombre.

Pour transmettre au plus trois paramètres, on se sert de registres* ; dans le cas contraire, on utilise un bloc de cases mémoire contigües. Par exemple, lors du transfert de deux paramètres, la première adresse est chargée dans HL** et la seconde dans DE (HL et DE sont les noms de deux registres).

Le programme Assembleur doit prélever le contenu de ces deux registres et les considérer comme des pointeurs (adresse de la case mémoire) vers la valeur de la variable.

L'appel CALL ADR(V) charge donc, dans le registre HL, l'adresse de la case contenant la valeur associée à la variable V ; le sous-programme Assembleur interprète le contenu de HL comme un pointeur et prélève donc la valeur numérique dans la case mémoire indiquée. La page 734 schématise ce procédé.

Le transfert d'un nombre de paramètres supérieur à trois s'avère plus complexe et appelle des sous-programmes système d'allocation de mémoire. En général, les deux premiers paramètres sont transmis à l'aide de deux registres à simple indirection, comme dans le cas précédent, et les autres données sont stockées dans un espace mémoire qui constitue le bloc de données.

Un nouveau registre (BC par exemple) contient un pointeur vers l'adresse de début du bloc (double indirection : BC contient « l'adresse de l'adresse »).

Le schéma ci-dessus détaille la logique mise en œuvre pour accéder aux valeurs.

---

* Rappelons que les registres sont des mémoires particulières implantées dans l'UC et désignées par des noms bien précis.

** HL est un couple de registres (H + L) utilisés conjointement.

COMMON. Cette instruction permet de transmettre certains paramètres d'un programme remplacé en mémoire par un autre (appelé par CHAIN). La syntaxe est la suivante :

COMMON Liste de Variables

Les variables spécifiées dans la liste sont transférées d'un programme à l'autre et non réinitialisées. Pour transmettre un tableau, il faut ajouter les symboles ( ) à la suite du nom. Par exemple, les instructions

```
10  DIM B(30), E(9)
20  COMMON V,A,B( ),E( ),K$
.....
276  CHAIN "ESSAI",10
```

transmettent au programme ESSAI les valeurs des variables numériques V et A, des tableaux B(30) et E(9), et de la chaîne K$.
En Fortran, un COMMON constitué de plusieurs variables (comme dans l'exemple précédent) peut aussi être désigné par un seul nom. En utilisant une instruction COMMON labellée (comportant un nom générique), le passage des paramètres aux sous-programmes « externes » (en Fortran ou en Assembleur) se simplifie, car il n'est plus nécessaire d'inclure dans l'instruction CALL toute la liste des variables à transférer ; il suffit simplement de citer le com « collectif », défini dans l'instruction COMMON du programme principal. La syntaxe de l'instruction est :

COMMON / NOM / V1, B( ), V3

Les variables V1, V3 et le tableau B sont résumés par le terme NOM. Tous les programmes appelés qui se réfèrent à une instruction COMMON doivent le déclarer expressément. Cette caractéristique du Fortran est offerte également par certaines versions du Basic étendu, pour le chaînage des programmes. Afin d'éviter la répétition des mêmes lignes (une pour chaque programme chaîné), une instruction Basic particulière (% INCLUDE) permet à chacun des programmes appelés d'utiliser les déclarations de l'instruction COMMON, détaillée dans un fichier séparé. De cette manière, la déclaration complète est écrite une seule fois, et chargée à chaque fois qu'elle est nécessaire.

Supposons, par exemple, que les programmes à chaîner soient MAIN, PROG1 et PROG2, mémorisés dans des fichiers distincts (de type .BAS), et utilisent en commun les variables V1, V2, B(30) et A$(6).
Les déclarations de l'instruction COMMON et le dimensionnement des tableaux sont mémorisés dans un fichier séparé, appelé par exemple COMMUN. Pour charger ce fichier dans chaque programme, on utilisera l'instruction

%INCLUDE COMMUN

Les trois programmes auront ainsi la structure suivante :

```
MAIN.BAS (Nom du premier fichier, qui appelle les autres)
10  %INCLUDE COMMUN
.....
165  CHAIN "PROG1"
.....
460  CHAIN "PROG2"

PROG1.BAS              (Premier chaînage)
10  %INCLUDE COMMUN
.....
PROG2.BAS              (Second chaînage)
10  %INCLUDE COMMUN
.....
COMMUN.BAS             (Déclarations)
20  DIM B(30), A$(6)
30  COMMON V1,V2,B( ),A$( ).
```

L'instruction de la ligne 10 provoque l'inclusion, dans chacun des programmes, du contenu du fichier COMMUN ; tout se passera donc comme si les instructions de celui-ci se répétaient dans chacun des programmes.

USR. Définit l'adresse de départ d'un sous-programme en Assembleur. La syntaxe est :

DEF USRN = VALEUR

où le nombre N (de 0 à 9) repère un des sous-programmes définis dans le programme (un numéro est ainsi associé à chaque routine Assembleur), VALEUR étant l'adresse de départ.
Le but de l'instruction USRN est de définir, à l'intérieur du programme Basic, les sous-pro-

Solutions du test 20

1 / Il y a deux erreurs. La première est fondamentale et ne permet pas l'exécution du programme ; la seconde ne se produit que sous certaines conditions.
Première erreur : l'instruction 110 (OPEN) s'exécute à chaque appel du sous-programme. La première fois, elle s'exécute correctement et a pour effet l'ouverture du fichier (ou sa création) ; la fois d'après, par contre, elle provoque une erreur système, avec arrêt de l'exécution du programme, car elle tente d'ouvrir un fichier déjà ouvert.
Seconde erreur : en phase de lecture ou d'écriture sur disque, il est bon de vérifier que le numéro d'enregistrement (REC, instruction 140) est compris entre 1 et le nombre maximal prévu pour le fichier. Avant l'appel au sous-programme, on pourrait en effet rencontrer, par erreur, l'affectation REC=0. En ce cas, l'exécution s'arrêterait car l'enregistrement 0 n'existe généralement pas (sur certaines machines, cependant, les fichiers débutent avec l'enregistrement numéro 0). D'autre part, si on affectait à la variable REC une valeur supérieure au nombre maximum d'enregistrements contenus dans le fichier, on irait écrire sur un espace du disque n'appartenant pas au fichier, entraînant ainsi la destruction d'autres données.

2 / a) faux ; b) vrai. L'affirmation c) doit s'interpréter ainsi : un programme compilé ne peut être copié en tant que source, mais peut être dupliqué sur un autre disque. Par des procédés spéciaux, on peut protéger un programme en interdisant sa copie, ou en la limitant. Il s'agit de méthodes très complexes, réservées uniquement à des programmes réellement secrets.

3 / Voici la succession des opérations à exécuter :
1 - écriture du source Basic et mémorisation sur disque en format ASCII ;
2 - lancement du Compilateur ;
3 - lancement de l'Editeur de liens.
On obtient un programme exécutable mémorisé dans un fichier disque. Pour l'exécuter (sous système d'exploitation), il suffit d'entrer son nom.

4 / Le Compilateur est étroitement lié à la machine, ou mieux, à l'UC utilisée.

grammes Assembleur comme s'ils étaient des fonctions. DEF USRN est comparable à DEF FNX : cette dernière définit des fonctions en Basic, alors qu'USRN appelle un sous-programme Assembleur.
Par exemple, l'instruction DEF USR3=1740, définit le sous-programme Assembleur qui débute à l'adresse mémoire 1740, et lui associe le numéro 3. On l'appelera ensuite comme s'il s'agissait d'une fonction utilisateur. L'instruction X=USR3(K) affecte à X le résultat du calcul exécuté par le sous-programme Assembleur 3, en utilisant le paramètre K.
L'argument (K) peut être de type quelconque (numérique ou chaîne). On indique au sous-programme Assembleur le type dont il s'agit grâce à un registre particulier (registre A), contenant un code lié au type d'argument (par exemple, la valeur 2 désigne un nombre entier, la valeur 3, une chaîne et ainsi de suite). On accède à la valeur de l'argument (valeur de K) de deux façons différentes : pour les valeurs numériques, un couple de registres indique l'adresse mémoire de la valeur ; pour les chaînes, deux autres registres contiennent l'adresse du descripteur de la chaîne.
En général, ce bloc de description est représenté par 3 octets, le premier indiquant la longueur de la chaîne en caractères (de 0 à 255), tandis que les deux autres contiennent l'adresse de début de l'espace mémoire où se trouvent les valeurs (caractères de la chaîne).

S'il s'agit d'une donnée numérique, les registres donnent directement l'adresse mémoire qui la contient et la valeur de l'argument est prélevée en un seul passage. Pour les chaînes, il faut une lecture de plus, puisque les registres contiennent l'adresse du descripteur où est mémorisée l'adresse de la donnée. En règle générale, le bloc de description des chaînes et les chaînes elles-mêmes sont stockés dans l'espace mémoire utilisateur; en revanche, les données numériques sont mémorisées dans des emplacements particuliers appelés Floating Point Accumulator (FAC), et les registres d'adressage pointent sur l'accumulateur particulier impliqué dans l'échange.
Dans certaines versions du Basic, les sous-programmes Assembleur ne peuvent être définis au moyen de l'instruction DEF USRN. Dans ce cas, des cases mémoire réservées reçoivent l'adresse de début avant utilisation de chaque sous-programme. Le nom symbolique de ces cases mémoire dépend de la version particulière du Basic. Pour appeler un sous-programme, on écrit d'abord son adresse de début, grâce à l'instruction POKE, dans la case mémoire réservée à cet effet. Ces versions du Basic ne permettent pas le choix du sous-programme par spécification de son numéro: le sous-programme dont l'adresse de début a été écrite avec l'instruction POKE sera appelé; pour exécuter un autre sous-programme, il faut d'abord opérer le transfert de son adresse de début.
A titre d'exemple, le programme listé page 739 utilise les instructions POKE et PEEK pour écrire deux symboles couleurs choisis parmi 5 (ligne 90) et relire à nouveau, pour chacun, la couleur sélectionnée. Le programme tourne sur COMMODORE VIC-20, et les adresses mentionnées ne sont valables que pour cette machine.

STOP. A pour fonction d'arrêter l'exécution du programme et de retourner en mode commande de la même manière que l'instruction END. La différence réside dans un message émis par le système, qui signale le numéro de ligne où l'instruction STOP a été rencontrée.
L'usage le plus répandu de cette instruction est de quitter l'exécution du programme en cas d'erreurs graves.
A la rencontre de l'instruction STOP, si l'exécution du programme est suspendue, les fichiers utilisés éventuellement ne sont pas refermés (contrairement à ce qui arrive avec l'instruction END); l'exécution du programme peut reprendre à partir de la ligne suivante en entrant la commande CONT (continuer).
L'arrêt momentané du programme permet d'analyser la situation qui se présente au moyen de commandes interactives, puis de reprendre l'exécution après correction éventuelle. Supposons, par exemple, qu'un programme exécute certaines divisions, en utilisant comme diviseurs les variables D1, D2 et D3, calculeés dans le programme lui-même. Il faut alors vérifier que les variables (diviseurs) ne sont pas nulles. Une erreur dans un calcul des diviseurs conduirait à d'autres erreurs dans le déroulement des instructions suivantes. On peut effectuer les contrôles nécessaires par les instructions IF et STOP, par exemple de la manière suivante:

```
IF D1=0 THEN STOP
IF D2=0 THEN STOP
IF D3=0 THEN STOP
```

A la rencontre d'une de ces erreurs, le déroulement du programme s'arrête, à l'écran apparaît le numéro de la ligne contenant le STOP exécuté, et le contrôle est rendu au système. Dans ces conditions, on peut procéder en mode interactif à l'analyse du contenu de toutes les variations qui ont participé au calcul du diviseur erroné, et relancer par la suite l'exécution du programme, afin d'examiner l'effet de cette erreur sur les calculs suivants.
De même, on peut affecter une valeur fictive au diviseur erroné avant de poursuivre l'exécution. Sous ce mode, les corrections du programme lui-même ne sont pas possibles.

Instructions de tracés graphiques

Plusieurs formes de Basic possèdent des instructions particulières pour la présentation de tracés graphiques. Le sujet sera traité en profondeur par la suite: dans ce chapitre, nous n'aborderons que l'étude des principales instructions graphiques, sans détailler leur emploi.
En mode graphique, l'écran est divisé en pixels (picture elements, éléments d'image). La « résolution » la plus répandue est de 512 pixels horizontaux et 256 pixels verticaux: à la

EXEMPLE D'UTILISATION DE PEEK ET POKE

```
0 REM: PROGRAMME ILLUSTRANT L'UTILISATION DE PEEK ET POKE
7 PRINT " ■ ": POKE 36879,29
10 DIM VCOL(5),NCOL$(5)
20 FOR I=1 TO 5
30 READ VCOL(I)
40 NEXT I
50 DATA 0,2,5,7,8
60 FOR I=1 TO 5
70 READ NCOL$(I)
80 NEXT I
90 DATA NOIR, ROUGE, VERT, JAUNE, BLEU
100 INPUT "PREMIERE COULEUR ";COL$
110 K1=0
120 FOR I=1 TO 5
130 IF COL$=NCOL$(I) THEN K1=I
140 NEXT I
150 IF K1=0 THEN PRINT "ERREUR": GOTO 100
160 N1=VCOL(K1)
170 INPUT "SECONDE COULEUR";COL$
180 K2=0
190 FOR I=1 TO 5
200 IF COL$=NCOL$(I) THEN K2=I
210 NEXT I
220 IF K2=0 THEN PRINT "ERREUR": GOTO 170
230 N2=VCOL(K2)
240 POKE 8040,83
250 POKE 38760,N1
260 POKE 8044,83
270 POKE 38764,N2
275 PRINT "CONTROLE COULEUR"
280 INPUT "COULEUR (1/2)";NC
290 IF NC=1 THEN A=PEEK(38760)
300 IF NC=2 THEN B=PEEK(38764)
310 IF A=N1 THEN PRINT "COULEUR= " NCOL(K1): GOTO 330
320 IF B=N2 THEN PRINT "COULEUR= " NCOL(K2)
330 END
```

place des 80 colonnes de caractères, il est possible d'adresser 512 points écran et, à la place des 24 lignes de texte, 256 points écran. L'instruction

HPLOT 10,5

allume sur l'écran le pixel de coordonnées (10,5). La représentation d'un segment de droite s'obtient en déplaçant le curseur du début à la fin du segment, les coordonnées des deux extrémités étant exprimées en pixels. Par exemple, les instructions

LINE (10,5)–(70,100)
HPLOT 10,5 TO 70,100

génèrent toutes deux le tracé d'un segment qui débute en coordonnées (10,5) et se termine au point (70,100).
Par une succession appropriée d'instructions de ce type, on peut dessiner les figures les plus diverses ; un rectangle, par exemple, sera tracé par quatre appels successifs.
Sur certains systèmes, une succession d'appels peut être mémorisée sur disque en format binaire, afin d'être rappelée dans son ensemble au moyen d'une seule instruction. L'instruction DRAW "n" provoque l'affichage sur l'écran du « fichier graphique » numéro n, mémorisé précédemment. Ce fichier peut contenir des figures très complexes, obtenues grâce à de nombreux déplacements élémentaires.
Presque tous les systèmes utilisant cette méthode prévoient des instructions pour le changement d'échelle, le déplacement du dessin ou la coloration de parties de l'écran. En ce qui concerne l'utilisation de ces instructions, présentes (sous des syntaxes différentes) sur presque toutes les machines, certains programmes d'application seront présentés dans le chapitre consacré au dessin sur ordinateur.

Instructions particulières

Grâce à son extrême simplicité, le langage Basic est implanté sur tous les micro-ordinateurs et ordinateurs domestiques. Sa large diffusion a incité les constructeurs à développer des interfaces spéciales, accompagnées de leurs logiciels de gestion, pour offrir l'accès aux périphériques les plus divers.
Ces périphériques, nés à l'origine avec les jeux vidéo, ont été par la suite utilisés par les professionnels, devenant ainsi un moyen de communication excellent entre l'utilisateur et la machine.

Synthétiseur. La génération de sons est un sujet vaste et complexe, qui va de la simple émission d'un « bip » jusqu'à la synthèse vocale, qui permet à l'ordinateur d'émettre des phrases dans des langues différentes. Les applications plus complexes, qui impliquent la présence d'un hardware spécialisé, seront illustrées par la suite. Dans ce chapitre, ne sont considérées que certaines instructions Basic, qui permettent d'engrendrer des sons ou de la musique.
La syntaxe des instructions varie d'un système à l'autre ; on peut toutefois distinguer deux catégories principales : les machines qui possèdent des sous-programmes système spécifiques, pour lesquelles des instructions en langage évolué existent et les machines qui demandent l'utilisation de l'instruction PEEK ou du langage Assembleur.
Cette distinction a pour origine la lenteur d'exécution des instructions en Basic interprété. On génère un son en envoyant les signaux opportuns au haut-parleur. Plus le son est aigu, et plus grande doit être la fréquence* du signal. Si on dépasse une certaine valeur, la fréquence demandée ne peut plus être réalisée, l'Interpréteur n'étant pas assez rapide. Dans ce cas, il faut mettre en œuvre des sous-programmes Assembleur (la génération de sons en Assembleur sera traitée dans le chapitre consacré à ce langage).
Si le système ne possède pas les instructions Basic nécessaires, on peut utiliser l'instruction PEEK. Beaucoup d'ordinateurs domestiques disposent d'une case mémoire spéciale, utilisée comme « interrupteur » du haut-parleur. Chaque fois qu'on accède à cette adresse, il y a émission d'un signal. Pour obtenir un son d'une hauteur déterminée (un « do » par exemple), il faut exécuter une boucle qui active périodiquement et pendant un certain temps, cette case associée au haut-parleur. En modulant correctement les temps, on engendre le son voulu, toujours dans des limites de fréquences assez basses.
On emploie la même méthode pour les fréquences plus élevées, mais avec un programme écrit en Assembleur.
Sur certains systèmes, il existe des programmes utilitaires qui peuvent être appelés par le Basic sous forme d'instruction. Par exemple, l'instruction

SOUND f,d

engendre un son de fréquence f (généralement comprise dans la bande 35 à 32 000 Hz) et de durée d (en secondes).
PLAY est une instruction plus complexe, par laquelle on spécifie différents paramètres (étroitement liés aux connaissances musicales du programmeur), qui permet de jouer de véritables morceaux de musique.

Joystick et paddle. Ces mots évoquent inévitablement les jeux vidéo ; en réalité, il s'agit de fonctions particulières gérant des interfaces grâce à des convertisseurs de déplacement, qui peuvent s'avérer utiles à de nombreuses applications.
Un convertisseur analogique/numérique est un dispositif capable de transformer une grandeur physique donnée en un signal électrique proportionnel à celle-ci.
Les convertisseurs peuvent être d'une autre nature : par exemple, le thermomètre constitue un convertisseur puisqu'il transforme les températures en longueurs (hauteur de la colonne de mercure).
L'ordinateur nécessite des convertisseurs analogique/numérique (A/N) : afin de pouvoir être interprétée par les circuits de la machine, une grandeur physique quelconque (analogique) doit être préalablement convertie en signal électrique (numérique).

* La fréquence d'un signal périodique représente le nombre d'impulsions qui se succèdent en une seconde et se mesure en Hz. Par exemple, à 35 Hz, 35 impulsions ont lieu en une seconde ; chacune a une durée de 1/35 de seconde (cette grandeur s'appelle période).

Le type le plus simple de convertisseur A/N est le **paddle** (levier), constitué par un capteur générant en sortie un signal proportionnel à un déplacement. La fonction Basic correspondante est PDL (n), qui renvoie un nombre entier, compris dans l'intervalle [0,255], et indiquant la « quantité » de déplacement effectué. Le nombre n désigne le port d'entrées/sorties auquel est relié le convertisseur. Par exemple, l'instruction

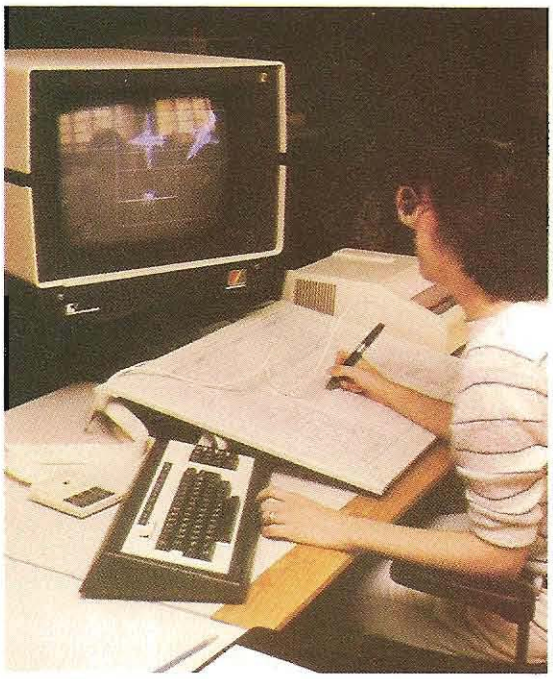
V = PDL(2)

affecte à V la valeur transmise par le paddle relié au port 2. Une fois lue, cette valeur peut être utilisée pour positionner un symbole à l'écran.
En exécutant de manière itérative (boucle) les opérations de lecture et de positionnement du curseur, on reproduit à l'écran le déplacement du levier du paddle de manière presque continue.
Le paddle permet des déplacements le long d'un axe unique, reproduits comme des déplacements du curseur le long d'une seule ligne de l'écran.
Pour un déplacement selon deux directions (sur tout le plan de l'écran), il faut utiliser le **joystick,** sorte de levier bi-directionnel, se comportant comme deux paddles, l'un disposé selon l'axe horizontal, l'autre selon l'axe vertical.
En rappelant que l'aspect formel peut différer d'une machine à l'autre, les fonctions essentielles de contrôle du joystick sont les suivantes :

STICK(n). Fournit les coordonnées de la fonction joystick, sur l'axe n. Par exemple, STICK(0) fournit le déplacement le long de l'axe horizontal, STICK(1) le long de l'axe vertical. Les coordonnées du point d'arrivée s'obtiennent donc avec les instructions

X = STICK(0)
Y = STICK(1)

STRING. Renvoie l'état du bouton (switch) généralement inclus sur le joystick (celui qui permet de « tirer » dans les jeux vidéo). En règle générale, l'état est indiqué par – 1 si le bouton est pressé et 0 en cas contraire. Sur certaines machines est prévu un paramètre [l'instruction devient STRING(n)], pour utiliser plusieurs joysticks, ou pour déterminer s'il existe des demandes en suspens (formulées avant l'exécution de l'instruction), ou encore pour désactiver ou réactiver le joystick (STRING OFF, STRING ON).



K. Reese/Marka

Emploi du stylo optique en CAO. L'opératrice construit la maquette d'un F15.

ON STRING... Cette extension de l'instruction précédente permet d'appeler un sous-programme en fonction de l'état du switch. Par exemple, ON STRING(2), GOSUB 1000 appelle le sous-programme 1000 sur pression du bouton 2.

Le stylo optique. Il s'agit d'un accessoire utilisé en liaison avec les terminaux graphiques, pour l'acquisition de points depuis une tablette graphique. Certains Basic proposent des instructions consacrées à l'activation du stylo et à l'acquisition des coordonnées. Les principales sont :

PEN ON active le stylo.
PEN OFF annule l'instruction précédente.
Z = PEN(N) acquiert les coordonnées. La valeur restituée dans la variable Z dépend de la valeur imposée au paramètre N. Avec N = 1, Z fournit la coordonnée X, avec N = 2 la coordonnée Y. Les valeurs possibles de N et les fonctions correspondantes dépendent étroitement du type de machine.

Précis du Basic 80

INSTRUCTIONS ET COMMANDES

	Instruction ou commande	Effet	Exemple
I	AUTO	Numérote automatiquement les lignes du programme	AUTO 10,5
I	CALL	Appelle un programme en Assembleur	CALL SUBA
I	CHAIN	Enchaîne un programme en lui transmettant éventuellement des variables	CHAIN "TEST"
C	CLEAR	Initialise à zéro toutes les variables	CLEAR
C	CLOAD	Charge un programme résidant sur cassette	CLOAD "PROG"
I	CLOSE	Ferme un fichier	CLOSE 2
I	COMMON	Transmet les variables entre programmes (avec CHAIN)	COMMON A(10)
C	CONT	Relance l'exécution d'un programme (après STOP)	CONT
C	CSAVE	Sauvegarde un programme sur cassette	CSAVE "PROG"
I	DATA	Définit les valeurs des constantes	DATA 3, 7, 9
I	DEF FN	Définit des fonctions utilisateur	DEF FNX=A+B
I	DEFINT	Déclare des variables entières	DEFINT K
I	DEFSNG	Déclare des variables en simple précision	DEFSNG R
I	DEFDBL	Déclare des variables en double précision	DEFDBL Z
I	DEFSTR	Déclare des chaînes de caractères	DEFSTR A
I	DEF USR	Définit l'adresse de début d'un programme en Assembleur	DEF USR2=1275
C	DELETE	Efface des lignes de programme	DELETE 5-25
I	DIM	Dimensionne des tableaux	DIM A(3), B$(10)
C	EDIT	Passe en mode éditeur pour corriger le programme	EDIT 20
I	END	Termine un programme	END
C	ERASE	Supprime les tableaux que l'on peut ainsi redéfinir	ERASE A, B$
I	ERR	Variable système pour la gestion des codes d'erreur	IF ERR=3 THEN...
I	ERL	Variable système pour signaler la ligne qui engendre l'erreur	IF ERL=127 THEN...
I	ERROR	Simule une erreur	ERROR 3
I	FIELD	Attribut une ou des mémoires tampon d'E/S à un fichier sur disque	FIELD 2,5 AS C$
I	FOR... NEXT	Boucle	FOR I=1 TO 100 NEXT I
I	GET	Lecture d'un enregistrement	GET 1,56
I	GOSUB	Appelle un sous-programme interne	GOSUB 5740
I	GOTO	Saute à la ligne spécifiée	GOTO 250
I	IF... THEN	Instruction conditionnée	IF K=1 THEN...
I	INPUT	Saisie de données au clavier	INPUT "données";A,B
I	INPUT #	Saisie de données depuis un fichier séquentiel	INPUT # 1,A

	Instruction ou commande	Effet	Exemple
C	KILL	Efface un fichier sur disque	KILL "données"
I	LET	Affecte une valeur à une variable	LET A=3+B
I	LINE INPUT	Saisie d'une ligne entière de caractères tapée au clavier	LINE INPUT A$
I	LINE INPUT #	Lecture d'une ligne de caractères depuis un fichier séquentiel	LINE INPUT # 1,A$
C	LIST	Affiche sur l'écran le programme résidant en mémoire centrale	LIST 10-50
C	LLIST	Liste le programme sur imprimante	LLIST 21-90
C	LOAD	Charge un programme à partir du disque	LOAD "A:TEST"
I	LPRINT	Imprime les variables spécifiées	LPRINT A,B,V
I	LPRINT USING	Imprime avec formatage	LPRINT USING "#.#";A
I	LSET	Transfère les données avec justification à gauche	LSET A$=C$
C	MERGE	Adjoint un programmme sur disque au programme en mémoire	MERGE "A:PROG"
C	NAME	Renomme un fichier sur disque	NAME "TEST"AS "NOUV"
C	NEW	Efface le programme contenu en mémoire centrale	NEW
C	NULL	Détermine le nombre d'espaces à écrire à la fin de chaque ligne	NULL 5
I	ON ERROR...	Permet la gestion par programme des erreurs système	ON ERROR GOTO 20
I	ON... GOTO	Exécute un saut conditionné par la valeur d'un paramètre	ON K GOTO 10, 20...
I	ON... GOSUB	Appel d'un sous-programme conditionné par un paramètre	ON L GOSUB 1000, 2000...
I	OPEN	Ouvre un fichier sur disque	OPEN "I", 2, "FISEQU" OPEN "R", 1, "FIRAND", 50
I	OPTION BASE	Définit la première valeur des indices (1 ou 0)	OPTION BASE 1
I	OUT	Transfère un octet sur le port de sortie	OUT 2, 170
I	POKE	Ecrit un octet dans la case mémoire spécifiée	POKE 2320, 170
I	PRINT	Affiche sur écran	PRINT A,B,C$
I	PRINT USING	Affiche avec formatage	PRINT USING "#.#";A
I	PRINT #	Ecrit dans un fichier séquentiel	PRINT # 3,V,C$
I	PRINT # USING	Ecrit dans un fichier séquentiel avec formatage	PRINT # 3 USING "#.#;A
I	PUT	Ecrit un enregistrement dans un fichier en accès direct	PUT 3,26
I	RANDOMIZE	Active le générateur de nombres aléatoires	RANDOMIZE
I	READ	Saisit les valeurs définies par un DATA	READ K,L,N$
I	REM	Permet l'insertion de commentaires. Sur certains Basic, il est remplacé par le symbole ', sur d'autres par ! ou *	10 REM * Comment. *
C	RENUM	Renumérote un programme résidant en mémoire	RENUM 10,20,5
I	RESTORE	Positionne le pointeur au début d'une ligne de DATA	RESTORE 1475

	Instruction ou commande	Effet	Exemple
I	RETURN	Rend le contrôle au programme appelant	RETURN
I	RESUME	Relance l'exécution d'un programme après une erreur	RESUME 5
I	RSET	Transfère les données avec justification à droite	RSET A$=B$
C	RUN	Lance l'exécution d'un programme	RUN
C	SAVE	Sauvegarde un programme sur disque	SAVE "A:TEST"
I	STOP	Suspend l'exécution d'un programme et retourne en mode commande	STOP
I	SWAP	Echange le contenu de deux variables	SWAP A$,B$
I	TRON	Active le mode trace (visualisation des instructions exécutées)	TRON
I	TROFF	Désactive TRON	TROFF
I	WAIT	Provoque une attente jusqu'à l'arrivée d'une valeur donnée sur le port spécifié	WAIT 2,240
I	WHILE... WEND	Exécute une boucle tant qu'une condition est vraie	WHILE K WEND
I	WIDTH	Indique à l'imprimante le nombre maximum de caractères par ligne	WIDTH 60
I	WRITE	Ecrit sur un terminal	WRITE A,B
I	WRITE #	Ecrit sur un fichier séquentiel	WRITE # 2,A,B,

FONCTIONS

Fonction	Description	Exemple
ABS(V)	Fournit la valeur absolue	ABS(– 3 * 5)
ASC(A$)	Fournit le code ASCII du premier caractère de la chaîne	ASC(A$)
ATN(A)	Arc-tangente de A	ATN(2)
CDBL(R)	Convertit en double précision	CDBL(37.4)
CHR$(N)	Fournit le caractère ASCII correspondant à la valeur numérique décimale N	CHR$(68)
CINT(K)	Convertit K en nombre entier pas arrondi	CINT(4.75)
COS(A)	Calcule le cosinus de l'angle A (en radians)	COS(2.1)
CSNG(D)	Convertit le nombre D en simple précision	CSNG(7.569114)
CVI(A$)	Convertit la chaîne en nombre entier	CVI(A$)
CVR(A$)	Convertit la chaîne en nombre réel	CVR(A$)
CVD(A$)	Convertit la chaîne en nombre en double précision	CVD(A$)
EOF(N)	Renvoie – 1 quand on rencontre la fin d'un fichier séquentiel	IF EOF(1)...
EXP(N)	Calcule l'exponentielle e^N	EXP(3.5)
FIX	Tronque la valeur X à sa partie entière	FIX(57.921)
FRE(N)	Fournit le nombre d'octets encore disponibles en mémoire centrale	FRE(0)
HEX$(M)	Fournit une chaîne qui est la représentation hexadécimale de M	HEX$(75)

Fonction	Description	Exemple
INKEY$	Saisie d'un caractère entré au clavier	A$=INKEY$
INP(N)	Saisie d'un octet en provenance du périphérique relié au port N	A=INP(4)
INPUT$(N)	Saisie de N caractères au clavier (sans les visualiser)	B$=INPUT$(2)
INSTR(A$,B$)	Recherche la chaîne A$ dans B$ et en fournit la position	K=INSTR(A$,B$)
INT(L)	Renvoie l'entier <=L	I=INT(87.3)
LEFT$(A$,N)	Prélève les N premiers caractères de A$	B$=LEFT$(A$,4)
LEN(A$)	Donne la longueur de la chaîne A$	K=LEN(A$)
LOC(N)	Donne le numéro du dernier enregistrement lu ou écrit dans un fichier à accès direct	N=LOC(1)
LOG(K)	Calcule le logarithme népérien de K	R=LOG(7.21)
LPOS(X)	Renvoie la position de la tête d'impression	N=LPOS(A)
MID$(I)	Extrait une partie de chaîne	E$=MID$(A$,3)
MID$(A$,I,N)	Prélève N caractères de A$ à partir de la position I	B$=MID$(A$,5,3)
MKI$(V)	Convertit l'entier V en une chaîne de deux octets	A$=MKI$(2)
MKS$(V)	Convertit V (simple précision) en une chaîne de quatre octets	A$=MKS$(71.5)
MKD$(V)	Convertit V (double précision) en une chaîne de huit octets	A$=MKD$(1521.76)
OCT$(N)	Renvoie la représentation hexadécimale de N	A$=OCT$(12)
PEEK(A)	Renvoie l'octet contenu à l'emplacement de mémoire A	PEEK(&H4701)
POS(N)	Donne la position du curseur	L=POS(1)
RIGHT$(A$,N)	Prélève N caractères de la chaîne A$ à partir de la droite	B$=RIGHT$(A$,7)
RND(K)	Génère un nombre aléatoire entre 0 et 1	R=RND(3)
SGN(A)	Renvoie −1, 0 ou 1 selon le signe de A	S=SGN(−3)
SIN(A)	Calcule le sinus de l'angle A exprimé en radians	R=SIN(3.14)
SPACE$(N)	Crée une chaîne de N espaces blancs	A$=SPACE$(3)
SPC(I)	Transmet I blancs au terminal	PRINT "A",SPC(7)
SQR(V)	Calcule la racine carrée de V	A=SQR(7)
STR$(N)	Convertit le nombre N en une chaîne	A$=STR$(3)
STRING$(N,A$)	Crée une chaîne de N caractères égaux au premier caractère de A$	B$=STRING$(5,A$)
TAB(N)	Avance le curseur à la position N	PRINT "X";TAB(6)
TAN(A)	Calcule la tangente de l'angle A exprimé en radians	Y=TAN(7)
USR(K)	Appelle un sous-programme en Assembleur en lui transmettant l'argument K	A=USR1(X)
VAL(A$)	Essaie d'interpréter A$ comme un nombre et renvoie cette valeur	N=VAL(A$)
VARPTR(A)	Donne l'adresse où est mémorisée la variable A	VARPTR(X)

Les variantes du Basic

Il existe de nombreuses versions du Basic, plus ou moins proches les unes des autres. Cependant, la plupart des instructions et fonctions du Basic 80 sont généralement disponibles malgré certaines variations de leur forme lexicale ou de leur syntaxe. Les différences les plus marquées portent sur les instructions affectant les domaines suivants :

- Gestion des chaînes de caractères
- Gestion des fichiers sur disque
- Gestion des opérations d'entrée/sortie
- Gestion des périphériques
- Transmission et réception des données.

Gestion des chaînes de caractères

Une chaîne peut contenir un nombre de caractères allant de 0 (chaîne vide) à un maximum variable selon les versions du Basic. L'espace mémoire qui lui est réservé dépend alors de ce nombre (allocation dynamique). A tout moment, la longueur d'une chaîne peut être déterminée grâce à la fonction LEN. Toutefois, certains Basic permettent de fixer, pour chaque chaîne, une longueur maximale (en caractères) inférieure à la limite initiale. Ce paramètre est déterminé soit par l'intermédiaire de l'instruction DIM, soit grâce à la fonction ALLOCATE, qui n'est prévue que sur les machines les plus puissantes. Celle-ci autorise d'ailleurs une allocation d'espace mémoire à caractère provisoire. Imaginons, par exemple, qu'un sous-programme nécessite la mémorisation de 1000 chaînes. L'emploi de l'instruction DIM A$ (1000) réserverait bien une zone de mémoire suffisante, mais de façon définitive. Ceci revient à « perdre » cet espace mémoire si les chaînes en question n'ont aucun intérêt pour la suite du programme. Il est préférable, dans ce cas, d'employer l'instruction

```
ALLOCATE A$(1000)
```

car il est possible, une fois les données utilisées, d'invalider cette instruction par la commande DEALLOCATE. Dans le cas présent, la syntaxe serait :

```
DEALLOCATE A$(1000)
```

Avec des Basic aussi complets, il est possible de limiter la longueur des chaînes à un certain nombre de caractères. Ce maximum devra être indiqué entre crochets après les instructions DIM, COM ou ALLOCATE. Voici quelques exemples :

DIM A$[5]	Attribution à la chaîne A$ d'une longueur maximale de 5 caractères
COM A$[5]	Même résultat mais en « COMMON », c'est-à-dire dans une partie de mémoire commune à tous les sous-programmes
ALLOCATE A$[5]	Résultat identique, mais l'espace réservé peut être libéré et, éventuellement, affecté à d'autres variables
DIM B$(7)[20]	Réservation d'espace mémoire pour 7 chaînes de 20 caractères chacune.

Les systèmes sur lesquels cette symbolisation est en vigueur présentent généralement deux caractéristiques très importantes :

- une longueur limite pouvant atteindre 32767 caractères pour les chaînes ;
- la mise en œuvre de sous-chaînes.

Alors qu'en Basic 80, l'extraction de parties de chaînes nécessite l'emploi des fonctions LEFT$, RIGHT$ et MID$, le recours aux crochets facilite considérablement cette tâche. Soit, par exemple, A$ = "Ceci est du Basic". Pour extraire le mot « Basic » de cette chaîne, il faudrait utiliser :

```
B$=RIGHT$(A$,5)          en Basic 80
ou
B$=A$[13]                dans les variantes
                         employant les crochets
```

Cette dernière instruction a pour résultat l'extraction de tous les caractères de A$ situés à partir de la position 13 et leur affectation à B$. La chaîne ainsi créée est une sous-chaîne. On peut préciser jusqu'à quel caractère doit s'effectuer l'extraction.
Ainsi, l'instruction :

```
B$=A$[3,8]
```

commande le transfert dans B$ du contenu

Basic 80	Variantes	Opération exécutée
B$=LEFT$(A$,3)	B$=A$[1,3]	Extraction des trois premiers caractères à partir de la gauche
B$=RIGHT$(A$,2)	—	Extraction des deux derniers caractères à partir de la droite
B$=MID$(A$,4,1)	B$=A$[4;1]	Extraction du quatrième caractère
DIM B$(3)	DIM B$(3)	Réservation d'espace mémoire pour trois chaînes
—	DIM B$(3)[20]	Réservation d'espace mémoire pour trois chaînes d'une longueur de vingt caractères chacune au maximum
—	ALLOCATE B$(3)	Réservation temporaire d'espace mémoire pour trois chaînes de caractères
Non prévue sous cette forme mais l'instruction ERASE a un effet comparable	DEALLOCATE B$(3)	Annulation de l'ALLOCATE correspondant. L'espace mémoire est à nouveau disponible

de A$ de la position 3 à la position 8. On obtient le même résultat par :

B$=A$[3;6]

Cette instruction demande, en effet, l'extraction de 6 caractères à partir de la position 3 de A$, ce qui équivaut bien à prélever les caractères situés de la troisième à la huitième position. On trouvera dans le tableau du haut de cette page une récapitulation des principales instructions de gestion des chaînes de caractères qui sont différentes en Basic 80 et dans les versions du Basic acceptant les sous-chaînes.
Les différences que nous venons de voir sont assez importantes, les autres restant purement formelles. Ainsi, dans certaines versions du Basic, la fonction ASC(A$) sera remplacée par VAL(A$), ou encore STRING$(N) deviendra VAL$(N). Il s'agit évidemment des mêmes fonctions bien qu'elles soient désignées sous une autre forme syntaxique.

Gestion des fichiers sur disque

C'est vis-à-vis des unités de disque que les instructions d'entrée/sortie varient le plus souvent d'un Basic à l'autre. Si la logique de ces accès reste relativement homogène, la syntaxe des commandes peut revêtir des formes extrêmement différentes.
A titre d'exemple, nous allons passer en revue les principales instructions du DOS 3.3 : système d'exploitation de disquettes employé aur Apple et tous les micro-ordinateurs compatibles.

CATALOG — Cette instruction provoque l'affichage de l'ensemble des fichiers présents sur la disquette. C'est l'équivalent de la commande DIR du CP/M.

LOAD, SAVE — Instructions de chargement et de sauvegarde (identiques sous CP/M).

INIT — Instruction de formattage d'une disquette. Homologue de FORMAT, elle présente toutefois une caractéristique particulière. En effet, cette commande doit être exécutée avec un programme présent en mémoire centrale. Celui-ci sera automatiquement sauvegardé sur la disquette après le formattage. Par la suite, ce programme appelé « programme de salutations » sera

chargé et exécuté à chaque mise sous tension de l'ordinateur. L'opérateur n'aura alors aucune commande à entrer : le logiciel se mettant en route directement.
La syntaxe exacte de cette instruction est :

INIT p,Vv,Ss,Dd

où p est le nom sous lequel sera sauvegardé le programme de salutations ;
v est le numéro affecté à la disquette (254 par défaut) ;
s est le numéro du port où est inséré la carte contrôleur des disquettes (6 par défaut) ;
d est le numéro du lecteur de disquettes relié à cette carte (1 par défaut).

DELETE Instruction de suppression d'un fichier (ERA en CP/M).

LOCK Instruction de verrouillage qui protège les fichiers de l'écriture ou de la destruction (annulée par UNLOCK).

RENAME Instruction servant comme en CP/M à rebaptiser un programme ou un fichier.

VERIFY Instruction servant à vérifier l'intégralité d'un fichier. Certains incidents peuvent occasionner l'altération de tout ou d'une partie d'un fichier. Ce contrôle d'une mémorisation correcte des données utilise une méthode comparable à celle du bit de parité. Pour chaque secteur de la disquette, est mémorisé un « total de contrôle » (en anglais, checksum) calculé en fonction de son contenu. Lorsqu'une donnée est modifiée accidentellement, ce total n'est pas mis à jour. Il n'y a donc plus concordance et cela sera détecté par le système lors de la vérification.

Le DOS et le CP/M distinguent les deux mêmes types de fichiers (accès direct ou séquentiel) ; cependant, ils utilisent une technique très différente pour les opérations de lecture et d'écriture.
Avec le DOS, celles-ci sont respectivement

EXEMPLE DE GESTION DES FICHIERS A ACCES SEQUENTIEL SOUS DOS

```
90 REM : Programme PROG1
100 REM : Opérations d'E/S sur disquette
120 REM : caractere de commande du DOS
140 D$=CHR$(4) : REM CTRL-D
160 REM
180 REM : *** OUVERTURE ***
200 PRINT D$;"OPEN ESSAI"
220 REM
260 INPUT " CHAINE : ";A$ : REM SAISIE AU CLAVIER
280 REM : *** ECRITURE ***
300 PRINT D$;"WRITE ESSAI" : REM PROCHAINES SORTIES VERS LA DISQUETTE
320 PRINT A$
325 PRINT D$ : REM ANNULE LA COMMANDE DOS PRECEDENTE
340 PRINT "DONNEE LUE : "; : REM AFFICHAGE A L'ECRAN
350 REM
360 REM : *** LECTURE ***
380 PRINT D$;"READ ESSAI, B0" : REM PROCHAINES ENTREES DE LA DISQUETTE
400 INPUT B$
410 REM : *** FERMETURE ***
420 PRINT D$;"CLOSE"
460 PRINT B$
480 END
```



K. Reese/Marka

La table traçante, un outil précis et rapide. Ici, les courbes de production de puits de pétrole.

commandées par READ et WRITE. Ces instructions sont envoyées sous forme de chaîne et précédées du code CTRL-D (ou CHR$(4)) qui indique que la chaîne suivante doit être interprétée comme une commande du DOS. Ainsi, une opération de lecture prendra la forme :

```
PRINT CHR$(4);"READ REPERTOIRE"
INPUT NOM$, PRENOM$, TEL$
```

La deuxième instruction indique au système que les prochaines données doivent être saisies sur la disquette. La seconde affecte les variables en conséquence.
Le programme dont le listage se trouve page 748 est destiné à illustrer les méthodes de lecture et d'écriture sur un fichier séquentiel. Il a été écrit sur un micro-ordinateur APPLE IIe, mais il fonctionnerait sur les autres modèles ainsi que sur toutes les machines compatibles Apple. Le caractère CHR$(4) (CTRL-D) doit être transmis avant chaque chaîne contenant une instruction du DOS. On a donc intérêt à l'affecter à une variable (D$: ligne 140). L'ouverture du fichier appelé ESSAI est demandée en ligne 200 par une instruction de sortie (PRINT) qui sera aiguillée vers le DOS à cause du caractère de contrôle. On indique ensuite, de la même façon, qu'on va procéder à une écriture sur la disquette (WRITE : ligne 300). Toutes les données envoyées en sortie seront alors orientées vers la disquette et non plus vers le périphérique précédemment en vigueur (écran, imprimante,...) et ce jusqu'à la prochaine commande DOS (précédée par D$). Les entrées/sorties sur disquette sont donc gérées par les commandes PRINT et INPUT classiques ; seules la destination et la provenance des informations sont modifiées au préalable.
Ainsi, une opération de lecture sera réalisée comme suit :

— Ouverture du fichier (déjà ouvert ligne 200)
— Transmission de la commande de lecture

LECTURE ET ECRITURE DE DONNEES NUMERIQUES EN ACCES SEQUENTIEL

```
10 REM : Programme PROG2
20 REM : FICHIER ACCES SEQUENTIEL
30 REM
100 REM : DONNEES NUMERIQUES
110 INPUT "Nombre de données ";N
120 DIM V(N)
140 PRINT "Entrez les valeurs"
160 FOR I=1 TO N
180 PRINT "Donnée numéro : ";I
200 INPUT V(I)
210 NEXT I
220 REM :
240 D$=CHR$(4)
260 PRINT D$;"OPEN ESSAI2"
280 PRINT D$;"WRITE ESSAI2"
300 FOR I=1 TO N
320 PRINT V(I)
340 NEXT I
360 REM :
400 PRINT D$;"OPEN ESSAI2" : REM Ferme puis réouvre le fichier
410 REM                      et réinitialise le pointeur en début de fichier
420 PRINT D$;"READ ESSAI2"
440 FOR I=1 TO N
460 INPUT V(I)
480 NEXT I
500 PRINT D$;"CLOSE ESSAI2"
520 REM === AFFICHAGE ===
540 PRINT " ** DONNEES LUES **"
560 FOR I=1 TO N
580 PRINT "Donnée N°: ";I,V(I)
600 NEXT I
620 END
```

(ligne 380 où B0 positionne le pointeur en début de fichier : inutile juste après l'OPEN)
- Lecture des données (ligne 400)
- Fermeture du fichier (ligne 420)

Une commande de lecture/écriture peut être invalidée par toute autre commande du DOS, voire même une chaîne vide (D$ seul : ligne 325). Le système est ainsi informé qu'on veut retourner aux unités d'E/S antérieures, généralement la console vidéo. Ainsi, bien que les lignes 320 et 340 soient d'une forme identique, leurs effets seront différents. En effet, la première sortie sera dirigée vers la disquette puisqu'elle intervient dans un processus d'écriture du DOS ; par contre, ce processus ayant été annulé en 325, la seconde sortie provoquera un affichage à l'écran.
En haut de cette page se trouve le listage d'un programme qui effectue d'abord l'écriture, puis la lecture de données numériques dans un fichier à accès séquentiel.
Les opérations d'écriture occupent les lignes 260 à 340. Après ouverture du fichier (ligne 260), on transmet la commande d'écriture, puis les données sont envoyées une à une par l'intermédiaire d'une boucle (lignes 300 à 340).
Le nombre N de ces données et leurs valeurs ont été préalablement saisis au clavier (lignes 110 à 210).
Le transfert donnée par donnée est caractéristique des systèmes qui travaillent sous DOS. Il est du au fait que celui-ci ne reconnaît comme séparateur que le caractère CR (retour chariot = CHR$(13)). Il est donc impératif d'insérer un CR entre chacune des valeurs numériques consécutives.
Dans notre exemple, le recours à une boucle autour de l'instruction PRINT permet d'envoyer chaque donnée séparément et suivie de CR, puisque l'on utilise ni « , » ni « ; ».
Il est aussi possible de transmettre systématiquement CHR$(13). On pourrait effectivement sauvegarder les variables X, Y et Z par l'un ou l'autre des programmes suivants :

1) D$=CHR$(4) Code de commande
CR$=CHR$(13) Retour chariot
PRINT D$;"OPEN ESSAI"
PRINT D$;"WRITE ESSAI"
PRINT D$

2) D$=CHR$(4)
PRINT D$;"OPEN ESSAI"
PRINT D$;"WRITE ESSAI"
PRINT X
PRINT Y
PRINT Z
PRINT D$

La seconde solution fait appel, comme avec l'emploi d'une boucle, à l'envoi automatique de CR à la fin d'une instruction PRINT.

Le DOS possède en outre deux commandes inexistantes sous CP/M. Il s'agit de :

- APPEND
- POSITION

L'instruction APPEND permet de compléter un fichier par adjonction d'enregistrements supplémentaires.

EXTENSION ET LECTURE D'UN FICHIER A ACCES SEQUENTIEL

```
10 REM : Programme PROG4
20 REM ' FICHIER ACCES SEQUENTIEL
30 REM : INSTRUCTION APPEND
35 HOME : REM VIDE ECRAN
100 D$=CHR$(4)
105 ONERR GOTO 1000
110 REM *** OUVERTURE ***
120 PRINT D$;"APPEND ESSAI"
160 PRINT "ENTREZ LES DONNEES"
180 PRINT "Tapez FIN pour terminer"
190 K=0
200 INPUT B$
220 IF B$="FIN" GOTO 320
240 PRINT D$;"WRITE ESSAI"
260 PRINT B$
270 PRINT D$
280 PRINT "La donnée est mémorisée"
290 K=K+1
300 GOTO 200
320 REM : SAISIE TERMINEE
340 PRINT D$;"CLOSE ESSAI"
365 IF K=0 GOTO 960
380 PRINT "IL A ETE AJOUTE : ";K;" ENREGISTREMENTS"
410 PRINT
420 PRINT "LECTURE DES DONNEES"
430 PRINT D$;"OPEN ESSAI"
680 INPUT "Combien y a-t-il d'enregistrements à lire ?";NE
700 INPUT "A partir de quelle position ?";P%
770 PRINT D$;"POSITION ESSAI,R";P%
780 PRINT D$;"READ ESSAI"
800 FOR I=0 TO NE-1
820 INPUT A$
845 IN%=I+P%
860 PRINT "Donnée n° ";IN%,A$
900 NEXT I
920 PRINT D$;"CLOSE ESSAI"
960 END
1000 REM : ** ERREUR **
1020 HTAB 10: VTAB 23: INVERSE : REM INVERSION VIDEO
1030 PRINT " ** FIN DE FICHIER ** "
1055 NORMAL : REM VIDEO NORMALE
1080 GOTO 860
```

Ceci est impossible sous CP/M, qui oblige à récrire intégralement le fichier modifié.
POSITION est une instruction permettant de placer le pointeur d'un certain nombre d'enregistrements. Ce déplacement ne peut s'effectuer que vers l'avant (déplacement relatif) et positionne le pointeur sur le premier caractère de l'enregistrement désigné. Les enregistrements sont reconnus et dénombrés grâce au séparateur (caractère CR).
Le listage reproduit sur cette page montre comment on procède à l'extension d'un fichier à accès séquentiel (APPEND) et à sa lecture, totale ou partielle, à partir d'un enregistrement donné (POSITION).
A la lecture d'un fichier dont on ignore la longueur, il peut arriver qu'on tente d'accéder à un enregistrement inexistant. Le système émet alors un message d'erreur (END OF DATA) et interrompt l'exécution. Cette fin horrible peut être évitée par l'emploi de l'instruction ONERR GOTO nn (identique sous CP/M). En cas d'erreur, le système effectuera un branchement à la ligne de numéro nn où on aura écrit une routine de traitement d'erreurs. Une telle routine peut être générale ou examiner le code d'erreur qui a été mémorisé et prévoir un traitement différent selon chaque cas.
Le programme du bas de cette page montre comment on peut utiliser l'instruction ONERR GOTO ... Suivant le type d'erreur apparu, un code est affecté à l'emplacement de mémoire 222.
L'instruction PEEK (222) fournit donc une valeur numérique correspondant à une erreur particulière que l'on peut traiter en connais-

EXEMPLE D'UTILISATION DE L'INSTRUCTION ONERR (DOS)

```
10 REM : PROGRAMME PROG5
20 REM :
30 REM : EMPLOI de ONERR GOTO
40 REM :
50 ONERR GOTO 1000
80 REM :
90 D$=CHR$(4)
100 PRINT D$;"OPEN ESSAI"
120 PRINT D$;"READ ESSAI"
140 FOR I=1 TO 100
160 INPUT A$
200 PRINT "DONNEE n° :";I,A$
240 NEXT I
260 PRINT D$;"CLOSE ESSAI"
280 PRINT "A LA FIN DU FICHIER"
300 END
1000 REM : CENTRALE DES ERREURS
1020 REM :
1040 REM : EMPLACEMENT : 22
1060 REM :
1080 REM : ERREURS PREVUES DANS CE CAS-CI
1100 REM : 5 = FIN DES DONNEES
1120 REM : 6 = FICHIER NON TROUVE
1140 REM : 8 = ERREUR D'ENTREE/SORTIE
1150 REM :
1160 T=PEEK(222)
1180 IF T<>5 GOTO 1200
1200 REM : FIN DES DONNEES
1220 PRINT D$;"CLOSE ESSAI"
1240 PRINT "Tentative de lecture après la fin du fichier"
1250 END
1260 IF T<>6 GOTO 1290
1270 PRINT "Vérifiez qu'il s'agit de la bonne disquette"
1280 GOTO 1310
1290 IF T<>8 GOTO 1330
1300 PRINT "Vérifiez que la porte du lecteur est bien fermée"
1310 INPUT "Voulez-vous continuer ?";R$
1320 IF R$="OUI" GOTO 80
1330 PRINT "ERREUR" : END
```

EMPLOI DE L'INSTRUCTION GET

```
10 REM : PROGRAMME PROG6
20 REM
30 D$=CHR$(4)
40 PRINT D$;"OPEN BIBLIO"
50 A$=""
60 PRINT D$;"READ BIBLIO"
70 GET B$
80 IF B$="," OR B$=CHR$(13) GOTO 110
90 A$=A$+B$
100 GOTO 70
110 PRINT D$
120 PRINT A$
130 PRINT "Voulez-vous continuer ?";
140 GET R$
150 IF R$="O" GOTO 50
160 PRINT D$;"CLOSE BIBLIO"
170 END
```

sance de cause. L'emplacement du code-erreur n'est évidemment pas général et, sur d'autres ordinateurs que les Apple et compatibles, il convient de le chercher dans le manuel d'utilisation.
Avec CP/M, l'instruction GET commande le transfert de la totalité d'un enregistrement. Par contre, sur les ordinateurs Apple et compatibles (qui utilisent le DOS), GET commande la saisie d'un caractère (équivalent de INKEY$). GET peut donc saisir des données sur la disquette, caractère par caractère. Ceci peut être utile si l'on désire utiliser un séparateur de donneés non reconnu par un ordre INPUT (virgule, par exemple). Le programme ci-contre vous est proposé pour illustrer l'utilisation de GET aussi bien sur un fichier (ligne 70) qu'au clavier (ligne 140).
Les instructions du DOS concernant les fichiers à accès direct sont identiques aux précédentes à un détail près : on doit spécifier un paramètre supplémentaire correspondant à la longueur d'un enregistrement pour la commande OPEN et au numéro d'enregistrement pour la lecture et l'écriture.
Soit par exemple :

```
D$=CHR$(4)
PRINT D$;"OPEN NOM,L20"
.....
PRINT D$;"WRITE NOM,R8"
.....
PRINT D$;"READ NOM,R5"
```

Ces lignes commandent les opérations suivantes :
- Ouverture du fichier NOM dont la longueur des enregistrements est fixée à 20 caractères.
- Ecriture de l'enregistrement 8 dans ce fichier.
- Lecture de l'enregistrement 5 dans ce fichier.

Vous trouverez page 754 le listage d'un programme de lecture et d'écriture dans un fichier à accès direct.
Dans les deux types de fichiers, le DOS offre une caractéristique très intéressante : la possibilité d'accéder directement à un octet donné (du fichier en accès séquentiel ou d'un enregistrement en accès direct). Il s'agit du paramètre B.
On peut ainsi extraire le second octet de l'enregistrement 7 du fichier NOM par l'instruction :

```
PRINT D$;"READ NOM,R7,B2"
```

Le schéma de la page 755 montre les niveaux de précision de l'accès suivant les commandes employées.

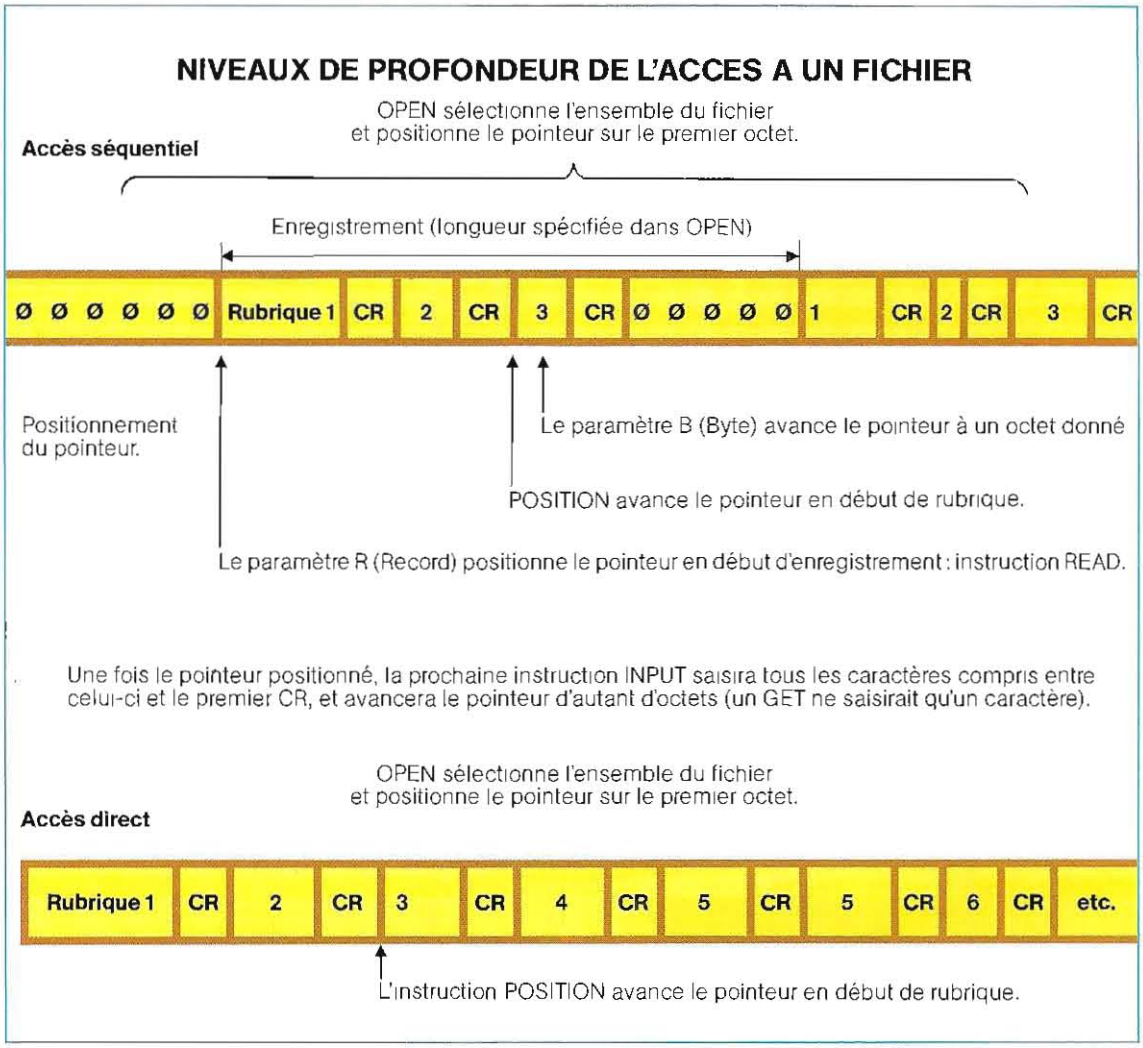
Gestion des opérations d'E/S

Les différents BASIC présentent aussi des variations importantes vis-à-vis des autres instructions d'E/S. La gestion des communications entre l'unité centrale et ses périphériques s'avère plus ou moins complexe selon les types de matériels et leur architecture interne.
En ce qui concerne la console vidéo, le mode

PROGRAMMES D'ECRITURE ET DE LECTURE EN ACCES DIRECT

```
10 REM  : PROGRAMMES PROG3
20 REM  FICHIERS ACCES DIRECT : ECRITURE
30 REM
40 D$=CHR$(4) : REM caractère contrôle du DOS
50 INPUT "Nombre de rubriques par enregistrement : ",NR
60 DIM A$(NR),A(NR),B$(NR)
70 FOR I=1 TO NR
80 PRINT "Nom rubrique ";I; : INPUT A$(I)
90 INPUT "Longueur en caractères ? ";A(I)
100 A(I)=A(I)+1 : REM longueur de la rubrique avec séparateur CR
110 LG=LG+A(I) : REM longueur totale d'un enregistrement
120 NEXT I
130 PRINT D$;"OPEN ESSAI3,L";LG
140 PRINT D$;"WRITE ESSAI3,R0"
150 PRINT LG : REM mémorisation de la longueur d'un enregistrement
                 dans l'enregistrement zéro
160 NE=0
170 PRINT D$;"OPEN ESSAI3,L";LG
180 NE=NE+1
190 PRINT "ENREGISTREMENT NUMERO ";NE
200 FOR I=1 TO NR
210 PRINT A$(I);" "; : INPUT B$(I)
220 IF LEN(B$(I))>A(I)-1 THEN B$(I)=LEFT$(B$(I),A(I)-1)
230 NEXT
240 PRINT "Est-ce correct ? "; : GET R$: PRINT R$
250 IF R$<>"O" GOTO 200
260 PRINT D$;"WRITE ESSAI3,R";NE
270 FOR I=1 TO NR: PRINT B$(I): NEXT
275 PRINT D$
280 PRINT "Etait-ce le dernier enregistrement ? "; : GET R$: PRINT R$
290 IF R$<>"O" GOTO 180
300 PRINT D$;"WRITE ESSAI3,R0": REM mémorisation en début de fichier
310 PRINT LG: PRINT NE: PRINT NR: REM des principaux paramètres
320 FOR I=1 TO NR: PRINT A$(I): NEXT : REM et des titres de rubriques
330 PRINT D$;"CLOSE ESSAI3"
340 END
```

```
10 REM : PROGRAMME PROG4
20 REM FICHIER ACCES DIRECT : LECTURE
30 REM
40 D$=CHR$(4)
50 PRINT D$;"OPEN ESSAI3": REM ouverture en accès séquentiel
60 PRINT D$;"READ ESSAI3"
70 INPUT LG: INPUT NE: INPUT NR
80 DIM A$(NR),B$(NR)
90 FOR I=1 TO NR: INPUT A$(I): NEXT
100 PRINT D$;"CLOSE ESSAI3"
110 PRINT D$;"OPEN ESSAI3,L";LG: REM ouverture en accès direct
120 INPUT "Quel enregistrement désirez-vous consulter ?";ENR
130 IF ENR>NE THEN PRINT "Enregistrement non affecté ": GOTO 120
140 PRINT D$;"READ ESSAI3,R";ENR
150 FOR I=1 TO NR
160 INPUT B$(I)
170 NEXT
180 PRINT D$
190 PRINT "ENREGISTREMENT ";ENR
200 FOR I=1 TO NR
210 PRINT A$(I);" : ";B$(I)
220 NEXT
230 PRINT "Voulez-vous continuer ? "; : GET R$: PRINT R$
240 IF R$="O" GOTO 120
250 PRINT D$;"CLOSE ESSAI3"
260 END
```



de liaison est à peu près standardisé autour de la norme RS 232.

Les mémoires de masse, par contre, (unités de disquettes ou disques durs) sont couplées au micro-ordinateur dans des conditions beaucoup moins homogènes et sont fortement conditionnées par l'aspect « matériel ». Ceci se traduit, comme nous l'avons vu, par un mode de gestion différent et des commandes spécifiques.

Les principales possibilités de liaison à des périphériques sont résumées sur le schéma de la page 756. Deux types d'interfaces peuvent être distingués en fonction du logiciel assurant leur contrôle :

- Interfaces gérées par le système d'exploitation.
- Interfaces programmables.

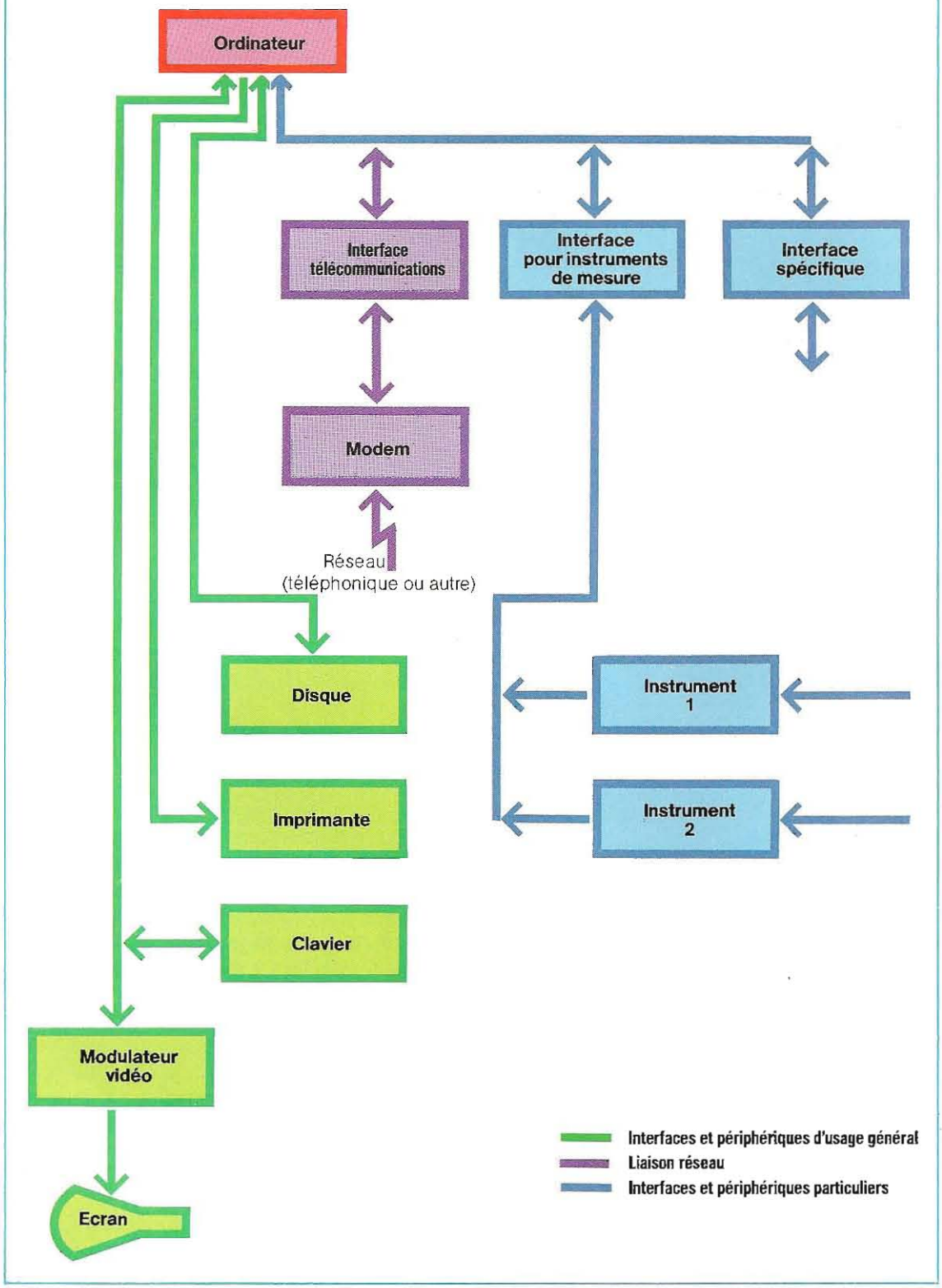
On trouve dans le premier groupe les interfaces des périphériques généralement intégrés à la configuration de base (console vidéo, imprimante, lecteur(s) de disquettes, etc.). Le langage employé comporte alors des instructions relatives à ces périphériques et leur utilisation ne nécessite pas de routines spécialisées.

Le second groupe correspond à des périphériques moins usuels pour lesquels, au contraire, n'est prévue aucune commande précise et dont la gestion implique l'écriture de routines spécialisées soit en Basic, soit en Assembleur.

Gestion des périphériques : l'interfaçage

Les ordinateurs ne sont pas aptes à commander directement toutes leurs extensions. On a

SCHEMA DE L'INTERFACE D'UN ORDINATEUR ET DE SES PERIPHERIQUES



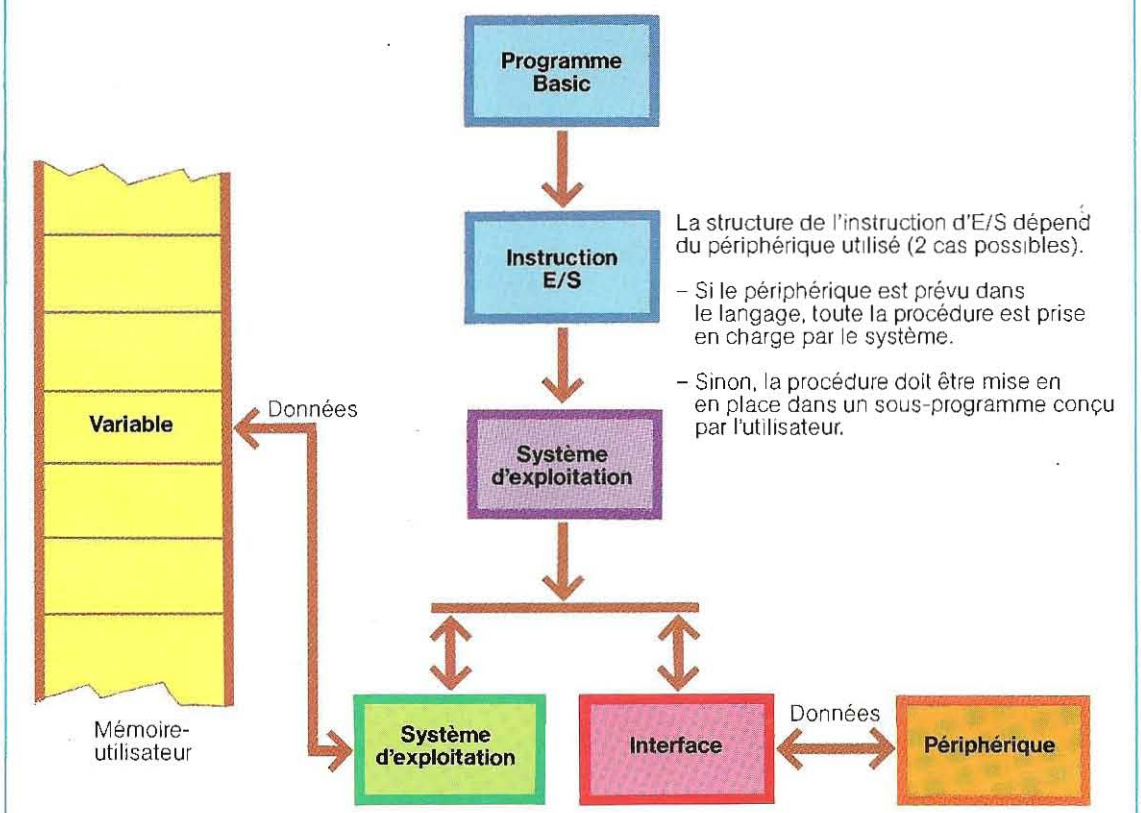
donc recours à des circuits contrôleurs couramment appelés : interfaces. Ces cartes électroniques sont branchées entre l'unité centrale et le périphérique. Elles jouent un rôle d'interprète entre ces deux unités, filtrant les signaux envoyés par l'une et/ou par l'autre, les adaptant à la logique du destinataire et synchronisant l'ensemble de ces échanges. On peut donc résumer ainsi les fonctions d'une carte d'interface :

— Adaptation du niveau des signaux.
— Mise des données au format.
— Temporisation des signaux.

Il arrive fréquemment que les signaux logiques (potentiels électriques correspondant aux états 1 et 0) d'une ligne donnée soient interprétés de façon inversée par l'ordinateur et le périphérique. C'est alors l'interface qui transforme les signaux de façon appropriée afin qu'ils puissent être reconnus par le destinataire.

De même, la transmission des signaux s'effectue en parallèle sur la plupart des micro-ordinateurs. Par contre, certains périphériques (modem, par exemple) fonctionnent avec des signaux transmis en série. C'est encore à l'interface qu'incombe la transformation parallèle-série et inversement. Enfin, les périphériques comportant souvent des pièces mécaniques en mouvement travaillent beaucoup moins vite que l'ordinateur. L'interface doit donc réguler le flot des données. Une des méthodes employée pour résoudre ce type de problèmes est la procédure dite de « handshaking ». Les données sont transmises au coup par coup : on attend un accusé de réception du destinataire pour envoyer la suite si tout s'est bien passé. Le mécanisme de l'échange des données est illustré par le schéma ci-dessous. Le programme d'application sera plus ou moins complexe selon le type de périphérique utilisé.
Si le langage disponible prévoit des instruc-

MECANISME DE L'ECHANGE DES DONNEES AVEC UN PERIPHERIQUE

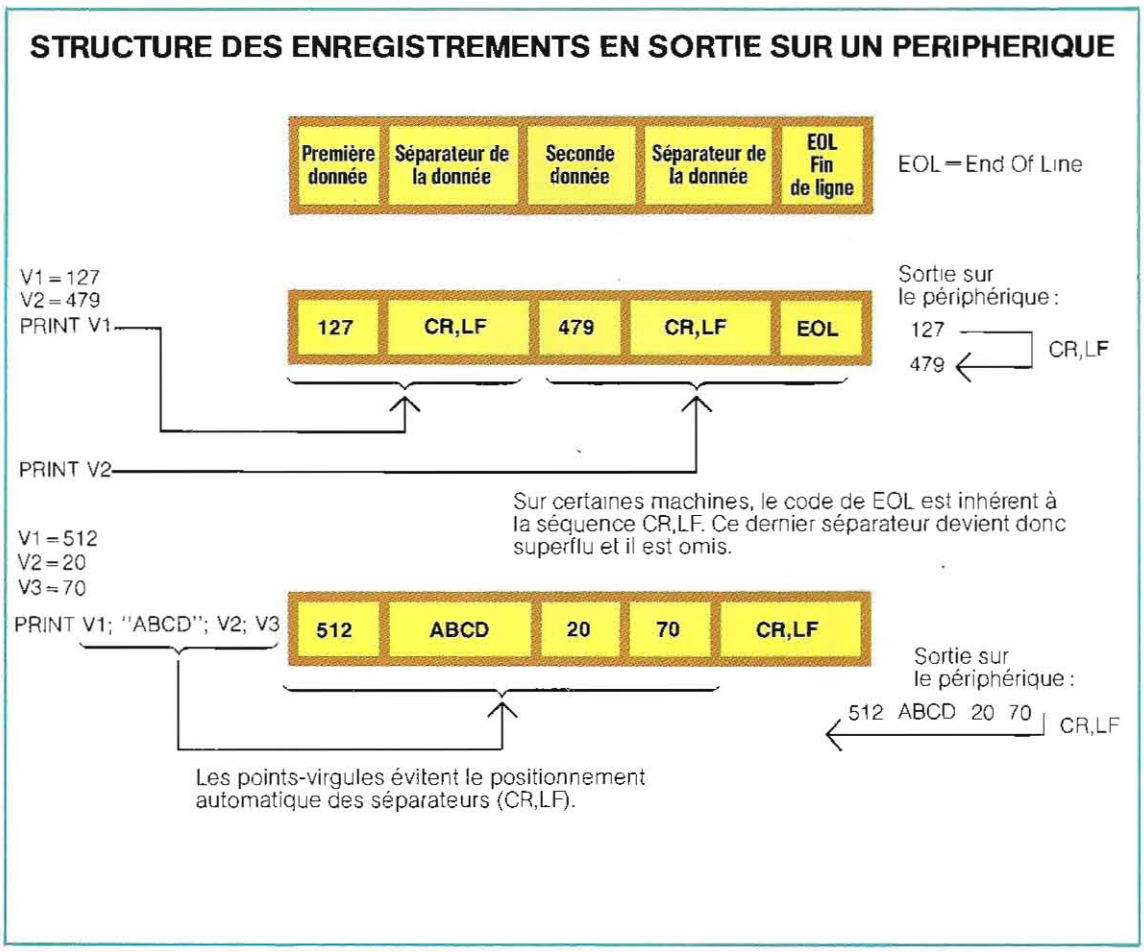


tions pour la gestion de la périphérique, tous les paramètres nécessaires à la procédure d'E/S seront mis en place par le système d'exploitation. Il suffira au programmeur d'employer la ou les instructions appropriées (INPUT, PRINT ou PRINT USING, par exemple). Leur décodage appelera automatiquement les programmes système spécifiques à ce périphérique.
Si, au contraire, le langage ne possède pas d'instructions relatives au périphérique voulu, l'écriture des modules de gestion incombera au programmeur.
La réalisation de ces routines dépendra alors essentiellement du matériel. En effet, certaines interfaces peuvent être commandées par des programmes en Basic. Dans le cas contraire, la routine devra être écrite en Assembleur. Deux exemples de transmission vers des périphériques de sortie (l'écran et l'imprimante, par exemple) sont présentés ci-dessous. L'instruction PRINT suffit à activer le mécanisme de sortie et c'est le système qui se charge d'ajouter les divers séparateurs et caractères de contrôle (par exemple, CR = Retour chariot ou LF = Ligne suivante). Est également pris en compte lors de l'exécution de la commande, un éventuel format de sortie. Le schéma du haut de cette page illustre deux emplois de l'instruction PRINT USING avec le symbole du Basic 80: #. D'autres versions du Basic offrent le choix entre plusieurs symboles, suivant le format désiré. Voici les principales options :

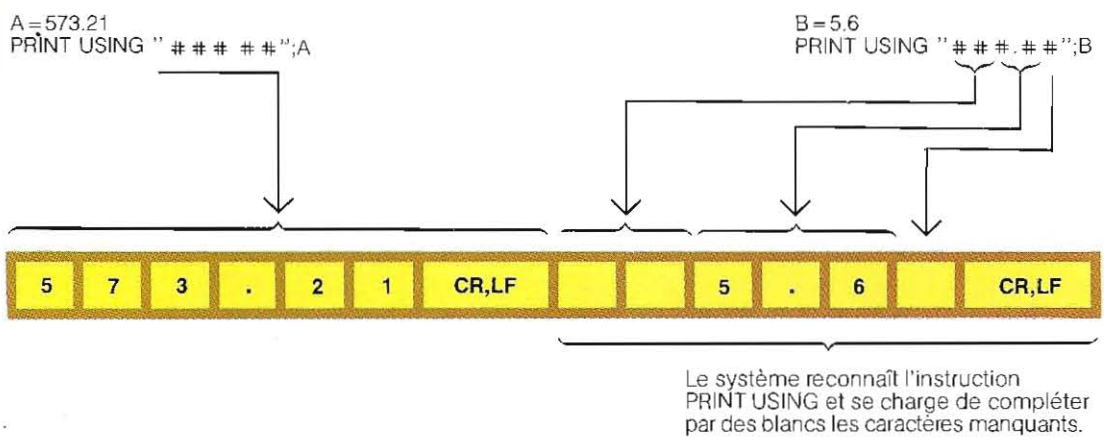
A représente un caractère alphanumérique. Ainsi, USING "4A" prévoit la sortie de 4 caractères ASCII.
D abréviation de Digit (= chiffre). Chaque D correspond à un chiffre décimal (de 0 à 9). L'expression numérique sera précédée de son signe si on mentionne l'option S. Soit :

USING "SDD.D"

STRUCTURE DES ENREGISTREMENTS EN SORTIE SUR UN PERIPHERIQUE



STRUCTURE DES ENREGISTREMENTS POUR UNE SORTIE FORMATEE



Cette instruction demande un nombre signé et composé de deux entiers et d'une décimale.

E signifie exposant. Le nombre sera représenté en notation exponentielle.

K désigne le format condensé. Tous les blancs seront supprimés.

X commande l'insertion d'un certain nombre d'espaces blancs. Soit l'instruction :

```
USING "SD.DD,5X,SDD.D"
```

Cinq espaces (5X) seront imprimés entre le premier nombre (SD.DD) et le second (SDD.D).

Nous avons volontairement omis, dans ces exemples, l'instruction de sortie dont USING doit obligatoirement être précédé. En effet, si PRINT est le plus souvent employé, d'autres commandes comme OUTPUT sont en vigueur sur certaines machines. Voici donc un exemple d'instruction complète :

```
R=-51.6
OUTPUT 3 USING "SDDD.DD";R
```

Celle-ci génère sur le périphérique N° 3 une donnée constituée d'un signe (S), de trois chiffres (DD), d'un point (.) et de deux décimales (D) :

```
-051.60
```

Nous avons ici le cas d'un ordinateur où le périphérique de sortie est désigné par son numéro. Ce numéro est celui du port (slot en anglais) dans lequel est insérée l'interface du périphérique concerné. Il peut être introduit dans chaque instruction de sortie (comme dans cet exemple) ou fixé préalablement à plusieurs sorties (ou entrées) : PR#n (ou IN#n) sur Apple. Cette notation évite le recours à des instructions spécifiques pour l'écran, l'imprimante, les disquettes, etc.

Le terme de port désigne de longs connecteurs, généralement reliés au bus de l'ordinateur, et dans lesquels peuvent être insérées non seulement des cartes d'interfaces mais aussi d'autres éléments tels que des extensions de mémoire.

Sur les ordinateurs utilisant ce système, le clavier et l'écran sont les organes d'E/S implicites. On change donc de périphérique en activant le port auquel il est relié. Très répandu sur les ordinateurs individuels, ce mode de gestion des périphériques présente plusieurs avantages et offre notamment à chacun la possibilité d'adapter la configuration du système à ses besoins. Ce système, en effet, facilite considérablement l'adjonction d'un nouveau périphérique : on le connecte à l'un des ports et il n'y aura qu'à préciser le numéro de celui-ci pour qu'il vienne supplanter l'écran et/ou le clavier.

En outre, il est possible de compléter l'installation par un second microprocesseur. Cela

est très important car les systèmes d'exploitation sont développés pour un microprocesseur donné. Ainsi, le CP/M et ses différentes versions sont prévus pour les microprocesseurs Z80, 8080, 8086,... alors que d'autres systèmes comme le DOS sont conçus pour des microprocesseurs différents.
L'adjonction d'une seconde unité de traitement permet donc de disposer de deux microprocesseurs et de leurs deux systèmes d'exploitation associés.
Outre l'impression de posséder deux ordinateurs, une telle configuration permet de choisir le système le mieux adapté pour chaque application et surtout, ouvre les portes des deux plus importantes bibliothèques de progiciels dans le cas de l'association DOS+ CP/M. A titre d'exemple, voici deux séquences d'instruction demandant une sortie sur l'écran, puis sur l'imprimante, l'une sous CP/M et l'autre sous DOS.

```
CP/M
10  A$="ESSAI"
20  PRINT A$    :' AFFICHAGE
30  LPRINT A$   :' IMPRESSION

DOS
10  A$="ESSAI"
20  PRINT A$    : REM  AFFICHAGE
30  PRINT CHR$(4);"PR#1"   : REM
                IMPRIMANTE EN LIGNE
40  PRINT A$    : REM  IMPRESSION
```

La ligne 30 sert, dans les deux cas, à diriger la sortie vers l'imprimante.
Sous DOS, CHR$(4) est le code CTRL-D précède la commande et la chaîne PR#1 est la commande proprement dite.
Cette procédure est commune aux nombreux ordinateurs bâtis autour du microprocesseur 6502 (APPLE, SIPREL, etc.) constituant aujourd'hui la plus grande famille de machines ne travaillant pas sous CP/M.
La gestion de l'écran présente également quelques particularités.
Les Basic conçus pour le CP/M ne possèdent généralement pas d'instructions de positionnement du curseur. Celui-ci est déplacé sur l'écran par l'intermédiaire de codes de contrôle dont l'ordre et la valeur dépendent de l'unité vidéo. En revanche, dans la famille des ordinateurs à microprocesseur 6502, il existe deux instructions Basic de positionnement du curseur :

HTAB n pour le positionnement horizontal
VTAB m pour le positionnement vertical

Les lignes suivantes montrent comment on peut les utiliser :

```
10  INPUT "Colonne";C
20  INPUT "Ligne";L
30  A$="Essai de positionnement"
100  HTAB C:VTAB L
110  PRINT A$
120  END
```

Contrairement à TAB(X), ces instructions s'utilisent donc en dehors d'une instruction PRINT.

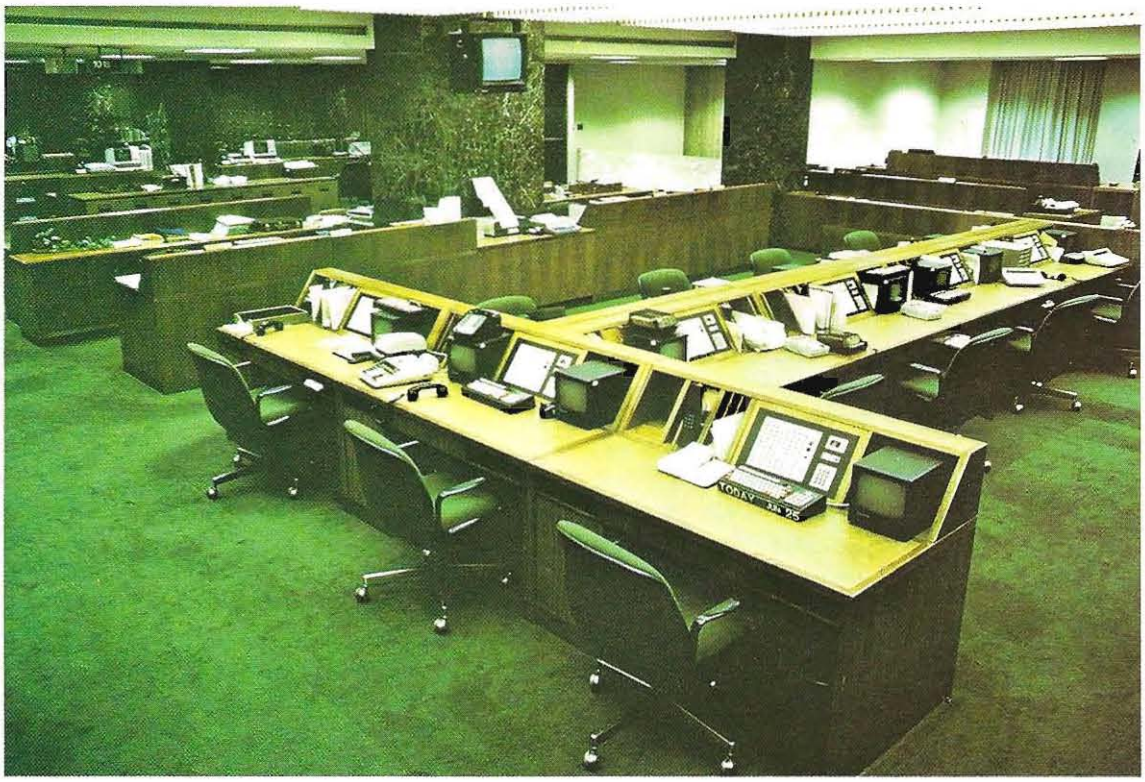
Transmission et réception des données

Le besoin croissant d'obtenir des informations en temps réel est à l'origine du développement des systèmes d'informatique répartie, c'est-à-dire de réseaux constitués de plusieurs micro-ordinateurs reliés entre eux ou à un gros ordinateur. Par rapport à de simples terminaux, les micro-ordinateurs offrent l'avantage d'une puissance de calcul indépendante qui leur permet d'assurer à la fois un rôle de terminal et d'unité de traitement autonome.
Le protocole de transmission des données le plus répandu est l'EIA RS-232-C. C'est ainsi que certains constructeurs équipent leurs machines d'interfaces répondant à ce standard et d'instructions Basic permettant de contrôler le protocole de transmission. L'instruction la plus importante est celle qui fournit les paramètres correspondant aux modalités de cette liaison, soit :

– **La vitesse de transmission**, exprimée en bauds (bits par seconde). Les différentes vitesses pratiquées sont : 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 bps.

– **La parité**. Elle indique le type du contrôle de parité, c'est-à-dire la façon dont est traité le bit de parité. Il y a cinq possibilités :

— non transmis
— toujours 0



K. Reese/Marka

Dans les agences de courtage, informatisation est synonyme de rapidité, et donc d'efficacité.

— toujours 1
— transmis pour la parité impaire
— transmis pour la parité paire

– **Bits de données**. Les données peuvent être symbolisées par un nombre de bits compris entre 4 et 8. Il faut donc configurer l'interface pour le format choisi.

– **Bits d'arrêt**. Des bits d'arrêt peuvent être ou non positionnés à la suite de chaque donnée. Il y a trois possibilités :

— pas de bit d'arrêt
— un seul bit d'arrêt
— deux bits d'arrêt

– **Numéro du fichier**. Pour utiliser un canal de télécommunications, on doit lui allouer un numéro de fichier de façon à pouvoir ensuite employer les instructions GET et PUT puisqu'elles agissent sur des fichiers.
Considérons l'instruction suivante :

```
OPEN "COM 1:2400,S,8,1"AS#3
```

Elle commande l'ouverture d'un fichier de télécommunication portant le numéro 3. Les sorties s'effectueront par l'interface 1 (COM 1) à une vitesse de 2400 bps, selon un format de 8 bits plus 1 bit d'arrêt et sans bit de parité (symbole S).
On peut également indiquer dans cette instruction les codes de contrôle des signaux de ligne. D'utilisation très simple, le RS-232-C est le protocole le plus adapté à la liaison de deux unités de calcul. En revanche, dès que ce nombre augmente, le RS-232-C nécessite la multiplication des ports d'accès ou un système de sélection extérieur. Une telle installation devient vite très complexe et remet en question le choix du protocole d'échange.
Or, des systèmes comme le contrôle automatique d'appareils de mesure requièrent l'échange simultané de données entre une unité de traitement et un certain nombre de périphériques. Ces applications ont donc suscité la conception et la mise au point de protocoles spéciaux dont le plus connu, l'HP-IB (Hewlett-Packard Interface Bus), a été proposé à l'IEC (International Electrotechnical Commission) comme standard international.

Le bus HP-IB comporte 16 câbles dont huit sont réservés aux données et les autres à des signaux de contrôle. Les données sont transmises en ASCII sur 7 bits (le huitième bit étant le bit de parité). Les appareils et, d'une manière générale, les unités à connecter sont répartis en trois catégories :

Les RECEPTEURS (listener) sont des unités capables comme l'écran ou l'imprimante de recevoir des données provenant d'autres unités.

Les EMETTEURS (talker) envoient, au contraire, des données vers l'extérieur. Ce sont des enregistreurs, ou tout appareil effectuant des mesures et les convertissant en signaux électriques (numériques) avant de les acheminer vers le bus.

Classement des listages et des états dans la salle des imprimantes. Des sigles permettent d'identifier l'utilisateur.



J. Pickerell/Marka

Les CONTROLEURS (controller) sont des unités qui gèrent et contrôlent les échanges entre les autres unités. Ce sont généralement des ordinateurs.

Certaines unités peuvent appartenir successivement à l'une ou l'autre de ces catégories : un instrument de mesure est RECEPTEUR lorsqu'il reçoit les instructions de l'ordinateur et EMETTEUR lorsqu'il communique les résultats de ses mesures. L'HP-IB est conçu pour qu'il n'y ait, à un instant donné, qu'un EMETTEUR et qu'un CONTROLEUR qui fonctionnent en tant que tels. Les autres appareils ne peuvent, à cet instant, jouer que le rôle de RECEPTEURS.

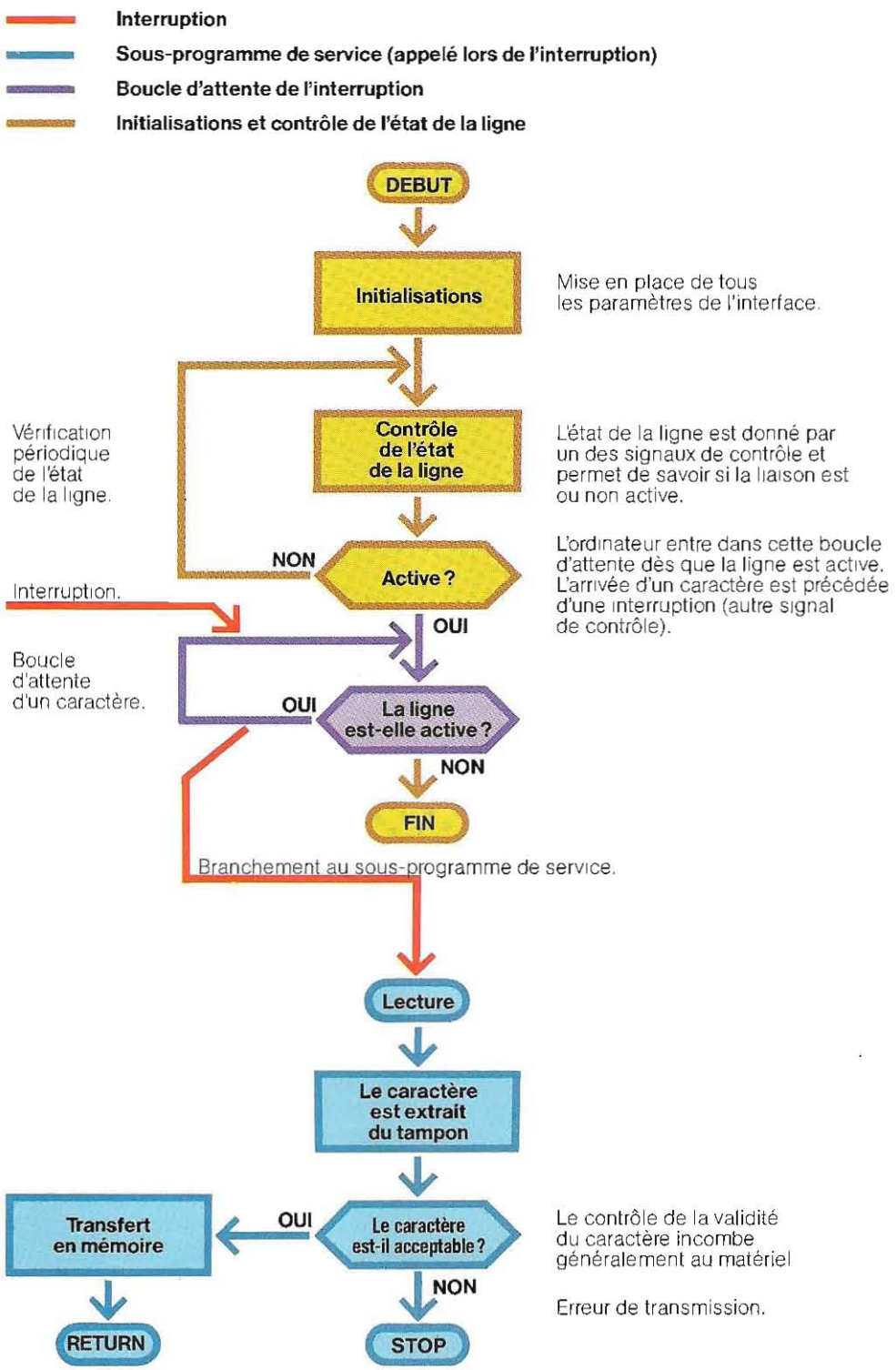
Certaines versions du Basic possèdent des instructions de piégeage des données acheminées par le bus. L'arrivée de ces données provoque l'interruption de l'opération en cours et l'appel d'un sous-programme de service conçu pour leur acquisition. Les mécanismes commandés par ces instructions permettent à l'ordinateur de poursuivre sa tâche entre la réception de deux caractères consécutifs, ne s'interrompant que pour la saisie de ceux-ci. L'organigramme de la page 763 illustre le principe de la réception des données transmises par le bus.

Déclenchée par un signal d'origine externe, l'interruption (en anglais, trap ou interrupt) est une opération primordiale pour l'acquisition des données car c'est elle qui déclenche l'appel des sous-programmes appropriés. Prenons l'exemple de la commande RESET qui est activée en CP/M par les codes CTRL-C et interrompt l'exécution de n'importe quel programme. Pendant le déroulement d'une application, l'ordinateur n'interroge cependant pas le clavier pour y acquérir d'éventuelles commandes. Ce type d'entrée n'intervient donc pas comme un caractère normal mais agit directement sur la ligne des interruptions. Le traitement en cours est alors suspendu et on peut éventuellement effectuer un branchement sur une opération particulière.

Sur certaines machines, on dispose de touches (key) ou de molettes (knob) pour provoquer une interruption.

Ces touches d'interruption sont des touches de fonction qui engendrent des interruptions

EXEMPLE DE PROGRAMME DE TELECOMMUNICATION



diverses donnant chacune lieu à l'exécution d'un sous-programme déterminé. Il existe donc dans le Basic de ces machines des instructions permettant au programmeur de définir les opérations associées à la frappe de chaque touche. Ainsi, l'instruction :

ON KEY 3 GOTO 100

détermine un branchement à la ligne 100 après l'interruption provoquée par la frappe de la troisième touche spéciale. Quant aux molettes, elles ont généralement une fonction de déplacement du curseur. Il s'agit d'un dispositif dont la rotation génère des impulsions. L'ordinateur calcule l'angle de rotation en fonction du nombre d'impulsions reçues et déplace le curseur en conséquence. Il existe également en Basic une instruction (ON KNOB) qui permet de sauter à l'endroit voulu d'un programme par la manipulation d'une molette au lieu d'une touche. La gestion des interruptions n'est toutefois pas aussi simple qu'on pourrait le croire, en raison notamment de la plus ou moins grande priorité des périphériques.

Nombreux sont les périphériques qui peuvent demander eux-mêmes une interruption. Aussi est-il tout à fait possible que deux demandes parviennent presque simultanément à l'ordinateur ou encore qu'une demande soit transmise en cours d'exécution d'une routine, elle-même appelée par une autre interruption. C'est pourquoi une priorité est accordée à chaque périphérique afin de permettre au système de décider s'il doit accepter l'interruption ou mettre la demande sur une « liste d'attente ». Le périphérique ayant la priorité la plus élevée provoquera une interruption immédiate alors que les autres demandes seront retardées, voire ignorées.

La priorité absolue d'une opération est déterminée soit par le matériel, soit par le logiciel. Ce dernier cas est réservé au traitement d'un incident ou d'une erreur et on le programme grâce à des instructions spéciales en suivant la logique exposée ci-dessous.

Si aucune interruption n'est en cours et que le programme s'exécute normalement, le degré de priorité est fixé à 0 (on appelle ce degré « priorité système » car il est attribué automatiquement par le système). Toute interruption pourra donc être prise en compte immédiatement – ou, plus exactement, au terme de l'instruction en cours d'exécution – puisque, quel que soit son degré de priorité, il ne pourra, en aucun cas, être inférieur à zéro.

Si une nouvelle demande d'interruption survient pendant l'exécution de la routine appelée par la première, deux cas peuvent se présenter :

1 / soit la nouvelle interruption a une priorité inférieure à la précédente et elle est mise en attente jusqu'au terme de celle-ci ;

2 / soit son degré de priorité est supérieur ou égal à celui de l'interruption en cours. Elle est alors prise en compte immédiatement. Le programme appelé par la première s'interrompt et ne reprendra qu'ensuite.

La priorité logicielle détermine l'ordre de prise en compte des interruptions dont la demande a été reçue, alors que la priorité « hardware » détermine l'ordre de réception des demandes. En d'autres termes, elle sert à planifier les demandes d'interruption en provenance des périphériques. Ces deux priorités sont indépendantes.

Lorsque l'ordinateur reçoit une demande d'interruption, il abandonne la tâche en cours d'exécution et recherche l'interface d'où émane cette demande.

Puis, il met cette demande sur une liste d'attente, négligeant désormais les autres demandes de cette interface. Après avoir exécuté toutes les opérations prioritaires, il examine à nouveau les demandes non satisfaites.

La gestion des interruptions est capitale dans les transmissions et ce, autant en ce qui concerne l'acquisition des données que pour le contrôle des durées d'exécution. En effet, le matériel n'est jamais totalement à l'abri d'un incident pendant la transmission des données. Une détérioration de la ligne ou, plus fréquemment, une micro-coupure de l'alimentation peuvent être à l'origine d'un défaut de transmission. Il est donc indispensable de prévoir un contrôle des temps de transmission pour éviter que l'ordinateur attende indéfiniment une donnée. Il existe donc une interruption, conditionnée par la durée de l'attente, qui provoque une sortie automatique de la boucle d'attente. Un message de diagnostic est émis et l'ordinateur passe à la tâche suivante.

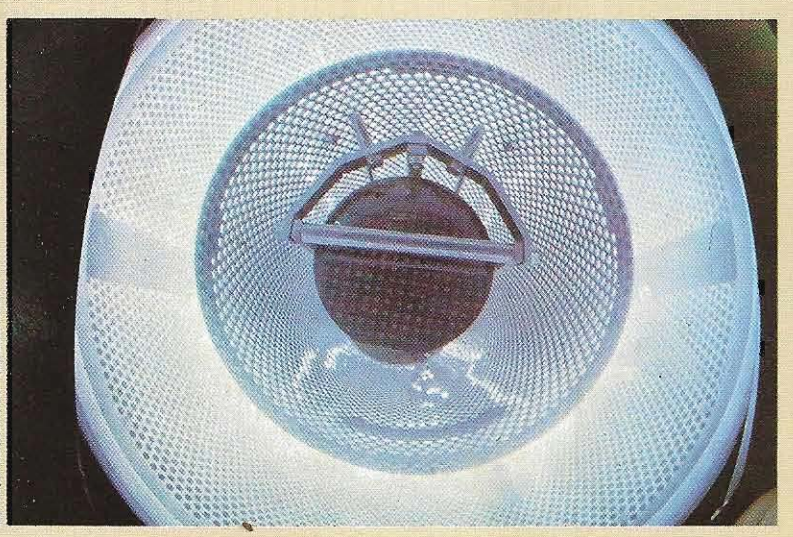
Du silicium à l'ordinateur



M. Wolf/Black Star-Grazia Neri



Stock Photos/Grazia Neri



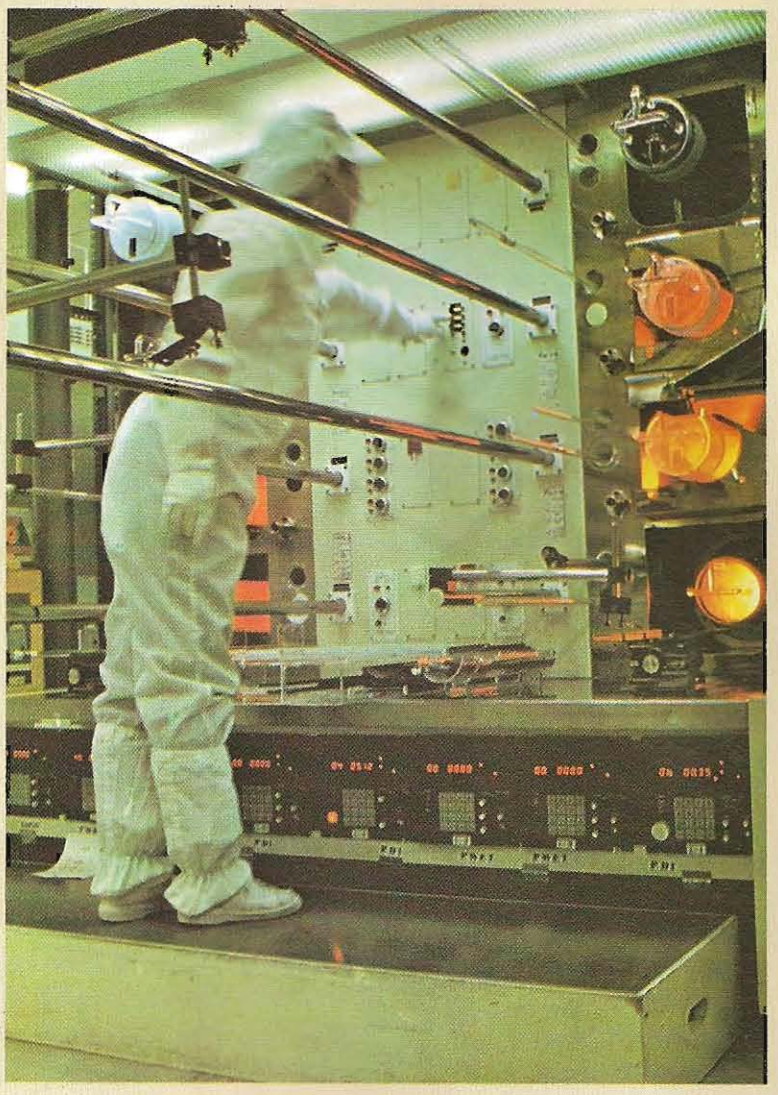
M. Wolf/Black Star-Grazia Neri

Les ingénieurs font appel à toutes les ressources de la C.A.O. (Conception Assistée par Ordinateur) pour développer les nouveaux circuits intégrés.

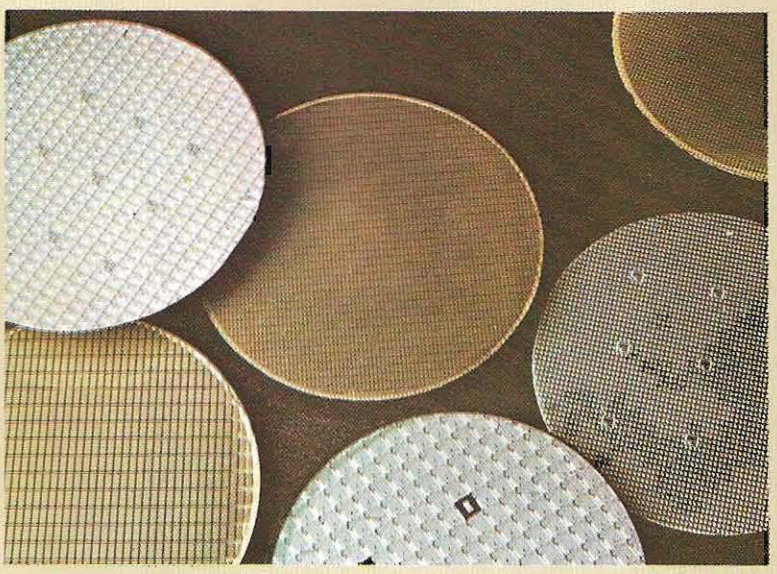
La photographie du haut a été prise pendant le dessin des motifs d'un circuit à l'aide d'un traceur à haute résolution. Le concepteur peut obtenir la visualisation sur un écran de la structure du circuit mémorisée par l'ordinateur, effectuer des modifications au clavier jusqu'à la mise au point définitive et, enfin, réaliser les masques nécessaires au procédé de photolithographie.

On additionne des impuretés au silicium très pur en fusion afin de lui conférer les propriétés électriques désirées. Les atomes de silicium viennent ensuite se ranger autour d'un minuscule noyau, donnant naissance à un monocristal presque parfait de silicium semi-conducteur. On extrait alors lentement (seconde photographie) le barreau de silicium auquel on donne un diamètre régulier avant de le découper transversalement en fins disques circulaires, les «tranches» (wafers, en anglais). Après avoir soigneusement poli ces tranches, on y imprime les différentes couches du circuit par photolithographie.

La photographie du bas illustre le bombardement des zones conductrices par de l'oxygène fortement ionisé.



M. Wolf/Black Star-Grazia Neri

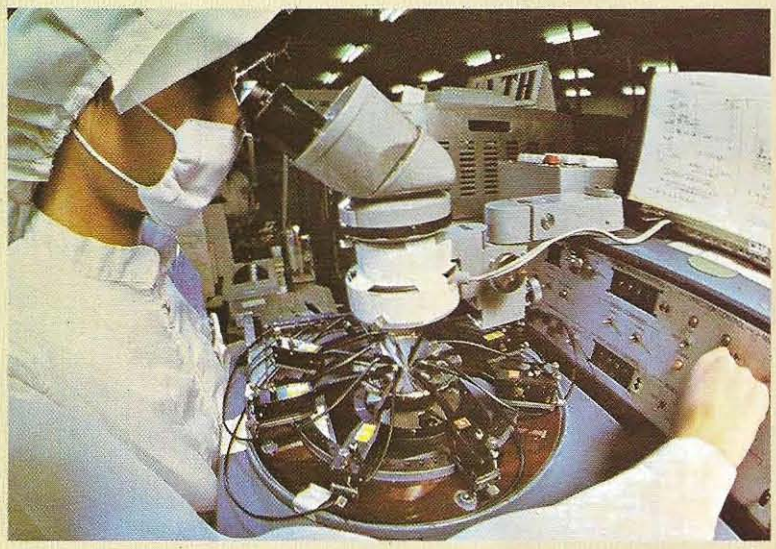


B. Coleman/Marka

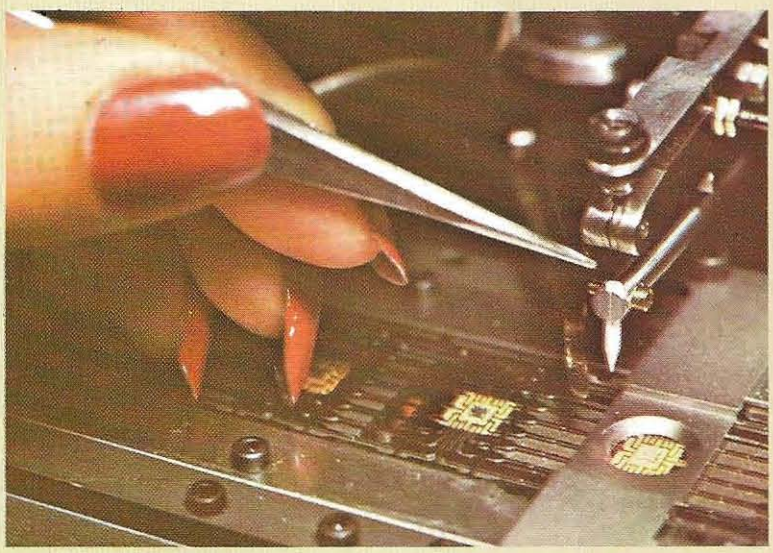
Les tranches de silicium doivent être traitées dans une atmosphère parfaitement contrôlée. Aussi les scelle-t-on avec des impuretés dans des ampoules de quartz qui sont ensuite placées dans des fours à haute température (photographie du haut) où les atomes des impuretés parvenues à l'état gazeux diffusent dans le silicium à travers les «fenêtres» créées par photolithographie : c'est le dopage.
On obtient ainsi les transistors, les résistances et les diodes constitutifs du circuit intégré.
A l'issue de la gravure, les tranches se présentent comme sur la photo du bas. On voit d'ailleurs nettement qu'en raison de la forme circulaire des tranches, les circuits qui se trouvent sur leur pourtour sont incomplets. Inutilisables, ils seront mis au rebut lors du découpage. Les puces coûtent d'autant moins cher que le circuit est plus petit, car on peut en imprimer davantage sur chaque plaquette. Avant le découpage, chaque tranche est examinée au microscope et les circuits défectueux sont marqués au vernis coloré.
Un mauvais fonctionnement des puces peut avoir des origines très différentes. Du fait de l'extrême miniaturisation, le moindre grain de poussière, invisible à l'œil nu, peut couper une connexion ou même plusieurs.
C'est pourquoi on apporte un soin rigoureux à toutes les étapes de la fabrication. On prend le maximum de précautions pour éviter la contamination des circuits par des corps étrangers. Une équipe du laboratoire



M. Wolf/Black Star-Grazia Neri



M. Wolf/Black Star-Grazia Neri



K. Reese/Marka

IBM new yorkais de East Fishkill a récemment développé une technique expérimentale de contrôle qui devrait permettre une nette amélioration du rendement de la production des puces : elle ne comprend pas moins de 300 phases.

La réalisation d'un transistor ou d'un autre composant consiste à déposer des impuretés, ou dopants, sur une tranche de silicium.

Ces dopants sont les éléments caractéristiques du composant. Ils doivent être disposés à des endroits déterminés avec une précision de l'ordre de l'atome.

Sinon, le circuit est défectueux. Le dépôt de la substance dopante suppose donc des mesures extrêmement minutieuses.

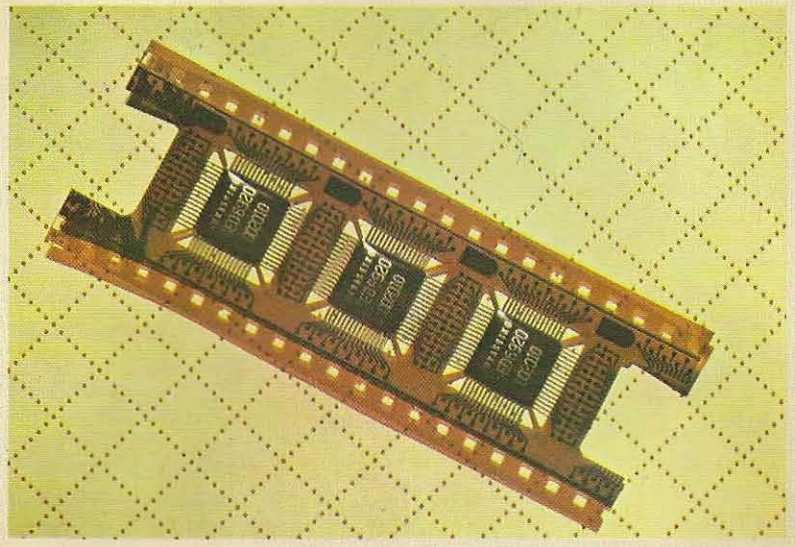
Cette nouvelle méthode de contrôle garantit le positionnement parfait des masques et donc, la disposition correcte des circuits sur la microplaquette.

Le positionnement des masques est, en effet, l'une des étapes les plus délicates.

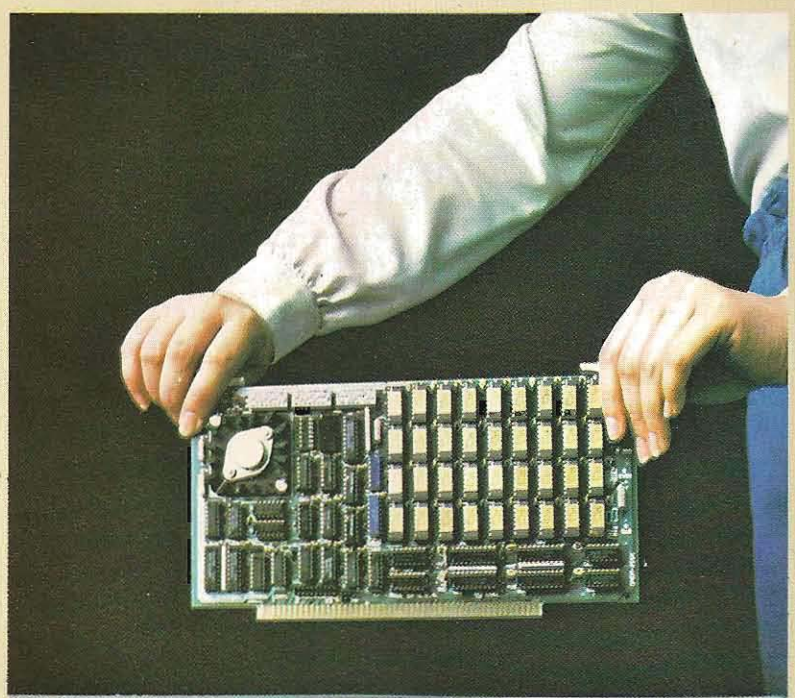
Il s'agit de disposer, sur des puces d'une quarantaine de millimètres carrés, des milliers de composants dont la taille est d'environ un micron (l'épaisseur moyenne d'un cheveu est de 250 microns).

Après cet examen optique, on teste le fonctionnement des circuits encore réunis sur les tranches (photo du milieu).

Chaque circuit est soumis au contrôle d'un testeur commandé par ordinateur. Des pointes conductrices, placées automatiquement aux endroits voulus, transmettent



M. Wolf/Black Star-Grazia Neri





B. Coleman/Marka

des signaux électriques de faible intensité. La réponse du circuit intégré est alors comparée à une réponse-type et la plus légère différence entraîne son élimination.
Lorsque tous les contrôles ont été effectués, on procède au découpage des plaquettes à l'aide d'une lame diamantée commandée par une machine micrométrique. Les « mauvais » circuits sont rejetés tandis que les « bonnes » puces sont câblées dans une machine spéciale afin de pouvoir être raccordées à l'extérieur (photo du bas de la page 768).
Une fois tous les fils de connexion soudés, les puces sont encapsulées dans des supports de céramique qui les protègent et les isolent définitivement. Seuls dépassent encore les fils de connexion.
Comme ils sont également très fragiles, on les protège à leur tour en insérant les éléments de céramique à la suite les uns des autres dans une bande qui rappelle une pellicule photographique (photo du haut).
Les perforations qui se trouvent de chaque côté permettent de manipuler les composants sans les endommager, notamment pour leur montage et leur soudage sur les plaques.
On peut voir sur la photographie ci-contre une plaque de composants prête à être placée dans le logement qui lui est destiné sur une machine.
L'insertion suffit généralement à établir les connexions, même s'il est parfois nécessaire de réaliser quelques soudures pour les rendre plus fiables.

Le logiciel

Un logiciel d'application est destiné à résoudre un problème déterminé. Progiciel standardisé ou programme « sur mesure », il doit :

- permettre la réalisation des traitements désirés et l'obtention des sorties voulues ;
- présenter une fiabilité satisfaisante.

La première de ces conditions est claire et impérative. Si un programme ne répond pas aux besoins exprimés; il n'y a plus qu'à le modifier ou, éventuellement, à revoir ses spécifications.
Il est simple de s'assurer de la correspondance entre prestations et spécifications : on fournit en entrée des données précises et l'on doit obtenir en sortie les résultats prévus.
Il s'avère beaucoup plus délicat de déterminer si la seconde condition est remplie, et la réponse n'est pas aussi catégorique. La démonstration de la justesse d'un programme n'est, en effet, jamais définie, car une erreur peut très bien échapper aux contrôles les plus rigoureux si elle ne se manifeste que sous certaines conditions particulières.
Ces erreurs masquées connaissent parfois des répercussions considérables. Ainsi, celles qui subsistaient dans les programmes du projet spatial Apollo – malgré les 600 millions de dollars que coûta leur développement ! – furent à l'origine de la plupart des difficultés qui entachèrent ce projet, et une erreur de ce type alla même jusqu'à entraîner l'échec de la première mission d'une sonde Vénus.
Toutefois, les logiciels sont loin d'atteindre une telle complexité dans les applications courantes et le risque de commettre des erreurs demeure beaucoup plus faible. De surcroît, on réussit généralement a les déceler et à les expurger.
Pour obtenir un programme sans erreur, il faut conjuguer deux actions complémentaires : la prévention et la correction.
La prévention commence dès la phase de conception du logiciel, et conditionne toute la suite de son développement. Quant à la correction, elle n'intervient, bien sûr, qu'en dernier

Une vue impressionnante des installations d'une salle de commandes.



Italcable

lieu, lors du contrôle et de la validation, mais elle est également déterminée par la façon dont le travail a été abordé.
Pour prévenir les erreurs, on peut développer le projet de façon à faire ressortir les modules du logiciel présentant les plus grandes difficultés. Il existe, pour chaque type de programme et pour chaque étape, des méthodes de conception et de programmation rigoureuses et complexes. C'est pourquoi nous ne traiterons que les techniques fondamentales, sélectionnant les aspects les plus importants pour les machines de la catégorie qui nous intéresse. Les points essentiels de la réalisation d'un logiciel d'application sont les suivants :

- définition des objectifs ;
- architecture du système, et, notamment, caractéristiques des interfaces ;
- structure du logiciel ;
- méthodes de programmation et choix des langages.

Il faut, avant tout, parvenir à une vision d'ensemble des buts à atteindre.
On distingue deux types d'objectifs : à côté des besoins spécifiques de l'utilisateur (qui constitueront l'essentiel des traitements), il existe un certain nombre d'objectifs à caractère général, que le programmeur ne doit jamais perdre de vue.
Tout bon logiciel d'une certaine ampleur doit en effet présenter, à un degré variable, les qualités suivantes :

Généralisation. Le développement d'un logiciel suppose toujours un lourd investissement. Aussi, un projet très spécifique, c'est-à-dire n'intéressant que peu d'utilisateurs, s'avère généralement peu rentable. Toutefois, il ne faut pas tomber dans l'excès inverse, car toute généralisation entraîne l'allongement du programme et les possibilités d'erreur s'en trouvent multipliées.
C'est au concepteur de savoir évaluer, cas par cas, si telle ou telle généralisation s'impose vraiment, et sera facile à mettre en œuvre sans que les délais d'apprentissage deviennent une charge trop lourde à l'utilisateur.

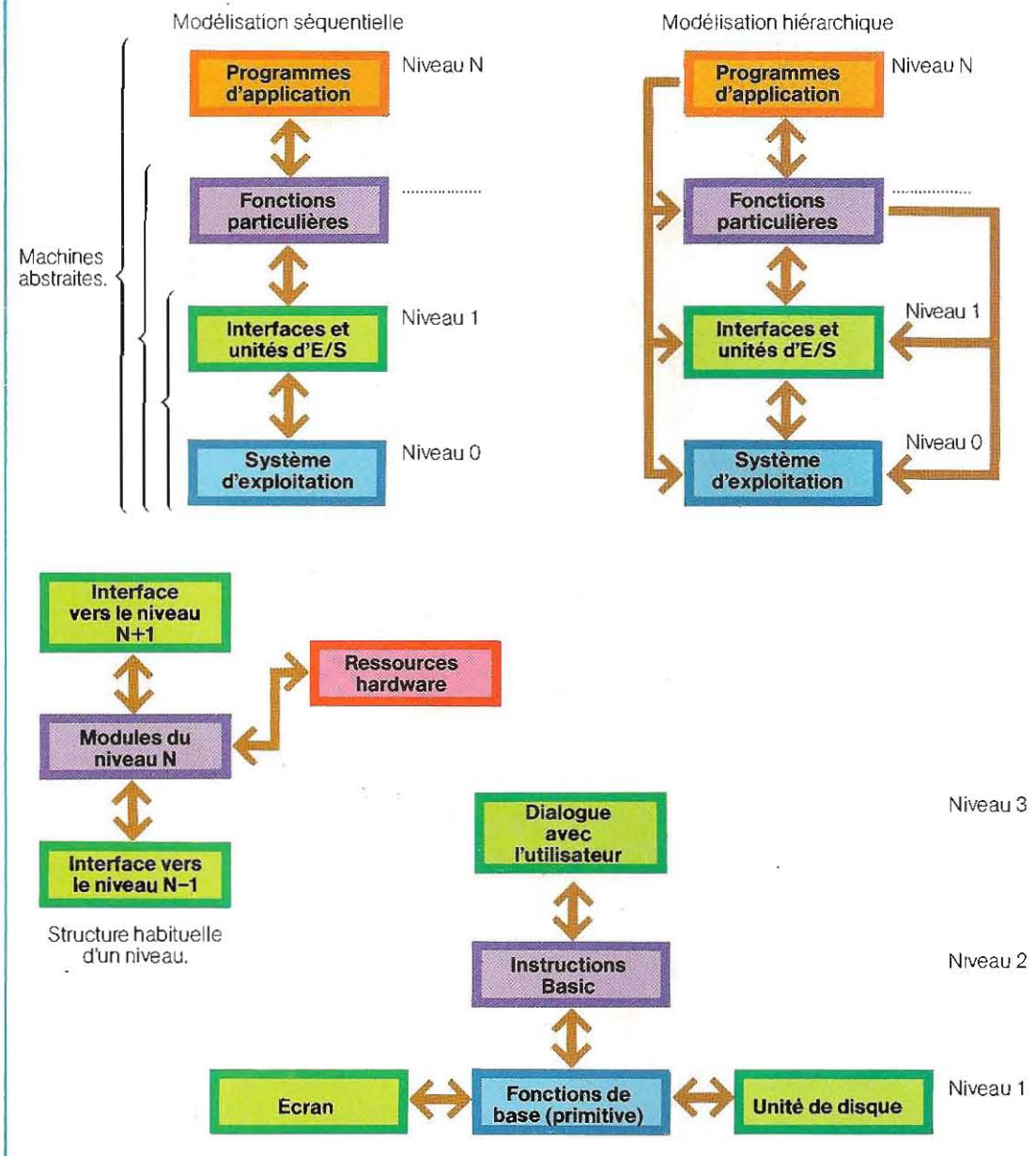
Adaptabilité. La trop grande spécificité d'un logiciel peut être avantageusement compensée par une bonne adaptabilité, c'est-à-dire par la possibilité d'ajouter ou de supprimer facilement des modules. On peut ainsi généraliser un programme ou l'adapter à de nouvelles applications sans aboutir à une structure trop complexe.
Les progiciels présentent souvent une grande souplesse, essentiellement assurée par leur segmentation en sous-programmes, voire en programmes chaînés. On peut alors ajouter ou remplacer des modules au fur et à mesure de l'évolution des besoins. Cette méthode renforce également la fiabilité, dans la mesure où elle permet d'échelonner dans le temps l'intégration des différentes fonctions.

Facilité de maintenance. La correction ou la mise à jour d'un logiciel doit être relativement simple. Cette qualité se rattache également à la fiabilité, car un programme facile à corriger est toujours plus fiable.

L'architecture du logiciel

Un logiciel d'application réunit un ensemble d'opérations distinctes mais corrélées, dont chacune peut être traitée dans un programme, ou dans une ou plusieurs routines. L'**architecture logicielle** du système vise à rassembler de façon cohérente tous ces modules en une procédure complète.
Cette approche nécessite, comme nous l'avons vu, la décomposition du programme principal en un certain nombre de sous-programmes, chacun devant à son tour se segmenter en niveaux inférieurs.
Cette modélisation en différents niveaux peut s'envisager de deux manières. On peut dire que la première méthode est séquentielle, car chaque niveau ne communique qu'avec l'étage précédent et l'étage suivant, tandis qu'avec la seconde méthode, la communication reste possible, non seulement entre niveaux contigüs, mais également entre un étage et tous ceux qui lui sont inférieurs. Sur les schémas qui représentent ces deux structures (page 771), le contenu mentionné pour chaque niveau n'a qu'une valeur d'exemple.
Cette analyse très générale s'avère surtout utile pour la réalisation de logiciels extrêmement complexes, destinés à tourner sur de gros systèmes, mais on peut également l'em-

ARCHITECTURE DU LOGICIEL ET NIVEAUX D'ABSTRACTION



Représentation schématique d'un programme Basic sur trois niveaux.

ployer avec profit sur un micro-ordinateur.
Chaque niveau d'abstraction permet de concevoir l'ensemble **(hardware + software)** comme une **machine abstraite,** dont les possibilités sont fonction de ce niveau : au degré 0, la machine abstraite ne comporte (outre le matériel) que le système d'exploitation. Celle du niveau 1 possède en sus les interfaces entre l'extérieur et les programmes système. La complexité continue d'augmenter à chaque degré, le niveau le plus haut étant celui du dialogue avec l'utilisateur.
Ainsi, par exemple, dans un programme de lecture d'un fichier, le niveau d'abstraction le

plus élevé correspondra au module qui saisit au clavier le numéro de l'enregistrement que souhaite lire l'opérateur. L'instruction GET (lecture de l'enregistrement) représente le niveau immédiatement inférieur. Il est lié à un niveau encore plus bas (traduction en Assembleur), lui-même lié au niveau 0 (fonction primitive d'accès au disque).
Exposons maintenant les principales propriétés de ce type d'architecture, dont certaines transparaissent dans notre exemple.

- Les caractéristiques d'un niveau ne sont pas connues aux niveaux inférieurs ;
- les échanges entre niveaux s'effectuent par le biais d'interfaces prédéfinies ;
- chaque machine abstraite possède des ressources propres, dont elle doit assurer le fonctionnement par interfaçage avec les ressources des autres niveaux ;
- les échanges entre niveaux ne peuvent porter que sur certaines données (les arguments des fonctions Basic, par exemple), tandis que les autres données ne peuvent circuler qu'entre les différents éléments d'un même niveau.

Ce type d'architecture, fondé sur la hiérarchisation des opérations à exécuter en différents niveaux de complexité, doit se compléter par une structure combinant trois grands types de communication :

- transfert du contrôle sans transfert de données (GOTO, par exemple) ;
- transfert du contrôle accompagné d'un transfert de données (sous-programmes) ;
- transfert de données exclusivement (par les ports d'accès).

On peut considérer le transfert de données entre deux modules (dont chacun dispose de plusieurs ports d'accès) comme un processus asynchrone. Le module qui doit envoyer des données à un autre les transmet au port d'entrée de ce dernier, qui ne les lit que lorsqu'il est prêt.
C'est précisément parce que chaque module opère indépendamment de l'autre qu'on dit que le processus (SEND/RECEIVE) est asynchrone. Dans ce type d'architecture, les éléments les plus importants sont les modules d'interfaçage, et plus particulièrement ceux qui gèrent le dialogue avec l'utilisateur.

C'est leur manque de souplesse qui explique une grande part des problèmes de fonctionnement des logiciels. Il faut adapter leur structure à chaque cas particulier, en observant quelques règles essentielles :

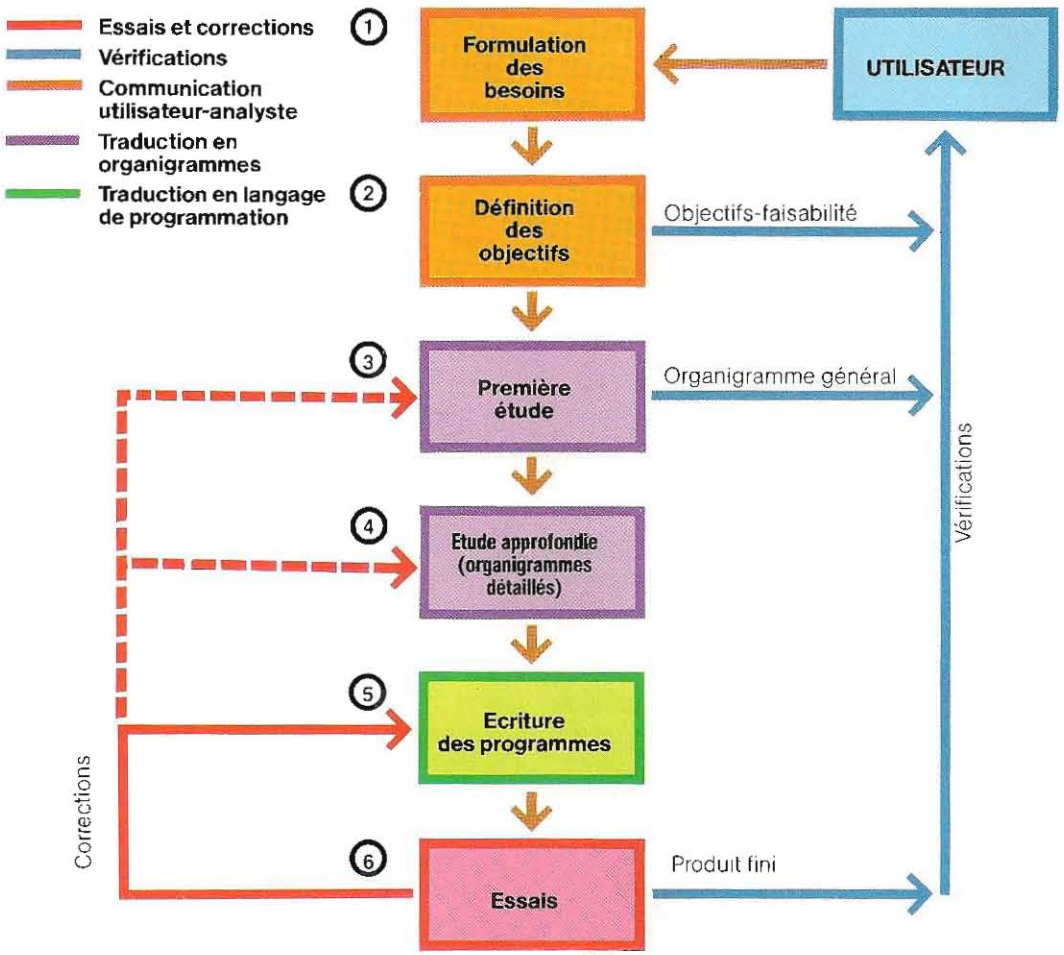
1 / les messages doivent être clairs, à la fois concis et complets. Si la forme abrégée peut convenir pour un message qui n'apparaît que rarement, un message qui revient souvent devra, surtout s'il s'adresse à des personnes menant plusieurs activités de front, être bref mais très explicite, de façon à pouvoir se comprendre instantanément, sans effort de réflexion ;
2 / il faut réduire au maximum les saisies de données, de commandes et de paramètres par l'utilisateur, en recourant le plus possible aux valeurs par défaut. On gagne ainsi un temps précieux lors de l'exécution ;
3 / les messages en entrée et en sortie doivent toujours être visualisés avec la même présentation, afin de respecter une plus grande continuité dans le dialogue avec l'utilisateur ;
4 / dans une procédure très complexe, il est bon de prévoir une commande qui active l'affichage d'un mode d'emploi du programme. En effet, il arrive souvent, notamment lors de l'apprentissage du logiciel, que l'opérateur ne sache plus très bien comment procéder. Des explications succintes fournies en temps réel lui éviteront de provoquer des erreurs en tapant des commandes non prévues. Ce module d'aide est couramment désigné par le terme anglais help.

La modélisation

On peut envisager le processus de réalisation d'un logiciel d'application comme une concertation à plusieurs niveaux. L'utilisateur explique ses besoins à l'analyste, qui décompose le problème à résoudre en opérations élémentaires et les transmet à son tour, sous forme d'organigrammes, au programmeur. Celui-ci les traduit en langage informatique, constituant le logiciel proprement dit. Au cours de sa réalisation, le projet passe donc par deux phases distinctes, comportant chacune plusieurs étapes de traduction :

- la première phase aboutit à un modèle macroscopique, définissant les objectifs et les grandes lignes du projet ;

REALISATION D'UN LOGICIEL : MODELE MACROSCOPIQUE



— la seconde phase (modèle microscopique) décompose chaque fonction (dont la nécessité est apparue pendant la première phase) en opérations élémentaires.

Les deux modèles peuvent correspondre chacun à la succession de ces deux phases, appliquées à un degré inférieur.
On obtient ainsi une modélisation sur deux niveaux – macroscopique et microscopique – qui serviront de guide, lors de la préparation, mais surtout pendant le travail de recherche des erreurs.
La plupart des erreurs sont commises au cours de l'une des opérations de traduction prévues dans les modèles. Il suffit de reprendre ceux-ci pour en trouver l'origine.

Le modèle macroscopique. Le schéma ci-dessus représente sous une forme simplifiée le modèle macroscopique du processus de développement d'un logiciel.
Lors de la première étape, l'utilisateur explique ses besoins à l'analyste (1). La découverte tardive d'une erreur commise au cours de cette phase peut coûter très cher, car sa correction nécessitera une modification importante de la structure même du logiciel. Il faut ainsi s'entourer dès le début du maximum de précautions, sans oublier que l'utilisateur lui-même parvient rarement à exprimer avec précision ses propres exigences. C'est donc à l'analyste de préparer soigneusement l'entretien et de le conduire judicieusement afin que rien ne puisse être omis ou mal compris.

La deuxième étape consiste à définir les objectifs (2). C'est en quelque sorte l'interprétation et la traduction du projet en langage d'informaticien. Il s'agit, en effet, de prévoir le coût de chaque objectif en fonction de la difficulté de programmation et de la configuration hardware nécessaire, et d'estimer si certains objectifs sont superflus ou si leur prix de revient est trop élevé par rapport au résultat escompté. Au terme de cette étape, véritable concertation entre l'analyste et l'utilisateur, on obtient une deuxième mouture des spécifications initiales, adaptée à la réalisation d'un logiciel d'une complexité limitée, en accord avec une configuration déterminée.

La définition des objectifs s'avère une étape d'autant plus importante que l'utilisateur est souvent peu averti des problèmes de l'automatisation. En effet, les utilisateurs néophytes surestiment fréquemment les possibilités des ordinateurs, et croient donc très simple la programmation de n'importe quel service. Ils ont ainsi tendance à demander une informatisation beaucoup plus poussée qu'il n'est nécessaire (puisque c'est la machine qui travaille !), sans se rendre compte que le coût de l'installation dépend essentiellement du coût du logiciel, et que les charges entraînées par une prestation supplémentaire sont parfois disproportionnées.

Toutefois, l'analyste ne doit pas guider son client vers le choix d'un logiciel réduit à l'essentiel. Des traitements qui lui paraissent secondaires sont peut-être en réalité indispensables ; s'ils ne sont pas pris en charge par le logiciel, il faudra passer beaucoup de temps à les effectuer à la main, ce qui risque de réduire à néant l'intérêt de la procédure automatisée.

L'étape suivante (3) consiste à représenter la structure du logiciel par un schéma logique (ou organigramme) général.

La méthode la plus rapide repose sur l'utilisation de l'analyse descendante (technique top-down).

Au cours de cette étape, il faut déterminer les transferts d'informations et les traitements à effectuer. Il faut également définir les sous-programmes éventuels et préparer leurs commentaires, en recourant par exemple à la méthode HIPO, principalement dans le but de vérifier la cohérence des différents échanges de données.

En général, les sorties d'un sous-programme constituent les entrées du sous-programme suivant. Des schémas clairs permettent de s'assurer rapidement que tout correspond bien. Quand le nombre d'informations à traiter est élevé, cette étape se place parmi celles où l'on court le plus de risques d'erreurs, d'autant plus graves qu'elles interviennent à un stade relativement précoce.

L'étape (4) consiste à établir l'organigramme de chaque module du logiciel (menu, sous-programmes, etc.). On peut alors déceler et corriger à moindres frais les erreurs commises au cours de la phase précédente.

C'est au cours de l'étape (5) qu'on traduit la structure logique du projet en langage de programmation. Les erreurs les plus nombreuses se produisent au cours de la rédaction du programme, mais elles sont les plus faciles à identifier et à corriger.

On pourrait croire que le processus s'achève ici. Il n'en est rien, car il reste les essais et les contrôles (6) qui constituent une extension de la programmation, dont ils sont même (au début du moins) véritablement indissociables. En effet, dès qu'une partie du programme est prête, on la teste, pour la modifier éventuellement en fonction des résultats, jusqu'à ce qu'elle donne entière satisfaction.

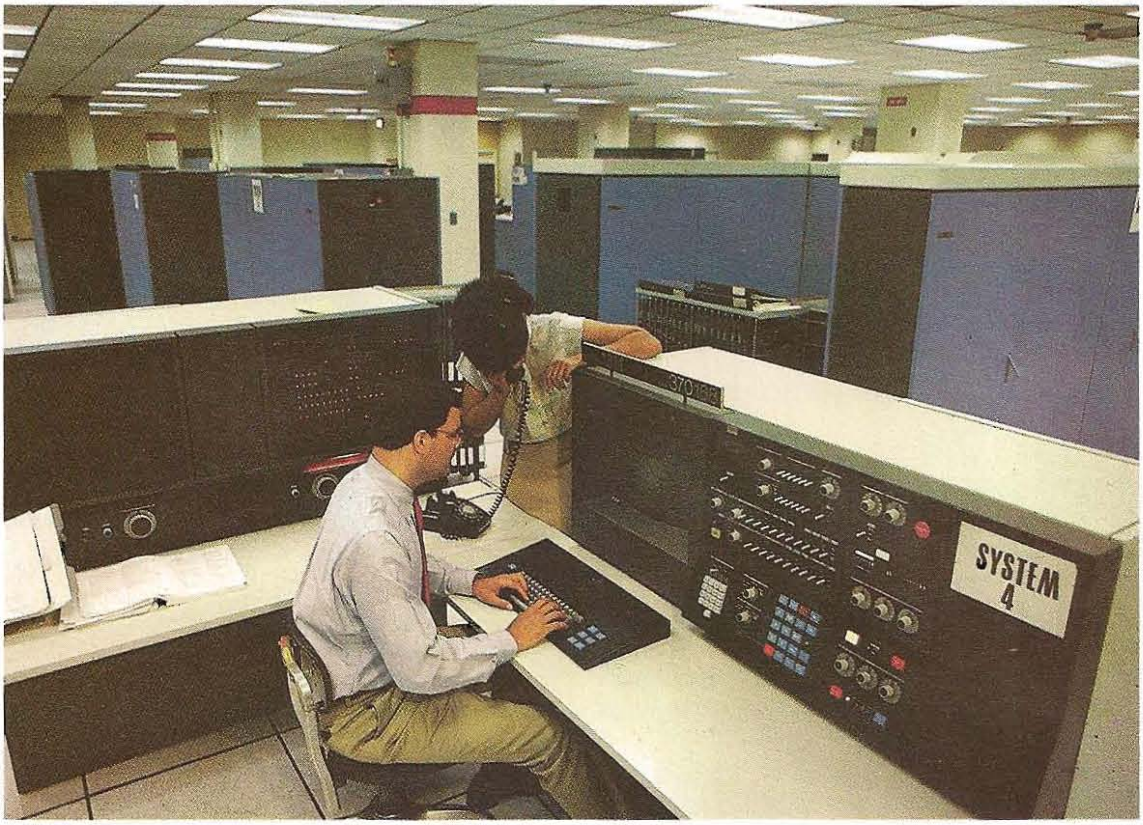
Véritable synthèse du projet, le modèle macroscopique sert de référence pendant la conception du logiciel, de guide pour la recherche des erreurs, et constitue un élément de comparaison pour évaluer le coût et la complexité des logiciels achevés ou en cours d'écriture.

Le modèle microscopique. Chacune des étapes qui permettent d'établir le modèle macroscopique se décompose à son tour en de nombreuses transformations microscopiques.

Voici les différentes phases d'une transformation microscopique :

1 / acquisition de l'information (par exemple, d'un besoin exprimé par l'utilisateur) ;
2 / élaboration (sur le papier) de la méthode à utiliser pour obtenir le résultat voulu ;
3 / formulation dans des notes du résultat de cette élaboration.

Si cette décomposition permet de faire le



J. Pickerell/Marka

Le système informatique IBM 370. On distingue au centre le tableau de contrôle du système de télétransmission des données.

parallèle entre une démarche intellectuelle et une procédure informatique (Entrée, Traitement, Sortie), il subsiste toutefois une différence de taille : c'est un homme et non une machine qui effectue ici le traitement.
Or, l'esprit humain est doué de gigantesques facultés associatives qui, dans l'optique de l'informaticien, constituent un handicap. Une information n'est jamais traitée isolément par notre cerveau ; au contraire, elle s'insère toujours dans un processus mental associatif, où le recours aux expériences passées permet généralement d'interpréter correctement des informations incomplètes ou erronées.
En revanche, il ne s'agit toutefois que d'une extrapolation qui peut conduire, sur la base d'a priori ou d'associations inconscientes, à déformer les informations les plus précises et les plus rigoureuses. En outre, la mémoire humaine est souvent défaillante et c'est là une autre source d'erreurs.
Dans le cas qui nous intéresse, il n'est pas rare d'oublier une information essentielle et de réaliser une transformation très incomplète.

On peut se prémunir partiellement contre ce type d'erreurs en prenant un certain nombre de précautions, et notamment, en préparant une documentation très détaillée, en utilisant des méthodes d'analyse rigoureuses et en optant pour le langage le plus approprié.
Il arrive que plusieurs personnes collaborent à un même projet. Des difficultés particulières surviennent alors, surtout lorsqu'on doit relier des modules rédigés par des programmeurs différents.
Pour éviter les incohérences, on applique alors la règle du « plus-un-moins-un », en vertu de laquelle la vérification de la partie **n** d'un projet doit être effectuée par les responsables des parties **n-1** et **n+1**.
De cette façon, le programmeur du module en amont **n−1** s'assure de la compatibilité entre les sorties fournies par son module et les entrées nécessaires au module à tester **n**, tandis que le programmeur du module en aval **n+1** vérifie, au contraire, la correspondance entre les sorties du module **n** et les entrées nécessaires au sien.

La structuration des programmes

Une fois terminée la conception de l'architecture générale du logiciel, il faut s'attaquer à la définition précise des différents modules.
Qu'il se compose d'un ou de plusieurs sous-programmes, ou qu'il constitue un programme à part entière, chaque module est défini par trois grandes caractéristiques :

1 / la fonction exécutée ;
2 / la logique du traitement ;
3 / le contexte qui lui est nécessaire (entrées, sorties, etc.).

La fonction du module a été définie au cours de l'étape précédente (architecture) : c'est la description des opérations qu'il exécute quand on l'appelle.
La logique du traitement indique la façon dont sont exécutées ces opérations, c'est-à-dire les algorithmes mis en œuvre. Elle dépend à la fois du langage utilisé et des préférences du programmeur.
Le contexte (ou environnement) regroupe les conditions matérielles et logicielles dans lesquelles intervient le module, ainsi que les particularités d'une utilisation éventuelle.
Chaque module doit être aussi indépendant des autres que possible et avoir une longueur raisonnable. Si la nature des opérations exécutées dans un module exige un grand nombre d'instructions, il faut segmenter ce module en plusieurs parties.
Il ne faut pas négliger la documentation lors de la définition des modules. Des commentaires complets et exacts constituent à la fois un véritable plan de travail pour la préparation des modules et un excellent moyen d'appréciation de la qualité d'un module déjà rédigé et de ses possibilités d'implémentation.
De plus, les commentaires représentent un outil indispensable aux contrôles de validité. Les renseignements suivants doivent y figurer impérativement :

- **Nom du module** — En Basic standard, le nom d'un sous-programme est le numéro de ligne où il commence. Si, par exemple, un programme de lecture sur disque débute en ligne 1000, on l'appelle par l'instruction GOSUB 1000. La ligne 1000 constitue son « point d'entrée » (entry point). Dans certaines versions du Basic et dans d'autres langages comme le Fortran, il est possible de désigner les sous-programmes par un nom symbolique : ainsi, en baptisant la routine « DISQUE », on fera de ce nom son point d'entrée et l'instruction d'appel sera GOSUB DISQUE.
  Que le nom du module soit un mot ou un numéro de ligne, il devra figurer dans la documentation. Lorsqu'un module possède plusieurs points d'entrée, ils doivent tous être indiqués.
- **Opérations exécutées** — Description des opérations effectuées.
- **Paramètres** — Liste des variables et des constantes utilisées.
- **Entrées** — Description des données à fournir en entrée.
- **Sorties** — Description des données produites en sortie.
- **Erreurs** — Description des erreurs qui peuvent se produire, avec indication des valeurs correspondantes des flags et des éventuelles procédures de rattrapage.

La méthode de développement descendante

La méthode descendante représente, à peu de choses près, la mise en pratique des notions présentées dans le chapitre consacré aux organigrammes et aux techniques de préparation des programmes. Il a toutefois semblé préférable de la présenter seulement ici – quitte à bousculer un peu l'enchaînement logique – afin que la connaissance du Basic rende son exposé moins abstrait et plus facile à comprendre.
Il est capital de doter un logiciel d'une structure claire, qui facilite sa modification ou l'adjonction de nouvelles fonctions. Pour cela,

Quel logiciel ?

Cela fait maintenant trente ans que la « crise du logiciel » prévue par B. W. Boehm secoue le monde du traitement de l'information. En 1973, un schéma publié pour la première fois dans la revue « Datamation » illustrait ce bouleversement de façon éclatante (voir schéma de gauche, page 778). Et pourtant, l'idée que la part du logiciel dans le coût total des systèmes informatiques était destinée à dépasser largement celle du matériel a mis longtemps à faire son chemin. Ce n'est que récemment qu'on a mesuré à quel point le rôle des programmes et, par conséquent, du contrôle de leur qualité serait prépondérant.

La crise du logiciel recouvre en effet une crise de « qualité » du logiciel, puisqu'en utilisant les échelles appropriées, on parvient à représenter l'évolution du coût relatif de cette qualité par une courbe analogue à la précédente (voir schéma de droite, page 778).

A cet égard, l'introduction de la **programmation structurée** a constitué un grand pas en avant, en promouvant la création des logiciels au rang de science à part entière. Nous allons retracer rapidement les grandes étapes de cette importante innovation méthodologique et conceptuelle, afin que le lecteur en apprécie toute la portée.

La complexité croissante des programmes rédigés dans les langages classiques comme le Fortran, le Basic ou l'Assembleur avait fait comprendre la nécessité de revoir les méthodes de programmation habituelles. Le tout premier jalon de cette nouvelle rigueur avait été posé par le professeur Edsger W. Dijkstra qui, dès 1965, observait dans un article que « ... la qualité du software est inversement proportionnelle au nombre d'instructions GOTO qu'il contient ».

Puis, en 1966, C. Boehm et Jacopini énoncent à leur tour un théorème selon lequel « il est possible de structurer n'importe quel programme classique en procédant à une série de manipulations élémentaires ou de transformations de ses structures de contrôle logique ».

En 1971, Niklaus Wirth établit les règles fondamentales d'une programmation structurée, qui procède par améliorations successives. Un peu plus tard, la même année, le docteur Halan Mills et F. Terry Baker démontrent pour la première fois les avantages qu'on peut tirer de l'application de cette nouvelle méthode au domaine industriel.

L'étape suivante dans l'histoire de la programmation structurée se situe en 1972, avec la mise au point par M. Parnas de la **décomposition modulaire**. Cette technique a pour résultat la subdivision de la version définitive d'un programme source en unités caractérisées par un certain « niveau d'abstraction ».

C'est également en 1972 que Weinberg, Dahl, Dijkstra et Moore publient les premiers manuels universitaires sur ce sujet. En décembre 1973, toute une série d'articles de la revue Datamation présente la programmation structurée comme une « révolution de la programmation ». Cette méthodologie, extrêmement bien accueillie, s'attache parmi les universitaires de nombreux adeptes qui, depuis, s'efforcent de la mettre en œuvre avec un zèle qui frôle parfois le mysticisme.

Le concept clé de la programmation structurée est l'**abstraction**, qui permet de faire ressortir sélectivement certains aspects d'un problème donné. Il s'agit de ne retenir que les propriétés essentielles d'un objet, tout en ignorant volontairement les détails évidents. Cette démarche est comparable à celle des programmeurs, qui se réfèrent généralement à des « machines abstraites », dans lesquelles sont laissées de côté toutes les questions de matériel, comme l'attribution des emplacements de mémoire.

Par cette élimination du détail, l'abstraction permet de réduire la complexité, réelle ou apparente, d'une tâche. En d'autres termes, une abstraction est la description simplifiée, ou **spécification**, d'un système dont certaines caractéristiques sont mises en relief et d'autres négligées. Naturellement, une bonne abstraction met l'accent sur l'information importante et les détails **non pertinents** sont supprimés (momentanément, du moins).

Cette maîtrise de la complexité par l'abstraction se rapproche de ce qu'on appelle, dans beaucoup d'autres secteurs, le « modèle analytique ». De nombreuses difficultés sont notamment communes aux deux méthodes. Il faut choisir les caractéristiques prépondérantes du modèle, les marges de variabilité de ses paramètres, le formalisme à adopter, les méthodes de validation, etc.

La description de ces modèles théoriques

s'éloigne tellement des systèmes réels qu'une phase de ***validation*** *explicite s'impose toujours.*

Comme dans d'autres domaines, on définit souvent des ***hiérarchies de modèles****, dans lesquelles les détails des opérations mentionnées à un certain niveau sont éclaircis au niveau inférieur.*

La description abstraite d'un modèle constitue sa spécification tandis que le modèle immédiatement inférieur en est la mise en œuvre. La vérification inclut donc le contrôle de la cohérence entre une spécification et sa mise en œuvre.

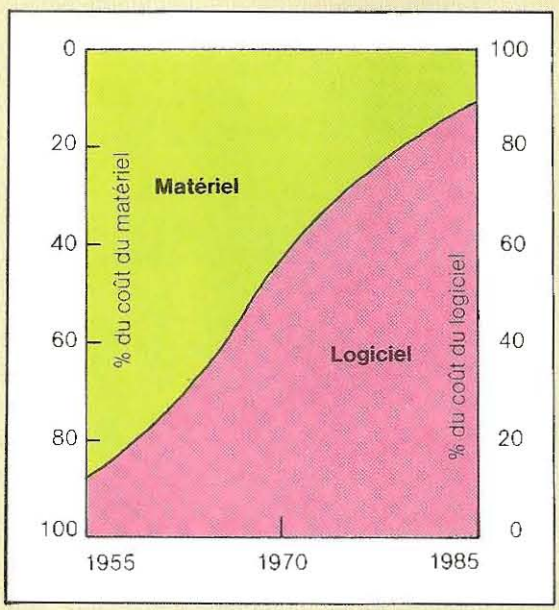
L'utilisation des abstractions, telle qu'elle est pratiquée en ***génie logiciel****, tend à mettre l'accent sur les propriétés fonctionnelles et les résultats voulus, au détriment de la manière de les obtenir.*

C'est ainsi qu'en 1960 les plus importants développements des méthodes et langages de programmation comme le Fortran concernaient les fonctions et les procédures (routines), qui permettaient de résumer une partie de programme par un nom et une liste de paramètres. Outre cette esquisse de modularité, on ne savait alors contrôler que l'aspect syntaxique, et les techniques de spécification se bornaient à attribuer un nom à une procédure. En 1970, le concept de ***structure de données*** *et les techniques de spécification devinrent les principaux objets de développement. On emprunta aux mathématiques leurs méthodes de logique et à la linguistique sa* ***sémantique****, pour créer une nouvelle science.*

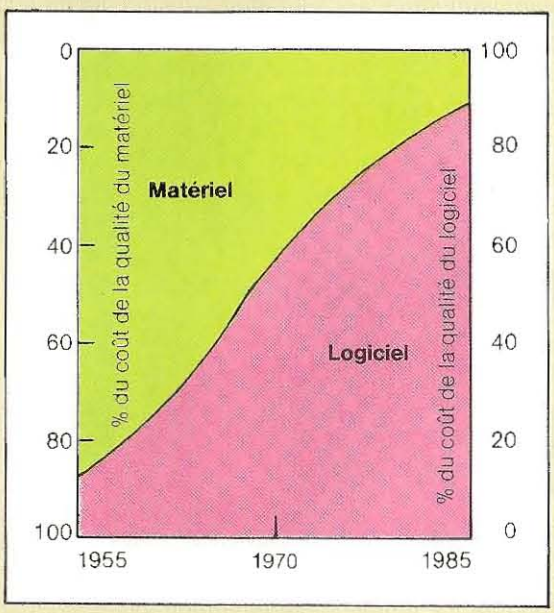
La recherche et le développement de nouvelles méthodologies et de nouveaux langages ont été menés en réponse aux besoins nés des limites étroites des langages (Basic, Fortran, Algol, etc.) et des techniques de programmation classiques.

C'est ainsi que la notion de ***méta-langage*** *s'est répandue. Cette formule désigne un langage formel qui décrit d'autres langages. Aujourd'hui, la recherche porte essentiellement sur la mise au point de nouveaux langages et sur la systématisation théorique des langages évolués, particulièrement des langages algébriques (formels). Des mathématiciens et des logiciens de renom international donnent le meilleur d'eux-mêmes pour créer un langage universel et rigoureux. Citons, par exemple, les travaux de l'école viennoise (langage VSM), où l'on considère qu'une programmation rigoureuse ne peut qu'être fondée sur les mathématiques et l'algèbre abstraite du plus haut niveau, ceux de l'école IBM, dirigée par Backus et Naur, où l'on prône le style de programmation pour « fonctionnels »*

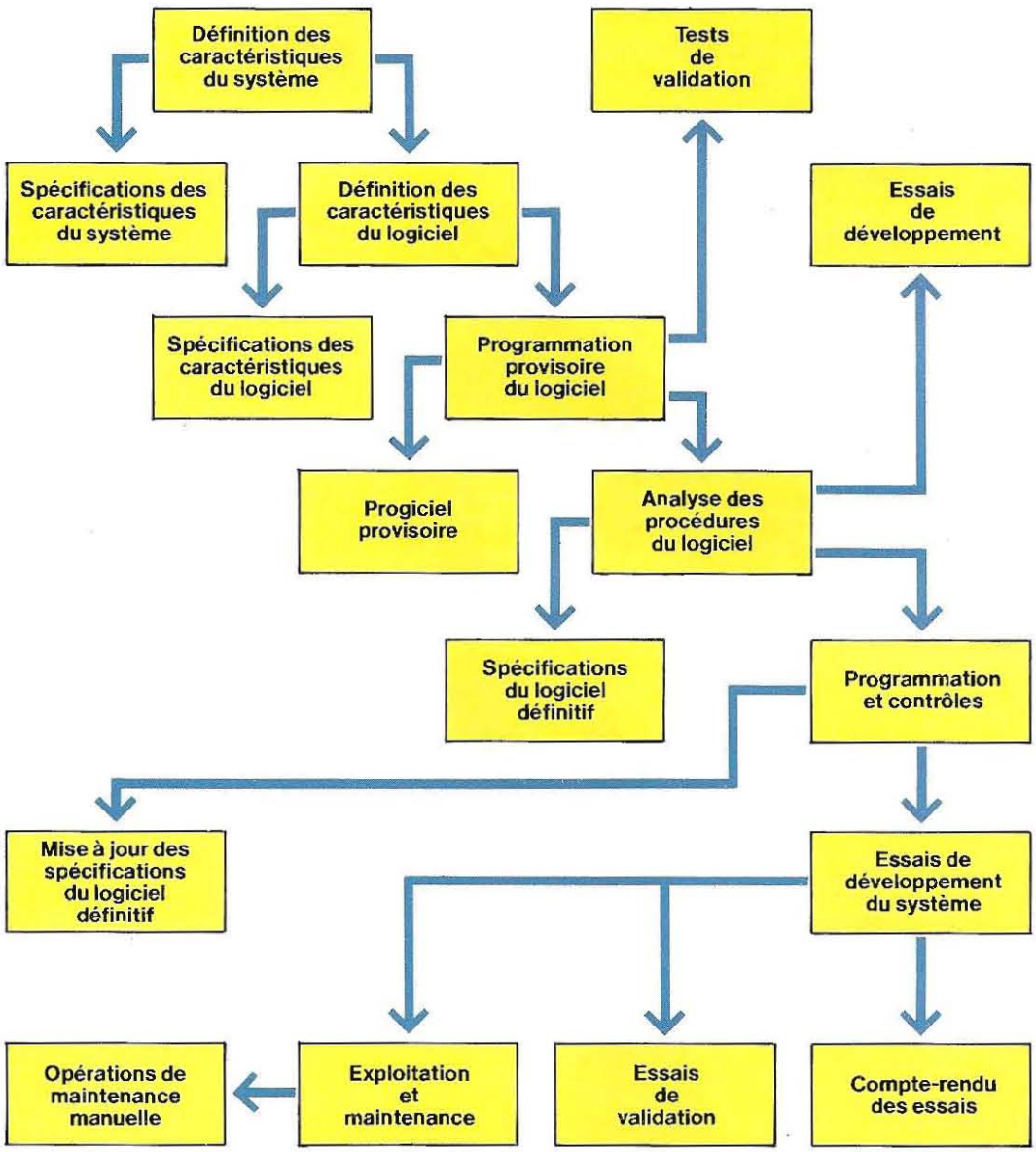
Répartition relative du coût d'un système informatique



Répartition relative du coût de la qualité d'un système informatique



LES ETAPES DU DEVELOPPEMENT D'UN LOGICIEL



(FP style) ou encore ceux de SRI International. Parallèlement, le département de la Défense américain développa spécialement ADA, nouveau langage général qui, associant les algorithmes les plus modernes aux structures de contrôle classiques, permet la définition de types et de sous-programmes, la programmation en temps réel et le traitement de processus parallèles. Il est, bien sûr, impossible de savoir ce que l'avenir nous réserve. Cependant, tout laisse prévoir le développement de nouveaux langages du même style qu'ADA pour répondre à la triple nécessité d'améliorer la qualité de la programmation, afin, notamment, de disposer de logiciels plus fiables et plus faciles à maintenir ; de reconnaître que la programmation constitue une activité humaine (le « warmware ») ; d'obtenir un excellent rendement à tous les niveaux.

Il faut recourir à l'une des nombreuses techniques de développement dont on dispose aujourd'hui et dont la mise en œuvre est plus ou moins délicate.
La méthode descendante ou **top-down** présente l'avantage d'être à la fois relativement simple et parfaitement fiable. Cette technique consiste à faire ressortir les grandes lignes d'un problème, en le décomposant en problèmes plus simples, qui sont ensuite résolus directement, ou décomposés à nouveau de manière récurrente, jusqu'à la définition d'opérations élémentaires.
On obtient ainsi une structure arborescente, dont les ramifications partent du problème principal et représentent les opérations à effectuer pour le résoudre de façon de plus en plus détaillée. Chaque fonction, ou groupe

La qualité d'un logiciel en 43 mots clés

Logiciel. C'est l'ensemble des programmes et des données nécessaires à l'exécution d'une procédure sur un ordinateur. On distingue le logiciel d'application du logiciel de base, qui comprend les systèmes d'exploitation, les superviseurs, les compilateurs, les utilitaires et les routines de contrôle. Par extension, le logiciel englobe également la documentation des programmes, c'est-à-dire la description des procédures, les spécifications, les règles d'utilisation, etc.

Génie logiciel. Lancé en 1968, ce terme désigne, par opposition à une programmation empirique et désordonnée, la mise au point et l'utilisation de méthodes rigoureuses, pour la production de programmes structurés, fiables et performants.

Clarté. Un produit logiciel est clair si son objet apparaît nettement à celui qui en fait la recette.

Achèvement. Un logiciel est achevé lorsque tous ses modules sont développés et prêts à fonctionner.

Concision. On dit qu'un logiciel est concis lorsque toute redondance de fonctions a été évitée.

Portabilité. Un logiciel est portable s'il est facile de le transférer et de le faire tourner sur d'autres systèmes que celui pour lequel il a été conçu.

Cohérence. Dans un programme cohérent, notations, symboles et terminologies sont homogènes. La cohérence externe suppose, en outre, l'homogénéité de son contenu avec les spécifications.

Maintenance. Un logiciel doit pouvoir être adapté facilement à de nouveaux besoins.

Preuve. Pour faire la preuve d'un produit logiciel, on cherche à définir des critères de validité et d'évaluation de ses prestations. Il faut, naturellement, vérifier que ces critères sont satisfaits après les avoir définis.

Fonctionnabilité. Un logiciel est fonctionnel s'il répond de manière satisfaisante aux objectifs fixés et s'il est facile à utiliser. Cela suppose une interface homme-machine, et donc des sorties de bonne qualité ainsi qu'une définition correcte des fichiers et des données en entrée. Par ailleurs, la possibilité d'intégrer partiellement ou totalement un programme à d'autres projets constitue également un critère de fonctionnabilité.

Fiabilité. Un programme est fiable, au sens large, s'il y a une bonne probabilité qu'il puisse fonctionner correctement dans les délais et les conditions prévus.

Structuration. Un logiciel est structuré si les liens entre ses différentes parties sont eux-mêmes organisés dans une structure.

Performance. Un logiciel performant fournit les résultats escomptés sans gaspiller inutilement les ressources du système.

Ressources du logiciel. Dans une acception large du terme, ce sont la taille de la mémoire utilisée, le nombre total d'instructions, le nombre de fichiers ouverts, la capacité du canal d'E/S, etc.

Indépendance. Un logiciel est indépendant des dispositifs fonctionnels si on peut le faire tourner sur un système dont la configuration est différente de celle pour laquelle il a été conçu au départ.

Précision. Un logiciel précis fournit des résultats aussi affinés que le nécessite le type d'application envisagé.

Accessibilité. Pour faciliter la tâche d'un utilisateur non spécialiste, les données à l'entrée d'un logiciel doivent être définies avec précision et ses sorties seront claires et faciles à utiliser, aussi bien dans leur forme que dans leur contenu.

Lisibilité. Un logiciel lisible fonctionne bien, et on peut saisir sa logique à la simple lecture des instructions.

Extensibilité. Un logiciel est extensible s'il se prête facilement à l'adjonction de nouvelles données ou d'opérations de contrôle.

Ergonomie. Un logiciel ergonomique, ou «amical», effectue ses prestations sans faire perdre de temps ou d'énergie à l'utilisateur, et sans influer négativement sur son moral.

Adaptabilité. Un logiciel est adaptable s'il est facile d'y introduire des variances.

Auto-description. Un logiciel est auto-descriptif s'il renferme suffisamment d'informations pour qu'un lecteur puisse déterminer les objectifs, les relations, les entrée/sorties, les éléments et les fonctions de ses différents programmes ou sous-programmes.

de fonctions, est traitée dans un sous-programme. Dans la mesure où les sorties d'une routine constituent les entrées de la suivante, il faut accorder une attention particulière à l'homogénéité des noms et des types des variables utilisés. Cela sera d'autant plus facile lorsqu'on s'astreindra à documenter et à commenter tous les modules.

La méthode descendante permet de construire un programme à la manière d'un puzzle. On commence par préparer les modules les plus importants alors que les autres sont simplement simulés. On intègre ensuite les sous-programmes au fur et à mesure de leur rédaction, en effectuant à chaque fois les essais nécessaires. On aboutit ainsi à une structure très souple, facile à faire évoluer puisqu'il suffit, pour modifier une fonction, de

Couverture fonctionnelle. Elle consiste à insérer des flags, des étiquettes et autres indicateurs (éventuellement complétés par des commentaires) à tous les endroits stratégiques d'un programme, de façon à s'assurer que l'exécution se déroule suivant l'enchaînement voulu.

Vérification du logiciel. Elle a pour objet de déterminer si le logiciel répond aux spécifications.

Validation du logiciel. Elle est destinée à établir si le fonctionnement du logiciel est satisfaisant dans le cadre d'un système global ; elle comprend donc une évaluation des qualités attendues du logiciel.

Certification du logiciel. Elle atteste la qualité (fiabilité, etc.) d'un logiciel et ne peut être effectuée que par un organisme agréé.

Test du logiciel. Il consiste à exécuter le logiciel, afin de montrer si les résultats sont corrects ou non.

Mise au point du logiciel. C'est le travail d'identification, de localisation et de correction des erreurs (debugging).

Inspection du logiciel. Il s'agit d'un examen du software et du hardware correspondant.

Contrôle du logiciel. C'est la dernière étape d'essais et de peaufinage avant la remise du produit à l'utilisateur.

Test des performances. Il permet de savoir si un logiciel satisfait certains critères, concernant notamment le temps calcul et l'espace mémoire requis par son exécution.

Recette. C'est l'ensemble des vérifications et formalités qui permettent de déterminer si le produit est acceptable par l'utilisateur.

Cycle de conception du logiciel. Il commence par la définition des besoins et des spécifications, et se poursuit par l'analyse, la programmation, la mise au point, l'exploitation et la maintenance.

Modèle. Rapporté à un logiciel, c'est la représentation théorique d'une procédure ou d'un produit.

Algorithme. C'est un texte concis (un ensemble fini de symboles clairs) par lequel on décrit la séquence d'opérations qui sera programmée pour exécuter une tâche.

Intervalle de fiabilité. C'est la durée pendant laquelle, à partir d'un instant donné T, il est probable qu'un système sera opérationnel.

Disponibilité. On distingue deux aspects :
– la disponibilité ponctuelle représente la probabilité qu'à un instant donné T, un logiciel pourra fonctionner dans les limites de tolérance prévues par la technique et l'environnement ;
– l'intervalle de disponibilité constitue la durée pendant laquelle le logiciel a toutes les chances de fonctionner dans ces mêmes limites de tolérance.

Erreur. C'est une inexactitude conceptuelle, syntaxique ou opératoire, qui risque d'empêcher le fonctionnement du logiciel.

Faute. C'est la manifestation spécifique d'une erreur de programmation qui rend impossible l'exécution d'une fonction. Une erreur peut être à l'origine de plusieurs fautes.

Panne. Elle se produit quand une donnée saisie révèle un défaut du programme, dont l'exécution ne peut donc se poursuivre correctement.
Une panne peut être :
– révélée, si elle s'est effectivement produite ;
– masquée dans le cas contraire ;
– critique, si elle stoppe l'exécution du logiciel ;
– physique, si elle bloque le système ;
– logique, si elle est critique sans être physique.

Logiciel sûr. C'est un logiciel à « l'épreuve des erreurs », c'est-à-dire d'une telle fiabilité qu'aucune erreur ne pourra provoquer d'incident critique.

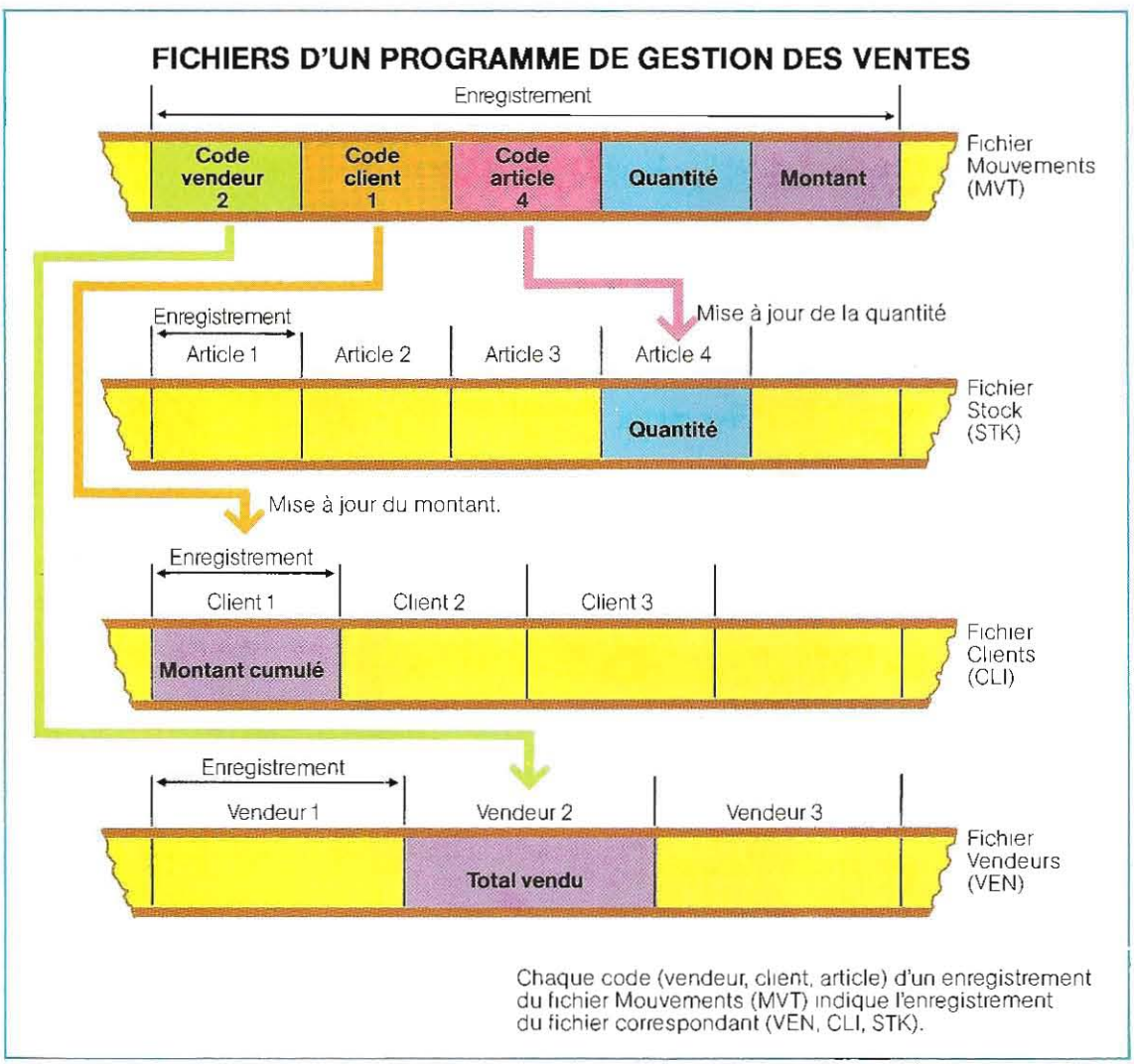
Code structuré. Codage des méthodes logicielles dans lequel l'accent est mis sur les structures logiques directes.

Méthode de l'arbre des erreurs. Méthode d'analyse de la propagation d'une erreur au cours d'un programme, et des disfonctionnements de niveau supérieur qui peuvent en résulter.

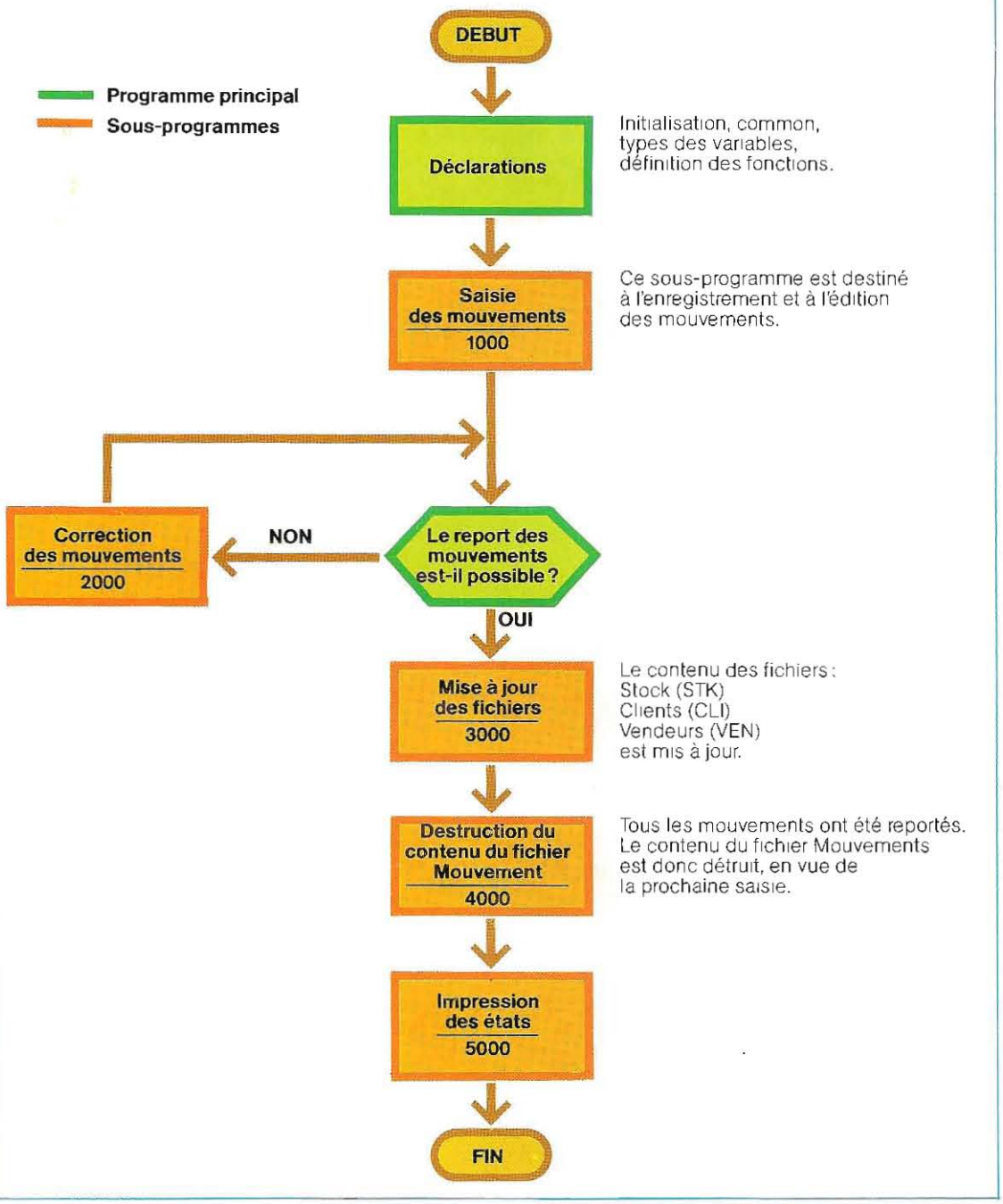
remplacer le module ou le sous-programme qui l'exécute. En outre, la validité du logiciel dans son ensemble est plus simple à établir.
L'exemple qui suit va nous permettre d'illustrer la technique de l'analyse descendante.
Une société emploie plusieurs vendeurs. On veut enregistrer les ventes de chacun (code article, quantité et montant), et le nom des clients correspondants.
Par ailleurs, ces données doivent servir à la mise à jour des quantités en stock et du total des achats de chaque client.
On veut obtenir en sortie :
- le chiffre d'affaires réalisé par chaque vendeur ;
- l'état des stocks ;
- la situation des clients.

Les ventes constituent ici les données d'entrée, tandis que les traitements se résument à la mise à jour des fichiers Stocks, Clients et Vendeurs.
Quatre fichiers sont donc nécessaires :
- Mouvements : données concernant les ventes ;
- Stocks : quantité en stock de chaque article ;
- Clients : cumul des achats de chaque client ;
- Vendeurs : cumul des ventes de chacun d'eux.

Le fichier Mouvements recevra les données en entrée. Périodiquement (chaque soir, par exemple), son contenu sera édité, puis éventuellement corrigé, avant d'être utilisé pour la mise à jour des fichiers permanents.
Le schéma ci-dessous détaille la structure



ORGANIGRAMME GENERAL DU PROGRAMME DE GESTION DES VENTES



des enregistrements des différents fichiers.
La page 783 retrace l'organigramme général du programme.
Dans un premier temps, les routines sont simplement simulées, et ne commandent que l'affichage d'un message à l'écran. Cela permet de vérifier, tout au long de l'élaboration du programme, si l'enchaînement logique est correct. Les véritables sous-programmes sont ensuite intégrés et testés l'un après l'autre. Une partie du programme principal est consacrée aux déclarations (dimensionnement des tableaux, définition des fonctions et des types de variables, etc.).

Les erreurs du logiciel

Pour prévenir comme pour corriger les fautes (bug) de programmation, il faut en connaître les causes. Mettons tout de suite de côté les erreurs de langage, syntaxiques ou lexicales, qui sont automatiquement diagnostiquées par le système et ne posent donc guère de problèmes. La moitié des autres fautes sont dues à des erreurs de rédaction.
On les distingue selon la fonction du bloc qui les renferme, ou encore selon le type d'instruction concernée. Les statistiques montrent que 15 % des erreurs affectent les zones d'E/S des programmes, destinées à échanger les informations avec l'extérieur, 30 % des erreurs sont commises dans les DATA ou dans d'autres instructions d'affectation, et 20 % proviennent d'inexactitudes dans les instructions conditionnelles (mauvais branchements ou inversion du sens des tests, par exemple). Enfin, 25 % sont dues à un enchaînement incorrect des instructions et le reste à des causes diverses.
Ces chiffres indiquent où il faut traquer les erreurs en priorité :

- instructions d'affectation (30 %) ;
- enchaînement des instructions (25 %) ;
- instructions conditionnelles (20 %) ;
- interfaces (15 %) ;
- divers (10 %).

Toutes ces erreurs de programmation sont relativement faciles à identifier, mais il existe d'autres types de fautes, en nombre à peu près égal, dont il s'avère parfois très délicat de découvrir l'origine : celles qui sont dues à un mauvais transfert des informations.
Réaliser un logiciel revient à décrire, dans un langage compréhensible par la machine, l'ensemble des opérations nécessaires à l'obtention d'un résultat déterminé. L'analyse consiste à définir ce résultat, et à trouver les algorithmes permettant d'y parvenir de la façon la plus satisfaisante. Pour cela, l'utilisateur transmet à l'analyste un très grand nombre d'informations sur ses besoins et ses méthodes de travail. L'analyste doit alors modifier la procédure décrite par son client en fonction des contraintes de la machine et proposer à celui-ci une solution parfaitement adaptée à l'automatisation des tâches.
Il s'établit ainsi un dialogue entre deux spécialistes aux domaines de compétence différents, qui tentent chacun de percevoir le point de vue de l'autre.
Cet échange est souvent limité par une certaine incommunicabilité : chacun des deux interlocuteurs omet de préciser certains points qui lui paraissent évidents. Il en résulte une analyse incomplète, qui ne peut déboucher que sur un programme inadapté, dont les défauts apparaîtront tardivement, lors des tests finaux ; on adoptera alors des solutions de secours, improvisées et toujours quelque peu bancales. Les corrections ne pourront guère constituer que des greffes sur un programme dont la structure n'aura pas été conçue pour ces nouvelles fonctions. Le logiciel risquera alors de perdre son architecture initiale, pour devenir un ensemble peu fonctionnel de modules plus ou moins bien articulés.
La seule façon d'éviter ce résultat catastrophique sera de respecter, dès le début du projet et pour chacune de ses phrases, un plan de travail en trois temps :

- exposition de besoins ;
- conception ;
- vérification.

L'application systématique de cette démarche rigoureuse garantira la réalisation d'un logiciel de qualité.

Test et contrôle du logiciel

Tester un programme, c'est s'assurer qu'il fonctionne dans toutes les situations qui peuvent se présenter.
Les résultats des essais apportent les réponses aux questions fondamentales que se posent l'utilisateur et le programmeur :

- le programme que je vais acquérir (ou que j'ai écrit) marche-t-il et dans quelle mesure ;
- en cas d'erreur, les données déjà saisies risquent-elles d'être perdues ;
- sera-t-il possible de modifier le programme, et combien cela coûtera-t-il ;
- quelle est la fréquence des blocages du système dûs à des erreurs à laquelle il faut s'attendre ; combien de temps le système restera-t-il hors-service ?

En fait, dans la pratique, on se borne généralement à tester les principales possibilités,

dont on estime qu'elles reflètent la quasi-totalité des cas. Il ne peut, bien sûr, s'agir d'une supposition gratuite. Les tests n'ont de valeur que si l'on recourt à des techniques bien définies pour contrôler les programmes et surtout, pour les structurer et les rédiger. En somme, il faut penser à la phase de contrôle dès le début de l'analyse, de façon à concevoir une structure qui facilite les essais.

Avant d'aborder les méthodes de recherche des erreurs, il convient de définir ce qu'est une erreur de programmation.

Prenons un exemple. Les mouvements des avions au voisinage des aéroports sont guidés par des systèmes radar très perfectionnés, contrôlés par ordinateur. L'approche de tout objet volant – avion ou non – est détectée par le radar, qui transmet un message à l'ordinateur. Ce dernier – piloté par le programme – associe alors à l'objet un symbole graphique qui se matérialise à l'écran. Or, l'affichage a lieu non seulement lorsqu'un avion est effectivement détecté, mais également quand l'écho est renvoyé par une nuée d'oiseaux ou une réflexion parasite. On observe ainsi des avions fantômes (artéfacts) ! Peut-on parler d'une erreur de programmation ?

Pour l'utilisateur, assurément : il ne veut voir ni ce qui ne l'intéresse pas ni, a fortiori, ce qui n'existe pas. Le programmeur ne partage pas du tout son avis. Selon lui, le programme fonctionne parfaitement puisqu'il montre et traite toute information perçue par le radar.

Le plus simple est de considérer comme erreurs toutes les fonctions du programme qui ne sont pas conformes aux spécifications (on entend par spécifications la description des prestations qui devront être assurées par le logiciel, ainsi que la définition de ses limites). Encore faut-il pouvoir se fier aux spécifications, dont le caractère inexact ou incomplet constitue malheureusement la source de bien des erreurs.

Il ressort de tout ceci qu'exception faite des fautes minimes, faciles à éliminer, les erreurs des logiciels ne sont pas uniquement imputables à la programmation mais à un ensemble de causes, au tout premier rang desquelles se trouve, sans nul doute, la mauvaise préparation des spécifications. Pour créer un produit « professionnel », il est donc indispensable d'attribuer au logiciel des limites et des objectifs précis.

Cependant, la préparation la plus soignée et la plus minutieuse des spécifications ne peut pas garantir le logiciel contre toutes les erreurs. Il faut, en effet, compter avec l'impondérable, c'est-à-dire avec le facteur humain. Si l'utilisateur n'est pas familiarisé avec le fonctionnement des ordinateurs, il faudra lui proposer des programmes très didactiques, où la marche à suivre lui sera constamment indiquée. Malheureusement, cela accroît la complexité du logiciel, et donc, le prix et le risque d'erreur. Il faut, une fois de plus, essayer de trouver un juste milieu.

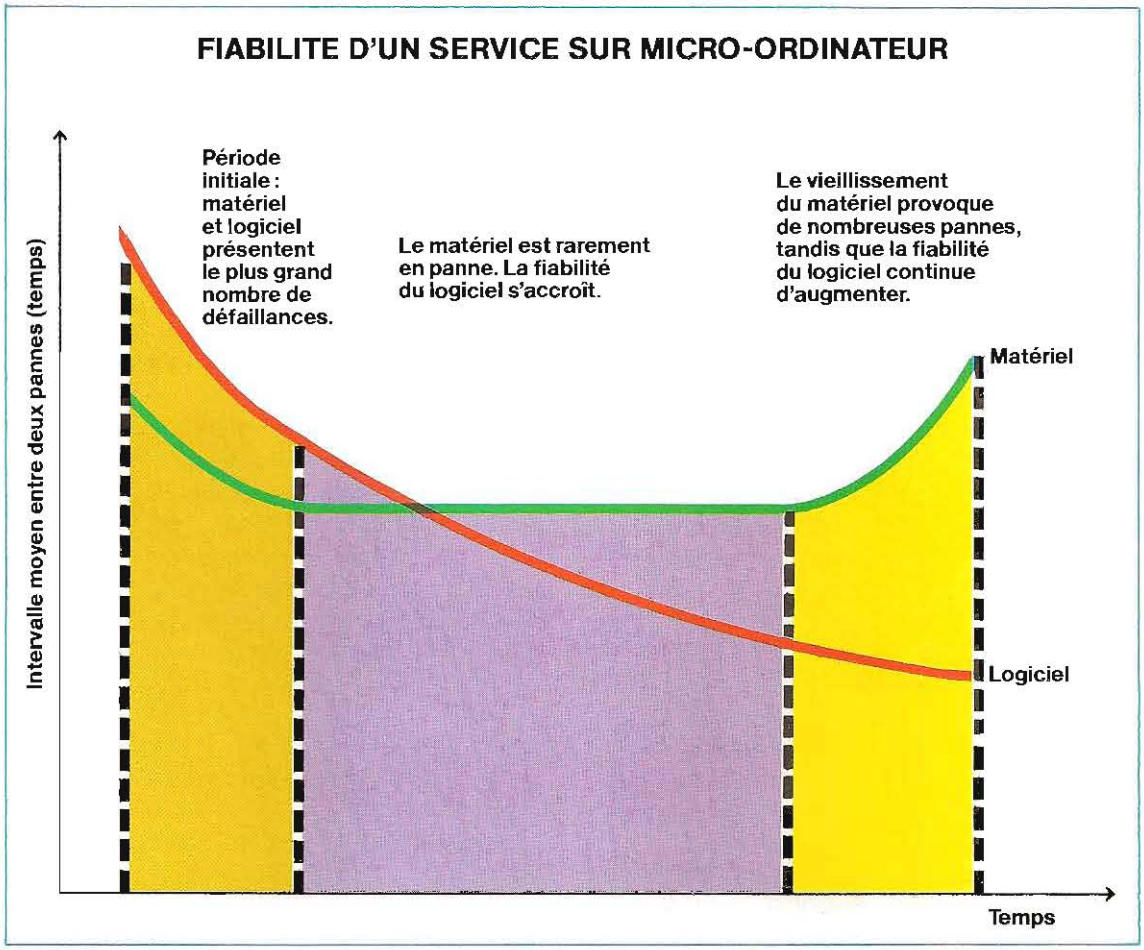
De toute façon, une erreur peut toujours se produire à la suite d'une mauvaise utilisation du logiciel ; il est impensable de prévoir toutes les actions incorrectes et de mettre en place un contrôle pour chacune d'elles. On ne peut protéger un logiciel que contre les erreurs les plus graves ou les plus probables.

Un autre critère doit être pris en compte pour compléter la validation d'un programme : la fiabilité. Appliqué à un logiciel, ce terme désigne la probabilité de son fonctionnement sans défaillance pendant une période déterminée.

La fiabilité d'un logiciel s'accroît avec le temps : d'abord plutôt faible, puisque c'est à la mise en place d'un système que l'on détecte le plus d'erreurs, elle augmente au fur et à mesure que ces erreurs sont corrigées ou que de nouvelles sécurités sont apportées. C'est exactement l'inverse pour la fiabilité du matériel : après une brève phase de rôdage, les pannes demeurent rares pendant une longue période, au terme de laquelle leur fréquence augmente rapidement, du fait du vieillissement des composants.

Le schéma de la page 786 illustre l'évolution parallèle de ces deux sortes de fiabilité, dont la conjugaison donne la fiabilité globale du système, traduite concrètement par un **intervalle moyen entre deux pannes**, dues à une défaillance du matériel ou du logiciel.

On ne peut observer une courbe de fiabilité du logiciel telle que nous l'avons présentée, que si les corrections n'engendrent pas de nouvelles erreurs, ce qui est rare et va de pair avec un programme bien structuré. Il est en effet indispensable, si l'on ne veut pas remplacer systématiquement une erreur par une autre, de conférer aux programmes une structure très linéaire, et d'y insérer de nombreux commen-



taires qui en expliquent, pas à pas, le fonctionnement. Nous reviendrons sur ce type de structure, qui est la condition essentielle d'une maintenance satisfaisante.

Les logiciels constituent rarement des produits statiques. La plupart d'entre eux doivent subir des modifications ou des mises à jour. Les programmes à caractère économique ou administratif, par exemple, doivent souvent être adaptés pour répondre à de nouvelles dispositions. Quant aux logiciels scientifiques, il s'avère parfois nécessaire d'en modifier les algorithmes ou les paramètres. Or, cette maintenance ne se justifie que si les programmes sont suffisamment clairs pour que leur mise à jour reste rentable par rapport à l'écriture de nouveaux programmes : ce n'est malheureusement pas toujours le cas, aussi surprenant que cela puisse paraître, car les programmes sont très souvent mal structurés.

Le schéma de la page 787 montre la part des différentes activités dans le coût total d'un logiciel. Environ 50 % des charges sont imputables à la maintenance, et ce chiffre peut même atteindre 75 % pour les systèmes très complexes. Cette activité reste donc prépondérante d'un point de vue économique.

Les méthodes de test

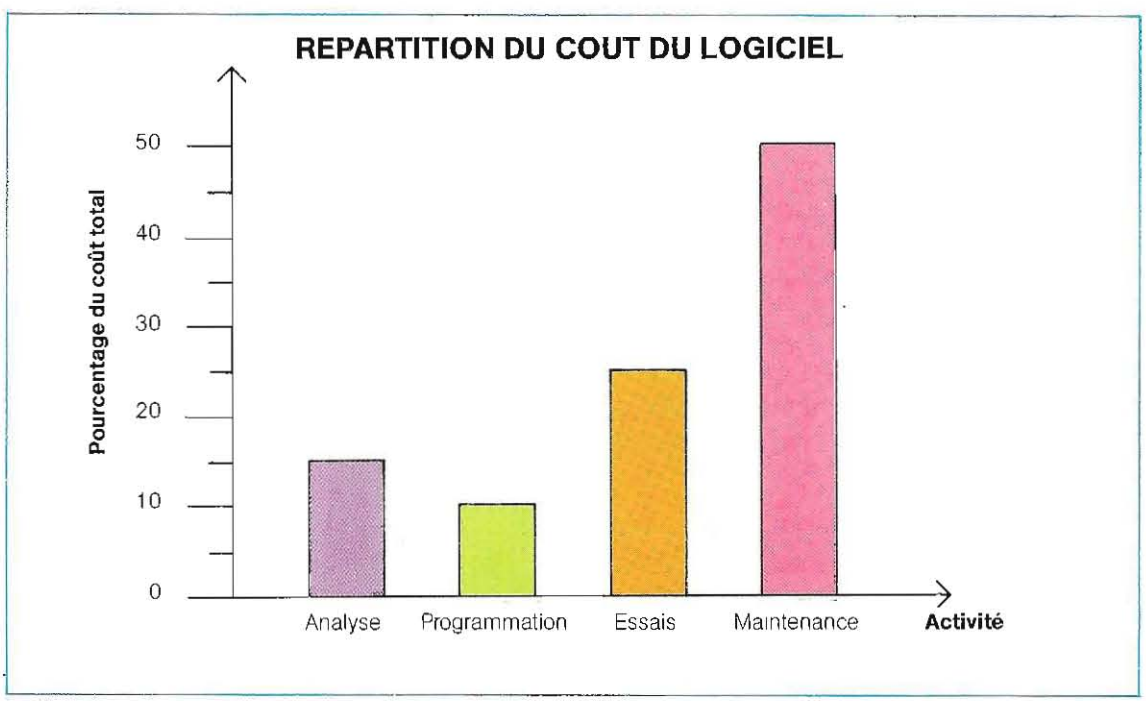
Cette étape de test est la plus délicate et la plus complexe lors de la production d'un logiciel, et cela pour deux raisons essentielles. En premier lieu, c'est une activité mal connue et généralement mal abordée. D'autre part, on la considère très souvent comme une simple démonstration de l'absence d'erreurs dans le logiciel, et l'on s'arrange donc pour éviter soigneusement toutes les opérations qui risqueraient de provoquer des erreurs.

Le but de la vérication d'un logiciel devrait être exactement l'inverse : en faire ressortir les points faibles. Il faut donc s'ingénier à ce qu'ils se manifestent, et le meilleur moyen d'y parvenir est encore de commettre délibérément

des erreurs de manipulation, et même de les exagérer. On peut ainsi évaluer les moyens dont disposera l'opérateur pour rattraper ses erreurs. Il est possible d'organiser les tests d'un logiciel et d'en planifier les étapes en recourant aux mêmes méthodes que pour l'écriture d'un programme.
On définit ainsi dans un schéma les différentes opérations à effectuer, en indiquant pour chacune les données à utiliser, les résultats à obtenir et le niveau de détail souhaité. Même s'il n'est pas très long, un programme comporte presque toujours de nombreuses ramifications, et le chemin emprunté dépend de la façon dont on l'utilise. Explorer tous les itinéraires possibles d'un programme est généralement irréalisable. Ainsi, on a calculé que pour le programme de guidage des missiles TITAN, un tel essai demanderait 60 000 heures de temps machine, soit environ sept ans. En outre, certaines erreurs n'apparaissent que pour des valeurs déterminées de variables numériques. Là encore, la durée nécessaire au test de toutes les combinaisons possibles se compte en années. Seules des méthodes très complexes permettent de surmonter ces difficultés.
Pour les applications réalisées sur micro-ordinateurs, on simplifie les essais et on se limite à tester les principales possibilités, en soignant particulièrement le contrôle des modules de dialogue avec l'utilisateur.
Les principales techniques d'essai sont :

- la méthode ascendante ;
- la méthode descendante ;
- la méthode big-bang ;
- la méthode sandwich ;
- les diagrammes de cause effet.

La méthode ascendante. Un programme se compose généralement d'un ensemble de modules qui s'appellent parfois les uns les autres mais sont, le plus souvent, appelés par des instructions GOSUB regroupées dans un module principal. La méthode ascendante consiste à tester séparément ce module, puis à lui adjoindre un à un les autres modules, en remontant jusqu'au premier. Cette méthode nécessite l'emploi de modules de simulation, qui servent à piloter des parties du programme pendant les essais. Si, par exemple, le module terminal représente un sous-programme d'édition, on a besoin pour le tester d'un module qui initialise les variables à imprimer. Celui-ci sera ensuite remplacé le véritable module de préparation des variables, et sera testé, à son tour, avec un module de simulation assurant les affectations nécessaires.



Ces modules de simulation sont appelés des **pilotes** (en anglais, drivers).

La méthode descendante. Cette méthode fonctionne à l'inverse de la précédente. On commence par tester le module initial (généralement le programme principal), puis on lui ajoute les autres modules dans l'ordre de l'organigramme. Là aussi, il faut prévoir des routines simulant les fonctions des modules qui ne sont pas encore intégrés. On les appelle des **stubs**. Ils ne servent souvent qu'à vérifier que l'appel d'un module s'effectue correctement, et se réduisent alors à une instruction RETURN.
La méthode descendante présente deux inconvénients. D'une part, le contrôle des interfaces avec les parties manquantes exige parfois la simulation de fonctions trop nombreuses, et donc des stubs trop complexes. D'autre part, le module à tester en premier – la partie initiale – est justement celle qu'on ne peut généralement mettre au point qu'en dernier, en fonction des modifications apportées aux autres modules. On peut certes contourner cette difficulté en testant séparément les modules avant leur intégration, mais cela oblige à simuler pour chacun, à la fois l'amont et l'aval. L'essai d'un module isolé suppose, en effet, un driver pour en préparer les entrées et un stub pour en contrôler les sorties.

La méthode big-bang. Elle consiste à tester séparément tous les modules, puis à les articuler avant de procéder à une vérification générale. On ne peut l'appliquer qu'à des programmes de taille relativement réduite, car elle pose vite des problèmes insurmontables.

La méthode sandwich. Elle consiste à appliquer simultanément les méthodes ascendante et descendante, ce qui dispense d'écrire certains modules de simulation. Sa mise en œuvre s'avère relativement délicate : aussi son emploi ne se justifie-t-il que pour les logiciels très complexes.

Les diagrammes de cause effet. Vérifier que tous les algorithmes fonctionnent dans la plupart des situations qui peuvent se présenter constitue l'une des tâches les plus ardues des tests. Il faut donc l'aborder avec beaucoup de méthode, afin d'éviter qu'elle ne dure trop longtemps et que le choix des paramètres ne soulève des difficultés insurmontables.
Les diagrammes de cause effet servent essentiellement pour la définition de la stratégie à adopter.
On effectue une analyse du contenu « sémantique » des spécifications, puis on en résume les conclusions dans des diagrammes. Ces derniers permettent ensuite de construire une sorte de trame, la **table de décisions**, qui indique les cas à tester.
On procède par étapes successives :

- décomposition des spécifications en opérations élémentaires ;
- analyse de chaque opération dans le but de déterminer toutes les **causes** possibles (entrée) et tous les **effets** possibles (sortie) ;
- établissement du diagramme ;
- codification des diagrammes sous forme de table de décisions.

On a recours, pour le tracé des diagrammes, à une symbolisation apte à représenter toutes les possibilités de relation élémentaire entre cause et effet (voir schéma page 789).
Pour illustrer cette méthode, nous allons l'appliquer à un sous-programme de conversion en majuscule de la première lettre d'une chaîne de caractères. Les causes envisageables sont ici les suivantes :

a le premier caractère est un nombre ;
b le premier caractère est une majuscule ;
c le premier caractère est une minuscule ;
d le premier caractère n'est pas de type alphanumérique.

Voici maintenant les effets possibles :

x la chaîne n'est pas modifiée (cause a) ;
x la chaîne n'est pas modifiée (cause b) ;

y le premier caractère est transformé en majuscule (cause c) ;
z un message d'erreur est émis (cause d).

Par ailleurs,

1 / s'il y a une erreur , l'exécution s'arrête ;
2 / s'il n'y a pas d'erreur, l'exécution se poursuit.

Le diagramme de décisions correspondant se trouve page 789 (second schéma). Les cau-

PRINCIPAUX SYMBOLES UTILISES DANS LES DIAGRAMMES DE DECISION

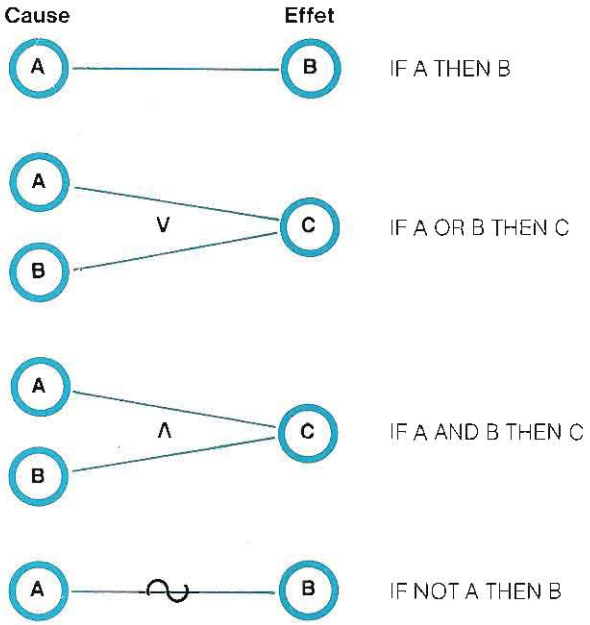
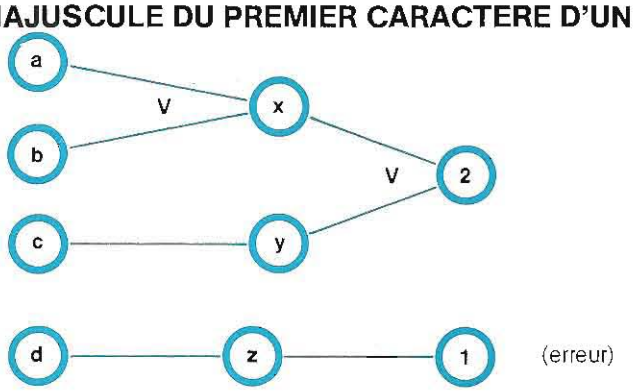


DIAGRAMME DE DECISION DU PROGRAMME DE CONVERSION EN MAJUSCULE DU PREMIER CARACTERE D'UNE CHAINE



ses ont été indiquées à gauche et les effets à droite. Pour construire la table de décisions, il suffit de considérer chaque effet et de noter toutes ses causes possibles.
On obtient ainsi un schéma qui indique clairement toutes les éventualités à envisager au cours des tests (voir page 790).
Cette table devient indispensable pour les problèmes d'une certaine complexité.

La mise au point

Les tests mettent en évidence les erreurs à éliminer : c'est la mise au point (**debugging**). En fait, ces deux phases (détection et élimination) sont étroitement liées.
Quelle que soit la méthode adoptée pour les tests, il faut les interrompre dès qu'une erreur se produit pour rechercher et corriger sa cause, avant de reprendre les essais.
Cette étape de mise au point nécessite également le respect d'une méthode précise et rigoureuse.
La technique la plus courante s'appelle tout simplement « la méthode » (the method). Elle consiste à créer un tableau à l'aide de trois critères principaux :

TABLE DE DECISIONS

Cause	Effet				
	x	y	z	1	2
a	OUI	—	—	—	OUI
b	OUI	—	—	—	OUI
c	—	OUI	—	—	OUI
d	—	—	OUI	OUI	—

OUI = Cas à envisager dans les tests
— = Cas à ne pas envisager

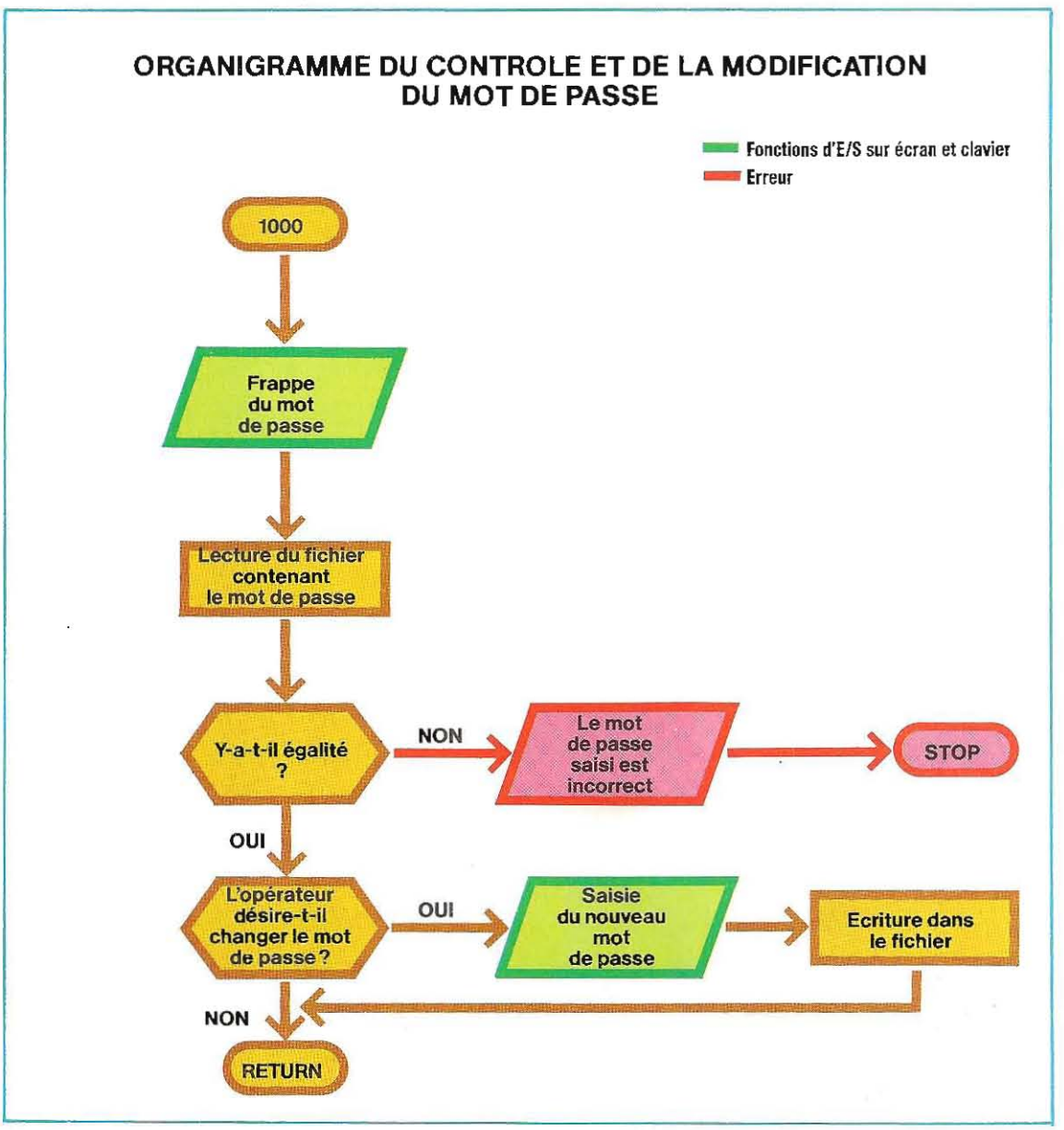
- What (quoi) : symptôme de l'erreur.
- Where (où) : endroit du programme où l'erreur se produit.
- When (quand) : conditions dans lesquelles l'erreur se produit.

On définit parfois un quatrième facteur, la finalité de l'erreur, mais uniquement pour les problèmes réellement très complexes. Ces trois ou quatre éléments serviront de guide lors de la recherche des erreurs.
En Basic, la mise au point est grandement facilitée par la possibilité de travailler en mode immédiat. Quand une erreur se produit, on peut arrêter l'exécution et demander la présentation de toutes les variables, afin de savoir lesquelles prennent, éventuellement, des valeurs erronées.
En revanche, lorsqu'on travaille avec un langage exclusivement compilé, on ne peut effectuer cette vérification qu'en insérant dans le programme lui-même les instructions nécessaires à l'affichage, et en demandant qu'il soit à nouveau compilé, puis exécuté. Ce n'est qu'au terme de ce long processus qu'on accède aux valeurs des variables incriminées. En outre, il est parfois nécessaire de répéter plusieurs fois cette opération avant de localiser l'erreur. Si le Basic interprété se voit souvent reprocher sa lenteur d'exécution, sa souplesse de travail en mode dialogué en fait le langage de choix des débutants.
C'est pourquoi les systèmes d'exploitation proposent de plus en plus souvent des programmes d'aide à la recherche des erreurs.
On appelle ces programmes des **debuggers** (debugging tools). Ils renseignent automatiquement le programmeur sur le contenu des zones mémoire utilisées par le programme (valeurs prises par les variables). Les plus perfectionnés l'autorisent même à modifier le contenu de ces zones, ce qui lui permet d'essayer différentes possibilités pour mieux localiser l'erreur.
L'inconvénient de ces programmes est qu'ils impliquent la connaissance de l'Assembleur. Les programmes compilés sont chargés en mémoire sous forme binaire. Une fois les instructions traduites en langage machine, les variables sont donc définies par leur adresse, et non plus par leur nom symbolique du programme source. Mis à part ceux destinés à de gros systèmes, les debuggers ne travaillent qu'avec cette représentation.
Sur les systèmes les plus perfectionnés, il existe des debuggers spéciaux avec lesquels le programmeur peut dialoguer pour mieux diriger ses recherches : exécution en pas à pas, mode trace, points d'arrêt.
Certains d'entre eux, l'EXDAMS (Extendable Debugging And Monitoring System) par exemple, enregistrent sur bande magnétique tout ce qui se passe pendant l'exécution. Le programmeur dispose alors d'une source précieuse d'informations sur le fonctionnement de ses logiciels. En outre, il peut demander à tout moment l'affichage des variables.
On peut ainsi examiner un programme à l'envers, ou revenir sur un moment précis de l'exécution. Toutefois, il n'est possible d'effectuer aucune correction puisque l'examen n'a lieu qu'après la fin de l'exécution du programme.
Certains langages possèdent de véritables instructions de mise au point, qui leur sont toujours extrêmement spécifiques. Ainsi, dans un programme en PL/1, l'instruction CHECK (qui signifie d'ailleurs « contrôle ») générera, dans certaines conditions, l'interruption momentanée du programme et la visualisation d'un message. Insistons sur le fait que ces différentes aides à la mise au point sont très peu utilisées dans le secteur de la micro-informatique.

La sécurité du logiciel

Plus le nombre des données s'accroît, plus il devient capital d'assurer leur sécurité. Tandis que, sur les gros systèmes, les données sont protégées par des mécanismes très complexes qui font notamment appel aux ressources du matériel, on est obligé, sur les micro-ordinateurs, de prévoir leur protection à l'intérieur même du programme d'application.
Pour protéger les données contre les pannes du matériel, il suffit de copier périodiquement les disques qui les contiennent (copies de sauvegarde). On peut contrôler la régularité des sauvegardes en prévoyant dans le logiciel un test sur la date.
La prévention des erreurs de manipulation s'avère beaucoup plus délicate. Au premier niveau, on peut subordonner l'accès au programme ou à certaines parties du programme, à la reconnaissance d'un mot de passe, connu des seules personnes autorisées. D'autres mesures de sécurité consistent à demander confirmation à l'utilisateur, avant de procéder à toute suppression ou modification de fichiers. On évite ainsi d'écraser des données, en attirant l'attention de l'opérateur sur un ordre qui peut constituer

ORGANIGRAMME DU CONTROLE ET DE LA MODIFICATION DU MOT DE PASSE



éventuellement une erreur de manipulation.
Page 791, figure l'organigramme simplifié d'une routine de contrôle du mot de passe, mise en œuvre dans le programme listé ci-dessous. On remarquera que la saisie du mot de passe est demandée à la ligne 1200 par l'instruction INPUT$. Ainsi, le mot de passe n'apparaît pas sur l'écran, et ne peut être déchiffré par d'éventuels observateurs. Pour assurer une protection efficace, ce sous-programme doit être mémorisé sous forme compilée : autrement, il suffit, pour déjouer le contrôle, de chercher, dans le listing du source, le nom du fichier où le mot de passe est mémorisé. On peut alors le lire, voire le modifier.
En revanche, la compilation garantit contre toute « indélicatesse », y compris venant du personnel expérimenté, si l'on a recours à quelques astuces. On peut par exemple masquer le mot de passe avant son transfert dans le fichier, afin qu'il soit impossible de le lire, à moins de connaître la valeur du masque utilisé. Cette valeur sera très difficile à décrypter dans un programme compilé.

PROGRAMME DE CONTROLE DES MOTS DE PASSE

```
10  ' **** PROGRAMME PRINCIPAL ****
12  INPUT "VOULEZ-VOUS INITIALISER LE FICHIER?";REP$
14  IF REP$<>"OUI" GOTO 95
20  OPEN "R",1,"A:MP",5
30  FIELD 1,5 AS M$
40  A$=SPACE$(5)
50  LSET M$=A$        ' Initialisation a blanc du buffer d'E/S
55  ' *** Initialisation des 9 premiers enregistrements du fichier
60  FOR R%=1 TO 9
70  PUT 1,R%
80  NEXT R%
90  CLOSE 1
95  GOSUB 1000
100 END
1000 '
1010 ' ** ROUTINE DE CONTROLE DES MOTS DE PASSE **
1020 K=0 : ORG=1
1030 PRINT "ENTREZ VOTRE CODE"
1040 ' Chaque operateur possede un code numerique qui indique
1050 ' dans quel enregistrement est memorise son
1060 ' mot de passe personnel
1070 A$=INPUT$(2)        ' Saisie sans echo
1080 IF VAL(A$)=0 OR VAL(A$)>10 GOTO 1300        ' Traitement des erreurs
1090 ' Seuls sont prévus les codes compris entre 1 et 10
1100 OPEN "R",1,"A:MP",5        ' MP = fichier des mots de passe :
1110 '                            enregistrements de 5 caracteres
1120 FIELD 1,5 AS M$
1130 R%=VAL(A$)        ' Numero d'enregistrement de l'operateur
1140 B$=SPACE$(5)
1150 GET 1,R%        ' Lecture du mot de passe dans le fichier
1160 LSET B$=M$
1170 IF B$=SPACE$(5) GOTO 1240        ' La premiere fois, l'enregistrement est vide
1180 '                                  et il faut y ecrire le mot de passe
1190 PRINT "TAPEZ LE MOT DE PASSE"
1195 ORG=2
1200 A$=INPUT$(5)        ' Saisie sans echo
1210 IF A$<>B$ GOTO 1300
1220 INPUT "VOULEZ-VOUS CHANGER DE MOT DE PASSE?";REP$
1230 IF REP$<>"OUI" GOTO 1350
1240 PRINT "TAPEZ VOTRE NOUVEAU MOT DE PASSE"
1250 A$=INPUT$(5)        ' Saisie sans echo
1260 LSET M$=A$
1270 PUT 1,R%        ' Ecriture sur disque
1280 GOTO 1350        ' Sortie
1290 ' *** TRAITEMENT DES ERREURS ***
1300 PRINT "** MOT DE PASSE INCORRECT **"
1310 IF K=1 THEN STOP
1320 PRINT "NOUVEL ESSAI"
1330 K=1
1340 ON ORG GOTO 1030, 1150        ' Retour d'erreur
1350 CLOSE 1 : RETURN
```

Les logiciels d'application généralisée

L'usage de l'ordinateur dans l'activité professionnelle n'est pas aussi fréquent qu'on pourrait le penser. En analysant les tâches habituellement dévolues à la machine, on distingue deux activités majeures :

— la gestion des fichiers ;
— le traitement des données.

Des similitudes se rencontrent fréquemment entre diverses applications. Même si, d'une fois à l'autre, les données et les calculs à exécuter varient, la logique, et donc la structure du programme, demeurent inchangées.
Par exemple, les fonctions qu'un programme de gestion d'un répertoire téléphonique doit exécuter se résument à la saisie et à la validation des données, à la mémorisation sur disque et à la recherche. Ces opérations peuvent être réalisées par des modules (saisie des données, gestion de disque, etc.) gérés d'une façon appropriée. Pour adapter ce logiciel à la gestion d'une bibliothèque, il suffira de modifier la structure des données : toutes les fonctions resteront inchangées et l'on pourra appeler les mêmes modules de traitement.
Ces considérations ont amené les concepteurs de logiciels à orienter leurs programmes vers les deux fonctions de base : traitement et gestion des fichiers.
L'évolution de ce type de programmes a conduit à des logiciels très complets, qui réunissent en un seul « package » tous les modules de base et qui, de plus, contiennent des fonctions spéciales comme la gestion de graphiques.

Le tableur

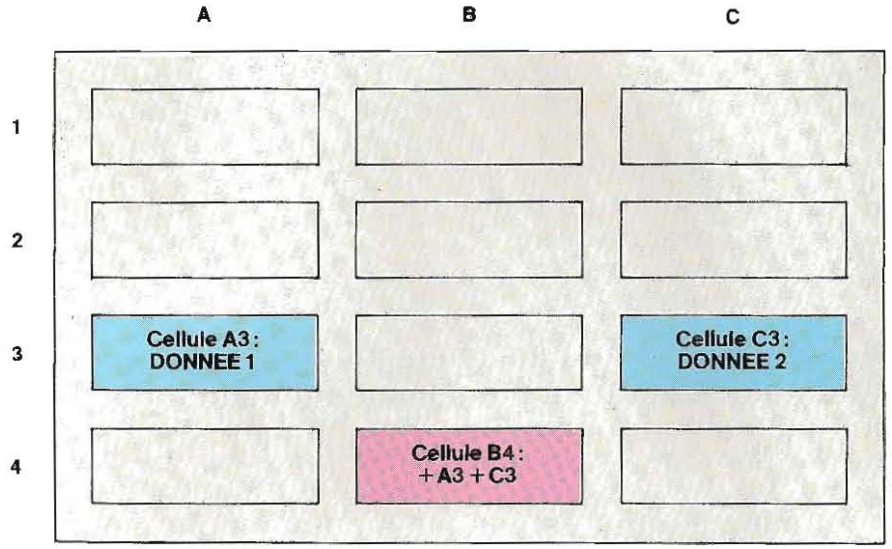
Les programmes dont le but principal consiste à traiter des données (Visicalc, Multiplan, Lotus, par exemple) possèdent tous une même structure dite de **feuille de calcul**, qui permet d'utiliser la mémoire de la machine à la manière d'une feuille de papier, divisée en lignes et en colonnes.
L'intersection d'une ligne et d'une colonne définit un élément dénommé **cellule**, contenant une donnée (numérique ou de chaîne) ou, au contraire, une expression.

Utilisation de l'ordinateur aux Olympiades de Moscou de 1980.

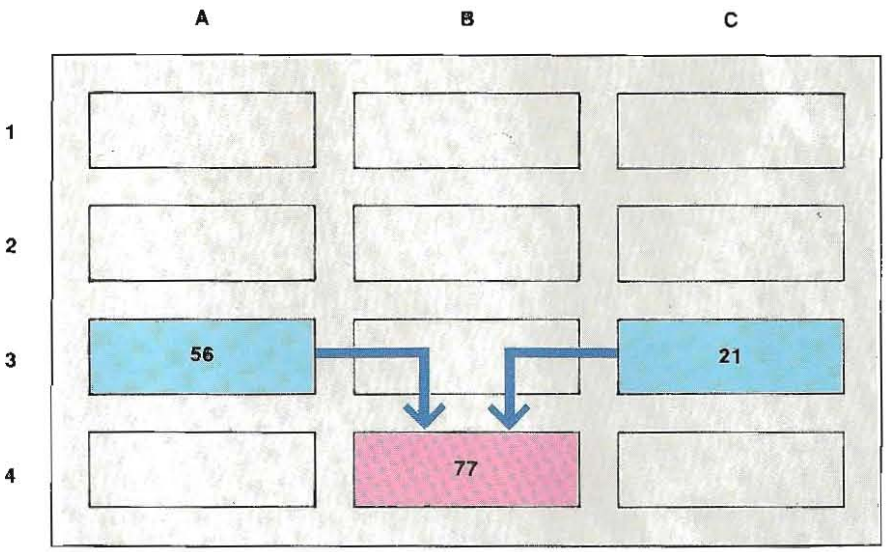


M. Vuorela/Marka

CALCUL D'UNE SOMME PAR TABLEUR

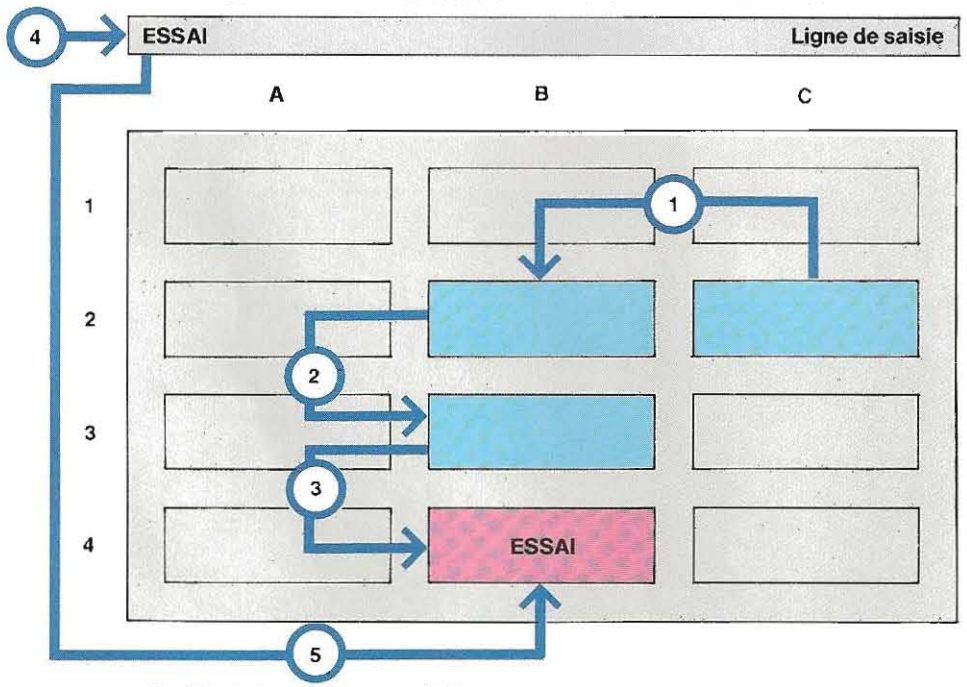


La cellule B4 contient l'indication du calcul à exécuter :
+A3 +C3 signifie « afficher la somme des données contenues dans les cellules A3 et C3 ». Après la saisie de deux valeurs numériques dans les cellules A3 et C3, on obtient le calcul automatique et l'affichage de la somme demandée dans la cellule B4.



Le contenu de chaque cellule peut être modifié à tout moment ; pour celles qui contiennent une formule, le résultat est recalculé immédiatement.
Le schéma en page 794 représente un tableur : chaque colonne est désignée par une lettre de l'alphabet et chaque ligne par un numéro. La capacité de la cellule se limite d'ordinaire à 8 ou 9 caractères, mais peut être modifiée au moyen d'instructions appropriées.
Ce schéma illustre les deux utilisations principales d'une cellule : mémorisation de données et stockage d'expressions. Les cellules A3 et C3 (qui devront contenir les données à traiter) sont vides au début, alors que l'expression (par exemple + A3 + C3) est écrite dans la cellule B4. Le programme détecte la présence d'une expression dans B4. A l'affichage, il fait apparaître non le contenu de la cellule (qui serait + A3 + C3), mais le résultat du calcul appliqué aux données.
Au début, la cellule B4 contiendra la valeur 0 ; son contenu évoluera automatiquement dès qu'on modifiera les valeurs A3 et C3.
Au moyen de ces fonctions et des nombreuses autres qui seront présentées par la suite, on peut mettre en place un calcul et l'exécuter de manière récursive : il suffit d'entrer les nouvelles données. Aucune préparation spéciale n'est requise de l'utilisateur. Il lui suffit d'établir les différentes opérations comme s'il voulait les effectuer à la main ; le programme s'adapte en conséquence et fait tous les calculs nécessaires.
Le nombre de lignes et de colonnes affichées dépend de la largeur de chaque cellule et de celle de l'écran vidéo. Un écran standard possède 80 colonnes et 24 lignes, mais certains ne comportent que 40 colonnes : seule la moitié du tableau sera alors visible. Pour visualiser les cellules situées par exemple à droite de celles en cours d'affichage, il suffit de déplacer le curseur, comme si l'on voulait le faire sortir par la marge droite de l'écran. Les premières colonnes de gauche (A, B, etc.)

SAISIE D'UNE DONNEE DANS UNE CELLULE

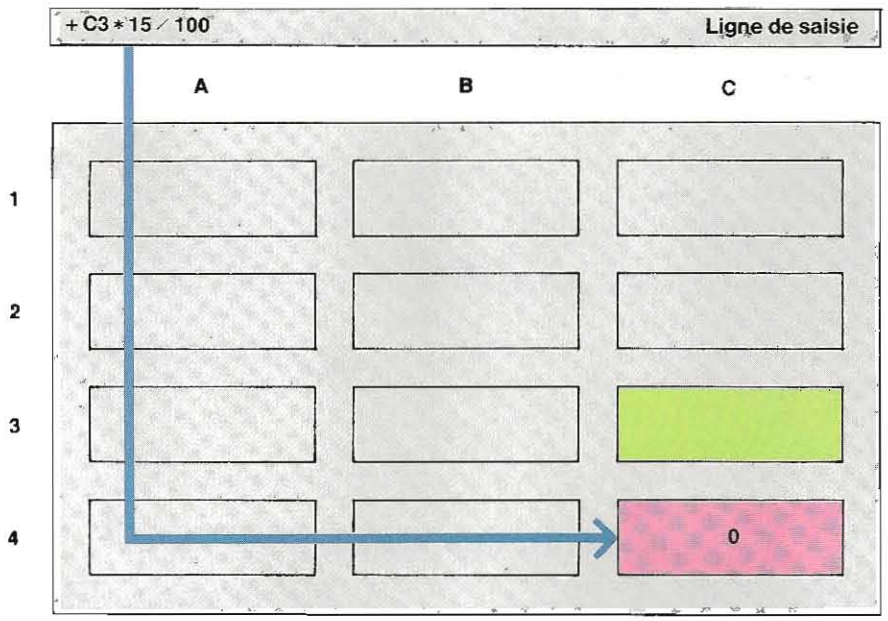


1 Position initiale du curseur (C2).
Premier déplacement : positionnement horizontal (touche ←).
2-3 Après le positionnement horizontal, deux déplacements verticaux consécutifs (espace + ←, espace + ←) amènent le curseur à la position voulue (B4)
4 Saisie de la donnée.
5 Toute donnée saisie est automatiquement transférée dans la cellule pointée par le curseur.

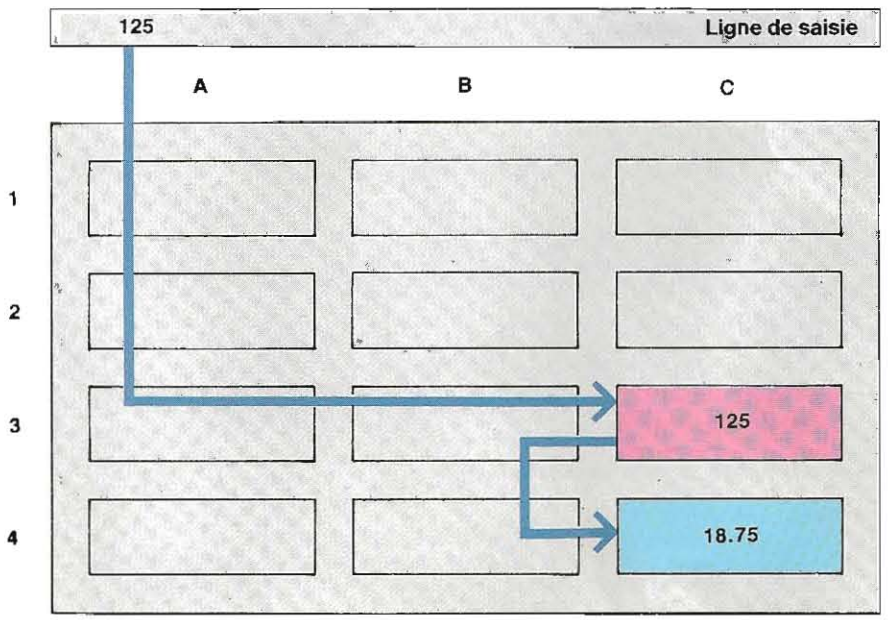
SAISIE D'UNE EXPRESSION

Cellule de la donnée
Cellule pointée par le curseur
Cellule de l'expression

EXACT



Dès que la saisie de l'expression a été effectuée, la valeur 0 apparaît dans la cellule C4 ; au début, le contenu de la cellule C3 est nul par défaut.

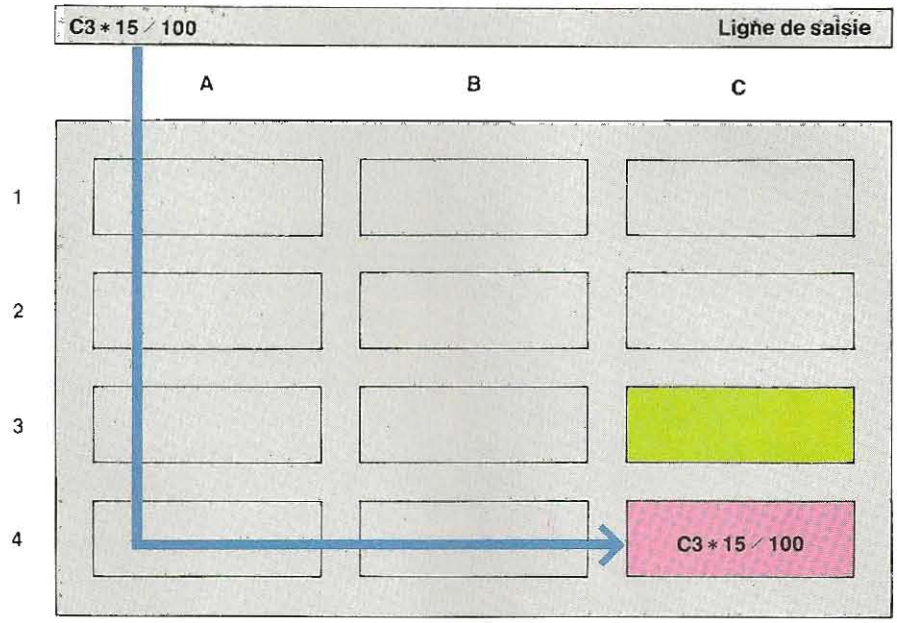


Après la saisie d'une valeur dans la cellule C3, le calcul s'exécute automatiquement et le résultat s'affiche en C4.

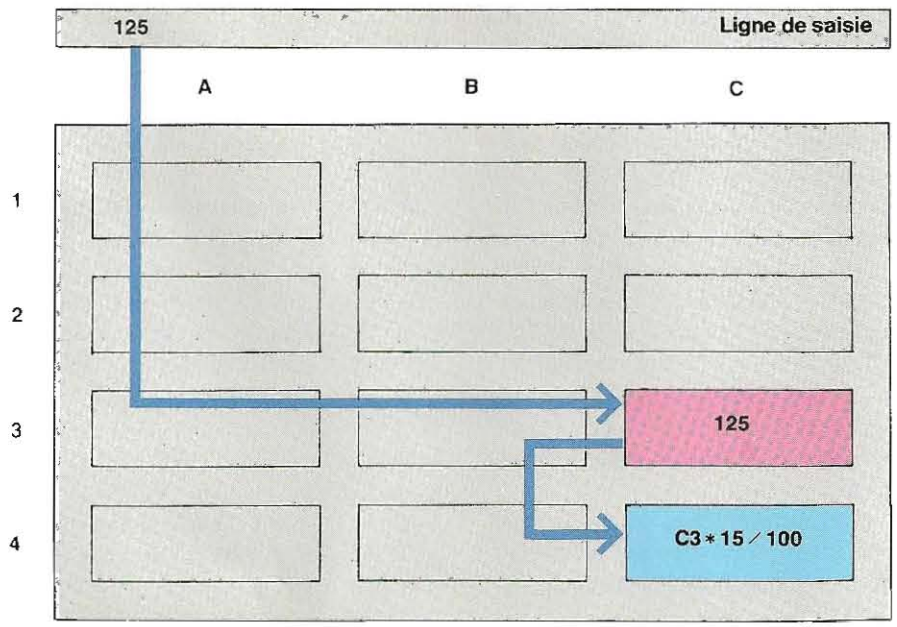
SAISIE D'UNE FORMULE

Cellule de la donnée
Cellule pointée par le curseur
Cellule de l'expression

ERRONE



L'omission du symbole + engendre une erreur d'interprétation : la ligne saisie est considérée comme un titre et simplement réécrite dans la cellule pointée par le curseur.



Une saisie effectuée dans la cellule C3 n'engendre aucune modification du contenu de la cellule C4, car cette dernière ne contient pas d'expression.

vont alors disparaître, laissant la place à de nouvelles colonnes à droite. De la même façon, on peut déplacer verticalement la feuille entière. Cette évolution de la position dans le tableau est appelée **scrolling** (défilement).
Les programmes qui gèrent la feuille ne présentent pas le curseur sous sa forme habituelle (tiret ou rectangle) mais affichent une zone de dimension égale à celle de la cellule, qui est mise en évidence par affichage inversé. Déplacer le curseur signifie positionner ce rectangle sur la cellule voulue. On peut le faire de deux façons : directement vers la cellule d'arrivée, au moyen d'une commande appropriée, ou bien en le déplaçant cellule par cellule à travers la feuille.
La direction du curseur est gérée par certaines touches de fonction. Si le clavier les possède, on utilise les touches habituelles de déplacement, désignées par les symboles ←, →, ↑, ↓ ; sinon, des instructions spéciales sont prévues. Les programmes de gestion des feuilles, destinés à un usage généralisé, ne font donc aucune référence au matériel employé. Dans certains cas, la machine ne possède pas un clavier complet ; le programme s'adapte donc à ces situations spécifiques. Ainsi, sur la machine employée pour le développement des applications présentées dans ce chapitre (SIPREL modèle 2040), les déplacements verticaux s'obtiennent par des séquences [espace, touche ←] pour les déplacements vers le bas et [espace, touche →] pour les déplacements vers le haut.
L'autre manière de déplacer le curseur consiste à employer une commande simple, spécifiant la case d'arrivée. Ainsi, l'instruction >A7 déplace le curseur depuis sa position courante jusqu'à la cellule désignée par A7 (colonne A, ligne 7). Sur d'autres types de programmes, la commande est différente : avec Multiplan, par exemple, elle sera G (= GO) suivi des coordonnées. Comme pour tous les logiciels, des différences formelles existent entre ces programmes ; les commandes s'expriment avec des symboles variés, mais assurent les mêmes fonctions.
Le déplacement direct du curseur n'est qu'une des multiples commandes possibles du tableur, dont la complexité est tributaire du type de logiciel.
Quel que soit celui qu'on utilise, une des lignes de l'écran est généralement destinée à la saisie des données, et une ou plusieurs autres lignes sont réservées aux commandes. Chaque donnée entrée au clavier s'affiche sur la ligne réservée à cet effet, puis se place dans la cellule pointée par le curseur ; s'il s'agit d'une commande, celle-ci s'exécute sans que rien n'apparaisse sur la ligne réservée aux données.

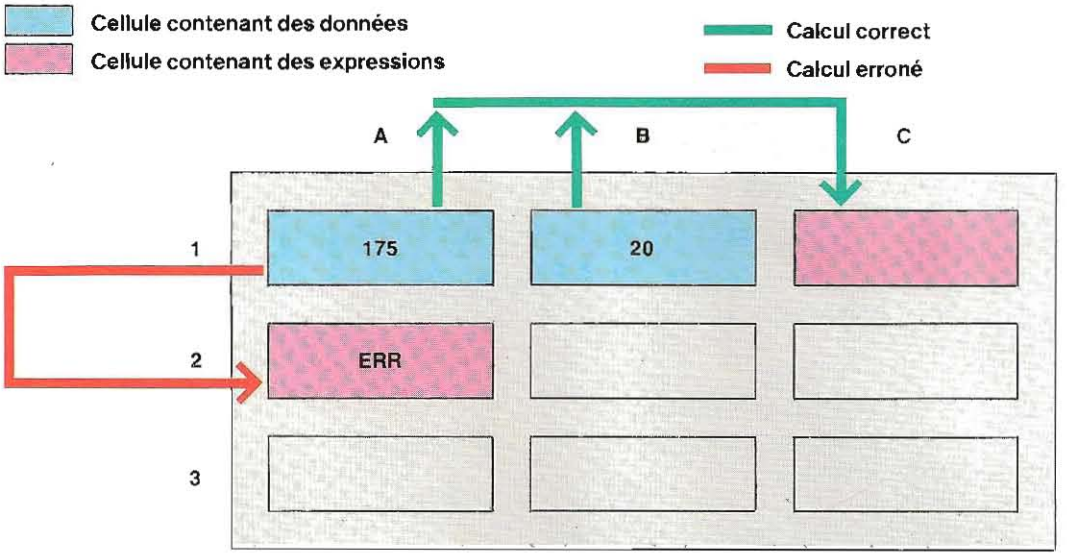
La saisie des données

La page 795 retrace la séquence des opérations à accomplir pour placer une donnée dans la cellule B4, en partant avec le curseur positionné en C2.
La première étape consiste à positionner le curseur dans la cellule B4. On peut alors entrer la donnée, qui apparaît sur la ligne réservée au fur et à mesure qu'elle est tapée. Une fois la saisie achevée, la donnée n'apparaît pas encore dans sa cellule de destination, parce que celle-ci est encore occupée par le curseur. En libérant la cellule par un déplacement quelconque du curseur, la donnée saisie apparaîtra.
Dans cet exemple, la donnée entrée est une chaîne de caractères et ne pourra donc faire l'objet d'un calcul ; si l'on écrivait, dans une autre cellule, une expression utilisant le contenu de B4, un message d'erreur apparaîtrait. Au contraire, si B4 contenait une valeur numérique, tout calcul s'effectuerait normalement. Par exemple, si l'on veut obtenir dans la cellule C4, 15 % d'un nombre contenu dans C3, l'expression à mémoriser dans C4 est :

$$+C3 * 15/100$$

L'expression commence par le symbole +, qui indique à la machine que ce qui suit est une expression et non une chaîne de caractères. Si l'on écrivait la même expression en omettant le symbole +, elle serait interprétée comme une chaîne alphanumérique quelconque, donc stockée dans la cellule d'arrivée sans être reconnue comme un calcul à exécuter.
Les pages 796 et 797 retracent les opérations à exécuter pour introduire une expression, et mettent en évidence ce type d'erreur. Le déroulement d'un calcul, s'il a été mémorisé de façon correcte, s'exécute automatiquement au moment de la saisie, et selon une priorité définie. La feuille comporte deux dimensions (lignes et colonnes) :

EXEMPLE D'ERREUR ENGENDREE PAR LE CHOIX D'UN SENS DE CALCUL ERRONE



La cellule C1 contient l'expression + A1 – B1, dont le résultat est utilisé dans la cellule A2.
La cellule A2 contient l'expression + A1/C1.
Si on effectue les calculs par colonne, on tente d'exécuter le calcul en A2, alors que celui de C1 n'a pas encore été effectué.
Le contenu de C1 est 0, ce qui engendre en A2 une erreur de division par 0.
Le calcul exact doit s'effectuer par lignes.

tout calcul peut se dérouler en accordant une priorité à l'une des deux directions. En effectuant le calcul par lignes, toutes les expressions figurant sur la première ligne (A1, B1, etc.) seront exécutées en premier, puis celles de la ligne 2 (A2, B2, etc.) et ainsi de suite. Au contraire, en effectuant le calcul par colonnes, on aura la séquence :

A1, A2, A3, etc.
B1, B2, B3, etc.

Dans l'établissement des expressions, il faut tenir compte de l'ordre d'exécution des calculs (la priorité peut être choisie au moyen d'une commande), sinon des erreurs pourraient se produire, à cause de valeurs non encore calculées.
L'exemple illustré ci-dessus utilise un résultat intermédiaire : dans la cellule A2, on se réfère au contenu de C1 qui, à son tour, est le résultat d'un calcul précédent. Le sens des calculs doit se faire par lignes, de façon à arriver à la cellule A2 avec un contenu correct de C1. Dans cet exemple, l'erreur commise en procédant dans l'autre sens (par colonnes) est due à ce que, dans la cellule C2, le diviseur est encore nul (le contenu de A2 est calculé avant C1, et C1 contient 0 par défaut). Le système détecte l'anomalie (un nombre ne peut être divisé par zéro) et émet un diagnostic. Si, dans la cellule A2, l'expression avait été par exemple une addition, le système n'aurait pas relevé d'erreur et un résultat erroné aurait été calculé sans aucun diagnostic.
Pour modifier l'ordre d'exécution des calculs, il faut appeler une commande, en choisissant son mot associé dans un menu qui apparaît sur une zone réservée de l'écran. Dans certains logiciels, la zone attribuée au menu est la seconde ligne de l'écran, pour d'autres la dernière, ou les dernières si les commandes sont particulièrement nombreuses.
Dans tous les cas, il y a deux modes de fonctionnement : le **mode exécution** et le **mode commandes**.
En mode exécution, on peut saisir des données, exécuter des calculs ou des fonctions particulières (édition, stockage sur disque, graphiques). Le mode commandes sert à

définir les conditions de travail qui seront en vigueur jusqu'à la prochaine modification.

Le mode commandes

Au début, le tableur (logiciel, programme) se trouve habituellement sous le mode exécution, avec le curseur placé sur la cellule en haut à gauche (A1). Pour passer en mode commandes, on frappe une touche de fonction particulière. L'activité de ce code a pour effet l'apparition du menu des commandes. Le schéma de la page 801 illustre un exemple de tableur en mode commandes.
Les indications présentées sont d'ordinaire les suivantes :

— position du curseur (dans l'exemple, B1) ;
— modalité de déroulement des calculs ;
— menu des commandes.

Les deux premiers indicateurs renseignent l'utilisateur sur la situation en vigueur, alors que le menu ne constitue qu'une liste le guidant dans son choix. Notre schéma présente un menu typique, commun à plusieurs versions de tableurs. D'un logiciel à l'autre, on peut noter des différences entre les identifications des différentes commandes, mais les fonctions exécutées sont presque similaires. Dans la version présentée, les symboles prennent les significations suivantes :

B BLANK — Met à zéro le contenu de la cellule pointée par le curseur.

C CLEAR — Met à zéro la feuille entière.

D DELETE — Supprime entièrement une ligne ou une colonne.

E EDIT — Permet de corriger le contenu d'une cellule.

F FORMAT — Appelle un sous-menu établissant le format de présentation des données dans la cellule concernée.

G GLOBAL — Appelle un sous-menu modifiant le format de présentation des données pour toute la feuille.

I INSERT — Insère, dans la feuille, une ligne ou une colonne.

M MOVE — Déplace une ligne ou une colonne d'un endroit à un autre.

P PRINT — Etablit les paramètres d'édition.

R REPLICATE — Recopie une zone de la feuille (source) dans un autre (but).

S STORAGE — Mémorise la structure de la feuille sur un fichier disque ou récupère une feuille précédente.

T TITLE — Appelle un sous-menu permettant l'affichage de titres, lignes ou colonnes qui resteront inchangées.

W WINDOW — Crée une fenêtre écran.

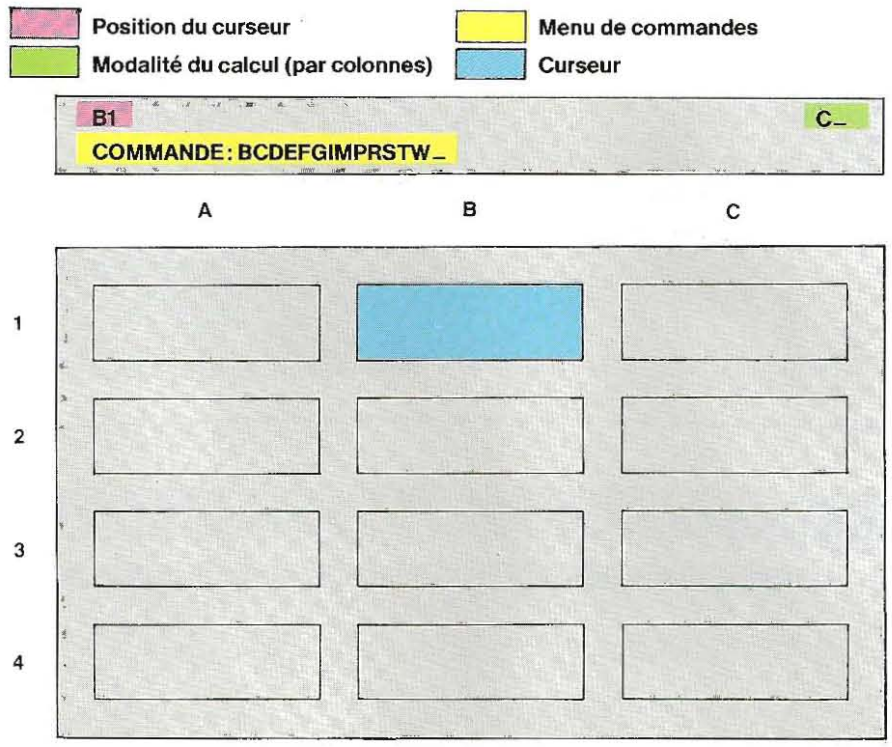
_ REPEAT — Répète le caractère alphanumérique spécifié, sur toute la longueur de la cellule.

Pour la plupart de ces commandes, la description de la fonction exécutée suggère d'elle-même les modalités d'emploi. D'autres commandes plus complexes possèdent aussi des options.
Nous allons présenter les plus fréquentes, en gardant à l'esprit que pour d'autres tableurs elles seront représentées par des symboles différents.

FORMAT : voici les options habituellement prévues :

D Retourne au format d'origine.
G Exécute les calculs avec le maximum de précision permis.
I Ne présente que la partie entière des valeurs.
L Justifie à gauche le contenu des cellules.
R Justifie à droite le contenu des cellules.
$ Ecrit les valeurs numériques avec deux décimales.
* Remplace les valeurs numériques par un nombre d'astérisques proportionnel à cette valeur. Par cette commande, on peut obtenir des représentations graphiques (très rudimentaires, il est vrai). Avec d'autres types de logiciels, il existe des options graphiques plus complètes, qui permettent l'obtention de graphiques x–y à barre, ou en cercle.

EXEMPLE DE PRESENTATION D'UN MENU DE COMMANDES



Les pages 802 à 805 illustrent quelques exemples montrant l'utilisation de la commande FORMAT, avec quelques-unes de ses options principales.

GLOBAL. Les principales options disponibles de cette commande sont :

C Change la largeur de la colonne. Le nombre de caractères contenus dans chacune des cellules peut être modifié dans de larges limites, ce qui permet d'adapter le tableau aux contraintes particulières de chaque application.

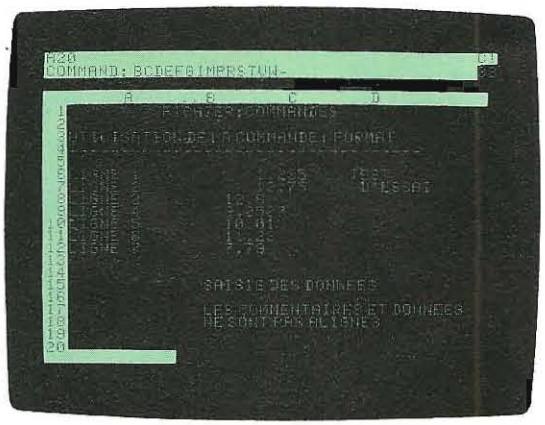
R Permet de choisir le mode de calcul : manuel ou automatique. Dans ce dernier cas, la feuille est recalculée à chaque donnée saisie. En mode manuel, le système ne procède au recalcul qu'après la reconnaissance d'un accord (par exemple, le symbole !). Cette option sert principalement lors d'une saisie massive de données. En mode automatique, le recalcul de la feuille s'effectuerait à chaque donnée entrée, avec une perte de temps notable pour l'utilisateur. Grâce à l'option manuelle, on peut entrer, sans attente, les données les unes à la suite des autres. A la fin, par la commande appropriée, on recalcule la feuille dans son entier.

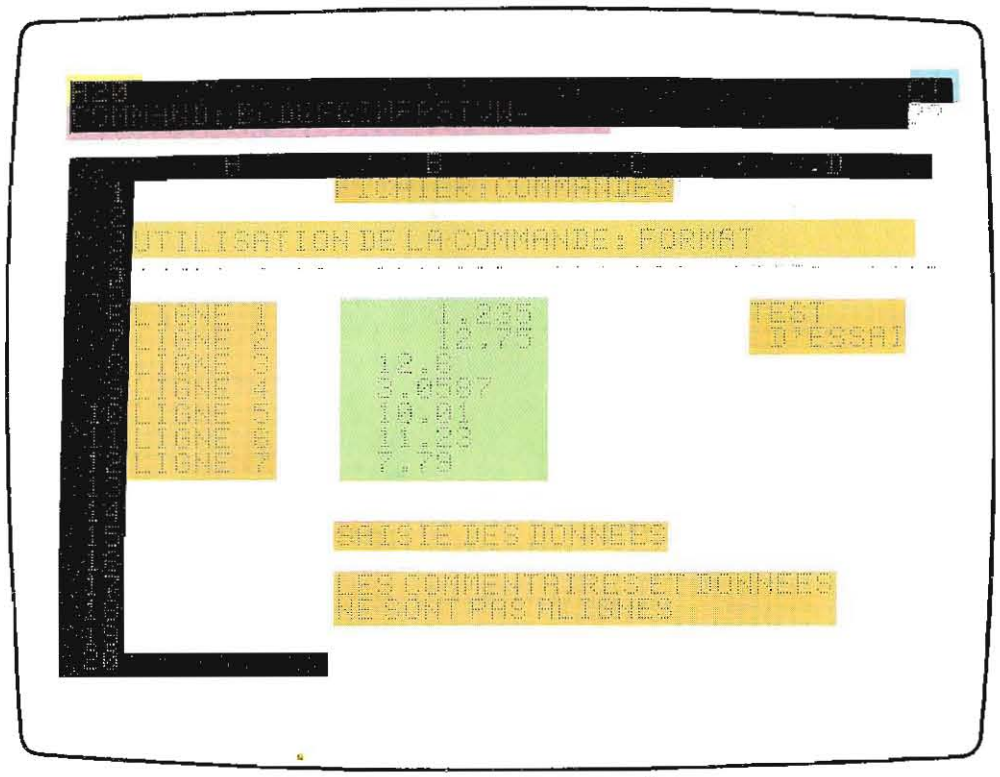
O Ordre du calcul (ligne, colonne). Avec cette option, on définit la modalité selon laquelle se déroulera le calcul.

TITLES. Permet de définir comme titres certaines lignes ou colonnes, qui resteront fixes même durant les déplacements dans la feuille. L'emploi de cette commande permet de conserver, à chaque instant, la description de chaque ligne (ou colonne), même si l'on travaille sur des zones éloignées du tableau (à cause du déplacement inévitable, l'affichage à l'écran d'une zone donnée implique la disparition d'une autre). Avec cette option, les zones définies comme tiers restent toujours visibles, mais cela implique une réduction du champ de travail. Un exemple d'utilisation de la commande TITLE est illustré de la page 802 à la page 805.

EXEMPLE D'UTILISATION DE LA COMMANDE FORMAT (1)

Cette page et les trois suivantes présentent quatre images vidéo qui illustrent l'utilisation de la commande FORMAT du tableur.
Nos exemples se réfèrent à un logiciel chargé sur un ordinateur domestique SIPREL 2030S.
Pour d'autres systèmes, les codes commandes pourront différer, tout en exécutant les mêmes fonctions.





La première ligne de l'écran contient l'indication A20, qui spécifie la position actuelle du curseur (colonne A ligne 20).

Le symbole C! indique que le calcul par colonnes (C) en mode manuel (!) est activé. De cette façon, lors de la saisie des valeurs dans les cellules voulues, les calculs ne sont pas effectués automatiquement, mais il est nécessaire d'introduire une commande explicite.
Le choix entre le mode manuel (calcul sur demande explicite uniquement) et automatique (calcul automatique après saisie), se fait en utilisant une des options prévues dans la commande GLOBAL (G).

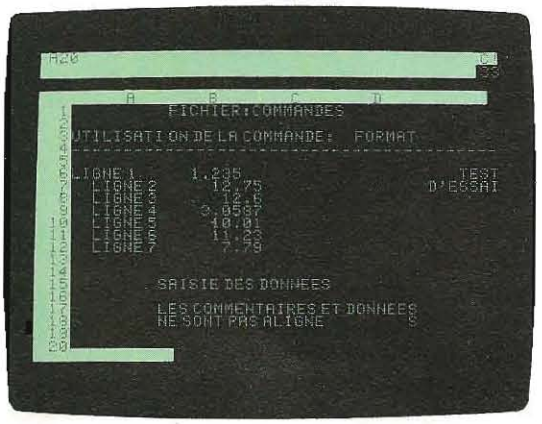
Sur la deuxième ligne, on peut voir le menu des commandes à disposition La commande FORMAT est symbolisée par la lettre F.
Cette ligne n'est affichée que sur demande de l'utilisateur. Sur la machine particulière considérée, la présentation est déclenchée par la touche ←┘
En mode saisie, la même ligne est utilisée par le programme pour afficher les données ou les expressions saisies.

Les données ont été entrées dans les différentes cellules de la feuille, sans porter d'attention particulière au positionnement correct dans les champs prévus.

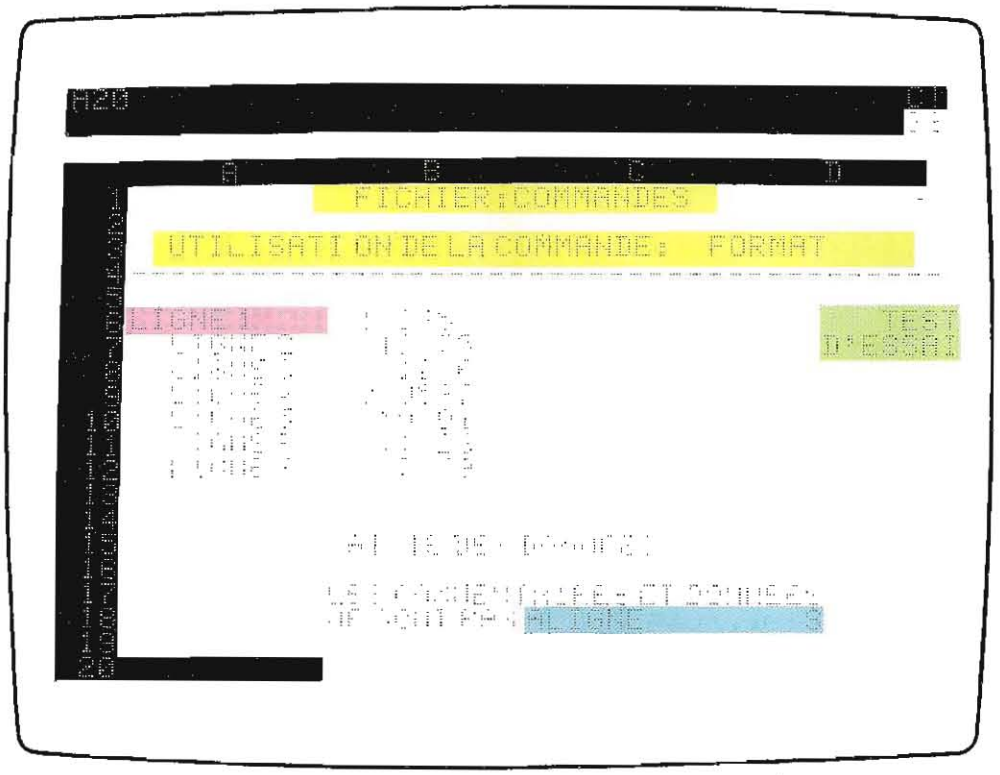
En particulier, les commentaires (titles) ont été placés n'importe où dans la feuille.

EXEMPLE D'UTILISATION DE LA COMMANDE FORMAT (2)

Sur cet écran, les données saisies à la page précédente sont affichées après justification à droite, obtenue par les commandes G et F (Format), paramètre R (Right).



Tous les contenus des cellules sont justifiés à droite, exception faite de la cellule A6, justifiée d'abord à gauche au moyen de la commande de justification d'une cellule déterminée.



En général, avec les tableurs, il existe deux modes différents de définition d'un format. Le premier, activé par la commande F, ne concerne que la cellule sur laquelle le curseur est positionné au moment de la saisie de la commande. Le second (global) concerne toutes les cellules à l'exception de celles pour lesquelles on a déjà activé la commande F. Du fait du recadrage, quelques champs sont décomposés en plusieurs parties.

Le mot ALIGNES par exemple, qui occupait deux cellules contiguës (C18, D18) est coupé en deux.

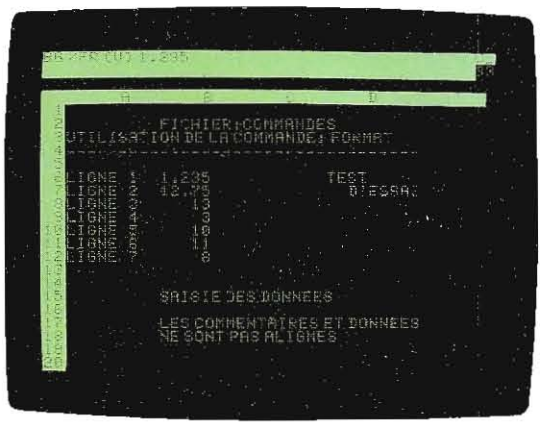
Le même problème se produit pour les champs FICHIER : COMMANDES et UTILISATION.

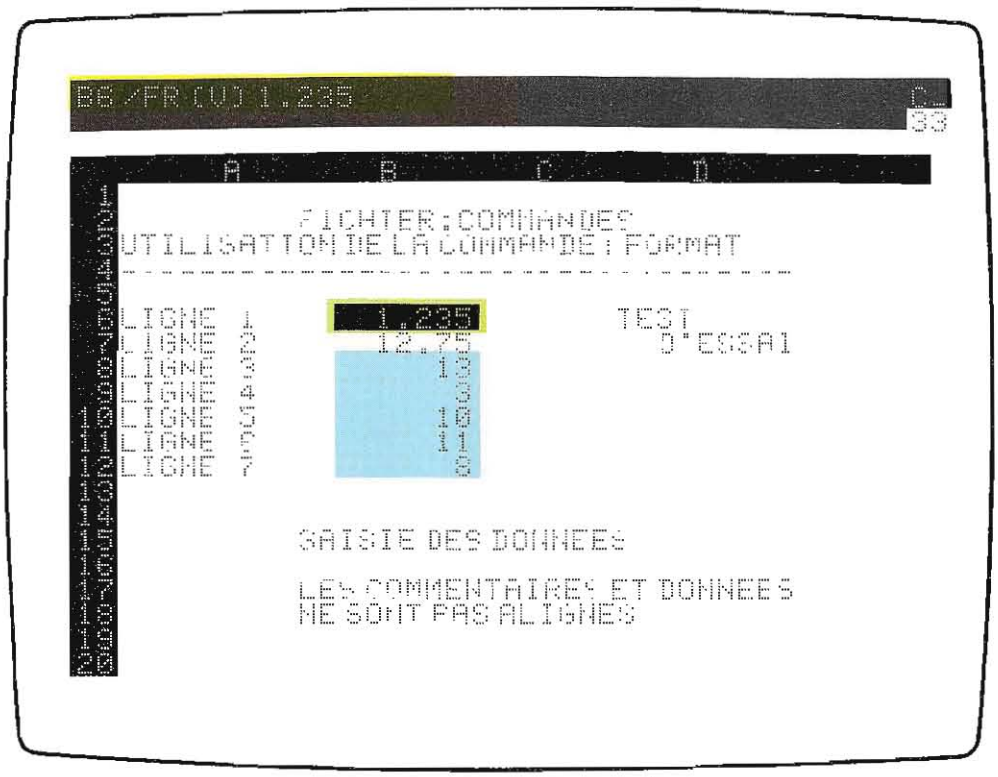
Le champ TEST d'ESSAI, au contraire, formé du contenu de deux cellules différentes mais qui appartiennent à la même colonne ne présente pas ce défaut.

En introduisant certaines écritures dans la feuille, on est porté naturellement à les poursuivre sur plusieurs cellules contiguës. On ne devra donc pas utiliser la commande GLOBAL-FORMAT. Une autre solution possible est d'augmenter la capacité des cellules jusqu'à y loger l'écriture la plus longue, mais de cette façon on réduit le nombre des cellules présentes à l'écran, puisque cette augmentation est une commande GLOBAL et concerne donc toutes les cellules.

EXEMPLE D'UTILISATION DE LA COMMANDE FORMAT (3)

Cette image d'écran illustre le fonctionnement de l'option qui permet de mettre en évidence le contenu et les spécifications de format attribuées à une cellule donnée.





Les informations relatives au contenu et aux spécifications de format assignées à une cellule donnée s'obtiennent simplement en positionnant le curseur sur la cellule à examiner.

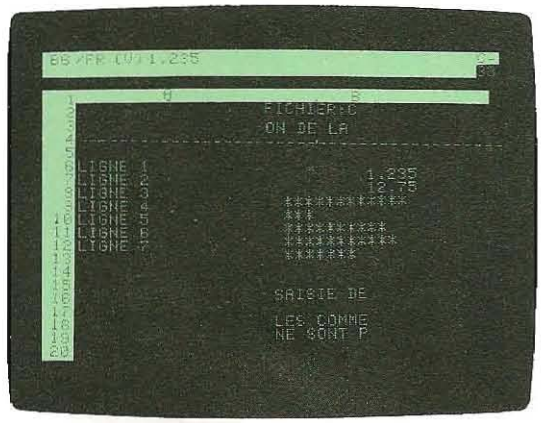
Sur la première ligne de l'écran apparaissent le numéro de la cellule sélectionnée (B6), son contenu (1.235) et les spécifications imposées [/FR et (V)]. Ici, la cellule B6 possède un format de cadrage à droite (/pour commande, F comme format, R comme right = droite) et contient une donnée [(V) = value = valeur] qui est 1.235. La même donnée aurait pu être saisie comme chaîne de caractères, dans ce cas, le symbole (V) serait absent, et la donnée 1.235 ne pourrait être utilisée pour l'exécution de calculs.

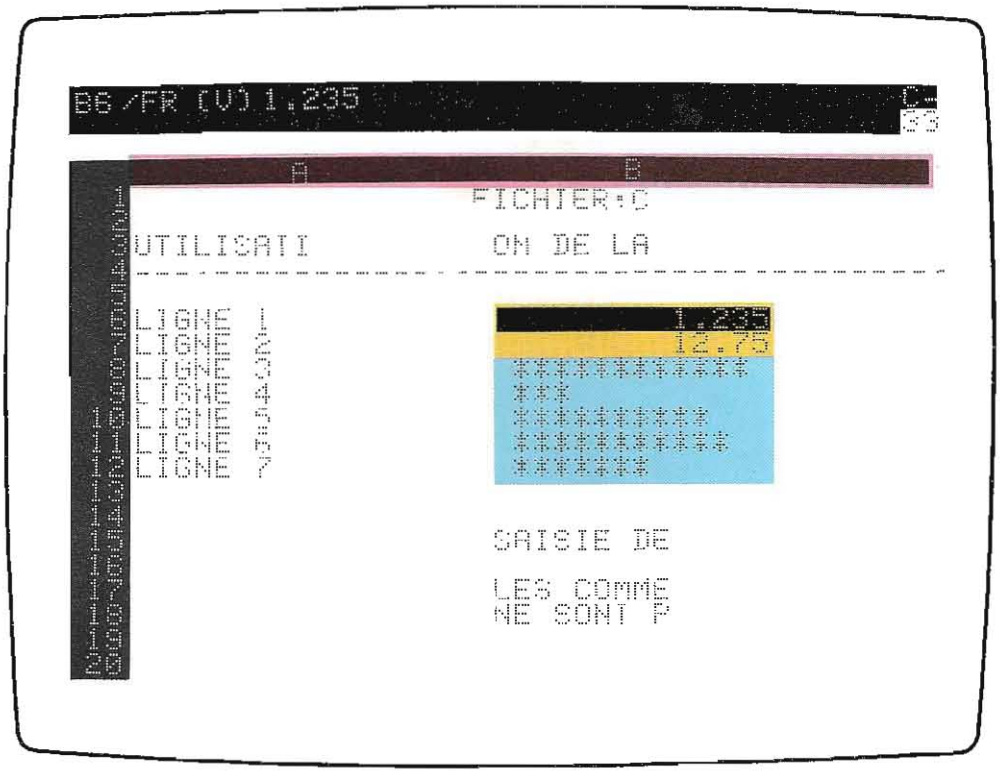
Les données sont celles de la phase de saisie (voir schéma précédent), mais après la commande G/I (GLOBAL/INTEGER), par laquelle on impose à toutes les cellules le format entier. Les valeurs ont été automatiquement arrondies à la valeur la plus proche et présentées sans décimales. Font exception les données contenues dans les cellules B6 et B7, pour lesquelles le format «local»/FR avait été précédemment défini.

Précisons que l'arrondi ne sert qu'à l'affichage: la valeur numérique assignée à chaque cellule reste telle qu'elle a été saisie.

EXEMPLE DE L'UTILISATION DE LA COMMANDE FORMAT (4)

La photo de cette page montre la feuille étudiée aux pages précédentes après agrandissement des cellules.





Le nombre de caractères par cellule a été doublé, et le nombre de colonnes simultanément présentes à l'écran automatiquement diminué de moitié.

La saisie de l'option G, * active une forme rudimentaire de représentation graphique des données saisies.

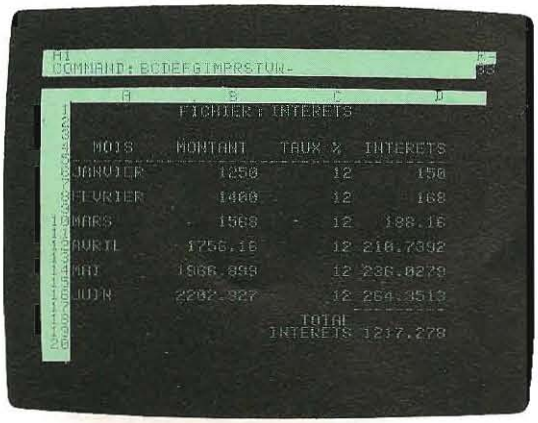
Dans chaque cellule apparaît une suite d'astérisques, proportionnelle à la valeur numérique qui s'y trouve (arrondie à la valeur inférieure).

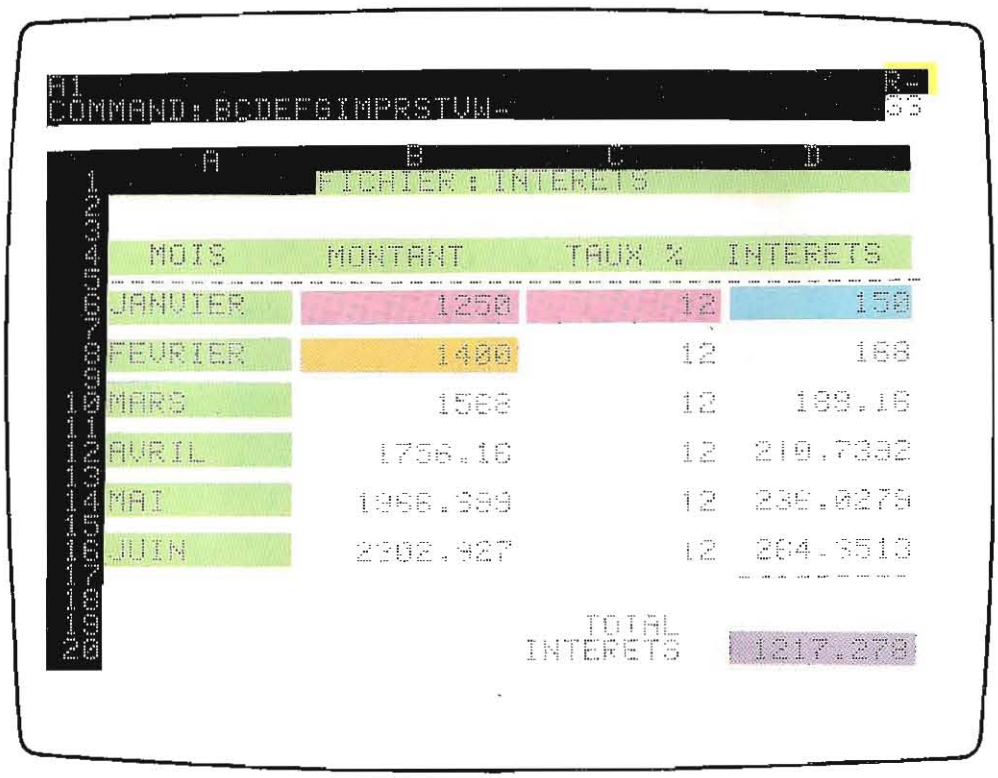
Les cellules B6 et B7 sont exclues du fait des spécifications précédentes de format « local »

Ce mode d'affichage peut s'avérer utile pour un examen global de l'évolution des valeurs numériques saisies. Certains logiciels plus sophistiqués proposent des options permettant de générer de véritables graphiques. Toutes les écritures accessoires disparaissent alors ; sur l'écran ne restent donc visibles que les graphiques demandés.

CALCUL D'INTERETS COMPOSES

La photo montre une feuille permettant le calcul des remboursements d'intérêts composés en fonction d'un capital initial
Pour chaque mois, le capital à prendre en compte est celui du mois précédent, augmenté des intérêts.





La première ligne montre qu'on a choisi le mode d'exécution des calculs par lignes.

Les cellules des lignes 1 et 4 et de la colonne A contiennent les descriptions.

En B6, on a saisi le capital initial, alors que la cellule C6 contient le taux d'intérêt (12 %).

Dans la cellule D6, on a entré la formule de calcul de l'intérêt, qui utilise les données de B6 et C6 : + B6 * C6/100. A partir du moment où le calcul par lignes a été établi, la valeur 150 (D6) apparaît automatiquement dès que les données 1250(B6) et 12(C6) ont été saisies.

Pour le calcul de l'intérêt relatif au deuxième mois, on doit se référer au capital du premier mois, augmenté des intérêts. Dans la cellule B8, on a donc entré l'expression + B6 + D6, qui évalue automatiquement la nouvelle valeur du capital.
Les contenus des cellules des colonnes B et D sont tous calculés pratiquement au même instant, suite à la saisie du capital initial en B6.

Le total des intérêts se calcule en D20, par la fonction SUM(D6...D16).

La bureautique (1)

L'automatisation du travail de bureau et sa rationalisation sont deux façons de désigner une réalité unique qui fait son apparition dans les sociétés industrielles du monde entier.

Cette perspective nouvelle peut faire faire un bond sans précédent à la productivité économique, mais aussi, si elle est mal employée, créer des situations de gaspillage technologique et aggraver les tensions inhérentes au monde du travail.

Les composantes sociales, méthodologiques et organisationnelles du phénomène jouent un rôle d'une telle importance qu'il serait illusoire d'envisager de l'affronter à l'aide de moyens et de compétences exclusivement technologiques : la réalité même du bureau est tellement complexe et riche d'interconnexions qu'on ne peut pas se dispenser d'une approche multidisciplinaire.

*En d'autres termes, on admet généralement que tout procédé d'**automatisation** ne constitue jamais la dernière étape, mais plutôt la première, d'une intense activité de révision et de reconception de l'organisation, dont les cadres de direction ont pour mission de définir l'orientation et d'assurer la bonne marche.*

Quand le processus concerne le travail de bureau, cet axiome est encore plus vrai, à cause du nombre de personnes impliquées et de l'impact profond sur leur mode de travail et sur la structure de leurs fonctions techniques d'exécution et d'encadrement.

L'automatisation du travail devient ainsi un paramètre stratégique de la gestion de l'entreprise, au même titre que les ressources humaines et financières auquelles elle est étroitement liée. La manière dont on affronte sa planification et sa réalisation influence la survie de l'entreprise par la compétitivité.

On peut dire qu'en Europe le retard dans ce domaine est considérable.

*Parlant d'informatique liée au travail, nous utiliserons, nous aussi, le terme de **bureautique** tout en soulignant deux aspects déterminants, à notre avis, pour une compréhension exacte de ce sujet.*

Remarquons tout d'abord que l'histoire de l'informatique appliquée au travail de bureau (ou, plus exactement, au travail des employés de bureau) a vingt ans d'existence, et qu'il s'agit d'étendre technologiquement ses zones d'applications. En second lieu, précisons que ce phénomène touche (ou devrait toucher) toute entreprise, petite ou grande.

*Par souci de clarté, nous continuerons à employer ce terme désormais reconnu, pour désigner l'état actuel de développement du processus d'**informatisation du travail de bureau**.*

Toutes les motivations qui soutiennent la croissance de la demande d'automatisation du traitement de l'information peuvent être ramenées à l'importance que revêt la bonne gestion de l'information.

En fait, dans les économies occidentales (ainsi qu'au Japon), on assiste depuis longtemps à un déplacement progressif de la force de travail, depuis les activités de production de biens (le travail ouvrier) vers des activités de traitement de l'information (le travail des employés de bureau).

Le schéma de la page 809 représente ce phénomène, ainsi qu'il apparaît dans la vie économique des pays membres de l'organisation pour la Coopération et le Développement Economique (OCDE) et montre comment l'activité consacrée au traitement de l'information y dépasse déjà le taux de 50 %.

Mais, fait moins connu et plus intéressant, cette reconversion s'effectue à une vitesse sans cesse croissante depuis environ 20 ans, particulièrement en Suède.

Même si on ne dispose pas toujours de statistiques exploitables, il ressort clairement que tous les pays se trouvent face à des tendances comparables : l'explosion du secteur tertiaire et l'attention croissante portée au mouvement des cadres sont des éléments qui en témoignent.

*A en croire certains économistes, on prévoit d'ailleurs la naissance d'un nouveau secteur, le « **tertiaire avancé** », où le bien d'échange sera l'information, la compétence, le savoir.*

Si telle est la perspective qui s'offre aux nations industrialisées, il faut se demander ce que sont les composantes du coût des activités liées au traitement de l'information, leur répercussion relative sur le coût global et leur évolution relative dans le temps.

Appliqué aux pays de l'OCDE, le tableau qui suit montre que les estimations actuelles révèlent une nette augmentation annuelle réelle (sans tenir compte de l'inflation) du coût du personnel et (rappelons que nous

raisonnons à égalité de prestations) une décroissance encore plus nette du coût du matériel nécessaire au traitement de l'information dans les domaines de l'archivage, du traitement et des communications.

Personnel	+ 8 %	par an,
Communications	− 10 %	dans les 10
Calcul	− 25 %	prochaines
Mémoire	− 35 %	années

Ce second fait ne doit pas surprendre. On doit considérer que les trois sortes de matériel se fondent sur la technologie électronique, qui se caractérise, depuis plus de dix ans, par les procédés de miniaturisation et par une baisse des coûts qui évolue de façon proportionnelle.

On peut toutefois se demander pourquoi cette diminution importante des coûts technologiques se répercute de façon aussi limitée dans le cas des coûts de communication tels que le présente notre tableau (vrai en moyenne pour les pays de l'OCDE), et même pourquoi elle se transforme parfois en augmentation, comme c'est le cas pour l'Italie.

Les raisons de cette anomalie apparente sont multiples, mais toutes se ramènent au fait que dans les pays européens, à cause du monopole des communications, les tarifs sont établis non pas uniquement en fonction du coût, mais aussi en regard de critères politiques et de nécessités sociales.

Il faut, en outre, ajouter que dans le domaine des communications, la technologie électronique n'a pas encore éliminé complètement du marché les systèmes électro-mécaniques, qui induisent des coûts indubitablement plus importants.

Ces considérations simples quant à l'évolution du coût des composants nécessaires au traitement de l'information conduisent à une première conclusion importante.

En l'absence de toute forme d'automatisation (adoption de techniques d'archivage, de calcul et de communication électroniques), dans le cours des huit années à venir, le coût du système d'information sera multiplié par un facteur de 2,14, à égalité de volume traité. En d'autres termes, on se retrouvera huit ans plus tard à accomplir exactement les mêmes tâches (en ce qui concerne le volume), mais avec des coûts plus que doublés.

Si, au contraire, on introduit un indice d'automatisation, exprimé par le rapport entre le coût de la technologie utilisée et le coût global du système d'information, on parvient à limiter la croissance des coûts jusqu'à l'annuler, dès lors que la valeur de cet indice atteint 0.5, maintenant ainsi ces coûts fixes dans le temps.

Avec des taux d'automatisation supérieurs à 50 %, on parvient même à réduire le coût du système d'information, jusqu'à un niveau minimum de 0,16 fois le coût d'origine, pour peu qu'on atteigne la limite de 100 % d'automatisation.

Un taux d'automatisation de 100 % doit évidemment être considéré comme une valeur asymptotique théorique, bien que des valeurs de 80 % à 90 % ne soient pas irréalisables dans certains secteurs industriels (entreprises japonaises en production continue), ou certains domaines administratifs (cycle intégré commandes-produits-production-fournisseurs de tant de sociétés produisant de manière ponctuelle, ou encore gestion des opérations de guichets dans un grand nombre d'organismes de crédit).

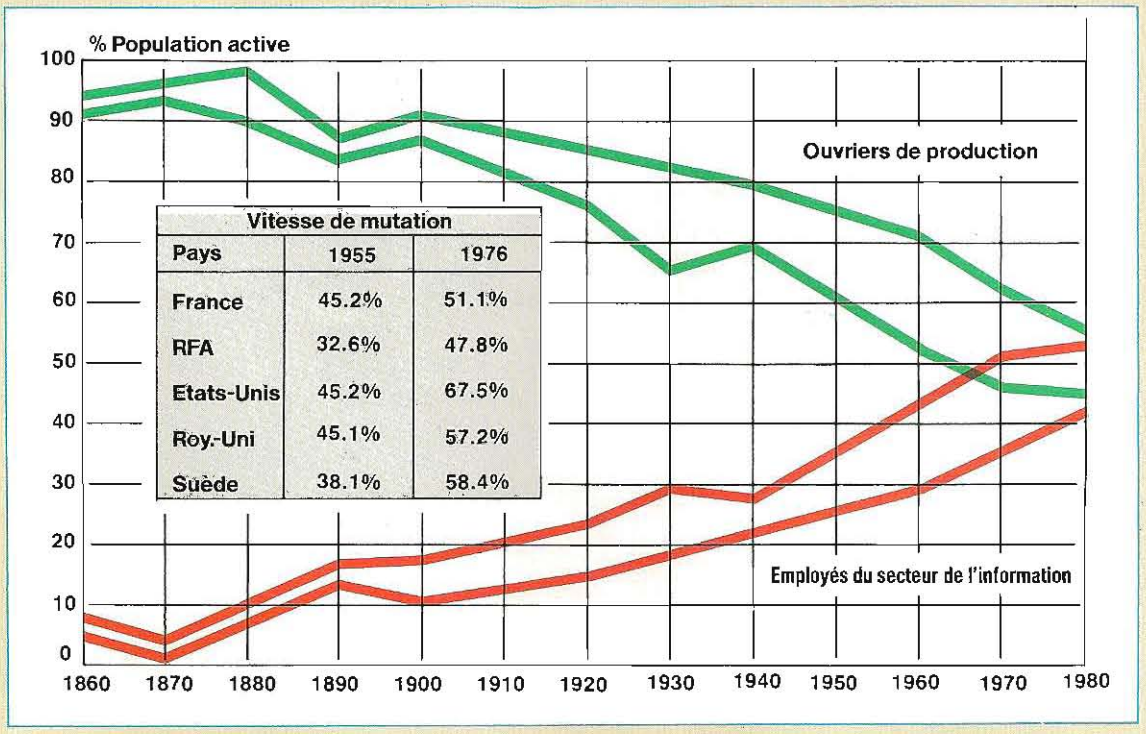
Nous avons déjà évoqué l'importance dominante que revêt l'**information** dans le monde industriel avancé.

Ce phénomène s'est particulièrement accentué dans les dix ou quinze dernières années, où l'on a vu une croissance exponentielle des exigences d'informations nécessaires à la gestion de toute entreprise, non seulement d'un point de vue quantitatif, mais aussi qualitatif.

Bien des raisons justifient cette évolution, destinée à s'accentuer au cours des prochaines années. Elles peuvent remonter aussi bien à l'internationalisation des marchés qu'à l'instabilité économique et politique diffuse, aux variations des taux de change et à la crise des sources d'énergie traditionnelles.

En fait, toutes ces causes, une fois soumises à une analyse suffisamment précise, se ramènent à une matrice conceptuelle unique, exprimable en ces termes : l'information reste la ressource la plus adéquate pour gérer, dans une situation d'incertitude, toute organisation porteuse d'une finalité d'efficacité économique et/ou sociale.

L'alternative à l'information se présente comme l'accumulation de réserves (matériel, personnel, capital) mobilisées à l'arrivée d'un événement imprévu (et imprévisible, si l'on



Vitesse de mutation

Pays	1955	1976
France	45.2%	51.1%
RFA	32.6%	47.8%
Etats-Unis	45.2%	67.5%
Roy.-Uni	45.1%	57.2%
Suède	38.1%	58.4%

Evolution de l'emploi dans les secteurs de la production et de l'information. Pour chaque catégorie, les deux courbes correspondent à des définitions plus ou moins larges de ce qu'on entend par le vocable « employés du secteur de l'information ».

considère le contexte d'incertitude générale), afin de l'affronter et le surmonter. Mais cette stratégie, vu les coûts actuels d'immobilisation, s'avère presque toujours perdante du point de vue économique, comparativement à celle qui consiste à mettre en œuvre un système d'information suffisamment ponctuel et flexible pour permettre à l'entreprise de s'adapter dynamiquement au contexte dans lequel elle se trouve, voire d'en tirer un maximum d'opportunité.

A la lumière de ce fait, les considérations précédemment évoquées prennent un nouveau relief. Il ne s'agit plus seulement de minimiser les coûts du traitement de l'information, mais plutôt de développer une capacité d'information (qui ne peut se cantonner à l'automatisation comme support essentiel) capable de maintenir ou d'accroître la compétitivité de l'entreprise en affinant son processus décisionnel.

La productivité, dans l'acception commune du terme, représente une mesure d'efficacité, définie comme la quantité de travail nécessaire pour produire un résultat déterminé, tant quantitativement que qualitativement (dans ce cas, la définition demande à être clarifiée). Si le résultat s'améliore sans qu'il soit nécessaire d'ajouter du travail, la productivité s'accroît ; de même, elle augmente si le même résultat peut s'obtenir en réduisant la quantité de travail.

Les statistiques portant sur la productivité du travail de bureau n'existent pratiquement pas, ou demeurent sujettes à controverses. Cependant, certaines études ont établi que les machines de ***traitement de texte****, entre autres, peuvent améliorer de plus de 200 % la rentabilité d'un travail de bureau qui consiste en des tâches répétitives comme les courriers, les rapports, les contrats, etc.*

Pour concevoir avec précision ce qui sera rentable pour son entreprise, un responsable devra mener son enquête selon des critères bien particuliers. En effet, la notion de productivité dans un bureau est sujette à caution. N'est pas plus productif celui qui produit dix documents alors que cinq suffiraient, et n'est certainement pas productif celui qui passe son temps en réunion, par exemple.

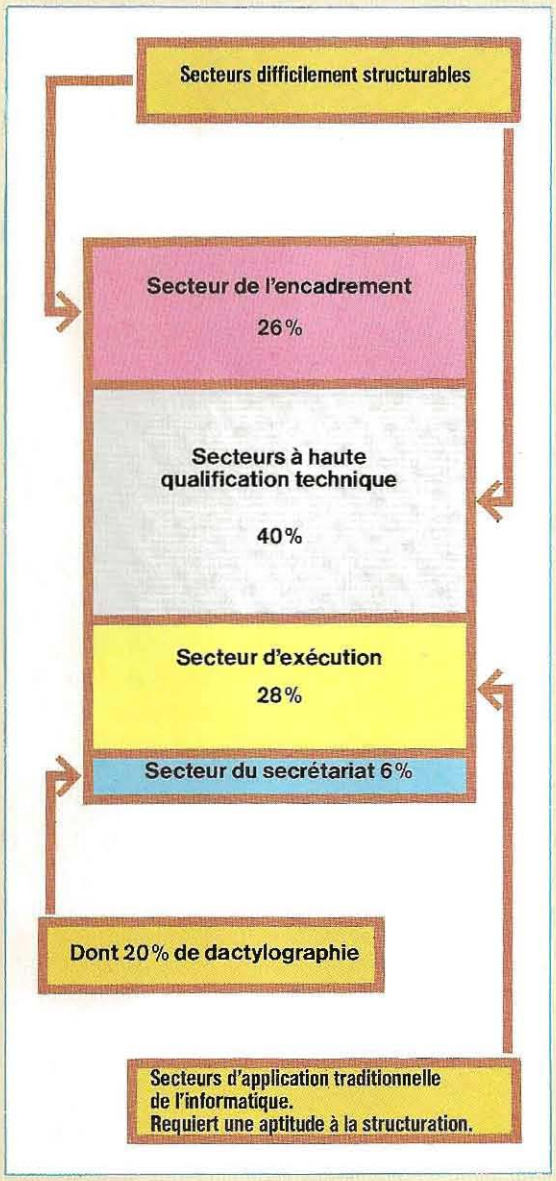
Une telle étude ne peut prendre en compte que des performances atteintes, c'est-à-dire le

degré ***de réalisation des objectifs visés***.
La planification du marketing d'une société, par exemple, peut se mesurer au coût, à la qualité, à l'opportunité de ses plans, mais aussi bien à sa capacité d'œuvrer en collaboration avec les responsables commerciaux.
De fait, même pour les tâches d'exécution, la performance est la mesure par excellence du rendement d'un individu, d'un groupe, ou d'un secteur de l'entreprise.
D'une certaine façon, la notion de performance est un concept plus vaste que celui de productivité ; améliorer une performance signifie obtenir des résultats bien plus efficaces, en regard des objectifs que s'est fixé la société, que ce qui découlerait d'un simple accroissement de la productivité.
Dans le domaine des tâches de production, on pourrait simplifier le problème en le limitant à son aspect quantitatif ; même dans ce cas, on ne doit pas oublier que ce type de productivité ne représente pas une valeur en soi, mais doit trouver sa finalité dans les objectifs de la société.
Nous ne disposons, hélas, que de données empiriques relativement limitées pour pouvoir établir la corrélation qui pourrait exister entre un matériel de bureautique et la qualité des prestations requises, et cela parce qu'il est très difficile de faire dire à quelqu'un qui se veut indispensable que la machine pourrait très bien exécuter la moitié de son travail.
En fait, ces corrélations mettent à jour une loi économique récurrente : un secteur naît afin d'accroître la productivité du secteur qui le précède, au moyen d'investissements dans l'automatisation, et, en termes de valeur ajoutée, du produit d'échange (qu'il s'agisse de l'agriculture, des ressources minières, de la manufacture ou des services) ; et donc, à travers son expansion et l'absorption de nouvelles ressources, il devient à son tour cause d'inefficacité du système.
Ainsi, la machine à vapeur fut à l'origine utilisée en agriculture (pour accroître l'efficacité du secteur primaire) ; le secteur tertiaire naquit de la nécessité de rendre plus efficace, par ses services, le secteur secondaire.
De nos jours, c'est le secteur tertiaire lui-même qui doit accroître ses performances, grâce au développement du tertiaire avancé, et dont l'informatique, par définition, fait partie.
Lorsqu'on parle d'investissements en bureautique, conformément aux réflexions précédentes au sujet de la productivité (ou des performances), il est nécessaire d'identifier, au moins en termes généraux, les fonctions auxquelles on veut en priorité s'adresser.
Le schéma de la page 811 indique la ***répartition*** *du coût du travail de bureau, dans une société du secteur secondaire (à l'exclusion, cette fois, de la composante ouvrière) ou du secteur tertiaire.*
Comme on peut le voir, un peu plus du tiers est consacré aux tâches d'exécution, tandis que presque deux tiers sont dévolus aux emplois qualifiés et d'encadrement.
Cette constatation suggère immédiatement quelques réflexions. Tout d'abord, on peut interpréter la part relativement faible du coût du travail lié aux tâches de production comme le résultat de vingt ans d'investissements en informatisation de ce secteur : en fait, on le sait, les applications informatiques usuelles ont été historiquement concentrées sur l'automatisation des tâches à caractère répétitif, et devraient en avoir réduit l'incidence sur le coût global.
Remarquons ensuite que dans le domaine des tâches d'exécution, et plus particulièrement dans celles de secrétariat, l'activité liée au traitement de texte présente une incidence sur le total de 20 % des coûts.
Il faut aussi se pencher sur les activités qualifiées et d'encadrement qui, à elles deux, constituent presque les deux tiers du coût total, et dans lesquelles jusqu'ici, sauf cas particulier, l'informatique n'a encore pénétré, en Europe du moins, que marginalement.
Le traitement de texte, nous l'avons vu, revêt une importance indiscutable en tant que moyen d'introduction de l'information au sein du système bureautique. On peut admettre que la dactylographie est au système d'automatisation ce que la saisie de données est au traitement de données traditionnel.
Mais cela ne doit pas conduire, pour les raisons citées plus haut, à confondre les investissements dans les systèmes de bureautique, et les investissements dans les systèmes strictement liés à la dactylographie ; ces derniers dépendent du contexte et peuvent parfois donner lieu à des incompatibilités insurmontables et à une parcellisation du travail, définitivement négative en terme de productivité et d'efficacité.

Une fois précisées les fonctions auxquelles s'adresse en priorité l'automatisation du travail, la question se pose de savoir comment leurs prestations peuvent être améliorées selon la définition établie plus haut.
Pour cela, il faut préciser les caractéristiques des tâches de traitement de l'information et leurs modalités d'automatisation.
Le travail de bureau se décompose en :

- *fonctions (comptabilité des clients, gestion des commandes, contrôle de la production, etc.) ;*
- *activités (traitement, échange des informations).*

Les fonctions dépendent de la structure de l'entreprise. Les activités sont à la base de nombreuses fonctions et concernent l'activité individuelle. L'automatisation du travail (Office Automation) est orientée vers l'automatisation des activités. Le traitement des données (Data Processing) vers celle des fonctions.
Le développement des applications traditionnelles de l'informatique a œuvré dans le sens d'une « procédurisation » des diverses fonctions de l'entreprise qui s'y prêtaient (c'est-à-dire les fonctions que l'on peut structurer) et où, de plus, les tâches répétitives jouaient un rôle notable. A titre d'exemple, mentionnons, en ce qui concerne le monde industriel, les différentes comptabilités, la gestion des commandes, celle de la production et des achats, etc., et dans le secteur bancaire, les différents services, des comptes courants aux livrets d'épargne, aux portefeuilles de titres, etc.
Dans cette optique, l'informatique intègre implicitement à ces fonctions les activités élémentaires d'archivage, de recherche et d'échange de l'information, en les introduisant dans les procédures et dans les programmes correspondants.
En fait, ces activités restent en grande partie indépendantes de l'application, puisque par définition elles représentent le support de toute fonction d'information qui a cours dans un bureau. On pourrait donc imaginer de les extraire des procédures et de les standardiser, afin de les rendre accessibles chaque fois qu'elles sont nécessaires, non seulement aux applications d'exécution, mais aussi aux



Répartition des coûts du travail de bureau.

tâches non structurables, et donc non abordables par l'informatique.
L'automatisation de telles activités constitue justement un des objectifs prioritaires de la bureautique, et un des éléments de différenciation d'avec l'informatique classique.
Une autre considération concerne la destination du service automatisé.
Dans le cas des fonctions structurées de l'entreprise, qui représentent précisément le champ d'action de l'informatique, les destinataires font partie, à l'intérieur de la structure organisationnelle de l'entreprise, de groupes

(et/ou de services) homogènes quant aux objectifs et aux méthodes de travail (ce qui constitue précisément la procédure d'exécution d'une fonction d'entreprise spécifique). Dans le cas d'activités élémentaires de traitement de l'information, au contraire – et justement par leur caractéristique de support – l'utilisateur ne peut être considéré qu'à titre individuel et non comme appartenant à un secteur opérationnel défini. L'emploi (ou non) de l'automatisation devient alors pour lui un choix personnel et professionnel.

En d'autres termes, les applications informatiques traditionnelles contiennent implicitement une contrainte d'usage, sous peine d'insuccès de l'application elle-même : il n'est pas cohérent, d'un point de vue organisationnel, que les mêmes procédures soient exécutées par certains manuellement et, par d'autres, de manière automatisée. En bureautique, au contraire, c'est à l'utilisateur de décider de l'opportunité de leur usage, en fonction de la tâche à accomplir et de sa qualification. Ceci signifie que la bureautique doit être considérée, dans une large mesure, comme une aide au travail, aujourd'hui non structurable, du spécialiste ou du responsable. C'est seulement dans ce sens qu'elle peut contribuer de manière significative à la qualité de leurs performances.

Si l'on devait schématiser, pour clarifier l'exposé, on pourrait affirmer que l'approche traditionnelle de l'informatisation du travail peut se répartir en deux catégories :

a) l'approche consistant à adopter des systèmes existants, de type multi-fonctions, aux applications de bureau, tout en laissant intacte l'interface informatique ;

b) l'approche consistant à introduire des matériels spécifiques pour des activités spécifiques.

Les solutions apportées par le développement d'un logiciel polyvalent approprié, implanté sur un ordinateur de gestion de petite, moyenne ou grande capacité, appartiennent à la première catégorie.

Un tel logiciel, utilisé depuis un terminal, possède généralement les fonctions classiques de traitement automatique des informations (de type texte), permet leur archivage sur mémoire de masse et leur mise à disposition, au moyen d'un dispositif appelé ***courrier électronique*** *(électronic mail), à tous les utilisateurs qui, toujours au moyen d'un terminal, y accèdent pour consultation et éventuellement modification.*

Puisque c'est sur ce même ordinateur que sont implantées les procédures informatiques de l'entreprise, cette solution possède le grand mérite d'intégrer, au moins physiquement sinon logiquement, à l'univers traditionnel du traitement des données, celui des informations du bureau. Ces systèmes traduisent dans les faits cette complémentarité des deux domaines qui reste essentielle à toute stratégie de développement de l'informatisation de l'entreprise.

Les limites à sa mise en œuvre se situent dans sa faible possibilité de diffusion auprès des utilisateurs qui acceptent mal de se plier à l'apprentissage des commandes caractéristique des systèmes informatiques. Par ailleurs, modifier cette interface afin de la rendre plus simple à utiliser impliquerait un alourdissement de la charge additionnelle du système, avec pour conséquence une réduction radicale de ses prestations générales.

Si l'on se réfère au fait que l'un des objectifs essentiels de la bureautique est précisément de pouvoir s'adresser à un personnel dépourvu de compétences informatiques, on comprend aisément que cette solution peu ergonomique n'ait suscité, jusqu'ici, qu'un enthousiasme mesuré.

L'ensemble des petits matériels de bureau, qui ont pour but de rendre plus efficaces certaines activités spécifiques liées au traitement de l'information, fait partie de la seconde catégorie. Ainsi en va-t-il de la machine à écrire électronique qui réduit les coûts de secrétariat, de la photocopieuse programmable qui rationalise le processus de reproduction, du téléphone à touches et à mémoire qui accélère la composition du numéro et l'appel des numéros les plus fréquemment utilisés.

Le vrai problème est, qu'en Europe, trop peu de dirigeants de petites et moyennes entreprises sont eux-mêmes convaincus de la nécessité absolue qu'il y a de mettre une structure de bureautique au service de leur personnel.

(Extrait de « L'informatique dans le travail de bureau » CAHIERS D'INFORMATIQUE dixième année, n° 1, 1983 Honeywell Information Systems).

Les fonctions du tableur

Les logiciels gérant un tableur possèdent de nombreuses fonctions prédéfinies de déroulement de calculs ou d'opérations logiques. Les fonctions numériques suffisent à la préparation de feuilles de calcul ne présentant pas de difficultés particulières ; les fonctions logiques permettent de développer des applications plus complexes. Certains logiciels de tableurs (Lotus par exemple), prévoient en outre l'emploi des **macro-instructions** (ou « macro »), fonctions définies et écrites par l'utilisateur, qui peuvent être appelées directement par la frappe d'une touche. Grâce aux macros, le système devient totalement programmable, mais requiert de nombreuses connaissances de base.
Quelle que soit la forme de la fonction à utiliser, on peut la loger à l'intérieur d'une cellule. Pendant la phase de calcul, la valeur numérique résultant de l'exécution de la fonction sera présentée dans cette cellule.
En général, les arguments d'une fonction sont des valeurs numériques, ou bien des adresses. Dans le premier cas, le calcul de la fonction s'exécute de façon immédiate sur les valeurs des arguments, dans l'autre cas la valeur numérique est, au préalable, extraite de l'adresse (ligne, colonne) spécifiée. La syntaxe générale d'une fonction est :

@ xyz(n)

où

@ est le symbole de reconnaissance d'une fonction (il dépend du type de logiciel) ;
xyz est le nom de la fonction ;
n représente l'argument.

Par exemple, le calcul de la valeur absolue d'un nombre s'effectue, dans presque toutes les versions du tableau électronique, grâce à la fonction @ABS. La syntaxe devient :

@ ABS(−7+10)	de façon immédiate, sur des valeurs numériques ;
@ ABS(+D3)	indirectement, sur la valeur contenue dans la cellule D3.

La variété des fonctions à disposition dépend du type de tableur, mais il existe un certain nombre de fonctions fondamentales communes à tous les logiciels.

Fonctions mathématiques

Les fonctions mathématiques employées se rapprochent de celles utilisées en Basic ; l'unique différence réside dans le symbole de reconnaissance qui doit obligatoirement précéder le nom de la fonction. En convenant de désigner par n une valeur ou une adresse, les principales fonctions mathématiques sont :

ABS(n)	valeur absolue ;
EXP(n)	exponentielle ;
INT(n)	partie entière ;
LN(n)	logarithme népérien (base e) ;
LOG(n)	logarithme décimal ;
SQRT(n)	racine carrée ;
COS(n)	cosinus ;
ACOS(n)	arc cosinus (fonction inverse du cosinus) ;
SIN(n)	sinus ;
ASIN(n)	arc sinus (fonction inverse du sinus) ;
TAN(n)	tangente ;
ATAN(n)	cotangente (fonction inverse de la tangente).

Sur cette liste, on a omis le symbole de reconnaissance de fonction, strictement dépendant du type de logiciel.
L'emploi de ces fonctions (que l'on peut appeler de la même façon qu'en Basic) ne requiert pas de connaissances spéciales du tableau électronique. Essayons, par exemple, d'écrire un programme qui, connaissant les deux côtés d'un triangle rectangle, calcule l'hypoténuse par le théorème de Pythagore. Les fonctions à exécuter sont :

— saisie des deux côtés ;
— calcul de l'hypoténuse ;
— affichage de la valeur de l'hypoténuse.

En Basic, le programme se présentera ainsi :

```
10 INPUT "Côté A";A
20 INPUT "Côté B";B
30 V=SQR(A^2+B^2)
40 PRINT "Hypoténuse = ";V
50 END
```

A l'aide du tableur, on peut écrire le même programme sans connaître le Basic (voir ci-dessous).
Les cellules B1 et B2 contiendront les données (correspondant aux variables A et B) ; la cellule B3 recevra la formule de calcul, et contiendra, lors de l'affichage, le résultat du calcul (elle remplace donc les deux lignes 30 et 40).

Fonctions numériques spéciales

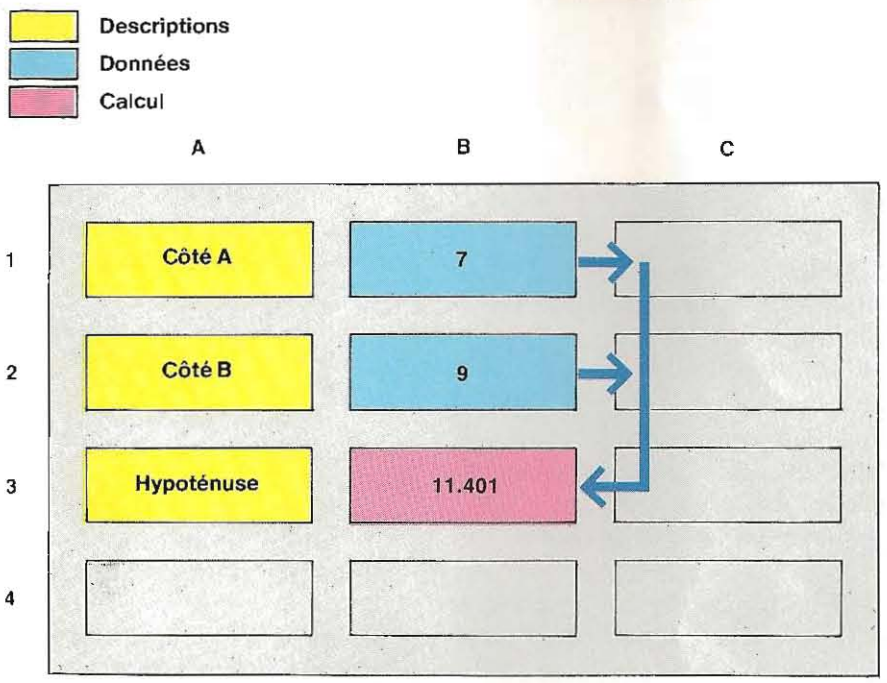
L'application majeure du tableur concerne les calculs économiques ou financiers (prévisions budgétaires, calculs d'intérêts, plans d'investissement, modélisation).
Pour répondre à ces problèmes particuliers, on a introduit une série de fonctions qui n'existent pas en Basic (à moins de les définir comme fonctions-utilisateur).
Les principales sont répertoriées ci-dessous :

AVERAGE (n1, n2,...). Calcule la moyenne arithmétique des contenus des cellules spécifiées comme paramètres. Par exemple :

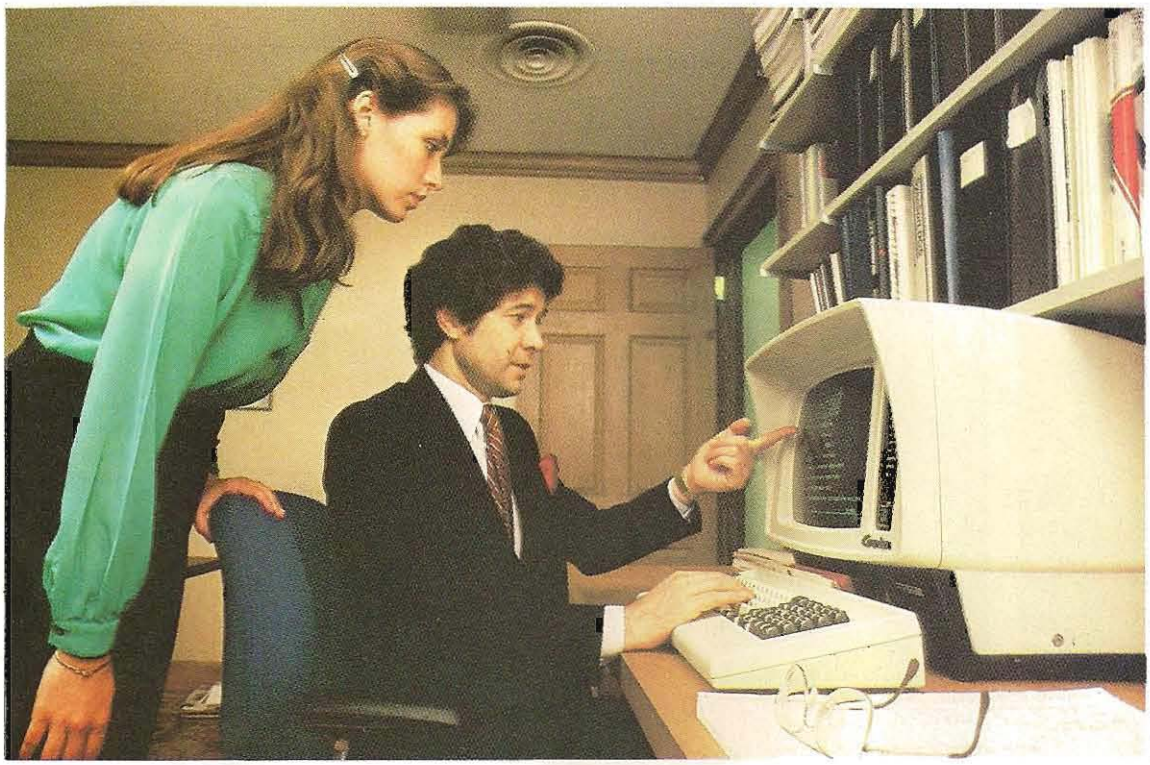
AVERAGE (+B1, +B3, +C6)

fournit la moyenne des contenus de B1, B3 et C6 (somme divisée par 3).
Pour le cas où l'on devrait calculer la moyenne de nombreuses cellules, il serait malaisé de les indiquer toutes.
Aussi, existe-t-il une notation permettant de définir un domaine de valeurs (**range**). Cette possibilité d'utilisation dépend d'un positionnement adéquat des données.
Dans l'exemple précédent (B1, B3, C6), la dispersion des emplacements (on saute aussi bien de colonne que de ligne) ne se prête pas à la définition d'un tel intervalle.
Il faut, en effet, que toutes les données soient placées sur la même ligne ou dans la même colonne, bien que dans certains cas, elles puissent figurer sur plusieurs lignes ou colonnes adjacentes.
En revenant à notre exemple, si l'on déplace le contenu de C6 en B2, les données se trouvent alors en positions B1, B2 et B3, c'est-à-dire alignées sur une même colonne (B). On peut alors définir un intervalle qui englobe les positions de B1 à B3. La syntaxe la plus utilisée pour définir un domaine est :

EXEMPLE D'APPLICATION DU TABLEUR



Les cellules B1 et B2 sont destinées à recevoir les données (valeur des côtés).
Dans la cellule B3 est entrée la formule de calcul de l'hypoténuse :
@SQRT (+B1^2+B2^2)



Int'l Stock Photo/Marka

L'emploi de systèmes informatisés apporte une contribution déterminante à la rationalisation du travail de bureau.

B1...B3

où la première valeur désigne la limite inférieure de l'intervalle, les pointillés « jusqu'à », et la dernière valeur, la limite supérieure.
Avec cette notation, la fonction de calcul de la valeur moyenne s'écrit

AVERAGE (B1...B3)

L'emploi de cette possibilité offre un autre avantage : il n'est plus nécessaire désormais de spécifier directement les numéros de cellules (de départ et d'arrivée) : il suffit de positionner le curseur sur la cellule concernée et le système pourvoit à l'écriture du numéro correspondant.
Cette désignation par le curseur demande toutefois une précaution : celle de savoir si le système travaille en mode absolu ou relatif.
En pointant une cellule quelconque, par exemple B8, on entend faire référence au contenu situé, dans la feuille à l'intersection de la colonne B et de la ligne 8, et ceci constitue la méthode directe (ou absolue).

Il existe par ailleurs, une seconde méthode d'adressage : le mode relatif. Supposons que le curseur occupe, sur la feuille la position B9. Désigner la position B8, par adresse relative, signifie qu'on se réfère à la cellule située sur la même colonne B, une ligne plus haut.
De façon analogue, le curseur étant toujours en B9, désigner C10 signifie pointer la cellule qui se trouve une colonne plus à droite (de B à C) et une ligne plus bas (de 9 à 10).
Supposons que la cellule B9 contienne la formule de calcul de la somme de B8 et C8 (+ B8 + C8). En mode d'adressage absolu, si l'on déplace le contenu de la cellule B9 vers une autre position (par exemple E3), le calcul reste inchangé car le système prend en compte le contenu des positions directement spécifiées. A l'opposé, en mode d'adressage relatif, on obtient des résultats faux, car le système modifie l'expression de façon à référencer non plus les positions précédentes, mais de nouvelles cellules, occupant, par rapport à E3, une position relative identique à celles qu'occupaient B8 et C8 par rapport à B9 (voir le graphique de la page 816).

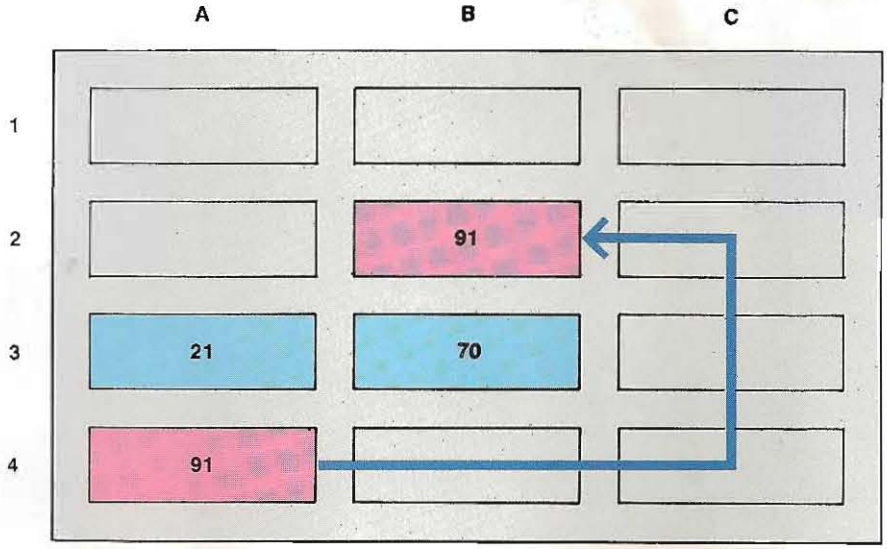
ADRESSAGE ABSOLU ET RELATIF

Cellule contenant l'expression

Cellule contenant des données non valides

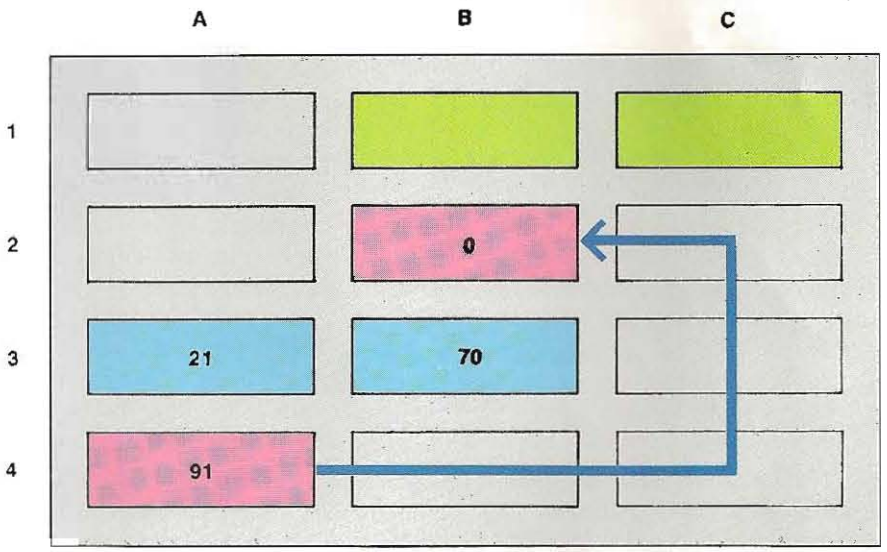
Cellule contenant des données valides

Adressage absolu



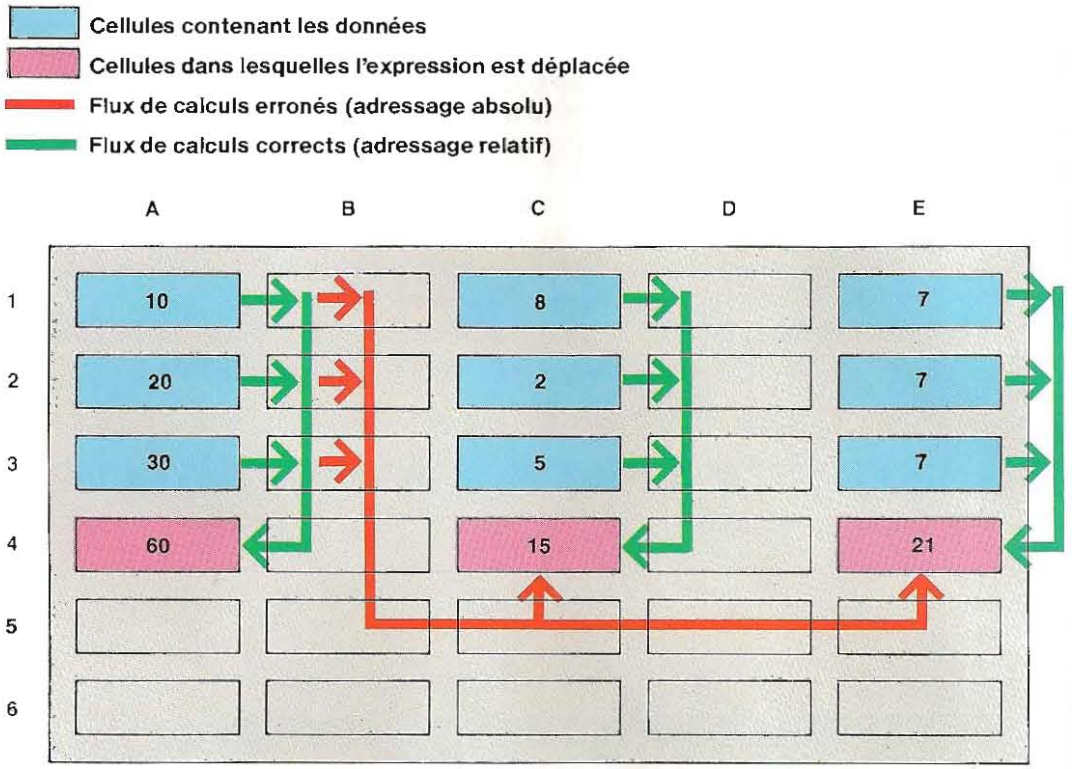
La cellule A4 contient l'expression + A3 + B3 et donne comme résultat la somme des contenus des cellules A3 et B3.
Si l'on déplace l'expression dans toute autre cellule, le résultat reste inchangé, étant donné que les coordonnées des cellules auxquelles le calcul se réfère restent les mêmes.

Adressage relatif



Avec le mode d'adressage relatif, le déplacement du calcul dans la cellule B2 provoque une erreur.
A la suite du déplacement, le système a modifié l'expression, qui devient + B1 + C1, puisque les positions relatives occupées par B1 et C1 par rapport à B2 sont les mêmes que celles qu'occupaient A3 et B3 par rapport à A4.

APPLICATION DE L'ADRESSAGE RELATIF A UNE SOMME PAR COLONNE



La cellule A4 renferme l'expression A1 + A2 + A3, qui affiche la somme des données de la colonne A. L'adressage relatif, introduit en déplaçant le contenu de A4 en C4, puis en E4, fournit le même résultat soit la somme des données mises en colonne en C et en E. En revanche, si la feuille de calcul (tableur) est régie par le système de l'adressage absolu dans les cellules C4 et E4, c'est encore la valeur 60 qui sera affichée, les adresses se référant toujours aux données de la colonne A.

L'adressage relatif présente un intérêt réel. Dans le tableau ci-dessus, les totaux sont appelés par colonnes. Si, avec l'adressage relatif on déplace la cellule A4 en C4 et en E4, l'expression est automatiquement calculée, selon la même méthode. En applicant l'adressage absolu aux cellules C4 et E4, on obtiendrait une expression similaire à A4 mais avec des résultats erronés.

COUNT (NS..NE). Cette fonction restitue la quantité de cellules « pleines » comprises dans le champ défini par les adresses NS et NE. Page 818, on trouvera une application de la fonction COUNT pour déterminer le nombre d'articles en stock. La ligne 4 (entrées des nouveaux articles) n'est pas intégrée au calcul tant que la description de l'article n'est pas réalisée.

SUM (NS..NE). Dans la page suivante, on a également utilisé la fonction SUM pour obtenir la somme des valeurs des différentes lignes. L'utilisation de cette dernière fonction est immédiate : SUM (C2..C5) équivaut à + C2 + C3 + C4 + C5. On remarquera que l'emploi conjoint des fonctions COUNT(...) et SUM(...) donne la fonction « Moyenne » AVERAGE(...) :

$$\text{AVERAGE (A1...A9)} = \frac{\text{SUM (A1...A9)}}{\text{COUNT (A1...A9)}}$$

MAX (NS..NE), MIN (NS..NE). Elles extraient respectivement les valeurs maximales et minimales contenues dans le champ indiqué.

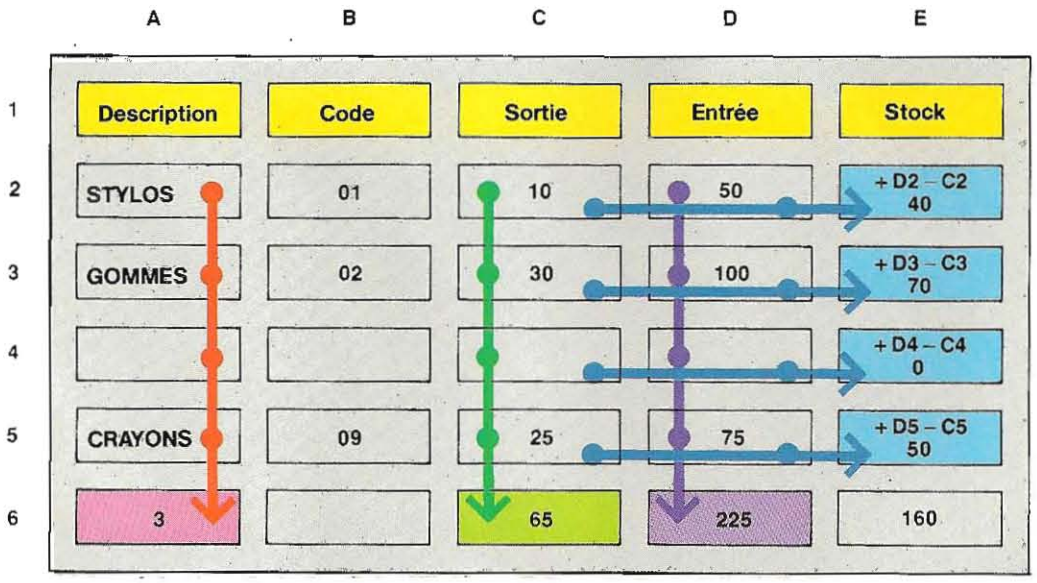
NPV (S, NS..NE). NPV signifie Net Present Value. On l'utilise pour calculer (à partir d'un résultat final et d'un taux d'intérêt fixé à l'avance) le montant du capital initial exigé par la transaction. Le paramètre S est le taux d'intérêt, les valeurs NS..NE sont les chiffres que l'on veut obtenir à l'issue de chaque période. Ainsi, pour aboutir à un montant (capital plus intérêts) égal à 1500, avec un taux de 20 % (S = 0,2) portant sur une seule période, il faudrait investir la somme de 1250 F. En fait, la rente de ce chiffre à 20 % est de 250 ; additionnée au capital initial, cette rente donne 1 500.

Fonctions de recherche

Toutes les fonctions permettant une sélection des données appartiennent à cette catégorie. La variété des fonctions disponibles dépend du type de tableur utilisé. Dans ce chapitre, nous allons illustrer uniquement la fonction principale LOOKUP, commune à tous les programmes.

LOOKUP (M,NS..NE). Cette fonction recherche la valeur M dans le champ NS..NE. La variable restituée est liée à la nature du tableur. Par exemple, dans le cas du programme Multiplan, c'est la valeur de la dernière cellule ou celle de la ligne où M a été trouvée.

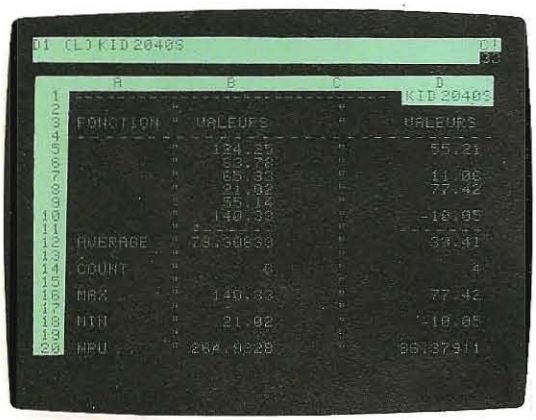
LES FONCTIONS COUNT ET SUM DANS LA GESTION D'UN MAGASIN

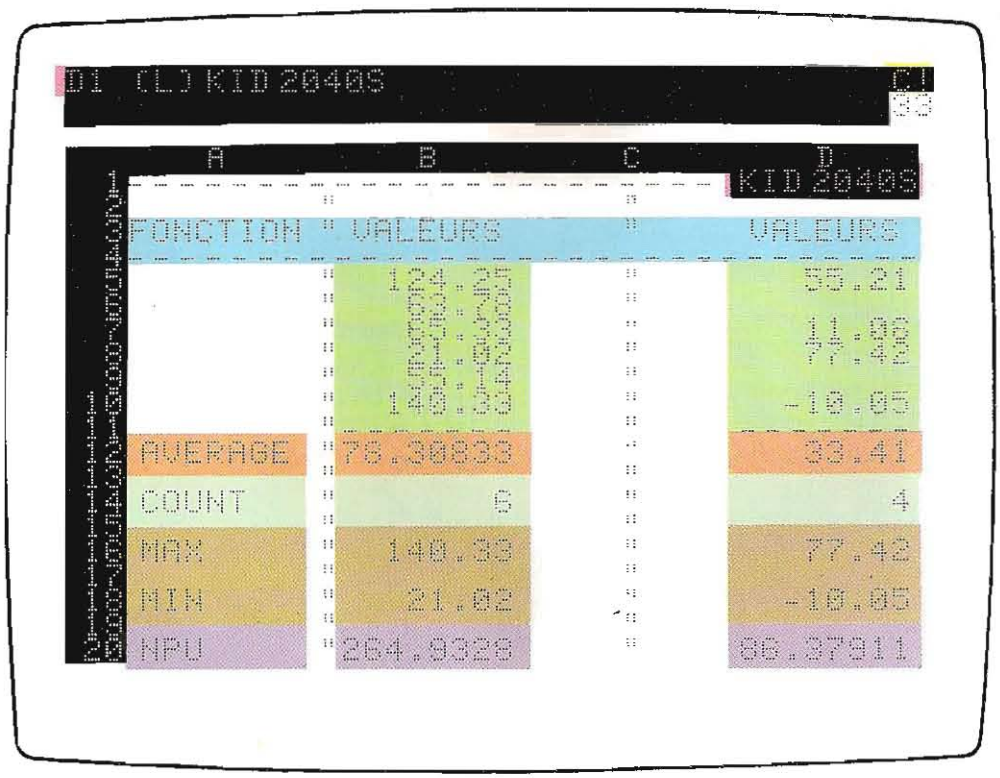


En mettant dans la cellule...	la fonction...	il apparaît...
A6	@ COUNT (A2..A5)	le numéro des différents articles présents dans le magasin
C6	@ SUM (C2. C5)	le total sorti
D6	@ SUM (D2..D5)	le total entré
E6	@ SUM (E2..E5)	le total en stock

FONCTIONS DU TABLEUR

L'écran montre le calcul de quelques fonctions communes à tous les tableurs.





Le calcul par colonnes est demandé.

La cellule D1 contient le nom de la machine utilisée (SIPREL KID 2040S). Dans d'autres systèmes, les fonctions affichées ont d'autres noms mais elles ont la même signification.

La ligne 3 contient les descriptions des colonnes.

Les cellules des colonnes B et D renferment les données utilisées pour le calcul des fonctions. Les valeurs introduites n'ont pas un sens particulier.

La fonction AVERAGE fournit la moyenne des valeurs. Ainsi en B14 apparaît la moyenne des valeurs de la colonne B et en D14 celles des valeurs de la colonne D.

La fonction COUNT donne le numéro des cellules occupées par les valeurs numériques pour chacune des deux colonnes. Pour la colonne D, la valeur est 4 : les cellules vides ne sont pas prises en considération.

Les fonctions MAX et MIN calculent les valeurs maximales et minimales dans chacune des 2 colonnes.

La fonction NPV calcule l'apport nécessaire pour produire un capital final placé à un certain taux d'intérêt.

Avec Visicalc, c'est celle de la colonne ou de la ligne immédiatement à droite ou au-dessous de l'endroit où M a été trouvée.

Fonctions logiques

Sur les tableurs, ont été prévues des fonctions logiques dont l'emploi permet de réaliser des choix ou de poser des conditions dans le déroulement de certains calculs. Leur utilisation est très proche de celle des instructions Basic correspondantes. Leur nombre et leur variété sont fonction de la nature du tableur. On ne présentera ci-dessous que les fonctions communes.

AND (liste d'expressions). Cette fonction rend la valeur vraie (TRUE) si les conditions reportées sur la liste ont toutes été vérifiées. Sinon, le résultat est faux (FALSE).
TRUE et FALSE apparaissent dans la cellule où la fonction AND est positionnée avec une signification d'indicateur (flag).

Ainsi, plaçons dans la cellule A7 la fonction :

AND(A1>10,C3 = 5)

on obtiendra le résultat TRUE en A7 si les deux relations sont vraies (A1>10 et C3 = 5). Le message apparaissant en A7 est considéré comme une chaîne (A7 devient une variable chaîne) et peut être appelé par d'autres fonctions. Par exemple, en introduisant dans la case A11 la fonction :

AND(A7 = TRUE,A9 = FALSE)

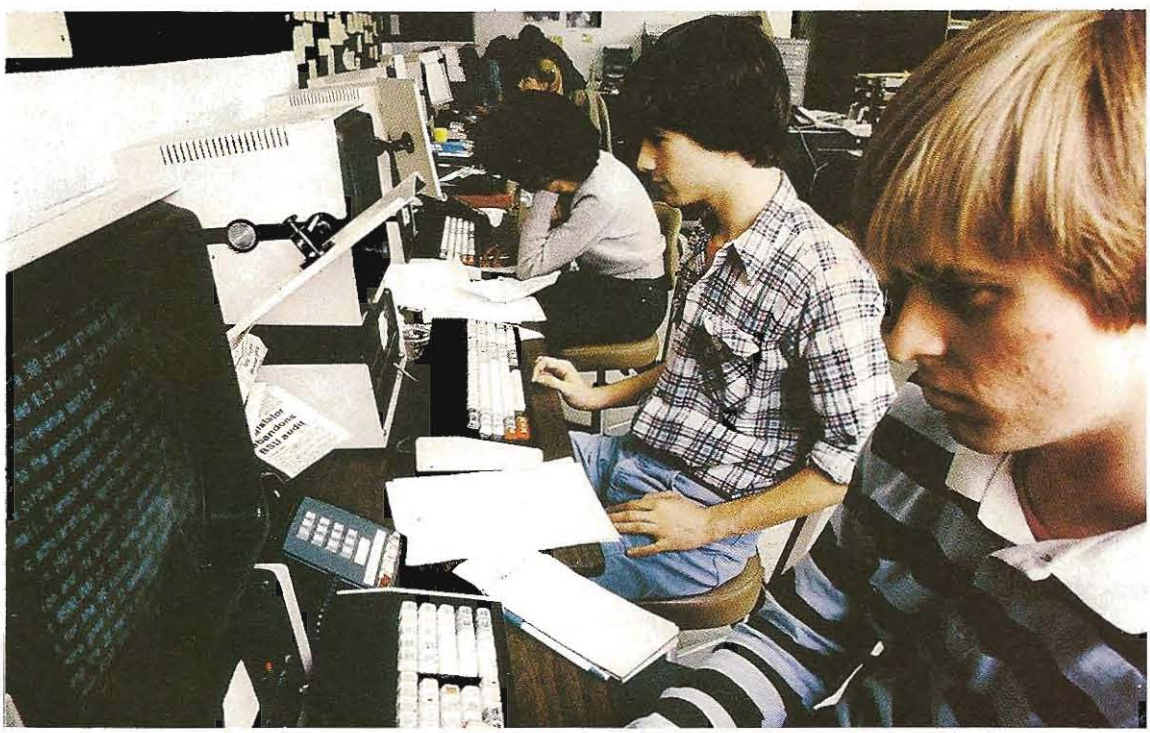
on obtient le contrôle simultané des conditions A7 = TRUE et A9 = FALSE qui, à leur tour, peuvent dépendre d'autres conditions mises en places respectivement en A7 et A9. Ci-contre, un exemple d'application de la fonction AND.

IF (condition vrai/faux). Elle a la même signification que le IF Basic. Une fois vérifiée la condition, le programmeur poursuit son test dans les zones indiquées « vrai » ou « faux » selon que la condition a, ou non, été vérifiée. En plaçant dans la cellule B2 la fonction :

IF(A7 = FALSE,A1 + C3,0)

le résultat du calcul A1 + C3 est transféré en B2 si A7 est fausse (condition vraie), autrement il est placé B2 = 0.

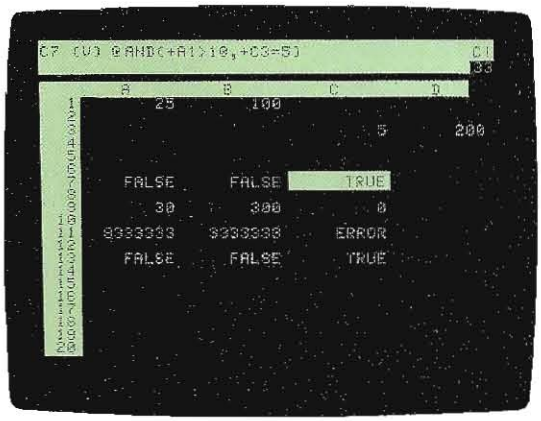
Un centre de traitement de l'université de Maryland aux Etats-Unis.

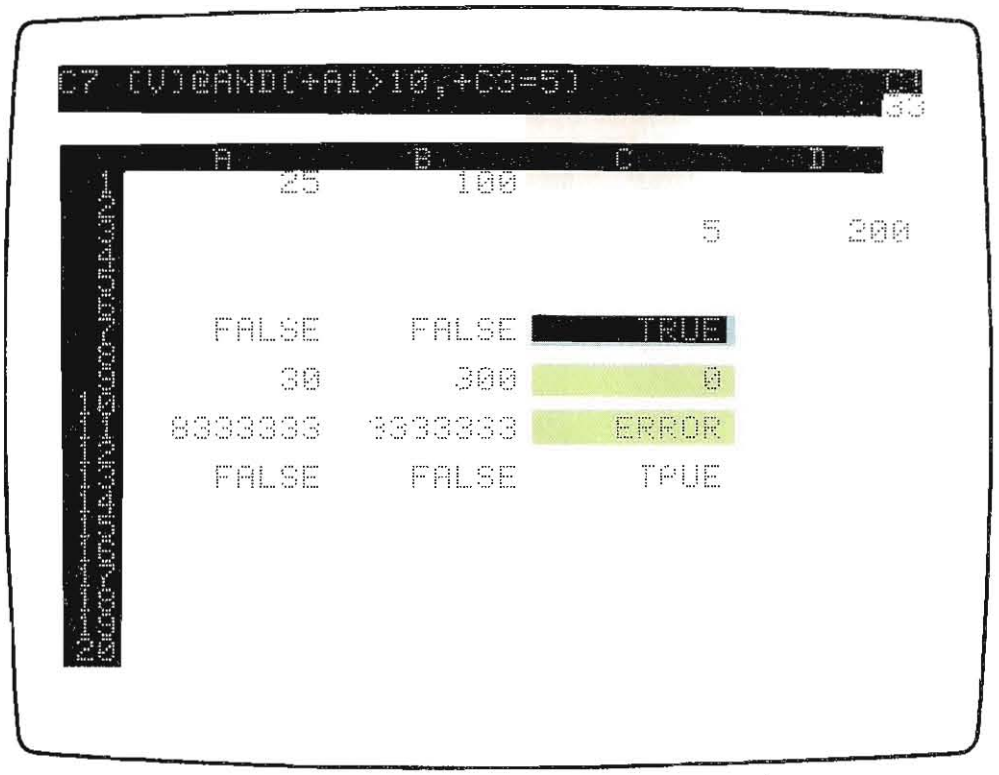


J. Pickerell/Marka

APPLICATION DE LA FONCTION AND

L'écran vidéo visualise un tableau sur lequel certaines expressions logiques utilisant la fonction AND ont été insérées.





Dans la cellule C7, on a introduit la formule @ AND (+ A1 > 10, + C3 = 5).

La réponse est TRUE (vrai) puisque A1 = 25 et C3 = 5. L'inscription TRUE ne remplace pas le contenu original de la cellule C7 (fonction logique). Il s'agit d'un indicateur (flag) qui est mis à jour dès que varient les données dans les cellules intéressées par l'expression logique.

Dans la cellule C11, on a illustré l'utilisation de ERROR. Une expression prenant comme diviseur le contenu de C9 a été mise en C11 ; le système détecte une division par 0 et signale l'erreur. En introduisant une valeur autre que zéro en C9, l'argument ayant motivé l'erreur disparaît automatiquement et le système affiche le résultat du calcul.

Les fonctions logiques sont très utiles pour contrôler les données entrées ou les résultats des calculs. Supposons, à titre d'exemple, que nous désirions obtenir des données statistiques. Les termes du problème sont les suivants. Trois enquêteurs ont échantillonné une population en fonction du sexe : chacun d'eux a communiqué le total des personnes examinées et le nombre de garçons. On veut en déduire le pourcentage de filles dans chaque groupe-échantillon ainsi que leur ratio global.
La préparation de la feuille ne présente aucune difficulté ; il suffit d'insérer un contrôle dans les instructions de calcul. Le nombre de personnes de sexe féminin est obtenu par la différence entre le total et le nombre de garçons :

$$\text{Filles} = \text{Total} - \text{Garçons}$$

Le pourcentage de filles par rapport au total est :

$$P = \frac{\text{Filles}}{\text{Total}} \times 100$$

$$= \frac{\text{Total} - \text{Garçons}}{\text{Total}} \times 100$$

Dans cet échantillonnage, si on retire tous les garçons, le ratio perd sa signification. Dans ce cas, la fonction IF évite un calcul inutile.
L'affichage des valeurs sur le tableur est représenté par le schéma de la page 823. Les données sont placées dans les cellules B2, B4, B6 (totaux) et C2, C4, C6 (nombre de garçons). Les cellules correspondantes de la colonne D reçoivent les pourcentages de filles et la ligne 8, les totaux généraux.
On contrôle les données en affectant des conditions au calcul des pourcentages de filles (colonne D), ainsi qu'aux valeurs contenues dans les cellules correspondantes (colonnes B et C). Dans cet exemple, on aura recours à la fonction IF au moment de calculer le contenu de la cellule D8. Pour les autres cellules, l'expression ne varie pas, mais ce sont les adresses qui changent. Ainsi, pour la cellule D4 (enquêteur n° 2), l'équation devient :

IF(C4<>B4,100*((B4 — C4) / B4),0)

NOT (condition). Elle affiche le résultat FALSE si la condition est vérifiée, ou sinon TRUE. Ainsi,

NOT(A3 = C6)

affecte la valeur TRUE à la cellule contenant la fonction, si A3 ≠ C6 ; et renvoie FALSE si A3 = C6. Le contenu de la cellule (TRUE, FALSE) est utilisable comme indicateur pour conditionner d'autres calculs.

ISNA (cellule). Elle prend la valeur TRUE ou FALSE selon que la fonction NA (Not Available = non disponible) est présente ou non dans la cellule spécifiée.
La fonction NA est un indicateur qui peut bloquer les calculs faisant référence à des cellules n'ayant pas encore été utilisées (par conséquent sans signification aucune).
Soit, par exemple, une cellule d'insertion de données. Pour conditionner un calcul à la présence ou non de valeurs en A1, on place la fonction NA dans la cellule en écrivant A1 @ NA (rappelons que la valeur du symbole @ peut différer selon le système utilisé) et la fonction ISNA (A1) dans une autre cellule (B1 par exemple). Tous les calculs devant être « conditionnés » à la présence de données en A1, devront être écrits en vérifiant la condition TRUE/FALSE en B1 ; par exemple, dans :

IF(B1 = FALSE...)

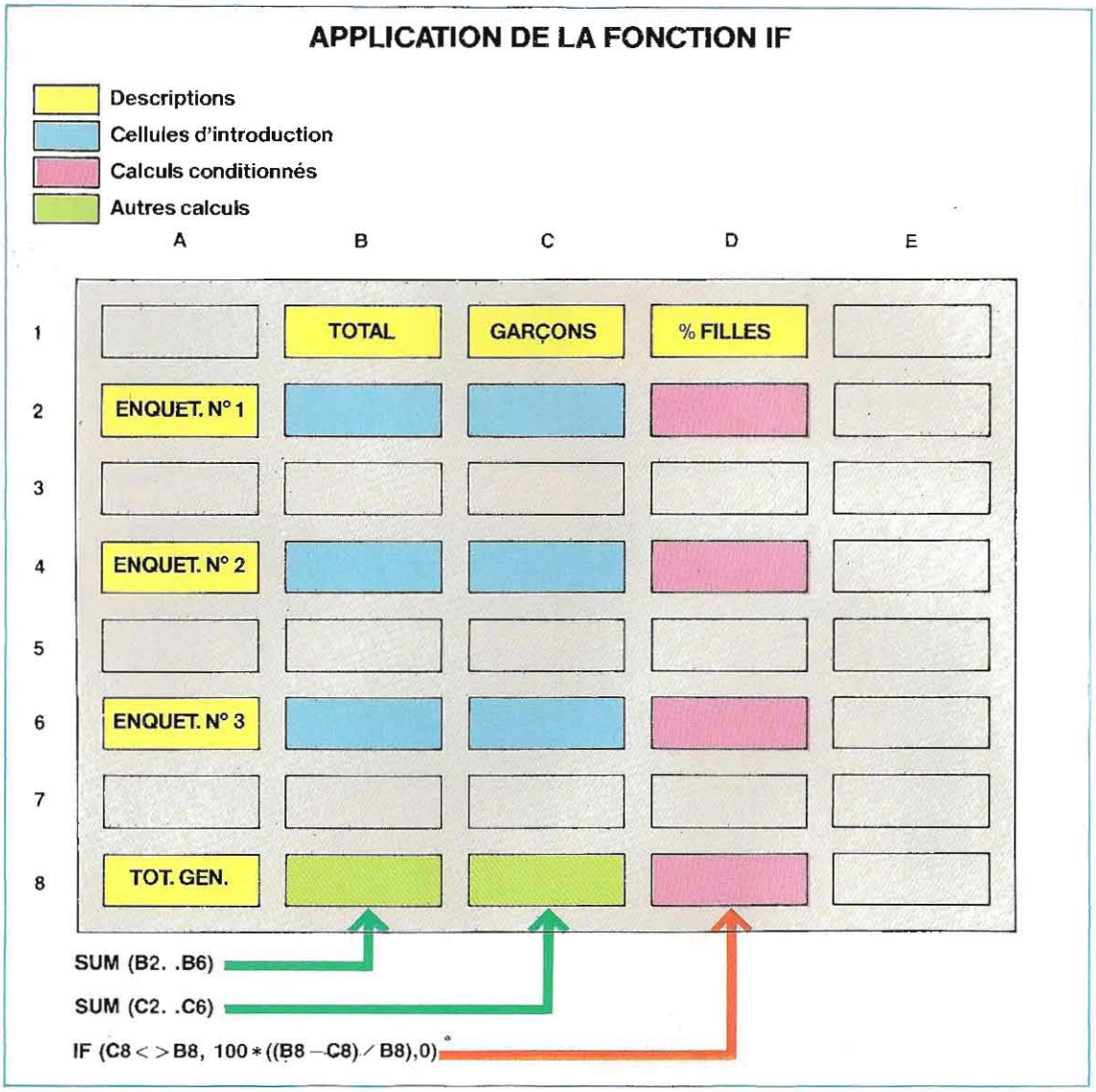
le contenu de B1 est TRUE jusqu'à ce que NA apparaisse dans la case A1. En introduisant une donnée quelconque en A1, FALSE est transféré en B1, et l'on effectue les calculs dans toutes les cellules comportant cet indicateur.

ISERROR (cellules). Elle contrôle l'apparition ou non d'une erreur dans la cellule spécifiée. Ainsi :

ISERROR (B6)

donne TRUE si une erreur a été détectée dans la cellule B6, et FALSE s'il en est autrement. Le message ERROR, dans une cellule contrôlée, alerte l'opérateur sur l'existence d'une situation anormale. Dans le cas où on manipule en même temps un nombre élevé de cellules actives, on procède à l'affichage partiel du tableau, ce qui ne permet pas de repérer

APPLICATION DE LA FONCTION IF



d'éventuelles erreurs dans une des cellules non affichées. Grâce à la fonction ISERROR (.....), l'opérateur garde le contrôle de ce qui se passe dans les zones non affichées.

OR (conditions). Elle a la même signification que OR Basic. Par exemple, si on écrit en A3

OR (A1 = 15, C= > D2,F4 <> 0)

on a A3 = TRUE dès que se vérifie une quelconque des conditions enregistrées.

Fonction sans arguments

D'autres fonctions utilisées par les tableurs donnent des constantes numériques ou logiques (sans argument). Il s'agit surtout de :

PI. Elle fournit la valeur $\pi = 3{,}14159265336$.

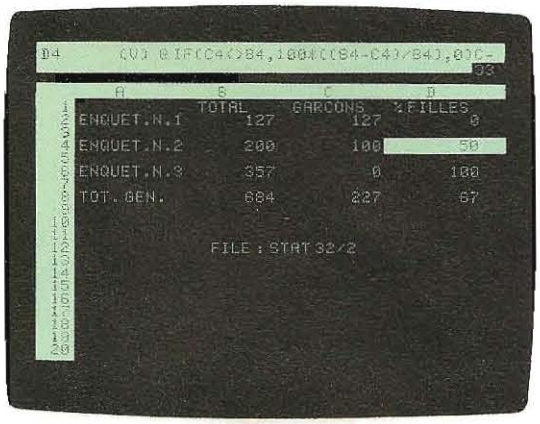
ERROR. Apparaît comme un message, dans toute cellule se rapportant à cette fonction.

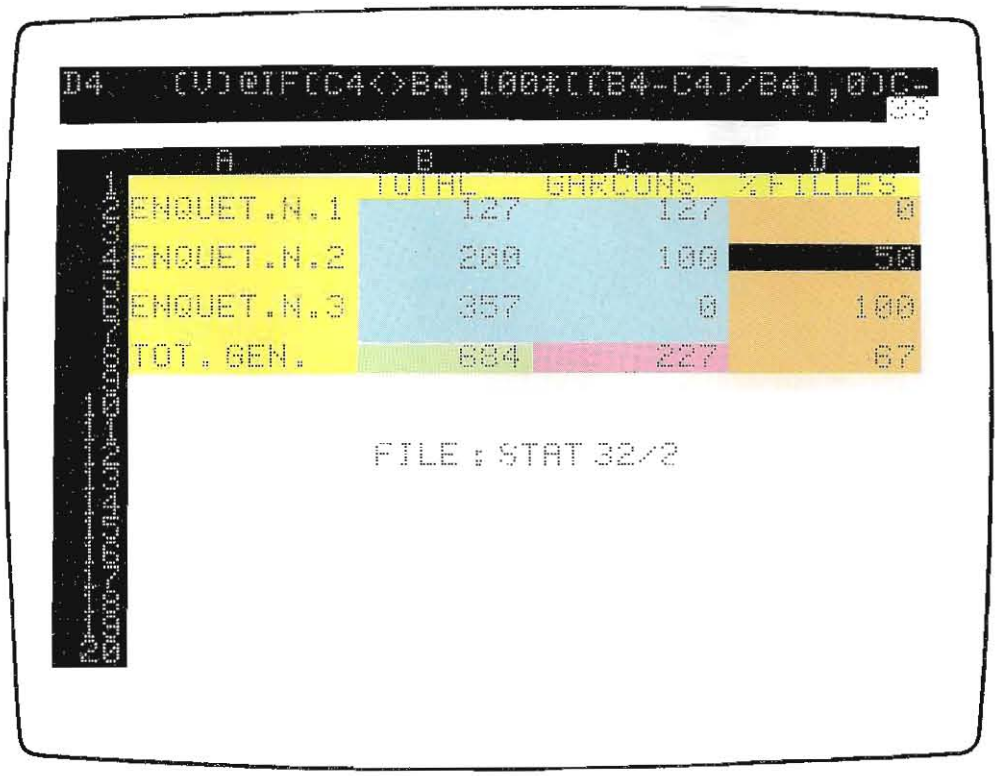
FALSE/TRUE. Fonctions qui affichent les indicateurs correspondants.

NA. Elle interrompt les calculs dans toutes les cellules faisant référence à celle qui contient la fonction NA. Le message NA apparaît aussi dans ces cellules jusqu'à ce qu'une valeur soit introduite dans la cellule contenant initialement NA.

APPLICATION DE LA FONCTION IF

Le monitor affiche le tableau cité en référence dans l'exemple d'application de la fonction IF aux calculs statistiques.





Les cellules de la colonne A et celles de la ligne 1 sont réservées aux descriptions.

Les cellules B2, B4, B6 et C2, C4, C6 renferment le total des personnes examinées par chaque opérateur et le total des garçons.

La cellule B8 contient la fonction SUM (B2..B6), qui déclenche le calcul de la somme des données contenues dans les cellules de la colonne B de B2 à B6.

La cellule C8 renferme la fonction analogue pour le calcul de la somme des données de la colonne C.

Les cellules de la colonne D contiennent le calcul des pourcentages de filles en fonction des données relevées par chaque enquêteur. Le calcul se fait sous condition et, pour toutes les cellules, la fonction introduite est analogue à celle que l'on voit sur la ligne des entrées, relative à la cellule D4.

La fonction IF met en route le calcul de pourcentage 100* ((B4 — C4)/B4) seulement si la condition C4 <> B4 est vraie Dans le cas contraire, elle transfère la valeur 0 dans la cellule.

Exemples d'utilisation du tableur

Le tableur est largement utilisé dans tous les packages de logiciels d'applications de gestion (Visicalc, Multiplan, Lotus, etc.). Il permet, en effet, de résoudre en temps réel des problèmes concernant la gestion et la comptabilité. Facilement accessible, il est structuré de telle manière que l'utilisateur dispose constamment, à l'écran, d'un tableau synoptique du procédé de calcul employé, tout en rappelant beaucoup les pages des livres de comptes traditionnels. Pratiquement, il ne lui est demandé aucun effort d'adaptation. Chaque procédure peut être projetée et structurée d'une façon naturelle et personnalisée, sans qu'il soit nécessaire de pénétrer au cœur des problèmes de programmation.
Trois applications typiques du tableur seront examinées dans les pages suivantes. La première concernera la répartition en millièmes des dépenses concernant les parties communes dans une copropriété ; la deuxième présentera la structuration du compte de stocks d'un magasin. Dans la dernière application, il sera question d'une méthode d'analyse prévisionnelle très répandue en gestion commerciale : le WHAT IF.

Répartition des frais de copropriété

Comme cela se passe dans la majeure partie des pays européens, le système juridique français prévoit que les frais de gestion d'un immeuble en copropriété doivent être répartis entre les différents copropriétaires, proportionnellement à un certain coefficient exprimé en ‰ (= pour mille). Ces millièmes traduisent quelle quote-part de toute la copropriété représente chaque propriété individuelle. Par convention, la valeur de la copropriété est fixée à 1 000 millièmes. Une quote-part de 100 millièmes, attribuée à un appartement en copropriété, signifie que les charges de copropriété qui le grèvent sont équivalentes à un dixième (100/1000 = 1/10 = 10 %) de la charge totale. Le propriétaire de l'appartement devra donc payer 10 % de tous les frais engagés par le syndic, du moins de ceux qui sont considérés comme devant être supportés par l'ensemble de la copropriété.

Assemblage des cartes dans une ligne de production



C. Pozzoni/Marka

REPARTITION DES FRAIS EN FONCTION DES MILLIEMES

Expressions (Formules) — Fonctions

Introductions non récurrentes — Introductions récurrentes

	A	B	C	D	E	F	G	H
1		REPARTITION DES MILLIEMES			MONTANTS			TOTAL
2								
3	APPAR-TEMENT	TAB.1	TAB.2	TAB. 3	TAB. 1	TAB. 2	TAB. 3	
4	1	250	65	120	+B15*B4/1000	+B16*C4/1000	+B17*D4/1000	@SUM(E4..G4)
5	2	100	35	80	+B15*B5/1000	+B16*C5/1000	+B17*D5/1000	@SUM(E5..G5)
6	3	50	290	250	+B15*B6/1000	+B16*C6/1000	+B17*D6/1000	@SUM(E6..G6)
7	4	300	410	150	+B15*B7/1000	+B16*C7/1000	+B17*D7/1000	@SUM(E7..G7)
8	6	100	100	300	+B15*B8/1000	+B16*C8/1000	+B17*D8/1000	@SUM(E8..G8)
9	10	200	100	100	+B15*B9/1000	+B16*C9/1000	+B17*D9/1000	@SUM(E9..G9)
10								
11								
12								
13								
14								
15	Dépense 1	500						
16	Dépense 2	360						
17	Dépense 3	2500						

La structure de la feuille, pour le calcul de la répartition des frais de copropriété, est schématisée ci-dessus sous forme de tableau.
Dans la colonne A (qui sera la colonne A du tableur), on a indiqué les numéros des appartements d'une même copropriété.
Dans les colonnes B et C, on a reporté les millièmes relatifs à chaque appartement pour trois différents types de dépense en prévoyant que des dépenses différentes puissent se voir affecter de coetficients différents (pour les cas spéciaux). Le tableau ci-dessus prévoit trois répartitions mais peut être étendu à volonté. La deuxième partie du tableau (colonnes E, F, G) reporte la quote-part de chacun. La dernière colonne représente le total dû par chaque copropriétaire, à savoir la somme des trois montants partiels correspondants.
On introduit chaque dépense aux lignes 15, 16, 17; la première renverra au tableau 1 (colonne B), la seconde au tableau 2 (colonne C), etc.
La portion de dépense attribuée à chaque copropriétaire est le produit du chitfre total par les millièmes du tableau relatif à cette dépense divisé par 1000. La préparation du tableau se fait en deux temps :

La bureautique (2)

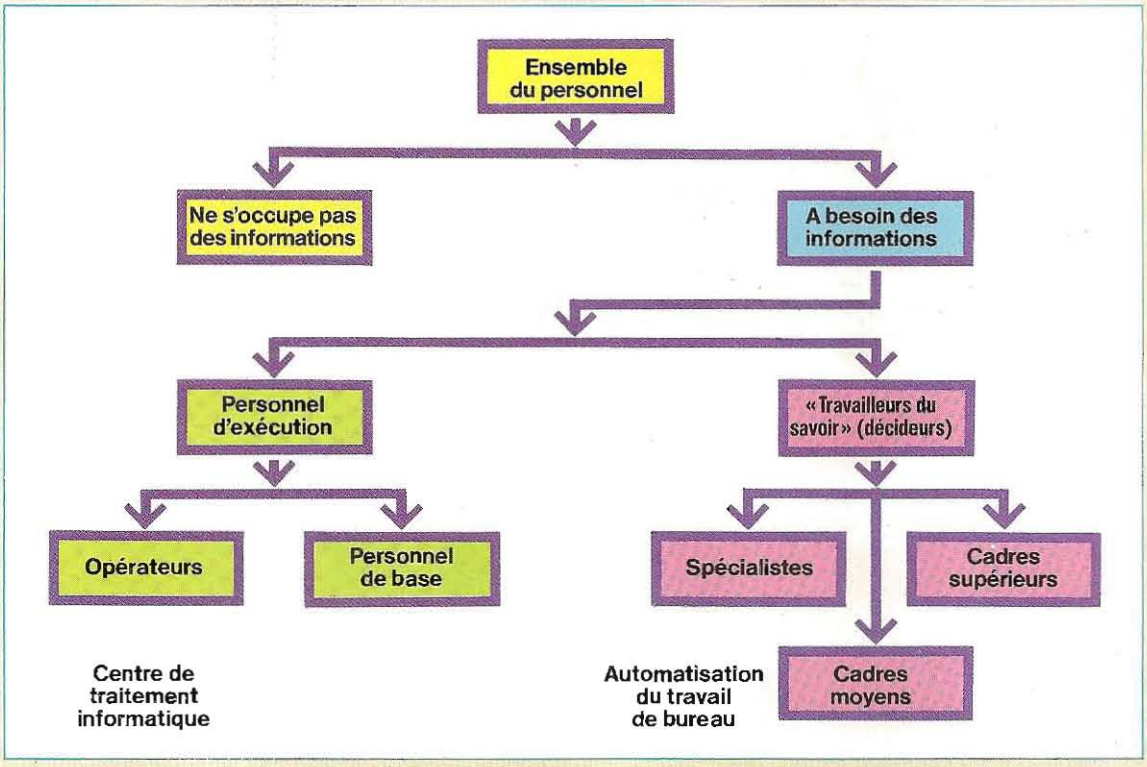
A la lumière de l'analyse menée jusqu'ici, il apparaît clairement que, par automatisation du travail de bureau, on entend :

- un modèle de ***technique d'organisation*** fondé sur l'utilisation coordonnée d'instruments automatiques de communication et de traitement de données dans le but de fournir directement au manager, au cadre et à l'employé responsable des informations et des services ;

- une technique qui inclut :
  le traitement de textes
  le courrier électronique
  la consultation d'archives
  le calcul
  l'aide à la décision
  l'aide au déroulement des tâches à un titre personnel (agenda, échéancier, etc.)
  ainsi que d'autres applications particulières ;

- des services qui doivent être constamment disponibles grâce à une interface intégrée évitant le recours à des moyens intermédiaires.

Une telle définition exige, bien entendu, quelques compléments.
Le modèle nécessite, sans aucun doute, une composante technologique qui, à la différence de ce qui se passait il y a quelques années, fait aujourd'hui l'objet d'un suivi sur le plan économique. En fait, il s'agit surtout d'un modèle d'organisation dans la mesure où il concerne la rationalisation du travail de bureau et donc le degré de professionnalisme des responsables. Cela implique que ***l'intérêt*** *d'une telle automatisation des tâches soit* ***très bien perçu*** *par les personnes concernées.*
On a déjà mis en évidence, plus haut, l'impact de la personnalisation des services et de la définition précise des rôles de chacun sur la rentabilité du travail de bureau.
Les fonctions énumérées précédemment comprennent, bien sûr, les activités de traitement des informations que nous avons qualifiées d'élémentaires, mais aussi celles à caractère plus évolué dont l'automatisation qui n'est réalisable qu'en termes d'individu si l'on tient à être vraiment efficace. Il s'agit de procédures simples d'analyse et de traitement des données stockées en mémoire, destinées à la ***planification des tâches*** *et à* ***l'aide à la décision.***
Par conséquent, pour l'accès aux différents services, on ne doit jamais perdre de vue la nécessité d'une interface unique qui ne dresse pas de barrières absurdes entre le monde de l'élaboration des données et celui du bureau.
Dernière exigence, le destinataire de ces services doit pouvoir les utiliser directement, sans intermédiaire. Ceci constitue une autre grande différence avec l'informatique classique.
On admet aisément cet impératif quand il n'est pas possible de sous-traiter la recherche de l'information. En effet, le choix des informations dépend étroitement de la personnalité de l'individu et de sa méthode de travail. Autrement dit, seule la personne ayant besoin des dites informations est en mesure de réaliser une sélection judicieuse parmi toutes celles dont elle dispose et de pallier ainsi l'absence éventuelle de certaines autres.
L'absence de rôles intermédiaires nécessite une révision radicale des modalités du dialogue entre l'homme et la machine, révision actuellement en cours mais pour laquelle il reste encore beaucoup à faire. Outre la linguistique, la branche de l'informatique, connue sous le nom d'intelligence artificielle, devrait donc constituer, avec la psychologie de la connaissance, un terrain d'application idéal.
Aucune entreprise, désireuse de maintenir sa compétitivité, ne peut se tenir à l'écart de la rationalisation et de l'automatisation. Une fois analysées et classées les raisons qui en font un choix obligé, notre propos est de fournir certains éléments pratiques : planification, mise en route et coût en particulier.
L'aspect ***économico-financier*** *doit faire l'objet d'une grande* ***attention*** *de la part des cadres, étant donné l'importance de l'investissement à moyen et à long terme (à court terme, l'investissement peut être très modeste si on se limite à un secteur bien précis). Reprenant ce qui a déjà été évoqué, le schéma de la page 828 donne une indication des fonctions que le processus d'automati-*



Organisation opérationnelle des fonctions de l'entreprise.

sation au sens large du terme devra prendre en charge.

Parmi les personnes qui utilisent l'information, on distingue le personnel d'exécution ou d'assistance des « travailleurs du savoir » *(décideurs) qui sont les cadres de l'entreprise, ayant un haut niveau de responsabilités et de compétence tant dans la spécialisation que dans l'organisation.*

Parmi le personnel d'exécution, on trouve les personnes qui traitent les informations (opérateurs), celles qui mettent en œuvre les procédures de gestion définies ci-dessus, le personnel de base, essentiellement constitué par des secrétaires, des dactylos, des archivistes et dont la tâche consiste à rendre ***plus productif*** *le travail des décideurs.*

Comme on l'a déjà vu, le travail des opérateurs et l'automatisation sont les points sur lesquels nous avons le plus insisté en informatique classique. L'informatisation du travail de bureau, en revanche, présente d'autres priorités.

A titre d'illustration, le tableau suivant fournit la répartition du temps de travail quotidien (et donc du coût) des spécialistes et des cadres de direction moyens et supérieurs.

Activités	Spécialistes %	Cadres moyens %	Cadres supérieurs %
Communication	38	54	64
Création de documents	15	11	8
Analyse	9	7	4
Lecture	4	8	6
Autres (non directement producteurs)	34	20	18

Les cinq activités ainsi réparties englobent la totalité des fonctions remplies par les trois catégories indiquées.

L'activité ***communication*** *comprend les réunions (avec d'éventuels délais de déplacement) et les conversations téléphoniques.*

L'activité ***création de documents*** *(ou encore messages) inclut les annotations et la*

redistribution de documents réceptionnés.
L'analyse de données ou de situations doit être entendue comme un stockage et un examen d'informations aidant à la prise de décisions, et non comme une simple lecture qui correspond à des objectifs plus généraux d'enrichissement des connaissances professionnelles.
Enfin, font partie des **activités non directement productrices,** l'attente d'un événement, l'organisation de son travail personnel ou de celui de ses collaborateurs, la coordination ponctuelle d'activités, la recherche d'informations ou de personnes (directement ou au téléphone), l'archivage, les codages, etc.
Compte tenu de l'effectif moyen du personnel dans les trois catégories, il est intéressant de relever (comme dans le tableau ci-dessous) que l'évaluation objective de la répartition du temps dans les cinq activités est loin de l'évaluation subjective, faite par les représentants de ces mêmes fonctions.

Activités	% du temps total		% Ecart
	Estimé	Effectif moyen	
Communication	29,1	45,6	− 36
Création de documents	12,7	12,9	− 2
Analyse	14,6	8,2	+ 78
Consultation	9,1	7,9	+ 20
Autres (non directement productrices)	34,5	25,4	+ 34

En particulier, on peut constater combien l'activité communication est notoirement sous-estimée par les intéressés alors que sont surestimés les chapitres concernant l'analyse et les activités non directement productrices. L'écart, dans le temps de communication, se justifie compte tenu de la tendance naturelle à considérer le mécanisme d'échange d'informations comme plus efficace qu'il n'est en réalité.
S'agissant de l'analyse, la tentative consistant à donner plus de poids à une activité jugée valorisante est évidente. Pour les tâches non productrices, en revanche, on constate immédiatement que leur surévaluation provient du jugement négatif porté presque inconsciemment à leur égard. Ceci s'explique par le fait qu'elles ont un caractère opérationnel non conforme à l'image qu'on en a.
De toutes façons, et pour en revenir à la répartition effective, on sait qu'une part supérieure à 70 % du temps de travail est consacrée à la communication et aux activités non directement liées à la production. Cette constatation suffit pour formuler une première **stratégie d'intervention**, à la fois simple, et claire dans ses objectifs.
C'est précisément en partant d'activités « improductrices » et en analysant leur structure que l'on se rend compte qu'il est aisé de les confier au personnel d'assistance, à condition d'en augmenter l'effectif, ou mieux encore, en le dotant de moyens (medias) qui augmenteront sa rentabilité.
Correctement menée (c'est-à-dire graduellement et avec les investissements nécessaires en formation et en automatisation), cette première intervention assure deux avantages significatifs pour l'organisation du travail. Le premier concerne l'amélioration des performances du personnel d'assistance; le second, l'allègement de la charge de travail dans les deux secteurs de la spécialisation du management.
Ces avantages se traduisent par une disponibilité en temps qui, à engagement égal, pourra être consacrée à des tâches institutionnelles.
Du point de vue de la motivation, dans les deux cas, on va dans le sens d'une amélioration des compétences dans l'entreprise. On trouvera ci-après une comparaison entre les temps moyens nécessaires à la communication téléphonique et au courrier électronique.

MESSAGE TELEPHONIQUE

1. Seulement 26 % des appels ont une suite dès le premier essai.
2. Sur les 74 % qui restent :
   Le destinataire ne peut être dérangé : 38 %
   Personne ne répond : 38 %
   La ligne est occupée : 14 %
   Autres causes : 10 %
3. Il n'est pas possible de rappeler le message (à moins de dicter - éditer - archiver - rechercher)
4. Difficultés pour faire parvenir le même message à plusieurs destinataires
5. Ce poste n'est pas économique
6. Les phrases de politesse ne peuvent être éliminées.

MESSAGE ELECTRONIQUE

1. Issue toujours positive : la présence du destinataire n'est pas nécessaire
2. L'activité du destinataire n'est pas interrompue
3. On peut vérifier qu'il a été reçu
4. Recherche simplifiée d'un destinataire
5. L'envoi de circulaires est simple
6. Il réduit les délais de transmission

Temps moyen message téléphonique	4,8 min
Temps moyen message électronique	1,3 min
Gain moyen par message :	3,5 min

Le courrier électronique consiste à transmettre un message à un ou à plusieurs destinataires, sans que les interlocuteurs se trouvent nécessairement devant leur terminal. Ce système mémorise les ***informations*** *et les rend* ***disponibles à la demande.***

Comme on peut le constater, le courrier électronique présente un ensemble d'avantages qui le rendent de loin supérieur au téléphone. Parmi ceux-ci, il convient tout particulièrement de noter que l'issue du message est toujours positive, c'est-à-dire que l'expéditeur ne perd pas de temps à refaire plusieurs fois la même tentative. Quant au destinataire, il n'est pas distrait des tâches qui l'occupent. Et le gain de temps dépasse très largement celui que la réception du message électronique épargne : le destinataire n'a pas à reprendre son dossier et à perdre du temps à retrouver où il en était exactement arrivé de sa tâche. En outre, il élargit l'éventail des moyens de communication dans le bureau.

Une étude menée par le Stanford Research Institute prévoit même que l'utilisation systématique et exclusive du courrier électronique dans les activités de direction ferait gagner 20 % du temps. Grâce à lui, on rationalise les échanges d'informations, pouvant emprunter trois circuits de communication différents, en fonction de leur contenu. Les thèmes à caractère « ***politique*** *» seront toujours traités au cours de réunions en* ***petit comité.*** *Ceux qui nécessitent des prises de décisions urgentes seront débattus au téléphone ; quant aux autres, ils pourront passer par ce nouveau canal dont l'intérêt est une meilleur planification du travail. (Il faut savoir, en effet, que les interruptions et les pertes de temps dues aux communications traditionnelles atteignent une heure et demie par jour.)*

Les moyens nécessaires à la mise en œuvre d'une telle stratégie sont également mis en évidence et tiennent compte de notre connaissance des techniques modernes de communication.

Le tableau ci-dessous donne une analyse du facteur d'amélioration de la productivité.

Media	Gain potentiel de temps (en %)	% du gain total de temps
Téléconférence	0,5	2,7
Transfert des informations	4,4	29,9
Mémorisation et repérage des informations	4,2	28,6
Elaboration personnelle	4,0	27,2
Gestion des activités	1,7	11,6
Total	14,8	100

Plus précisément, le transfert d'informations *concerne l'utilisation du courrier électronique. La* mémorisation et le repérage des informations *se rapporte à la possibilité d'emmagasiner les données et de les rendre accessibles à qui de droit. La* gestion des activités *représente celles qui ne sont pas directement productrices telles que les services d'agenda électronique ou d'échéancier. Par* élaboration personnelle, *on entend la possibilité de donner à l'utilisateur (spécialiste, cadre moyen, cadre supérieur) une capacité autonome de création de séquences de calcul simples sur les données l'intéressant particulièrement et justifiées par une certaine répétitivité. Enfin, la* ***téléconférence*** *est un excellent moyen de communication entre personnes géographiquement éloignées. Elle vient en plus du téléphone et du courrier électronique pour les cas particulièrement délicats nécessitant plusieurs avis. La téléconférence permet de mener des réunions par des canaux de communication audio, ou même vidéo (utilisant des fréquences deux fois supérieures aux fréquences phoniques, et donc aujourd'hui encore à des coûts prohibitifs). Cela fait des années que la NASA exploite cette technique*

avec succès mais l'Europe ne la connaît pas encore bien. En effet, mis à part son coût, elle exige un temps d'adaptation certain.

Pour ce qui concerne la gestion des activités (traitement automatique des tâches, de l'échéancier, de la planification des ressources communes) il faut quand même reconnaître que la contribution des medias à l'amélioration du travail spécialisé et de direction est encore relativement modeste.

Au terme de cette analyse des perspectives offertes par l'automatisation du travail de bureau – ce qui sous-entend une meilleure rationalisation – certains problèmes restent à considérer.

Il a été largement démontré qu'une entreprise moderne ne peut longtemps différer la décision de s'informatiser, en tout cas pour ce qui relève du traitement de l'information.

Dans le cadre général, on a débattu des différences : de ce qui serait à conserver et de ce qu'il faudrait modifier des procédures dans le cadre de leur informatisation. A partir de là, des **stratégies d'automatisation** ont été mises au point. Le thème de la structuration des activités de bureau n'a pas été vraiment abordé. En fait, la stratégie d'intervention suggérée ici, et qui se limite à l'analyse des rôles tenus par les personnels concernés par l'automatisation, élude cet aspect.

Sur de telles bases, la planification d'une intervention dans le cadre du bureau peut être schématisée de la façon suivante.

Le modèle de départ est constitué par l'univers des utilisateurs potentiels des systèmes d'automatisation du travail du bureau ; on définit les **profils de ces utilisateurs** à l'aide de questionnaires.

On repère ensuite les moyens existants capables d'améliorer les performances. Ils tiennent compte, avec précision, des choix informatiques de l'entreprise.

Etant donné l'importance de l'intervention et de son impact sur l'organisation du travail, le projet doit faire partie des objectifs et des stratégies de l'entreprise. Ceci est encore plus vrai dans le cadre d'une informatique classique. Ainsi, **opportunités** et **ressources** doivent être conciliées avec les plans de développement de l'entreprise, à moyen et à long terme (3-5 ans).

Mais qui doit **assumer la responsabilité** de ce plan ? La réponse à cette question nous est donnée par une enquête menée, fin 81, par la revue « Computer World » auprès de 400 entreprises abonnées.

- La moitié des entreprises assigne cette responsabilité à son service informatique, sa compétence étant jugée comme un préalable nécessaire même à l'automatisation du bureau. Les contraintes sévères d'intégration entre les deux mondes plaident en faveur de cette thèse.
- 20 % constituent un comité de diffusion ad hoc, dans lequel, outre l'informatique, sont représentés tous les secteurs intéressés par le projet pour impliquer au maximum l'ensemble du personnel. On est en droit de penser, même si cela n'est dit par l'enquête, que les propositions et les opérations spécifiques relèvent dans ce cas du secteur informatique.
- Enfin, 30 % des entreprises interrogées estiment qu'il s'agit là d'un problème de coût, donc relevant de l'administration. Rappelons toutefois que, même aux Etats-Unis, le service informatique fait souvent partie de la direction administrative.

En continuant d'analyser les réponses des entreprises à ce questionnaire, il est intéressant de noter l'importance de plus en plus reconnue de l'informatisation du travail de bureau.

Comme les résultats montrent qu'une infime minorité (3 %) ne s'est toujours pas, fin 81, décidée à examiner le problème, on en déduit que la quasi totalité des entreprises a déjà réalisé des projets, qu'ils sont en cours, à moins qu'on ne soit en train de se livrer à une étude d'opportunité.

Cependant, pour les raisons déjà exposées ci-dessus, c'est sur le terrain du traitement de l'information que la **compétitivité du système économique** sera mesurée dans un avenir proche. C'est sur ce même terrain que l'Europe aura à relever le défi lancé par les Etats-Unis (et, ne l'oublions pas, par le Japon) sous peine de ne plus faire partie du peloton de tête.

Il convient donc de prendre des décisions le plus vite possible afin de rattraper le retard et d'être en mesure de tirer le maximum d'atouts des expériences réalisées.

(Extrait de « L'informatique dans le travail de bureau », CAHIERS D'INFORMATIQUE, dixième année, n° 1, 1983 Honeywell Information Systems.)

- introduction et mémorisation sur disque des millièmes et des expressions. A partir de là, les données sont stockées en mémoire et rappelées à tout moment pour mise à jour, modification ou édition,
- introduction des dépenses. La feuille de calcul préparée auparavant est rappelée en mémoire et toute nouvelle dépense est introduite dans le chapitre correspondant.

Le programme se charge automatiquement de faire tous les calculs et de présenter les résultats. En demandant l'impression, on obtient le détail des calculs.
A la manière dont il est structuré, le tableur ne sait pas additionner des dépenses. Ainsi, en introduisant une dépense à la ligne 15, le programme réalise le décompte des dettes de chacun mais uniquement pour cette dépense. Si une nouvelle somme est entrée, le tableur effectue une répartition identique de cette nouvelle charge et non pas le total des montants accumulés. Il faut donc insérer une nouvelle colonne qui accumule les introductions successives et qui présente le total général pour chaque copropriétaire.
On peut ainsi mémoriser séparément la colonne des totaux pour la rappeler dans un tableur suivant qui l'additionnerait aux résultats précédents. On obtient la mémorisation des données en les transférant dans un fichier avec des commandes semblables à celles utilisées pour tout le tableau (SAVE, LOAD, etc.) avec, en plus, des caractères identifiant la nature particulière du fichier.
Ainsi, le symbole # dans certains tableurs indique que l'on veut stocker des données. Les paramètres à fournir sont alors : le nom du fichier (données), la ligne ou la colonne à transférer. Par la suite, d'autres tableurs peuvent utiliser les valeurs ainsi mémorisées en les chargeant de la même façon (par exemple la commande L = LOAD, suivie du symbole #).
On doit faire très attention aux valeurs que l'on transfère. Dans cet exemple, la colonne H, qui se présente comme une série numérique (somme des montants partiels), ne contient pas réellement de valeurs numériques mais seulement les expressions nécessaires à leur calcul. En mémorisant cette colonne comme si elle contenait des données, on peut éventuellement provoquer des erreurs.

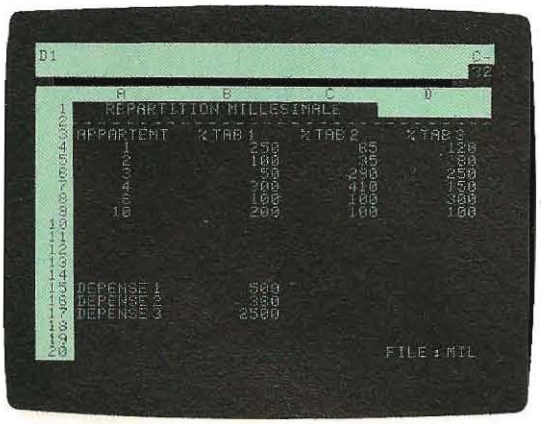
Deux cas peuvent se rencontrer : soit le programme ne transfère rien, dans la mesure où il reconnaît la présence d'expressions, soit il en effectue le transfert. En entrant cette colonne dans un nouveau tableau, des valeurs erronées seront affichées puisque les expressions seront appliquées d'une manière irrégulière. Pour surmonter cet obstacle, il suffit de créer une colonne intégrant la précédente. Le transfert se passe au niveau numérique et l'on obtient une colonne qui renferme les résultats des calculs, non des expressions. Ainsi, le calcul @ SUM (E4...G4), (somme des valeurs comprises entre E4 et G4) se trouve-t-il en H1. Le résultat est une valeur numérique présentée comme telle en H1, mais le véritable contenu de la cellule est une expression. Même en transférant cette cellule « expression » dans un nouveau tableau (si le programme l'autorise), le résultat ne sera pas celui qu'on attend.
Pour copier le contenu (numérique) de la cellule H1 (où a lieu le calcul), il faut avant tout transférer la valeur résultante dans une autre cellule du même tableau, puis la mémoriser sous la forme de données. Elle sera ainsi accessible à d'autres tableaux. Le transfert de la valeur numérique à l'intérieur du même tableau ne peut avoir lieu avec la commande de copie car celle-ci transférerait le contenu de la cellule (c'est-à-dire l'expression) et non le résultat.
Pour transférer le résultat, il suffit d'introduire, dans la cellule d'arrivée, l'adresse (les coordonnées) de la cellule de départ. Ainsi, en écrivant l'adresse + H1 dans la cellule L1, on obtient le transfert de la valeur contenue en H1 dans la cellule L1.
Le symbole + qui précède H1 est nécessaire pour signaler au programme qu'il s'agit d'une adresse et non d'une chaîne de caractères. En cas d'omission, H1 serait transféré en L1 sans aucune indication de commande.

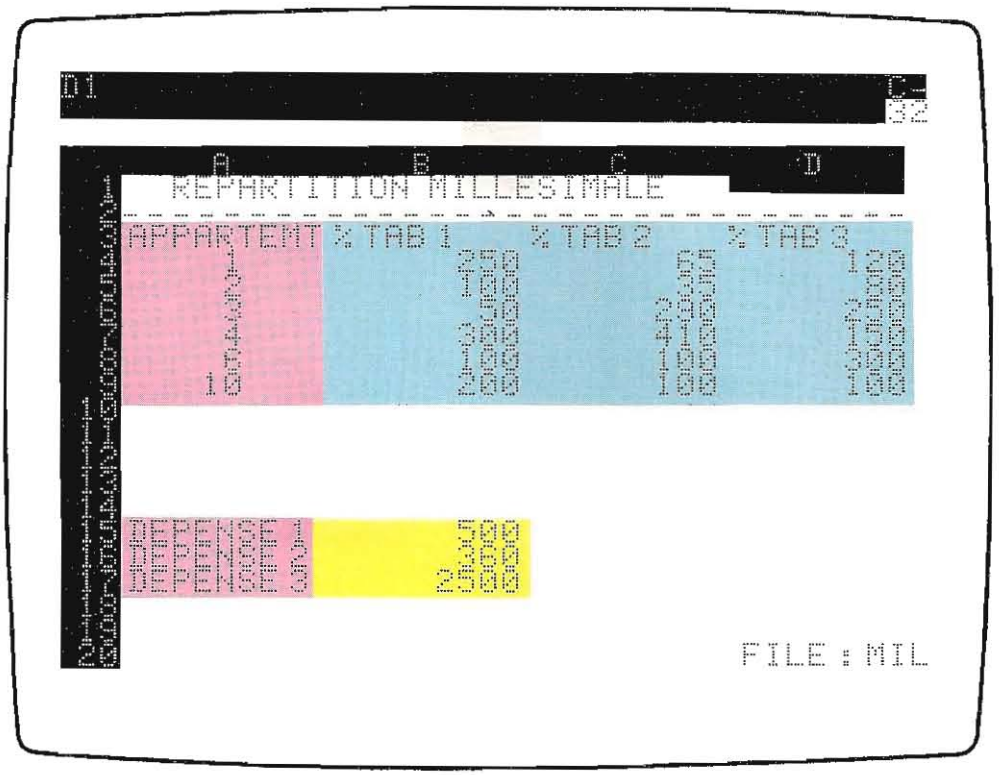
Gestion d'un magasin

En exploitant l'extraordinaire capacité de mémoire des ordinateurs personnels, on en arrive à développer des applications complexes.
L'utilisation des tableurs évite le lourd investissement exigé par le développement du logiciel et réduit le coût d'une application. Dans tous les cas, même pour l'utilisation des

REPARTITION DES FRAIS DE COPROPRIETE EN FONCTION DES MILLIEMES (1)

Cette page et la suivante illustrent l'utilisation du tableur pour le calcul de la répartition des frais de copropriété sur la base des millièmes.





La colonne A renferme une série d'indications variables (numéro de l'appartement de chaque immeuble et décomposition des dépenses).
Si l'on veut apporter des modifications au contenu de la colonne A, il faudra éliminer l'option TITLE, modifier le contenu d'une ou plusieurs cellules et rétablir l'option.

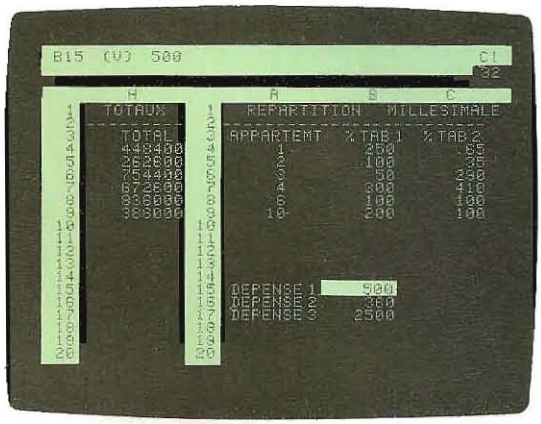
Les colonnes B, C, D renferment les millièmes exprimés en pour mille. Dans l'illustration, on a adopté le symbole % (pour cent), car le symbole exact (‰) ne figure pas sur le clavier ASCII du standard américain.
Chacune des 3 colonnes renferme une répartition différente, liée à un indice de dépense particulier. Les millièmes ne varient pas et doivent être entrés une fois pour toutes.

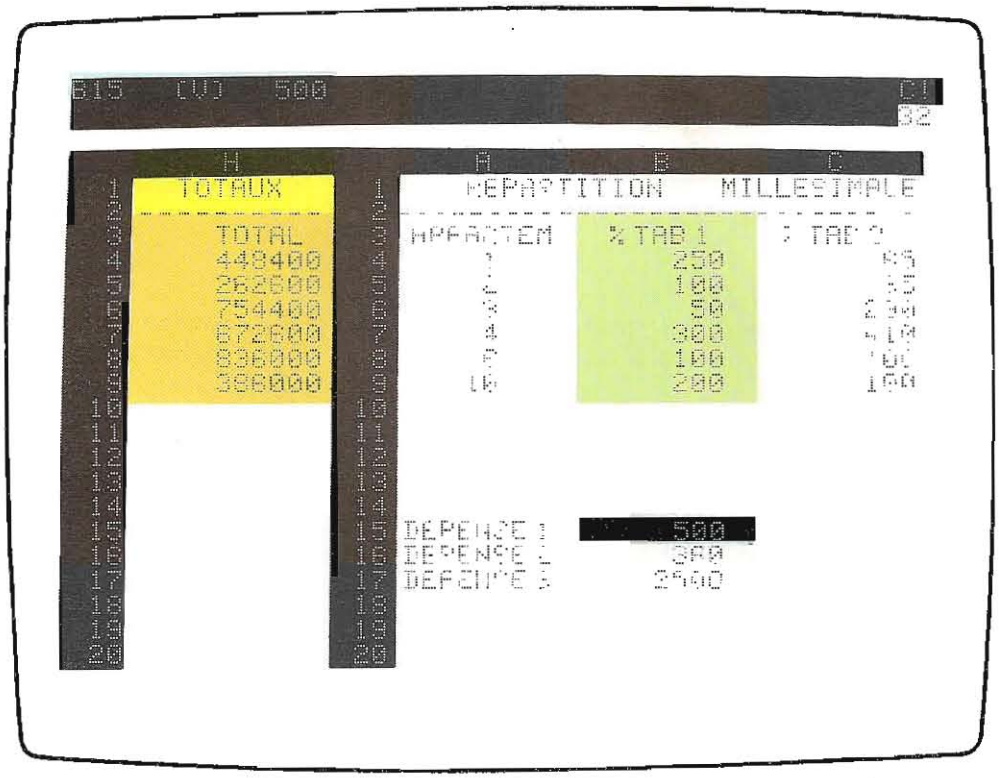
Les cellules B15, B16, B17 sont réservées à l'entrée des sommes à répartir. Après l'entrée d'une valeur, les répartitions sont calculées d'après les millièmes correspondants.

La présentation des montants nécessite le scroll (déplacement) latéral du tableau, qu'on obtient en déplaçant le curseur vers la droite, au-delà de la limite de la zone affichée.

REPARTITION DES FRAIS DE COPROPRIETE EN FONCTION DES MILLIEMES (2)

L'écran illustre l'utilisation d'une option particulière permettant de créer une fenêtre dans le tableur.





La colonne H, la première apparaissant sur la gauche, est en réalité positionnée à l'extrême droite du tableau. Si l'on utilise la commande W = WINDOW (fenêtre) on obtient l'affichage simultané des deux zones du tableau physiquement éloignées, de manière à les comparer directement.

Le curseur, positionné en B15, provoque l'introduction du montant de la dépense 1, mis en évidence à la ligne des introductions en tant que valeur numérique (V).

Pour chaque appartement, la quote-part des dépenses est calculée automatiquement d'après le tableau 1 (colonne B).

Les quote-parts calculées sont cumulées dans les totaux pour chaque appartement.



Globe Photos/Marka

Une opératrice aux unités bandes d'un centre de traitement informatique. Sur la gauche : une série d'unités de disques.

tableurs, il faut procéder à l'analyse du problème. Le but de cette phase ne consiste plus à préparer un cahier des charges mais à déterminer la meilleure méthode de structuration pour une utilisation optimale des possibilités du programme.
Ainsi, la gestion d'un magasin peut consister en une simple énumération des articles et des quantités disponibles ou bien être structurée de manière à permettre le calcul des marges, des simulations...
L'ensemble des fonctions est classé en quatre différents groupes :

- **Identification.** Cette fonction contient les caractéristiques de chaque article, son code, le délai d'approvisionnement, son coût unitaire.
- **Mouvements.** Celle-ci fournit la situation comptable du magasin. Les données traitées concernent l'évolution des entrées et des sorties, l'état des stocks.
- **Valorisation.** On y trouve les données précédentes valorisées d'après les coûts unitaires.
- **Coûts de production.** Si les composants du magasin entrent dans la fabrication de produits finis, le programme calcule automatiquement les quantités entrant dans leur composition ainsi que leurs coûts.

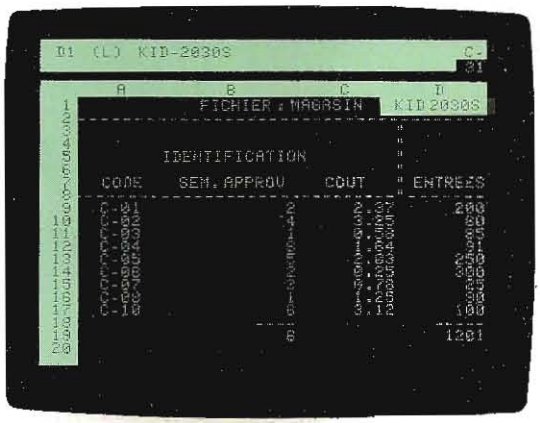
On trouvera un exemple d'application pages 836 à 840.
Pour garder un contact permanent avec l'écran, on a préféré illustrer l'exemple (identification, mouvements, coûts de production...) avec une séquence d'écrans. Mais cette sortie n'est pas la seule possible : on peut faire aussi des copies papier de tout ce qui est affiché. Le transfert, vers l'imprimante, du contenu de la feuille (hard copy) est généralement prévu dans le programme de gestion du tableur. On trouvera, page 840, un exemple de sortie sur imprimante.

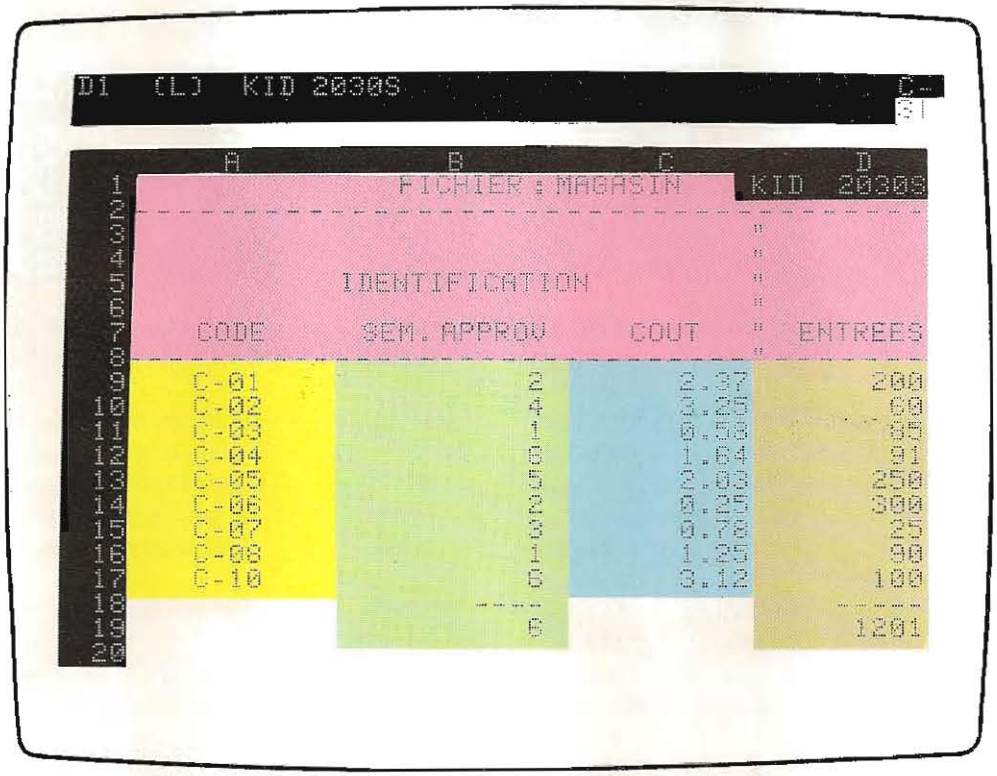
Analyse des variations (WHAT IF)

Cette dénomination correspond à un type particulier d'analyse d'événements à caractère économique ou commercial, dans le but de suivre leur évolution en faisant varier un ou

GESTION D'UN MAGASIN : CODIFICATION DES PRODUITS

Sur cet écran : la partie codification d'un tableau de gestion d'un magasin.





Les lignes 1 à 8 renferment les titres (TITLES).

Chaque article est repéré grâce à un numéro de code ; les codes sont reportés dans les cellules de la colonne A.

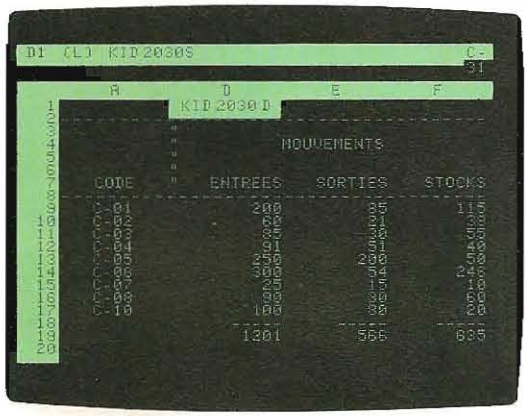
Dans la colonne B sont affichées les périodes d'approvisionnement (en semaine) de chaque article. La cellule B19 contient la fonction @ MAX (B9...B17) qui calcule le maximum des valeurs contenues dans la colonne B. La période d'approvisionnement de 6 semaines est commune aux deux articles C-04 et C-10.

Dans la colonne C, on a entré les coûts des articles.

La colonne D fait partie de la zone suivante du tableau (MOUVEMENTS).

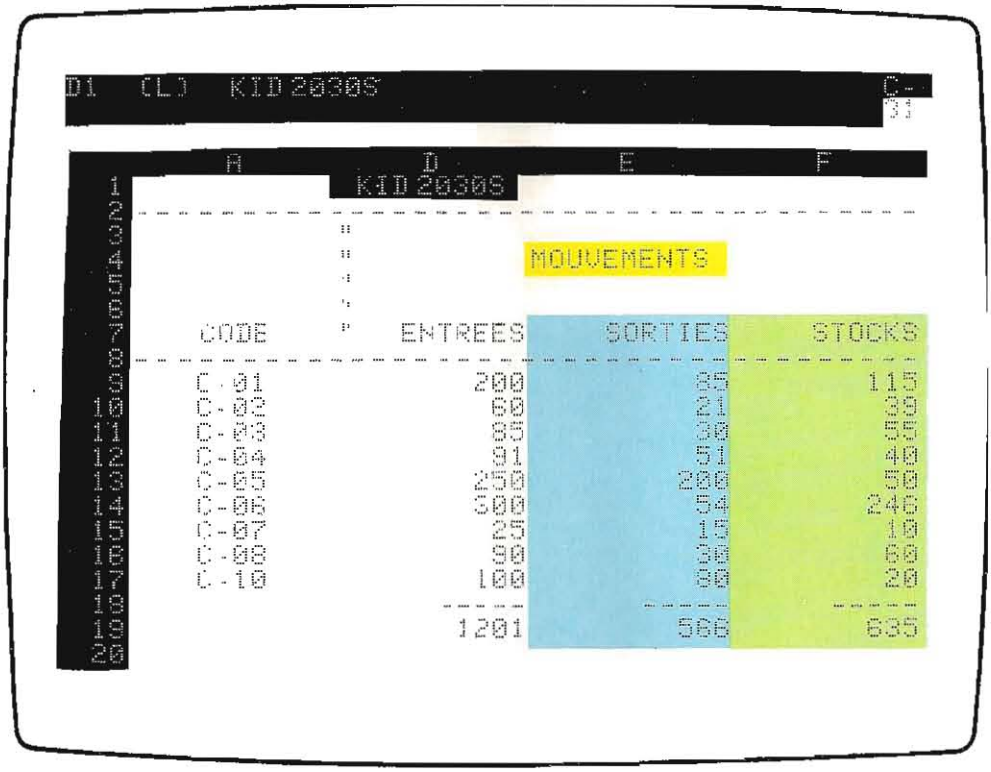
GESTION D'UN MAGASIN : MOUVEMENTS

En actionnant le SCROLL (déplacement) latéral du tableau, on affiche la zone des mouvements.



D1 (L) KID2030S C-
31

	A	D	E	F
1		KID2030D		
2				
3		MOUVEMENTS		
4				
5	CODE	ENTREES	SORTIES	STOCKS
6	C-01	200	85	115
7	C-02	60	21	39
8	C-03	85	30	55
9	C-04	91	51	40
10	C-05	250	200	50
11	C-06	300	54	246
12	C-07	25	15	10
13	C-08	90	30	60
14	C-10	100	80	20
15		1201	566	635



D1 (L) KID2030S C-
31

	A	D	E	F
1		KID2030S		
2				
3		MOUVEMENTS		
4				
5	CODE	ENTREES	SORTIES	STOCKS
6	C-01	200	85	115
7	C-02	60	21	39
8	C-03	85	30	55
9	C-04	91	51	40
10	C-05	250	200	50
11	C-06	300	54	246
12	C-07	25	15	10
13	C-08	90	30	60
14	C-10	100	80	20
15		1201	566	635

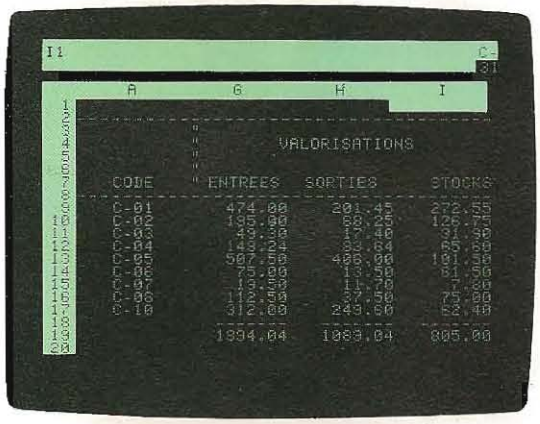
La partie mouvements occupe une zone de tableau placée loin de la colonne A, qui contient les codes articles. Pour afficher les mouvements, il est nécessaire de définir la colonne A (TITRE/TITLE) verticale.

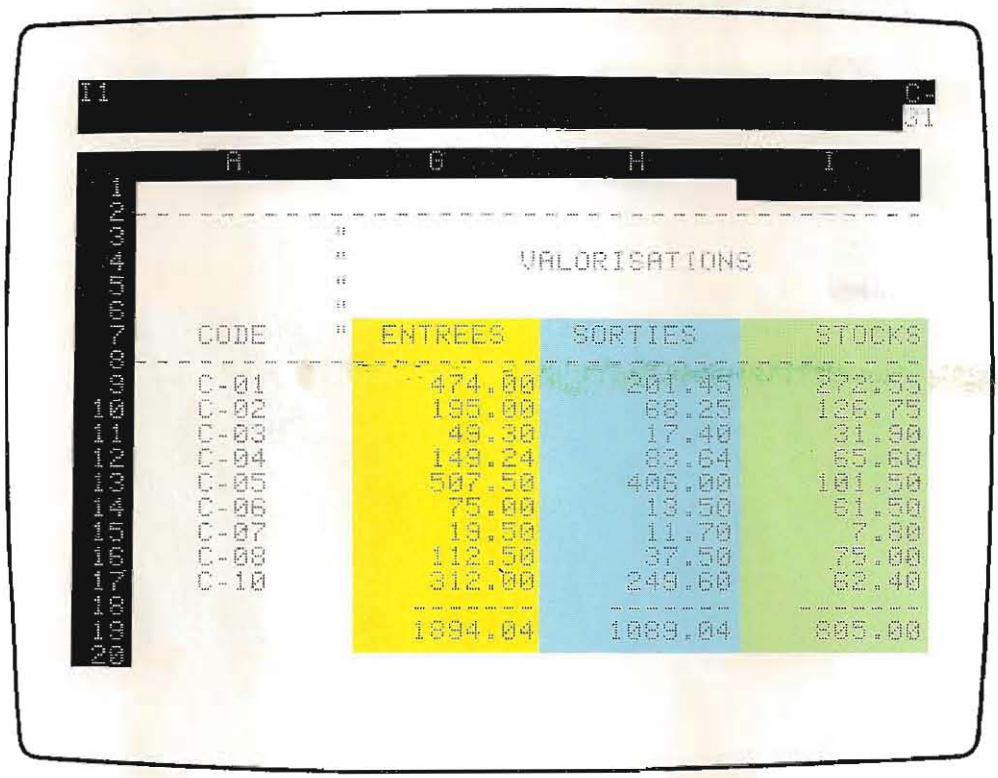
En entrant une donnée relative à un mouvement de sortie (colonne E), on a de nouveau le calcul en E19 du total de la quantité sortie (la cellule E19 renferme la fonction @SUM(E9...E17)). Rappelons qu'on a introduit le calcul par colonnes.

Aussitôt après, on calcule le stock par l'opération de sortie (colonne F). Ainsi, si l'on a sorti un certain nombre d'articles C-01, la cellule F9 est recalculée comme la différence entre l'entrée D9 et la sortie E9.

GESTION D'UN MAGASIN : VALORISATIONS

Dans la zone réservée aux valorisations, pour chaque article, on a affiché les valeurs proportionnelles des entrées/sorties et du stock. Les totaux sont reportés à la ligne 19.





Les valorisations des entrées (colonne G) sont obtenues en calculant le produit de la quantité en entrée d'un article (zone mouvements, colonne D), par le coût unitaire (zone identificatrice, colonne C). Ainsi, la donnée 474.00 qui apparaît en G9 est obtenue à l'aide du calcul : + C9*D9.

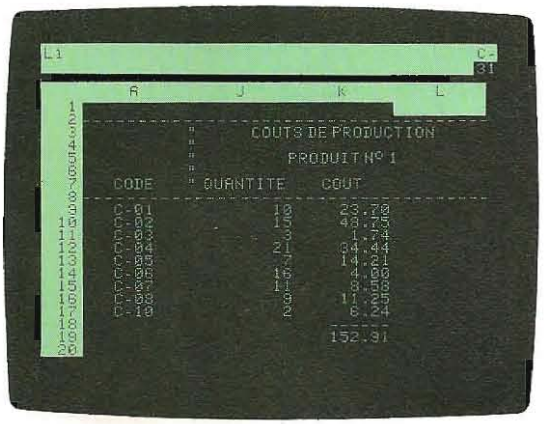
Les valeurs proportionnelles des sorties (colonne H) sont le produit de la quantité en sortie d'un article (mouvements, colonne H) par le coût unitaire (identification, colonne D). La donnée 201.45 en H9 est obtenue par l'expression : +C9*E9.

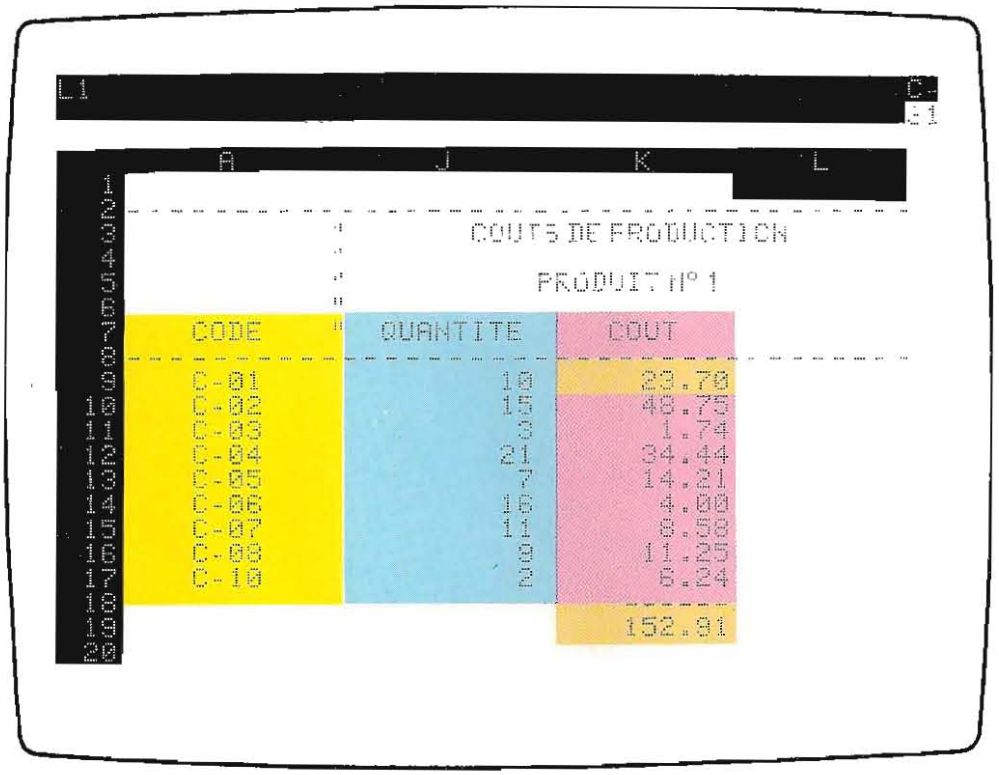
La valorisation du stock s'obtient de façon analogue à l'aide de l'expression : +C9*F9.

En faisant varier une des quantités qui apparaissent dans la zone des mouvements, on recalcule automatiquement toutes les valorisations.

GESTION D'UN MAGASIN : COÛTS DE PRODUCTION

L'écran affiche les coûts de production de produits finis dont les composants sont les articles mentionnées dans la partie descriptive.





La colonne A renferme les codes des articles disponibles et sert uniquement de mémento.

La colonne J est utilisée pour introduire les quantités de tous les composants entrant dans la fabrication du produit fini. Dans l'exemple considéré, le produit n° 1 (cité dans le titre) demande 10 unités du composant C — 02, etc.

La colonne K renferme les expressions permettant le calcul du coût de fabrication du produit, main-d'œuvre non comprise. Une extension du tableur pourrait prévoir aussi cette décomposition en termes de coûts unitaires et de temps requis pour l'assemblage des composants.

La quantité apparaissant en K9 est donnée par l'expression : +C9*J9.

La structure du tableau se prête également à l'analyse des coûts. Ainsi, en changeant de fournisseur de composants on aboutit à des baisses de coûts de certains articles et à des augmentations pour d'autres. Le programme calcule alors immédiatement en K19 les nouveaux coûts. Ce qui permet de décider de conserver, ou non, ce fournisseur.

EXEMPLE D'IMPRESSION DU TABLEUR

FICHIER : MAGASIN KID 20305

DESCRIPTIF			MOUVEMENTS		
CODE	SEM.APPROV	COUT	ENTREES	SORTIES	STOCKS
C-01	2	2.37	200	85	115
C-02	4	3.25	60	21	39
C-03	1	0.50	85	30	55
C-04	0	1.64	91	51	40
C-05	5	2.00	250	200	50
C-06	2	0.25	300	54	246
C-07	3	0.70	25	15	10
C-08	1	1.25	90	30	60
C-10	5	2.12	100	80	20
	6		1201	586	635

VALORISATION			COUTS DE PRODUCTION	
			PRODUIT N° 1	
ENTREES	SORTIES	STOCKS	QUANT.	COUT
474.00	291.45	272.55	10	23.70
135.00	54.25	128.75	15	48.75
41.30	17.40	21.30	3	1.74
149.24	83.64	65.60	21	34.44
507.50	405.00	161.50	7	14.21
75.00	13.50	61.50	16	4.00
19.50	11.70	7.90	11	3.58
112.50	37.50	75.00	9	11.25
312.00	243.00	82.40	2	6.24
1394.04	1038.64	305.00		152,31

plusieurs paramètres.
L'emploi de WHAT IF (« que se passe-t-il si... ») permet précisément de jouer sur les données qui régissent une situation afin d'en observer les effets sur le résultat final.
Dans les cas de simulations plus complexes, on utilise un modèle mathématique du phénomène, qui fait appel à des algorithmes très sophistiqués. Dans les cas les plus courants (prévisions de vente, profits d'investissements, etc.), le tableur suffit amplement.
La méthode la plus simple consiste à afficher un tableau décrivant, pas à pas, l'évolution du phénomène à analyser, ensuite à introduire les nouvelles données.

Pour chaque nouvelle donnée, le tableur effectue automatiquement tous les calculs et affiche le résultat. Ci-dessous : le tableau reproduisant le calcul prévisionnel des bénéfices en fonction du nombre de pièces fabriquées.
Ici, seuls certains chapitres entrant dans l'établissement du compte d'exploitation ont été pris en considération. Il est toujours possible d'en élargir la structure en introduisant de nouvelles lignes (en fonction d'autres facteurs) et de nouvelles colonnes, afin d'analyser une période supérieure à 4 mois. Le résultat (positif ou négatif) est obtenu par la différence entre le montant du chiffre d'affaires et celui des coûts.

SCHÉMA DU TABLEAU POUR LE CALCUL DU RESULTAT

Entrée des données
Calcul des coûts de production
Calcul du chiffre d'affaires
Calcul du résultat

	A	B	C	D	E	F	G
1							
2							
3							
4			JANVIER	FEVRIER	MARS	AVRIL	TOTAL
5							
6		QUANTITE PRODUIT					@SUM (C6. .F6)
7							
8		COUTS					
9							
10		PRODUIT	+ C6 * 0.2 + 50				
11		DISTRIBUTION	+ C6 * 0.1				
12		AGIOS	+ 0.18 * (C10 + C11)				
13							
14		TOTAL DES COUTS	@SUM (C10...C12)				
15							
16							
17		CHIFFRE D'AFFAIRES	+ C6 * 1.02				
18							
19		RESULTAT	+ C17 – C14				
20		MARGE %	+ 100 * (C19/C14)				

Trois types de coûts ont été prévus dans notre exemple :

- **Coût de production** (PRODUIT), égal à une valeur fixe (50) plus un chiffre proportionnel au nombre de pièces produites [50 + (0,2 × nombre de pièces)].
- **Coût de distribution** (DISTRIBUTION), proportionnel à la quantité de produits (0,1 × quantité de produits).
- **Frais financiers** (AGIOS). Ce chapitre tient compte des agios dus aux emprunts effectués pour l'achat de matières premières ou d'autres composants. Il est donc proportionnel aux montants précédents [0,18 × (PRODUIT + DISTRIBUTION)].

Ces formules, citées en exemple, devront être adaptées à chaque situation. Le total des coûts s'obtient en additionnant ces trois postes. Si l'on introduit la quantité de produits prévue pour un mois (ligne 6), le programme effectue les calculs pour toutes les formules et affiche les résultats. La signification de ces chiffres dépend de l'unité de mesure adoptée par l'utilisateur. Ainsi, la quantité ou les coûts peuvent être exprimés en nombre de pièces, en poids ou en volume.

Dans le tableau ci-dessous – copie de la feuille sur l'imprimante – seuls les entiers sont représentés. Les décimaux ne sont pas affichés mais les calculs en tiennent compte. La structure (masque de saisie) ainsi construite peut être mémorisée (ici, on a utilisé un fichier : PROJECTIONS) afin d'être rappelée au gré de l'utilisateur. On applique la technique WHAT IF (Que se passe-t-il si...) en faisant varier une ou plusieurs données d'entrée, (en l'occurrence la quantité de produits) et en analysant les effets sur le total. Si, par exemple, en mars, la production est tombée à 75 produits, on accuse une perte de 11 % sur la somme investie alors que le rendement total du trimestre (colonne TOT) est égal à 12 %. Supposons que l'objectif des 4 premiers mois soit un résultat de 20 %, quelle doit être la production d'avril pour compenser la baisse de mars ? La réponse s'obtient en introduisant des quantités supérieures en avril et en observant les variations du résultat en cellule G20 (marge). On répète le processus jusqu'à l'obtention d'une valeur finale proche de celle souhaitée. Le calcul à effectuer est donc de type itératif. On calcule un résultat à partir d'une estimation et on décide si l'on a ou non recours à une valeur différente. Les itérations

EVOLUTION DU CHIFFRE D'AFFAIRES ET DU RESULTAT EN FONCTION DE LA PRODUCTION

72-81	FICHIER PROJECTIONS				KID
	JANV	FEV	MAR	AVR	TOT
QUANT. PROD	104	96	75	150	425
COUTS					
PRODUCTION	71	69	65	80	135
DISTRIBUTION	10	10	5	15	43
AGIOS	15	14	12	17	33
COUT TOTAL	96	93	82	112	366
CH. AFF.	106	96	77	152	424
RESULTAT	10	5	-9	41	47
MARGE %	11	5	-11	36	12

sont beaucoup plus faciles avec des tableurs qu'avec des programmes en Basic (ou en d'autres langages).
La caractéristique principale du tableur consiste en effet à présenter la structure entière, tout en autorisant l'introduction des données en n'importe quel point. Ainsi, si l'on veut analyser les effets d'éventuelles variations de production en janvier, il suffit de changer le contenu de la cellule C6 pour obtenir un nouveau calcul complet. En Basic, une simulation avec calculs itératifs nécessite de nombreuses instructions.
Faire varier les formules de calcul constitue un autre avantage. Pour analyser les variations du résultat en fonction des frais de distribution engagés, il suffit de modifier le contenu de la cellule C11 (et de celles correspondant aux autres mois).
En Basic, il faudrait modifier le programme, ou tout au moins l'écrire dans une forme totalement paramétrée pour obtenir une structure proche. Dans d'autres langages non interprétés, les structures sont encore plus lourdes.
Il ne faudrait pas en déduire que le tableur est capable de se substituer aux langages de programmation. Pour des problèmes plus complexes ou des applications nécessitant des logiques diversifiées, le tableur s'avère inadapté.
L'exemple cité illustre un cas d'interactivité gérée par l'utilisateur (répétition du calcul et contrôle de validité du résultat dépendent de lui). Deux difficultés se présentent dans les applications plus complexes : trop de variables dans chaque itération, nombre élevé d'itérations nécessaires pour obtenir un résultat acceptable. Dans ce cas, des fonctions particulières facilitant le déroulement des itérations ont été incluses dans certains types de tableurs.

Les fonctions pour le calcul itératif

Dans les formes simples du tableur, le calcul est effectué avec des commandes fournies par l'opérateur. Dans les formes plus évoluées, des fonctions spéciales servent aux calculs itératifs, principalement :

ITERCNT (). Cette fonction restitue la valeur de l'itération en cours, en partant de 1 pour la première. Elle ne comporte pas de paramètres ; le symbole () utilisé en appel a le sens d'un argument fictif. Il sert essentiellement de test en fin de processus d'itération. Ainsi, la condition
ITERCNT () = 10 provoque l'arrêt des calculs à la dixième itération.



J. Piekerell/Marka

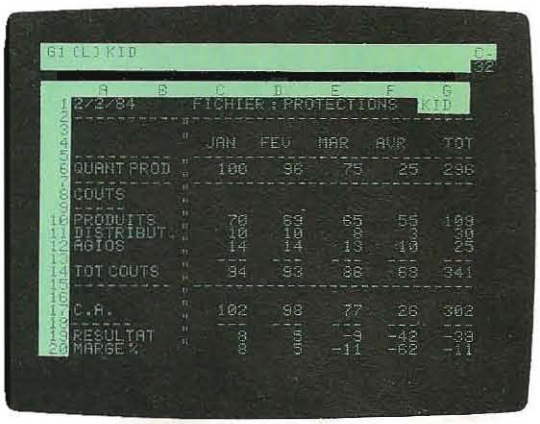
Assemblage de composants électroniques dans une unité de production.

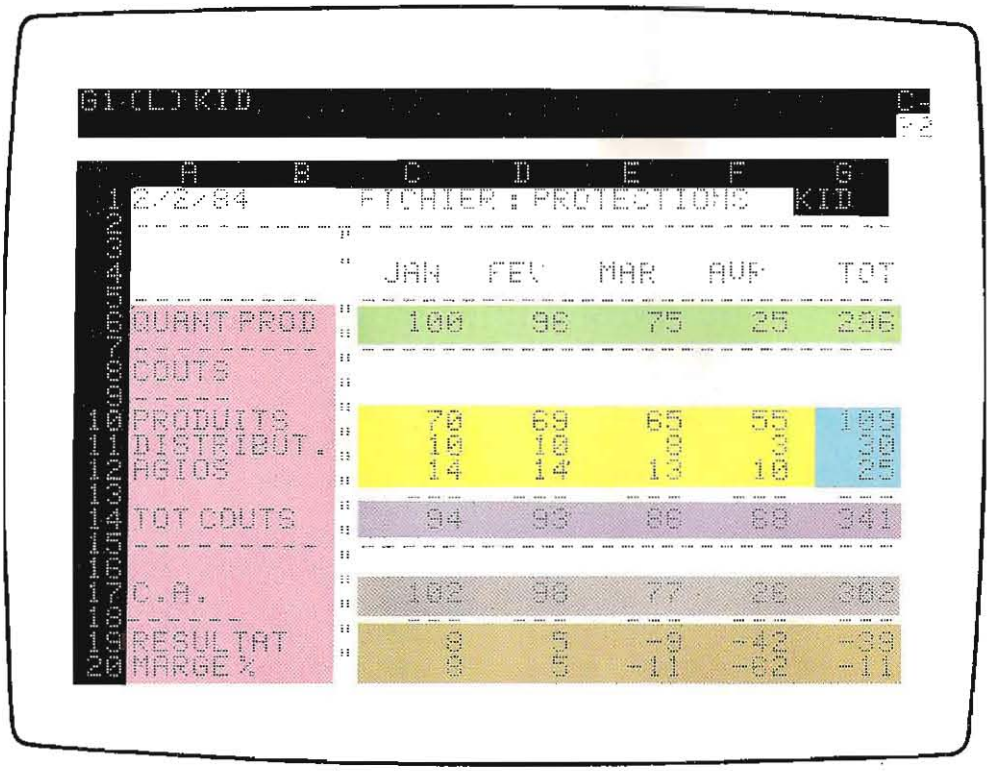
DELTA (). Cette fonction fournit la valeur absolue maximale enregistrée lors des changements de valeurs entre deux itérations successives. On peut donc l'utiliser comme critère de convergence. Ainsi, DELTA () < 0,1 arrête le calcul quand le maximum de variations est inférieur à 0,1.

Ces deux fonctions sont indispensables pour arrêter le calcul sur les feuilles comportant des fonctions itératives. Il existe alors une commande (généralement appelée ITERATION) qui met automatiquement en activité le processus de calcul récursif. Dans d'autres tableurs, deux méthodes sont suivies. Soit on a recours à des artifices particuliers pour simuler la commande ITERATION. Soit c'est l'opérateur qui introduit la nouvelle valeur de la variable (indépendante), d'une itération à l'autre. Le calcul étant relancé par l'utilisateur, il gère lui-même ses interruptions.

ANALYSE DES VARIATIONS METHODE « WHAT IF »

On a illustré ici l'application de la méthode WHAT IF pour l'analyse des variations sur le tableur. L'écran propose une nouvelle structure de feuille. Pour obtenir une vision complète de toutes les colonnes utilisées, le format entier a été imposé (en présentation, et non pas dans les calculs !).





Les descriptions contenues dans la colonne A ont dû être écrites en utilisant les cellules contiguës aux colonnes A et B. Ceci implique l'impossibilité d'ordonner des alignements des contenus des cellules car on risquerait de fragmenter les descriptions.

Les cellules de la ligne 6 sont utilisées pour l'introduction des quantités.

Les cellules appartenant au rectangle C10-F12 contiennent les calculs relatifs aux coûts.

Les cellules 10, 11, 12 de la colonne G contiennent le calcul des totaux des lignes : PRODUIT, DISTRIB., AGIOS utilisant la fonction SUM.

Les cellules de la ligne 14 contiennent les totaux des trois lignes supérieures.

La ligne 17 contient le calcul du chiffre d'affaires.

Les cellules des deux dernières lignes sont destinées au calcul du résultat ainsi que de la marge (%).

La méthode WHAT IF peut être appliquée tout simplement en introduisant une nouvelle donnée dans une des cellules de la ligne 6. Le calcul du tableau sera immédiat.

Evolution du tableur

Les commandes et les fonctions examinées concernent les tableurs ordinaires. Comme pour le Basic, ces programmes ont rapidement abouti aux versions actuelles, plus complètes et enrichies de commandes et de fonctions nouvelles.
A titre d'exemple on a comparé, dans le tableau ci-dessous, les fonctions Visicalc et Multiplan dans leur version Olivetti M20. On constatera que les principales fonctions sont inchangées même si elles utilisent parfois un symbolisme différent.
Vous trouverez page 846 (tableau de gauche) de nouvelles fonctions spécifiques de Multiplan ; dans le tableau de droite, une comparaison des commandes Multiplan-Visicalc. Là aussi, les principales commandes sont communes, avec de nouvelles versions pour Multiplan.
Nous avons retenu :

FONCTIONS MULTIPLAN ET LEURS EQUIVALENTS VISICALC

Multiplan	Visicalc
ABS(N)	@ ABS()
AND(liste)	@ AND(liste)
ATAN(N/SQRT(1-N*N)	@ ASIN(N)
ATAN(N)	@ ATAN(N)
AVERAGE(liste)	@ AVERAGE(liste)
COS(N)	@ COS(N)
COUNT(liste)	@ COUNT(liste)
	@ ERROR
EXP(N)	@ EXP(N)
FALSE()	@ FALSE
IF(L,V1,V2)	@ IF (1,v1,v2)
INDEX(zone,index)	@ CHOOSE(N,liste)
INT(N)	@ INT(N)
ISERROR(N)	@ ISERROR(N)
ISNA(N)	@ ISNA(N)
LN(N)	@ LN(N)
LOG10(N)	@ LOG10(N)
LOOKUP(N,zone)	@ LOOKUP(N,domaine)
MAX(liste)	@ MAX(liste)
MIN(liste)	@ MIN(liste)
NA()	@ NA
NOT(L)	@ NOT(L)
NPV(N,liste)	@ NPV(N,domaine)
OR(liste)	@ OR(liste)
PI()	@ PI
PI()/2-ATAN(N/SQRT(1-N*N))	@ ACOS(N)
SIN(N)	@ SIN(N)
SQRT(N)	@ SQRT(N)
SUM(liste)	@ SUM(liste)
TAN(N)	@ TAN(N)
TRUE()	@ TRUE

FONCTIONS SPECIFIQUES MULTIPLAN

Fonction	Description
COLUMN(N)	Numéro de la colonne.
DOLLAR(N)	Convertit N en texte, le laisse sous forme chiffrée et l'affiche précédé du signe $. Si N est négatif, le chiffre est mis entre parenthèses.
FIXED(N,d)	Convertit N en texte et l'affiche avec choix du nombre de décimales visibles.
LENT(T)	Longueur du texte T en caractères.
MID(T,s,c)	Fournit une chaîne de c caractères du texte T à partir de la position s.
MOD(N1,N2)	Reste de N1/N2.
REPT(T,C)	Texte composé par C répétitions du texte T.
ROUND(N,d)	Valeur de N arrondie à d chiffres décimaux.
ROW()	Numéro de ligne en cours.
SIGN(N)	−1, 0, ou +1 si c'est négatif, nul ou positif.
STDEV(liste)	Déviations standard des valeurs de la liste.
VALUE(T)	Valeur numérique du texte T; T peut présenter un nombre en tout format, y compris un montant en dollars.

COMMANDES MULTIPLAN ET VISICALC

Multiplan	Visicalc
Blank	/B
Transfer Clear	/C
Delete Column, Delete Rows	/D
Edit, Alpha	/E
Format Cells	/F
Format Width	/GC
Format Default	/GF
pas nécessaire	/GO
Option	/GR
Insert Columns, Insert Rows	/I
Move Columns, Move Rows	/M
Print	/P
Copy	/R
Transfer Load	/SL
Quit	/SQ
Transfer Save	/SS
Window Split Titles	/T
Option (N° version, copyright)	/V
Window Open, Window Split	/W
Window Link	/WS, /WU
Goto Row-Col	>
NEXT WINDOW key	;
RECALC key	!
use references	#
REPT	/-
Format Options	
Help	
Lock	
Name	
Sort	
Window	
External	

Avec Multiplan, vous entrez seulement la première lettre du nom de la commande.

Format Options. Sélectionne certaines options du format de visualisation du contenu des cellules.

Help. Présente à l'écran une page d'explications pour l'utilisation du programme.

Window. Permet la gestion des fenêtres à l'écran. Grâce à cette commande, l'écran est divisé en « fenêtres », chacune d'elles pouvant être utilisée comme une feuille indépendante.

External. Permet l'utilisation de données mémorisées ailleurs, par exemple, dans une ou plusieurs feuilles indépendantes stockées sur disques.

Lock. Empêche la modification d'une (ou de plusieurs) cellule : très utile lorsque viennent

s'intercaler des cellules d'introduction de données et des cellules de calcul.
En déplaçant le curseur pour introduire des données, on peut facilement confondre les positions et écrire la valeur là où une expression était mémorisée (d'où perte de l'expression). Si, en revanche, on protège la cellule, l'insertion devient impossible.

Name. Permet de définir plusieurs cellules à l'aide d'un même nom symbolique. C'est une commande qui met en activité une logique semblable à celle mise en service par la fonction DIM du Basic. Dans certains tableurs, l'ensemble des cellules de même nom appartient à un domaine (« range »). On peut exécuter toutes les fonctions s'y rapportant et ayant pour paramètre commun l'appartenance à un domaine au lieu des adresses.

Sort. Assure le tri du contenu d'une colonne. Il peut être alphabétique ou numérique, croissant ou décroissant.
D'autres versions de tableurs (par exemple Lotus 1-2-3) offrent la possibilité de construire des graphiques et de gérer une base de données.
Pour donner une idée de la diversité des tableurs, une application de Multiplan a été décrite à titre d'exemple page 848, dans sa version Olivetti M20.

Les programmes de gestion des fichiers

Une deuxième série de programmes se rapporte à la gestion de fichiers. Les programmes les plus simples prévoient, en général, de n'utiliser qu'un seul fichier ainsi qu'un nombre limité de fonctions. Les plus sophistiqués proposent, en outre, la gestion d'une véritable base de données avec des possibilités de programmation très proches du langage Basic.
Les principales fonctions de ces programmes sont :

- la création des fichiers et la définition des zones,
- l'introduction et la mise à jour des donnée,
- la recherche,
- le classement,
- le traitement des données,
- l'édition d'états.

Création de fichiers et définition de zones

C'est la première fonction à mettre en service pour créer une base de données. Les paramètres à fournir sont :

- le nom du ou des fichiers,
- l'unité disque (A/B ou 0/1 selon le système d'exploitation),
- les noms et attributs des différentes zones.

Généralement, la syntaxe de la commande se traduit ainsi :

CREATE X:NOM

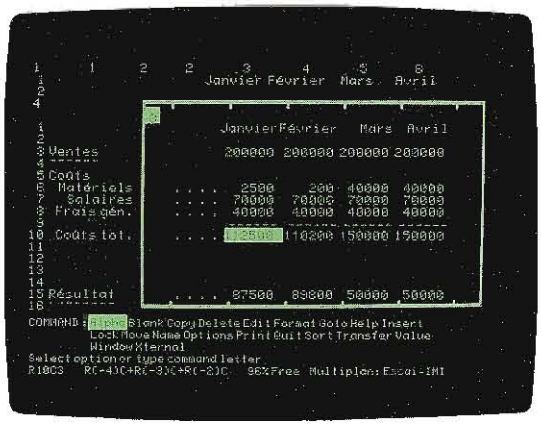
où X est l'unité disque sélectionnée (elle peut être omise, auquel cas le système l'affecte automatiquement par défaut) et NOM le nom du fichier à créer. Après avoir vérifié l'espace disponible sur le disque, le programme sauvegarde le nom dans un dictionnaire et passe à l'introduction des attributs des zones constituant l'enregistrement. Dans ces programmes, les enregistrements appartenant au même fichier ont la même longueur. Exprimée en nombre d'octets, il s'agit de la somme des longueurs des zones constituant l'enregistrement, au fur et à mesure que l'on introduit la description. En général, les données une fois entrées, il n'est guère possible d'en changer le format ou le type de zones sans perdre les données précédemment introduites.
Les paramètres définissant les différentes zones sont :

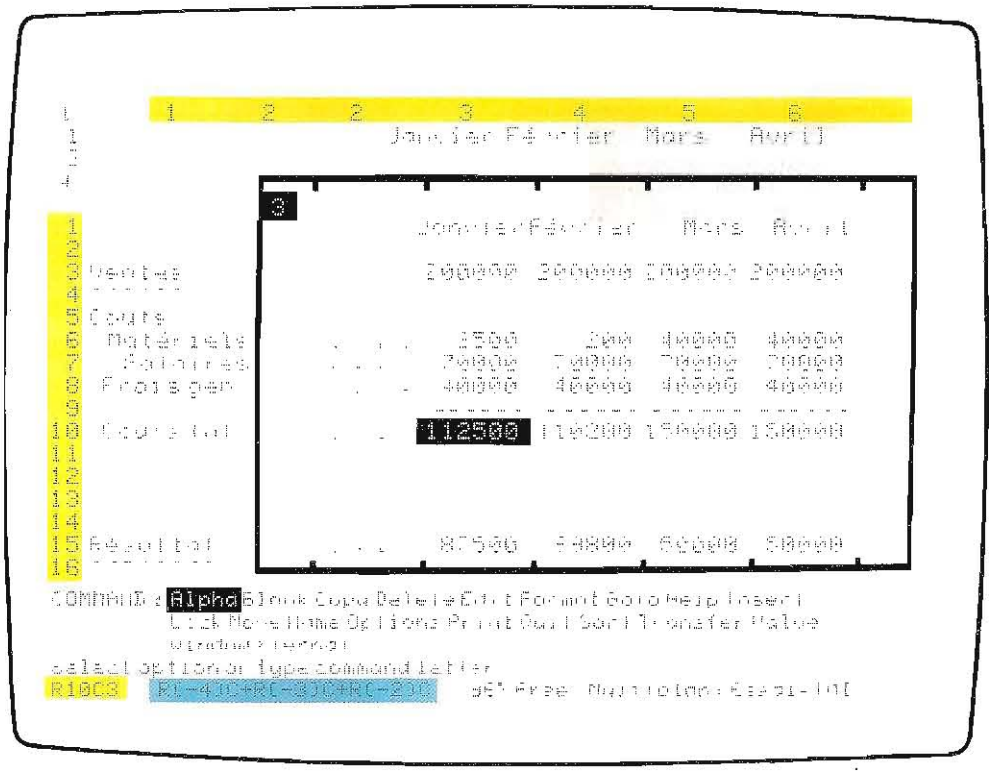
- nom : il s'agit du nom symbolique par lequel l'utilisateur et le programme identifient le contenu de chaque zone ;
- type : définition du type de zone (alphanumérique ou numérique) ;
- longueur : c'est le nombre maximum de caractères prévus dans la zone.

Pour les zones numériques, on demande également le nombre de chiffres décimaux. Il convient de ne pas oublier que la longueur totale tient compte de l'espace réservé aux décimales. Ainsi, une zone numérique avec trois entiers et deux décimales occupera six caractères (XXX.XX). Si l'on spécifie une zone de type numérique, un contrôle (numérique) des données a systématiquement lieu ; ne seront alors acceptées que des valeurs numériques.
Le nombre de zones définies dans un enre-

TABLEUR MULTIPLAN

Par certains aspects, le tableur Multiplan diffère de celui considéré jusqu'ici. Ainsi la présentation des commandes est située en bas tout comme le contenu de la cellule occupée par le curseur.





La différence la plus marquée concerne l'adressage des cellules. Les adresses sont constituées de deux valeurs numériques précédées de la lettre L (L pour ligne = R pour Row en anglais) et C (Colonne). L10C3 est l'adresse de la cellule au croisement de la ligne 10 et de la colonne 3, pointée ici par le curseur.

Dans les formules, les adressages (relatifs) sont exprimés en termes de déplacement de la cellule contenant la formule. Avec le curseur en R10C3, la formule permettant de faire référence à la cellule L6C3 est [−4]C, indiquant toujours son appartenance à la colonne (C) mais positionnée 4 lignes plus haut (−4).

Les déplacements en haut ou à gauche sont exprimés par des numéros négatifs ; les déplacements vers le bas à droite par des numéros positifs.
La formule introduite dans la cellule L10C3 calcule la somme des contenus des cellules L6C3, L7C3 et L8C3.

gistrement varie beaucoup en fonction du type de programme. En général, il est supérieur à 30.

Introduction et mise à jour des données

La structure du fichier une fois définie, on passe à l'introduction des données. Normalement, le programme présente un masque d'écran qui reporte les noms des zones et l'espace disponible pour chacune d'entre elles. Dans des programmes plus évolués, l'utilisateur spécifie certains éléments de présentation du masque : position de chaque zone, le mode de présentation (écran vidéo inversé, souligné).

La principale différence entre les méthodes de présentation adoptées par les programmes est due au nombre maximum de colonnes gérées par le masque écran. Certains programmes présentent toutes leurs zones sur une seule colonne (une pour chaque ligne) dans l'ordre où elles ont été définies. Dans d'autres, il est possible de positionner différentes zones sur la même ligne (on aura alors un masque avec plusieurs colonnes), choisissant n'importe quel ordre de présentation.

La phase d'introduction des données présente deux aspects selon qu'on opère sur un fichier existant ou en cours de création. Dans le premier cas, les nouvelles données sont tenues pour ajoutées et la commande mettant en service le transfert est normalement APPEND. Dans le deuxième cas, c'est le programme lui-même qui, au terme de la création, revient à la phase d'introduction. En dehors de leur formalisme, les deux méthodes appliquent la même fonction : entrer les données dans le format défini au cours de l'organisation du fichier, en associant à chaque nouvelle donnée un numéro d'enregistrement incrémenté. L'insertion constitue une troisième manière d'introduire les données. Habituellement, le mot-clé mettant en service cette fonction est INSERT.

En phase d'écriture, en création comme en APPEND, les données sont sauvegardées l'une après l'autre, dans leur ordre d'arrivée, alors que certaines applications – par exemple pour accélérer les phases de recherche – admettent l'insertion dans n'importe quelle position intermédiaire (insertion adressée).

La commande d'insertion adressée possède une syntaxe qui dépend du logiciel utilisé. La majeure partie des programmes utilise deux méthodes : le **pointage** et l'**adressage direct.**

Dans le premier cas, chaque donnée introduite comporte sa propre adresse (numéro d'enregistrement). Dans de nombreux programmes, il est possible de se positionner sur un enregistrement précis à l'intérieur du fichier. Une commande ultérieure d'insertion y provoquera l'écriture de la nouvelle donnée. Les autres données seront automatiquement transférées dans l'enregistrement suivant. Dans certaines bases de données, on se positionne sur un enregistrement à l'aide d'une commande du type GOTO n, où n est le numéro de l'enregistrement. Ainsi, la séquence

```
GOTO 3
INSERT BEFORE
```

positionne le pointeur sur l'enregistrement 3 et insère « avant » (BEFORE) la nouvelle donnée. Cette dernière sera positionnée entre les anciennes données 2 et 3 et occupera l'enregistrement 3. Quant à la donnée qui occupait précédemment l'enregistrement 3, elle est déplacée en 4 et ainsi de suite jusqu'à la fin du fichier. On peut corriger les données existantes grâce à ce même système de pointage. La commande EDITn est très usitée dans ces programmes. Elle apporte des corrections au contenu de l'enregistrement spécifié. Cette méthode de pointage des données, s'appuyant sur le numéro d'enregistrement, n'est indispensable que dans certains cas particuliers. Elle n'est pas très pratique dans la mesure où elle oblige à recourir à un état de référence mettant en évidence le numéro d'enregistrement de chaque donnée. D'autre part, on peut commettre des erreurs en utilisant une référence juste après une opération d'insertion ou d'ajout. Aussi est-il préférable d'exploiter les possibilités de recherche offertes par ces programmes. Cette recherche du contenu d'une zone déterminée s'appelle **FIND** (trouver), terme normalement utilisé pour désigner la commande.

Recherche (FIND)

Cette fonction sélectionne les données d'après la valeur contenue dans une ou plusieurs zones. Elle fait partie des principales fonctions de gestion de fichiers : recherche, visualisation des données, fusion de fichiers.

Les programmes appartenant à cette catégorie concernent tous l'automatisation du travail de bureau, essentiellement la collecte des informations en provenance d'autres bureaux et des unités périphériques. Après traitement, on obtient des rapports synthétiques à partir desquels seront prises certaines décisions stratégiques pour l'entreprise. Ce processus se réalise par étapes successives de croisement. Les logiques de sélection sont diverses ; cependant, elles peuvent être ramenées à une sélection utilisant les valeurs de certaines zones. En d'autres termes, pour ce qui est des fichiers, c'est la recherche qui est la fonction primordiale. On peut la mener selon trois méthodes liées au type d'organisation des données.

- Si le fichier ne comporte pas de clé d'accès et n'a subi aucun tri, la recherche se fait en lisant tous les renseignements et en comparant le contenu de la zone sélectionnée avec la valeur (ou limite) voulue.
- Si le fichier est trié, on peut adopter la technique de rupture du code. La recherche est alors menée comme précédemment, mais elle prend fin avant la lecture de l'ensemble du fichier, après la rupture du code de contrôle.
- Si le fichier possède un ou plusieurs index, la méthode est plus rapide. Au cours de la création du fichier, on peut établir un index autorisant la sélection sur une zone déterminée (zone clé). Dans ce cas, le numéro des clés d'accès aux enregistrements fournit la limite.

Plusieurs clés peuvent également être prévues, donc de multiples possibilités d'accès avec toutefois quelques inconvénients. Il faut, en effet, créer un nombre de fichiers index égal à celui des clés utilisées. Ce qui signifie un gaspillage de place mémoire et de temps : chaque donnée introduite nécessite une mise à jour de tous les index. Dans les grands systèmes, la capacité mémoire ne constitue pas un véritable problème et le temps de mise à jour des index est négligeable. Mais en micro-informatique, les besoins d'économie sont réels.
Autre inconvénient : le peu de souplesse d'une telle organisation. En envisageant la création d'une série d'index, on désire généralement couvrir les principales situations, mais on ne peut toutes les prévoir. Les applications évoluant, de nouvelles nécessités peuvent surgir, impliquant l'utilisation d'un type de fusion qui ne figure pas dans la série d'index établie. C'est ce qui a amené à concevoir une autre logique d'exploitation des index.
Au départ, aucun index n'est créé. On considère les données comme ne comportant pas de clé. Ce n'est qu'au moment de la fusion ou d'une quelconque opération de sélection que l'on écrit un index de la ou des zones utiles. Cette méthode est commune à presque tous les programmes de gestion de fichiers. L'enchaînement des fonctions est défini de la manière suivante :

- création du fichier,
- introduction des données,
- définition de la clé de sélection (zone clé),
- création de l'index d'après le contenu de la zone clé.

Au terme de ces opérations, on dispose d'un fichier affectant des index aux données d'après chaque zone. En utilisant l'index, on peut mener toutes les opérations de recherche ou de fusion.
La syntaxe utilisée pour la création du fichier se présente ainsi :

INDEX ON « CHAMP » TO « FICHIER »

où
CHAMP est le nom symbolique de la zone à indexer,
FICHIER le nom du fichier index. Pour opérer une sélection ou une recherche, il faut donner au système le nom de l'index (il peut y en avoir plusieurs). La syntaxe est alors :

USE « NOM » INDEX « CHAMP »

qui signifie : utiliser (USE) le fichier (index) NOM indexé sur le contenu de CHAMP. A l'aide de cette commande, on ordonne au système d'appeler le fichier index NOM et d'exécuter des opérations de recherche (définies par la commande suivante) sur la zone portant le nom symbolique de CHAMP.
L'instruction suivante fournira la valeur de comparaison. La syntaxe a la forme :

FIND XXXX

où XXXX est la donnée (clé) à rechercher. Ainsi, la séquence :

1 - INDEX OU NOM TO ANAGR
2 - USE ANAGR INDEX NOM
3 - FIND Michel Martin

exécute les fonctions suivantes :

1 - Création d'un fichier index ANAGR portant sur le contenu de la zone NOM définie au moment de la création du fichier «données».
2 - Utilisation (USE) du fichier ANAGR comme index (INDEX) de la zone NOM.
3 - Recherche de tous les enregistrements contenant, dans la zone NOM, la donnée Michel Martin.

Les commandes sont décrites dans la forme utilisée par un programme particulier. Dans d'autres logiciels, leur syntaxe peut être différente mais les fonctions accomplies sont généralement identiques.

Tri (SORT)

Tous les programmes de gestion de fichiers disposent de la fonction SORT (tri). La méthode communément suivie consiste à créer un fichier complémentaire dans lequel données ou adresses sont réécrites dans l'ordre désiré.
Aucune règle n'oblige à en connaître le contenu. Si le programme gère la fonction SORT, ce sera toujours lui qui, au cours des éditions, obtiendra la donnée grâce aux adresses des enregistrements, en la prélevant du fichier. L'utilisateur n'a qu'à lancer cette fonction en spécifiant la zone du classement et le nom du fichier qui contiendra les données classées (ou leurs adresses). Une des formes typiques d'exécution d'un classement est traduite par :

SORT ON nnn TO mm

La fonction est activée par la commande SORT ; l'expression ON nnn indique la zone dans laquelle on souhaite effectuer le classement (nnn est le symbole d'une des zones définies dans la base de données en service) et l'expression TO mm indique le nom du fichier en sortie. Ainsi, la phrase :

SORT ON Adresse TO Fichier A

active un classement sur la zone Adresse et mémorise la sortie dans un fichier A. Si on utilise ce fichier pour une impression, on obtient les données du contenu de la zone Adresse en ordre croissant. Le fichier données n'est pas modifié et garde son ordre d'introduction. L'utilisation d'un fichier différent pour la sortie du tri est nécessaire si on ne veut pas changer le classement d'origine du fichier données. Faute de quoi, les possibilités d'erreurs seraient considérables, à cause des différents positionnements des données réalisées au cours du tri.

Traitement des données

L'archivage des données donne généralement lieu à l'élaboration du contenu des différentes zones. Dans de nombreux programmes de gestion, des fonctions de calcul sont prévues pour répondre à cette nécessité. D'une manière générale, ces fonctions obéissent aux mêmes règles et à la même syntaxe qu'en Basic. Cependant, elles sont mises en œuvre par des commandes spécifiques liées au logiciel employé. La commande REPLACE est très courante. Elle permet de remplacer le contenu d'une zone par un autre, ou par le résultat d'un calcul.
Ainsi, la phrase :

REPLACE ALL Total WITH Quantité * Coût

remplace, dans tout le fichier (REPLACE ALL), le contenu de la zone Total par (WITH) le résultat du produit des deux autres zones (Quantité * Coût). Quantité et Coût peuvent se référer à des zones du même enregistrement ou indiquer des variables quelconques externes (par rapport à la base de données). Dans certaines bases, la commande REPLACE est utilisée sous condition. Ainsi, l'opération précédente s'écrirait :
REPLACE ALL Total WITH Quantité * Coût FOR Total = 0.

La partie FOR Total = 0 conditionne la substitution du contenu de toute la zone. Si cette zone n'a pas encore été calculée (son contenu est 0) l'opération a lieu ; dans le cas contraire, l'enregistrement n'a pas été fait et le programme passe au suivant.
Pour les programmes plus complexes, des instructions très proches du Basic ont été prévues. Elles permettent d'écrire de véritables routines de calcul ou de créer le contenu des enregistrements. L'instruction, donnée dans le dernier exemple, est une forme de pro-

La bureautique (3)

Définir un système d'informatisation du travail de bureau nécessite un examen attentif d'un « poste de travail » traditionnel : le bureau en tant que meuble. On note les différentes **configurations** *possibles, les* **outils** *habituellement disponibles, les* **fonctions** *impliquées, les* **relations** *entre les différentes activités et les exigences qu'impose toute opération d'automatisation.*

Un poste de travail devra effectuer électroniquement (avec un minimum d'opérations sur papier et d'intervention du personnel de bureau) les mêmes opérations (ou leur équivalent) que celles qui, auparavant, étaient réalisées à la main, oralement ou à l'aide de machines sans mémoire électronique (création et/ou mise à jour et/ou transformation de documents ou de fichiers, communication, archivage, consultation, reproduction d'informations écrites ou orales).

Certaines activités sont pratiquées sur place, au poste de travail, d'autres nécessitent des interconnexions avec d'autres secteurs de l'entreprise ou avec l'extérieur. Ci-contre, on a reproduit un schéma possible de cette situation complexe. On y voit les différents sous-systèmes qui peuvent, aujourd'hui, constituer un bureau. Les composants de ces sous-systèmes *concernent :*

- les communications internes
  gestion de la correspondance et des communications internes ; communications informelles ;
- les archives de bureau
  archives de la documentation de bureau accessibles à plusieurs personnes : catalogues, carnets d'adresses, annuaires, archives personnelles ;
- les procédures de bureau
  soit l'ensemble des normes (formalisées ou non) qui règlent les différentes activités et les circuits d'information.

Dans cette figure, le sous-système traitement de données *(data processing) a été mis en évidence comme entité à part entière pour indiquer l'ensemble des activités et le réseau d'informations qui, dans le cadre d'un bureau traditionnel, sont actuellement gérés par les sites informatiques.*

Dans le modèle examiné, ces tâches sont exécutées selon des règles très précises. Par environnement extérieur, *on entend le* **système global** *dont fait partie le bureau : l'entreprise, les procédures d'organisation, les dispositions législatives, la correspondance avec les organismes extérieurs, les archives publiques d'informations.*

Par conséquent, il devient possible de définir une architecture générale des relations (voir figure de la page 855) dans le cadre d'entreprises équipées d'ordinateurs pour la gestion de postes de travail reliés entre eux. Soulignons ici que l'on pense à un ordinateur déjà utilisé pour des procédures de traitement de données.

On peut imaginer que tous les sous-systèmes (traitement des données, bureau, station de travail, communication et terminal d'édition) communiquent les uns avec les autres. L'interface avec l'utilisateur du terminal doit satisfaire à des exigences de simplicité et de souplesse telles que :

– dialogue guidé (menus et messages autodocumentés, détection et correction des erreurs, fonctions d'aide...) ;

– très grande souplesse dans le dialogue et dans l'accès à l'information (l'opérateur doit être en mesure de changer à tout moment le contexte opérationnel) ;

– langage de commandes réduit au minimum (utilisation des touches de fonction, du curseur pour faire des sélections...) ;

– possibilité de spécialiser (dédier) le système (sous-ensembles en groupes de fonctions « autonomes », formation de base réduite au minimum, utilisation de concepts et d'une nomenclature usuels dans le cadre du bureau...) ;

– possibilité de personnaliser l'interfaçage en fonction de l'utilisateur ;

– « robustesse » de l'interfaçage.

Sous des formes différentes, tous les « objets » manipulés au bureau touchent, d'une façon ou d'une autre, à l' **information écrite.** *Seules leur structure et leurs modalités d'utilisation les différencient.*

On continue donc à recourir aux mêmes instruments pour le traitement des informations, en préservant la cohérence des critères (sauvegarde et consultation). La structure de ces objets peut encore prendre d'autres formes dans la mesure où les règles de composition

ne sont pas rigides : des notes deviennent une lettre, des feuillets peuvent être agrafés et se transformer en document, des rapports et les listes peuvent constituer une partie d'un document. En fonction de leur structure et de leur utilisation, on a identifié cinq classes de textes.

■ Documents

Il s'agit de textes non structurés, accompagnés d'un certain nombre d'« attributs » qui en ***facilitent l'identification,*** *le classement, l'archivage et l'accès aux informations qu'ils contiennent : nom du document, classe, caractère plus ou moins confidentiel, version, auteur, titre des commentaires, annotations, mots-clés, index. Et aussi, informations sur les dimensions et sur le format (nombre de pages, de lignes par page...).*

Tous les paramètres ne sont pas obligatoires. Certains, comme l'identification par exemple, sont même automatiquement affectés par le système.

■ Lettres

Ce sont des textes avec des attributs spécifiques : l'expéditeur remplace l'auteur, l'objet le titre. On note la liste des destinataires (certains pouvant figurer « pour information »), la référence « Nos/Vos références », le degré de confidentialité, les pièces jointes (PJ), la date d'expédition.

L'opérateur doit obligatoirement préciser les parties expéditeur/destinataire(s). Le système indique automatiquement la date et l'heure d'expédition.

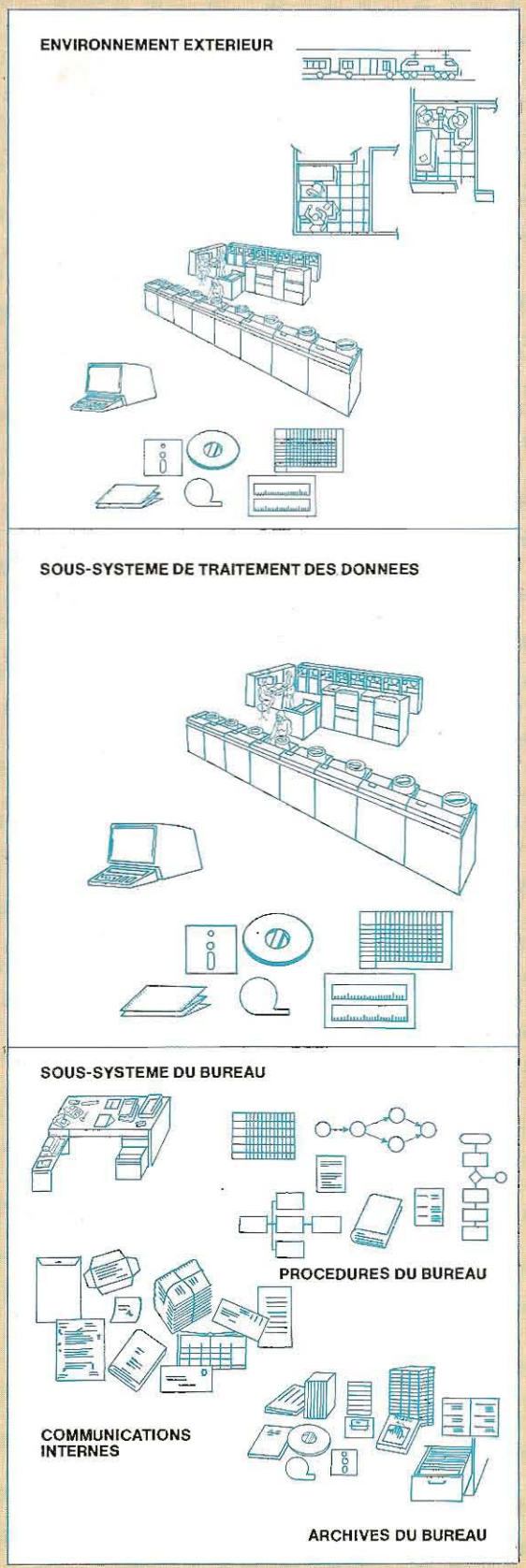
On remarquera que ces lettres sont des objets « normalisés » se rattachant à la correspondance formelle d'un bureau traditionnel. Elles se différencient, en cela, des « messages ».

■ Messages

Généralement, ce sont des objets de « consommation » correspondant à des communications brèves et informelles. Normalement, ils sont détruits par le système après lecture du destinataire qui peut les archiver.

Leurs attributs sont réduits au minimum : De ... à ..., date.

Chaque poste de travail peut se trouver dans une situation complexe et en rapport avec trois sous-systèmes : le bureau, le centre de traitement des données, l'environnement extérieur.



■ Liste

Il s'agit d'objets constitués de séquences d'enregistrement d'une structure identique. On peut les utiliser pour échanger des données à partir et vers des archives Traitement des données en vue de la construction de listes de distribution, de la ***personnalisation*** *de lettres ou de documents contenant des zones variables (les « documents type »).*

■ Formulaires

Plus complexes que les listes, ils correspondent à la gamme habituelle des formulaires de bureau. Ils sont constitués par des zones portant un nom et un type (une zone déterminée peut contenir un type spécifique de données). Malgré la variété de leur format, ils sont dotés d'un nombre minimum d'attributs fixes. Généralement, un objet est créé, archivé, mis à jour, assemblé avec d'autres objets, expédié, copié, annoté.

Pour effectuer de telles opérations sur un objet, il doit être disponible seul ou parmi d'autres objets. Soulignons ici la nécessité de « composer » des objets de nature différente ou de les transformer (en changeant leurs attributs).

Tous les objets sont archivés. Ces ***archives*** *sont* ***personnelles ou collectives, locales ou centralisées.*** *Elles peuvent aussi contenir des objets de différents types. Chaque utilisateur est relié à deux « boîtes aux lettres » (l'une pour l'entrée, l'autre pour la sortie). Elles reçoivent le courrier au « départ » et à l'« arrivée ». On entend par courrier tous les objets expédiés aux différents utilisateurs.*

Afin d'assurer un maximum de souplesse dans l'organisation, une liste d'accès est associée à chaque boîte aux lettres ou aux archives. Cette liste spécifie les utilisateurs autorisés.

Il existe plusieurs niveaux d'accès : lecture, lecture et écriture, lecture et possibilité de copier sans toutefois modifier les originaux. Logiquement, on peut représenter le système comme un réseau de nœuds. A chaque nœud est associée une paire de boîtes et un ensemble de ramifications, avec les listes d'accès. Les fonctions offertes par le système se subdivisent ainsi :

1. CREATION/REVISION D'OBJETS

Elles ont lieu sur un objet « courant », hors archives, sans toutefois modifier l'état des archives. Pour réviser un objet archivé, il faut le « saisir ». Automatiquement et de manière « transparente » pour l'utilisateur, le système crée une copie de travail. Au terme de la révision, l'utilisateur pourra demander l'archivage de la version définitive.

Toutes les fonctions de création/révision fournies par un programme de traitement de textes sont disponibles :

– fonctions de « cut and paste » : assemblage d'objets à partir d'autres objets, mémorisation de « coupures » à réemployer par la suite, mémorisation de documents-type ;

– insertion de notes hors texte ;

– construction automatique d'index, avec accès direct aux « paragraphes » du document.

Dans le cas d'objets formatés (listes et formulaires) :

– fonctions de définition du format (utilisation aisée grâce au dialogue guidé permettant à l'utilisateur de définir l'image de la liste ou du formulaire) ;

– fonctions de compilation, avec entrées guidées par le format (tabulations verticale/horizontale, validation des types de données, compilation automatique des zones calculées, totaux, pourcentages) ;

– sélection sur listes : création d'une sous-liste de toutes les valeurs jouissant d'une certaine propriété dans une autre liste ;

*– fonctions de fusion (***construction de documents****/lettres avec insertion de données présentes dans une liste) ;*

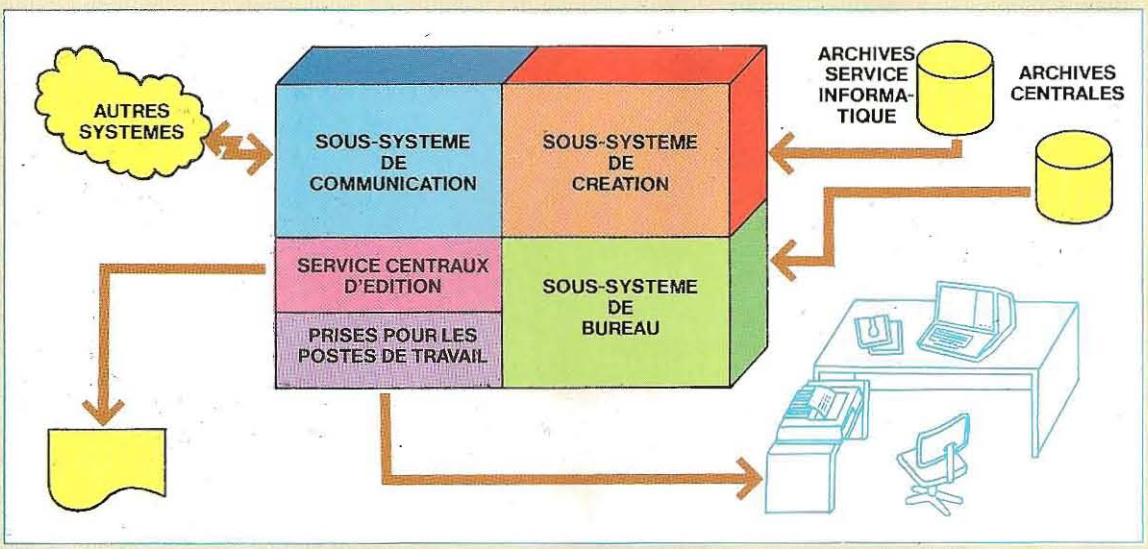
– acquisition/transfert de données à partir ou vers des archives.

2. IMPRESSION

En plus des fonctions normales de mise en page pour la sortie sur imprimante locale ou éloignée, l'impression-fusion regroupe – directement en phase d'impression et sans conserver les copies des résultats de la fusion – des documents qui contiennent des parties variables et des listes (avec possibilité de sélection des valeurs remplissant des conditions spécifiques).

3. RECHERCHE DANS LES ARCHIVES

Ce groupe de fonctions permet la recherche d'objets dans une ou plusieurs archives. L'utilisateur sélectionne des groupes d'objets selon ***trois niveaux de tri.***



Structure du système pour la gestion des activités de bureau.

1) sélection dans le cadre d'une archive spécifique, ou de toutes les archives ;
2) sélection sur le type des objets : lettres, documents ;
3) sélection fondée sur :
- *les mots-clés spécifiés en phase d'introduction de l'objet,*
- *les valeurs spécifiées des attributs (titre, auteur...).*

Ces trois niveaux de tri peuvent être utilisés simultanément.
La fonction de sélection visualise la liste (sommaire) de tous les objets sélectionnés. Avec une telle liste, l'utilisateur sélectionne un seul objet ou examine, de façon séquentielle, les différents objets en aval ou en amont. L'objet sélectionné est visualisé avec tous ses attributs, dans un format analogue à celui des documents ou des lettres traditionnelles.
Toutes les opérations de l'éditeur, qui ne modifient en aucune manière le texte préexistant, sont possibles : déplacements horizontaux et verticaux, recherche de passages du texte ou d'un paragraphe particulier, extraction de coupures de texte pour une utilisation ultérieure.
Les opérations suivantes sont également possibles. Par exemple, avec « grasp », l'objet est considéré comme « objet courant », du fait d'une série de révisions : impression, transfert ou copie dans une autre archive, effacement, annotation, envoi à la copie.
Par ailleurs, bien qu'il ne soit pas possible de modifier des objets à partir de l'examen d'une archive (il faudrait pour cela faire une copie dans le cadre de travail), il est toujours possible d'y ajouter ***des annotations*** *qui se distingueront du texte original.*
Les objets listés peuvent être manipulés collectivement à l'aide de plusieurs opérations : effacement/transfert ou copie dans une autre archive, édition de tous les éléments listés, impression de la liste.
Ultérieurement, on peut pratiquer des sous-sélections sur la liste au moment d'opérer sur des objets strictement définis.

4. GESTION DU COURRIER AU DEPART
Pour pouvoir être expédié par l'intermédiaire du service « courrier électronique », chaque lettre doit être déposée dans la boîte à lettre sortie. Les opérations, en général réalisables pour des archives, y sont permises ainsi que celles qui suivent :

Signature — *Le propriétaire de la boîte (et personne d'autre) appose sa propre signature sur la lettre correctement visualisée, à l'aide d'une commande. Cela devient un attribut de la lettre qui sera affiché avec cette dernière.*

Expédition — *La lettre terminée, on l'expédie grâce à une simple commande.*

Les objets autre que des lettres peuvent être expédiés en pièces jointes.

5. GESTION DU COURRIER A L'ARRIVEE

Les lettres à l'arrivée sont déposées par le système dans la boîte arrivée. A la demande de l'utilisateur, on peut voir son contenu : lettres et éventuellement pièces jointes. Remarquons qu'une boîte peut être examinée par toute personne appartenant à la liste d'accès et non pas exclusivement par son propriétaire.

Cependant, lorsque l'objet et le texte d'une lettre ont été déclarés **confidentiels** par l'expéditeur, ils deviennent alors visibles uniquement par le **destinataire habilité.**

Les opérations autorisées sur cette boîte sont celles que l'on peut pratiquer sur des archives en général.

Les lettres reçues ne peuvent être modifiées. Une commande prévue à cet effet provoque la construction automatique de la lettre visualisée et son envoi immédiat à l'expéditeur, ainsi que celui d'un accusé de réception, sans que l'utilisateur doive en spécifier le texte ou les attributs. Après lecture, l'utilisateur peut conserver la lettre en spécifiant dans quelle archive. Sinon, la lettre n'est pas mise en archives ; elle est visualisée une nouvelle fois à l'ouverture suivante de la boîte.

6. AGENDA PERSONNEL

L'utilisateur dispose d'un agenda facile à consulter en spécifiant le jour (aujourd'hui, demain, une date précise) ou en sélectionnant toute une semaine. Il est possible de programmer le système afin qu'il rappelle à l'utilisateur certains messages contenus dans l'agenda aux dates opportunes.

7. DEFINITION DES PROCEDURES

Le système que l'on vient de décrire a été conçu selon le critère de la plus grande **souplesse opérationnelle** : dans son ensemble, il n'impose, à l'utilisateur, aucune procédure de travail préétablie. Il doit pouvoir s'insérer dans les organisations les plus diverses, sans pour autant en perturber le fonctionnement.

Cependant, chaque entité disposera, en général, d'un ensemble de règles opérationnelles et de procédures plus ou moins formalisées.

Aussi un système avancé d'automatisation des activités de bureau devra-t-il fournir un effort d'adaptation. En d'autres termes, il faudra que l'utilisateur spécifique ait la possibilité de personnaliser le système en fonction des différentes procédures en usage. Par exemple, il devra être facile de communiquer au système que, dans un contexte d'organisation donné :

- « l'expédition d'une lettre se fait uniquement si elle est signée par l'expéditeur »,
- « toute lettre arrivée avec certains attributs devra être conservée dans une archive spécifique et envoyée, avec une annotation spécifique, à un destinataire déterminé »,
- « toute variation sur la liste d'accès à certaines archives devra être communiquée par une lettre standard à tous les utilisateurs intéressés ».

Ceci est aisé si l'on permet à l'utilisateur de définir des « **macrocommandes** » (par ex. : ENVOI, TRI...) en termes de :

- commandes élémentaires (toutes les commandes traitées par le système et résumées ci-dessous),
- conditions rendant possibles l'exécution de ces commandes (ex. : « signature présente », « attribut x = yyyy »).

8. DEFINITION DES ARCHIVES

Ce groupe de fonctions comporte une gestion normale des archives (création - effacement - duplication) et des listes d'accès correspondantes.

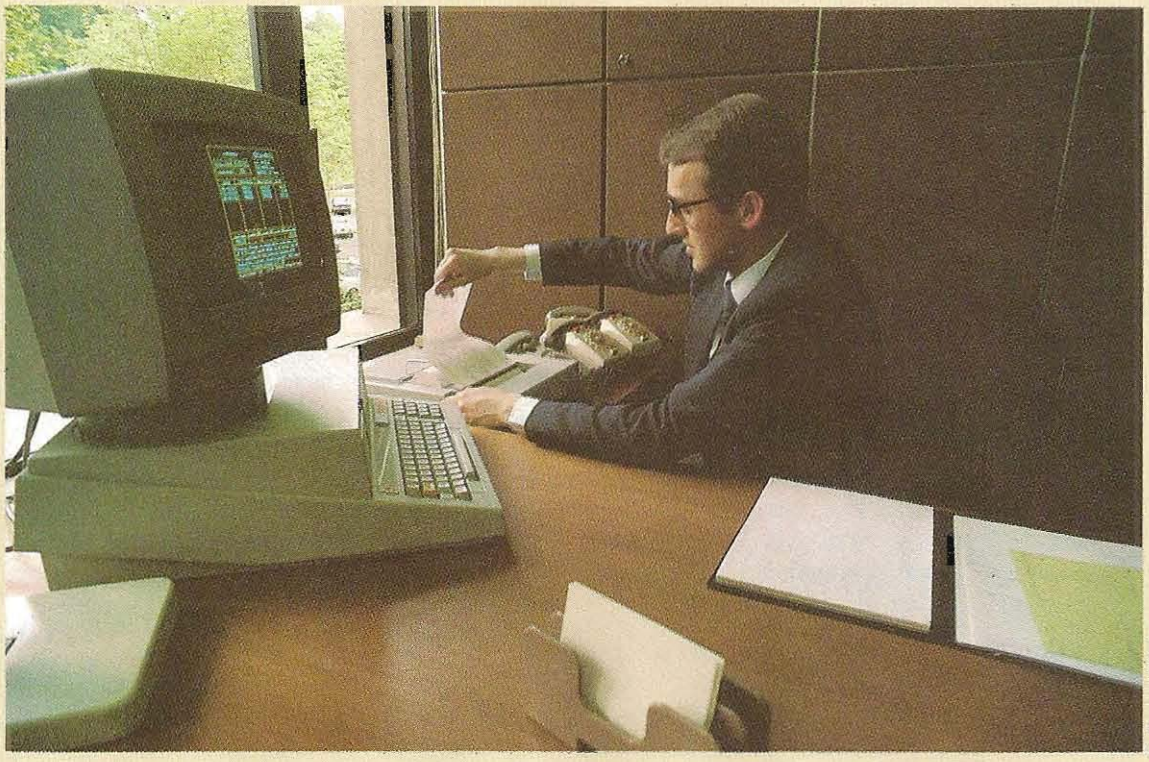
9. ADMINISTRATION DU SYSTEME

Ce noyau de fonctions ne s'adresse qu'à un utilisateur spécifique ayant pour tâche de gérer le système :

- création et gestion de tous les utilisateurs et de leurs profils, autorisation donnée au personnel pour bénéficier des différentes classes de fonctions du système,
- création et gestion des archives centrales et des archives de sauvegarde,
- définition des procédures à un niveau global.

Cette subdivision en classes a été pratiquée sur la base de stricts critères d'homogénéité. Chaque classe a un sous-menu de fonctions élémentaires à choisir à partir d'un menu principal du sous-système de gestion des activités de bureau.

Bien que toutes les fonctions du système se présentent avec une interface uniforme, l'opérateur peut apprendre progressivement



C. Pozzoni/Marka

Utilisation du tableur Multiplan sur Olivetti M20 par le directeur financier d'une entreprise.

le maniement du système, selon la séquence d'entraînement-utilisation suivante.

- Noyau de base de gestion de textes : *création/révision/composition de textes, impression, recherche dans les archives (pas toutes les fonctionnalités). Il opère sur des archives déjà définies. Ceci ne comprend pas les fonctions (plus complexes) de création et de restructuration d'archives entières, ni les fonctions de communication avec les autres postes de travail.*
- Noyau de communication : *gestion du courrier à l'arrivée et au départ. Ces fonctions reposent entièrement sur le concept d'archives utilisé au point précédent (les boîtes à lettres entrée et sortie sont des archives) avec, en plus, de simples opérations spécifiques.*
- Noyau avancé de gestion de textes : *définition des archives, recherche dans les archives (toutes les fonctionnalités). Il permet des opérations à caractère global sur les ressources accessibles à l'utilisateur particulier : définition ou restructuration d'archives, recherche d'informations sur l'ensemble des archives.*
- Noyau de fonctions personnelles : *agenda personnel. Ce groupe est indépendant et permet la gestion de notes et de rendez-vous personnels.*
- Noyau de procédure locale : *définition des procédures, définition de macro-instructions qui commandent l'exécution d'opérations et de procédures d'usage courant.*
- Noyau d'administration du système : *création de nouveaux utilisateurs et définition des standards opérationnels auxquels ils doivent se conformer.*

C'est ainsi que s'achève notre excursion dans le domaine de l'automatisation du travail de bureau. On a décrit d'une façon synthétique les qualités fonctionnelles d'un système intégrant des fonctions de type traditionnel (traitement de textes), d'autres plus sophistiquées (courrier électronique, intégration avec traitement de données, archivage et recherche des objets du bureau) jusqu'à des fonctions permettant l'adaptation du système à des normes et à des procédures spécifiques à un bureau.

(Extrait de «Un modèle de bureautique», CAHIERS D'INFORMATIQUE, 9e année, n° 21982, Honeywell Information Systems).

grammation, si simple soit-elle. En Basic on aurait eu :

IF Total = 0 THEN Total = Quantité * Coût

Il s'agit évidemment de la même instruction, mais sous une autre forme. En fait il existe une différence : la technique de recherche dans le fichier correspondant à la commande est différente. En Basic, un programme capable de remplacer le contenu de chaque enregistrement d'un fichier nécessite une boucle de lecture et d'écriture du premier au dernier enregistrement du fichier.
Les opérations à effectuer sont :

- lecture de la longueur du fichier (mémorisée par exemple dans l'enregistrement 1),
- boucle entre la première et la dernière donnée avec lecture d'un enregistrement - calcul - écriture.

Les instructions de boucle, de lecture et d'écriture du programme sont condensées dans la commande REPLACE ALL. Le système lit la longueur du fichier et met en fonction la boucle et les fonctions E/S.
La variété et le caractère exhaustif du jeu d'instructions prévu dans la base de données dépendent du type de programme. Dans sa forme avancée, l'ensemble des instructions prend l'aspect d'un langage de programmation et devient un instrument davantage destiné au programmeur qu'à l'utilisateur. C'est pourquoi, dans certains programmes, il a été prévu de sauvegarder sur disque et sous forme d'instructions les fonctions à exécuter. Elles sont donc lancées avec une seule commande. Le programmeur peut ainsi utiliser instructions et commandes comme s'il s'agissait d'une forme particulière de Basic ; l'utilisateur, lui, n'a plus qu'à exécuter les procédures prédéfinies.

Edition d'états

La préparation et la sortie d'états constitue la dernière fonction d'un programme de gestion de base de données.
L'utilisateur peut organiser l'édition en spécifiant quelles zones il souhaite imprimer et dans quelles conditions. Dans des procédures plus complètes, on peut aussi effectuer des calculs simples (des totalisations par exemple).
Dans certains cas, il est également possible d'imprimer les résultats intermédiaires en utilisant le contenu d'une zone comme le code de rupture (l'impression est mise en activité dès que varie le contenu de la zone considérée). Bien qu'elle soit présente dans certaines bases de données, cette forme de conditionnement des traitements est particulière au langage RPG (Report Program Generator) orienté vers la création de rapports.

Traitement de textes

L'automatisation du travail de bureau intègre la gestion de textes, lettres et documents. C'est dans ce but que différents programmes (**éditeurs de texte**) ont été écrits. Ils permettent la préparation d'un texte comme s'il s'agissait d'un ensemble quelconque de données, ce qui simplifie à l'extrême la correction et la duplication du texte. Les avantages d'une machine à traitement de texte, par rapport à une machine à écrire normale, sont évidents :

- mémorisation du texte ; on peut donc corriger ou modifier sans devoir tout réécrire ;
- sauvegarde sur disque souple, support beaucoup plus compact que le papier ;
- possibilité d'obtenir plusieurs impressions, éliminant la photocopie et évitant les problèmes de reproduction rencontrés lorsque l'original n'est pas parfait.

Dans ce domaine également, la variété des fonctions dépend du programme utilisé. Dans les éditeurs de textes récents, on trouve des fonctions très particulières telles que :

- vérification orthographique, dans plusieurs langues, d'après un dictionnaire défini par l'utilisateur ;
- possibilité d'insérer un ou plusieurs mots à l'intérieur d'un texte déjà complet, avec une nouvelle mise en page automatique ;
- contrôle des coupes de mots à droite (césure) ;
- préparation des tableaux ;
- fonctions simples de recherche sur le contenu des documents.

Certains systèmes permettent même l'introduction directe de documents. Des équipements complexes lisent le texte tapé sur papier et le traduisent en données élémentaires traitées et mémorisées par l'ordinateur. De la même manière, on transforme ou on traite des données manuscrites, des dessins ou des schémas.

GLOSSAIRE

GLOSSAIRE DE MULTIPLAN (VERSION OLIVETTI M20)

Actif (ACTIVE). Désigne ce qui est immédiatement accessible tel que fenêtre active, cellule active ou champ actif d'une commande.

Alignement (ALIGNMENT). Commande l'affichage du contenu des cellules (on dit aussi cadrage). Ce contenu peut être justifié à gauche, à droite ou centré.

Caractères (CHARACTERS). Symbole élémentaire pouvant être affiché à l'écran. Les caractères comportent des lettres, des chiffres, des symboles de ponctuation et des signes comme : $, +, & et %.

Cellule (CELL). Position élémentaire de la feuille de calcul dans laquelle il est possible de stocker une valeur numérique, une expression ou un texte. Le contenu d'une cellule détermine sa valeur. Le format d'une cellule indique comment son contenu doit être affiché. Une cellule a des coordonnées et il est possible de faire référence à une cellule par ses coordonnées ou par son nom.

Cellule active (ACTIVE CELL). Cellule en surbrillance (ou en mode vidéo inverse). Son contenu est affiché à la fois dans la cellule et dans la ligne état. Il peut être modifié au moyen de la commande EDIT (Édition).

Champ (FIELD). Partie d'une commande dans laquelle vous devez donner une information à Multiplan pour l'exécution de la commande. Quand Multiplan montre un champ, il propose une réponse appelée « option proposée » (voir : **Réponse proposée**). Vous pouvez accepter cette option ou en entrer une autre, si elle ne convient pas.

Charger (LOAD). Action consistant à charger une feuille de calcul, préalablement sauvegardée sur disque, en vue de la rendre active. Cette action se fait au moyen de la commande TRANSFER LOAD, la commande TRANSFER SAVE étant utilisée pour transférer la feuille vers le disque de la mémoire centrale du système.

Colonne (COLUMN). Rangée verticale de cellules allant de haut en bas de la feuille. Une feuille peut contenir jusqu'à 63 colonnes numérotées de 1 à 63.

Commande (COMMAND). Instruction pour demander à Multiplan d'effectuer une tâche. Une commande peut avoir un ou plusieurs champs pour indiquer à Multiplan comment exécuter la commande.

Contenu (CONTENTS). Désigne ce qui se trouve à l'intérieur d'une cellule. Si une cellule est vide, son contenu est appelé Blanc, ou Vide ou Rien. Sinon la cellule contient soit une donnée (texte ou nombre), soit une expression (formule). Si elle contient une expression, la valeur résultant de son calcul est généralement affichée.

Couplée (LINKED). Deux fenêtres sont couplées lorsque leur défilement est synchronisé. Deux feuilles de calcul sont couplées ou liées s'il y a des liens permettant la recopie automatique d'informations d'une feuille vers l'autre.

Curseur d'édition (EDIT CURSOR). Partie en surbrillance de la commande sur la ligne commandes qui peut être réduite à un caractère ou contenir un champ tout entier. Le curseur d'édition est déplacé, par l'intermédiaire des touches d'édition, pour permettre d'éventuelles modifications de la commande.

Défilement (SCROLLING). Déplacement de l'écran, ligne par ligne, au dessus de la feuille de calcul. Le défilement est obtenu au moyen des touches de direction et peut se faire dans les quatre directions : vers le haut ↑, vers le bas ↓, vers la droite →, vers la gauche ←. Quand Multiplan déplace l'écran à gauche, colonne par colonne, on parle de « défilement horizontal ». Lorsque l'écran est déplacé ligne par ligne, on dit alors que c'est un « défilement vertical ».

Domaine (RANGE). Groupe d'au moins deux cellules pouvant être référencées au moyen d'une expression du type L3:L8 ou C18:C25 qui désignent respectivement les cellules comprises entre les lignes 3 et 8 incluses et les cellules comprises entre les colonnes 18 et 25 incluses. Il est défini par le symbole : (deux points). Voir aussi **Référence.**

Ecran/Vidéo. Moniteur affichant les caractères alphanumériques. L'écran est divisé horizontalement en 25 lignes. Multiplan utilise les 4 dernières pour information et contrôle. Elles reproduisent de bas en haut la ligne état, la ligne messages et les deux lignes commandes. Les 21 lignes restantes sont consacrées à la construction de la feuille et servent à sa visualisation, en partie par l'intermédiaire d'une ou plusieurs fenêtres.

Edition (EDIT). L'édition consiste à modifier une réponse d'un champ de commande. Les touches d'édition sont utilisées pour déplacer le curseur à l'intérieur de la réponse, les touches caractères pour insérer ou remplacer des caractères.

Expression (FORMULA). Formule décrivant la façon dont une valeur doit être calculée. Quand le contenu de la cellule est modifié, en mode « recalcul automatique », Multiplan recalcule toutes les expressions de la feuille de calcul.

Fenêtre (WINDOW). Portion de l'écran qui permet de voir une partie de la feuille de calcul. L'utilisateur peut ouvrir jusqu'à 8 fenêtres simultanément. Le numéro de fenêtre (de 1 à 8) est affiché dans le coin supérieur gauche de la fenêtre. C'est la commande WINDOW qui les ouvre et les ferme.

Fenêtre active (ACTIVE WINDOW). Fenêtre contenant la cellule active. Le numéro de la fenêtre active figure en surbrillance sur l'écran.

Feuille de calcul (WORKSHEET). Grille de cellules affichées par Multiplan pour stocker des expressions ou des valeurs dont la dimension maximale est de 63 colonnes sur 255 lignes.

Feuille de soutien (SUPPORTING SHEET). Feuille contenant des valeurs qui sont recopiées dans la feuille récapitulative une fois celle-ci chargée en mémoire. Cette opération est réalisée à l'aide de la commande EXTERNAL COPY et NAME. Voir aussi **Liaison externe.**

Feuille récapitulative (DEPENDENT SHEET). Une feuille est dite récapitulative « dépendante » si son chargement entraîne celui des valeurs venant d'une feuille de soutien. Les liens entre feuille de soutien et feuille récapitulative sont établis au moyen de la commande EXTERNAL COPY (Externe recopie)

Fichier (FILE) Ensemble de données portant un nom, stocké sur un disque ou sur une disquette. Multiplan sauvegarde les feuilles de calcul sous forme de fichiers. Tous les fichiers disque ne correspondent pas à des feuilles de calcul, mais les fichiers représentant les feuilles de calcul peuvent être chargés par Multiplan ou liés à d'autres feuilles.

Fonction (FUNCTION). Calcul mathématique ou statistique intégré dans Multiplan. Par exemple SUM ou AVERAGE (Somme ou Moyenne).

Format (FORMAT). Le format indique comment une cellule doit être affichée. Il fournit la ponctuation et l'alignement des valeurs à afficher. Le format peut être spécifié pour une ou plusieurs cellules au moyen de la commande FORMAT CELL (Format cellule) Les cellules non formatées sont affichées en format standard, modifiable avec la commande FORMAT DEFAULT (Format standard).

Groupe de cellules (GROUP OF CELLS) Ensemble d'une ou de plusieurs cellules de la feuille de calcul, auquel on peut donner un nom. Exemple *Ventes*.

Itération. Répétition d'un calcul jusqu'à l'obtention, par approximation, du résultat souhaité.

Liaison externe. En Multiplan, ce terme indique la connexion entre feuilles, permettant à une cellule active d'utiliser les données d'une feuille non active appelée, pour la circonstance, « feuille de soutien ». Il est nécessaire d'attribuer un nom aux données à copier (avec la commande NAME) ou de les marquer par une référence absolue. Il est alors possible d'utiliser les données d'une feuille de soutien comme expressions dans la feuille active.

Ligne (ROW). Rangée horizontale de cellules allant de gauche à droite de la feuille. La feuille de calcul contient jusqu'à 255 lignes numérotées de 1 à 255.

Ligne commandes (COMMAND LINE). Une ou deux lignes physiques situées juste au dessous de la feuille de calcul et destinées à l'affichage du menu principal ou des sous-commandes

Ligne état (STATUS LINE). Dernière ligne de l'écran où Multiplan affiche certaines informations telles, par exemple, le contenu de la cellule active ou le pourcentage de mémoire disponible.

Ligne messages (MESSAGE LINE). Avant-dernière ligne de l'écran où Multiplan affiche les messages.

Menu (MENU). Liste d'alternatives (proposées dans la ligne des commandes) qui permet à l'utilisateur de choisir la commande désirée.

Message (MESSAGE). Informations données par Multiplan dans la ligne messages, destinées à prévenir l'utilisateur soit de problèmes rencontrés ou de lui suggérer le type d'action attendue par Multiplan.

Nom (NAME). Dans Multiplan, un nom permet de désigner une cellule ou un groupe de cellules. Les noms sont, par exemple, utilisés dans des expressions pour faire référence à ces cellules

Nom de fichier (FILENAME). Nom attribué à un fichier pour servir de référence. Pour sauvegarder une feuille de calcul sur disque, il faut lui attribuer un nom de fichier. Ce nom servira ultérieurement à recharger la feuille de calcul ou à la « coupler » avec une autre feuille. Voir **couplée**.

Pointeur de cellule (CELL POINTER). Appelé aussi curseur lumineux, le pointeur est déplacé par l'utilisateur, au moyen des touches de direction ou de la commande GOTO (Aller à), afin de mettre en surbrillance une cellule parmi toutes les cellules de la feuille de calcul Mise en surbrillance, cette cellule devient active.

Protection (LOCK). Verrouillage de certaines cellules contenant des expressions ou du texte afin de les prémunir contre toute modification par inadvertance.

Référence (REFERENCE). Désignation d'une cellule ou d'un groupe de cellules. La référence la plus simple consiste à donner ses coordonnées : exemple L9C2. Il existe d'autres moyens pour établir une référence qui peut être relative ex. : [L(−1)C] ou absolue. L6 fait référence à la ligne 6, C4 à la colonne 4. Une référence peut se composer d'intersections de références, de domaines de références ou d'unions de références. Un nom déjà défini fait référence à une ou à plusieurs cellules.

Référence absolue (ABSOLUTE REFERENCE). Elle utilise les numéros de ligne et de colonne. Par exemple L17C12.

Référence relative (RELATIVE REFERENCE). Elle indique un nom ou un déplacement par rapport à la position de la cellule active : exemple L(+1) C(−2).

Répertoire (DIRECTORY). Table contenant les noms des fichiers contenus sur un volume (disque ou disquette)

Réponse proposée (PROPOSED RESPONSE). Réponse proposée par Multiplan à partir de la réponse la plus récente donnée par l'utilisateur ou à partir du contexte pour aider l'utilisateur.

Sauvegarde (SAVE). Action consistant à sauvegarder sur disque ou sur disquette la feuille de calcul afin de pouvoir l'utiliser ultérieurement.

Surbrillance (HIGHLIGHT). Elle est utilisée par Multiplan pour mettre en évidence certaines parties de l'écran. elle indique, en particulier, la cellule active, la position du curseur d'édition, le numéro de la fenêtre active et l'élément courant du menu. Cette surbrillance peut apparaître sur certains systèmes comme le mode « vidéo inverse ».

Touche d'annulation (CANCEL KEY). Touche fonction provoquant l'annulation de la commande en cours et l'affichage du menu principal

Touche suivante non protégée (NEXT UNLOCKED CELL KEY). Elle déplace le curseur d'une cellule non protégée à la suivante. Cette touche est utilisée pour déplacer la surbrillance sur des cellules contenant des nombres (ni expressions ni texte) afin de réaliser rapidement des simulations du type : que se passe-t-il si je modifie tel paramètre ? (WHAT IF).

Touches de direction (DIRECTION KEY). Elles déplacent le pointeur de cellule. Les touches ↑, ↓, ←, →, déplacent le pointeur d'une cellule à la fois. La touche HOME le renvoie sur la cellule en haut et à gauche de la fenêtre active.

Touches d'édition (EDIT KEY). Elles déplacent le curseur sur la ligne commandes. Elles comprennent par exemple : WORD RIGHT, WORD LEFT (mot droit, mot gauche), CHARACTER RIGHT, CHARACTER LEFT (caractère droit, caractère gauche).

Touches de fonction (FUNCTION KEY). Elles déclenchent immédiatement l'exécution d'une opération par le système Multiplan. Elles comprennent : CANCEL (Annule), NEXT WINDOW (fenêtre suivante), CR (Retour chariot). Voir également **Touches de direction** et **Touches d'édition.**

STRUCTURE DES COMMANDES MULTIPLAN

Les commandes figurant dans le menu principal sont sélectionnées en tapant leur initiale (exemple: A pour ALPHA) ou bien en positionnant le curseur directement sur la commande avec les touches de tabulation et en appuyant sur la touche CR pour l'exécution de la commande. On peut introduire des valeurs numériques dans la cellule active sans qu'il soit nécessaire de passer par la commande VALUE.
Des sous-commandes sont couplées à la plupart des commandes principales. Leur menu est visualisé après une première sélection de la commande principale.

CTRL + C		Annule une commande et retourne au menu principal.
S1	**Tabulation droite**	Déplace le curseur dans le menu vers la commande/sous-commande suivante. Dans la sous-commande, elle déplace le curseur vers le champ suivant.
S2	**Tabulation gauche**	Déplace le curseur vers la commande/sous-commande précédente.
SHIFT + ↑ ↓ ← →		Déplace le curseur dans le sens des flèches, bloque l'insertion en maintenant le statut ALPHA/VALUE.
CR		Exécute la commande. Introduit le texte ou la valeur.
@		Transforme les références (relatives) des cellules en références absolues.

A ALPHA
Introduit un texte ou une valeur numérique non utilisable dans la cellule active.

CR ou SHIFT + ↑ ↓ ← →
L'utilisation des touches direction confirme l'introduction et déplace le pointeur dans la direction indiquée.

B BLANK
Spécifie les références d'une ou de plusieurs cellules.

CR
Le contenu des cellules spécifiées est annulé.

C COPY

R Right
Détermine le nombre de cellules à droite de la cellule actuelle (dans laquelle on veut recopier le contenu de la cellule active).

CR

D Down
Indique le nombre de cellules placées sous la cellule actuelle sur laquelle on veut copier le contenu de la cellule active.

CR

F From
Définit les références d'une ou de plusieurs cellules à copier dans les références décrites comme cellules destinataires.

CR
La commande COPY FROM inclut aussi bien COPY RIGHT que COPY DOWN. Elle permet, en outre, la copie de cellules non contiguës.

D DELETE

R Row
Définit les références à tout ou partie des cellules d'une ou de plusieurs lignes.

CR
Les zones spécifiées sont éliminées de la feuille. L'exécution de la commande provoque la compression du tableau vers le haut.

C Column
Définit les références à tout ou partie des cellules d'une ou de plusieurs colonnes.

CR
La zone spécifiée est éliminée de la feuille. L'exécution de la commande provoque la compression du tableau vers la gauche.

E EDIT
Edite des textes et des formules grâce aux touches d'édition.

CR ou SHIFT + ↑ ↓ ← →

F FORMAT

C Cells
Définit les références d'une ou de plusieurs cellules dont on veut modifier le format d'affichage.

CR

D Default

C Cells Définit le format d'affichage et le type d'alignement des cellules.

CR

W Width Définit la largeur (en nombre de caractères) pour toutes les colonnes de la feuille.

CR

	O Options	Définit et vérifie les options de format relatives aux cellules.
	CR	Les réponses mises en évidence dans les champs des commandes correspondent aux options actuelles.
	W Width	Définit, pour les colonnes spécifiées, une largeur différente de celle par défaut (Voir commande Format Default Width).
	CR	
G GOTO	**N Name**	Positionne le pointeur sur la cellule correspondant au nom spécifié.
	CR	Au cas où le nom repérerait une zone de cellules, le pointeur se positionne sur la première cellule de la zone spécifiée (cellule en haut à gauche).
	R Row-Col	Introduit le numéro de la Rangée-Colonne.
	CR	Positionne le pointeur sur la cellule pour laquelle la référence a été spécifiée.
	W Window	Positionne le pointeur sur la cellule dont la référence a été spécifiée à l'intérieur de la fenêtre indiquée.
	CR	La commande provoque le défilement de la fenêtre jusqu'à ce que la cellule spécifiée soit en première position, en haut à gauche de la fenêtre.
H HELP	**R Resume**	Remet en vigueur la feuille active à partir du point où la commande HELP avait été rappelée.
	S Start	Affiche la première page du fichier HELP.
	N Next	Affiche la page suivante du fichier HELP.
	P Previous	Affiche la page précédente du fichier HELP.
	A Applications	Affiche une liste de problèmes communs pouvant se présenter dans l'utilisation de Multiplan ainsi que les commandes à utiliser pour les résoudre.
	C Commands	Affiche la description de chaque commande de Multiplan, à commencer par la première : ALPHA.
	E Editing	Affiche la description de l'éditeur de Multiplan.
	F Formulas	Met en évidence des informations concernant les règles à suivre et une liste de fonctions pouvant être utilisées dans la définition des expressions.
	K Keyboard	Met en évidence une liste des touches et leur signification respective, à utiliser en Multiplan.
I INSERT	**R Row**	Insère tout ou partie d'une ou de plusieurs lignes, avec un contenu nul dans les positions qui précèdent la ligne active.
	CR	
	C Column	Insère tout ou partie d'une ou de plusieurs colonnes, avec un contenu nul dans les positions qui précèdent la colonne spécifiée.
	CR	
L LOCK	**C Cells**	Bloque/débloque une ou plusieurs cellules pour prévenir une éventuelle modification de leurs valeurs.
	CR	
	F Formulas	Bloque toutes les cellules contenant des textes ou des expressions pour prévenir leur modification accidentelle.
	Y	Confirme la demande.
M MOVE	**R Row**	Définit les références relatives à des lignes à déplacer à l'intérieur de la feuille ainsi que les références de la position sur laquelle elles doivent être déplacées.
	CR	
	C Column	Définit les références relatives à des colonnes à l'intérieur de la feuille ainsi que les références de la position sur laquelle elles doivent être déplacées.
	CR	

N NAME

Affecte un nom à une ou à plusieurs cellules selon ce qui a été spécifié dans les champs de la commande.

CR
SHIFT + ↑ ↓ ← →

O OPTIONS

Sélectionne certaines options ayant une incidence sur le calcul de la feuille, telles qu'un nouveau calcul automatique, ou encore la définition d'une cellule contenant le « test d'aboutissement » pour les opérations d'itération.

CR

P PRINT

P Printer

Imprime la feuille active selon le format défini à l'aide des commandes Print Margins et Print Options.

CR

F File

Sauvegarde la feuille active dans un fichier sur disque, dans un format pouvant être imprimé.

CR

M Margins

Définit le format des pages d'imprimante en termes de marge gauche, de longueur de ligne et de nombre de lignes par page.

CR

O Options

Définit les options d'impression :
1/ impression d'une portion de la feuille,
2/ impression des expressions plutôt que des valeurs,
3/ suppression des numéros de lignes ou de colonne,
4/ définition de certains paramètres matériels (imprimante).

CR

Q QUIT

Sort du programme Multiplan en cours. La confirmation est demandée.

Y

Sert à l'exécution de la commande. La feuille n'est pas sauvegardée sur disque.

S SORT

Définit tout ou partie de la colonne dont on veut trier les valeurs.

CR

Le tri se fait dans un ordre croissant ou décroissant, pour les valeurs numériques comme pour les textes ou pour les valeurs logiques.

T TRANSFER

L Load

Définit un nom de fichier qui sera chargé en mémoire en vue d'un calcul.

SHIFT + ↑ ↓ ← →

En utilisant les touches de direction et en ne spécifiant pas le nom (champ vide), on appelle à l'écran la liste des fichiers sur disque ; on peut ainsi sélectionner le fichier à rappeler en mémoire.

CR

S Save

Sauvegarde la feuille sur disque et lui attribue le nom spécifié.

CR

Dans le cas d'une feuille préexistant sur le disque avec le même nom, on demande confirmation pour l'exécution de la commande.

Y

C Clear

Efface la feuille active de la mémoire. La confirmation est demandée pour l'exécution de la commande.

Y

D Delete
SHIFT + ↑ ↓ ← →
CR

Annule la feuille spécifiée dans le disque. On peut utiliser les touches de direction pour visualiser la liste de tous les fichiers existant sur le disque.

O Options
CR

Définit l'unité disque et le format de mémorisation/rappel des feuilles.

R Rename
CR

Sauvegarde la feuille active sur disque en lui attribuant un nom différent. La commande redéfinit d'éventuelles connexions externes.

V VALUE
CR

Introduit une expression.
L'expression est introduite.

W WINDOW

S Split

H Horizontal
CR

Définit une fenêtre dans le cadre du tableau. La feuille active est fractionnée horizontalement.

V Vertical
CR

Définit une fenêtre dans le cadre de la feuille. La feuille active est fractionnée verticalement.

T Titles
CR

Définit une fenêtre contenant les titres des lignes et des colonnes visualisées.
La fenêtre active est subdivisée en 2 ou 4 fenêtres selon que les titres sont relatifs aux lignes, aux colonnes ou aux deux à la fois.

B Border
CR

Recadre ou élimine le cadrage de la fenêtre spécifiée.

C Close
CR

Elimine de l'écran la fenêtre spécifiée.

L Link
CR

Définit/élimine des connexions entre les fenêtres spécifiées. La connexion implique le défilement simultané des fenêtres connectées.

X EXTERNAL

C Copy
CR

Demande le nom de la feuille, l'adresse (Rangée-Colonne) des cellules destinataires et le mode de connexion.
Les valeurs sont copiées depuis la feuille de soutien.

L List

Donne les noms des feuilles de soutien et des feuilles dépendantes de la feuille active.

U Use
CR

Introduit le nouveau nom du fichier.
Substitue ce nom à celui précédemment utilisé.

Le langage Assembleur

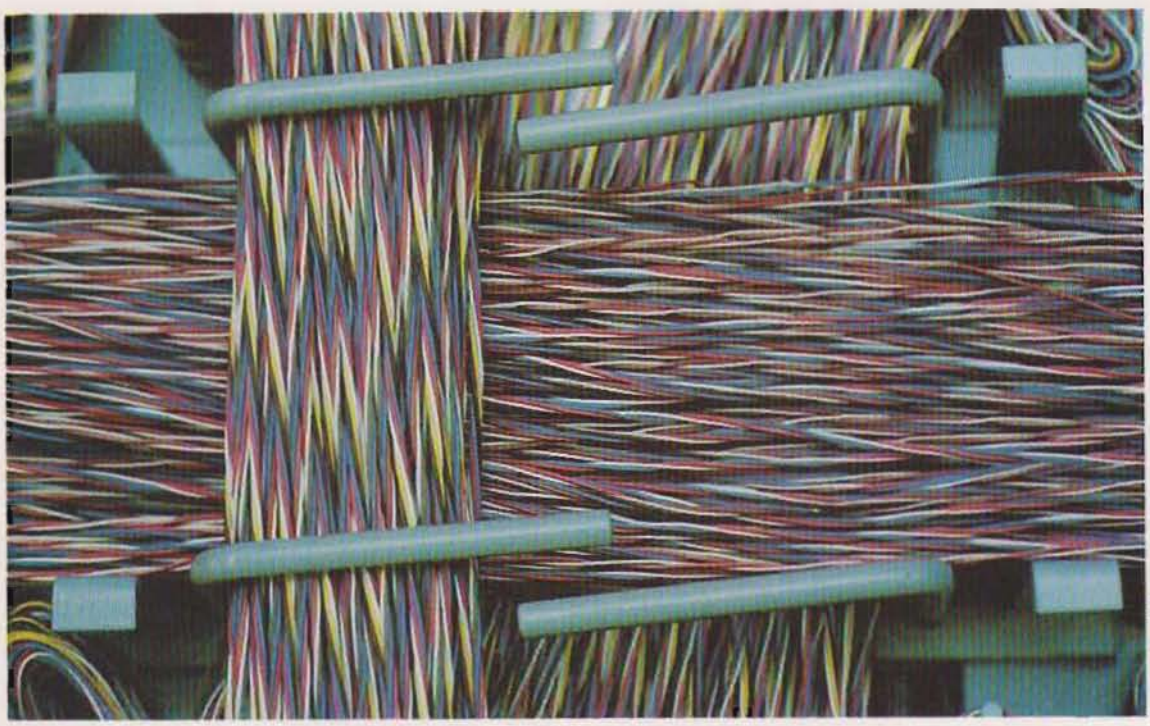
Au centre de chaque système, qu'il s'agisse d'un mini-ordinateur ou d'un micro à usage professionnel ou domestique, se trouve le microprocesseur : c'est l'unité centrale (UC). Il est constitué d'un ensemble de circuits intégrés, transistors, diodes, résistances et condensateurs occupant un espace très réduit. On peut faire entrer jusqu'à 450 000 transistors dans un carré de 5,5 mm de côté ; 25 000 transistors pourraient tenir dans une tête d'épingle.

Le microprocesseur a deux fonctions principales : il contrôle l'ensemble des unités physiques du micro-ordinateur et exécute les calculs arithmétiques et logiques. Ses tâches sont commandées par un programme écrit dans un langage très différent du Basic : le langage-machine, seul langage compréhensible par le microprocesseur. Les instructions sont constituées par les séquences d'états logiques 0 et 1, organisées en octets ou en mots selon le type de microprocesseur.

Chaque instruction a une fonction précise. C'est pourquoi, même pour une opération simple telle que le transfert d'une donnée (de/ou vers la mémoire externe, par exemple), toute une séquence d'instructions est nécessaire. L'écriture d'un programme, directement en langage-machine, n'est pas facile. Les risques d'erreurs sont nombreux et on obtient des programmes qu'il est difficile de comprendre ou de modifier. Dans la programmation d'un micro-ordinateur en un langage évolué tel que le Basic, la traduction des instructions et des commandes en langage-machine est faite par un ensemble des programmes résidents, en mémoire vive (RAM) ou morte (ROM) : l'interprète, le compilateur et le superviseur. Toutefois, si on demande à la machine d'exécuter certaines tâches, il n'est pas possible de recourir uniquement aux langages de haut niveau.

Ces derniers sont très souples (on dit souvent qu'ils sont orientés vers l'homme) mais leur

Les bus de connexion externe d'un ordinateur.



A. Trocchia/AFE

exécution est très lente. Pour une raison simple : il faut sans cesse rappeler le programme interpréteur qui traduit chaque instruction Basic par son équivalent en séquences d'instructions langage-machine intelligibles et directement accessibles par le microprocesseur. Une telle lenteur n'est pas supportable pour certaines applications graphiques ou vidéo qui font couramment appel à des animations à l'écran et qui manipulent de volumineux fichiers de données. Aussi la programmation est-elle souvent faite en langage-machine. Pas directement toutefois. Pour atteindre cet objectif plus commodément et plus facilement qu'avec le recours (fastidieux) aux codes hexadécimaux, on utilise le langage d'assemblage.

Bien que très proche du langage-machine, l'Assembleur est plus maniable que le langage-machine. Il permet notamment d'éviter les codes numériques. A chaque instruction en Assembleur correspond une instruction en langage-machine (une séquence d'états logiques 0 et 1, généralement contenus dans un ou plusieurs bits). De plus, ce langage met à la disposition de l'informaticien des outils d'aide à la programmation qui vont lui permettre d'écrire plus aisément des applications, par exemple définition d'étiquettes ou de noms symboliques pour les variables. L'Assembleur, le plus précis de tous les langages, traduit donc un programme en langage-machine. Avant d'analyser les caractéristiques de ce nouveau langage, nous allons procéder à une description d'ensemble d'un microprocesseur type.

Cette étape est nécessaire dans la mesure où l'Assembleur ressemble au langage-machine et que son fonctionnement est très proche de celui du microprocesseur. Ses caractéristiques sont donc définies, en grandes parties, par la structure du microprocesseur.

Structure et fonctionnement du microprocesseur

Chaque microprocesseur possède une logique de fonctionnement particulière, qui respecte toutefois certaines nécessités fondamentales, à savoir des fonctions dont la présence est indispensable.

Un microprocesseur contient un jeu d'instructions codifiées (répertoire) qui permet d'exécuter des opérations bien définies. On appelle codes de programme une séquence de ces instructions résidant en mémoire centrale. Les instructions et les données nécessaires à leur exécution doivent être présentes en mémoire. Dans les architectures très simples, il n'existe pas de différence physique entre mémoires de programmes (liste d'instructions) et mémoires de données : les données et les instructions sont généralement rangées dans des adresses contiguës. Dans les architectures plus complexes, la mémoire est divisée en deux zones dédiées, l'une aux **codes** (instructions destinées à la machine), l'autre aux **données** (permettant l'exécution des instructions). L'ensemble des instructions du programme peut alors être chargé dans une mémoire morte (ROM) et les données, différentes à chaque exécution, dans une mémoire vive (RAM) (voir schéma, page ci-contre).

Supposons que l'on veuille calculer la somme de deux nombres et mémoriser le résultat, le programme Basic serait (le code LET peut être omis) :

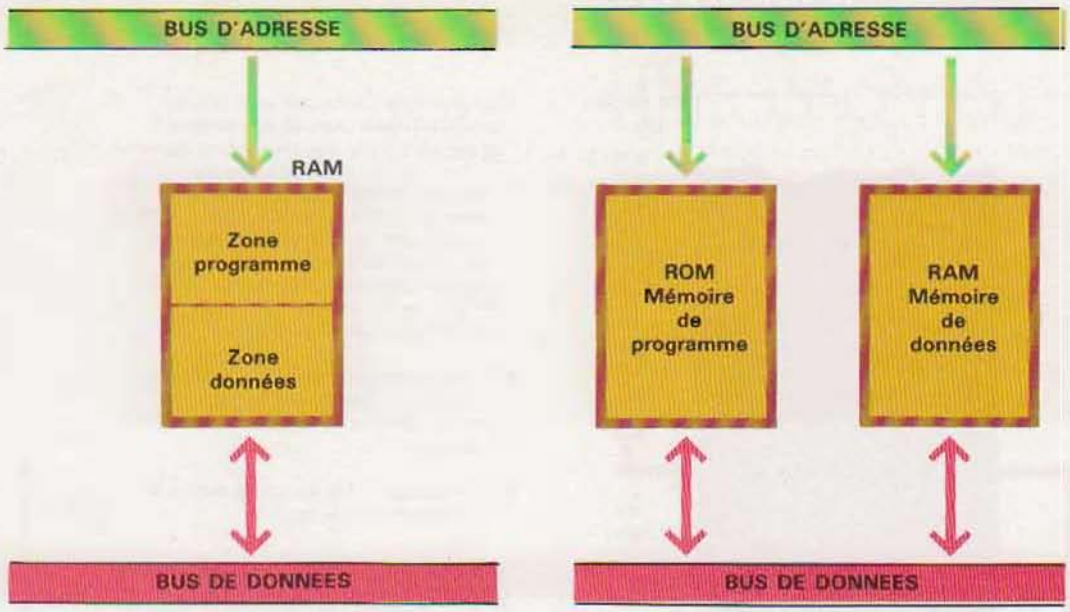
```
10 LET V=10
20 LET S=V+20
```

L'unité centrale (UC) doit donc effectuer les opérations suivantes (séquence) :

1 / Extraction et stockage de la valeur 10 dans une mémoire interne (lecture)
2 / Extraction de la valeur 20 et addition au contenu de la mémoire interne (exécution)
3 / Rangement du résultat en cellule mémoire symbolisée par S (écriture)

Dans cet exemple, la valeur 10 a été rangée dans une adresse V, alors que la valeur 20 est fournie comme donnée numérique directement associée à l'instruction d'addition (ligne 20). L'UC doit pointer la mémoire V pour extraire la valeur 10. En revanche, pour calculer la somme S, elle doit utiliser une valeur numérique (20) qui suit le code de l'instruction. La séquence des opérations à lancer sera donc :
1 / Lire le contenu de la mémoire V
2 / Ajouter à cette valeur le nombre 20
3 / Ecrire le résultat dans la mémoire S

STRUCTURE DE LA MEMOIRE
en fonction de l'architecture du microprocesseur



Page 868, vous trouverez un exemple d'exécution de ce programme sur un microprocesseur à mots de 8 bits.

Comme on l'a déjà vu au début de l'encyclopédie (pages 125 et suivantes), un microprocesseur renferme différents dispositifs qui lui permettent d'exécuter un ensemble d'instructions selon une séquence précise pas à pas. Il s'agit de l'**horloge interne** qui constitue une des pièces maîtresses du système. Le rôle de ce dispositif, intégré ou non au microprocesseur est à la base de la synchronisation de tous les événements ou actions élémentaires que l'unité centrale doit accomplir dans un ordre pré-établi.

L'horloge génère périodiquement des signaux électriques qui se transforment en états logiques successifs (0 ou 1), dont la durée varie de 125 nanosecondes (1 ns = 10^{-9} seconde) à environ 55 ns et davantage dans les systèmes plus rapides (voir schéma page 869, en haut).

Ce signal électrique agit sur l'UC qui lance à son tour une opération à chaque changement d'état.

Selon la nature du microprocesseur ainsi que de la tâche à exécuter, chaque opération de l'UC peut durer un ou bien plusieurs intervalles d'horloge.

Examinons dans le détail les différentes phases de fonctionnement de l'UC. On appelle **cycle d'instructions** le temps nécessaire à l'UC pour lire une instruction en mémoire et pour l'exécuter. Une instruction peut être contenue dans un ou plusieurs mots-octets de la mémoire.

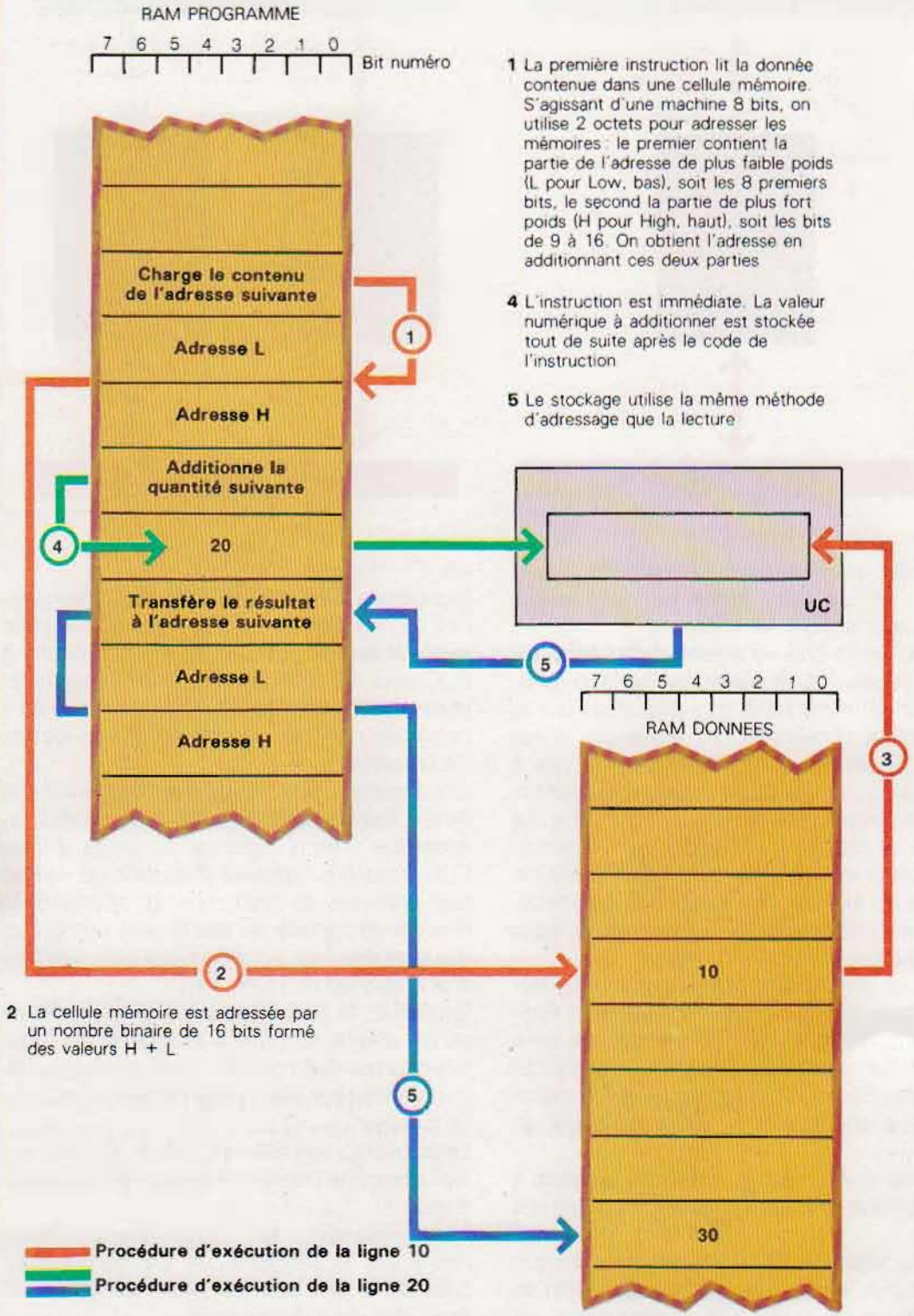
On appelle **cycle machine** l'intervalle de temps (qui correspond, pour la machine, à un ensemble d'états logiques) au cours duquel l'UC transfère l'adresse d'un mot sur le bus des adresses et recherche en mémoire la donnée demandée qu'elle charge sur le bus des données (voir schéma page 869 : Lecture d'une donnée en mémoire).

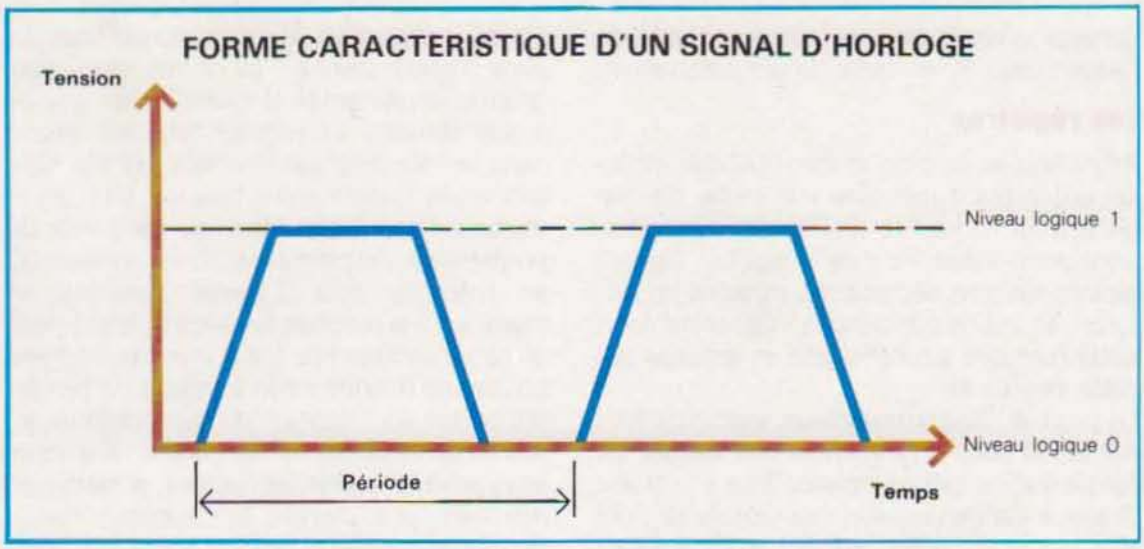
En réalité, la procédure décrite ici ne donne qu'un aperçu simplifié à l'extrême du fonctionnement réel de l'UC, dont la complexité est étroitement liée à l'architecture du microprocesseur.

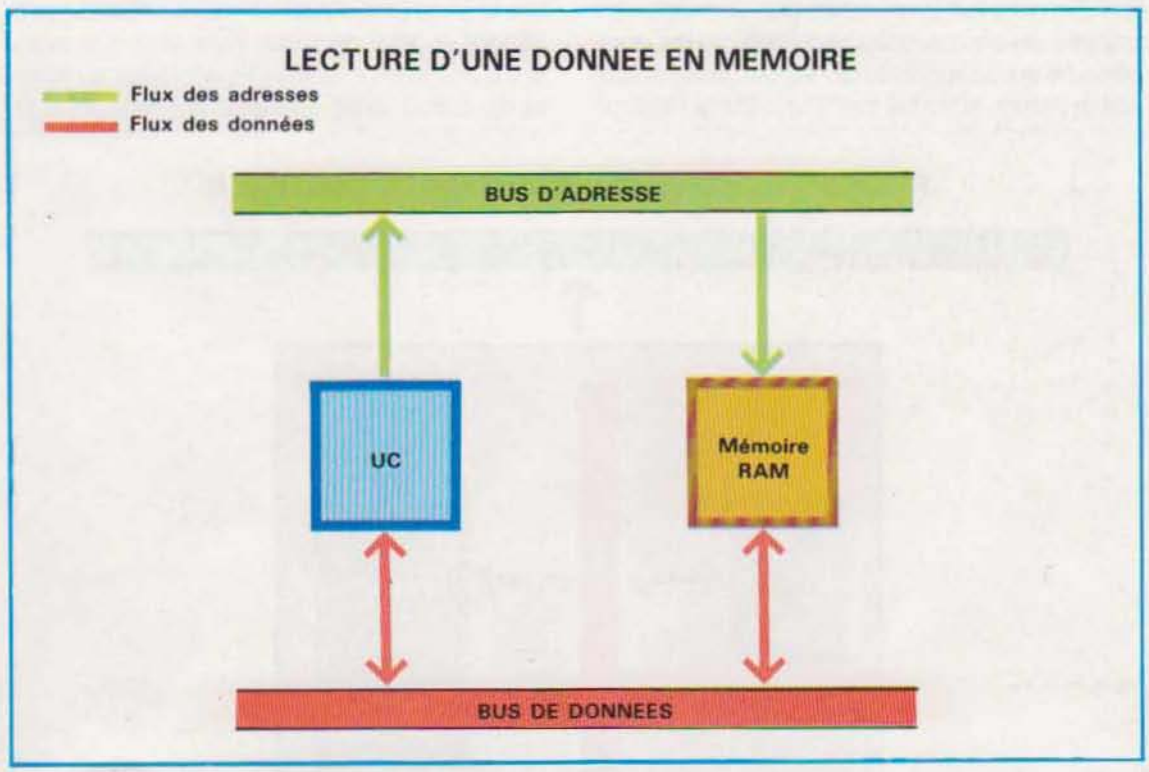
Le décodage suivi de l'exécution de l'instruction constitue l'essentiel d'un cycle d'instructions.

Pour décoder les informations transmises par la mémoire, le microprocesseur procède à leur analyse pour les comparer avec son jeu d'instructions.

EXECUTION D'UN PROGRAMME







Tout microprocesseur renferme un dispositif appelé **Unité de contrôle** (Control Unit). Celle-ci gère une mémoire morte (**mémoire de contrôle**) contenant les codes de l'ensemble des instructions que le microprocesseur est capable d'exécuter (voir schéma page 870)

Dès réception d'une instruction, l'UC lance un ou plusieurs cycles machine, en fonction du nombre d'octets qu'elle comporte. Il existe une kyrielle de microprocesseurs ; aussi est-il difficile d'entrer dans le détail de leur fonctionnement sans préciser le type de matériel employé.

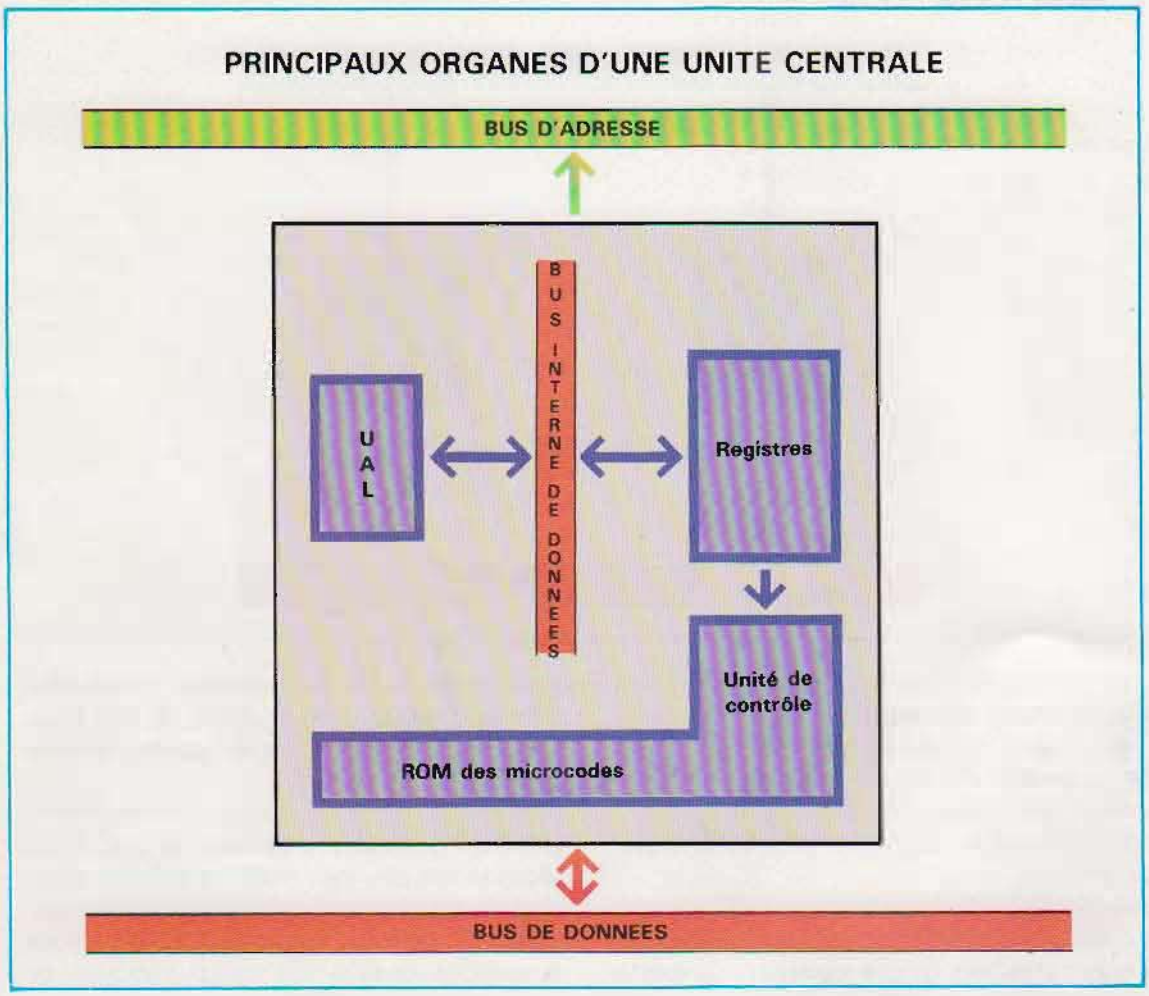
D'une façon générale, il faut avant tout comprendre le mécanisme de base de l'UC. Nous allons le voir plus loin, mais il a déjà été exposé dans les chapitres précédents sur les systèmes à base de microprocesseurs. Nous illustrerons ensuite les caractéristiques de

certains d'entre eux, pour pouvoir décrire l'Assembleur et ses outils de programmation.

Les registres

Pour dérouler un programme, l'UC doit disposer d'une place mémoire suffisante, de manière à lui permettre d'effectuer les opérations demandées. Pour ce faire, l'UC "libère" les informations nécessaires, effectue les lectures et les modifications. Concrètement, cette mémoire est découpée en espaces appelés **registres**.

Le premier, l'**accumulateur** (accumulator), est utilisé pour le rangement des valeurs sur lesquelles les calculs doivent être effectués. Grâce à l'accumulateur, les calculs se font plus vite (à l'intérieur de l'UC) qu'en agissant directement sur la mémoire des données. Les valeurs présentes dans la mémoire des données ne sont modifiées qu'au terme du calcul entièrement effectué par l'UC. Dans l'exemple de la page 868, la valeur 10 est chargée dans l'accumulateur ; au cours du calcul (instruction suivante), la valeur 20 est ajoutée à son contenu. Le résultat final est stocké dans la mémoire des données où elle sera conservée jusqu'à son annulation par l'introduction d'une autre valeur ou par l'arrêt du programme. On peut aussi écrire la valeur 20 en mémoire pour l'ajouter (toujours en mémoire) à la précédente. En procédant ainsi, on serait contraint de faire un accès mémoire pour toute donnée à additionner. Si le problème portait sur l'addition de sept nombres, au lieu de deux, on aurait sept accès : il faudrait alors prendre le premier nombre, le mettre en mémoire, puis prendre le deuxième, l'additionner au premier et stocker le résultat, prendre le troisième, l'additionner au résultat précédent et ainsi de suite. Pour limiter le nombre d'accès en mémoire et accroître la rapidité du calcul, on a couramment recours à un



registre spécial, l'accumulateur, qui fonctionne comme une mémoire intermédiaire pendant tous les calculs. En réalité, certaines opérations élémentaires sont faites directement dans la mémoire des données. Ainsi, pour ajouter 1 au contenu d'une mémoire, il existe une instruction qui n'exige pas de transfert dans l'accumulateur et qui procède directement à cette addition.

Le **compteur de programme** ou "compteur ordinal" est un autre registre aussi indispensable. C'est là qu'est stockée l'adresse de l'instruction que le microprocesseur doit exécuter. Ce registre, appelé PC (Program Counter) contiendra une valeur initiale représentant l'adresse en mémoire de la première instruction à exécuter. Le cycle d'instruction commence lorsque l'UC transfère le contenu du PC dans le bus d'adresses et en extrait la première instruction. L'UC incrémente (c'est-à-dire qui augmente d'une valeur constante) ensuite le contenu du PC, si bien que le cycle d'instruction suivant chargera en mémoire l'instruction suivante. Si l'instruction occupe plus d'un mot dans la mémoire, elle est transférée par étapes, et l'UC incrémente chaque fois la valeur du PC. Dans le schéma de la page 868, la première instruction (charge le contenu...) occupe 3 mots ; il faut donc la charger en 3 phases (l'instruction nécessite au total 4 cycles machine : 3 pour le chargement des codes et un pour celui de la donnée). L'UC exécute les instructions séquentiellement, à moins qu'une instruction de branchement ne change la valeur contenue dans le compteur de programme.

Un autre registre, le **registre d'instruction** (IR, Instruction Register), existe généralement dans l'UC. Il renferme le code de l'instruction jusqu'à son décodage. Certains micro-ordinateurs en ont deux, ce qui permet de prélever une instruction pendant la phase d'exécution de la précédente. Cette technique, appelée "pipelining", permet des vitesses d'exécution très élevées, mais il est rare qu'un programme-utilisateur puisse gérer le registre IR. Il existe, en outre, certains registres qui repèrent les positions des données, positions qui ne sont pas nécessairement séquentielles.

Pour avoir accès à une donnée résidant en mémoire (en vue d'une opération de lecture ou d'écriture), il faut connaître l'adresse de la donnée. Toujours pour aller plus vite, on crée des **registres de données** (data counters) ou des **registres d'adresse mémoire** (memory address registers) qui contiennent l'adresse mémoire de la donnée considérée.

Fonctionnement de l'unité centrale de traitement

Pour pouvoir être interprétée par la machine, chaque instruction doit être traduite dans une notation codée en binaire, octal ou hexadécimal. Le programme simple, vu plus haut, (V=10, S=V + 20) se transforme alors en une longue séquence de symboles (1 et 0) dans un langage accessible à l'ordinateur. Pour programmer de cette manière, il faut connaître le code de chaque instruction à réaliser.

Les instructions de notre programme peuvent se présenter ainsi :

1 / Charger, dans l'accumulateur, la valeur contenue dans la cellule mémoire dont l'adresse est symbolisée par V.
Code instruction = 0 0 1 1 1 0 1 0 =
$=(2^5+2^4+2^3+2^1)_{10}=(58)_{10}=(72)_8=(3A)_{16}$*.
La structure spécifiant l'adresse (V) sera expliquée plus loin.

2 / Additionner la valeur 20 (immédiate).
Code instruction = 1 1 0 0 0 1 1 0 =
$=(2^7+2^6+2^2+2^1)_{10}=(198)_{10}=(306)_8=(C6)_{16}$.

3 / Ranger, dans la cellule mémoire dont l'adresse est S, la valeur contenue dans l'accumulateur.
Code instruction = 0 0 1 1 0 0 1 0 =
$=(2^5+2^4+2^1)=(50)_{10}=(62)_8=(32)_{16}$.

On doit fournir, à toutes les instructions qui en feront usage, l'adresse qui servira à extraire ou à ranger les données.

Supposons que le programme parte de l'adresse mémoire $(100)_8$ ou $(40)_{16}$ et que les données soient stockées à partir des adresses $(300)_8$ ou $(C0)_{16}$. Les données (valeurs décimales 10 et 20) en notation binaire, octale et hexadécimale sont :

$(10)_{10} = 00001010 = (12)_8 = (A)_{16}$
$(20)_{10} = 00010100 = (24)_8 = (14)_{16}$.

* La notation $(58)_{10}$ fait référence au nombre 58 en base 10, de même que les notations $(72)_8$ et $(3A)_{16}$ font référence au même nombre exprimé en base octale ou hexadécimale.

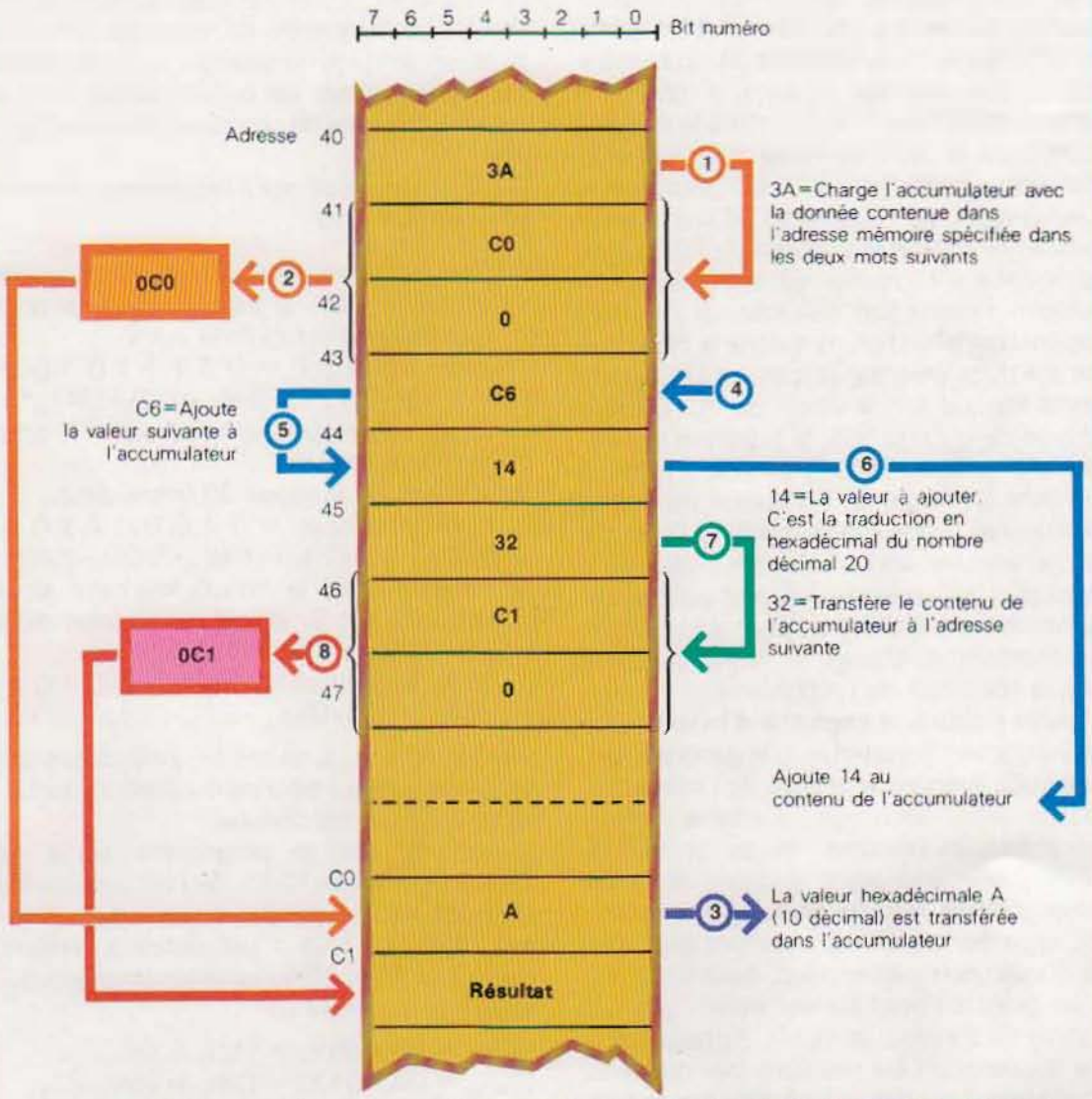
Ces valeurs devront être mises en mémoire dans les positions de mémoires adéquates pour pouvoir ensuite être utilisées par le programme. Ci-dessous, on a reporté le schéma complet de tout le programme et illustré les étapes de son exécution.
Pages 873 à 875, le déroulement du programme est suivi pas à pas. La lecture attentive des légendes permettra de se faire une première idée du déroulement effectif du traitement par un microprocesseur.

Afin de simplifier la description, on a supposé qu'il était possible d'indiquer les adresses sur 8 bits.

L'utilisation des codes hexadécimaux présente deux difficultés principales, même si on souhaite écrire un programme très simple :

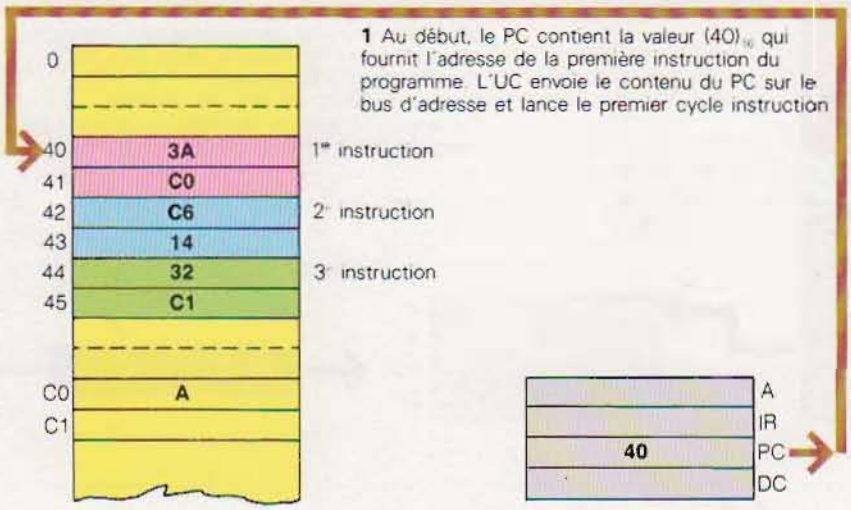
— les instructions (traduites en codes hexadécimaux) sont difficiles à identifier. La vérification du programme devient une

EXECUTION D'UN PROGRAMME ECRIT EN LANGAGE-MACHINE



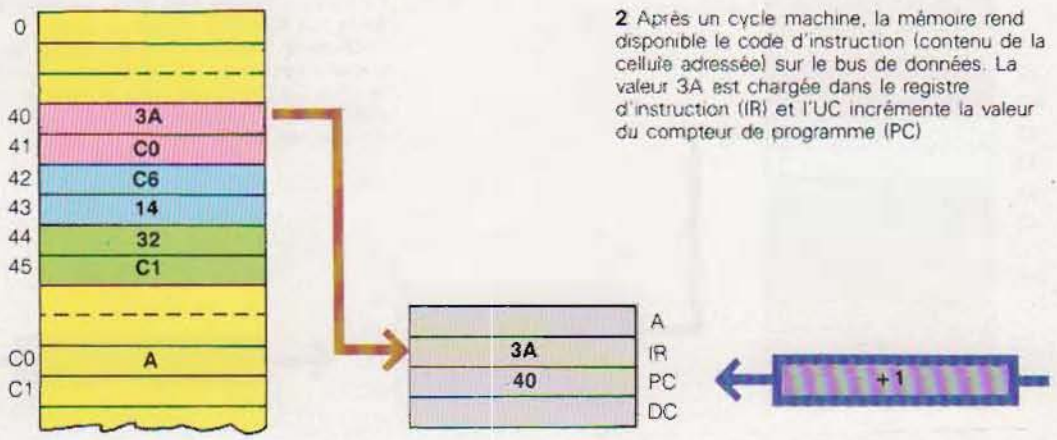
Ici, on a utilisé deux types d'instructions de chargement. Dans le premier, le code 3A (Charger l'accumulateur...), les données sont prélevées par l'intermédiaire d'un adressage vers n'importe quelle cellule mémoire (ici C0), même loin de celle contenant l'instruction. Le second (immédiat, code C6) utilise comme donnée (ici 14) le contenu de la mémoire suivante qui contient le code d'instruction.

VARIATIONS DU CONTENU DES REGISTRES EN PHASE D'EXECUTION

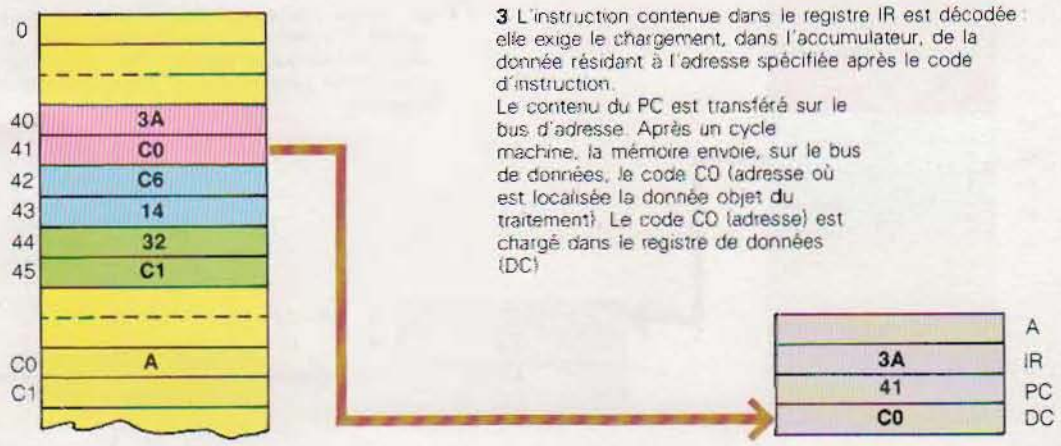


1 Au début, le PC contient la valeur $(40)_{16}$ qui fournit l'adresse de la première instruction du programme. L'UC envoie le contenu du PC sur le bus d'adresse et lance le premier cycle instruction

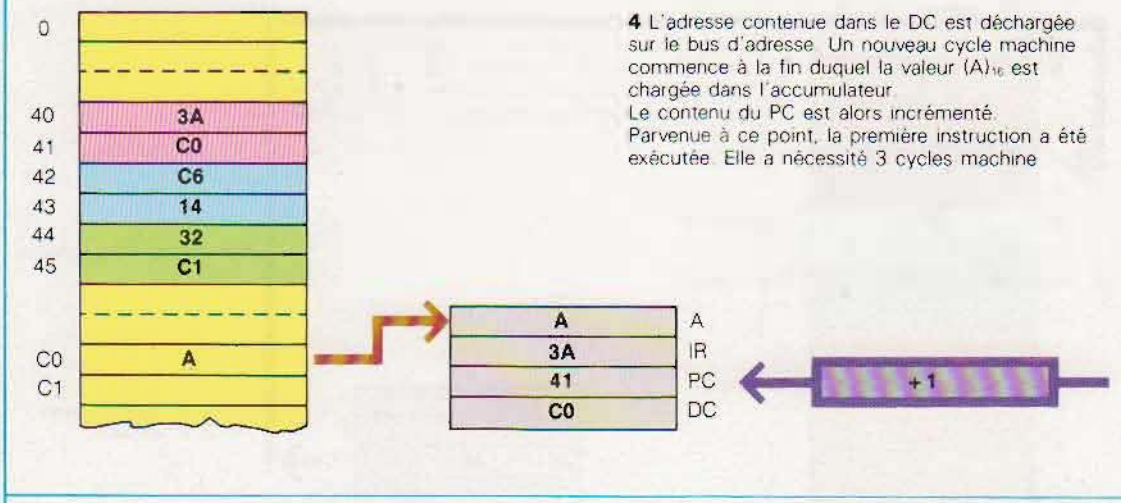
A = Accumulateur
IR = Registre d'instruction
PC = Compteur de programme
DC = Registre de données



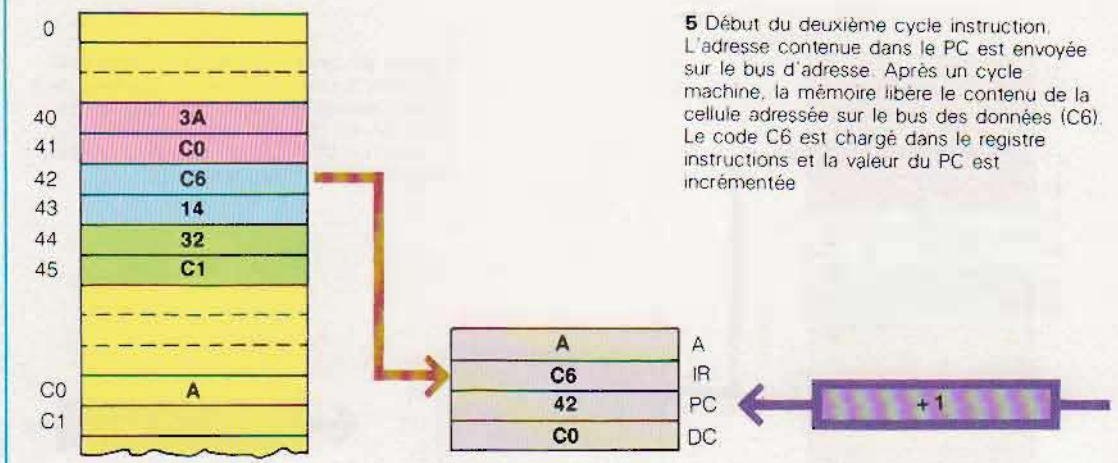
2 Après un cycle machine, la mémoire rend disponible le code d'instruction (contenu de la cellule adressée) sur le bus de données. La valeur 3A est chargée dans le registre d'instruction (IR) et l'UC incrémente la valeur du compteur de programme (PC)



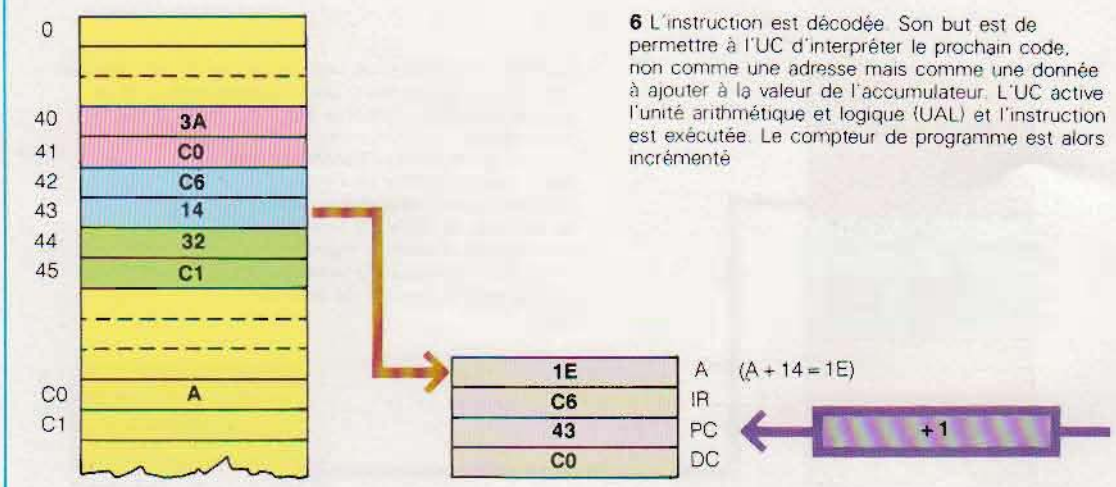
3 L'instruction contenue dans le registre IR est décodée : elle exige le chargement, dans l'accumulateur, de la donnée résidant à l'adresse spécifiée après le code d'instruction.
Le contenu du PC est transféré sur le bus d'adresse. Après un cycle machine, la mémoire envoie, sur le bus de données, le code C0 (adresse où est localisée la donnée objet du traitement). Le code C0 (adresse) est chargé dans le registre de données (DC)



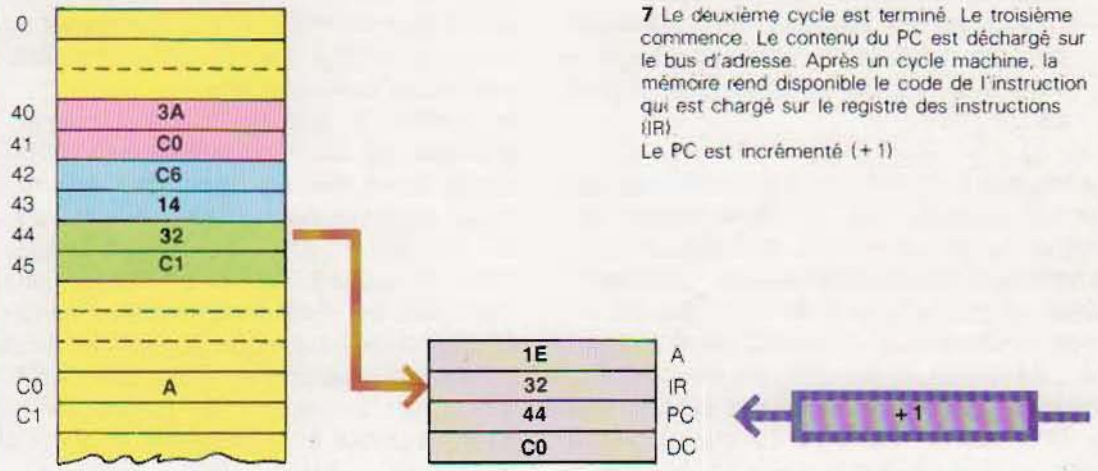
4 L'adresse contenue dans le DC est déchargée sur le bus d'adresse. Un nouveau cycle machine commence à la fin duquel la valeur $(A)_{16}$ est chargée dans l'accumulateur.
Le contenu du PC est alors incrémenté.
Parvenue à ce point, la première instruction a été exécutée. Elle a nécessité 3 cycles machine



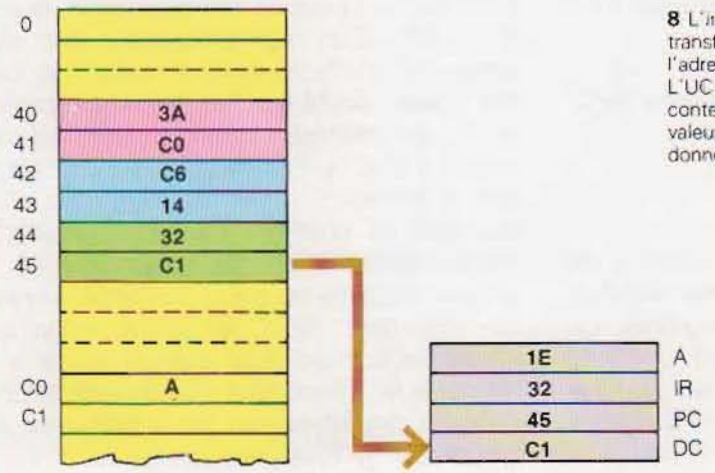
5 Début du deuxième cycle instruction.
L'adresse contenue dans le PC est envoyée sur le bus d'adresse. Après un cycle machine, la mémoire libère le contenu de la cellule adressée sur le bus des données (C6). Le code C6 est chargé dans le registre instructions et la valeur du PC est incrémentée



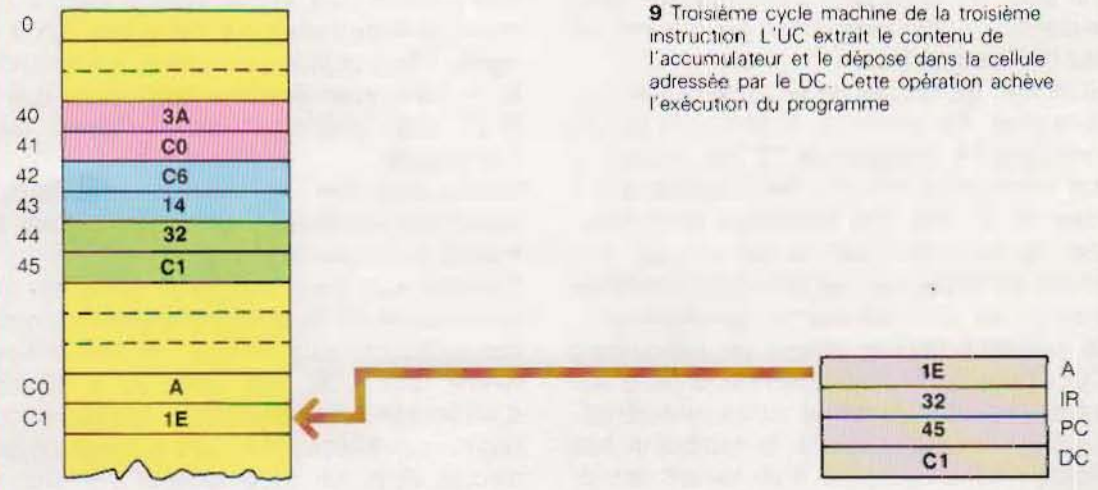
6 L'instruction est décodée. Son but est de permettre à l'UC d'interpréter le prochain code, non comme une adresse mais comme une donnée à ajouter à la valeur de l'accumulateur. L'UC active l'unité arithmétique et logique (UAL) et l'instruction est exécutée. Le compteur de programme est alors incrémenté



7 Le deuxième cycle est terminé. Le troisième commence. Le contenu du PC est déchargé sur le bus d'adresse. Après un cycle machine, la mémoire rend disponible le code de l'instruction qui est chargé sur le registre des instructions (IR).
Le PC est incrémenté (+ 1)



8 L'instruction est décodée : le système doit transférer le contenu de l'accumulateur à l'adresse suivant le code de l'instruction. L'UC décharge, sur le bus de données, le contenu du PC. Après un cycle machine, la valeur C1 sera transférée dans le registre de données (DC, pour Data Counter)



9 Troisième cycle machine de la troisième instruction. L'UC extrait le contenu de l'accumulateur et le dépose dans la cellule adressée par le DC. Cette opération achève l'exécution du programme

opération ardue, débouchant de toute façon sur des résultats incertains ;
— l'adressage des variables, qui n'utilise pas de nom symbolique, oblige le programmeur à se rappeler le contenu de chaque cellule mémoire.

Le langage d'assemblage a été créé pour pallier ces inconvénients. En effet, malgré son orientation plutôt machine, il réalise une programmation relativement simple. En Assembleur, les instructions sont identifiées par un code mnémonique, composé généralement de trois lettres, et les adresses peuvent être fournies à travers des étiquettes (labels). Ainsi, l'instruction : « charger l'accumulateur A avec... » (c'est-à-dire le code 3A) est représentée par le symbole LDA (LoaD A - charger A) qui devient, si on associe l'étiquette XY à la mémoire C0

LDA XY = charger A avec le contenu de la
(3A) (C0) mémoire XY

L'architecture interne du microprocesseur

Un microprocesseur comporte un bus de communication entre les différents organes physiques et un ensemble de registres. Un microprocesseur est généralement à mots de 8 bits si l'accumulateur est à 8 bits, à 16 si l'accumulateur est à 16... Il faut donc que les dimensions du bus interne — utilisé pour toutes les communications entre les registres et l'Unité Arithmétique et Logique (UAL), ainsi que pour les échanges d'informations avec l'extérieur — correspondent au nombre de bits de l'accumulateur.
En réalité, les choses ne se passent pas toujours ainsi. Par exemple, le Motorola 68000 comporte 16 registres de 32 bits, mais son bus interne (et externe) de données est à mots de 16 bits. Ces limitations sont imposées uniquement par la technologie employée qui ne permet pas de placer plus de 64 broches sur le boîtier du microprocesseur.
Ce boîtier à double rangée de connexions (DIP : Dual In-line Package) relie la puce aux autres circuits du système. Autre caractéristique des microprocesseurs : le nombre de bits circulant simultanément, à un instant donné, sur le bus d'adresses. Ce paramètre teste les performances du microprocesseur en indiquant le nombre de cellules mémoire adressables directement par l'UC. Autrement dit, il évalue le nombre d'accès mémoire réalisé au cours d'un cycle machine.
En général, un bus de 16 bits comporte 16 broches sur le microprocesseur, chacune d'elles étant reliée à une piste distincte du circuit imprimé. Avec 16 bits, on peut adresser directement $2^{16} - 1 = 65535$ cellules.
Dans la plupart des cas, chaque cellule contient 8 bits, il est donc possible d'adresser 65535 cellules à un octet ou bien la mémoire adressable peut être au maximum de 64 K-octets (on sait que 1 Ko [prononcer kilo-octet] équivaut à 1024 octets et que 1 Mo [prononcer Méga-octet] équivaut à 1 048 576 octets).
C'est surtout pour ne pas augmenter le prix de revient d'un microprocesseur que plusieurs constructeurs n'ont, en catalogue, que des boîtiers de 16 broches pouvant dépasser le nombre habituel d'adresses. Dans ce cas, au cours d'un cycle machine, on envoie sur le bus d'adresse (formé de 8 pistes, par exemple) la première moitié de l'adresse (généralement la partie supérieure) et la deuxième partie au cours du cycle suivant. On maintient ainsi la valeur maximale d'adressage, mais pour chaque accès à la mémoire on utilise deux fois plus de temps, ce qui a des incidences sur la puissance de l'ensemble du système.
Ainsi, le microprocesseur Intel 8088 a une architecture interne à 16 bits (pour les registres comme pour le bus interne) mais communique avec l'extérieur par 8 bits. Chez le même constructeur on trouve un autre modèle, le 8086, avec la même architecture que le 8088 mais disposant de 20 lignes pour l'adressage.
Notons enfin que le prix de revient de la mémoire tend à baisser tandis que les besoins en espace mémoire augmentent.
C'est la raison pour laquelle de nombreux microprocesseurs de la nouvelle génération ont des architectures à 16 bits, certains en sont même déjà à 32 bits, avec 20 à 32 bits d'adressage. La multiplication des bus et des registres améliore également la précision des calculs. Avec un accumulateur à 8 bits, le nombre le plus grand que l'on puisse repré-

senter est 256. Avec un accumulateur à 16 bits, en revanche, on arrive à 65536, et s'il est à 32 bits à plus de quatre milliards. La mémoire adressable va du Méga-octet (20 bits d'adressage pour l'Intel 8086) à 16 Mo (24 bits d'adressage pour le Motorola 68000), 500 Mo (28 bits d'adressage pour le Hewlett-Packard 9000). Dans des cas semblables, même si la mémoire est toujours mesurée en octets, chaque cellule est constituée d'au moins le même nombre d'octets que ceux de l'accumulateur. Par la suite, nous travaillerons avec un microprocesseur disposant d'une architecture à 8 bits (accumulateur et bus interne communiquant avec l'extérieur avec 16 bits [PC, DC]. L'Intel 8080, le Motorola MC 6800, le Zilog Z80 et le Rockwell 6502 appartiennent à cette catégorie.

En résumé :
Les registres sont des partitions de mémoire à l'intérieur de l'UC. Des données et des instructions sont momentanément stockées en mémoire, dans un registre, tant qu'un bus ou un autre dispositif n'est pas prêt à les recevoir. L'UC puise des données dans les registres sans passer par la mémoire centrale. Les registres et les bus internes ont en général la même taille.

Le prix de revient des branchements internes limite le nombre des registres dans l'UC. Mais la tendance actuelle des microprocesseurs toujours plus rapides amène les constructeurs à utiliser un plus grand nombre de registres. Dans son UC, le microprocesseur Hewlett-Packard 9000 ne contient pas moins de 28 registres de 32 bits chacun.

L'unité arithmétique et logique (UAL)

Outre les registres, l'unité centrale renferme des dispositifs réalisant les opérations de calcul. Il s'agit de l'Unité Arithmétique et Logique (UAL).
Les dispositifs d'une UAL permettent au microprocesseur d'effectuer les opérations suivantes (voir schéma page 878, en haut) :
— addition
— complément d'un mot
— soustraction (parfois)
— opérations logiques ET, OU, NON (AND, OR, NOT)
— décalage (SHIFT) à gauche ou à droite d'un ou plusieurs bits.

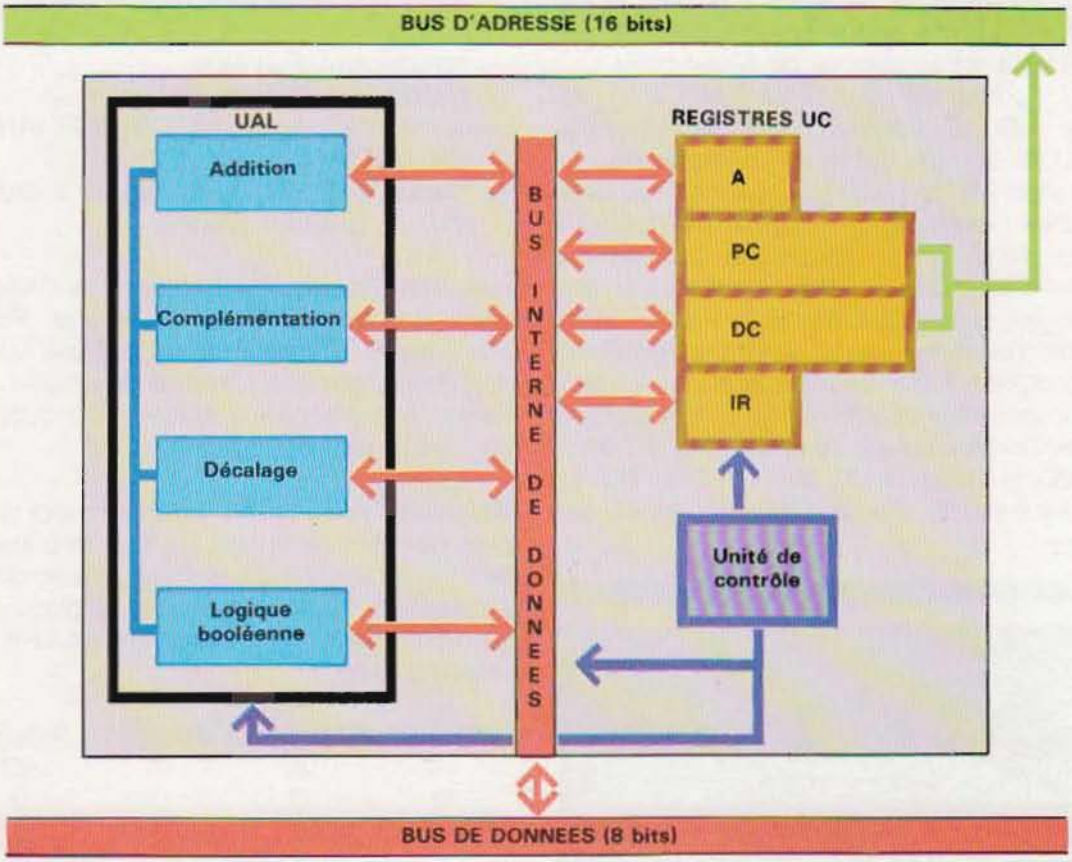
Bien évidemment, ces opérations ne s'effectuent que sur des nombres binaires. Pour comprendre le fonctionnement d'une UAL, une description plus détaillée des règles de l'arithmétique binaire, ainsi que des circuits et dispositifs, est nécessaire.

Addition binaire. Ces règles sont très proches de celles de la base 10, tout en présentant l'avantage d'être beaucoup plus simples. L'opération fondamentale est l'addition de deux bits ; deux bits à additionner peuvent se présenter ainsi :

Cumulande (A)	Cumulateur (B)	Somme (S)	Retenue (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Le symbole A indique le premier terme (cumulande) de la somme et B le second (cumulateur), S indique la somme des deux bits et C la retenue (en anglais : Carry). Il y a toujours un **circuit additionneur** (adder) dans chaque UAL ; il est capable de calculer la somme de deux nombres binaires d'une longueur égale à 8 ou 16 bits, selon le type de microprocesseur.
Avant de passer à la description d'un additionneur de nombres de 8 ou 16 bits, examinons d'abord le processus dans un circuit à deux bits.
La fonction S (somme) est réalisée par le circuit logique qui figure au bas de la page 878 où la retenue C est assurée par un simple circuit ET (AND) comme on peut le vérifier avec la table de vérité.
Un tel circuit est appelé demi-additionneur (Half-adder). On peut alors imaginer un circuit

STRUCTURE INTERNE D'UN MICROPROCESSEUR



SCHEMA D'UN ADDITIONNEUR A DEUX ENTREES

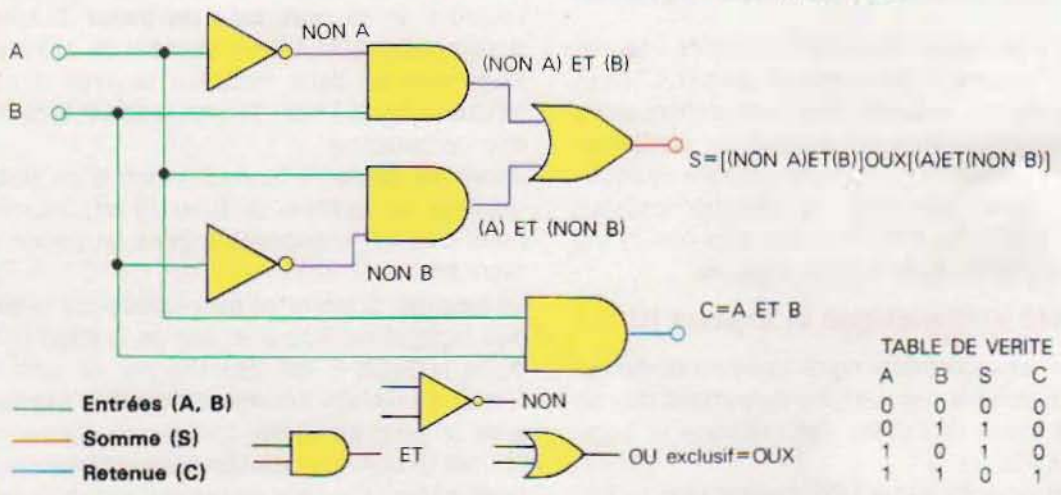


TABLE DE VERITE

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

ADDITIONNEUR A HUIT BITS

HA = Half-adder (demi-additionneur)
A = Premier terme (cumulande)
B = Deuxième terme (cumulateur)
S = Somme
C = Retenue (carry)

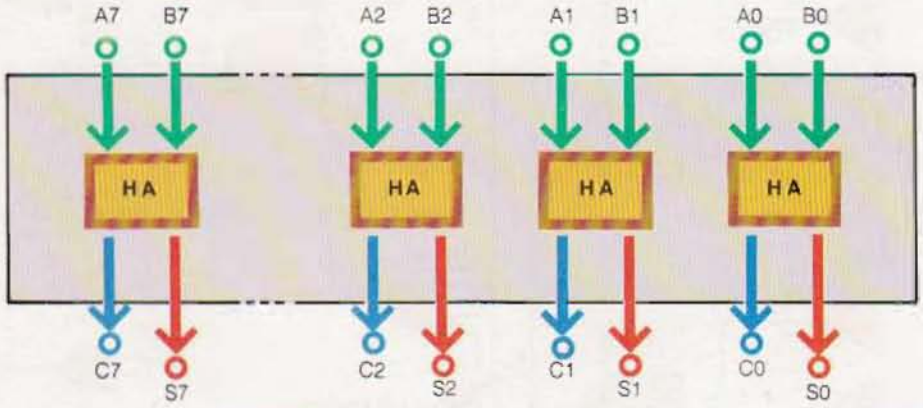
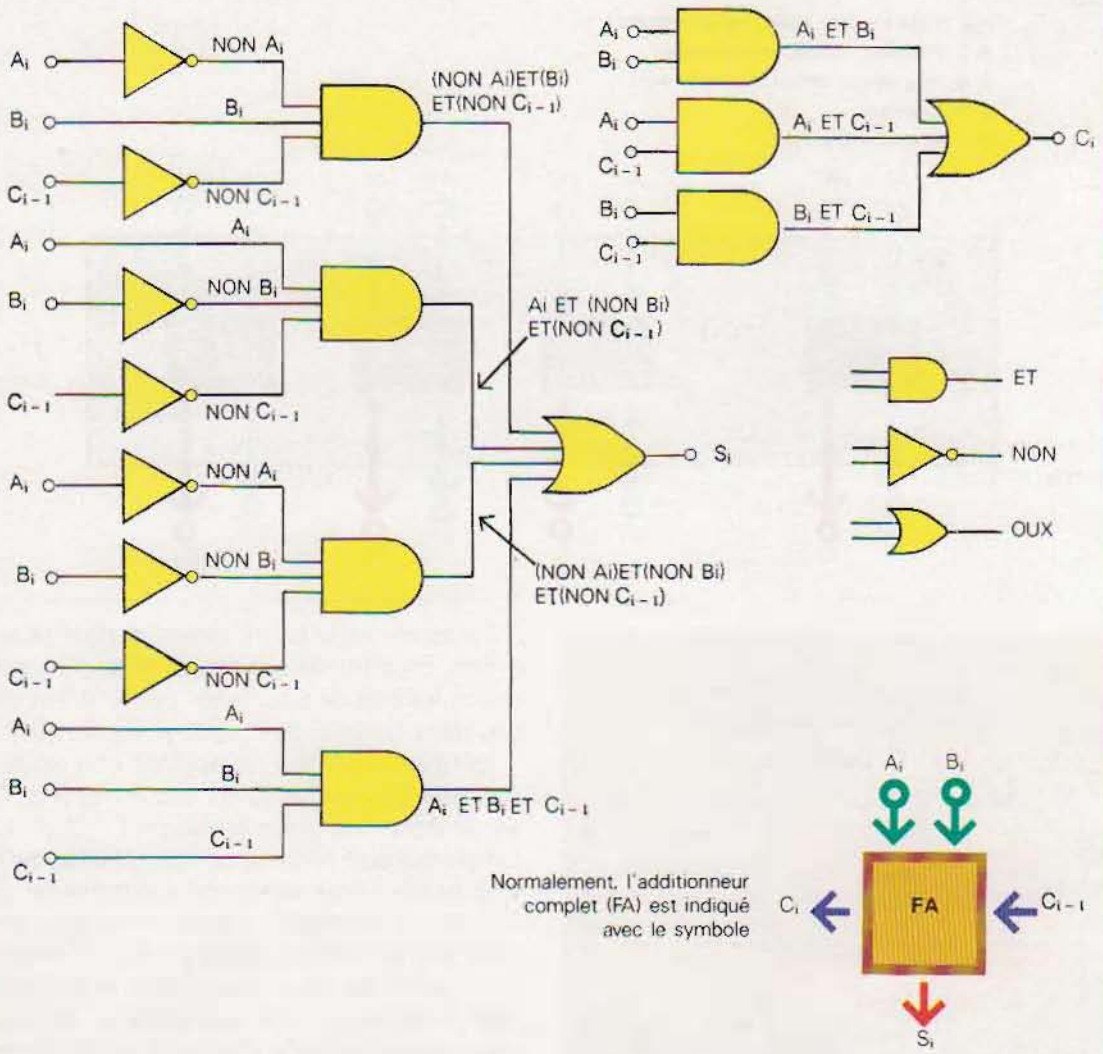


TABLE DE VERITE D'UN ADDITIONNEUR COMPLET

ENTREES			SORTIES	
Retenue précédente Ci-1	Cumulande Ai	Cumulateur Bi	Somme Si	Retenue Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

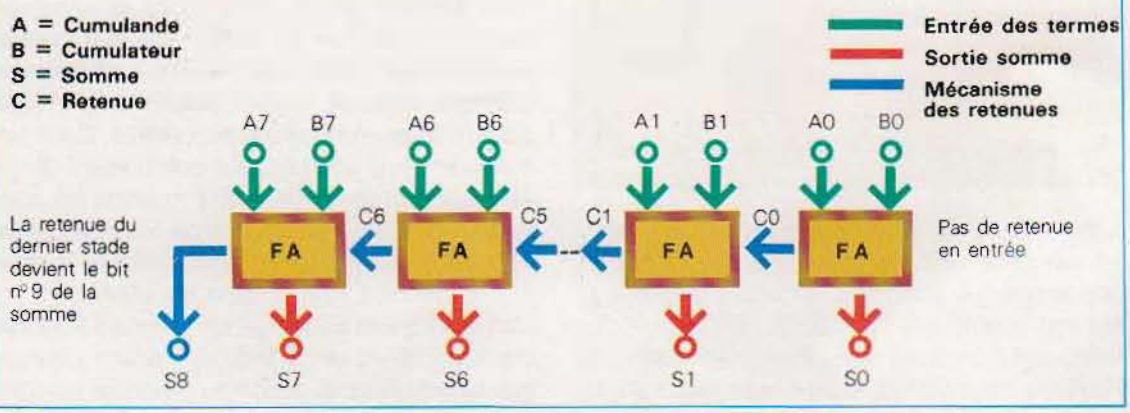
à 8 entrées avec 8 demi-additionneurs placés en parallèle comme ci-dessus, où le circuit additionneur à 2 entrées (exemple précédent) est représenté par les lettres HA.
Cette solution n'est cependant pas viable car l'additionneur à 2 entrées ne tient pas compte d'éventuels reports provenant d'états plus à droite. En effet, le schéma ci-dessus montre que si les bits de plus faible poids (A0 et B0) des deux termes de la somme équivalaient à 1, la retenue C0 = 1 ne pourrait être ajoutée au stade suivant (avec les bits A1 et B1). Ce serait donc une grave omission.
On utilise l'additionneur complet (Full adder) à trois entrées pour surmonter ce problème. Ce dernier est en mesure d'additionner deux bits ainsi que la retenue générée par la somme présédente. La table de vérité d'un additionneur complet — voir ci-contre — est plus compliquée que celle d'un demi-additionneur, de même que la réalisation pratique de l'additionneur complet est plus complexe (voir page 880, en haut).
Une des principales différences provient du type particulier de ET (AND) qui doit prévoir trois entrées : deux pour les bits à additionner (comme dans le circuit précédent) et une pour la retenue du stade précédent. Dans notre exemple, l'additionneur complet est divisé en deux parties : la première exécute les opérations de calcul de la somme (colonne Si de la table de vérité), la seconde calcule la retenue (colonne Ci de la table de vérité).
On peut construire un circuit, pour additionner deux nombres de 8 bits, en tenant compte des retenues et en utilisant un certain nombre

FONCTIONNEMENT D'UN ADDITIONNEUR COMPLET



FONCTIONNEMENT D'UN ADDITIONNEUR PARALLELE A 8 ENTREES

A = Cumulande
B = Cumulateur
S = Somme
C = Retenue



d'additionneurs complets : il s'agit de l'additionneur parallèle, schématisé au bas de la page 880. Il se compose d'autant d'additionneurs complets en parallèle qu'il y a de bits à additionner. Une fois créée l'éventuelle retenue C0 à partir de la somme des deux bits de plus faible poids (A0 et B0), il passe au stade suivant où elle est ajoutée aux bits A1 et B1. Il en résulte la somme S1 et la retenue C1 qui est prise en compte au stade suivant.
Le processus se répète jusqu'au stade 8 où la retenue C7 devient le bit de plus fort poids (S8) de la somme. De cette façon, le résultat de l'addition de 8 bits peut être un nombre de 9 bits.

Soustraction binaire. On peut utiliser le même processus pour la construction d'un soustracteur binaire. Ci-dessous la table de vérité du circuit où D indique la différence entre les deux bits (A et B) et C la retenue négative. Lorsque A=0 et B=1, il faut retenir 1 pour calculer la différence, comme pour une soustraction décimale.
Cependant, dans les UAL des microprocesseurs, il n'y a normalement pas de circuit soustracteur dans la mesure où on peut facilement réaliser cette opération avec les circuits existant déjà dans l'UAL : l'additionneur et le circuit servant à complémenter. Ceci provoque bien sûr une dégradation du système et de ses performances en termes de vitesse, mais rend la conception de l'UAL plus simple et moins coûteuse.

Complémentation. L'arithmétique décimale illustre d'une manière simple comment effectuer une soustraction avec une complémentation et une addition. Soustraire un nombre décimal à un autre équivaut à ajouter, à ce dernier, le complément à 10 du plus petit nombre.
Le complément à 10 d'un nombre s'obtient en retirant à 10 la valeur de ce nombre. Ainsi, le complément à dix de 3 est 7 (10 — 3), puisqu'il suffit, pour obtenir le nombre 10, d'ajouter 3 à 7.
Pour effectuer une soustraction avec la méthode du complément, il faut tout d'abord calculer le complément à 10 du plus petit nombre et additionner ensuite le plus grand nombre au résultat en laissant de côté une éventuelle retenue finale.
Considérons la soustraction :

$$8 - 3 = 5$$

7 est le complément à 10 de 3, on a donc :

$$8 + 7 = 15$$

Retenue : 1 ; résultat : 5

On peut se demander pourquoi calculer la soustraction avec l'addition d'un nombre complémenté alors que pour trouver le complément d'un nombre il faut quand même faire une soustraction.
La réponse est simple : l'équivalent binaire du complément à 10 est le complément à 2, très simple à calculer. Le complément à deux d'un nombre binaire s'obtient en remplaçant le symbole 0 par le symbole 1 et vice versa, et en ajoutant 1 au résultat binaire.
L'opération de substitution du symbole 0 par le symbole 1 et vice versa est dite complément à 1.
Ainsi, le complément à 1 du nombre 10101110 est 01010001. Le complément à 2 du même nombre sera :

```
0 1 0 1 0 0 0 1 + (complément à 1)
              1 =
-----------------
0 1 0 1 0 0 1 0   (complément à 2)
```

Voyons maintenant comment effectuer une soustraction binaire au moyen de la complémentation. Soit la soustraction :

TABLE DE VERITE D'UN SOUSTRACTEUR BINAIRE

Le plus grand nombre A	Le plus petit nombre B	Différence D	Retenue (négative) C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Plus grand nombre		Plus petit nombre
1 1 1 0 1 0 1 0	—	1 0 1 0 1 1 1 0

Le complément à 2 du plus petit nombre a déjà été trouvé ; il est égal à 0 1 0 1 0 0 1 0, on a donc :

```
  1 1 1 0 1 0 1 0 +
  0 1 0 1 0 0 1 0 =
  ---------------
1 0 0 1 1 1 1 0 0   Résultat
Bit de retenue à négliger
```

Comme dans l'opération décimale, on néglige le bit de retenue. Vérifions l'exactitude de l'opération en transformant les nombres binaires en décimaux

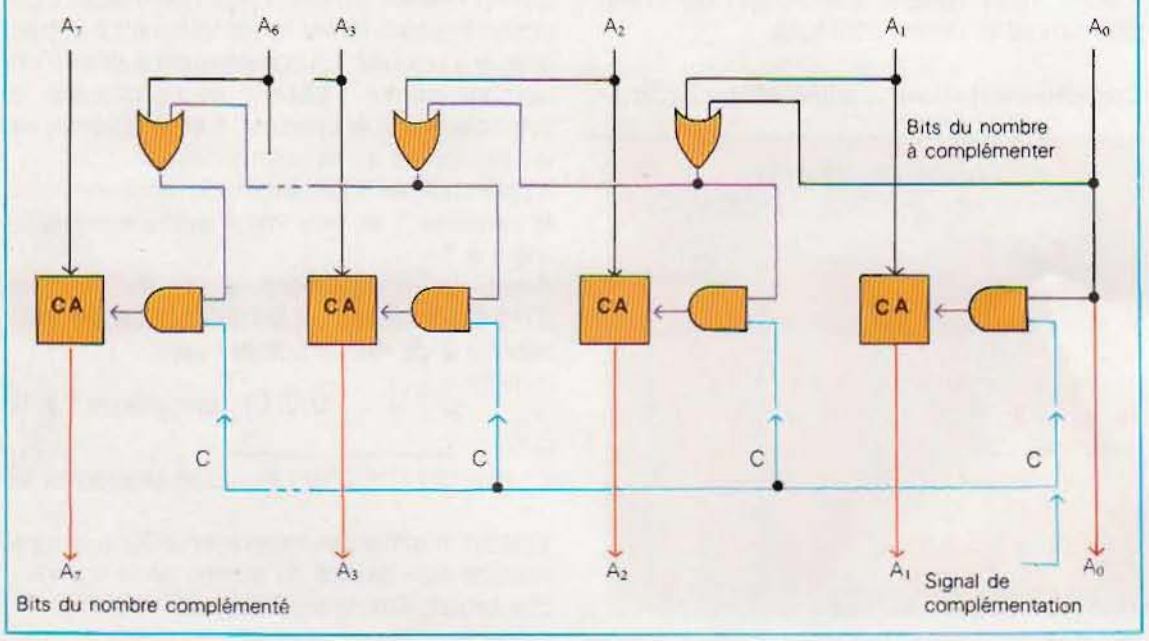
	Binaire	Décimal
Plus grand nombre	1 1 1 0 1 0 1 0 —	234 —
Plus petit nombre	1 0 1 0 1 1 1 0 =	174 =
Résultat	0 0 1 1 1 1 0 0	60

Dans les circuits de l'UAL, l'opération de complémentation à deux est faite différemment. On prend le nombre à compléter, par exemple 1 0 1 0 1 1 1 0, et on le lit de droite à gauche. Lorsqu'on rencontre le premier 1, on le laisse inchangé alors que les symboles suivants sont intervertis, 1 prend la place de 0 et vice versa : 0 1 0 1 0 0 1 0.

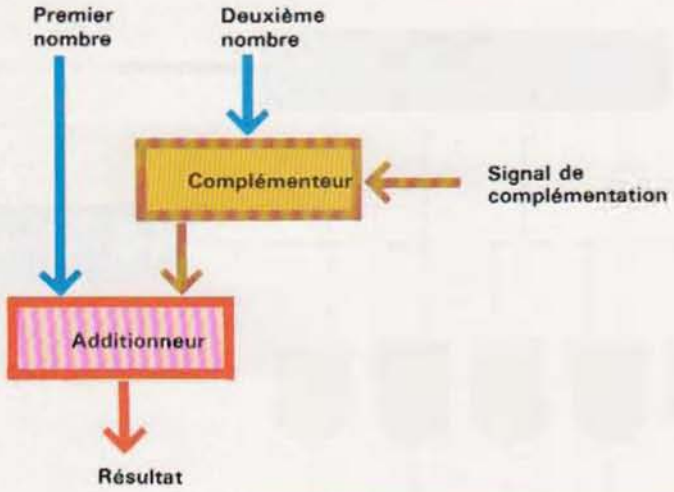
Sur cette page, on a représenté un circuit calculant le complément à 2 d'un nombre de 8 bits. Le symbole CA indique le circuit qui se charge de l'inversion du bit du nombre binaire à complémenter. Cette opération est subordonnée à la présence du signal C. S'il est égal à 1, le nombre binaire est complémenté, sinon il est inchangé. Ce signal est utilisé pour faire fonctionner le circuit qui réalise simultanément l'addition et la soustraction (additionneur-soustracteur). Ci-contre, en haut, on peut voir un schéma des opérations d'addition et de soustraction entre deux facteurs, selon l'état (1,0) de la ligne C. Quand l'addition est demandée, le signal de complémentation (C) est égal à 0, le deuxième terme reste inchangé et on calcule une somme. Si on souhaite une soustraction, le signal est égal à 1 et on ajoute au premier nombre le complément à 2 du second.

Multiplication binaire. Les règles de la multiplication binaire sont celles de la multiplication décimale, comme on peut le voir sur le schéma de la page 883.
Chaque produit partiel est égal au multipli-

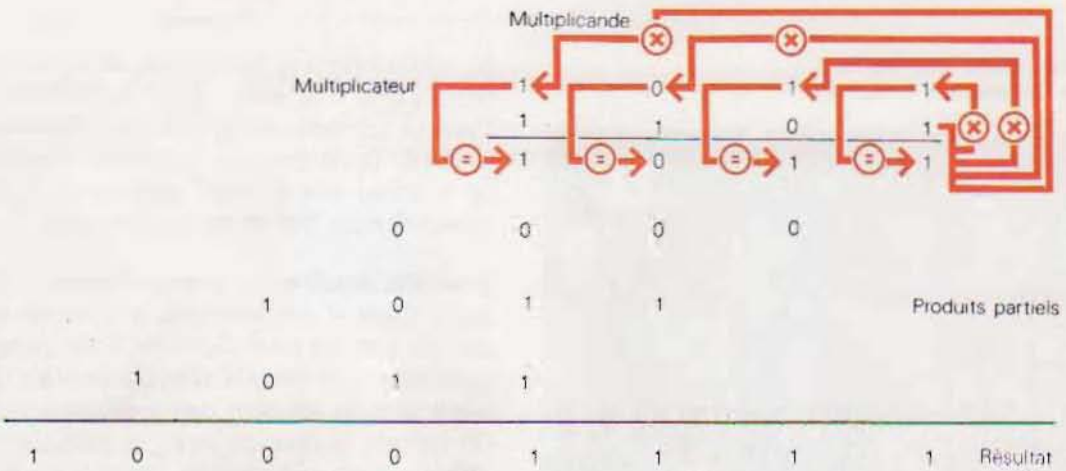
CIRCUIT PARALLELE DE COMPLEMENTATION A 2



FONCTIONNEMENT D'UN ADDITIONNEUR-SOUSTRACTEUR

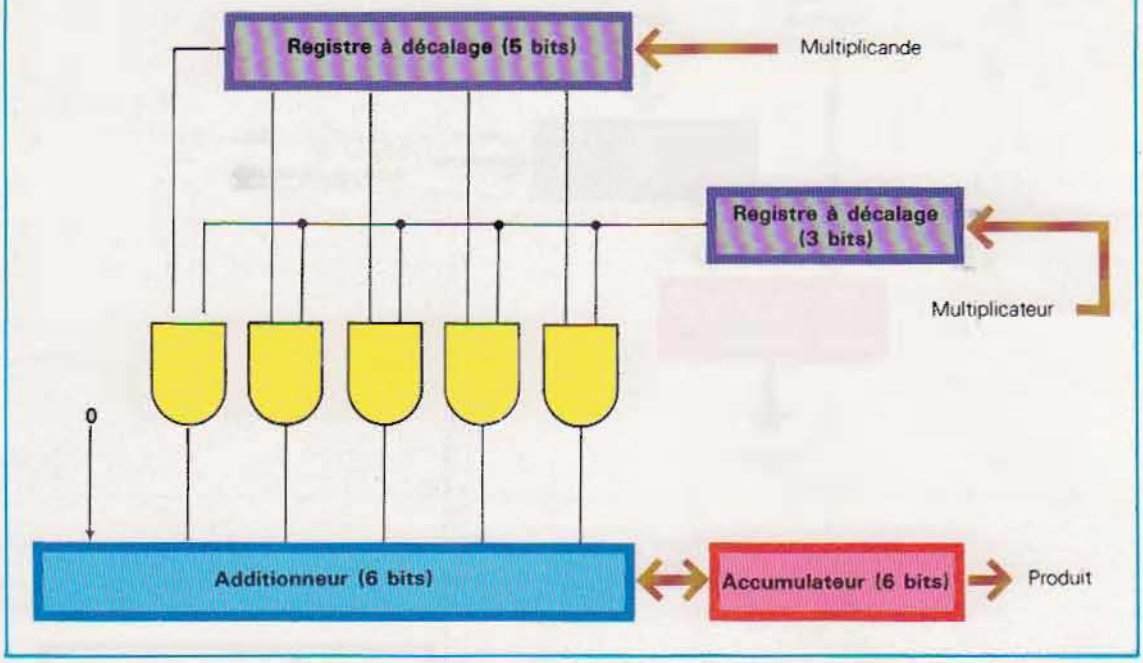


MULTIPLICATION BINAIRE



cande si le multiplicateur est égal à 1, ou à 0 si le bit relatif au multiplicateur est 0.
Les produits partiels doivent être défalqués et additionnés. Notons que le produit est composé d'un nombre de bits égal à la somme des bits des deux termes. Ici, le produit a une longueur de 8 bits car multiplicande et multiplicateur ont chacun 4 bits. Page 884, figure un circuit capable de multiplier deux facteurs de 3 bits chacun. Il se compose de deux circuits appelés **registres à décalage** (Shift registers) de deux additionneurs à 6 bits et d'une série de 5 circuits ET.
Normalement, le circuit multiplicateur ne figure pas dans l'UAL des microprocesseurs utilisés dans les ordinateurs personnels pour des raisons de simplicité et de coût. La multiplication est alors effectuée selon la méthode des sommes répétées.
C'est ainsi que l'on effectue la multiplication

CIRCUIT MULTIPLICATEUR A TROIS BITS



Une phase des travaux de connexion externe.



A. Tracchia/AFE

en additionnant le multiplicande un nombre de fois égal à la valeur du multiplicateur. En d'autres termes, multiplier 8 par 3 équivaut à 8+8+8. Dans d'autres types de machines, on a prévu des circuits semblables à celui présenté mais beaucoup plus rapides.

Division binaire. La division binaire s'effectue à l'aide d'une méthode proche de celle utilisée dans la multiplication. Il est possible de réaliser des circuits spécialisés mais, normalement, ils ne sont pas intégrés à l'UAL. On calcule la division avec la méthode des soustractions successives jusqu'à ce qu'on arrive au nombre de chiffres voulu après la virgule.

Outre ces opérations arithmétiques, l'unité arithmétique et logique effectue des opérations booléennes et des opérations de décalage. Les opérations booléennes sont des opérations logiques sur des données binaires selon les règles arithmétiques de Boole*. Les

* George Boole (1815-1864) est considéré comme le père de l'arithmétique des ordinateurs. Il fut le premier à formuler un ensemble de règles de type algébrique permettant d'effectuer les quatre opérations à l'aide des fonctions logiques ET, OU, NON.

opérations de décalage sont des opérations de déplacement à droite ou à gauche des bits composant un mot binaire.
Ces opérations sont très employées dans la programmation en langage d'assemblage car elles permettent de lancer simplement des opérations d'analyse et de comparaison de tous les bits d'un mot.

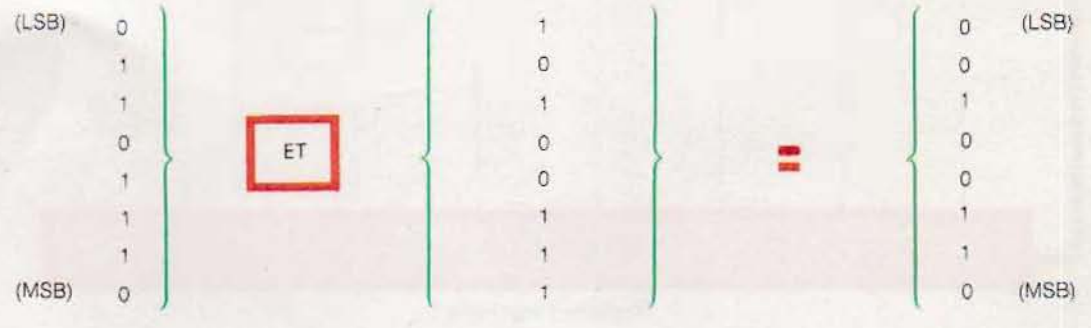
Opérations booléennes. ET, OU, NON et OUX (OU exclusif) sont des opérations logiques couramment effectuées par l'UAL.
Elles ont déjà été décrites (page 72) et on a vu comment il est possible de les représenter d'une manière synthétique à l'aide de la table de vérité des fonctions, ET, OU, OUX, NON.
Supposons que l'UAL ait une longueur de mot de 8 bits, l'opération ET entre deux mots binaires se fait en appliquant ET entre les bits correspondant aux deux mots. Ainsi, l'opération ET entre deux mots binaires 01110110 et 11100101 s'obtient, comme on peut le voir sur le deuxième schéma ci-dessous. A noter que les bits de plus faible poids (LSB, c'est-à-dire les symboles binaires situés plus à droite) ont été écrits sur la première ligne et les bits de plus fort poids (MSB, ceux situés plus à gauche) sur la dernière. D'un point de vue pratique, le circuit AND pour deux mots binaires à 8 bits est montré page 886.

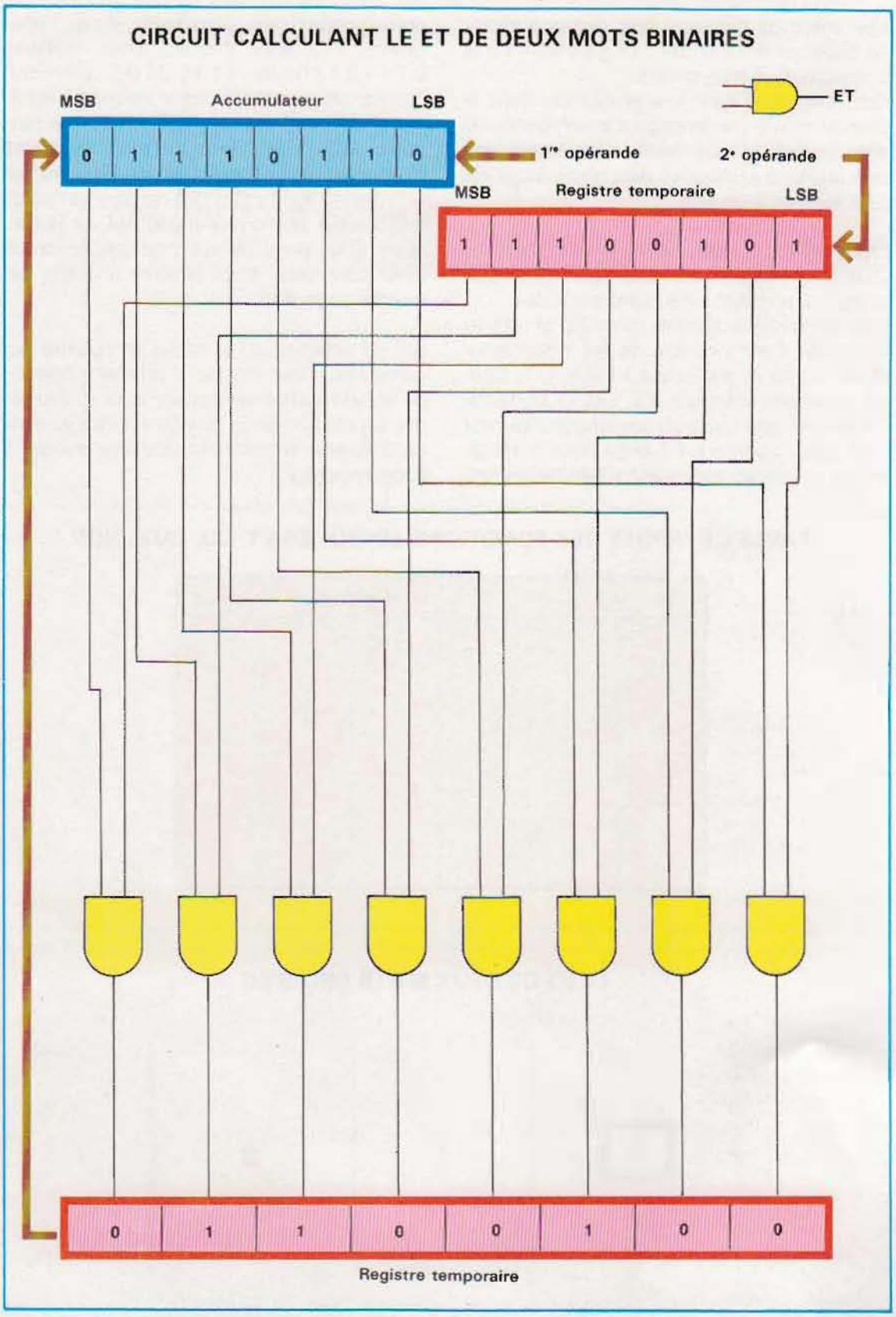
Sur ce schéma, on a utilisé un registre accumulateur pour stocker la première opérande et un registre temporaire pour la deuxième. Le résultat de ET (AND) est chargé dans un troisième registre avant d'être envoyé à l'accumulateur.

TABLE DE VERITE DES FONCTIONS LOGIQUES ET, OU, OUX, NON

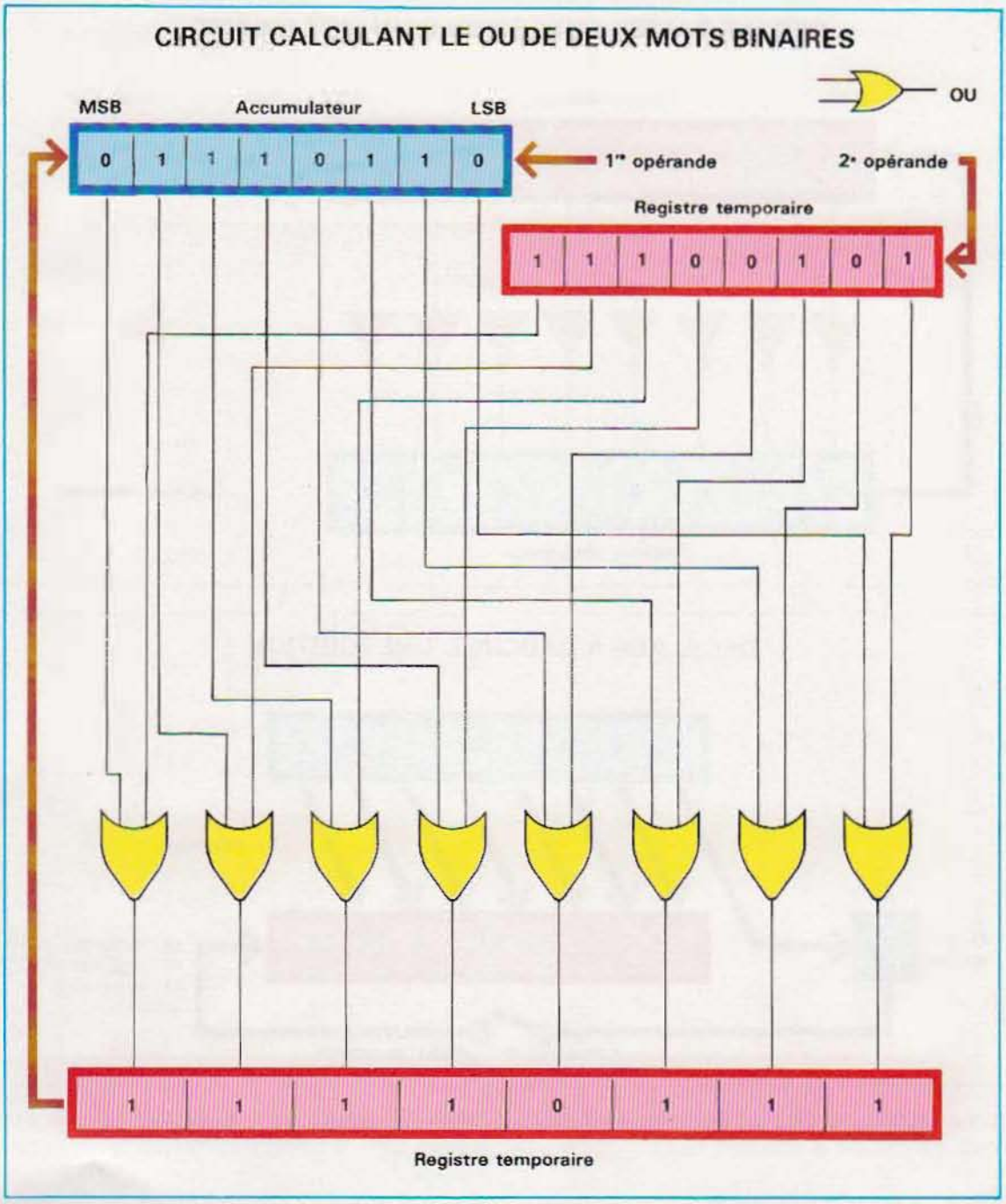
A	B	ET	OU	OUX	NON A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

LE ET DE DEUX MOTS BINAIRES




CIRCUIT CALCULANT LE ET DE DEUX MOTS BINAIRES
MSB
Accumulateur
LSB
ET
0 1 1 1 0 1 1 0
1er opérande
2e opérande
MSB
Registre temporaire
LSB
1 1 1 0 0 1 0 1
0 1 1 0 0 1 0 0
Registre temporaire

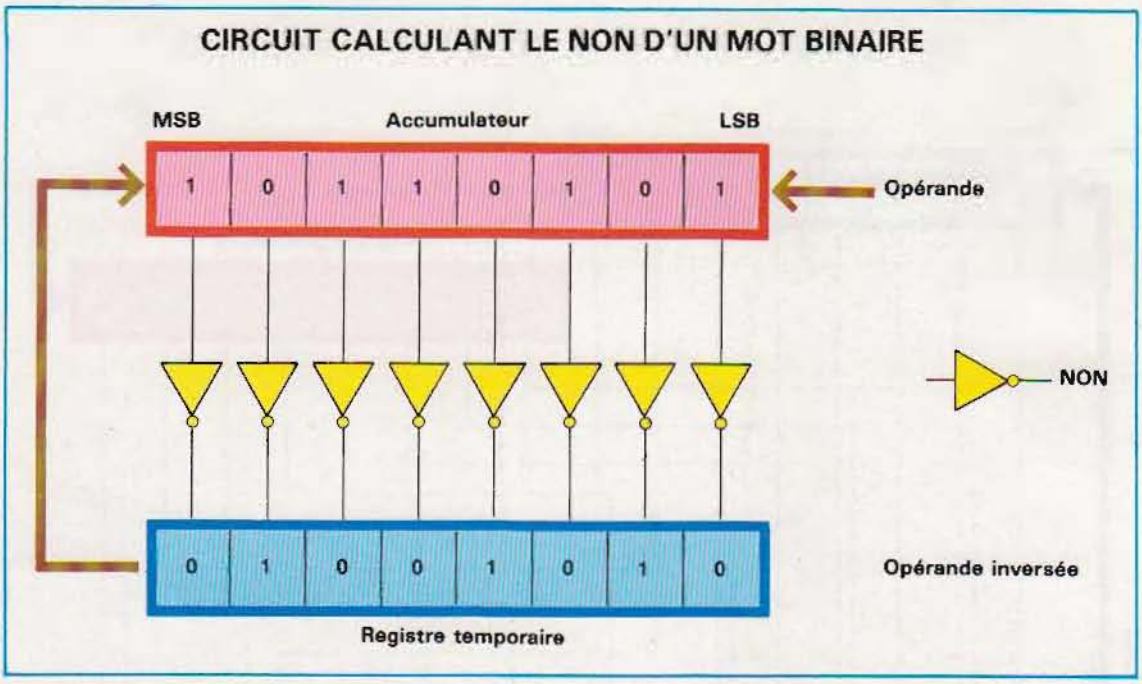
CIRCUIT CALCULANT LE OU DE DEUX MOTS BINAIRES

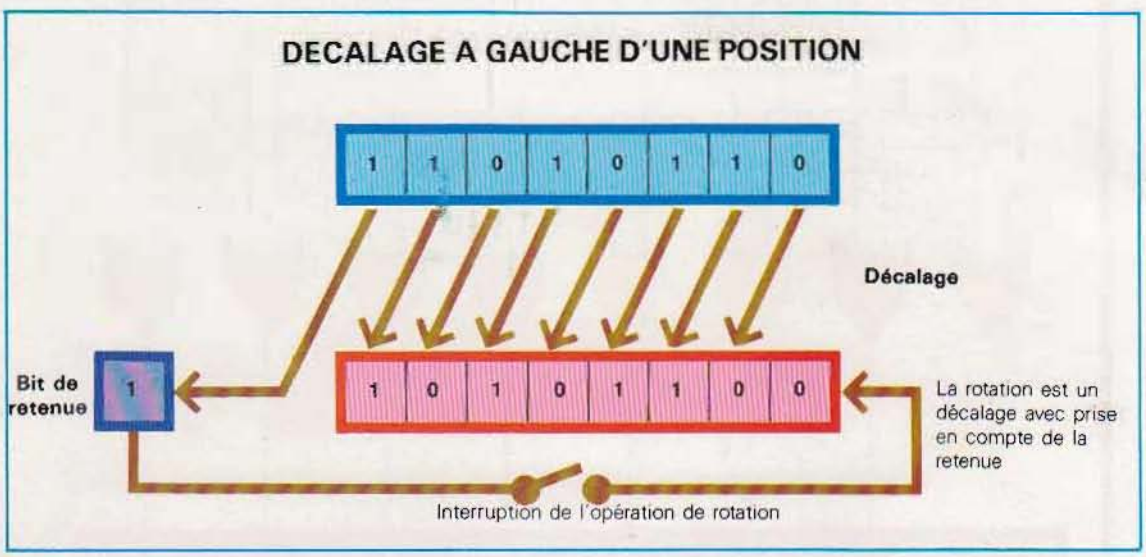


L'opération OU peut être réalisée comme ci-dessus, où les opérateurs ET ont été remplacés par des opérateurs OU.
Il n'est pas nécessaire de disposer de tous les circuits correspondant aux fonctions logiques pour les calculer. En principe, n'ont été prévus que deux sortes de circuits : soit le couple NON et ET, soit le couple NON et OU, qui suffisent à la réalisation de toute fonction booléenne. Par exemple, la fonction booléenne A OU B équivaut à :

NON [(NON A) ET (NON B)]

Donc, pour réaliser le OU de deux mots binaires A et B, il suffit d'inverser les deux mots (NONA, NONB), de calculer le ET et ensuite d'inverser le résultat.



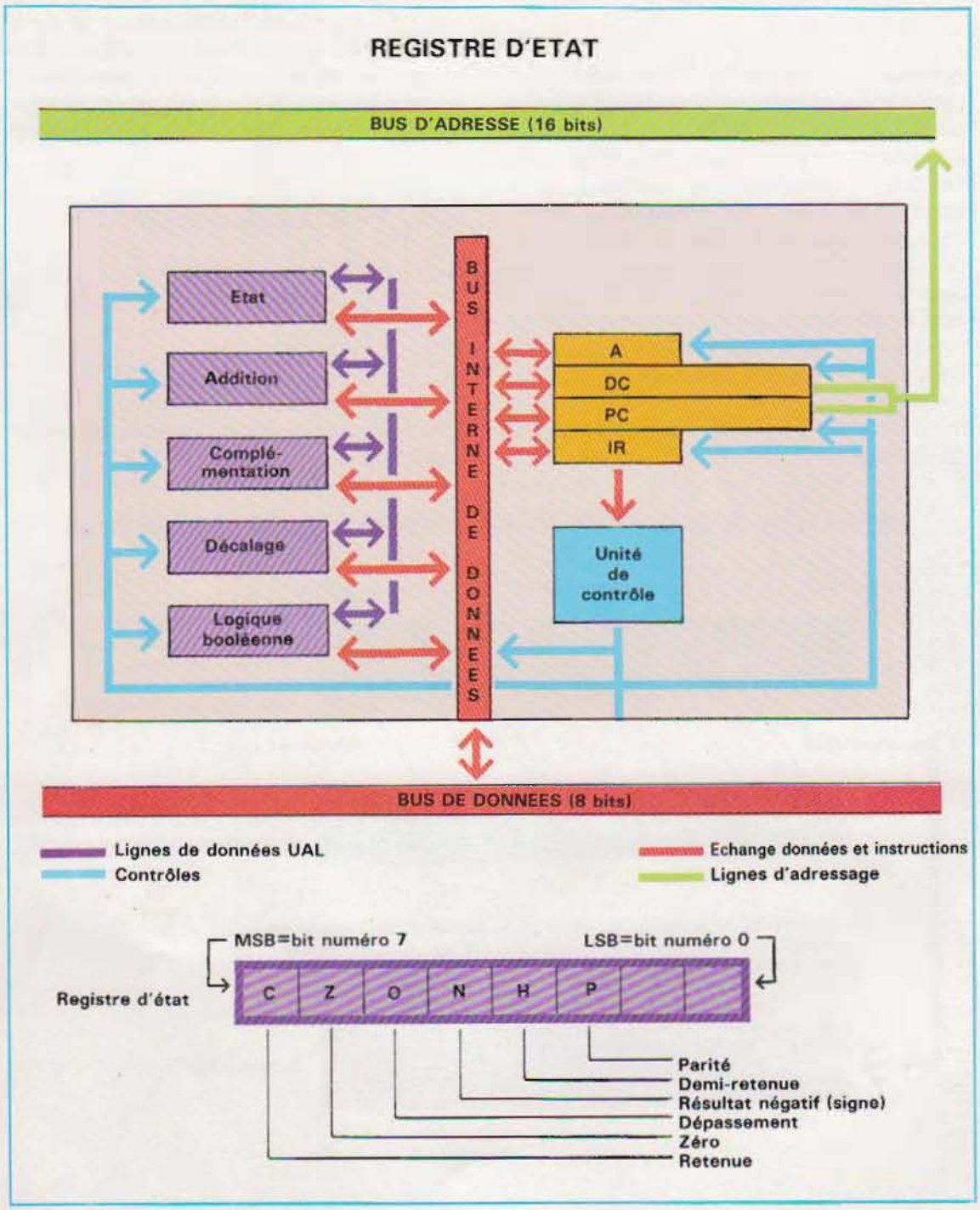


On a reporté, ci-dessus, le schéma d'un circuit qui calcule la fonction NON.

Opérations de décalage. Dans ces opérations, les symboles (0, 1) du mot binaire se déplacent à droite ou à gauche. On trouve, ci-dessus, le schéma de déplacement d'un mot binaire vers la gauche.
Généralement (mais pas toujours), le bit de plus fort poids d'un mot est transféré dans un bit de retenue, après le déplacement.
De plus, le mot binaire peut être modifié par des opérations de rotation. Dans ce cas, le chiffre contenu dans le bit de retenue est inséré dans le bit du mot binaire après le décalage.
On peut effectuer ces opérations de décalage et de rotation un nombre de fois préfixé, tant vers la gauche que vers la droite.
Décaler d'une place vers la gauche équivaut à multiplier par 2, décaler d'une place vers la droite équivaut à diviser par 2. D'une façon générale, le décalage de n positions vers la gauche équivaut à la multiplication par 2^n, tandis que décaler de n positions vers la droite signifie diviser par 2^n.

Le registre d'état

Dans l'Unité Arithmétique Logique (UAL), l'exécution des opérations est associée à l'emploi d'indicateurs (flags) qui signalent si une certaine condition est validée ou non. Les indicateurs sont regroupés dans un registre de l'UAL appelé **registre d'état** (status register). Ils sont positionnés, (portés au niveau 0 ou 1 selon les cas), et testés individuellement, soit automatiquement par l'UC, soit directement par le programmeur. Le nombre de bits contenus dans le registre d'état est fonction du type de microprocesseur (voir schéma ci-dessous).



En principe, le registre d'état comporte les indicateurs d'états suivants :

Retenue (Carry)
Zéro (détecteur de zéro)
Dépassement (Overflow)
Signe (résultat négatif) (Negative ou Sign)
Demi-retenue (Half carry)
Parité (Parity)

Retenue. L'indicateur de retenue est positionné à 1 si la dernière opération a entraîné un report sur le MSB, (le bit le plus significatif). Il est employé dans l'addition des nombres composés de plusieurs mots (longueur de mot = nombre de bits présents dans l'accumulateur). Dans les opérations de décalage, l'indicateur de retenue est utilisé comme si le mot possédait un bit supplémentaire (le bit de retenue) : le bit décalé doit être mis dans la retenue (voir schéma ci-dessous). Dans certains microprocesseurs, un tel système charge l'instruction qui active le déplacement « circulaire » des bits en utilisant le bit de retenue (voir schéma, page 891).

Zéro. L'indicateur de zéro est à 1 si le résultat de la dernière opération est 0.

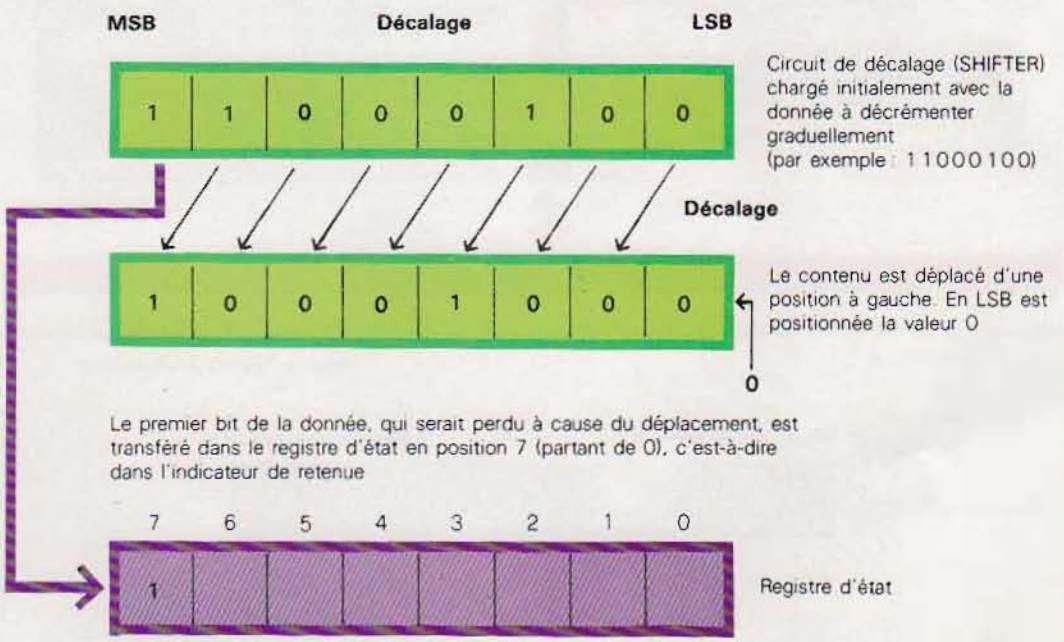
Dépassement. L'indicateur de dépassement prend la valeur 1 quand le résultat de la dernière opération arithmétique n'est pas correct, à cause de la présence d'une retenue après le chiffre le plus significatif. Normalement, le dernier bit (MSB) indique le signe. Ceci signifie qu'il y a eu une retenue dans l'avant-dernier bit et que le résultat dépasse la capacité du mot. Par exemple, en additionnant les nombres $(64)_{10}$ et $(96)_{10}$, on obtient :

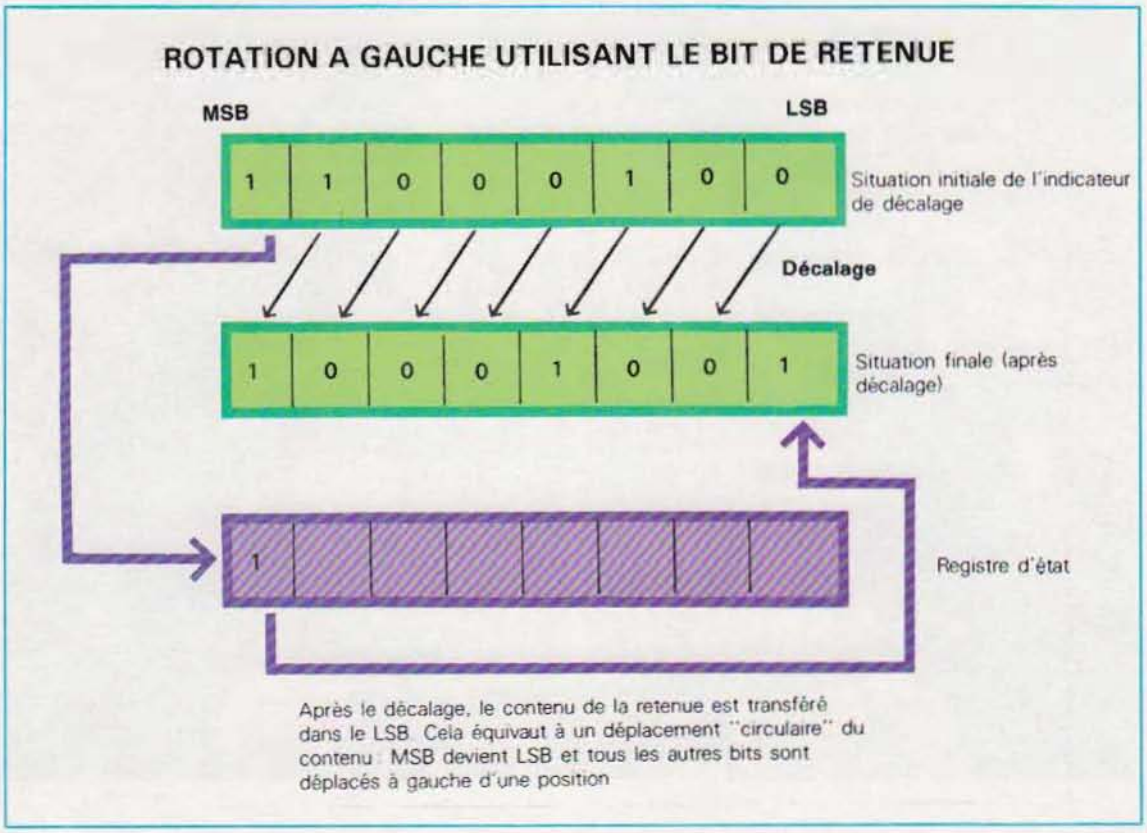
Bit du signe — Bit le plus significatif

```
  0 1 0 0 0 0 0 0      (2^6 = 64 décimal)
  0 1 1 0 0 0 0 0  (2^6 + 2^5 = 96 décimal)
  -----------------
1 0 1 0 0 0 0 0
```

Il y a donc dépassement de capacité (7 bits plus 1 bit pour le signe). En effet, on obtient un résultat supérieur à 111 1111.

OPERATION DE DECALAGE AVEC UTILISATION DU BIT DE RETENUE





Signe (résultat négatif). C'est l'indicateur du signe. Il est à 1 quand, à la fin de la dernière opération, le MSB vaut 1. Ce bit est également dit « bit négatif » en ce sens que, dans la rotation en complément à 2, le MSB indique le signe avec la convention suivante :

0 = signe positif
1 = signe négatif

Le bit du signe est utilisé pour tester le signe du résultat (positif ou négatif).

Demi-retenue (ou retenue intermédiaire). Cet indicateur est réservé exclusivement aux microprocesseurs de 8 bits travaillant en arithmétique bécédique (BCD : Binary Coded Decimal). Il est à 1 si la dernière opération a donné une retenue sur les 4 premiers bits du mot. Pour représenter chaque chiffre décimal, le code BCD utilise 4 bits*. Dans un 8 bits, deux codes BCD peuvent se trouver dans chaque mot. Lors d'une addition, on pourrait avoir un report entre le chiffre BCD le moins significatif (4 premiers bits) et le chiffre suivant (4 derniers bits). Dans ce cas, le bit de demi-retenue (ou retenue provisoire - intermediate carry) est positionné à 1.

Parité. L'indicateur de parité se positionne à 1 quand, dans une opération, le nombre de bits ayant la valeur 1 est pair (parité paire). Le bit de parité est positionné à 1 quand le nombre de bits de valeur 1 est impair. Le contrôle de parité est couramment employé dans la manipulation des caractères ou en transmission. Dans ce dernier cas, il est important de savoir si un caractère a été correctement reçu. Le test doit être effectué dans le dispositif de réception. En cas d'erreur due à l'inversion d'un bit par exemple (un bit positionné à 1 devient 0 et vice versa), le contrôle de parité résultant est erroné : la retransmission du caractère est alors demandée. Malheureusement, ce système n'offre pas un degré de sécurité suffisant : en cas de double inversion, l'erreur n'est pas détectée. Le test sur

*Le code BCD est composé de 4 bits dont la valeur normale s'exprime en puissance de 2 (8, 4, 2, 1) et il exclut toutes les configurations qui donnent une valeur décimale supérieure à 9. Il faut se rappeler qu'avec 4 bits on peut écrire des nombres binaires jusqu'à la valeur 8+4+2+1=15 (décimal).



J. Pickerell/Marka

Une chaîne de montage de circuits imprimés.

l'indicateur de parité est utilisé uniquement pour des instructions particulières.

Indicateurs spécifiques. Il s'agit, par exemple, de l'indicateur d'autorisation d'interruption (interrupt enable). Dans ce cas, un dispositif envoie sur une ligne spéciale un signal d'interruption à l'UC. Ce signal n'est pris en considération que si le bit d'autorisation d'interruption est positionné à 1. En d'autres termes, l'unité de contrôle ne peut se servir d'un dispositif externe que lorsqu'elle reçoit une autorisation. L'unité de contrôle envoie alors des signaux qui déclenchent l'activation d'une routine.
Cette routine suspend le programme en cours d'exécution, procède à la sauvegarde de son « contexte », c'est-à-dire du contenu des registres (compteur ordinal, accumulateur). A chaque type d'interruption est associé un programme de service spécifique. Les demandes du dispositif externe étant satisfaites, l'exécution du programme précédemment suspendu reprend normalement. Le programmeur a la possibilité de mettre les interruptions hors service (on dit « inhiber ») pour préserver le déroulement normal d'un programme. Dans certains cas, en effet, des zones de programme particulièrement critiques ne peuvent être interrompues (on dit que ces zones sont « masquées »). L'interruption est alors inhibée ; les demandes d'interruption ne sont acceptées qu'en fin de traitement. Même l'UC peut automatiquement inhiber les interruptions, par exemple à l'initialisation du système ou lorsqu'une tâche est déjà en cours d'interruption.

Il est très important de savoir que, dans un microprocesseur, les indicateurs d'état ne sont pas toujours remis à jour. Cela signifie que l'on peut avoir affaire à des indicateurs à 1, non pas parce qu'ils ont été positionnés dans cet état par l'instruction en cours, mais parce que c'est le résultat de l'instruction précédente. Dans l'écriture d'un programme Assembleur, il est donc vital que le programmeur dresse l'inventaire des indicateurs qui ont été modifiés par un précédent cycle d'instructions.

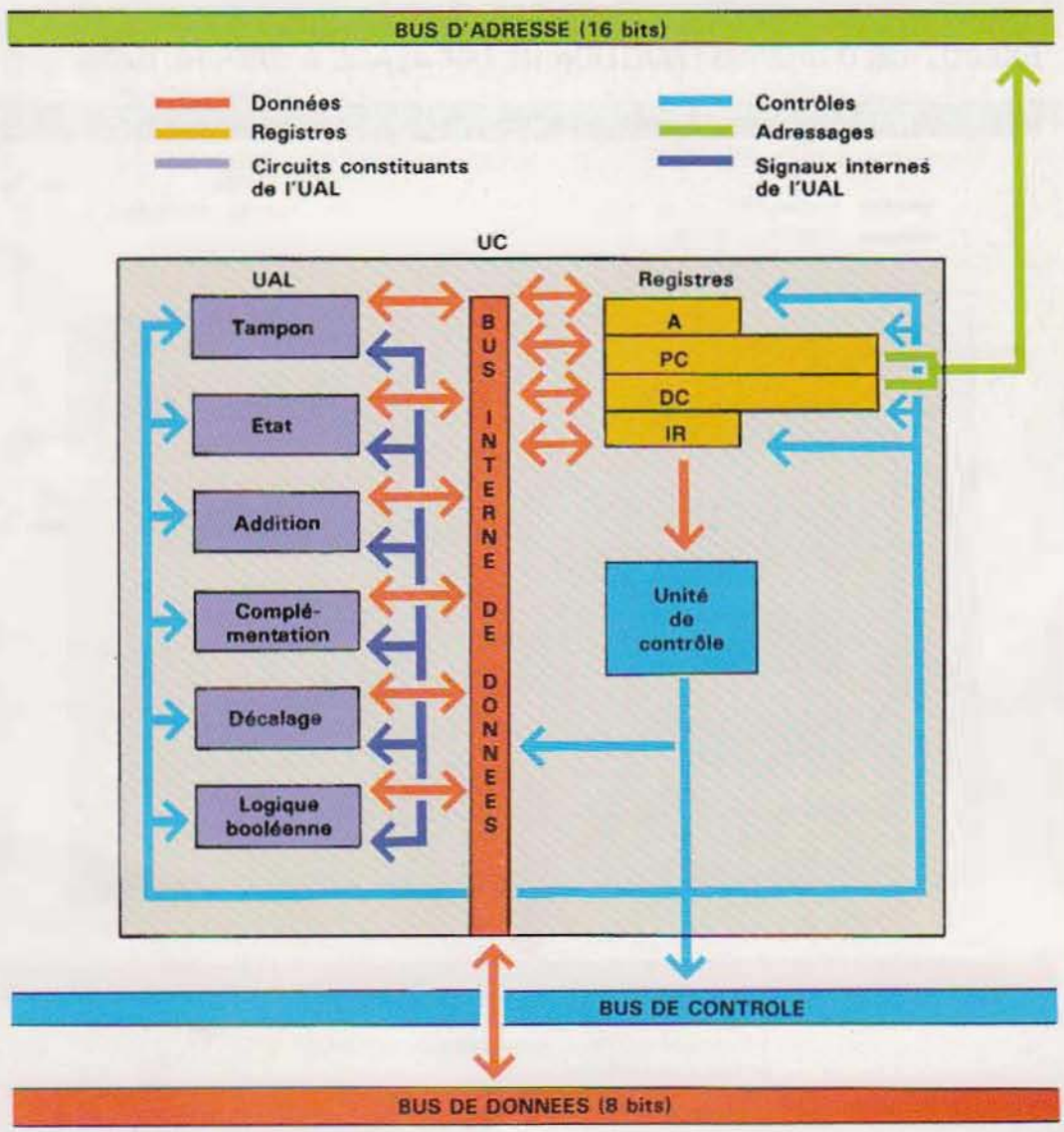
Le tampon de l'UAL

On trouve généralement, dans l'UAL, un registre temporaire, utilisé comme mémoire-tampon **(buffer store)** pendant le déroulement des calculs (voir schéma ci-dessous). Exemple : dans l'opération d'addition, le premier chiffre est chargé dans l'accumulateur, le second est temporairement stocké dans le registre-tampon. Les circuits de l'UAL réalisent ainsi la somme entre les contenus de l'accumulateur et du tampon en écrivant le résultat dans l'accumulateur.

L'unité de contrôle

L'existence, dans l'UAL, de circuits permettant le déroulement des opérations arithmétiques, booléennes ou de décalage, n'est pas suffisante à elle seule pour garantir le fonctionnement correct du microprocesseur.
Toutes les opérations que l'UAL est capable d'exécuter sont gérées par l'unité de contrôle.
Dans la séquence d'exécution d'une instruction, la première phase concerne le décodage du contenu du registre d'instructions (IR).

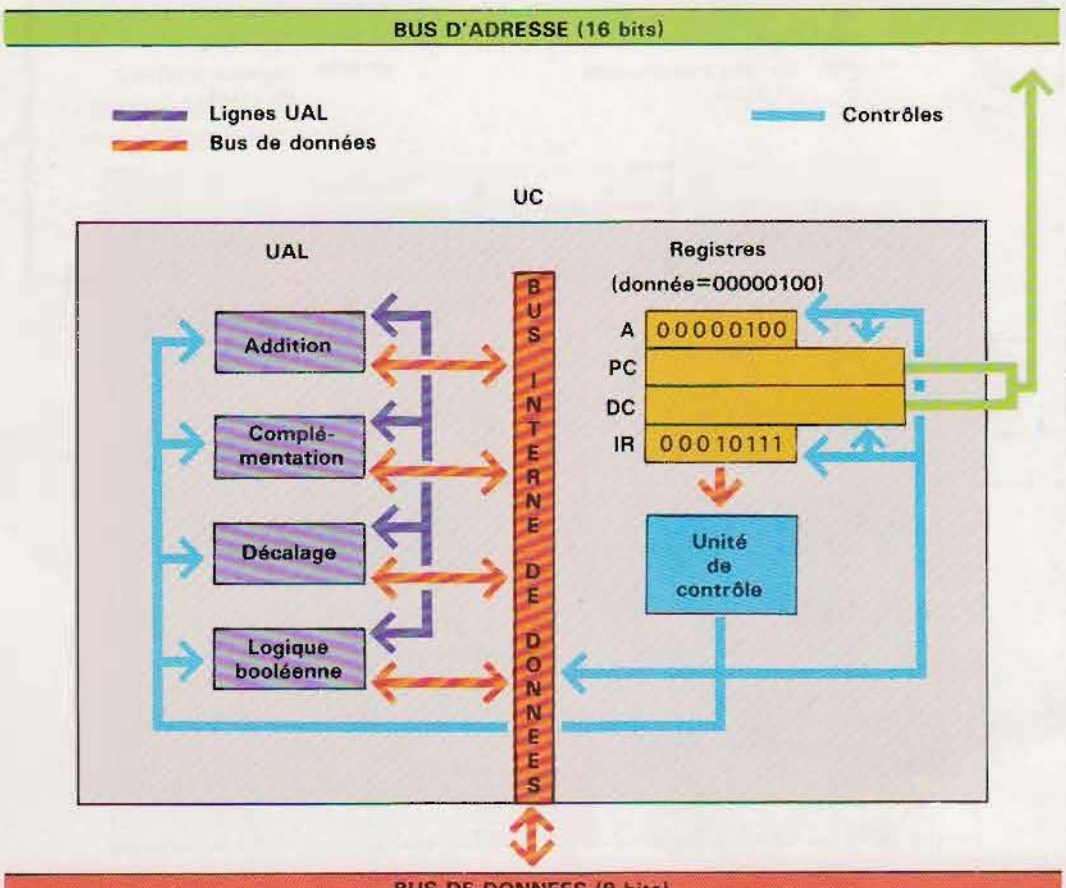
SCHEMA COMPLET D'UNE UC



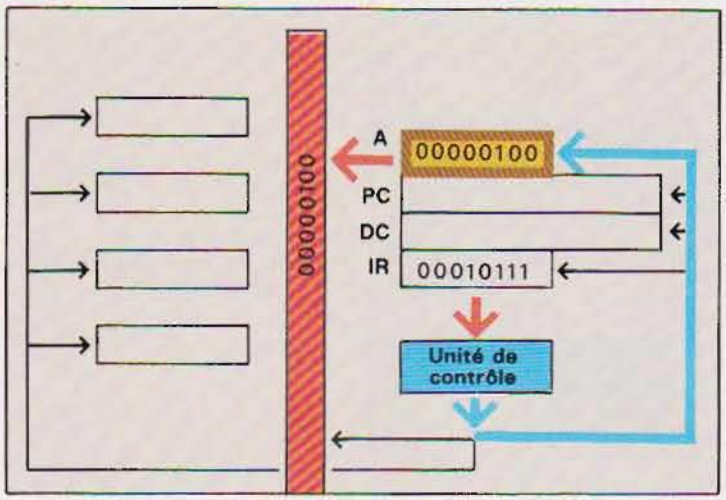
En fonction du code binaire contenu dans l'IR, l'unité de contrôle émet les signaux d'autorisation qui permettent aux éléments de l'UAL de recevoir les données. L'exécution d'une instruction de décalage à gauche du contenu de l'accumulateur est illustrée pages 894 à 896. Les signaux de contrôle jouent un rôle essentiel dans le déclenchement de l'exécution d'une fonction ou dans l'échange d'information (interruption, fin d'exécution).

Tous les dispositifs internes d'un microprocesseur échangent des informations entre eux et avec l'extérieur en se servant des mêmes lignes (bus). L'unité de contrôle (Control Unit) ouvre l'accès du bus à un seul dispositif à la fois. Les dispositifs ayant accès au bus possèdent des circuits d'entrée/sortie particuliers, dits « à trois états » (three states), qui se relient au bus dès qu'un signal est détecté. Ce mécanisme s'applique aussi aux dispositifs reliés à des bus externes (circuits E/S, mémoires...). Dans ce cas, il est prévu, dans les puces, une broche de sélection de circuit dite « chip enable » (autorisation du circuit) ou « chip select » (sélection de circuit), reliée à un bus de contrôle externe. Sous le contrôle du microprocesseur, ce circuit autorise l'accès du bus à un des dispositifs, en même temps qu'il déconnecte tous les autres (voir schéma page 897).

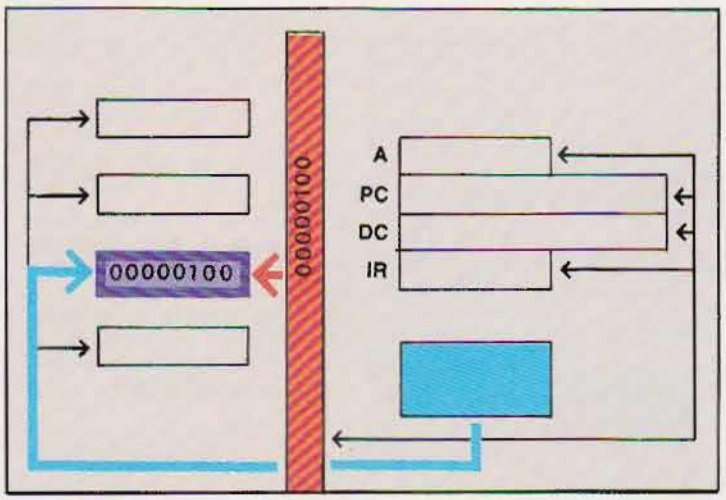
EXECUTION D'UNE INSTRUCTION DE DECALAGE A GAUCHE DANS L'UC



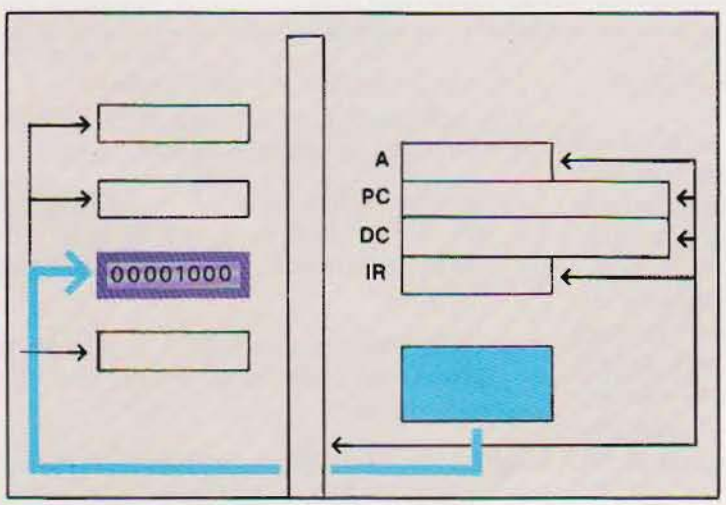
1 Etat initial de l'UC. L'accumulateur A contient la donnée à décrémenter.
Le registre d'instructions IR contient le code de l'instruction à exécuter (décalage à gauche)



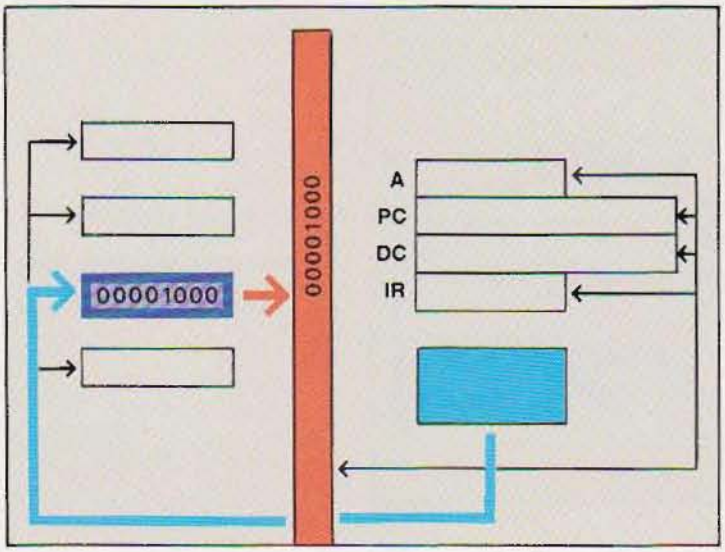
2 L'unité de contrôle décode et reconnaît l'instruction.
Le premier signal de contrôle active le transfert du contenu de l'accumulateur sur le bus interne de données



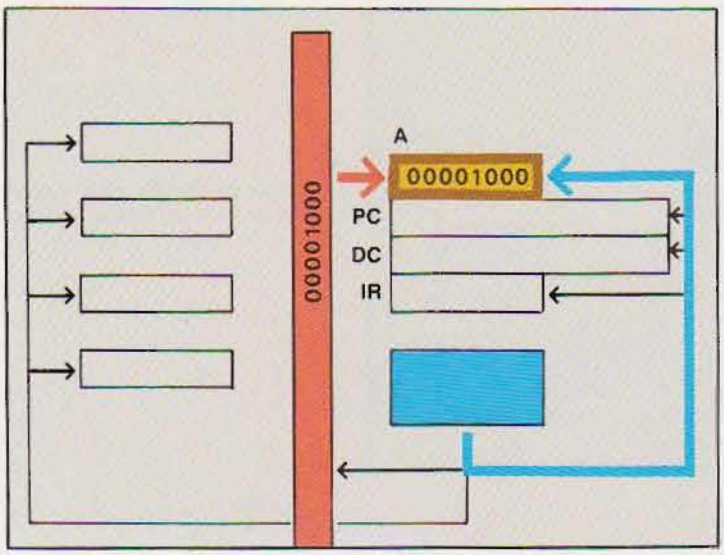
3 Le signal de contrôle suivant active le transfert de la donnée du bus vers le circuit de décalage (interne à l'UAL)



4 A la suite de l'envoi d'un nouveau signal de l'unité de contrôle, le contenu du circuit de décalage est déplacé d'une position à droite
La dernière position à droite (LSB, bit le moins significatif) est sur le zéro



5 A la fin de l'opération de décalage, le contenu du shifter est transféré sur le bus interne de données



6 La donnée est transférée du bus à l'accumulateur. Elle se substitue à la valeur précédente (donnée de départ).
Dans l'accumulateur la valeur initiale (00000100) est maintenant présente, décalée d'une position vers la gauche (00001000)

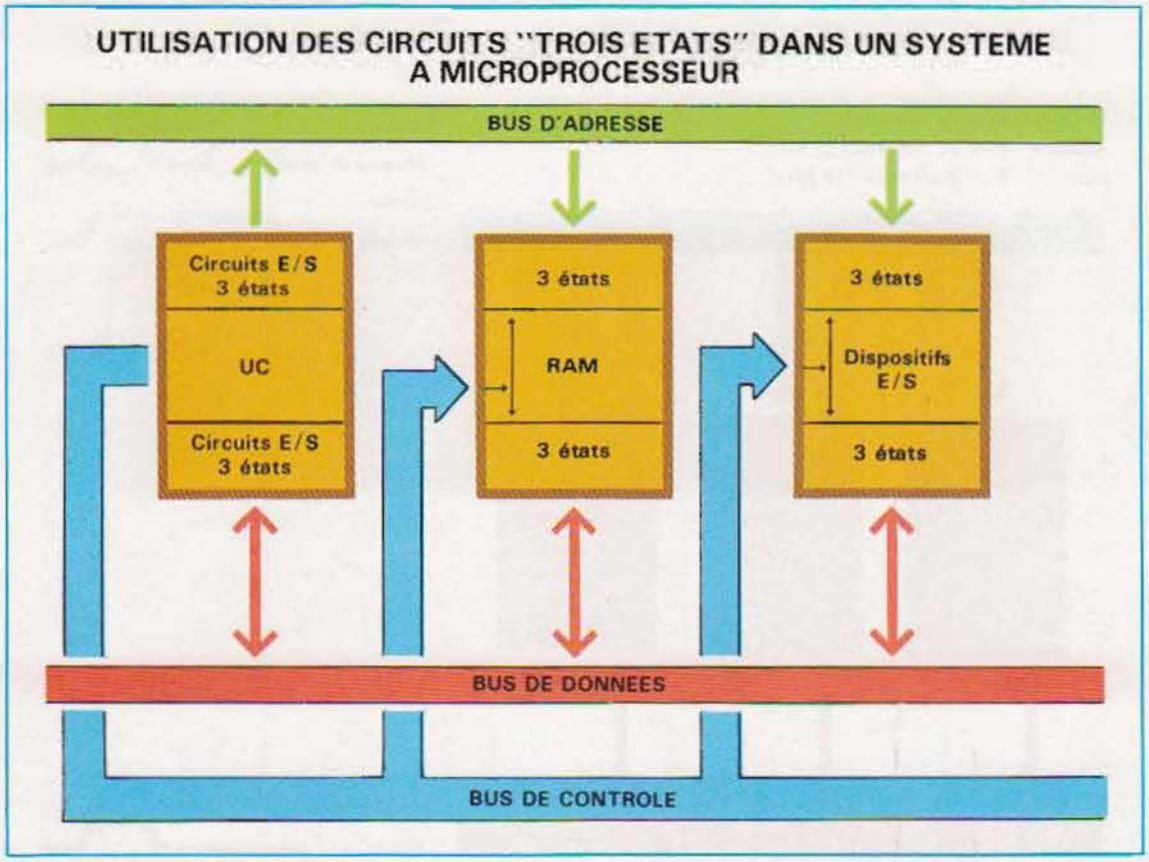
L'exécution des instructions

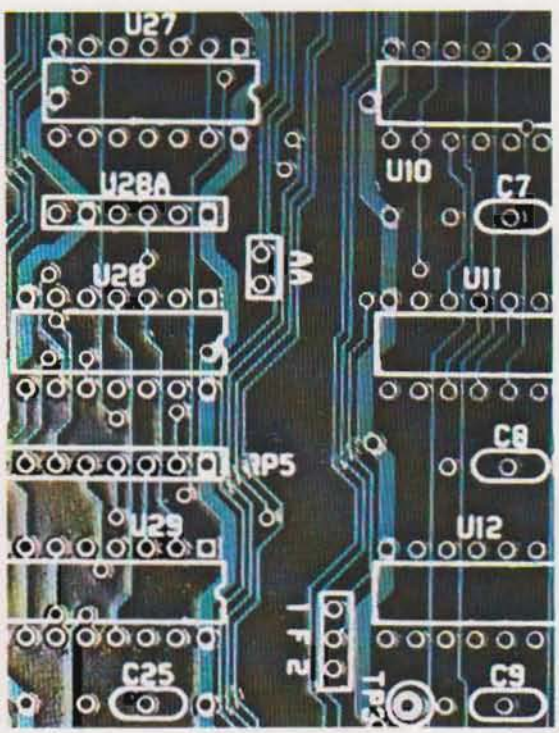
La phase d'exécution des instructions est peut-être celle qui utilise la plupart des ressources internes d'un microprocesseur. Elle est aussi celle qui échappe le plus à une logique simple de déroulement.
La lecture des paragraphes qui suivent demandera une attention particulière : la compréhension des mécanismes internes de l'UC facilitera incontestablement l'étude des instructions du langage Assembleur, très proches du fonctionnement pas à pas de la machine.

La synchronisation des signaux

Les bus gérés par l'UC sont détaillés sur le schéma de la page 898. Ils comportent :

- les bus d'adresse à 16 fils. Chaque bit circule sur une piste, et on peut ainsi adresser jusqu'à 65535 éléments de mémoire
- les bus de données à 8 fils. Il est possible de transférer en même temps les 8 bits d'une donnée
- les bus de contrôle : ensemble de circuits qui signalent le type d'opérations à effec-





© Pozzoni/Marka

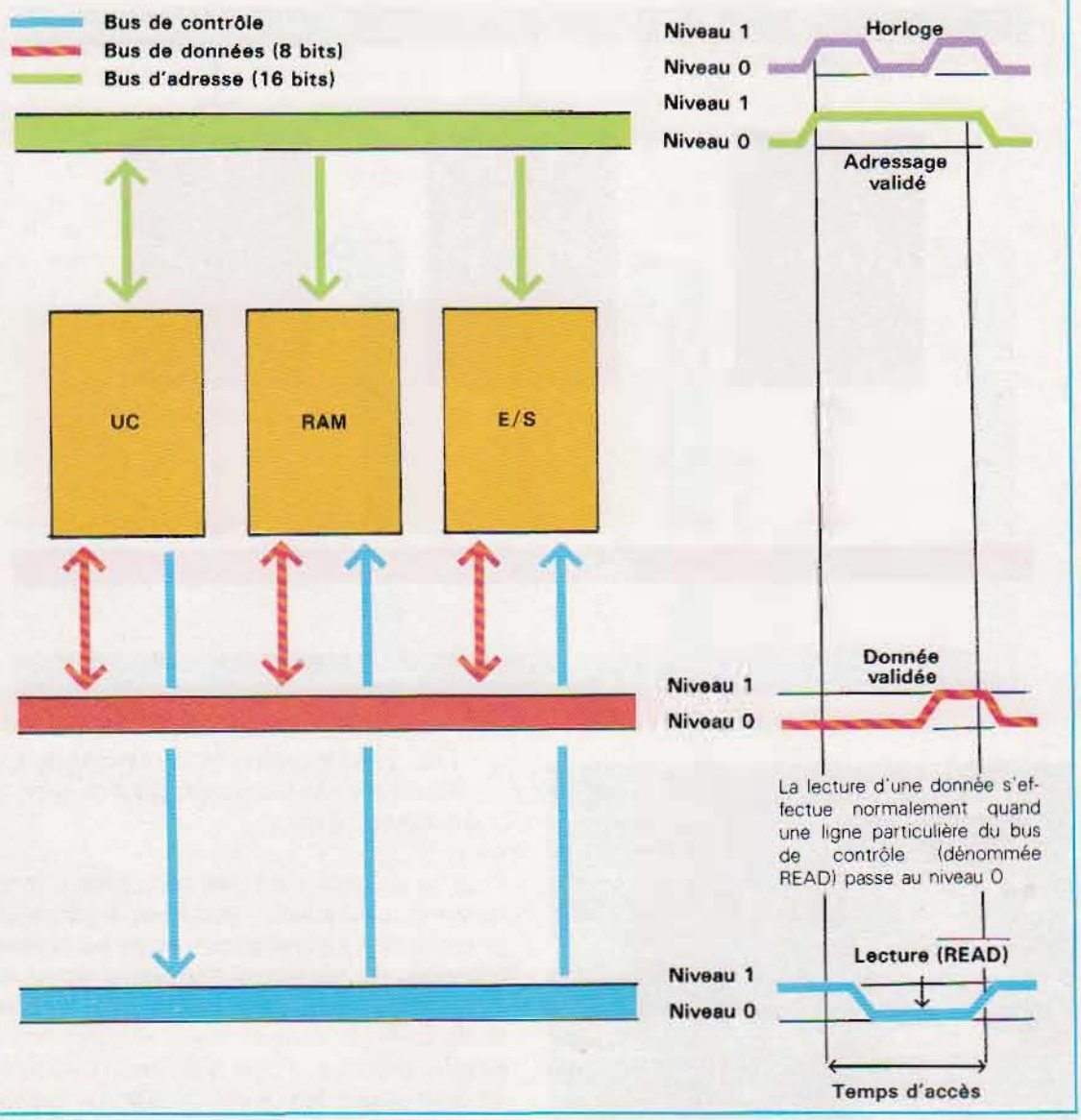
Gros plan d'un circuit imprimé.

tuer. La disposition et le nombre de ces lignes dépend essentiellement du type de microprocesseur.

Pour qu'un système à microprocesseur fonctionne correctement, il doit obéir à des règles précises de synchronisation entre les signaux échangés par les divers dispositifs. Aussi, les transferts sur les bus d'adresse, de données et de contrôle sont-ils liés à des synchronisations précises. Page 898, est représenté un diagramme des temps de signaux correspondant au déroulement d'une opération de lecture en mémoire. Le processus commence par le transfert du contenu du compteur ordinal (PC) sur le bus d'adresse, synchronisé avec l'horloge du système.

La mémoire fournit la valeur demandée au bus de données à l'instant déterminé par un certain nombre de périodes d'horloge (trame temporelle). Pour notre démonstration, nous avons jugé suffisant de travailler sur une seule période. Le laps de temps qui s'écoule pendant l'exécution de cette opération est appelé **temps d'accès**. Il représente le délai

DIAGRAMME DE TRANSMISSION DES SIGNAUX EN MODE LECTURE



qui s'écoule entre le lancement, par un organe de commande, d'un ordre de lecture ou d'écriture en mémoire, et le moment où celle-ci peut commencer à émettre ou à recevoir les premières données correspondantes. Les signaux doivent être stables pendant une durée suffisante pour permettre au microprocesseur de les transférer dans ses registres. La nature de l'opération est spécifiée par les signaux qui circulent sur les lignes de contrôle. Celles-ci permettent au dispositif de recevoir ou de transmettre l'information.

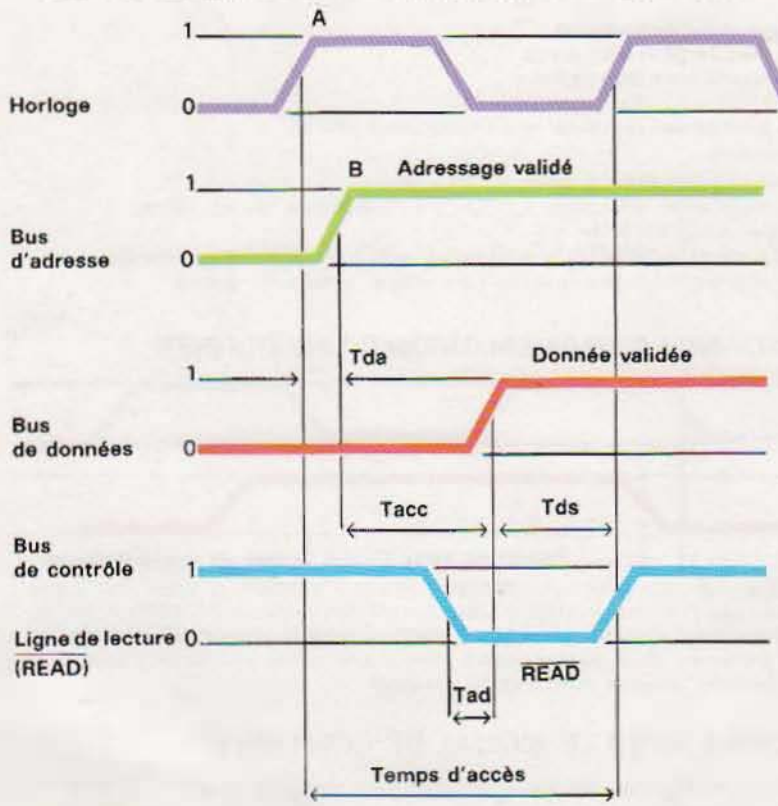
En réalité, le déroulement de ces opérations est beaucoup plus complexe. La synchronisation des signaux est un des paramètres les plus critiques dans la conception d'un système à base de microprocesseur. Les temps d'accès (voir schéma à droite) varient, par exemple, en fonction de la température. Les informations présentes sur les portes du microprocesseur ne sont jamais parfaitement synchronisées avec l'horloge ; cette discordance engendre des retards non négligeables. Donc, pour en tenir compte, le système doit

mesurer avec précision le décalage né entre l'impulsion de l'horloge et l'apparition des adresses sur le bus (Tda). Il calcule également la différence entre le temps de réponse de la mémoire (qui est fonction de la nature de l'opération demandée), et le délai nécessaire pour « stabiliser » la donnée sur les bus des données.
L'opération d'écriture est plus compliquée que l'opération de lecture et prend en considération les facteurs suivants (schéma du haut page 900) :

1 / la longueur de l'impulsion d'écriture (Tw, write pulse)
2 / le délai d'attente avant d'envoyer l'impulsion d'écriture, afin que l'adressage soit validé (Tas, adress setup time)
3 / l'intervalle minimum de validité de la donnée avant que l'impulsion d'écriture soit finie (Tds, data setup time)
4 / l'intervalle de temps minimum durant lequel la donnée reste disponible sur les lignes des données (Tdh, data hold time).

L'ordre de succession des signaux sur les divers bus est critique, en ce sens que l'UC introduit non seulement un retard sur les bus d'adresse (Tda, adress delay time), mais aussi sur l'impulsion d'écriture (Tdw, write pulse delay time, voir bas de la page 900).

RETARDS DES SIGNAUX ACTIVANT UNE OPERATION DE LECTURE



L'UC, synchronisée avec le point A de l'horloge, envoie une adresse sur le bus. Par rapport à l'instant A, les signaux seront en état haut avec un certain retard (Tda, point B). A partir de cet instant, l'adressage est validé

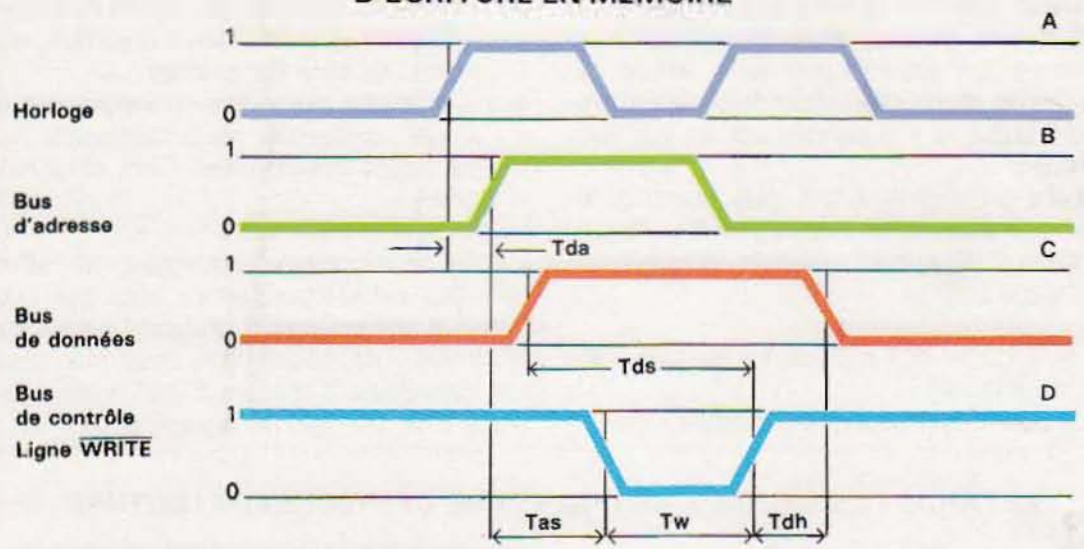
La mémoire ainsi adressée place son contenu sur le bus de données. Cette opération a aussi besoin d'un délai (Tacc) ; la donnée est donc validée en retard par rapport à l'apparition de l'adresse

La ligne qui active la lecture est normalement indiquée par le symbole de négation (READ négatif est $\overline{\text{READ}}$) parce qu'il s'active au niveau zéro

La ligne READ au niveau 1 indique que la donnée n'est pas prête. Un intervalle de temps est nécessaire après l'apparition de l'adresse (point B) ; la ligne $\overline{\text{READ}}$ est positionnée au niveau zéro et permet l'opération de lecture

Tda =Address delay time = Retard sur les lignes adresse
Tacc=Memory access time = Temps d'accès à la mémoire
Tds =Data setup time = Temps nécessaire à l'obtention d'une donnée validée
Tad =Data access time = Retard entre la commande de lecture et l'obtention de la donnée

SIGNAUX ET TEMPS DE RETARD D'UNE OPERATION D'ECRITURE EN MEMOIRE



Tda=Retard nécessaire à l'apparition de l'adresse
Tds=Temps durant lequel la donnée peut être écrite
Tad=Temps durant lequel l'adresse peut être utilisée
Tw =Durée du signal WRITE
Tdh=Temps durant lequel la donnée est présente après l'impulsion WRITE

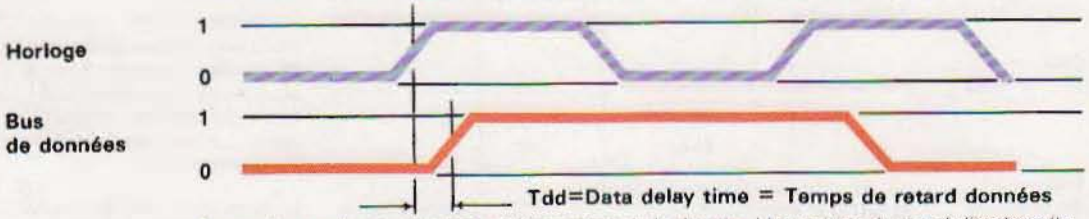
A L'horloge active l'opération d'écriture

B Les signaux sur le bus d'adresse ne sont validés qu'après un certain retard

C La donnée à écrire arrive avec retard après l'adressage. A ce point, la configuration des signaux est complète, mais la donnée n'a pas encore été écrite

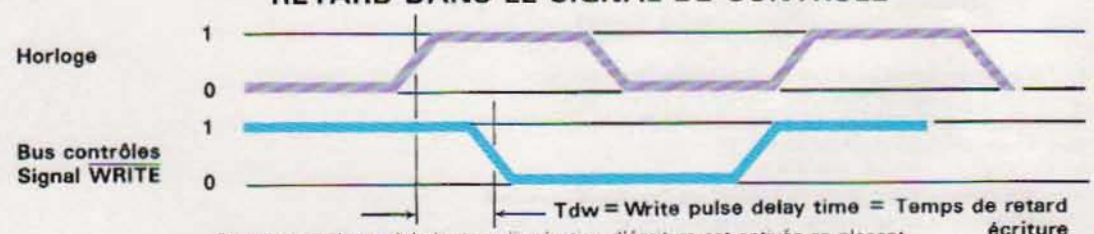
D En dernier lieu, arrive l'impulsion d'écriture (WRITE) ; la mémoire adressée au point B reçoit la donnée. A la fin du WRITE, la donnée reste disponible pour le temps Tdh ; ensuite, le cycle est terminé

RETARD DANS LA TRANSMISSION D'UNE DONNEE



Temps de retard entre l'horloge et l'apparition de la donnée. L'apparition du signal d'horloge (la ligne passe du niveau 0 au niveau 1) active la fonction qui doit se dérouler. Cependant, à cause de certains phénomènes physiques, les signaux électriques correspondants arrivent seulement après un certain temps. On a donc un retard entre la commande (montée de l'horloge) et l'exécution (apparition du signal correspondant à la donnée)

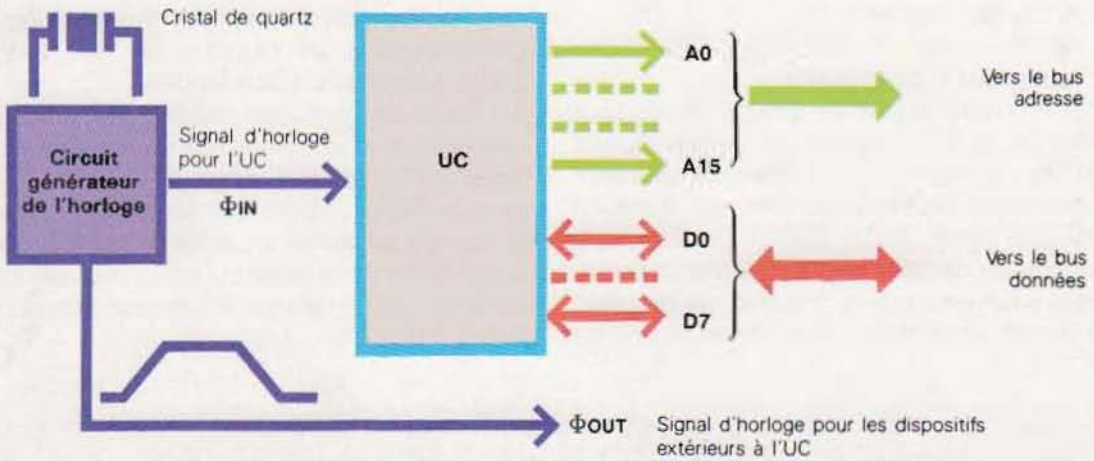
RETARD DANS LE SIGNAL DE CONTROLE



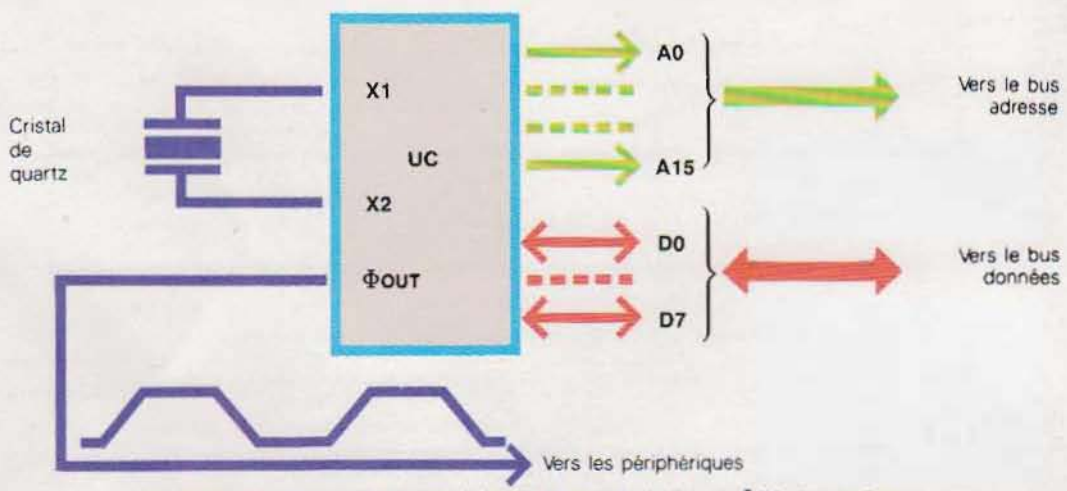
De façon analogue à la lecture, l'opération d'écriture est activée en plaçant à zéro le niveau d'une ligne particulière appelée WRITE. Le temps de retard de cette impulsion relative à l'horloge est indiqué par Tdw

Tous ces signaux de synchronisation sont en phase avec l'horloge du système. Dans certains microprocesseurs, les impulsions d'horloge (représentées par Φ - phi) sont produites par un circuit externe : un séquenceur dont la fréquence est contrôlée par un stabilisateur à quartz.
Dans le premier schéma situé ci-dessous, une UC utilise un générateur d'horloge externe. Le même circuit fournit le signal de temporisation (Φ_{out}) aux dispositifs externes devant être synchronisés avec l'unité centrale. Dans d'autres microprocesseurs, les circuits de génération du signal d'horloge sont intégrés dans l'UC. Dans ce cas (second schéma), l'oscillateur à quartz doit être directement relié à l'UC, et le signal de synchronisation (Φ_{out}) est disponible sur une broche du microprocesseur. Le signal d'horloge est utilisé par l'unité de contrôle pour synchroniser toutes les opérations internes du microprocesseur. Un cycle d'instruction

SCHEMAS DE REALISATION DU SIGNAL D'HORLOGE



L'UC avec oscillateur externe : le signal d'horloge est produit par un composant externe spécialisé



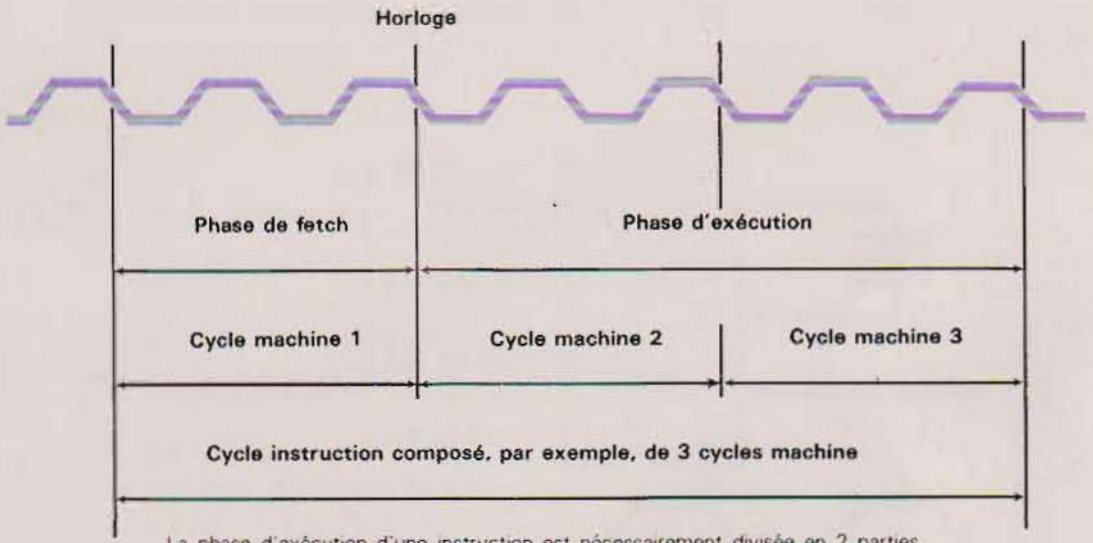
Oscillateur d'horloge intégré dans l'UC. Dans ce cas, la liaison ΦIN n'a pas lieu puisque le signal correspondant est interne à l'UC. De plus, les deux liaisons X1 et X2 du stabilisateur à quartz arrivent sur l'UC

se décompose en cycles de base (ou cycles machine) qui sont eux-mêmes des multiples de la période d'horloge, également appelée **état** (voir schéma ci-dessous).
Pendant ces intervalles, l'unité de contrôle envoie les signaux nécessaires aux dispositifs externes pour exécuter l'opération demandée. Un de ces signaux, par exemple, active la ligne lecture (READ) ou la ligne écriture (WRITE) d'un code d'instruction ou d'une donnée en mémoire. L'ensemble des signaux de contrôle que l'UC émet ou reçoit se classent, selon les fonctions (schéma page 903) :

- signaux pour le transfert des données
- signaux pour le contrôle des bus
- signaux système
- signaux de synchronisation
- signaux d'interruption

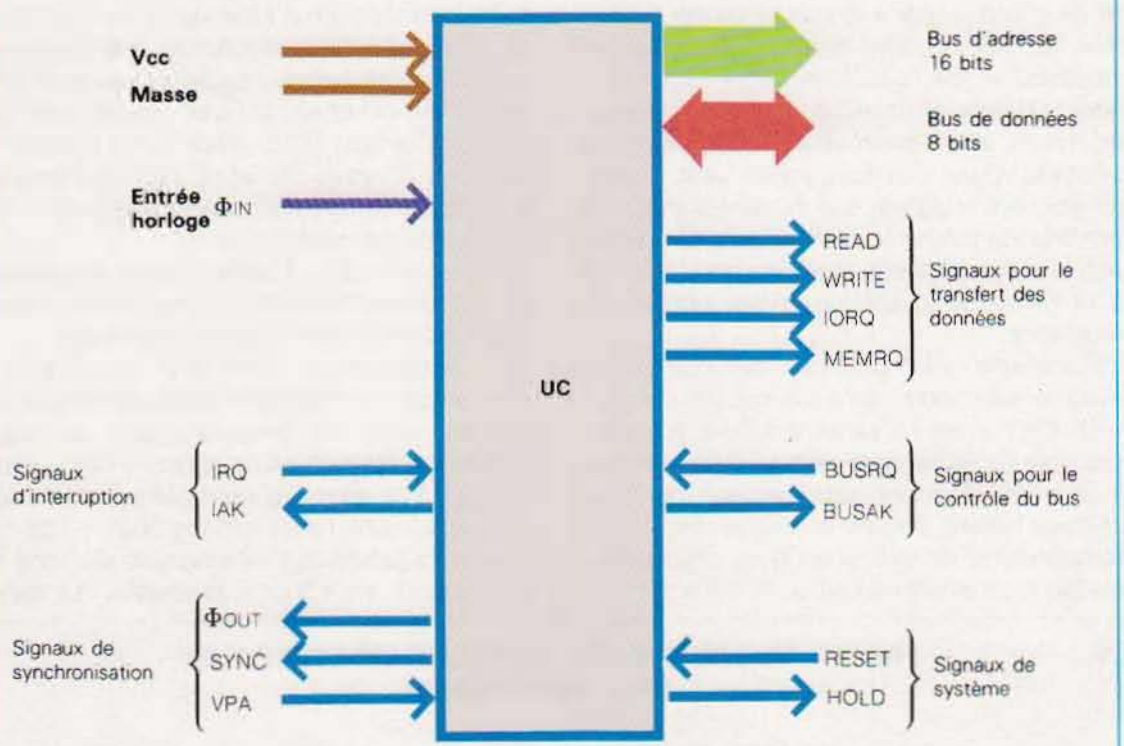
Appartiennent au premier groupe, les signaux spécifiant que l'opération de lecture (READ) ou d'écriture (WRITE) doit être effectuée vers la mémoire (MEMRQ) ou vers un dispositif général d'entrée/sortie (IORQ). On a recours aux signaux de contrôle de bus quand un dispositif externe exige le transfert de données de ou vers la mémoire, sans l'intervention du microprocesseur : c'est une opération d'accès direct à la mémoire (DMA, Direct Memory Access).
Généralement, l'échange en DMA se passe en trois étapes. Le dispositif envoie d'abord une demande (signal BUSRQ pour BUS ReQuest) d'attribution des bus (adresses, données, contrôles). Le microprocesseur répond par le signal BUSAK (BUS AcKnowledge, demande reçue), renvoie le contrôle des bus et se met en attente. Le dispositif ayant demandé le DMA contrôle ainsi le système, sans faire appel à l'UC. Il peut, par exemple, transférer directement des données en mémoire, alors que la pratique habituelle prévoit que les données à transférer soient d'abord chargées dans les registres de l'UC avant d'être transmises à la mémoire.
Au cours du DMA, les circuits extérieurs au microprocesseur sont informés de l'état de désactivation de l'UC par le signal HOLD qui, avec le RESET, appartient au groupe des signaux du système. La mise en activité du signal RESET interrompt n'importe quelle activité de l'UC et ramène le système à son état initial.

REPARTITION DES CYCLES SUR LE SIGNAL D'HORLOGE



La phase d'exécution d'une instruction est nécessairement divisée en 2 parties principales : la phase de fetch et la phase d'exécution. Pendant le fetch (un cycle machine) est prélevé un code d'instruction. Pendant l'exécution (qui occupe un nombre de cycles machine dépendant du type d'instruction) tous les signaux nécessaires sont activés (bus données, contrôles, adressages) pour le déroulement effectif de l'instruction

DESCRIPTION DES LIGNES DE CONTROLE D'UN MICROPROCESSEUR



Certains microprocesseurs reliés à l'UC ont besoin, pour leur fonctionnement, de recevoir le signal d'horloge appartenant au groupe des signaux de synchronisation. Ce signal (Φ_{out}, clock out) n'est pas suffisant pour synchroniser l'UC et le circuit externe (même si les deux dispositifs sont du même constructeur). Deux autres signaux sont nécessaires : la ligne VPA (Valid Peripheral Address) qui informe le microprocesseur du moment où le dispositif est prêt à commencer le transfert (lecture ou écriture, selon les autres lignes de contrôle) ; et le signal SYNC qui confirme ou bloque le transfert.

Le dernier groupe des signaux de contrôle est constitué des interruptions, dont nous parlerons plus loin. Pour l'instant, nous dirons que les signaux d'interruption prévus dans les microprocesseurs des micro-ordinateurs nécessitent des lignes de contrôle qui activent les fonctions d'interruption. Enfin, il est indispensable de fournir au microprocesseur une alimentation, généralement 5 V cc (courant continu) et de le relier à la terre.

La gestion de l'interruption

La fonction d'interruption, comme déjà mentionné, consiste à suspendre temporairement l'exécution d'un programme. Cette pause est mise à profit par l'UC pour répondre à une demande provenant d'un dispositif externe, lequel peut être, par exemple, une imprimante ou une unité de disque. Une fois la demande satisfaite (saut intermédiaire à un sous-programme de gestion du périphérique), l'UC revient à l'exécution du programme précédemment interrompu.

On a déjà parlé de la fonction d'interruption aux pages 132-133 du premier volume. L'exemple suivant illustre son intervention au niveau de l'UC, chaque fois que lui parviennent les lignes de contrôle de l'interruption. Supposons que nous voulions faire appel à un ordinateur personnel pour contrôler et régler la température d'un local ou d'un appartement. Nous aurions besoin d'un thermomètre pour mesurer la température et d'un dispositif qui convertisse cette donnée en un nombre binaire intelligible pour l'ordinateur. Celui-ci

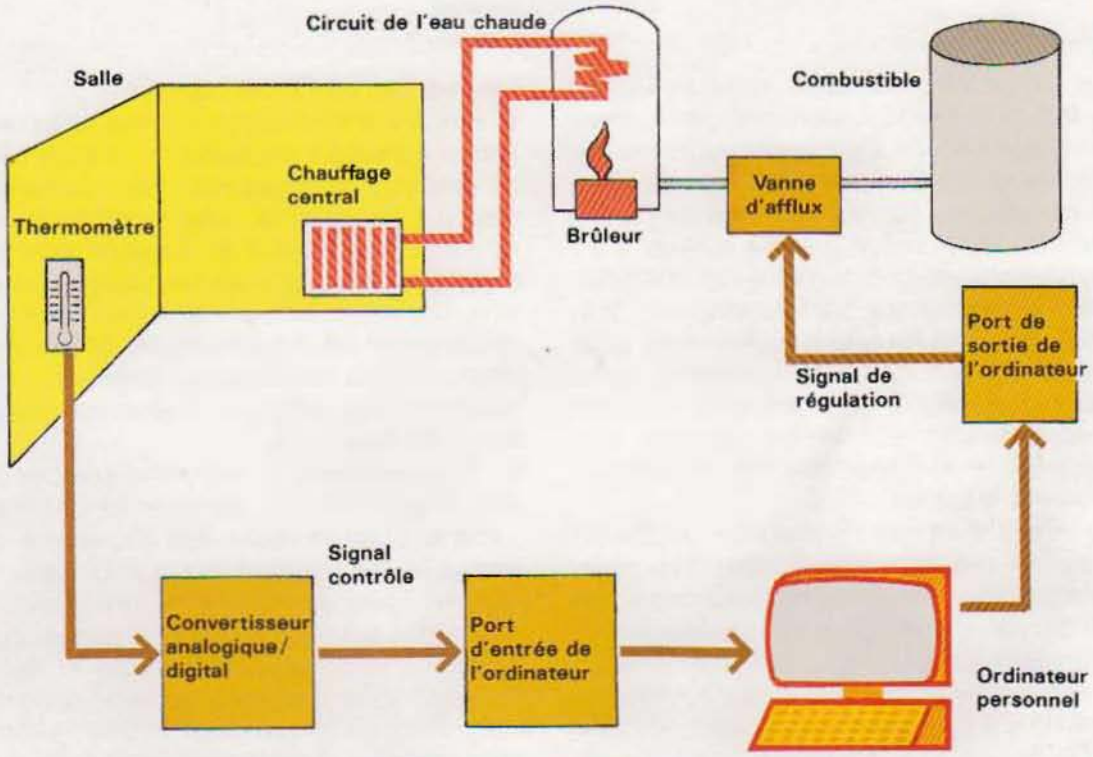
comparera donc cette valeur avec la température souhaitée. En fonction du résultat, il enverra les données nécessaires à un dispositif de contrôle (par exemple la vanne d'émission du combustible dans la chaudière de l'installation de chauffage) pour réguler ou ajuster la température. Le schéma ci-dessous illustre la conception d'un tel système de contrôle. Dans son fonctionnement interne, on pourrait imaginer que l'ordinateur lit « en continu » la température ambiante, la compare à la valeur imposée et envoie parallèlement à la vanne de la chaudière des signaux de régulation.

Ce scénario n'est pourtant pas compatible avec le fonctionnement normal de l'ordinateur. Ce dernier ne serait employé que pour des opérations banales : lecture d'une donnée à la porte d'entrée, comparaison de deux chiffres (valeur désirée et valeur réelle de la température) et exécution d'un programme simple de transformation de la différence en une donnée de sortie. De plus, les temps de réaction des dispositifs thermiques sont très longs, à tel point qu'ils rendent superflus le contrôle en continu. Mesurer la température (et donc exécuter la routine de contrôle) à des intervalles de temps réguliers (par exemple toutes les minutes), suffirait amplement. Le reste du temps, l'ordinateur serait occupé à exécuter d'autres tâches. C'est une manière d'optimiser le fonctionnement de l'unité centrale et de ses ressources.

Avec l'interruption, il est très facile de réaliser un système de contrôle qui fonctionne selon ces modalités (voir schéma ci-contre).

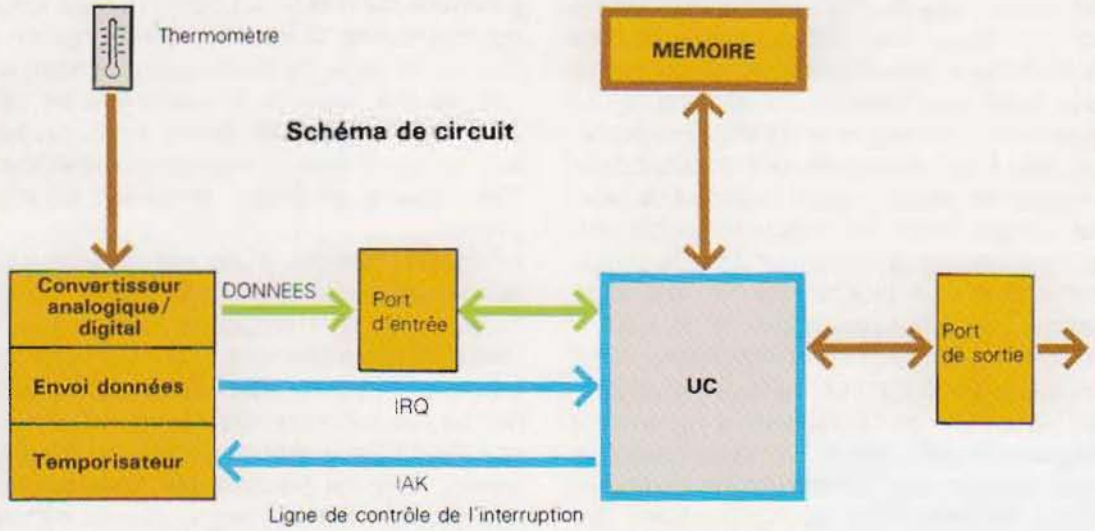
Un temporisateur (« timer ») associé au convertisseur analogique-digital engendre, à des intervalles de temps imposés, une demande d'interruption plaçant à la valeur logique 1 la ligne de contrôle appelée IRQ (voir aussi schéma de la page 903). L'UC reconnaît la demande d'interruption, suspend le programme en cours d'exécution, se rend

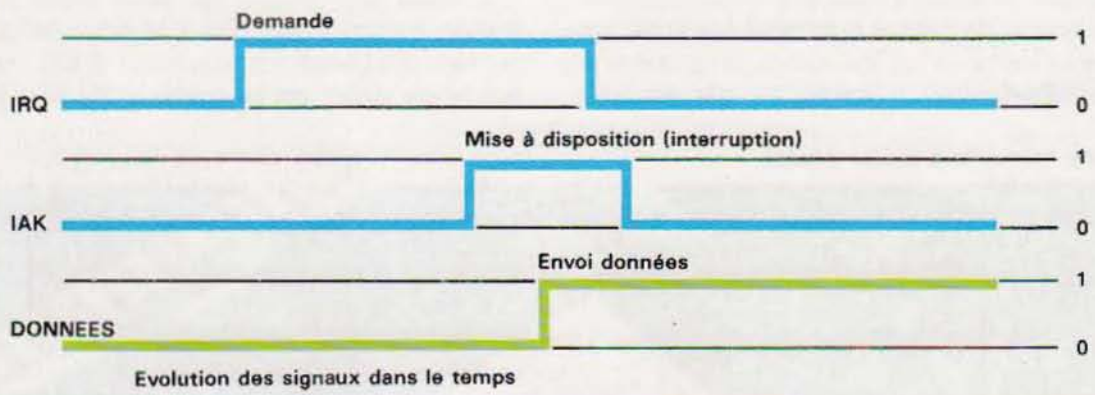
REGULARISATION DE TEMPERATURE EFFECTUEE PAR ORDINATEUR



Le convertisseur analogique/digital transforme la mesure de température en un volume binaire qui est envoyé sur le port d'entrée de l'ordinateur

FONCTIONNEMENT DU SYSTEME DE CONTROLE DE TEMPERATURE PAR INTERRUPTION





disponible pour répondre à la demande, mettant à 1 la ligne IAK. A réception de ce signal, le circuit externe envoie, aux dispositifs d'entrée, la donnée représentant la mesure de température et annule la demande d'interruption, plaçant à 0 la ligne IRQ.

L'ordinateur exécute ensuite une série d'ordres : déroulement d'un programme de lecture du message au port d'entrée, comparaison de la température mesurée avec celle désirée, envoi du signal de contrôle au port de sortie. La gestion de l'interruption achevée, l'UC reprend l'exécution du programme précédent.

Analysons maintenant de quelle manière l'UC exécute la routine de contrôle de la température, et comment elle réalise la reprise du programme suspendu.

La pile. Une fois la demande d'interruption acceptée, le dispositif externe place l'adresse initiale de la routine de contrôle dans le compteur ordinal. Ainsi, dans le cycle suivant, l'UC exécute un programme de mise en place (fetch program), qui lui permet de prélever la première instruction de cette routine et non celle prévue par le programme en cours de déroulement. Mais avant d'accepter la demande d'interruption, l'UC accomplit une série d'opérations nécessaires pour garantir une reprise correcte du programme précédent. Dans le programme, on a peut-être prévu de

ranger les données importantes dans le registre d'état, dans le compteur ordinal et dans l'accumulateur. Avant d'accepter l'interruption, l'UC sauvegarde systématiquement dans un espace-mémoire le contenu de tous les registres. C'est une sorte de photographie de la situation précédant l'acceptation de l'interruption. Une fois le programme exécuté, les valeurs sauvegardées sont rétablies dans les registres de l'UC ; donc l'adresse de l'instruction qui devait être exécutée au moment de l'interruption est chargée dans le compteur ordinal. Le programme reprend alors comme s'il n'avait jamais subi de coupure.

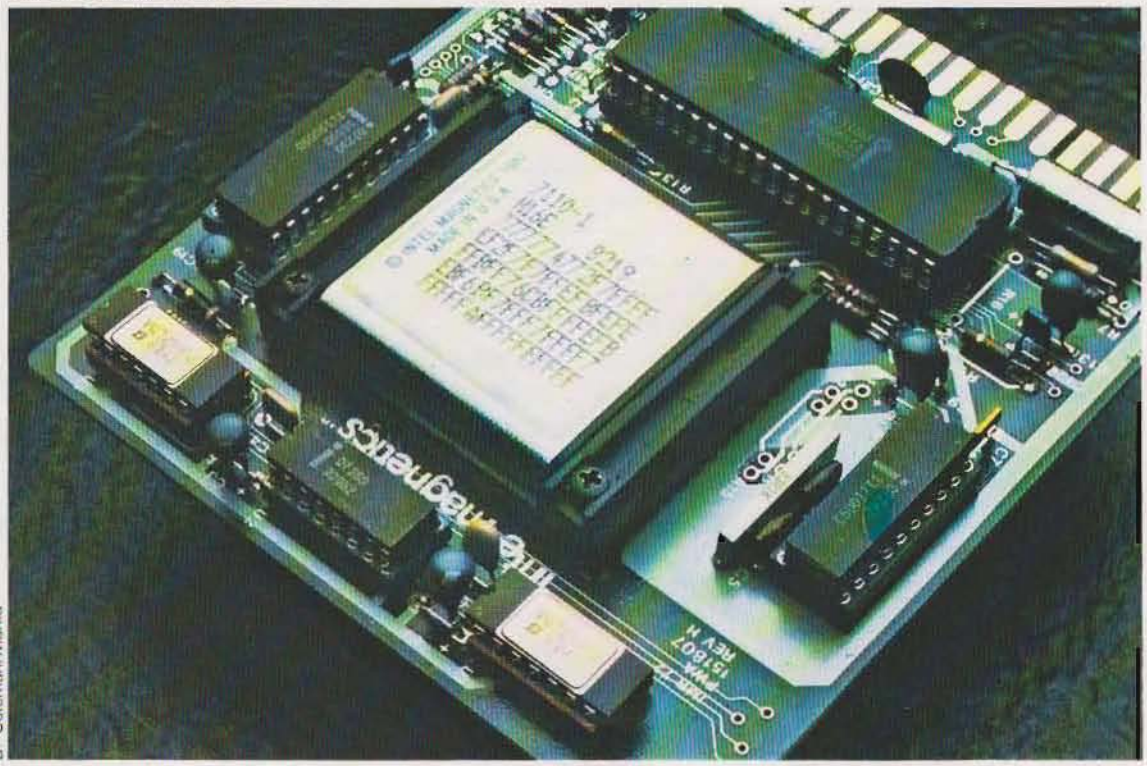
Le contenu du registre d'instructions n'est pas sauvegardé car l'UC achève l'exécution de l'instruction en cours avant d'accepter la demande d'interruption. Par conséquent, le code contenu dans l'IR correspond à une exécution déjà effectuée, qu'il n'est donc pas nécessaire de sauvegarder.

Le contenu des autres registres est sauvegardé dans un espace de la mémoire vive (RAM) constitué de petites mémoires linéaires, empilées les unes sur les autres, d'où son nom de **pile** (anglais : **stack**). La pile est aussi utilisée pour la sauvegarde des adresses de retour des sous-programmes. Pour gérer la pile, on utilise un registre particulier appelé **pointeur de pile** (stack pointer), dans lequel est mémorisée la situation de charge de la pile. Cette zone particulière de mémoire est vue par l'UC comme une structure de type LIFO (Last In First Out, dernier entré, premier sorti), c'est-à-dire comme une pile d'élément dans laquelle le dernier arrivé est servi en premier.

Le fonctionnement d'une pile de type LIFO est montré à la page ci-contre (haut). Dernière donnée à être introduite dans la pile, la valeur 33 est la première donnée qui sera lue. La valeur 14 pourra être lue seulement après 33. La pile fonctionne pratiquement comme une série d'assiettes empilées les unes sur les autres : elles ne pourront être retirées de la pile que dans l'ordre inverse de celui où elles ont été déposées.

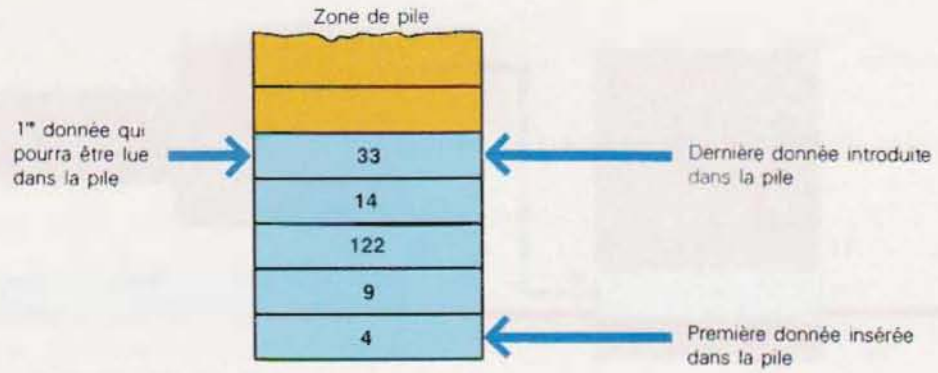
Sur la page ci-contre (au centre), est montré le schéma de l'UC à la réception d'une demande d'interruption. Dans le pointeur est contenu l'adresse hexadécimale 0100, qui pointe au début de la mémoire de la pile.

Une mémoire à bulles d'Intel.

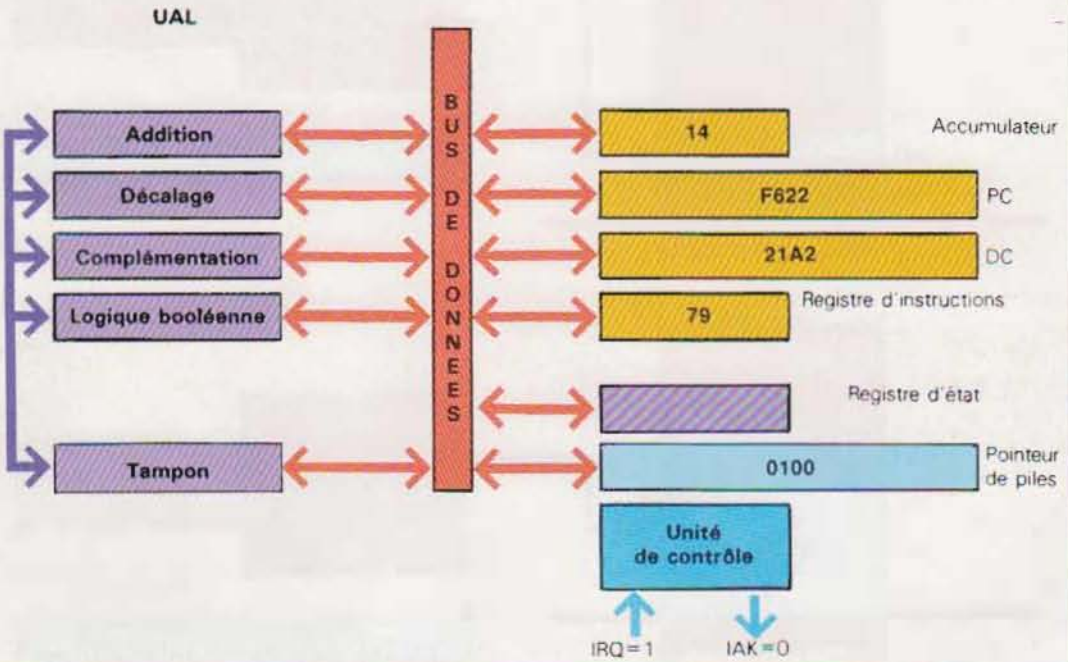


B. Coleman/Marka

STRUCTURE D'UNE PILE TYPE LIFO



ETAT DE L'UC A LA RECEPTION D'UNE DEMANDE D'INTERRUPTION

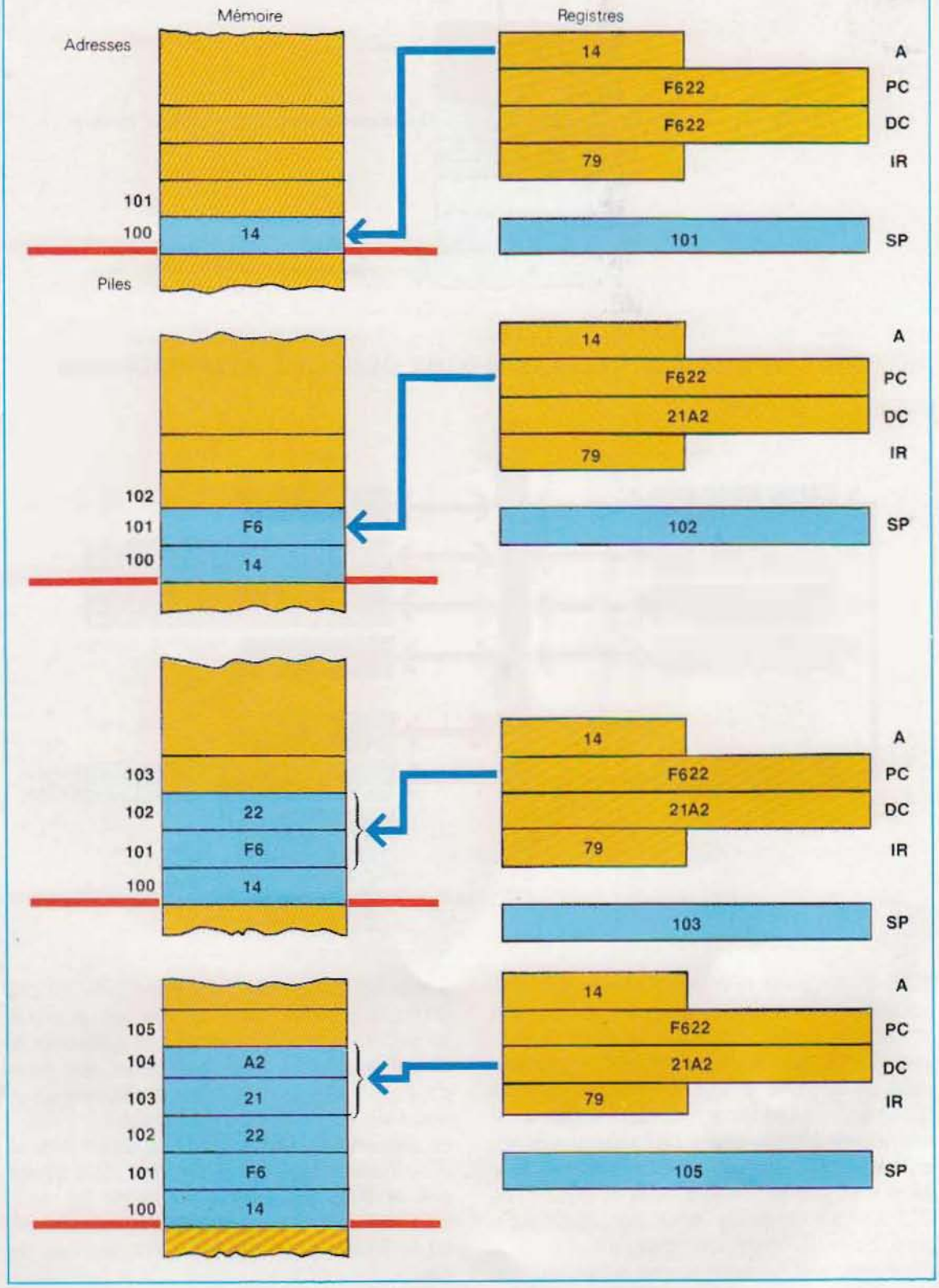


L'UC sauvegarde le contenu des registres à partir de cette adresse, place à 1 la ligne IAK et accepte l'interruption. L'opération de sauvegarde est détaillée en page 908. L'UC sauvegarde le contenu du premier registre dans la pile en écrivant la donnée dans la cellule de mémoire adressée par le pointeur de pile et, ensuite, incrémente de 1 la valeur du pointeur lui-même. Les registres à deux octets (PC et DC) sont sauvegardés, octet par octet, dans deux zones de mémoire contiguës.

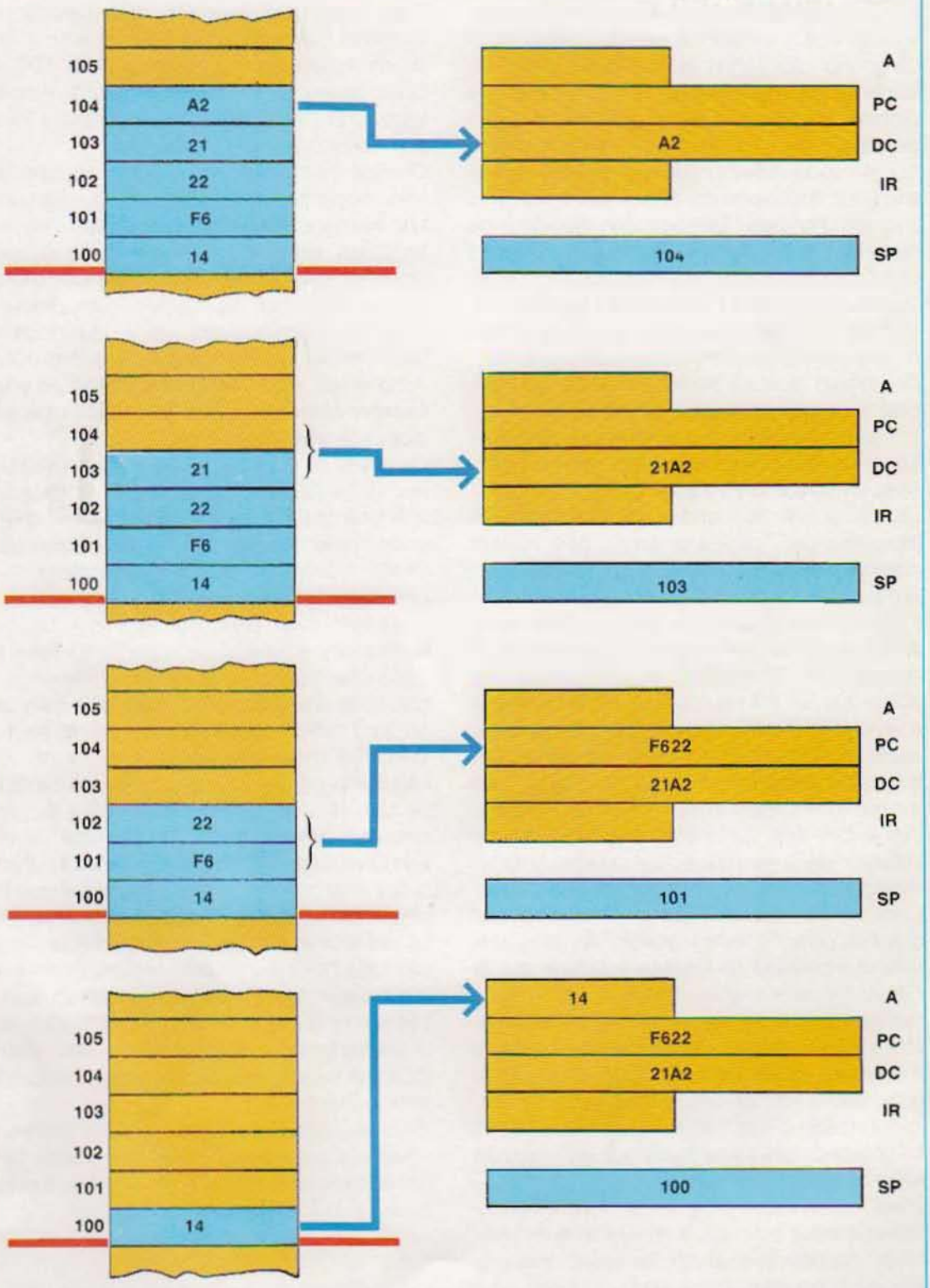
Pour réactiver le programme suspendu, on suit la logique inverse (voir page 909). L'opération d'écriture dans la pile se poursuit jusqu'à saturation. Dans certains systèmes, la pile a des dimensions fixes alors que dans d'autres systèmes elle est extensible et varie selon les exigences du programme.

En Assembleur, les opérations de lecture et d'écriture dans la pile sont facilement exécutables, grâce aux instructions qui réalisent l'accès à la mémoire en augmentant ou en diminuant automatiquement le pointeur de pile.

SAUVEGARDE DU CONTENU DES REGISTRES DANS LA PILE A LA SUITE D'UNE INTERRUPTION



ETABLISSEMENT DU CONTENU DES REGISTRES A LA FIN DE LA GESTION DE L'INTERRUPTION AVANT LA REACTIVATION DU PROGRAMME SUSPENDU



Structure des instructions Assembleur

L'UC a accès à une mémoire où sont enregistrés les codes hexadécimaux des instructions de gestion des registres. A chaque code instruction correspond une série d'opérations élémentaires dont la durée se mesure en cycles machine. L'UC doit les lancer à la réception de chaque code. Théoriquement, les instructions écrites en hexadécimal ne sont pas très nombreuses. Lorsque l'on décidera de recourir à une technique de programmation la plus proche possible de la machine, on utilisera presque toujours l'Assembleur pour les instructions. Ce langage utilise un jeu de codes mnémoniques (eux-mêmes codés en hexadécimal) plus faciles à traiter, et où chaque symbole a un correspondant précis en code instruction hexadécimal. **Le langage Assembleur et le langage machine sont en correspondance univoque.**

La traduction des codes mnémoniques en hexadécimaux correspondants, directement compréhensibles par l'UC, est faite par un programme utilitaire appelé **Assembleur.**

L'Assembleur traduit une suite de caractères ASCII (code mnémonique dépourvu de signification pour la machine) en bits compréhensibles par le microprocesseur. A l'intérieur d'un cycle d'instruction et dans le premier cycle machine (cycle de mise en place), cet ensemble de bits est chargé dans le registre instruction et décodé par l'unité de contrôle. Les autres bits de l'instruction sont ensuite prélevés de la mémoire puis copiés en fonction des signaux de contrôle générés.

Dans les premiers ordinateurs, il n'existait pas d'autres possibilités de charger les programmes directement en langage machine que le clavier hexadécimal ou binaire.

Les registres (comme par exemple le compteur ordinal) étaient chargés bit par bit, et la mémoire remplie octet par octet. Ce système comportait de graves problèmes de documentation et de correction d'erreurs (très fréquentes), qui s'ajoutaient à la difficulté de relire le programme écrit.

C'est exactement pour pallier ces inconvénients que fut introduit le programme Assembleur, qui permit d'utiliser les codes mnémoniques. L'écriture d'une instruction est plus aisée que sous forme binaire où une banale erreur, comme par exemple le déplacement d'un bit, compromet tout le code.

Supposons, par exemple, que l'instruction « charger l'accumulateur avec la valeur 10 » ait en Assembleur la forme LDA # 10*. Le code binaire correspondant pourra être du type 0011101000001010, et aura une forme peu claire, sujette à l'erreur.

Chaque instruction Assembleur occupe une ligne du programme source et est traduite en une instruction en code binaire appelée **instruction machine**. Chaque ligne du programme (ou chaque instruction) est divisée en quatre parties, ou champs, dont chacun a sa propre signification, liée à leur position. Les champs qui composent une instruction Assembleur sont l'**étiquette** (label), le **code mnémonique** de l'instruction, les **opérandes** et le **commentaire.**

Quand ils ne sont pas utiles, certains champs peuvent être omis. Par exemple, l'instruction précédente (LDA # 10) contient seulement le code mnémonique (LDA) et l'opérande (#10). L'étiquette est un identificateur de ligne utile seulement quand il faut passer à cette instruction dans le programme. Le code mnémonique représente la partie qui sera décodée par l'unité de contrôle. A l'inverse, le champ opérandes indique l'adresse mémoire où se trouve l'opérande, ou directement la valeur de l'opérande elle-même.

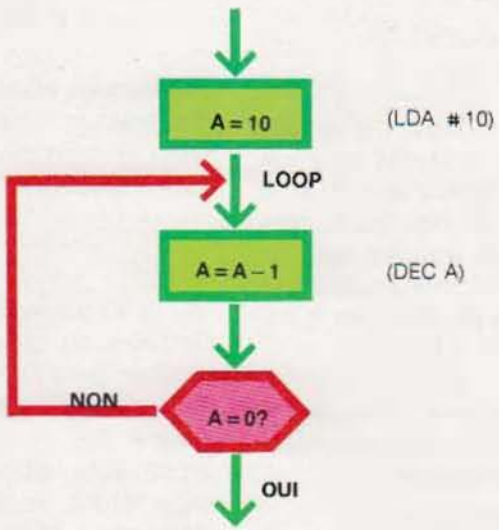
Le champ commentaire est utilisé à seule fin de documenter le programme. Sur la page ci-contre, une partie de programme en Assembleur illustre la syntaxe employée. La première instruction « charge » (mnémonique LD pour load) la valeur 10 dans l'accumulateur. La deuxième instruction décrémente (mnémonique DEC) d'une unité le contenu de l'accumulateur. La troisième effectue un saut à l'étiquette LOOP si le contenu de A n'est pas 0 (mnémonique BNZ, Branch Not Zero). Quand A sera égal à zéro, l'instruction suivante sera exécutée.

Il est important de comprendre la méthode employée pour envoyer une instruction. Si le programme devait être écrit en Basic, il aurait pu avoir la forme suivante :

* La signification du symbole # sera expliquée plus loin.

EXEMPLE D'INSTRUCTIONS ASSEMBLEUR

Label	Code mnémonique	Opérande	Commentaire
	LDA	#10	; charge dans l'accumulateur la valeur de 10
LOOP	DEC	A	; décrémente de 1 le contenu de l'accumulateur
	BNZ	A, LOOP	; saute à "LOOP" si A est différent de 0




```
100 A=10
110 A=A—1
120 IF A<>0 GOTO 110
```

La ligne 120 renvoie à la ligne 110 sous certaines conditions. Elle est l'équivalent de la BNZ A, LOOP, à la seule différence qu'en Assembleur la ligne 110 porte le nom de LOOP. En d'autres termes, si l'interprète Basic était capable de reconnaître des labels de branchement conditionnels avec un nom symbolique, le programme Basic aurait la forme :

```
      A=10
LOOP  A=A—1
      IF A<>0 GOTO LOOP
```

après avoir confondu l'instruction A=A—1 avec l'étiquette LOOP, et non plus avec le numéro de la ligne.
Le parallèle établi ici est uniquement qualificatif, car il existe entre les deux programmes une différence essentielle. Dans la version Assembleur, le symbole A n'est pas une quelconque mémoire, comme dans le Basic, mais un registre précis du microprocesseur (l'accumulateur).
En pratique donc, les deux programmes possèdent des fonctions très différentes. Le programme Basic effectue le calcul et le test de condition sur une mémoire générale qui, par hasard, s'appelle A. Le programme Assembleur n'implique aucune mémoire et exécute calculs et test dans l'accumulateur.

Mode d'adressage

Tout d'abord, voyons de quelle façon l'Assembleur accède aux données résidentes en mémoire.
A chaque mémoire utilisée dans un programme Basic est associé un nom symbolique grâce auquel on peut rappeler le contenu. En Assembleur, les choses sont plus complexes. Pour utiliser une mémoire, il faut d'abord

l'**adresser**, c'est-à-dire fournir au programme son adresse réelle (et non pas un nom symbolique). Les méthodes d'adressage les plus communément usitées dans les microprocesseurs — et par conséquent dans les langages Assembleur — sont les suivantes :

1 / adressage immédiat
2 / adressage absolu ou direct
3 / adressage indirect
4 / adressage conditionnel
5 / adressage relatif
6 / adressage du registre
7 / adressage de la pile

Dans certains microprocesseurs, ces modes d'adressage peuvent être combinés entre eux pour augmenter ultérieurement la puissance des instructions. Examinons maintenant chacune des méthodes, en introduisant dans les exemples certaines instructions Assembleur qui seront expliquées plus en détail sur la base d'un microprocesseur à 8 bits.

Adressage immédiat. Dans ce mode, la partie adresse de l'instruction contient directement l'opérande : l'instruction*

ADD A, #100

additionne la valeur 100 au contenu de l'accumulateur indiqué par le code A. Le symbole # a été adopté pour indiquer que le chiffre 100 est directement l'opérande et non son adresse. Dans le schéma ci-dessous, on trouve la forme typique d'une instruction d'adressage immédiat, qui utilise donc directement comme opérande une valeur explicitement mentionnée (dans ce cas, 100).
L'exécution d'une instruction de ce type est très rapide, car l'UC, une fois l'instruction chargée, n'a pas à accomplir d'accès en mémoire pour prélever l'opérande.

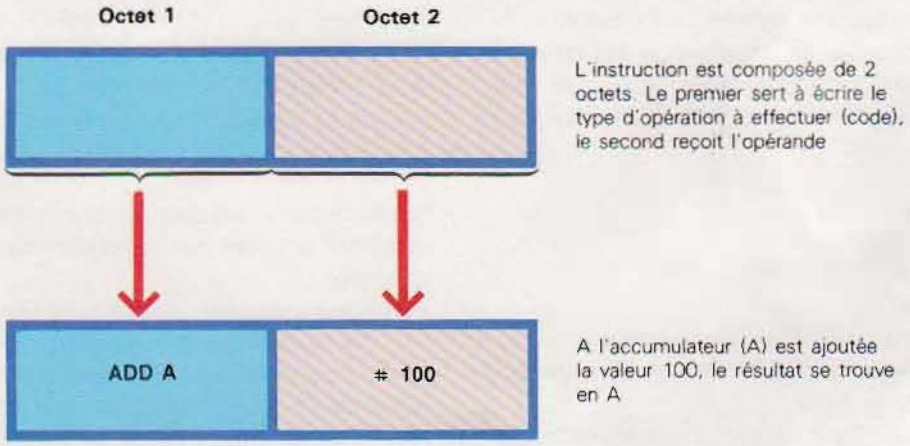
Adressage absolu ou direct. Avec ce type d'adressage, l'instruction contient l'emplacement en mémoire de l'opérande. Par exemple, l'instruction

ADD A,/100

ajoute à l'accumulateur la donnée contenue à l'adresse de mémoire 100, en déposant le résultat dans l'accumulateur même, comme le montre le schéma du haut de la page ci-contre.
Le symbole / est utilisé pour identifier l'adressage direct. Normalement, ces instructions demandent 3 octets : 8 bits pour le code opérateur et 16 bits pour spécifier une adresse parmi les 64 K disponibles en mémoire. Avant de prélever de la mémoire la donnée, l'UC effectue deux autres accès pour charger les 16 bits représentant l'adresse.

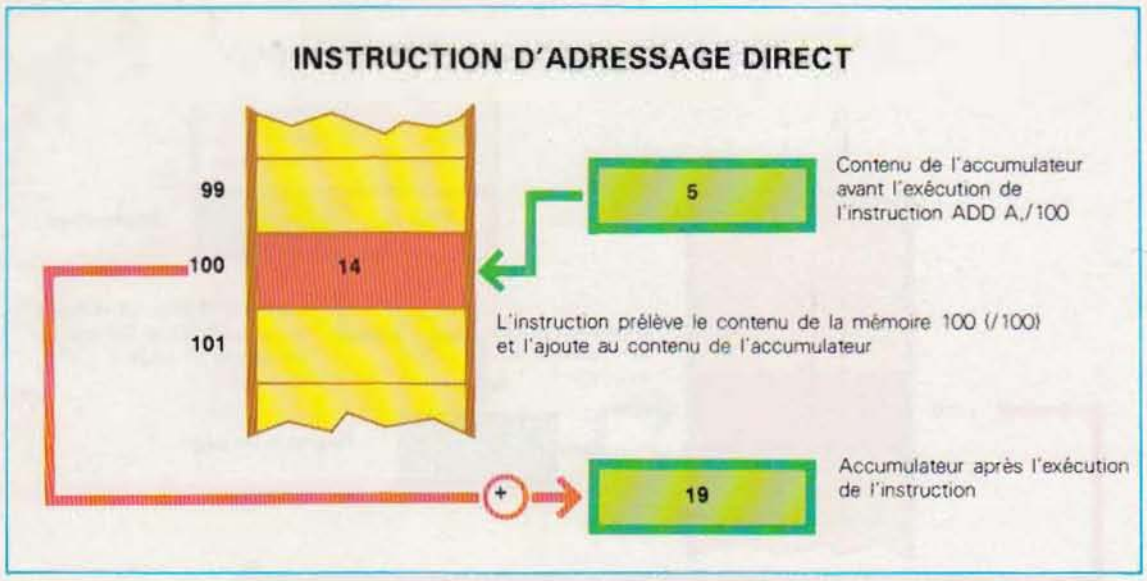
*Les codes mnémoniques qui sont utilisés dans les exemples ont un caractère indicatif. Pour s'en servir réellement, il faut les réécrire dans la forme prévue pour le type particulier de microprocesseur utilisé.

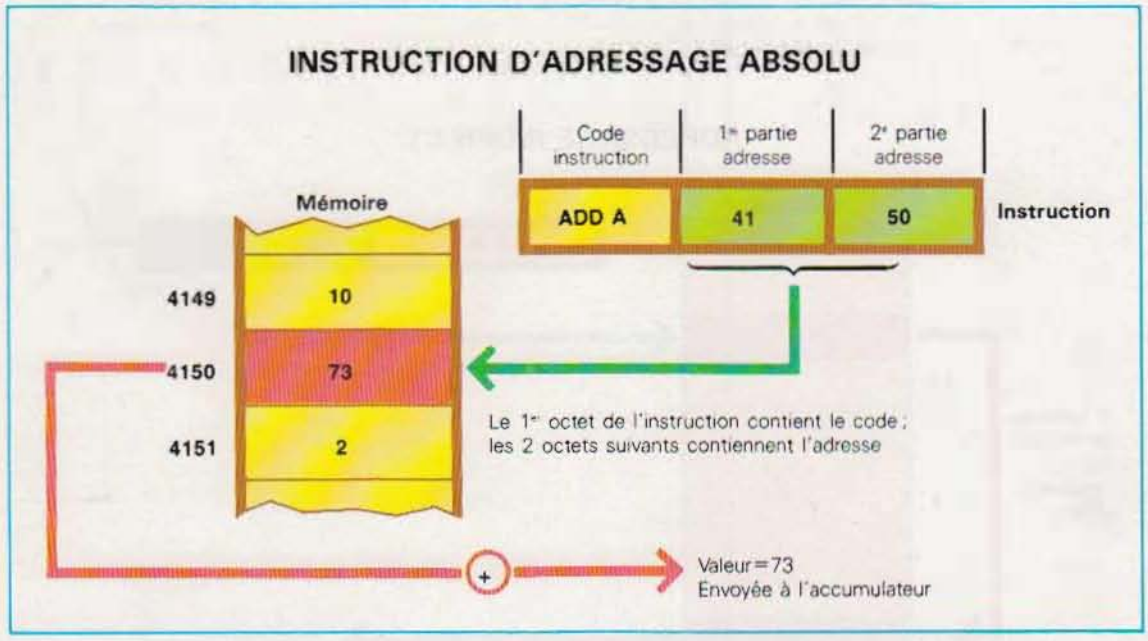
INSTRUCTION AVEC ADRESSAGE IMMEDIAT



L'instruction est composée de 2 octets. Le premier sert à écrire le type d'opération à effectuer (code), le second reçoit l'opérande

A l'accumulateur (A) est ajoutée la valeur 100, le résultat se trouve en A

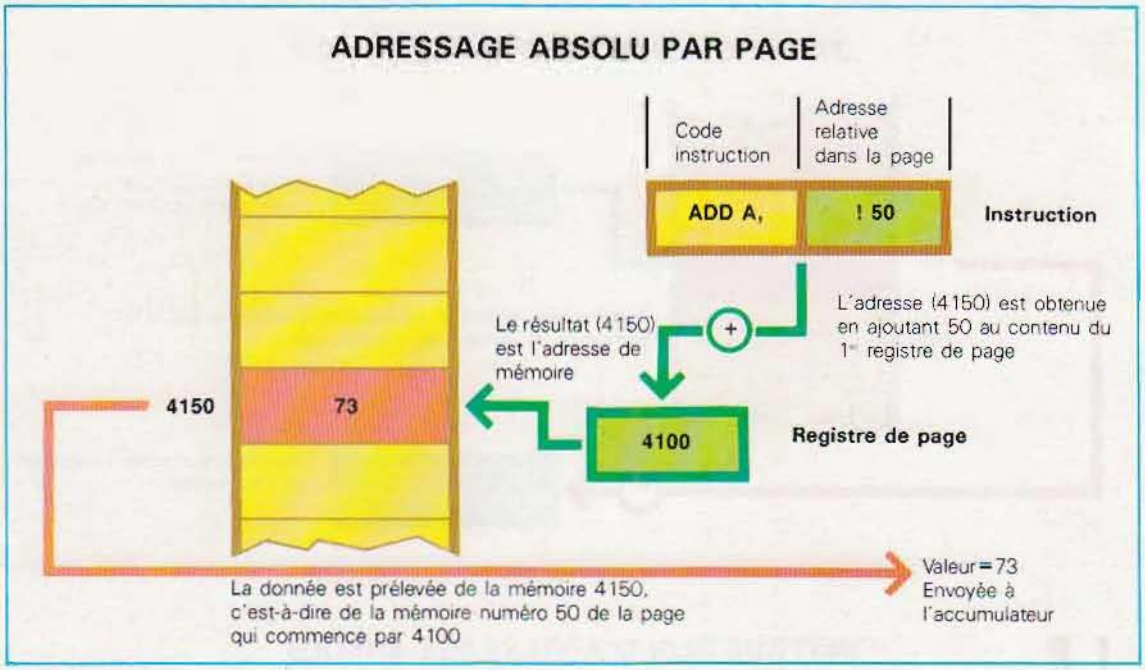


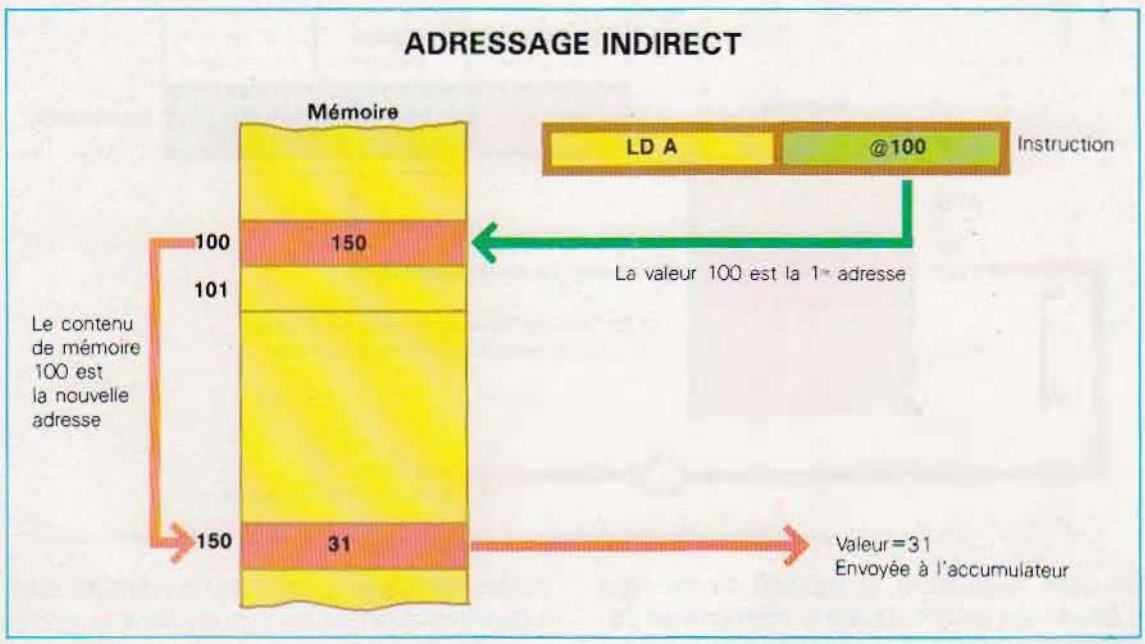


On peut augmenter la rapidité de ce type d'adressage en découpant la mémoire en pages de 256 octets. L'instruction adresse alors un des 256 octets d'une **page**. A cet effet, il existe un registre de page (**Page Register**). Il contient l'adresse de départ de la page qui, ajoutée à l'adresse (relative) spécifique de l'instruction, détermine l'adresse effective de l'opérande (voir page 914, en haut). Dans notre exemple, l'instruction est formée de deux octets. Avant toute opération, la valeur de l'instruction est chargée dans le registre de page. Ce type d'adressage se révèle particulièrement approprié lorsqu'on doit accéder à diverses données contenues dans la même page de mémoire, ce qui évite une éventuelle modification du contenu du registre.

Adressage indirect. Dans ce mode d'adressage, l'adresse désigne un emplacement qui ne contient pas l'information, mais à nouveau une adresse désignant l'emplacement final. Considérons, par exemple, l'instruction :

LD A, @100





Celle-ci chargera, dans l'accumulateur, non pas le contenu de la cellule mémoire d'adresse 100 mais celui de la cellule dont l'emplacement est contenu en cellule d'adresse 100, comme indiqué ci-dessus. La cellule 100 contient l'adresse 150; la donnée chargée dans l'accumulateur sera 31, c'est-à-dire le contenu de la cellule mémoire 150. Le symbole @ a été adopté pour spécifier le mode d'adressage indirect. Cet adressage est moins rapide que l'adressage direct, car l'UC doit effectuer deux accès mémoire successifs avant de prélever la donnée. Toutefois, ce type d'adressage est utilisé lorsque l'on écrit des sous-programmes qui traitent des données stockées dans des zones de mémoire différentes. En adressage indirect, la forme de l'instruction est comparable à celle de l'adressage direct : la longueur de l'instruction est toujours de 3 octets.

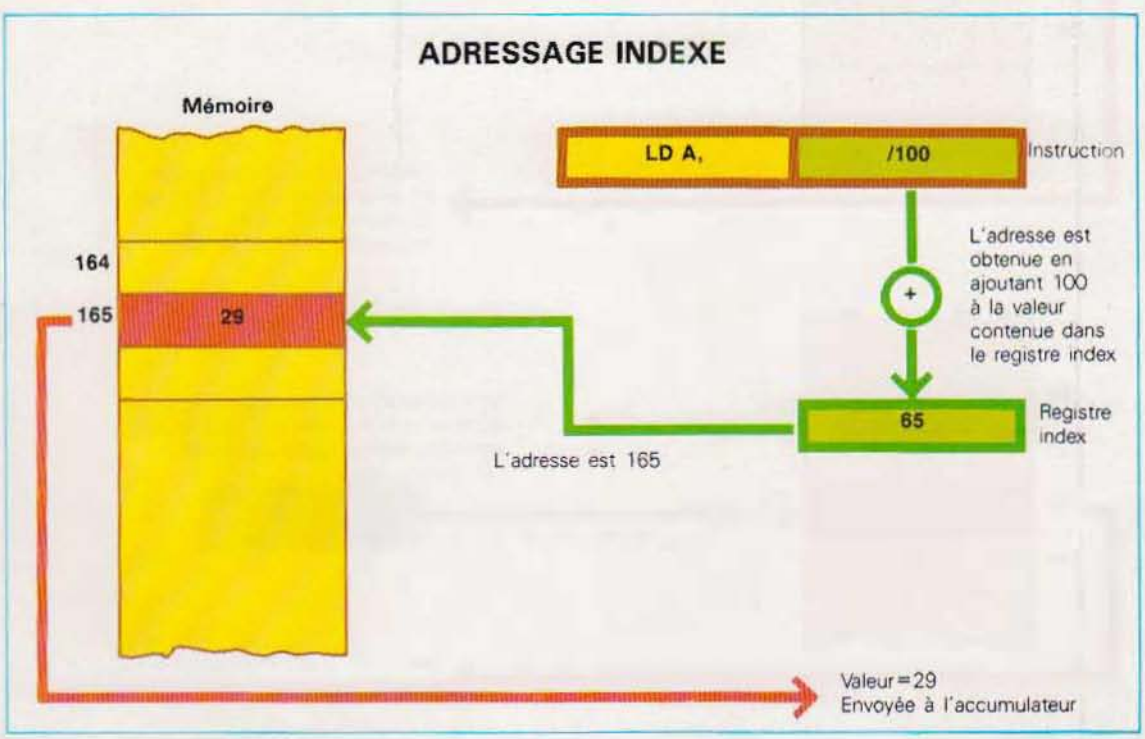
Adressage indexé. Dans ce mode, l'UC additionne l'adresse de base de l'instruction avec le contenu d'un registre particulier, appelé **registre index**, pour calculer l'adresse effective de l'opérande. Le programmeur charge au préalable le registre index (que nous appellerons X) avec la valeur qu'il désire utiliser pour l'indexation avec une instruction du type LD X=3. Lorsque l'instruction

LD A,100(X)

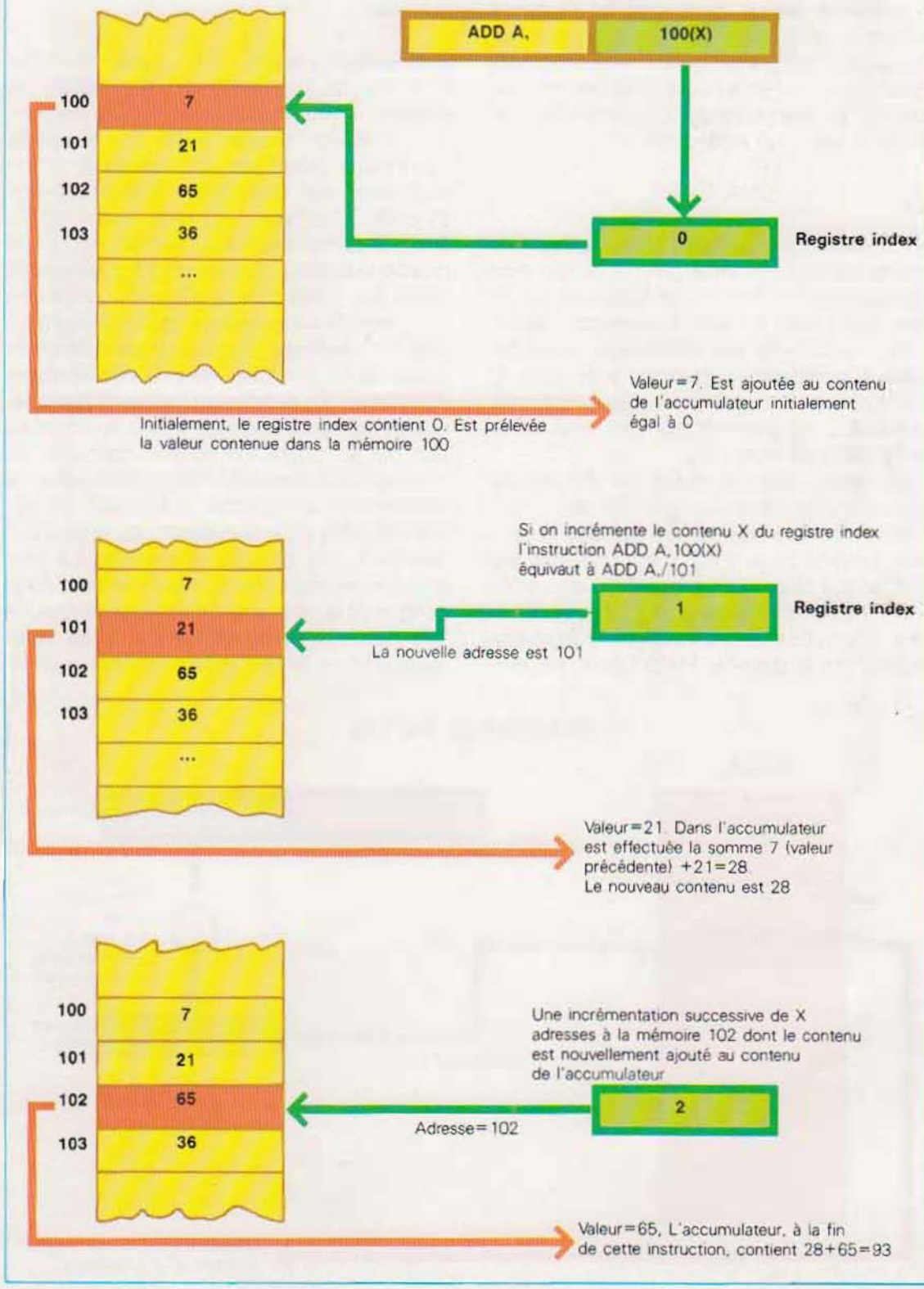
est exécutée, l'UC charge dans l'accumulateur la donnée contenue dans la cellule dont l'adresse est 100+X (voir schéma). Les parenthèses indiquent le mode d'adressage indexé. Il est très proche de l'adressage direct par page. A la différence du registre de page, le registre index ne contient pas nécessairement l'adresse d'une page de mémoire, mais n'importe quelle adresse.

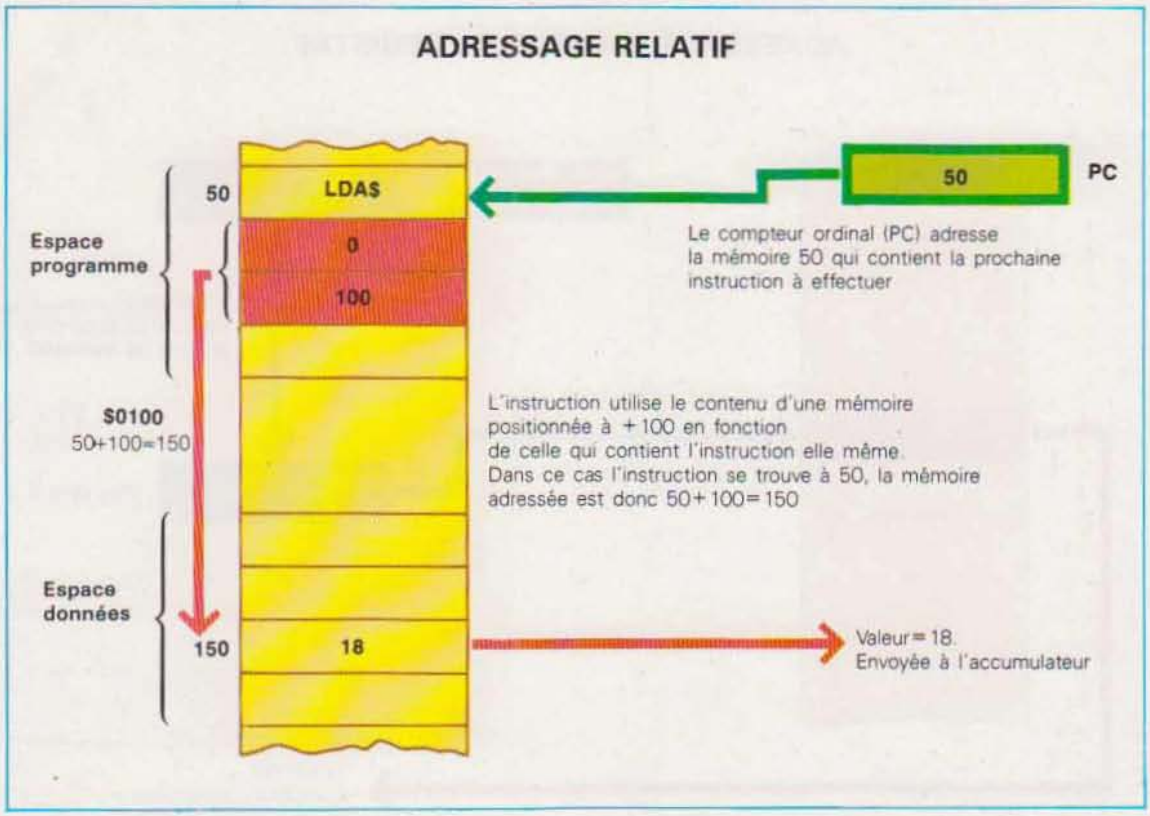
L'adressage indexé se révèle très utile pour le traitement des données organisées en mémoire sous forme de tableaux de matrices (array) - (voir schéma page 916). Il est plus lent que l'adressage direct parce que l'UC doit additionner le contenu du registre index et l'adresse de l'instruction avant d'accéder à la mémoire pour lire la donnée. Malgré cela, les programmes qui utilisent l'adressage indexé semblent plus rapides quand il faut accéder à des données ayant des adresses consécutives en mémoire.

Adressage relatif. Avec cette méthode, l'adresse de l'opérande est déterminée en ajoutant, à l'adresse de l'instruction, la valeur du compteur ordinal (PC). En désignant l'adressage relatif par le symbole $, et en supposant que l'instruction à exécuter soit stockée à l'adresse 50, alors écrire LD A, $ 100 équivaut à écrire LD A,/150, comme le montre le schéma de la page 917. L'adressage relatif est utilisé lorsque l'on veut créer des programmes relogeables, c'est-à-dire qui peuvent être déplacés et exécutés en différentes zones de la mémoire, donc à des adresses différentes. En définissant l'adresse des opérandes de manière relative à celle de l'instruction qui les utilise, on ne doit donc pas les modifier si l'adresse de l'instruction varie. Le mécanisme consiste donc à adresser une cellule mémoire en la localisant par sa position relative à une adresse déterminée. Le programme est alors chargé dans une zone quelconque de la mémoire. En décrivant à chaque fois l'adresse de référence (**base de réallocation**), on évite d'avoir à modifier le code.



ADRESSAGE INDEXE





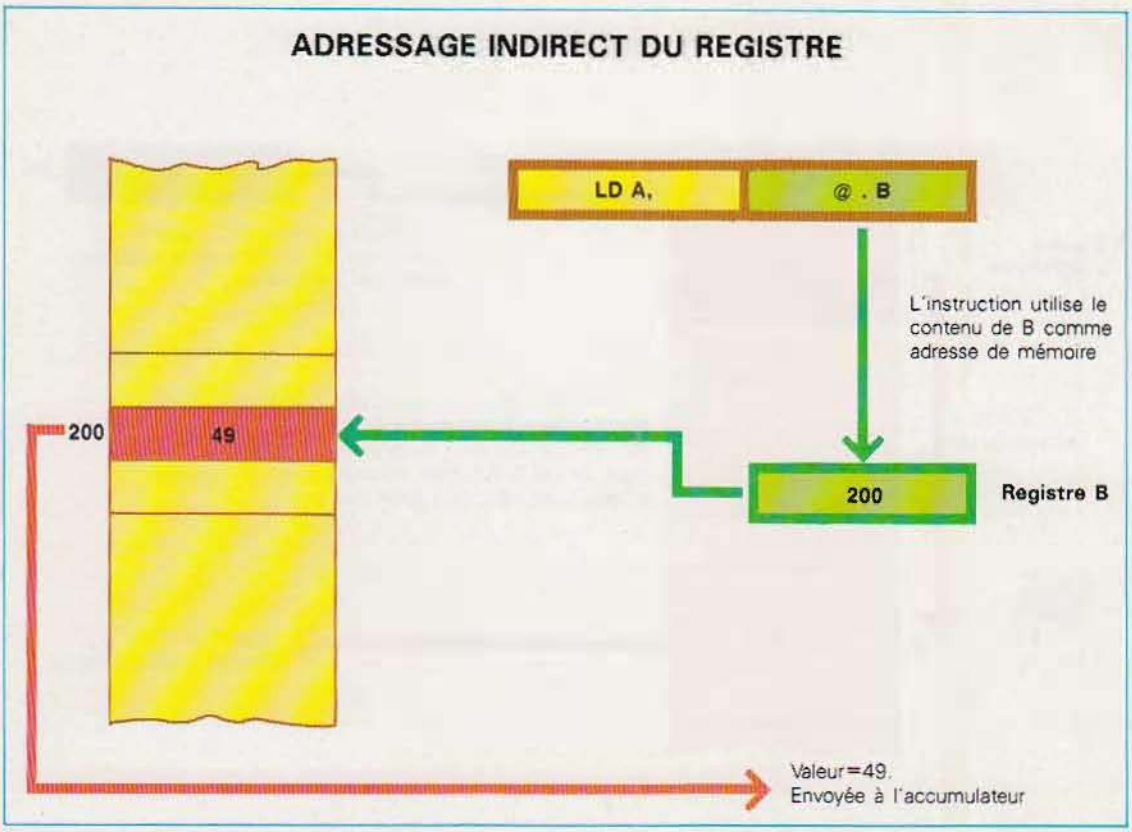
Adressage par registre. On a un adressage par registre quand l'instruction spécifie un registre. Les registres sont contenus dans l'UC et sont en nombre limité.
Cette méthode d'adressage permet d'accroître la vitesse d'exécution des instructions et de réduire la longueur du code. Par exemple, l'instruction

ADD A,.B

ajoute au contenu de l'accumulateur le contenu du registre B. Le symbole utilisé pour distinguer ce type d'adressage est le point. L'exécution d'une telle instruction est très rapide, les accès mémoire n'étant pas nécessaires.
C'est une instruction qui va charger au préalable dans le registre B la valeur qui sera ensuite ajoutée à A.
L'adressage par registre se révèle surtout utile quand une valeur constante doit être appelée de nombreuses fois dans le calcul. Il s'avère intéressant (comme on le voit page 918) quand le registre B ne contient pas directement la valeur de l'opérande mais son adresse. Même alors, l'exécution de l'instruction paraît plus rapide, car on effectue un seul accès à la mémoire.

Adressage par pile. Grâce à ce mode d'adressage, les données utilisées par une instruction sont rangées dans une pile. L'adressage par pile présente de nombreux avantages : en particulier, les instructions sont plus courtes et ont donc un temps d'exécution très bref.
En fait, après chaque accès à la pile, qu'il s'agisse d'une lecture ou d'une écriture, l'UC recalcule automatiquement l'adresse d'entrée de la pile. C'est pourquoi le programmeur ne doit pas prévoir d'instructions pour la mise à jour des adresses. L'adressage par pile est, lui aussi, employé de manière directe et indirecte.

En général, une méthode d'adressage n'est pas utilisable avec n'importe quel type d'instruction. Cette possibilité n'est prévue que dans certaines formes particulières de code binaire, appelées **codes orthogonaux**.
Ces codes sont peu usités, les constructeurs préférant spécialiser les instructions.



Les instructions Assembleur

Nous avons maintes fois signalé que les instructions Assembleur permettent de transmettre au système des commandes et des instructions en rapport univoque avec les codifications hexadécimales propres au langage machine, en utilisant des codes mnémoniques (alphabétiques et numériques). •

En effet, le code binaire d'une instruction donnée est unique et normalement non modifiable par l'utilisateur. Il est, en effet, écrit et figé à l'intérieur de l'UC (plus précisément dans la mémoire de contrôle) par le constructeur, de manière identique pour tous les microprocesseurs du même type. A l'inverse, la codification mnémonique Assembleur est modifiable. Cependant, chaque instruction sera toujours traduite dans les codifications binaires prévues pour ce microprocesseur.

Techniquement, rien ne justifie l'existence de codes mnémoniques propres à chaque type de microprocesseur. Malheureusement, les constructeurs créent leurs codes particuliers selon le microprocesseur. L'absence de norme entre les divers Assembleurs disponibles est uniquement liée à des motifs de caractère commercial. Cela entraîne des problèmes en phase d'écriture des programmes. Ainsi, il est souvent nécessaire de réécrire complètement des routines déjà développées, uniquement parce que le microprocesseur utilisé a changé. C'est pourquoi l'Institut d'ingénierie électrique et électronique (IEEE, Institute of Electrical and Electronics Engineering, Task T 694/D) a recommandé un standard pour la mise en forme des instructions.

Le langage Assembleur proposé par l'IEEE, auquel nous ferons référence par la suite, peut parfois sembler très complexe. Il définit les règles précises à respecter et, dans certains cas, devient redondant. Il faut cependant avoir à l'esprit qu'à ce niveau, la "compacité" excessive d'un langage rend souvent illisible le langage lui-même, alors que si des règles communes étaient définies pour toutes les instructions, on pourrait sans conteste améliorer la lisibilité des programmes en dépit de leur lourdeur. Dans tous les cas de figure, le programmeur doit avoir une connaissance approfondie du microprocesseur sur lequel il travaille.

En effet, toutes les instructions existantes ne sont pas disponibles dans une UC. Un langage d'assemblage comporte les types d'instructions suivants :

1 / Instructions conditionnelles
2 / Instructions arithmétiques
3 / Instructions logiques
4 / Instructions de transfert de données
5 / Instructions de branchement (branch)
6 / Instructions de saut d'une instruction (skip)
7 / Instructions d'appel d'un sous-programme
8 / Instructions de retour à la fin d'un sous-programme
9 / Instructions diverses

Tous les codes mnémoniques respectent la règle de base qui décrit la formation de chaque mot-instruction à partir des initiales de mots de l'action en cours. Par exemple,

ADDC (Add with Carry) = addition dans l'accumulateur avec retenues.
BGT (Branch if Greater Than...) = branchement si le contenu de l'accumulateur est plus grand que...

En cas de besoin, il est même spécifié dans le code mnémonique le type d'opérande auquel renvoie l'instruction.

B (pour Byte) si l'opérande est un mot (octet)
H (pour Half) si l'opérande est un demi-mot
L (pour Long) si l'opérande est un double mot
D (pour Decimal) si l'opérande est décimale
F (pour Float) si l'opérande est en virgule flottante
I si l'opérande est un bit
G si l'opérande est un mot de 4 bits

Si la déclaration n'est pas donnée, l'opérande est un mot de 16 bits.

Instructions de test

Ce sont les instructions qui contrôlent l'exécution du programme en testant si une condition donnée est vérifiée ou non. Celle-ci est généralement définie par les bits du registre d'état. Elle se traduit par la mise à 1 ou 0 (vrai ou faux) d'un des bits zéro (Z), retenue (C), dépassement (O ou V), signe (N), parité (P) ou de l'ensemble de leurs combinaisons.

Les instructions de branchement (Branch), le saut d'une seule instruction (Skip) ou l'appel à un sous-programme (Call) sont conditionnés par un des événements suivants :

Z (Zéro) : l'instruction qui suit le test est exécutée si le bit de zéro a la valeur logique 0, c'est-à-dire si le résultat de la dernière opération effectuée est zéro.
NZ (Non Zéro) : l'instruction est exécutée seulement si le bit de zéro a la valeur logique 1, donc si le résultat de la dernière opération effectuée n'est pas zéro.

D'autres instructions sont associées à des opérations de comparaison. Ce type de test se base toujours sur les valeurs des bits du registre d'état par l'intermédiaire d'opérateurs logiques.
Du fait de leur complexité, on retrouve ces instructions seulement dans les microprocesseurs très évolués. Sont alors disponibles des opérateurs tels que égalité (E, Equal), différence (NE, Not Equal), plus grand que (GT, Greater Than), etc.

Utilisation d'un langage symbolique adapté à la gestion du trafic ferroviaire.



K. Berger/Marka

TESTS CONDITIONNELS

Condition	Code mnémonique
Zero	Z
Not Zero	NZ
Carry	C
Not Carry	NC
Négative	N
Positive	P
Overflow	V
Not Overflow	NV
Equal	E
Not Equal	NE
Greater Than	GT
Less Than	LT
Greater or Equal	GE
Less or Equal	LE
Higher	H
Lower	L
Not Higher	NH
Not Lower	NL

Dans le tableau ci-contre sont résumés les codes de certains tests conditionnels. Rappelons que les actions dépendent du microprocesseur utilisé. Mais il est encore plus important de rappeler que tous les bits du registre d'état ne sont pas toujours modifiés par une instruction. Dans les instructions de branchement en particulier (Branch ou Skip), il est indispensable de savoir quels ont été les bits modifiés à la suite de l'exécution de l'instruction précédente. Il existe généralement, dans les manuels de référence Assembleur, des tableaux spéciaux qui indiquent, instruction par instruction, quels bits sont modifiés, lesquels demeurent inaltérés et ceux qui sont indéfinis. Un exemple est donné dans le tableau ci-dessous.

Instructions arithmétiques

Dans le tableau de la page 921 sont énumérées certaines instructions arithmétiques qui, normalement, se retrouvent dans les langages d'assemblage. Elles permettent d'effectuer

EXAMEN DES CODES DE CONDITION (Tableau théorique)

Opération	Registre d'état						Type d'opération
	C	Z	O	N	H	P	
ADD	*	*	*	*	*	*	Somme
AND	0	*	0	*	–	*	ET logique
LD	–	–	–	–	–	–	Chargement de registre
SHL	*	*	0	*	U	*	Décalage gauche
ST	–	–	–	–	–	–	Mémorisation

*** Positionné selon le résultat de l'opération**
– Pas de modification due à l'opération
0 De toute façon positionné à zéro
U Indéfini après l'opération (c'est-à-dire modifié, mais sans signification)

QUELQUES INSTRUCTIONS ARITHMETIQUES

Instruction	Code mnémonique	Description
Addition (Add)	ADD	Exécute une addition
Addition avec retenue (Add with carry)	ADDC	Addition avec retenue
Soustraction (Substract)	SUB	Soustraction
Soustraction avec retenue (Substract with carry)	SUBC	Soustraction avec retenue provenant d'une opération précédente
Incrémentation (Increment)	INC	Ajoute 1 à l'opérande
Décrémentation (Decrement)	DEC	Soustrait 1 à l'opérande
Multiplication (Multiply)	MUL	Multiplication
Division (Divide)	DIV	Division
Comparaison (Compare)	CMP	Comparaison entre 2 opérandes
Négation (Négate)	NEG	Exécute un complément à 2 de l'opérande

toutes les opérations d'addition avec et sans retenue (respectivement ADDC et ADD). Les opérations avec retenue sont utilisées quand il est nécessaire d'avoir une plus grande capacité, pour additionner des nombres représentés par plusieurs octets.

Les instructions d'incrémentation ou de décrémentation ajoutent ou soustraient 1 à l'opérande spécifiée. Par exemple, l'instruction INC A augmente de 1 le contenu de l'accumulateur. Ces instructions sont très utiles pour la mise à jour rapide des compteurs.

Les instructions de multiplication (MUL) et de division (DIV) ne sont présentes que dans les microprocesseurs les plus sophistiqués (généralement ceux à mots de 16 bits). Les opérations correspondantes ne sont pas réalisées par l'intermédiaire de circuits spécifiques, mais par des signaux de contrôle envoyés par l'unité de contrôle à l'additionneur et au complémenteur.

Dans l'instruction de comparaison, on compare deux opérandes. Son exécution comporte une soustraction entre les opérandes. En fonction de l'examen du bit de zéro, on conclut à l'égalité ou à la différence des deux opérandes.

L'opération de complément à 2 est effectuée par l'intermédiaire d'un circuit spécifique. Il est possible de l'exécuter avec l'instruction NEG. Par exemple, en introduisant NEG A, on complémente à 2 le contenu de l'accumulateur. Si ce contenu est $(4)_{10}$,

MSB LSB

0	0	0	0	0	1	0	0

Accumulateur

L'exécution de l'instruction NEG A nous donne

64321 6 8 4 2 1

1	1	1	1	1	1	0	0

Accumulateur

c'est-à-dire $(-124)_{10}$, le bit le plus significatif (MSB) étant celui du signe.

Instructions logiques

Les principales instructions logiques des langages Assembleur sont listées pages 923 et 924. ET (AND) est la plus usitée. Elle permet d'effectuer des opérations de masquage : sélectionner certains bits d'un mot par l'intermédiaire d'un masque pour examiner ensuite leur valeur. Supposons que nous voulions examiner la valeur logique (0 ou 1) du bit 5 dans l'accumulateur (qui contiendra le résultat). La méthode la plus rapide consiste à effectuer une opération ET entre le contenu de l'accumulateur et une valeur numérique particulière, de façon à mettre en évidence la valeur du bit considéré. Tous les bits du nombre à utiliser (c'est-à-dire le masque) doivent valoir 0 à l'exclusion du bit 5 qui doit conserver la valeur réelle. Dans notre cas, le masque doit être réalisé ainsi :

MSB LSB
7 6 **5** 4 3 2 1 0 = n° des bits

0	0	**1**	0	0	0	0	0

Si l'accumulateur contient au départ :

7 6 5 4 3 2 1 0

1	1	1	0	1	1	0	0

et que l'on exécute un ET entre cette valeur et le masque 0 0 1 0 0 0 0 0, on obtient d'après la table :

1 1 1 0 1 1 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0

On en déduit que le bit 5 vaut 1.

Le schéma de la page 925 illustre le diagramme de déroulement et le programme Assembleur qui réalisent cette opération.
L'instruction OUX (XOR) peut être utilisée pour mettre à zéro l'accumulateur, exploitant le fait que le OU exclusif entre un chiffre quelconque et lui-même vaut 0.

OUX A,A
0 1 0 1 0 0 1 1 valeur (accumulateur)
OUX
0 1 0 1 0 0 1 1 valeur (accumulateur)
0 0 0 0 0 0 0 0 résultat (à l'accumulateur)

L'instruction **LOAD A,0** (charge dans l'accumulateur la valeur immédiate 0) obtient le même résultat (A=0), mais avec des temps plus longs.
Les instructions de déplacement transforment des données de forme série en données de forme parallèle, normalisent des nombres (présentation dans un format standard), compactent des données avant de les mémoriser ou les extraient avant utilisation...
L'instruction **TEST** modifie les bits du registre d'état. Elle effectue une soustraction entre l'opérande spécifiée et la valeur 0. Si le résultat de l'opération n'est pas mémorisé, les bits du registre d'état sont toutefois modifiés en fonction du résultat du test. De cette façon, il est possible de charger les indicateurs (conditionnels) sans modifier les données. L'instruction TEST est analogue à COMPARE (CMP) qui, elle, effectue une soustraction entre deux opérandes (généralement le contenu de l'accumulateur et le contenu d'une cellule mémoire).
De même, l'instruction **COMPARE** ne transfère aucun résultat vers l'accumulateur, mais elle charge les bits du registre d'état utilisables pour conditionner des sauts. Par exemple, l'instruction de branchement conditionnel BZ LOOP (pour Branch if Zero, Branchement si Zéro à l'adresse LOOP), effectue un test sur le bit 0 du registre d'état et saute à l'étiquette LOOP si le bit de zéro vaut 1.

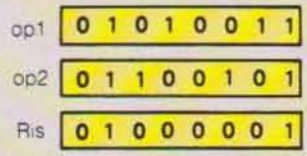
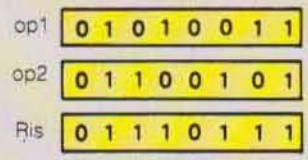
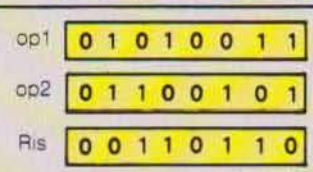
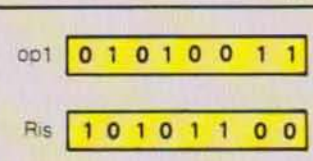
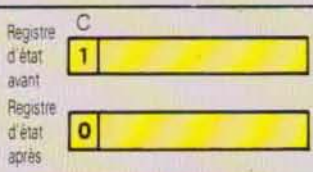
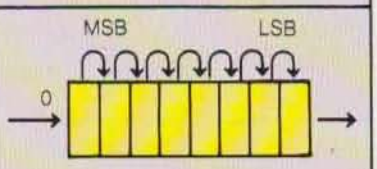
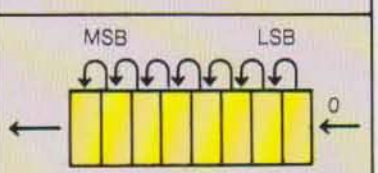
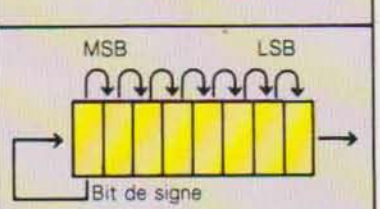
Instructions de transfert des données

Ce groupe d'instructions assure l'échange de données entre le microprocesseur et ses périphériques. Les transferts ont lieu entre la mémoire et les registres du microprocesseur grâce aux instructions **LOAD** (charger) et **STORE** (ranger). La première charge le registre indiqué (première opérande ou destination) avec la valeur donnée comme seconde opérande (source, origine) :

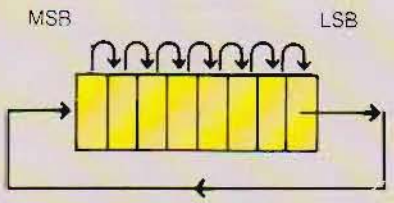
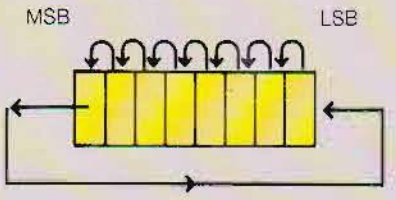
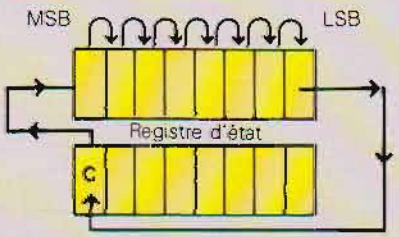
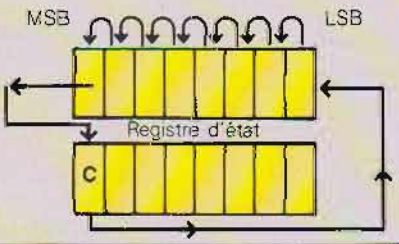
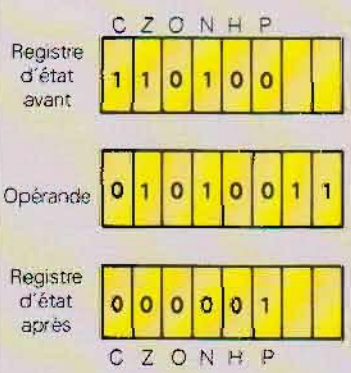
Code instruction	Première opérande	Seconde opérande
LD	A,	0

met à zéro le contenu de l'accumulateur A (première opérande). La seconde opérande indique la valeur à charger (avec effet immédiat, donc sans accès à la mémoire).

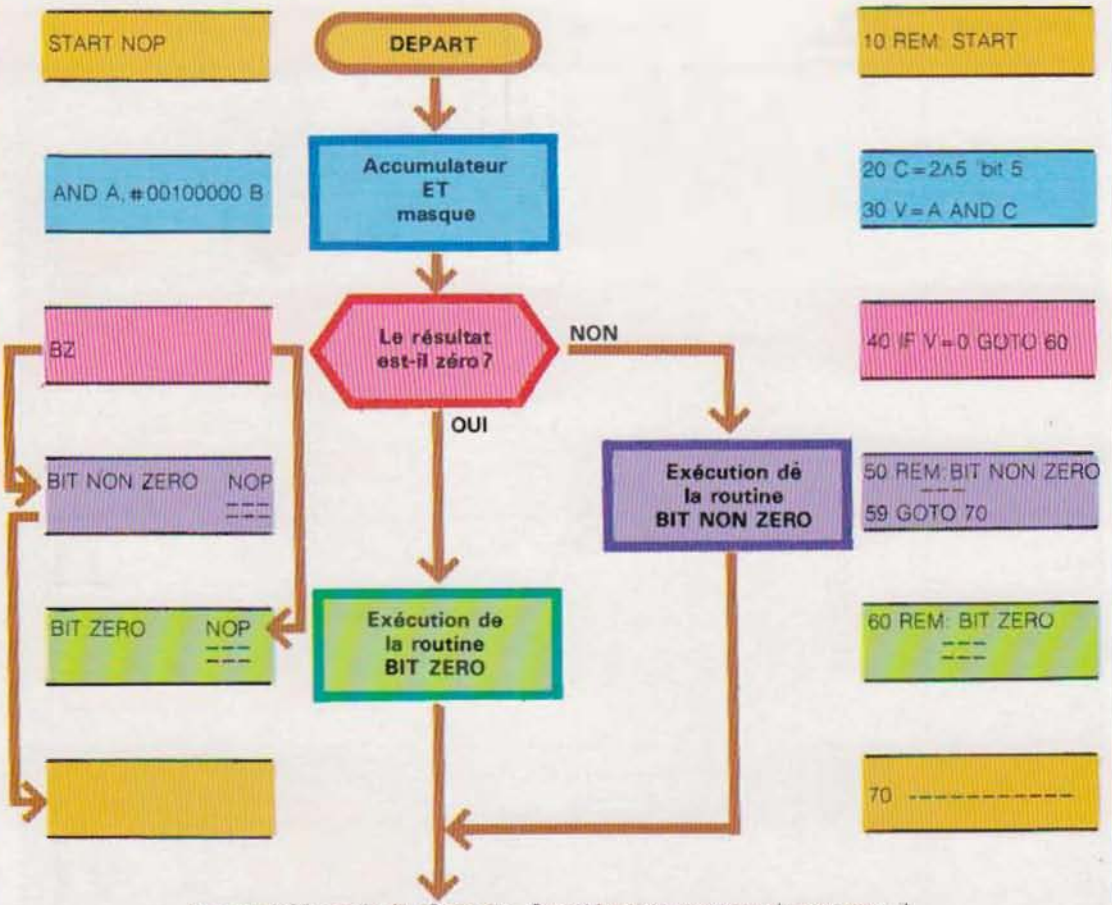
INSTRUCTIONS LOGIQUES

Instruction	Code mnémonique	Description	Exemple
And	AND	Opération de ET logique	
Or	OR	Opération de OU logique	
Exclusive or	XOR	Opération de OUX (exclusif)	
Not	NOT	Opération de NON (complémentation de l'opérande)	
Not carry	NOTC	Complémentation du bit de retenue dans le registre d'état	
Décalage (shift) à droite	SHR	Décalage d'une ou de plusieurs positions vers la droite (vers les LSB) avec mise à zéro des bits les plus significafifs	
Décalage (shift) à gauche	SHL	Les bits de l'opérande sont décalés d'une ou de plusieurs positions vers la gauche (vers les MSB). Les bits les moins significatifs sont mis à zéro	
Décalage (shift) à droite arithmétique	SHRA	Décalage des bits de l'opérande d'une ou de plusieurs positions vers la droite (vers les LSB). Le bit le plus significatif (MSB), c'est-à-dire celui du signe, est maintenu dans sa position mais il est également décalé	

INSTRUCTIONS LOGIQUES

Instruction	Code mnémonique	Description	Exemple
Rotation à droite (Rotate right)	ROR	Décalage des bits de l'opérande de une ou de plusieurs positions vers la droite avec substitution des MSB par les LSB	
Rotation à gauche (Rotate left)	ROL	Décalage de l'opérande d'une ou de plusieurs positions à gauche avec substitution des LSB par les MSB	
Rotation à droite avec retenue (Rotate right through carry)	RORC	Décalage des bits de l'opérande d'une ou de plusieurs positions vers la droite. Le bit est copié dans le MSB tandis que le LSB est chargé dans le bit de retenue	
Rotation à gauche avec retenue (Rotate left through carry)	ROLC	Décalage des bits de l'opérande d'une ou de plusieurs positions vers la gauche. Le bit de retenue est copié dans le LSB tandis que le MSB est mis en retenue	
Test (Test)	TEST	Effectue un examen (test) des bits de l'opérande en modifiant exclusivement les bits du registre d'état. Généralement les bits de retenue ou de dépassement sont mis à zéro. Le bit de demi-retenue n'est pas modifié, tandis que les bits zéro, parité et signe sont modifiés en fonction de l'opérande 	

OPERATEUR ET, ET TEST DE RESULTAT



Le code NOP signifie No Operation. Quand le microprocesseur le rencontre, il n'effectue aucune action (il a une signification semblable à celle du REM en Basic). Dans le schéma, on a reporté l'équivalent du programme en Basic qui a seulement une valeur indicative car, avec le nom symbolique A, on entend une variable quelconque et non l'accumulateur comme c'est le cas en Assembleur

L'instruction STORE, symétrique de LOAD, transfère le contenu d'un registre vers une case mémoire spécifiée par une adresse ou un nom symbolique. L'Assembleur traduit le symbole par une adresse en exécutant les directives données par le programmeur dans son programme. Par exemple, l'instruction

ST A,/100H
(charge le contenu de A dans la cellule d'adresse hexadécimale 100)

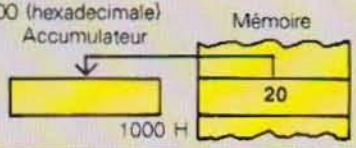
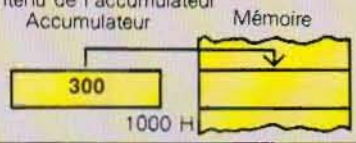
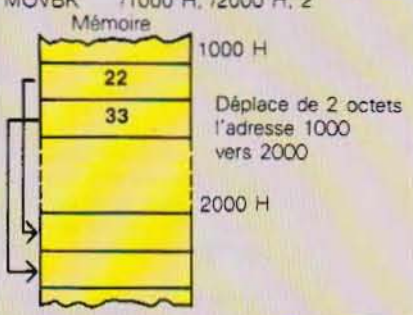
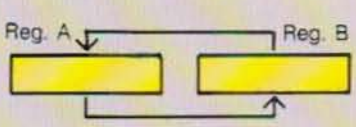
transfère le contenu de l'accumulateur dans la case mémoire d'adresse hexadécimale*

* Le nombre hexadécimal est signalé par la lettre H.

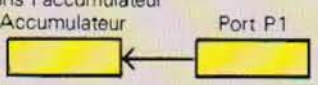
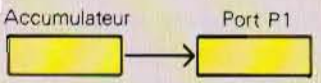
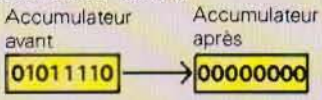
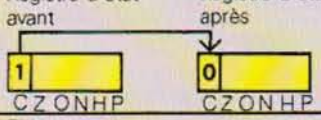
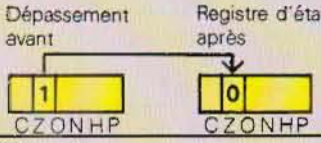
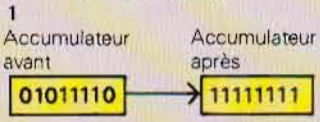
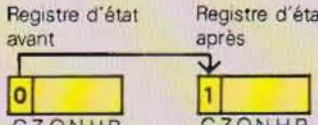
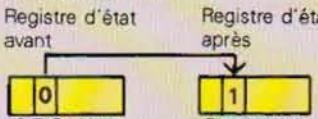
100 (adressage absolu). Les principales instructions de transfert sont résumées en pages 926 et 927.

Les instructions de déplacement de bloc (MOVBK pour Move Block) et de déplacements multiples (MOVM) transfèrent des blocs de données, ainsi que des octets simples ou doubles. Ces opérations, parmi les plus complexes, ne sont disponibles que sur les UC les plus évoluées. Elles sont très importantes pour le programmeur, car elles améliorent les performances d'une machine, surtout dans les opérations d'E/S. L'instruction d'échange (XCH) exécute un double déplacement de données, donc un échange de contenus entre deux opérandes. Comme, par

INSTRUCTIONS DE TRANSFERT DE DONNEES

Instructions	Code mnémonique	Description	Exemple
Chargement registre (Load)	LD	Le contenu d'une cellule mémoire spécifiée comme origine est transféré dans un registre spécifié comme destination	LD A, /1000 H Charge, dans A, le contenu de la cellule mémoire d'adresse 1000 (hexadecimale) 
Mémorisation (Store)	ST	Le contenu d'un registre spécifié comme origine est transféré dans une cellule mémoire spécifiée comme destination	ST A, /1000 H Charge, dans la cellule d'adresse 1000, le contenu de l'accumulateur 
Transfert (Move)	MOV	Le contenu d'un registre mémoire est transféré dans un autre registre mémoire	MOV A, B A origine, B destination 
Transfert en bloc (Move block)	MOVBK	Cette instruction provoque le transfert d'un bloc de données	MOVBK /1000 H, /2000 H, 2 
Transferts multiples (Move multiple)	MOVM	Le contenu d'une cellule mémoire est copié dans plusieurs cellules mémoire	
Echange (Exchange)	XCH	Les registres spécifiés échangent leur contenu	XCH A, B Echange de contenu entre A et B 

INSTRUCTIONS DE TRANSFERT DE DONNEES

Instruction	Code mnémonique	Description	Exemple
Input	IN	La donnée, présente sur le port d'entrée, est transférée dans un registre ou une position mémoire	IN A, (P1) La donnée, lue sur le port d'entrée P1, est rangée dans l'accumulateur 
Output	OUT	Le contenu d'un registre ou d'une position mémoire est transféré vers un port de sortie	OUT (P1), A Le contenu de l'accumulateur est envoyé au port P1 
Clear	CLR	L'opérande spécifiée est mise à 0	CLR A Le contenu de l'accumulateur est à 0 
Clear carry	CLRC	Le bit de retenue est mis à 0 (condition fausse ou retenue inexistante)	CLRC Le bit de retenue est mis à 0 
Clear overflow	CLRV	Le bit de dépassement est mis à 1 (condition vraie, dépassement existant)	CLRV Dépassement à 0 
Set	SET	L'opérande spécifiée est remplacée par une série de "1"	SET A Contenu de l'accumulateur à 1 
Set carry	SETC	Le bit de retenue est mis à 1 (condition vraie, retenue existante)	SET C Retenue à 1 (C) ← 1 
Set overflow	SETV	Le bit de dépassement est mis à 0 (condition fausse ou dépassement inexistant)	SET V Dépassement à 1 (V) ← 1 

exemple, d'échanger les contenus des registres XL et du couple de registres DE :

XCH DE, XL

Les instructions d'échange utilisent généralement comme opérande un des registres du microprocesseur. Les instructions d'E/S exécutent des opérations analogues aux transferts de données, sauf que la source (ou la destination) est un port (d'entrée ou de sortie) ou une portion de mémoire. Pour cette raison, certains microprocesseurs traitent les ports d'E/S comme des pointeurs de mémoire et ne prévoient pas d'instructions spécifiques d'E/S. Seul inconvénient rencontré dans ces opérations de transfert en mémoire : la lenteur de l'exécution des instructions. Les instructions d'E/S ne sont parfois pas fournies, parce qu'elles exigent des signaux de contrôle supplémentaires pour les entrées/sorties (en plus du code machine qui identifie l'instruction). Ces signaux utilisent une ou plusieurs broches (pins) du microprocesseur, leur nombre étant généralement limité à 40.

Les instructions d'effacement sont utilisées pour metre à zéro de manière immédiate et directe un registre, une cellule mémoire ou des bits d'états particuliers : effacement retenues (Clear Carry) ou effacement de dépassement (Clear Overflow). Les instructions SET sont à mettre en parallèle avec les précédentes. Elles chargent tous les 1 dans un registre ou dans une cellule mémoire et mettent à 1 un bit du registre d'état. Ces instructions particuliè-

INSTRUCTIONS DE SAUT

Instruction	Code mnémonique	Description
Branchement si zéro (Branch if Zero)	BZ	Branchement si le résultat de la dernière opération est 0, c'est-à-dire si le bit zéro du registre d'état est à 1
Branchement si pas zéro (Branch if Not Zero)	BNZ	Branchement si le résultat de la dernière opération est différent de 0, c'est-à-dire si le bit de zéro du registre d'état est 0
Branchement si égal (Branch if Equal)	BE	Même opération que pour le branchement if Zero
Branchement si différent (Branch if Not Equal)	BNE	Même opération que pour le branchement if Not Zero
Branchement si retenue (Branch if Carry)	BC	Branchement si le résultat de la dernière opération a une retenue, c'est-à-dire si le bit de retenue du registre d'état vaut 1
Branchement si pas de retenue (Branch if Not Carry)	BNC	Branchement si le résultat de la dernière opération n'a pas de retenue, c'est-à-dire si le bit négatif (signe) du registre d'état vaut 0
Branchement si positif (Branch if Positive)	BP	Branchement si le résultat de la dernière opération est positif, c'est-à-dire si le bit négatif (signe) du registre d'état vaut 0
Branchement si négatif (Branch if Negative)	BN	Branchement si le résultat de la dernière opération est négatif, c'est-à-dire si le bit négatif (signe) du registre d'état vaut 1
Branchement si dépassement (Branch if Overflow)	BV	Branchement si le résultat de la dernière opération provoque un dépassement, c'est-à-dire si le bit de dépassement (V ou O) du registre d'état vaut 1
Branchement si pas de dépassement (Branch if Not Overflow)	BNO	Branchement si le résultat de la dernière opération ne provoque pas de dépassement, c'est-à-dire si le bit de dépassement (V ou O) du registre d'état vaut 0

res à certains microprocesseurs ne sont pas indispensables car il est possible d'obtenir le même résultat avec une ou plusieurs instructions ordinaires du type LOAD ou STORE. L'intérêt des instructions "spécialisées" réside évidemment dans leur rapidité d'exécution.

Instructions de branchement (Branch)

Une instruction de branchement ou de saut donne l'ordre au microprocesseur de passer d'un point à un autre du programme. Le saut interrompt un déroulement séquentiel normal. L'UC exécute alors une autre partie du programme.
En d'autres termes, l'instruction de branchement charge l'adresse où on veut aller dans le compteur ordinal (PC), et le microprocesseur continue l'exécution à partir de l'instruction se trouvant à cette nouvelle adresse. Un branchement peut être conditionnel ou inconditionnel, c'est-à-dire dépendant ou non d'un test. Un branchement (ou saut) inconditionnel qui place une nouvelle valeur dans le PC est déconseillé parce qu'il altère le déroulement logique du programme sans motif apparent, alors qu'un branchement lié à une condition validée ou non rend plus logique et plus lisible le programme.
Cette idée sera reprise plus complètement au chapitre de la programmation structurée qui évite le recours aux instructions de branchement inconditionnel. Pour conditionner un branchement, on utilise les bits du registre d'état (indicateurs d'état) qui sont modifiés en fonction des résultats des diverses instructions.

Branchement si plus grand) (Branch if Greater Than)	BGT	Branchement s'il existe une relation de supériorité c'est-à-dire si les bits du registre d'état vérifient la relation logique suivante : NORV OR (NOTZ)+(NOTN) OR (NOTV) OR (NOTZ)=1
Branchement si égal ou supérieur (Branch if Greater)	BGE	Branchement si la relation "supérieur ou égal" est vraie, c'est-à-dire si les bits du registre d'état vérifient la relation logique (NORV) AND [(NOTN)OR(NOTV)]
Branchement si inférieur (Branch if Less Than)	BLT	Branchement si la relation "inférieur" est vraie, c'est-à-dire si parmi les bits du registre d'état la relation logique [NOR(NOTV)] AND [(NOTN) ORV]=1 est vérifiée
Branchement si inférieur ou égal (Branch if Less Than or Equal)	BLE	Branchement si la relation "inférieur ou égal" est vraie, c'est-à-dire si en examinant les bits du registre d'état on constate que la relation logique Z AND [NOR(NOTV)] AND [(NOTN)OR V]=1 est vérifiée
Branchement si "plus grand" (Branch if Higher)	BH	Branchement si la relation "plus grand" est vraie sans tenir compte des signes des nombres, c'est-à-dire si la relation logique (NOTC) OR (NOTZ)=1 est vérifiée
Branchement si "pas plus grand" (Branch if Not Higher)	BNH	Branchement si la relation "pas plus grand" est vraie sans tenir compte des signes des nombres, c'est-à-dire CANDZ=1
Branchement si plus petit (Branch if Lower)	BL	Branchement si la relation "pas plus petit" est vraie sans tenir compte des signes de nombres, c'est-à-dire si C=1. Même fonctionnement que Branch if Carry
Branchement si pas plus petit (Branch if Not Lower)	BNL	Branchement si la relation "plus petit" est vraie sans tenir compte des signes des nombres, c'est-à-dire C=0. Même fonctionnement que Branch if Not Carry
Branchement si parité paire (Branch if Parity Even)	BPE	Branchement si le bit de parité vaut 1 (parité paire) c'est-à-dire si le nombre des bits positionnés à 1 est pair
Branchement si parité impaire (Branch if Parity Odd)	BPO	Branchement si le bit de parité vaut 0 (parité impaire), c'est-à-dire si le nombre de bits positionnés à 1 est impair

Grâce aux instructions de branchement conditionnel, l'UC répétera une même séquence d'instructions jusqu'à ce qu'une condition soit vérifiée (boucle).
Le branchement est effectué exclusivement quand la condition imposée est vérifiée. Les conditions les plus simples se basent sur la valeur directe des bits d'état comme la retenue (BC, Branch if Carry - Branchement si retenue) ou le zéro (BZ, Branchement si zéro). Généralement sont disponibles soit les instructions qui vérifient si la condition BZ est vraie, soit celles qui leur sont symétriques et pour lesquelles la condition n'est pas vraie. Le programme suivant

```
LD A,3      (charge la valeur 3 dans l'accumulateur)
CMP A,3     (compare l'accumulateur avec la valeur 3)
BZ ZERO     (saute à l'étiquette ZERO si le bit de zéro est positionné à 1)
```

effectue systématiquement le branchement, car la condition est toujours vraie. Inversement, si la dernière instruction était

```
BNZ NON ZERO (saute à l'étiquette NON ZERO si le bit de zéro est positionné à 0)
```

la condition serait toujours fausse et le branchement jamais effectué.
D'autres instructions de branchement sont conditionnées comme si elles étaient précédées d'une instruction de comparaison (CPM) ; il est donc possible d'avoir des opérations comme BE (Branch if Equal - Branchement si égal), BGT (Branch if Greater Than - Branchement si plus grand que)... dans lesquelles le branchement s'effectue seulement si la valeur contenue dans l'accumulateur satisfait à la comparaison.

Instructions de saut (Skip)

Il s'agit de sauts (SKP pour Skip : passer par dessus) conditionnés : si la condition posée est vérifiée, l'instruction qui suit celle de skip est sautée. Une étiquette d'arrivée du saut n'est pas spécifiée : si la condition est vraie, le programme saute seulement l'instruction suivante et reprend l'exécution séquentielle.
Après avoir effectué le test (SKP) dans le programme suivant :

```
          LDA,/100H  (charge dans l'accumulateur le contenu de l'adresse hex. 100)
          SUB A,1    (soustrait 1 à l'accumulateur)
          SKP        (saute l'instrution suivante si le résultat est positif)
          B NEGATIF  (branchement à l'étiquette NEGATIF)
POSITIF   NOP        (routine POSITIF)
          . . .
          B FIN      (sauteàl'étiquetteFIN)
NEGATIF   NOP        (routine NEGATIF)
          . . .
```

le résultat est soit l'exécution de l'instruction suivante (B NEGATIF), soit le saut à l'instruction POSITIF NOP. Un équivalent en Basic peut être le programme suivant :

```
10 A=16 ∧ 2
20 A=A—1
30 IF A>0 GOTO 50
40 GOTO....
50 .....
```

Ou bien :

```
          LD A,/100 H
          SUB A,1
          BN NEGATIF (saute l'étiquette NEGATIF si N vaut 1)
POSITIF   NOP
          . . .
NEGATIF   NOP
          . . .
```

Quand le code Assembleur ne prévoit pas d'instructions de branchement, on obtient donc le même résultat qu'avec une instruction de branchement en recourant à une instruction SKP et à un saut inconditionnel. Généralement, les SKP conditionnelles disponibles sont analogues aux instructions de branchement. La différence entre elles est simple : si la condition est vérifiée, l'exécution se poursuit à l'instruction suivant le saut de manière séquentielle. Les codes de cette classe d'instructions sont analogues à ceux du tableau pages 928-929, en remplaçant la lettre B par SK (mettre Skip à la place de Branch).



C. O'Rear/Grazia Neri-West Light

Ordinateur personnel Apple utilisé en gestion de travail.

Par exemple, l'instruction équivalente à BZ (Branchement si zéro) est SKZ (Skip if Zero - Saut si zéro), celle de BGT (Branchement si plus grand que) est SKGT (Skip if Greater Than - Saut si plus grand que).

Instructions d'appel de sous-programmes et de retour

Les instructions d'appel d'une routine exécutent des sauts. La syntaxe est du type :

CALL LABEL (saut inconditionnel à l'étiquette (LABEL)

L'instruction d'appel peut être conditionnée :

CALLZ LABEL (saute à LABEL si Z=1)
CALLGT LABEL (saute à LABEL si plus grand que...)

Mais à la différence de ce qui se produit dans les instructions de saut (branchement ou skip), le contenu du compteur de programme est sauvegardé avant d'être remplacé par la nouvelle valeur. C'est seulement après cette opération que l'on peut exécuter les instructions se trouvant à l'adresse spécifiée par l'étiquette. Lorsque l'on rencontre une instruction RETURN (instruction de retour, code mnémonique RET), le contenu du compteur ordinal est restauré et l'exécution du programme continue par l'instruction suivant le CALL.
L'instruction RETURN peut être conditionnée comme pour l'instruction CALL :

RETZ (retour si le bit zéro vaut 1)
RETGT (retour si plus grand que...)

L'originalité de ces instructions est donc de sauvegarder et de rétablir le contenu du PC. Ces opérations se produisent par l'intermédiaire de la pile. La façon dont le microprocesseur opère dépend du type d'appel. Si l'appel est conditionné et la condition vérifiée, le contenu du PC est empilé (push) dans la pile ; le contenu de la cellule mémoire suivant le code CALL (c'est-à-dire LABEL) est donc chargé dans le PC et l'exécution se poursuit à l'instruction qui se trouve à la nouvelle adresse chargée dans le PC.

A l'inverse, si la condition spécifiée dans l'appel n'est pas vérifiée, c'est l'instruction suivant séquentiellement l'appel qui est exécutée. Dans le cas d'un appel inconditionnel, on a toujours un branchement à l'instruction dont l'adresse est spécifiée par l'étiquette.
Dans les deux cas, la valeur du PC est restaurée lors de l'exécution de l'instruction RET (retour). Cette opération comporte le transfert dans le PC de l'adresse précédemment sauvegardée dans la pile. Les instructions d'appel à une routine et celles de retour sont résumées ci-dessous.
On peut également appeler une routine à partir d'une autre routine. Dans ce cas, à chaque fois qu'un CALL s'exécute, la valeur du compteur ordinal (PC) est sauvegardée dans la pile. D'une manière analogue, à chaque fois que s'exécute une instruction Return, la valeur au sommet de la pile est restaurée dans le PC.

Instructions diverses

Ci-contre sont reproduites certaines instructions particulières communes aux langages Assembleur et n'appartenant à aucune des classes précédemment examinées.

NOP. Quand le microprocesseur décode l'instruction NOP (No OPeration, aucune opération), aucune fonction ne se déroule pendant la durée d'un cycle machine. Une fois cet intervalle de temps écoulé, l'instruction qui suit la NOP est exécutée. Le rôle de la NOP est de seulement retarder l'instruction suivante. Elle est souvent utilisée pour insérer des commentaires.

INSTRUCTIONS D'APPEL A SOUS-PROGRAMME

Instruction	Code mnémonique	Description
Appel (Call)	CALL	Branchement à l'étiquette spécifiée en sauvegardant le contenu du PC dans la pile
Appel si zéro (Call if Zero)	CALLZ	Si le bit zéro est égal à 1, branchement à l'étiquette spécifiée en sauvegardant le contenu du PC dans la pile
Appel si retenue (Call if Carry)	CALLC	Si le bit de retenue vaut 1, branchement à l'étiquette spécifiée en sauvegardant le contenu du PC dans la pile
Appel si négatif (Call if Negative)	CALLN	Si le bit du signe est égal à 1, branchement à l'étiquette en sauvegardant le contenu du PC dans la pile

INSTRUCTIONS DE RETOUR

Instruction	Code mnémonique	Description
Retour (Return)	RET	Restaure la valeur du PC qui était sauvegardé dans la pile
Retour si zéro (Return if Zero)	RETZ	Restaure la valeur du PC sauvegardé dans la pile uniquement si le bit zéro vaut 1
Retour si retenue (Return if Carry)	RETC	Restaure la valeur du PC sauvegardé dans la pile si le bit de retenue vaut 1
Retour si négatif (Return if Negative)	RETN	Restaure la valeur du PC sauvegardé dans la pile uniquement si le bit de signe vaut 1 (valeur négative)

INSTRUCTIONS DIVERSES

Instruction	Code mnémonique	Description
Aucune opération (No Operation)	NOP	Aucune opération n'est effectuée et le contrôle passe à l'instruction suivante
Empiler (Push)	PUSH	Les opérandes spécifiées sont empilées dans la pile
Dépiler (Pop)	POP	Le premier élément de la pile est chargé dans l'opérande spécifiée
Arrêt (Halt)	HALT	Bloque l'exécution des instructions jusqu'à ce qu'intervienne un événement extérieur (interruption)
Attendre (Wait)	WAIT	Similaire à la précédente mais l'exécution peut reprendre même pour un événement intérieur
Interruption (Break)	BRK	Permet l'exécution d'un programme d'interruption
Adapter (Adjust)	ADJ	Le contenu de l'opérande, généralement l'accumulateur, est réajusté pour représenter le résultat correct BCD. Pour les opérations en BCD, il faut ajouter 6 pour éviter les configurations non admises par le code
Autorisation d'interruption (Enable Interrupt)	EI	Permet les interruptions de programme
Interdiction d'interruption (Disable Interrupt)	DI	Empêche les interruptions de programme jusqu'à ce qu'on autorise à nouveau les interruptions avec l'instruction EI
Traduction (Translate)	TR	Suivant une table (par exemple la table BCD), on obtient par l'intermédiaire d'un indice la valeur correspondant à un élément de la table

PUSH et **POP**. Ces instructions gèrent la pile. Avec PUSH, on empile une valeur dans la pile ; avec POP, on retire de la pile la dernière valeur déposée.

HALT et **WAIT**. Avec les instructions HALT (arrêter) ou WAIT (attendre), on arrête ou on suspend l'exécution du programme jusqu'à ce qu'un événement extérieur, généralement une interruption, débloque le microprocesseur. Ce type d'instruction permet par exemple de suspendre l'exécution jusqu'à ce que l'on soit sûr qu'un périphérique est correctement synchronisé (par exemple pour commencer le transfert d'un bloc de données).

Interruption (Break). L'instruction BRK lance un des programme d'interruption, sans provoquer d'arrêt machine. Dans cette situation également, le contenu du PC est sauvegardé.

Autorisation d'interruption (Enable interrupt) et **Interdiction d'interruption (Disable Interrupt)**. Les instructions EI (Enable Interrupt) et DI (Disable Interrupt) permettent également de gérer les interruptions par l'intermédiaire de l'instruction DI, on interdit les interruptions qui sont autorisées avec l'instruction EI.
De cette façon, aucune cause extérieure ne peut interrompre l'exécution du segment de programme compris entre les deux instructions. Elles sont utilisées pour exécuter en

toute sécurité les opérations les plus délicates, comme le transfert des blocs de données.

Adjust et **Translate**. Les instructions ADJ (ADJust, adapter) et TR (TRanslate, traduire) effectuent des opérations numériques particulières, comme par exemple des transferts de codification.

Les directives d'assemblage

Les directives d'assemblage sont des instructions insérées dans un programme Assembleur. Elle s'assurent que la traduction des codes mnémoniques du programme en langage machine est correcte. Ces directives sont généralement appelées **pseudo-instructions**, car elle ne produisent pas un code machine exécutable par le microprocesseur. Il s'agit plutôt de commandes au programme Assembleur pour définir des symboles, allouer de la mémoire aux programmes et à leurs données, indiquer la fin du programme et le formatage du listing.

Les directives d'assemblage sont aussi représentées par des codes mnémoniques analogues à ceux des instructions. Ils sont toutefois analysés à part, car ils n'ont pas de correspondance avec les codes machine en binaire. Chaque Assembleur doit prévoir (inclure) des routines Assembleur qui exécutent les actions demandées par les directives. Les directives les plus usitées par les programmes Assembleur sont résumées dans le tableau ci-dessous.

DIRECTIVES D'ASSEMBLAGE

Instruction	Code mnémonique	Description	Exemple
Origine (Origine)	ORG	L'opérande associée est positionnée dans le compteur de programme et spécifie la position de mémoire où commenceront le programme, la routine ou les données.	ORG # 100 le programme commence à l'adresse 100
Egalité (Equal)	EQU	Associe, à un symbole, une constante, une adresse ou une expression.	ALPHA EQU 2 partout où se trouve ALPHA on lui substitue la valeur 2
Fin (End)	END	Informe le système qu'il est arrivé à la fin du programme.	END
Page (Page)	PAGE	Provoque la poursuite de l'édition du programme sur une nouvelle page.	PAGE
Titre (Title)	TITLE	Provoque la poursuite de l'édition sur une nouvelle page et le titre associé au code TITLE est imprimé en tête de page.	TITLE "Exemple" va à une nouvelle page et écrit "Exemple"
Mémoire transcrite (Rescrive memory)	RES	Un bloc mémoire est réservé pour pouvoir mémoriser des données par la suite.	VET RES 50 50 octets sont réservés à partir de l'adresse VET
Données (Data)	DATA	Les adresses mémoire spécifiées sont chargées par l'Assembleur avec la valeur donnée lors de l'exécution.	MATR DATA 3 tous les octets de MATR sont chargés avec la valeur 3

Le diagnostic assisté par ordinateur (1)

L'extraordinaire évolution que l'électronique a connue pendant les vingt dernières années n'a pas manqué de produire ses effets dans tous les domaines de la technologie avec l'entrée massive du microprocesseur à tous les niveaux.

C'est sans doute dans les **secteurs hospitaliers** et même, de façon plus large, dans l'ensemble du milieu médical spécialisé que la microélectronique avancée a réellement porté ses fruits. La gamme des appareils biomédicaux intégrant un microprocesseur effectue aussi bien le diagnostic (en utilisant les rayons X, la résonance magnétique nucléaire, les ultrasons, la thermographie) que la thérapeutique (cardiologie et surveillance des patients en réanimation). Les industries qui s'occupent des applications biomédicales de l'électronique et de l'informatique sont, aujourd'hui, dotées d'importants laboratoires de recherche appliquée qui assurent une amélioration constante des appareils médicaux.

L'explosion de la micro-électronique a été suivie d'une évolution aussi rapide des applications biomédicales en général qui, en l'espace de quelques années, a rendu définitivement obsolètes les grandes conquêtes remportées depuis la fin du dix-neuvième siècle jusqu'aux années soixante. La panoplie des aplications technologiques dans le domaine biomédical s'est considérablement élargie avec la naissance d'une nouvelle science appliquée, la **bio-ingénierie**, alors que les outils d'utilisation commune (rayons X par exemple) ont subi des transformations telles qu'ils permettent des analyses et des enquêtes très sophistiquées.

La construction des premiers appareils radiologiques date du début du siècle. Cette technique d'exploration se fonde sur l'absorption sélective, par les différents tissus, d'un faisceau de rayons X.

Dans les premiers appareils, la radiation était produite par un tube cathodique et captée par une plaque photosensible ou sur un écran fluorescent. Le sujet à radiographier était placé entre le tube et la plaque. Les doses intenses de radiation auxquelles étaient exposés les patients n'éveillaient alors aucune préoccupation particulière, leur **toxicité** n'étant alors pas connue.

La conception des premiers appareils capables de radiographier le thorax constitua à l'époque une conquête sans précédent, qui assura un diagnostic complet dans le domaine des infections pulmonaires. Au cours des dernières années, les industries de ce secteur ont développé une vaste gamme d'appareils et d'accessoires, depuis la simple table horizontale jusqu'aux **procédés tomographiques multidirectionnels**, avec chargefilms et développeurs en ligne automatiques. Ce perfectionnement a notamment permis de réduire considérablement les doses de radiation auxquelles étaient soumis les patients, tout en augmentant la qualité de l'image.

Les techniques d'exploration aux rayons X ne se limitent pas aux projections normales de l'organe à examiner. La tomographie est une technique qui permet de réaliser des sections anatomiques, en imposant au tube émetteur des mouvements précis (elliptiques, circulaires, linéaires, épicycloïdaux). Les automatismes intégrés contrôlent le mouvement de la table avec des rotations du sujet jusqu'à 180°. Des accessoires comme les collimateurs automatiques et les sériographes font désormais partie de l'équipement standard des machines.

L'usage de **l'amplificateur de brillance**, du circuit de télévision fermé et de tables entièrement télécommandées préserve l'opérateur de l'exposition aux rayons X. Il travaille dans des locaux complètement isolés des appareils, condition fondamentale pour la radioprotection. Le système de radiodiagnostic de dernière génération s'appelle Diagnost 90 et est construit par Philips. C'est un mastodonte de 1650 kg conçu pour satisfaire les exigences toujours plus grandes des laboratoires. La table est suspendue pour permettre une projection à l'abri d'une paroi. Le socle est enfermé dans un placard métallique contenant également tout l'équipement électronique. Les mouvements d'exploration sont commandés par un microprocesseur qui oriente le faisceau en contrôlant automatiquement la parallaxe, en faisant varier la distance entre la source et l'image. Toutes les commandes de pilotage et de contrôle du positionnement de la table sont dédoublées

et sont placées soit sur le panneau frontal de la même table, soit sur une console placée à distance.

Le système est équipé d'un chargeur de plaques en série et d'un **contrôleur automatique de l'exposition et de la fluoroscopie.** La distance séparant le film de la surface de la table est de 7 cm seulement. Cette caractéristique permet d'opérer dans de parfaites conditions de géométrie, en éliminant totalement les problèmes associés aux émissions secondaires.

Le chargeur accepte des cassettes de film de dimensions différentes. Le centrage se produit automatiquement et l'incidence est commandée par un bouton spécial. Les programmes d'exposition prévoient, en plus, l'option tomographique.

Le cœur d'un système radiologique est le générateur de rayons X. L'appareil est parfois équipé de générateurs dont la puissance varie de 50 à 100 kW. On obtient des temps d'exposition inférieurs à la milliseconde. Le tube radiogène est équipé d'un collimateur automatique et d'amplificateurs de brillance. Il est capable de projeter un faisceau exactement conique, éliminant ainsi de gênantes radiations extrafocales. Le collimateur s'ouvre et se ferme automatiquement en fonction du type de film inséré dans le chargeur, du programme d'exposition sélectionné et de la distance source-film choisie.

Le chargeur en série est complété par un **amplificateur de brillance** doté d'un écran d'entrée de 9 pouces. Les caractéristiques élevées d'absorption de l'écran et les propriétés de transmission de l'image permettent de réduire énormément les doses nécessaires pour obtenir des images radioscopiques nettes. L'image est visualisée à travers un circuit vidéo. Les contrôles voltamétrique et ampérométrique maintiennent au niveau minimum indispensable le flux de radiation à travers l'écran d'entrée. Du même coup, l'exposition du patient se trouve considérablement réduite.

L'image, telle qu'elle paraît sur le moniteur, est automatiquement enregistrée sur film ou sur plaque sensible grâce à une **caméra radiographique** de 105 mm placée sous le panneau antérieur. Elle reproduit, en marge de l'image, le code d'identification du patient.

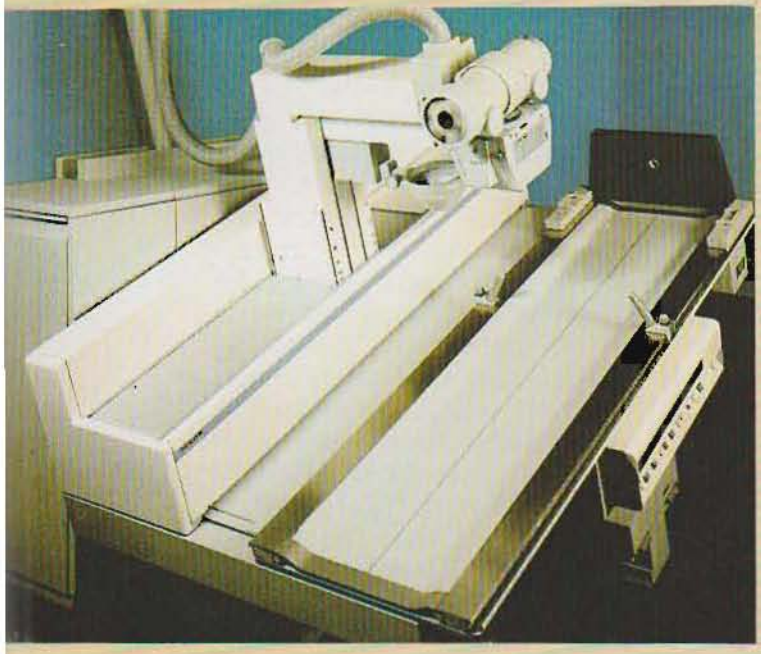
Les possibilités de mouvement du système lors de l'examen sont considérables : examens radiologiques de l'appareil gastro-intestinal et tomographie plane-parallèle (c'est-à-dire effectuée sur un plan parallèle à celui de la table).

Pour réaliser une série de projections rapides de l'œsophage par exemple, un programme spécial combine le mouvement d'exploration du tube radio avec le mouvement de la table, de façon à réaliser un mouvement relatif très rapide (et donc rapidité d'exploration très élevée) sans inconvénient pour le patient.

De cette façon, il devient vraiment possible d'opérer à la **même vitesse** que la progression de déglutition et de suivre ainsi pas à pas le mouvement du produit de contraste. La tomographie et la zonographie plane-parallèle exigent des rotations du tube autour d'un axe perpendiculaire à l'axe du corps et parallèle au plan de la table. Grâce à ce mouvement, les détails sans intérêt sont complètement effacés, ne laissant apparaître qu'une zone étroite, approximativement plane, autour de l'axe de rotation.

A noter que dans un système tel que Diagnost 90, l'option tomographique est complètement automatique et programmable par une simple touche de la console.

Radiodiagnostic informatisé avec le Diagnost 90 de chez Philips.



Philips



Un système de radiodiagnostic assisté par ordinateur : Poly-Diagnost C. A gauche, au premier plan, le bras latéral.

Cette opération impose automatiquement une distance source-image (cassette) de 110 cm. Initialement, le tube émetteur de rayons X est positionné sur l'axe perpendiculaire, de façon à permettre un contrôle correct du centrage à travers l'image radioscopique. Il effectue donc un mouvement de rotation avec retour automatique en position initiale.

Le système de contrôle intégré est commandé à partir d'une console. Commandes et indicateurs contrôlent séparément un ensemble de fonctions.

Les circuits de contrôle de ces fonctions sont montés sur des cartes distinctes et supervisés par un microprocesseur doté d'une mémoire programmée seulement pour la lecture (ROM).

*Les techniques radiologiques de visualisation angiographique ont profité, comme d'ailleurs les autres méthodes d'investigation radiologique, des progrès réalisés dans le domaine de l'**automatisme industriel**.*

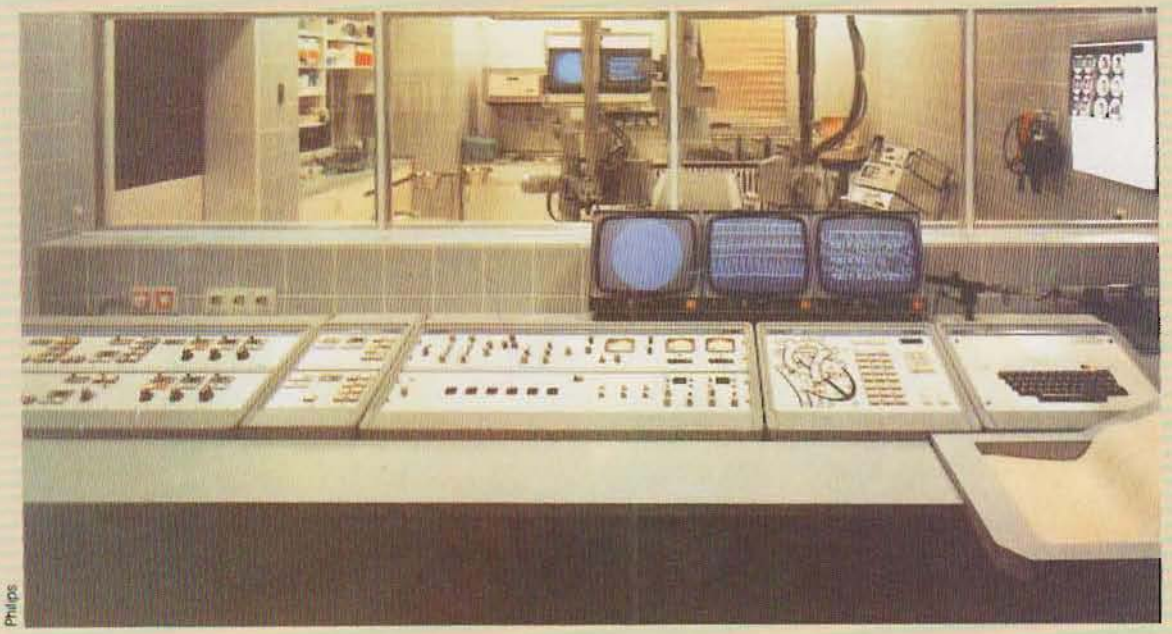
Les deux systèmes angiographiques de la dernière génération sont représentés par le Poly-Diagnost UPI pour l'angiographie générale et par le Poly-Diagnost C, spécialement conçu pour l'angiocardiographie.

*Le premier équipement de type multidisciplinaire a une structure de **bras en U** qui permet de tourner le tube émetteur de 330° autour du patient en observation. Cette **large possibilité de rotation** est indispensable pour mettre en évidence les détails caractéristiques de l'image angiographique.*

Le poids global de l'appareil (émetteur et table) atteint une tonne et l'on comprend combien ce système est redevable aux dernières conquêtes de l'automatisme industriel qui permet de manier sans problème des appareils aussi lourds et aussi délicats. D'autant plus que l'importance et le poids des blindages nécessaires à la protection contre la diffusion multidirectionnelle du rayonnement X ne peuvent être diminués sans risque.

Aux deux extrémités du bras en U sont installés le tube émetteur et l'amplificateur de brillance doté d'un chargeur de film.

Le contrôle de l'ensemble est programmable sur cartes perforées. Le programme dirige le mouvement de la source en s'adaptant de très près à la nature de l'investigation, ce qui per-



Philips

Console de contrôle et de manipulation du système.

met de procéder à des examens très fins et donc de détecter très rapidement la moindre anomalie. Comme on le sait, le diagnostic précoce est particulièrement précieux dans des domaines où la chirurgie reste d'un usage très délicat.

Le système Poly-diagnost C est spécialisé, lui, dans l'angiocardiographie. On peut en voir un modèle sur la photo ci-dessus où se trouve mise en valeur l'architecture particulière du bras articulé. Elle permet des examens sous les angles les plus variés, ce qui est particulièrement utile dans le cas d'organes richement vascularisés. Le positionnement initial du bras s'effectue alors manuellement, par une légère action sur la poignée.

Autre avantage du système : la mobilité des bras qui permet d'effectuer des balayages sur n'importe quelle région de l'organisme ***sans*** *pour autant nécessiter un* ***déplacement du sujet*** *examiné. Cette possibilité est particulièrement précieuse pour deux raisons. Tout d'abord, elle élimine une source de fatigue au malade, souvent affaibli quand de tels examens s'avèrent nécessaires. Par ailleurs, la présence d'un personnel manipulant risque d'augmenter le stress... et fait durer le temps de l'examen. Enfin, moins le malade est manipulé, plus il se trouve au repos et plus les observations sont fiables. Le système vasculaire est, en effet, très délicat et le moindre choc (involontairement produit au cours de manipulations) comme la moindre émotion, gênent considérablement l'examen.*

La configuration du système peut être complétée par un second bras latéral qui permet l'examen sur deux plans de la région du cœur. Dans ce cas particulier, la mobilité du système par rapport au malade est d'autant plus appréciée. Deux bras se meuvent de façon synchronisée sous le contrôle du microprocesseur. Le procécé angioradiographique utilisé ne se différencie pas des dispositifs existants par ailleurs en radiodiagnostic. La maîtrise de la distance entre le tube émetteur et le patient élimine tout problème de contact particulièrement dangereux comme dans le cas d'un cathétérisme.

Comme on peut le noter sur la photo ci-contre, la console de pilotage d'un appareil n'a rien à envier au panneau de contrôle d'une station ferroviaire. Tout le système est géré en ***temps réel*** *par un ordinateur.*

Un nouveau procécé, dit de ***radiographie vasculaire digitalisée****, a été mis au point. Il est en passe de devenir la technique standard d'investigation, comme le justifient les avantages de fiabilité de finesse et de rapidité d'investigation que ce nouveau système assure.*

L'image radiologique est numérisée et conservée en mémoire. Le schéma du principe de fonctionnement du système est représenté page 940.

Il existe trois niveaux de digitalisation (numérisation) du signal en provenance de l'appareil radiologique. Chacun d'entre eux est représenté par le bloc « algorithme » programmable par l'opérateur au moyen du panneau de contrôle qui comporte trois touches (figuré par l'indication « Mode » sur le panneau de contrôle). Une quatrième touche (post-processing) permet d'intervenir sur les données saisies par les précédentes fonctions.

Le signal analogique provenant de la scopie et de l'amplificateur de brillance est renvoyé, via l'interface, à un autre amplificateur de type logarithmique.

A sa sortie, le signal est transformé, par le convertisseur analogique/digital, en signaux numériques et stockés en mémoire. Leur traitement s'effectue en temps réel avec l'un des trois algorithmes. Le résultat, affiché à l'écran, est sauvegardé et la sortie est enregistrée sur disque ou sur bande magnétique.

Appelé "Serial imaging mode", le premier algorithme est représenté page 940 (voir schéma Ⓐ). L'injection de l'opacifiant dans le système veineux sous le contrôle radioscopique permet de sélectionner l'image qui servira ensuite de « masque » radioscopique.

Ce masque sera ensuite affiné par des procédés électroniques dont le résultat est une succession d'images produites à intervalles réguliers d'environ 1 seconde. Ceci permet d'aboutir, ***par élimination, à la zone pathologique*** *à explorer (par exemple le vaisseau). A noter que le "Serial imaging mode" a été utilisé avec un succès particulier dans l'examen des vaisseaux carotidiens, périphériques et rénaux.*

Le second mode de digitalisation, appelé "Continuous imaging mode" (voir schéma Ⓑ, page 940), construit de la même manière un masque mais à des intervalles de 16,6 millisecondes. L'image produite est projetée en temps réel sur écran et sauvegardée, ce qui permet, le cas échéant, de la rappeler. Cette technique s'avère très efficace dans l'exploration des artères pulmonaires et des mouvements ventriculaires et valvulaires.

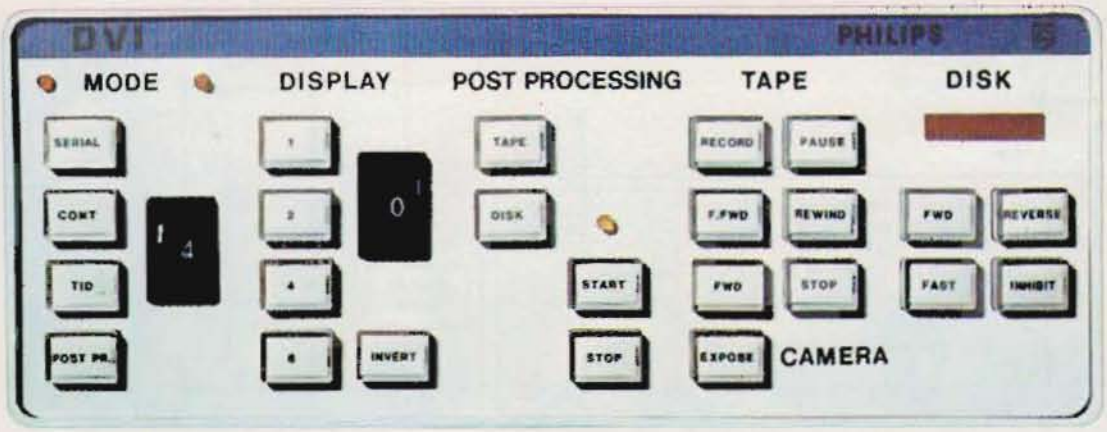
Le "Time interval difference mode" (TID), dernier des trois algorithmes, consiste à traiter les images à des intervalles constants par le Continuous imaging mode (voir schéma Ⓒ, page 949).

Le "Post-processing mode" (voir schéma Ⓓ) est, à la base, un système de traitement "off-line" (autonome) qui ne nécessite pas la présence du patient. Il travaille à partir des images précédemment enregistrées en leur appliquant l'une des trois possibilités décrites ci-dessus.

L'angiographie digitalisée offre de nombreux avantages et, notamment, celui de ne présenter aucune incompatibilité avec les méthodes existantes. Les images obtenues en temps réel par procédé d'élimination sont, à tout moment, ***disponibles en sortie sur papier*** *grâce à un dispositif vidéo sophistiqué, connecté à l'appareil.*

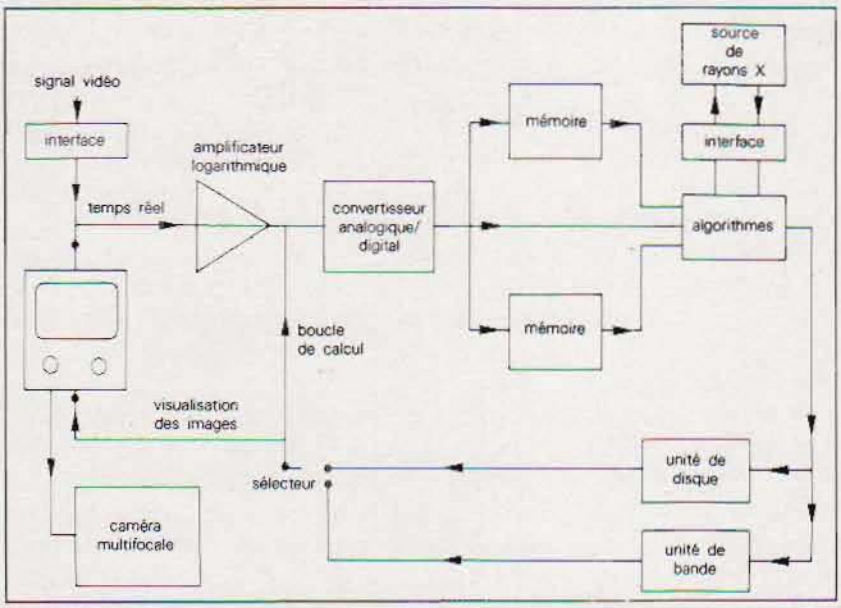
*Notons enfin que le système digital accepte, en entrée, les images produites par d'****autres équipements radiologiques.*** *Ces propriétés en font un outil de très large diffusion scientifique dans un proche avenir.*

Tableau de commandes et de contrôle du système angiographique.

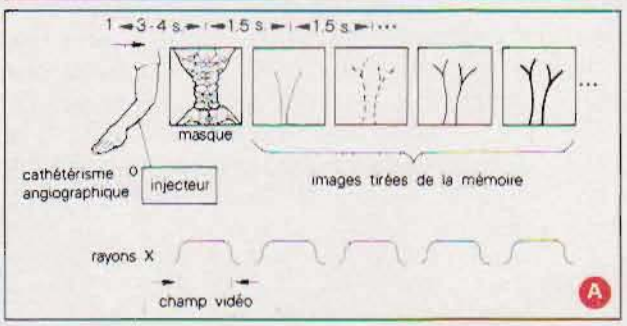


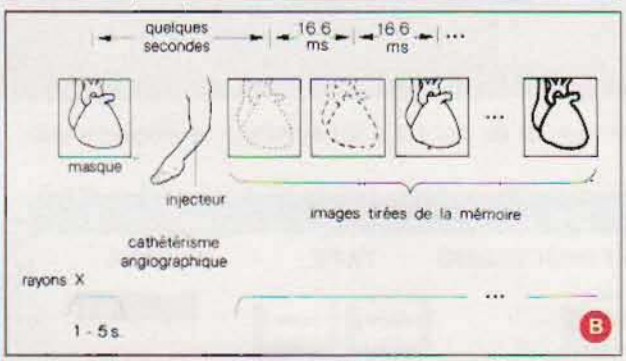
Philips

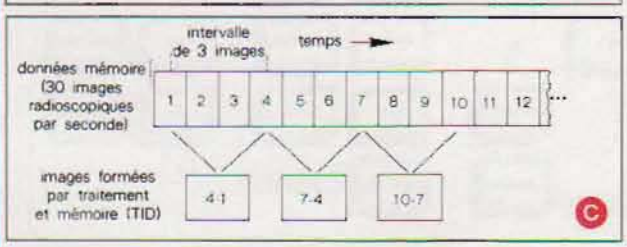
SCHEMA ET FONCTIONNEMENT D'UN SYSTEME DIGITAL (OU NUMERIQUE) ANGIOGRAPHIQUE

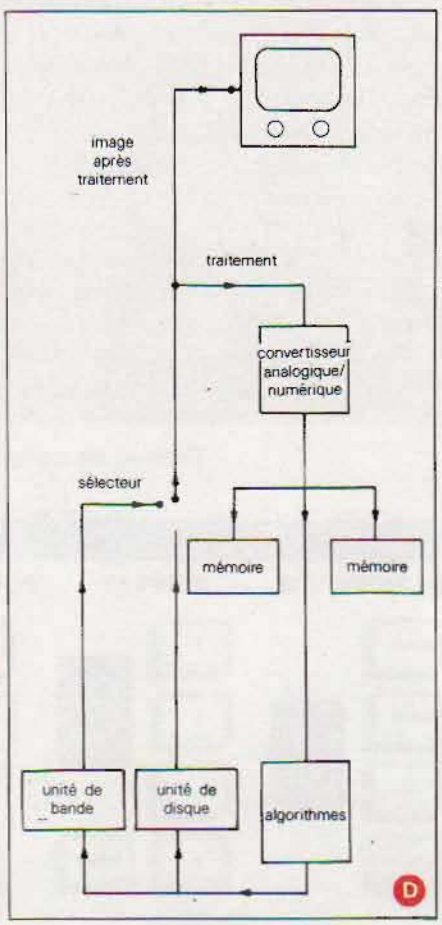


A gauche, on trouve le schéma de fonctionnement d'un appareil d'analyse du système vasculaire. Quatre modes d'examen sont utilisés :
En Ⓐ : "Serial imaging mode", en Ⓑ : "Continuous imaging mode" en Ⓒ, "Interval difference mode (TID), en Ⓓ : "Post processing mode" qui peut intervenir sur les images traitées avec un des algorithmes précédents.
De plus amples détails sont donnés en page 939.









L'instruction ORG (code mnémonique pour ORIGINE) définit une adresse initiale de rangement en mémoire d'un programme, d'une routine ou d'un ensemble de données. ORG doit être suivie par une opérande, de type immédiat. En écrivant :

```
ORG 100
LD A, ALPHA
...
```

on charge en mémoire la portion de programme commençant par LD A, ALPHA à l'adresse 100 et suivantes.
Avec l'instruction ORG, on évite que le programme principal et les sous-programmes se superposent en mémoire ou à des espaces normalement occupés par le système d'exploitation. Un programme Assembleur peut recevoir plusieurs instruction ORG.
L'instruction EQU (code mnémonique pour EGALITE) définit le nom d'une constante et lui assigne une valeur numérique. A chaque fois que la constante sera appelée dans le programme, on lui substituera la valeur numérique assignée. Le nom de la constante est logé dans le champ étiquette de l'instruction, comme le montre l'exemple suivant :

```
COUNT   EQU 1
ALPHA   EQU 20
        LD A, ALPHA
        . . .
```

Précédée de ALPHA EQU 20, l'instruction LD A, ALPHA devient équivalente dans le code machine à l'instruction LD A, 20. Les instructions EQU sont généralement positionnées en début de programme à titre de documentation et pour faciliter la lisibilité. L'instruction RES (pour RESERVE) attribue un espace RAM aux variables et assigne un nom à la première adresse de l'espace réservé. Par exemple :

```
BUFF    RES 50
```

crée un espace mémoire de 50 positions et assigne, à la première d'entre elle, le nom BUFF. De cette façon, l'accès à l'espace mémoire réservé pourra être effectué en se référant à BUFF.
Par exemple, avec l'instruction :



J. Pickerell/Marka

Une salle machine de la General Electric.

```
LD A, BUFF + 10
```

c'est le contenu de la dixième position mémoire, à partir de celle de BUFF (créée avec l'instruction RES) qui est chargé dans le registre A.
Enfin, l'instruction DATA réserve une zone mémoire et l'initialise avec une valeur donnée, contrairement à l'instruction RES qui ne fait qu'allouer un espace mémoire pour les variables.
Il est toutefois possible de réserver des positions mémoire avec l'instruction RES pour ensuite les remplir avec des valeurs.
On peut également écrire :

```
BETA    DATA 2
```

cela signifie que la première position mémoire libérée sera, dès lors, appelée BETA (adresse) et contiendra la valeur 2.
De même, l'instruction :

```
GAMMA   DATA 3, 4, 5, 6, 7
```

créera un tableau de 5 éléments dont le premier aura, pour adresse, GAMMA avec la valeur 3. Les éléments suivants contiendront les autres valeurs (4, 5, 6 et 7).

Exemple de programmation en Assembleur

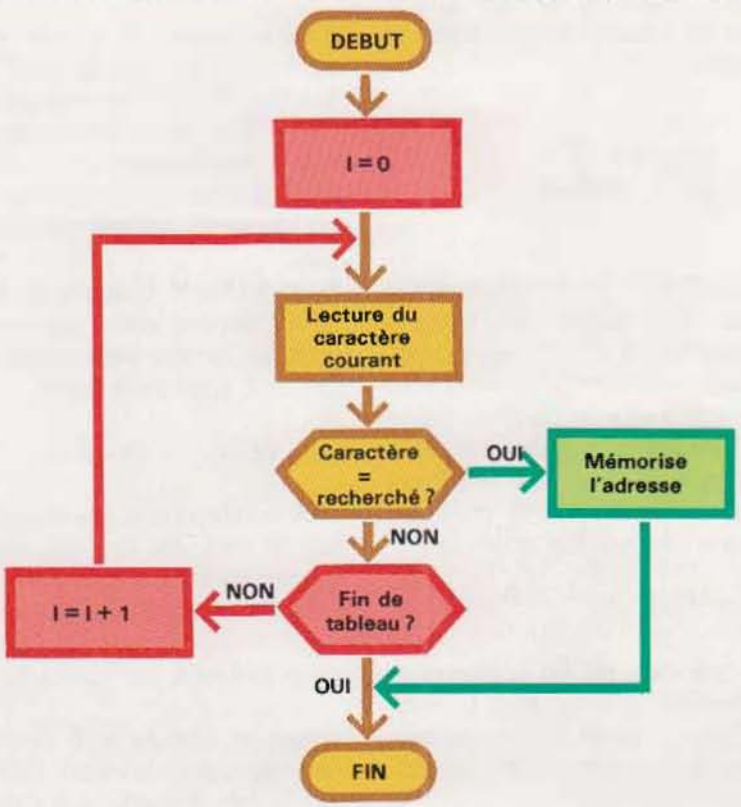
Comme on vient de le voir, le langage Assembleur est évidemment plus complexe que les langages symboliques comme le Basic. Il demeure encore aujourd'hui indispensable dans la résolution d'applications. Son rapport étroit avec le fonctionnement réel du microprocesseur exige, de la part du programmeur, une phase d'analyse du problème beaucoup plus minutieuse. En langage d'assemblage, il ne suffit plus d'analyser une procédure à travers un organigramme décrivant une succession d'opérations complexes (entrée des données, traitement des différentes fonctions, enfin présentations des résultats). Il faut, en plus, examiner dans le détail le déroulement de chaque opération à l'intérieur de la machine. Chaque bloc fonctionnel d'un organigramme doit être détaillé pour arriver à une succession d'opérations élémentaires directement traduisibles en Assembleur.

Ceci est valable pour chaque langage évolué de programmation, l'expérience seule permettant de mettre en évidence les détails superflus. Ainsi, le programmeur pourra expliciter un bloc fonctionnel directement en phase de programmation. A titre d'exemple, examinons maintenant un cas simple de programmation en Assembleur. Le but sera d'obtenir un sous-programme qui développe la recherche d'un caractère dans un tableau. L'organigramme de recherche est représenté ci-dessous.

Prenons un autre exemple. Chargeons des caractères dans une série de positions mémoire contiguës (tampon) à partir de l'adresse INI. Rangeons, de même, le caractère de contrôle à l'adresse CH. Supposons, de plus, que NC soit le nombre de caractères contenus dans le tableau.

Notre programme devra utiliser deux registres. Dans le registre X, nous mémorisons le déplacement par rapport à l'adresse INI à l'intérieur du tampon. Le premier caractère est en position INI, le second en INI + 1, etc. Un caractè-

ORGANIGRAMME DE RECHERCHE DANS UN TABLEAU



re quelconque se trouve à l'adresse INI + X, avec X allant de 0 à NC — 1. Dans le déroulement du programme, l'accumulateur A prend la valeur du caractère courant que l'on compare au caractère de contrôle. La valeur du pointeur sur l'élément du tampon devra satisfaire à la condition d'égalité.
Examinons maintenant la structure du programme, sachant que les codes mnémoniques employés dans les instructions sont étroitement liés au type de microprocesseur.

LDX # 0 Initialise le registre X à la valeur 0.

LDA CH Lit, à l'adresse CH, le caractère à comparer et le charge dans l'accumulateur.

CMP INI,X (CMP adresse, valeur du déplacement par rapport à l'adresse INI).
C'est une instruction particulière à certains microprocesseurs. Elle lit le contenu de la position mémoire (l'adresse INI + déplacement) et compare la donnée trouvée avec le contenu de l'accumulateur. S'ils sont égaux, l'indicateur Z est mis à 1 (voir graphique en page 944). Dans la forme utilisée, l'instruction compare le contenu de A avec celui de la mémoire INI + X. A ce point du programme, X = 0, on le compare donc avec le contenu de la position INI, c'est-à-dire le premier élément du tampon. Si le résultat de la comparaison est positif, l'indicateur Z est mis à 1.

BEQ TROUVE Le code BEQ (Branch if EQual, pour Branchement si égal) exécute un branchement à l'adresse spécifiée (de nom TROUVE), si Z = 1. Sinon (NON TROUVE), on poursuit en séquence.

LOOP INX Incrémente X pour prélever un nouveau caractère du tampon.

CMP INI, X Effectue la comparaison décrite ci-dessus.

BEQ TROUVE Branche à TROUVE si Z = 1.

STX SAUVE Range le contenu de X à l'adresse SAUVE. De cette façon, le registre X peut être temporairement employé à d'autres fins ; le contenu d'origine est rétabli en rechargeant SAUVE.

CPX NC Compare X avec la longueur du tampon ; s'ils sont égaux, place Z à 1.

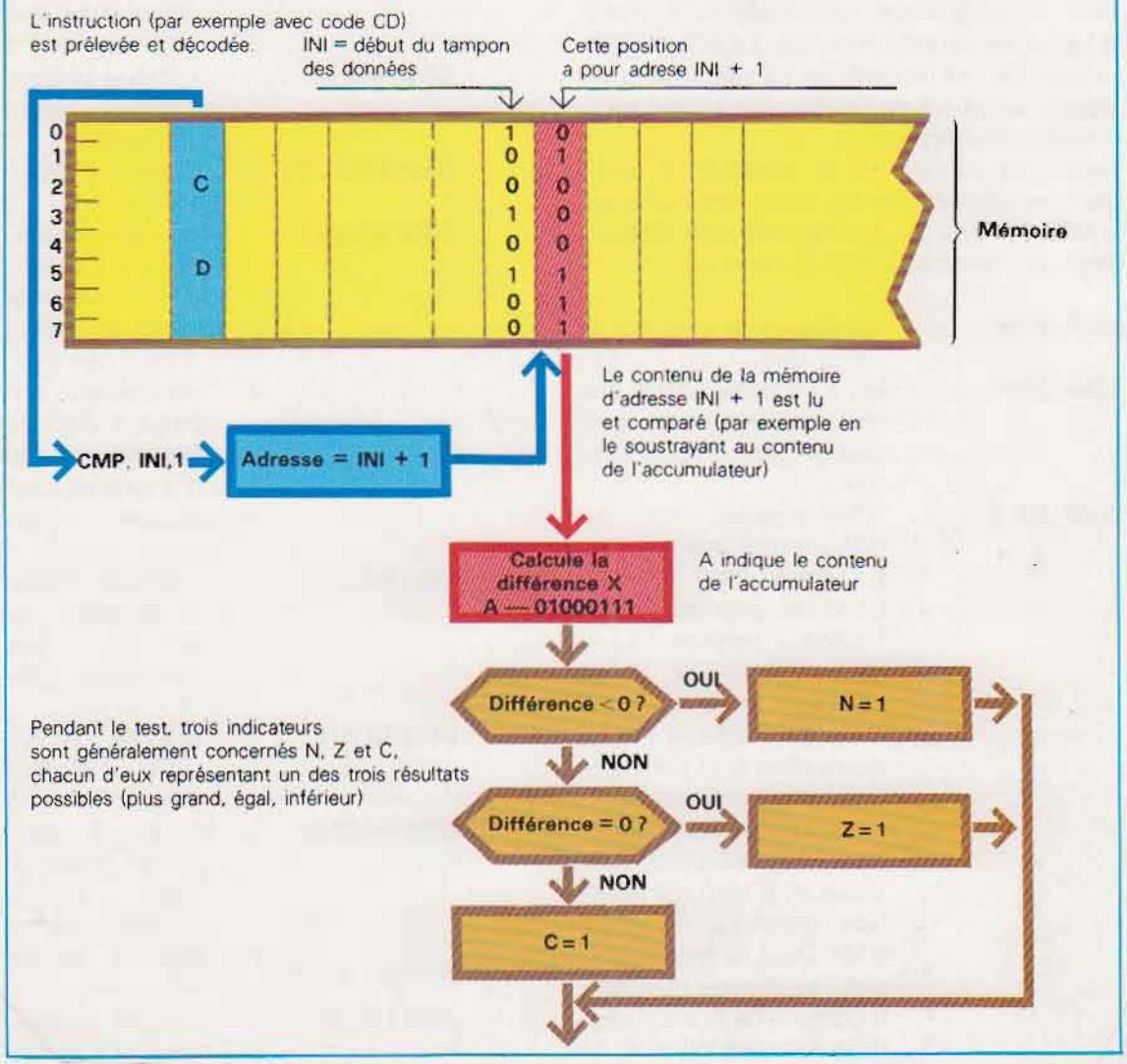
BEQ NONT Branche à NONT (NON TROUVE) si égal.

LDX SAUVE Recharge, dans X, sa valeur X avant le test (le pointeur sur la dernière donnée examinée).

JMP LOOP Branche à la position LOOP où commence une nouvelle boucle.

TROUVE NOP Point de retour si le caractère est trouvé. Le code NOP (non opération) n'a aucun effet ; c'est un commentaire. A ce point, le registre X contient la position (à partir de 0) de l'élément du tampon égal au caractère de contrôle.

SCHEMA LOGIQUE DE L'INSTRUCTION CMP



TXA Les éléments suivants, s'ils existent, ne sont pas pris en considération, même s'ils satisfont à l'égalité. Le contenu de X est transféré dans l'accumulateur. En sortie, A contient la position de l'élément du tampon égal au caractère de contrôle.

NONT Non trouvé (commentaire).

NOP

RTS Retour. Dépile le contenu de la pile pour obtenir le contenu du PC et l'augmente de 1. RTS prépare donc l'exécution de l'instruction qui suit l'appel du sous-programme (JSR ou RJP selon le type de microprocesseur).

L'Assembleur du Rockwell 6502

L'Assembleur du Rockwell 6502 a eu un grand succès. On le retrouve dans ses diverses versions sur plusieurs ordinateurs personnels comme Commodore (VIC-20 et 64) et Apple II. La disposition et la jonction des broches du microprocesseur sont décrites ci-dessous alors que, page 946, on peut voir la structure interne et la configuration du registre d'état.
Dans le 6502, on trouve, outre l'accumulateur, les registres X, Y et S. X et Y sont utilisés comme indices et le S contrôle la pile. Le registre S contient l'adresse de la première zone disponible dans la pile qui, dans le 6502, s'étend des adresses physiques 256 à 511, soit 255 octets.

Le registre d'état (P), comprend les indicateurs suivants :

N Bit du signe (1 si négatif)

V Bit de dépassement (1 si c'est le cas)

B Bit d'interruption logiciel (positionné à 1 si demande d'interruption)

D Bit d'utilisation du code BCD (positionné à 1 quand on utilise la représentation BCD)

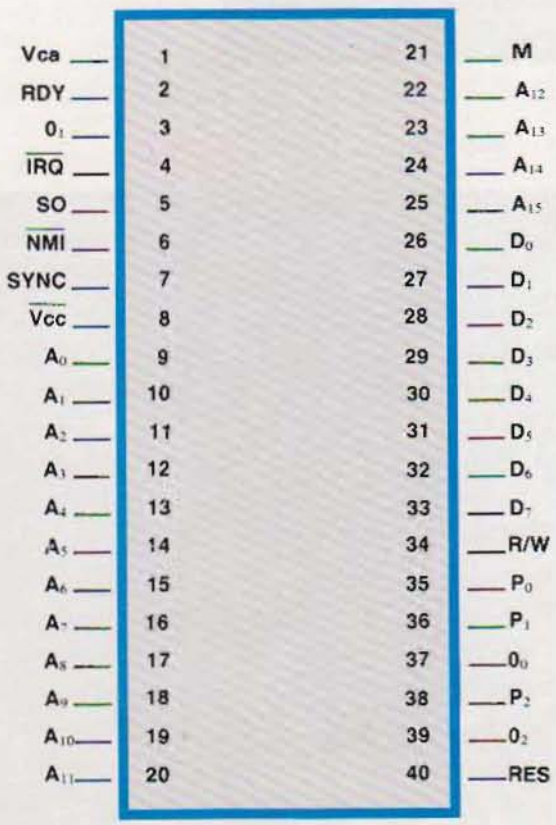
I Bit d'interruption matériel (positionné à 1 quand les interruptions sont impossibles)

Z Bit zéro (1 si la dernière opération a eu pour résultat zéro)

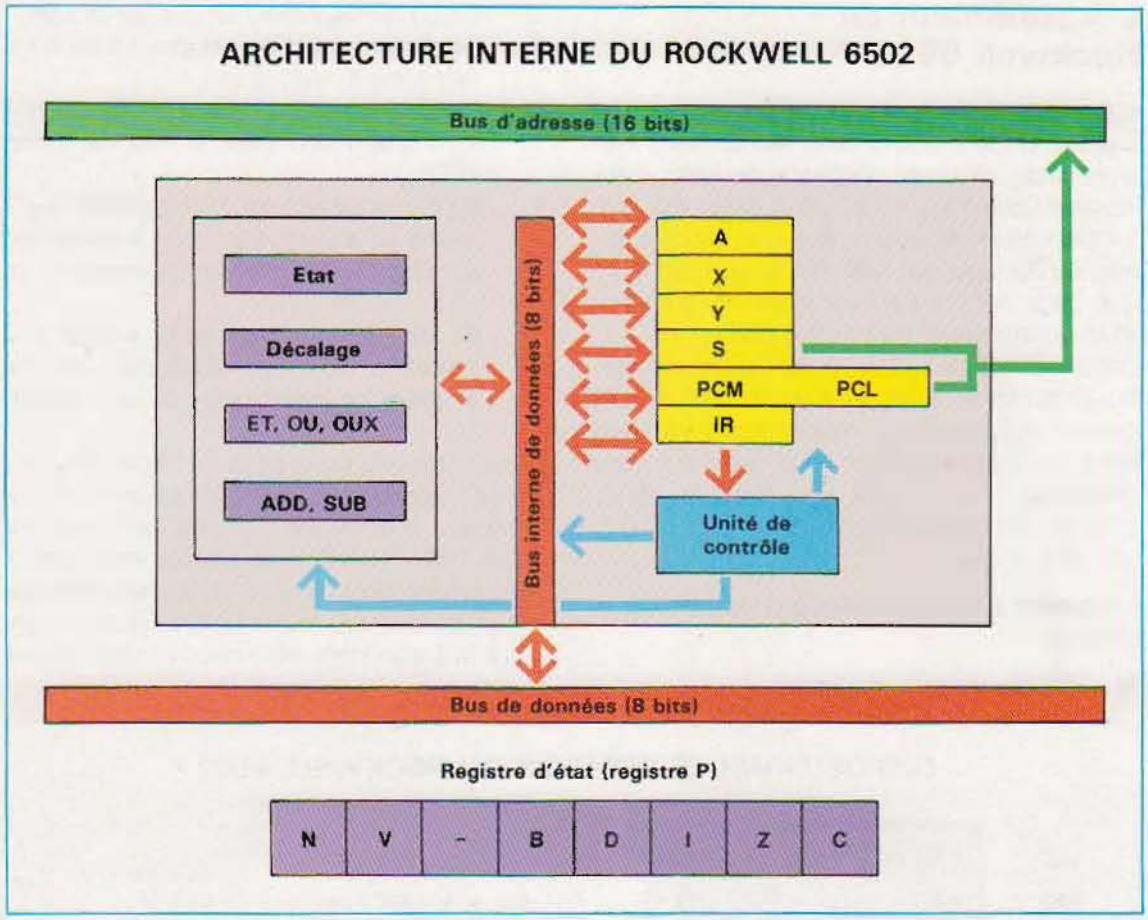
C Bit de retenue (1 s'il y en a une). Cet indicateur est aussi utilisé pour les instructions de permutation ou de rotation.

Dans le tableau de la page 947 sont listés les codes mnémoniques des instructions du 6502 comparés aux codes standard proposés par l'IEEE Task P694/011 (organisme international de normalisation). Notons particulièrement les spécialisations des instructions et plus précisément la façon dont elles opèrent directement sur les registres présents (A, X, Y, S).

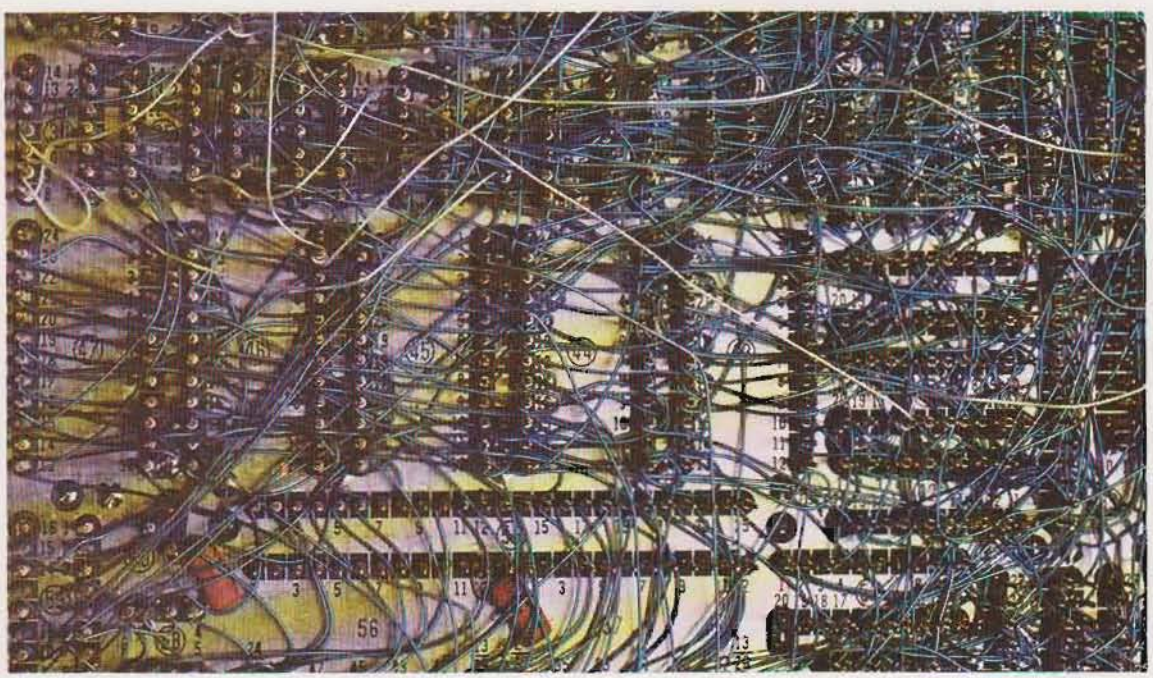
DISPOSITION DES BROCHES DU ROCKWELL 6502



Bus d'adresse, ligne :	$A_0 \div A_{15}$
Bus de données, ligne :	$D_0 \div D_7$
Alimentation :	Vca, Vcc, M (masse) (± 5 volts)
Bus de contrôle :	
Signaux d'horloge	0_0, 0_1, 0_2
Signal de mise à zéro (ou initialisation)	RES
Contrôle de l'indicateur de dépassement	SO
Signaux de synchronisation	SYNC, RDY
Lecture/écriture	R/W
Lignes d'interruption	IRQ, NMI



Système complexe des liaisons externes d'un ordinateur.



J. Pickerell/Maika

INSTRUCTIONS ASSEMBLEUR DU ROCKWELL 6502

Instructions	Code mnémonique IEEE	Code mnémonique 6502
Instructions arithmétiques		
Add with carry	ADDC	ADC
Subtract with carry	SUBC	SBC
Increment	INC	INC
Increment X		INX
Increment Y		INY
Decrement	DEC	DEC
Decrement X		DEX
Decrement Y		DEY
Compare to A	CMP	CMP
Compare to X		CPX
Compare to Y		CPY
Instructions logiques		
AND	AND	AND
OR	OR	ORA
Exclusive OR	XOR	EOR
Shift Right	SHR	LSR
Arithmetic Shift Left		ASL
Rotate Right	ROR	ROR
Rotate Left	ROL	ROL
Test Bit	TEST	BIT
Instructions de transfert de données		
Load A	LD	LDA
Load X		LDX
Load Y		LDY
Store A	ST	STA
Store X		STX
Store Y		STY
Clear Carry	CLR	CLC
Clear Decimal		CLD
Clear Interrupt		CLI
Clear Overflow		CLV
Set Carry	SETC	SEC

Instructions	Code mnémonique IEEE	Code mnémonique 6502
Set Decimal		SED
Set Interrupt Mode		SEI
Transfer A to X	MOV	TAX
Transfer A to Y		TAY
Transfer SP to X		TSX
Transfer X to A		TXA
Transfer X to SP		TXS
Transfer Y to A		TYA
Instructions de branchement		
Branch if Zero	BZ	BEQ
Branch if Not Zero	BNZ	BNE
Branch if Negative	BN	BNI
Branch if Positive	BP	BPL
Branch if Carry	BC	BCS
Branch if No Carry	BNC	BCC
Branch if Overflow Clear	BNV	BVC
Branch if Overflow Set	BV	BVS
Jump		JMP
Instructions d'appel à sous-programme		
Jump to Subroutine	CALL	JSR
Instructions de retour		
Return from Subroutine	RET	RTS
Return from Interrupt		RTI
Instructions diverses		
No Operation	NOP	NOP
Push A	PUSH	PHA
Push P (status)		PHP
Pop A	POP	PLA
Pop P (status)		PLP
Break	BRK	BRK

L'Assembleur du Zilog Z80

Le Zilog Z80 est l'un des microprocesseurs 8 bits les plus sophistiqués. Développé en vue d'être compatible avec l'Intel 8080, tout en ayant des possibilités supplémentaires, on le trouve dans les plus puissants micro-ordinateurs. Il a servi de base au développement du système d'exploitation CP/M (Control Programming for Microcomputer).

Le tableau de la page 948 illustre la disposition et les fonctions des broches du microprocesseur ainsi que les fonctions des bus de liaison les plus importantes. L'architecture interne du Zilog Z80 et la structure du registre d'état sont schématisées à la page 949.

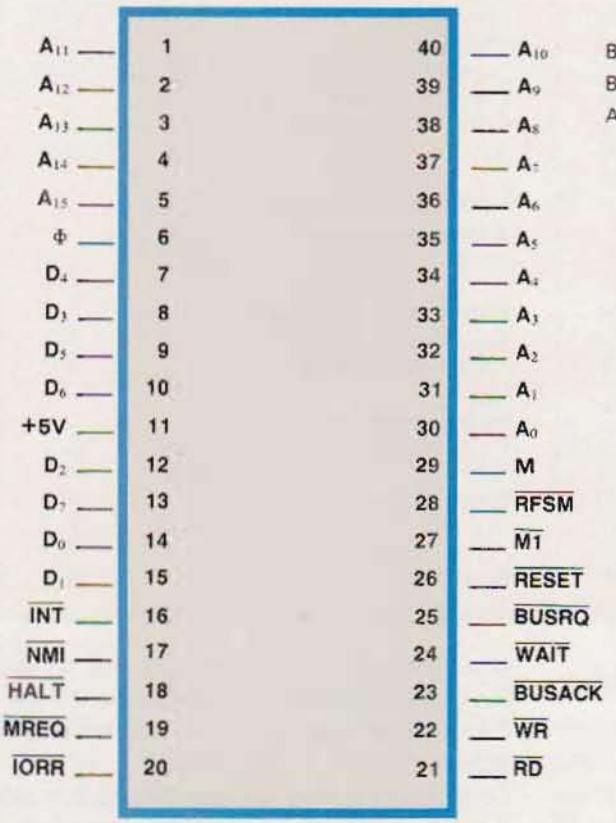
La structure interne du Z80 est très complexe. Il est composé de deux accumulateurs (A, A') et de deux couples de 6 registres (BC, DE, HL, B'C', D'E', H'L') avec, en plus, deux registres index (IX, IY), un pointeur de pile (SP), un compteur ordinal (PC), le registre d'instruction (RI) et deux registres spécialisés I et R, le premier étant utilisé par les interruptions, le se-

cond pour la gestion des rafraîchissement de la mémoire. Deux registres temporaires sont présents dans l'unité arithmétique et logique : TMP et Acc. Temp. utilisés avec l'accumulateur pour effectuer les calculs.
Il y deux registres d'état identiques, appelés F et F', qui contiennent les bits suivants :

S Bit du signe
Z Bit de zéro
H Bit de demi-retenue (BCD)
P/V Bit de parité ou de dépassement (selon la valeur des autres bits)
N Bit de soustraction utilisé par le système dans les opérations en BCD
C Bit de retenue

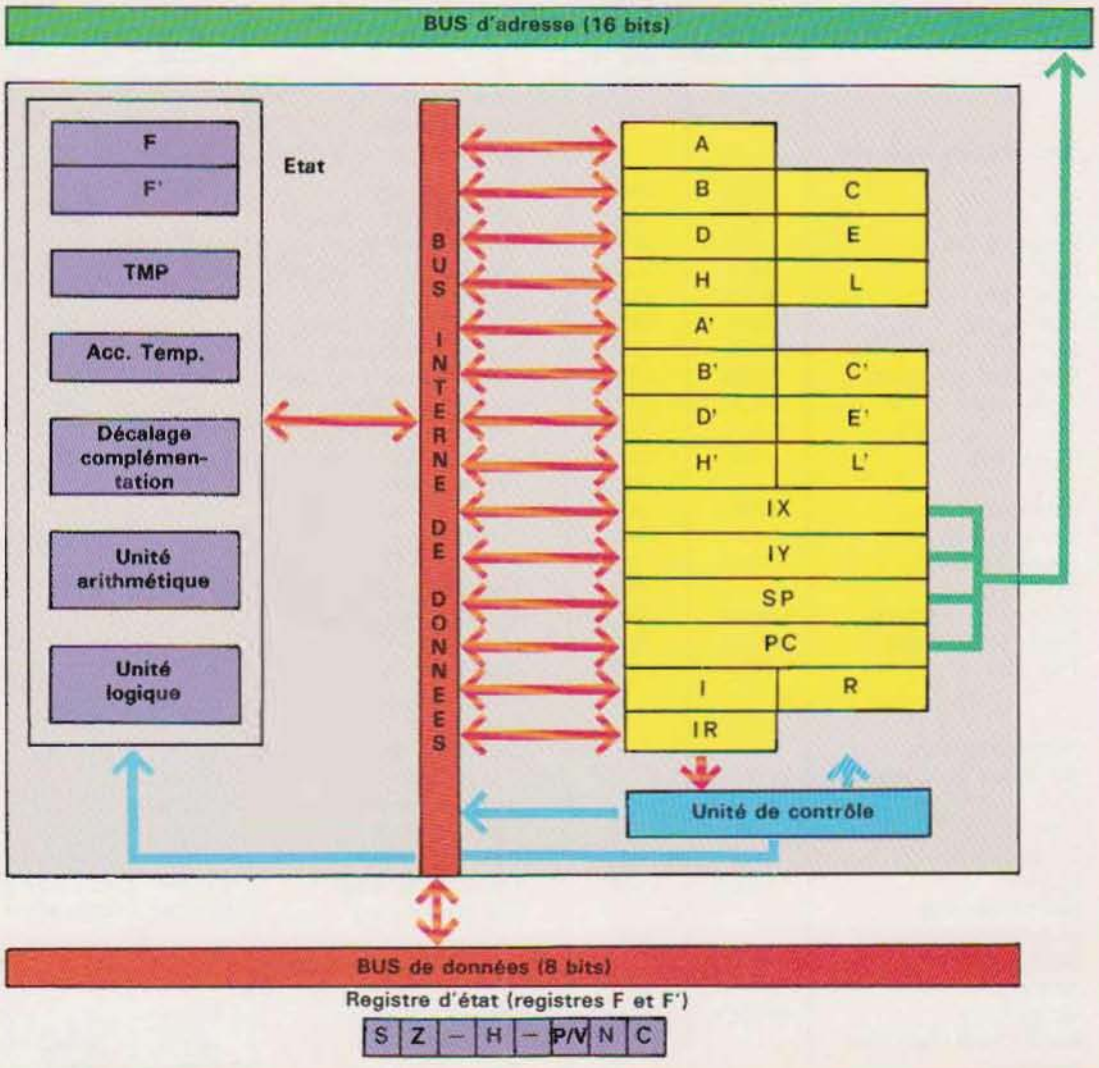
Le tableau des pages 949-950 répertorie les codes mnémoniques des instructions du Zilog Z80 comparés aux codes standard IEEE. Comme on peut le voir, l'ensemble des instructions disponibles est très large, ce qui donne une grand souplesse d'exécution au microprocesseur.

ARCHITECTURE DES BROCHES DU ZILOG Z80



Bus d'adresse :	$A_0 - A_{15}$
Bus de données :	$D_0 - D_7$
Alimentation :	+5V, Masse
Bus de contrôle :	
Horloge de l'UC :	Φ
Cycle machine de mise en place (Fetch) :	$\overline{M1}$
Signale que l'UC effectue un accès mémoire :	$\overline{MREQ}$
Signale une opération E/S :	$\overline{IORR}$
Signale que l'UC lit dans la mémoire :	$\overline{RD}$
Signale que l'UC écrit dans la mémoire :	$\overline{WR}$
Signal de rafraîchissement de la mémoire :	$\overline{RFSM}$
Arrêt effectué par l'UC :	$\overline{HALT}$
Demande d'attente :	$\overline{WAIT}$
Demande d'interruption :	$\overline{NMI}$, $\overline{INT}$
Réinitialisation de l'UC :	$\overline{RESET}$
Ligne de contrôle des bus :	$\overline{BUSRQ}$, $\overline{BUSAK}$

ARCHITECTURE INTERNE DU ZILOG Z80



INSTRUCTIONS ASSEMBLEUR DU ZILOG Z80

Instructions	Code mnémonique IEEE	Code mnémonique Z80	Instructions	Code mnémonique IEEE	Code mnémonique Z80
Instructions arithmétiques					
Add	ADD	ADD	Compare with		
Add with carry	ADDC	ADC	Increment		CPI
Subtract	SUB	SUB	Compare with		
Subtract with carry	SUBC	SBC	Decrement		CPD
Increment	INC	INC	Compare multiple		CPIR
Decrement	DEC	DEC			CPDR
Compare	CMP	CP	Negate	NEG	NEG

Instructions	Code mnémonique IEEE	Code mnémonique Z80
Instructions logiques		
AND	AND	AND
OR	OR	OR
Exclusive OR	XOR	XOR
NOT	NOT	CPL
NOT carry	NOTC	CCF
Shift right	SHR	SRL
Shift left	SHL	SLA
Shift right arithmetic	SHRA	SRA
Rotate right	ROR	RRCA
		RRC
Rotate left	ROL	RLCA
		RLC
Rotate right through carry	RORC	RR
		RRA
Rotate left through carry	ROLC	RL
		RLA
Rotate right decimal	ROR4	RRD
Rotate left decimal	ROL4	RLD
Test bit	TEST 1	BIT
Instructions de transfert de données		
Load	LD	LD
Store	ST	LD
Move	MOV	LD
Block load with increment		LDI
Block load with decrement		LDD
Move block	MOVBK	LDIR
Repeat block load with decrement		LDDR
Exchange	XCH	EX
Exchange alternate register		EXX
Input	IN	IN
Input with increment		INI
Input with decrement		IND
Input block	INBK	INIR
Block input with decrement		INDR
Output	OUT	OUT
Output with increment		OUTL
Output with decrement		OUTD
Output block	OUTBK	OTIR
Block output with decrement		OTDR
Set bit	SET1	DET
Clear bit	CLR1	RES
Set carry	SETC	SCF
Set interrupt mode	SETI	IM

Instructions	Code mnémonique IEEE	Code mnémonique Z80
Instructions de branchement		
Branch	BR	JP
Branch if zero	BZ	JPZ
Jump relative if zero		JRZ
Branch if not zero	BNZ	JPNZ
Jump relative if not zero		JRNZ
Branch if carry	BC	JPC
Jump relative if carry		JRC
Branch if not carry	BNC	JPNC
Jump relative if no carry		JRNC
Branch if positive	BP	JPP
Branch if negative	BN	JPN
Branch if parity even	BPE	JPPE
Branch if parity odd	BPO	JPPO
Decrement and branch if not zero		DJNZ
Instructions d'appel à sous-programme		
Call	CALL	CALL
Restart at address		RST
Call if zero	CALLZ	CALLZ
Call if not zero	CALLNZ	CALLNZ
Call if carry	CALLC	CALLC
Call if not carry	CALLNC	CALLNC
Call if positive	CALLP	CALLP
Call if negative	CALLN	CALLN
Call if parity even	CALLPE	CALLPE
Call if parity odd	CALLPO	CALLPO
Instructions de retour		
Return	RET	RET
Return if zero	RETZ	RETZ
Return if not zero	RETNZ	RETNZ
Return if carry	RETC	RETC
Return if not carry	RETNC	RETNC
Return if positive	RETP	RETP
Return if negative	RETN	RETN
Return if parity even	RETPE	RETPE
Return if parity odd	RETPO	RETPO
Return from interrupt		RETI
Return from interrupt Non maskable		RETN
Instructions diverses		
No operation	NOP	NOP
Push	PUSH	PUSH
Pop	POP	POP
Wait	WAIT	HALT
Adjust deminal	ADJ	DAA
Enable interrupt	EI	EI
Disable interrupt	DI	DI

Le langage Cobol

L'ordinateur doit sa réputation à sa puissance de calcul. Sa rapidité de fonctionnement le rend également apte à d'autres applications éloignées des exigences purement mathématiques : la gestion. Il peut, par exemple, accélérer le processus d'échange des informations archivées en masse et dont la consultation aurait exigé des années pour un homme. Les contraintes posées par la nécessité de résoudre rapidement des problèmes très pointus dans des domaines très particuliers ont poussé à la construction d'un langage de programmation évolué. Ce langage d'un type nouveau devait remplir trois fonctions.

- **Mettre à la disposition d'un personnel non informaticien un langage à la fois concis et le plus près possible de la structure du langage humain.**
  Il fallait que l'utilisateur n'ait à se concentrer que sur l'examen de son problème, de la même façon qu'un peintre utilise ses propres couleurs sans en connaître la formule chimique.

- **Etre structuré de manière à traiter simplement tout type d'information, même spécialisé.**
  En se référant au langage commercial, il est évident que le responsable qui a en charge le calcul du salaire des employés d'une société a besoin de représenter d'une manière simple la chaîne de caractères alphabétiques qui constitue le nom de l'employé et de n'utiliser que les quatre opérations de base de l'arithmétique. Dans un tel langage orienté vers la résolution de problèmes commerciaux, des fonctions mathématiques complexes seraient superflues.

- **Rester, dans la limite du possible, indépendant de la machine utilisée pour l'exécution du programme.** Ainsi, le logiciel pourrait être « porté », moyennant quelques modifications, d'un calculateur à un autre. Cette possibilité, qui permet d'utiliser les mêmes programmes sur différentes machines, constitue une garantie de la pérennité des investissements d'une entreprise.

Le langage Cobol a été conçu, à partir de 1959, à l'initiative d'un comité regroupant des utilisateurs, des constructeurs et des fonctionnaires des services publics. Leur objectif : définir un langage de programmation servant tout spécialement aux activités de gestion, de façon à faciliter la mise en œuvre de programmes ayant à traiter des grands fichiers, à manipuler des informations de type alphanumérique, à éditer des états sans qu'il soit exigé de calculs ou de manipulations d'une grande complexité.
Le résultat vit le jour à l'occasion de la "COnference of DAta SYstem Languages" (CODASYL) sous la forme de notes détaillées de standardisation du COBOL (COmmon Business Oriented Langage), en d'autres termes, un langage de programmtion tout spécialement adapté au monde des affaires.
L'étendue de son domaine d'application explique, en partie, la diffusion du Cobol et de ses évolutions.
Par la suite, nous parlerons de la version normalisée "ANS" (American National Standard) du Cobol pour les raisons, déjà citées, de diffusion du langage. Les écarts constatés avec ce standard seront, à chaque fois, signalés et explicités pour toutes les instructions.
Notons deux particularités intéressantes du Cobol : les mots-clés, très proches du langage naturel, confèrent une grande facilité de lecture ; l'utilisation de mots de remplissage (ignorés par le compilateur) permettent de rédiger un texte proche du langage courant.
Pour se rapprocher le plus du langage humain, on l'a vu, l'élément à la base du Cobol est le mot. Autrement dit, le Cobol se veut un langage plus familier que symbolique. On peut mélanger, selon des règles syntaxiques précises, des mots créés par le programmeur et des mots, toujours en clairs, propres au Cobol. Par exemple, si on désire calculer le gain net d'une certaine opération commerciale comme la différence entre le produit et le montant brut de la marchandise, il suffit d'écrire l'instruction suivante :



J. Pickerell/Marka

Section des unités de mémoire à disque dans une salle machine.

SUBSTRACT MONTANT-BRUT FROM PRODUIT GIVING GAIN-NET

dans laquelle MONTANT-BRUT, PRODUIT, GAIN-NET sont des variables définies par le programmeur qui donnent immédiatement la signification logique de l'opération.
Cet exemple donne une idée de la façon dont le programmeur peut ordonner l'exécution d'une opération arithmétique simple dans un langage clair.

Généralités

Avant de décrire les caractéristiques du Cobol et la syntaxe de ses instructions, il est bon d'illustrer la séquence d'étapes logiques dans la construction du langage Cobol. Rappelons que l'ordinateur n'est qu'un instrument au service de l'homme, tant pour ses activités que pour ses loisirs. Dans la recherche d'une solution à un problème, nous procédons tous, même de manière inconsciente, selon une logique d'analyse qui comporte plusieurs étapes.

1/ Analyse du problème et choix de la solution.
On suppose devoir simplement fixer deux axes de bois entre eux (problème). Comment les fixer ? Avec des clous, des vis ou de la colle ? (analyse). D'après la fonction à remplir, par exemple la construction d'un meuble, on décide de choisir la colle (instrument).

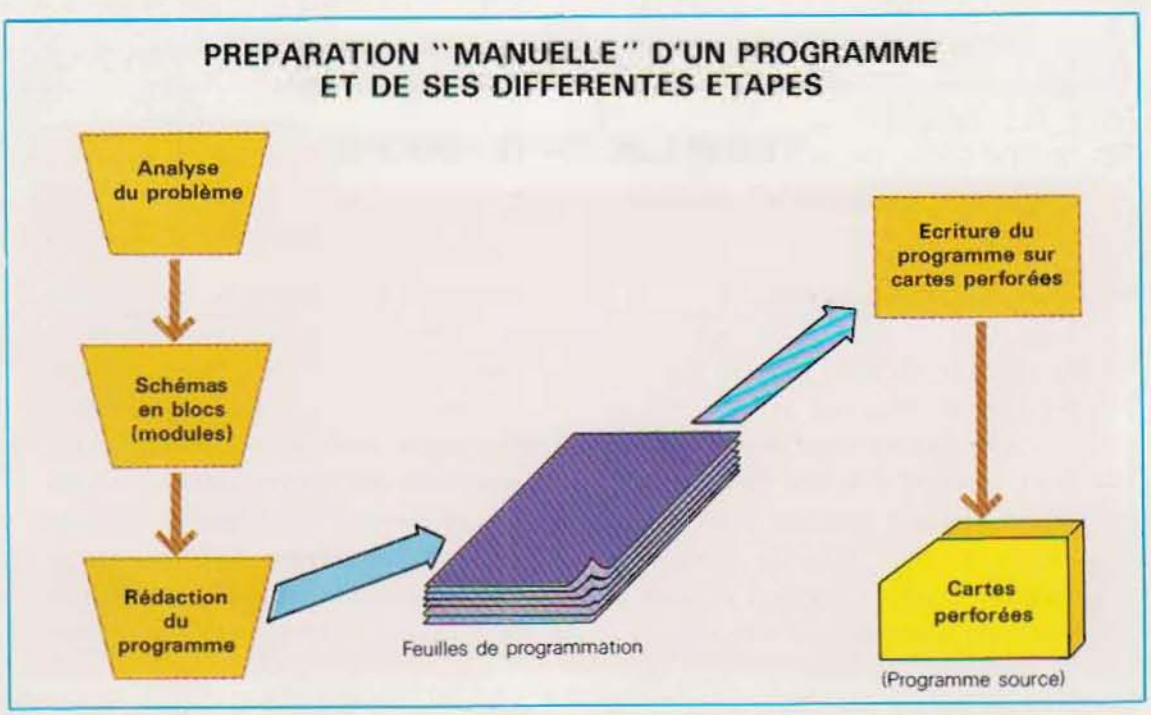
2/ Préparation des matériaux.
Pour pouvoir fournir les résultats attendus, la colle doit agir sur les matériaux à rassembler après qu'ils ont été soigneusement préparés.

3/ Recours à l'instrument.
Une fois les matériaux prêts à l'emploi, on a recours à l'outil choisi (dans ce cas, la colle).
On obtient alors, l'action terminée (à condition que l'on ait soigneusement suivi les consignes du fabricant !), l'objet en question, plus ou moins proche du projet initial.

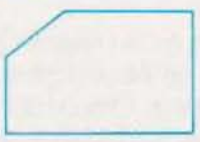
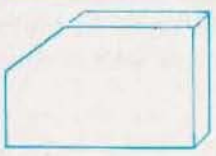
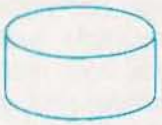
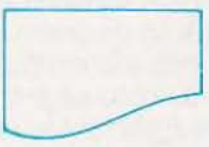
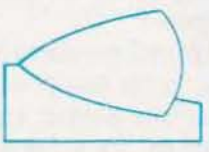
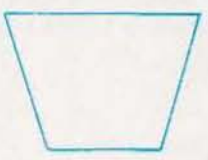
De même que pour la colle utilisée pour assembler deux axes, l'ordinateur n'intervient qu'en fin de parcours. Ce n'est qu'une des étapes que le programmeur doit accomplir avant d'aboutir au résultat. Ci-dessous sont représentées toutes les étapes nécessaires pour obtenir une première ébauche du programme compréhensible par l'ordinateur. Toutes ces actions sont exécutées en dehors de la machine. L'analyse du problème relève purement de la réflexion et débouche sur des directives fournies par l'analyste au programmeur. Grâce à ces directives, dont l'ensemble constitue le cahier des charges, on rédige un programme capable de produire les résultats escomptés.
La représentation graphique du découpage en blocs d'un programme donne également lieu à une réflexion par laquelle le programmeur construit, de manière précise, les différentes phases logiques du programme.
Même si elle n'est pas toujours indispensable à la rédaction du programme, cette étape est fortement recommandée car elle permet de corriger rapidement les erreurs de logique.
Pour la rédaction des organigrammes, on utilise des symboles graphiques particuliers (quelques exemples sont donnés dans le tableau de la page suivante).

Le programme source

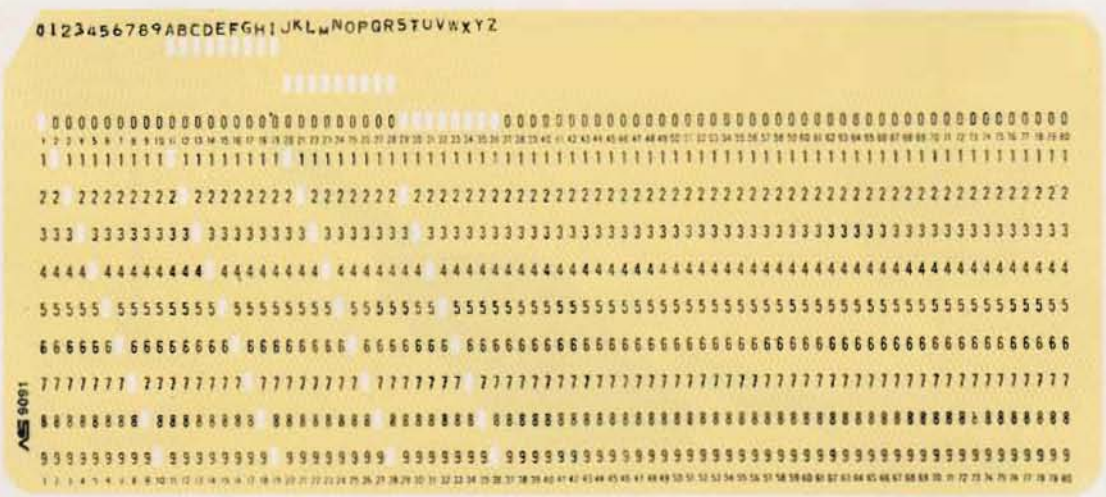
Une fois le découpage en blocs terminé, le programmeur commence la rédaction du programme, à savoir la traduction des différentes étapes de l'organigramme en série d'instructions (qui respectent la forme prévue par le langage choisi) et écrites sur des formulaires spéciaux appelés **feuilles de programmation.**
La rédaction des instructions selon les standards qui seront décrits plus loin ne constitue que le premier pas d'une série d'opérations.
Le résultat est un programme écrit en langage symbolique sur des feuilles de papier ordinaire qui ne peuvent constituer une entrée valable pour un ordinateur. Il est nécessaire de saisir les instructions contenues dans les feuilles de programmation sur un support spécial, la carte perforée, acceptable par l'ordinateur. Comme on peut le voir page 954, une carte est un petit carton capable d'accueillir jusqu'à 80 caractères consécutifs.
La codification des instructions est réalisée par des machines perforatrices qui ne sont généralement pas reliées à l'ordinateur. Chaque perforatrice est dotée d'un clavier semblable à celui d'une machine à écrire sur lequel sont numérisées, caractère par caractère, les instructions



SYMBOLIQUE COBOL DES ORGANIGRAMMES

SYMBOLE	SIGNIFICATION
	Phase initiale ou finale d'un programme
	Groupe les instructions asservies à la réalisation d'une même étape logique (procédure)
	Décision. Détermine un point du programme dans lequel peuvent être entreprises deux actions différentes en fonction du contrôle d'un événement déterminé
	Opérations de lecture ou d'écriture sur un fichier (en particulier fonction E/S)
	Carte perforée
	Fichier de cartes perforées
	Fichier résident sur disque
	Fichier résident sur bande
	Fichier d'impression
	Ecran vidéo et console
	Opération accomplie à vitesse humaine sans aide de machine

EXEMPLE DE CARTE PERFOREE



Cobol des feuilles de programmation. Un caractère numérisé n'est pas seulement reproduit sur le haut de la carte, mais il est également codé en une combinaison de trous rectangulaires disposés le long d'une colonne.

Le tableau ci-dessous établit la correspondance entre les caractères (alphabétiques, numériques et spéciaux) et leur codification sur carte perforée selon un code très répandu (la notation 6-8 indique que le caractère est codé avec deux trous disposés sur la sixième et la huitième ligne de la carte). La numérotation des lignes d'une carte, en partant du haut, est :

12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Le paquet de cartes obtenu à la fin de la phase de numérisation contient le **programme source** qui, sur ce support, constitue une entrée valable pour le calculateur. A travers une unité périphérique spéciale (lecteur de cartes), l'ordinateur est capable d'interpréter correctement l'ensemble des symboles contenus dans les cartes.

Pour les petits calculateurs, on ne trouve presque jamais de lecteur de cartes ; c'est pourquoi la numérisation des instructions du programme doit être exécutée directement sur la console du système.

Ces deux opérations sont complètement équivalentes quant aux effets logiques. C'est pourquoi nous indiquons, sur les figures qui suivent et avec le symbole du fichier de cartes, toutes les entrées qui peuvent être assimilées à un ensemble de cartes perforées.

En d'autres termes, les programmes sources et les informations symboliques (avec un maximum de 80 colonnes), qu'ils résident sur disque ou qu'ils soient numérisés directement sur une console, seront indiqués par ce symbole.

Toutes les phases décrites jusqu'à présent permettent de passer de la recherche de la solution d'un problème à l'écriture d'un programme qui le réalise, de la rédaction des instructions à leur introduction dans l'ordinateur.

A ce point de rédaction, les instructions du programme sont encore exprimées sous forme symbolique, c'est-à-dire dans la syntaxe du langage.

Par exemple, si le programmeur a rédigé l'instruction

MOVE A TO B

il existera dans la machine une combinaison de caractère identique à celle qui est émise.

Puisque l'ordinateur n'est pas en mesure de comprendre la signification de la phrase MOVE A TO B, il est donc nécessaire de soumettre le programme à deux opérations successives.

Dans notre exemple, ces opérations traduisent MOVE en une série d'instructions en code binaire, de façon à ce qu'elle soit correctement interprétée et exécutée par l'ordinateur. On obtient ainsi le transfert (MOVE) demandé par la commande. Les deux opérations suivantes — compilation et l'édition de liens (linking) — doivent être exécutées dans l'ordre indiqué.

CODE DE PERFORATION 026

Symbole	Codification	Symbole	Codification	Symbole	Codification	Symbole	Codification	Symbole	Codification
A	12-1	P	11-7	0	0	Espace		>	6-8
B	12-2	Q	11-8	1	1	+	12	?	12-0
C	12-3	R	11-9	2	2	'	4-8	@	7-8
D	12-4	S	0-2	3	3	(	0-4-8	[	12-5-8
E	12-5	T	0-3	4	4	)	12-4-8	"	0-6-8
F	12-6	U	0-4	5	5	*	11-4-8	_	11-5-8
G	12-7	V	0-5	6	6	.	0-3-8	#	12-7-8
H	12-8	W	0-6	7	7	–	11	$	11-3-8
I	12-9	X	0-7	8	8	,	12-3-8	%	0-5-8
J	11-1	Y	0-8	9	9	/	0-1	&	2-8
K	11-2	Z	0-9			:	5-8	b	11-7-8
L	11-3					;	11-6-8	!	11-0
M	11-4					<	12-6-8		
N	11-5					=	3-8		
O	11-6								

Compilation et édition de liens

Quel que soit le langage dans lequel il est écrit, un compilateur est un programme lié au système d'exploitation. Il traduit les commandes symboliques du programme source en une série d'instructions machine, constituant un programme relogeable. Nous avons déjà évoqué le compilateur dans notre étude du langage Basic.

Les tâches exécutées pour compiler un programme sont communes à tous les types d'ordinateur.

Elles sont directement activées par le système d'exploitation grâce aux commandes intégrées dans des **cartes de contrôle.** De telles cartes, dont le format et le nombre sont en correspondance avec la machine utilisée, possèdent trois fonctions essentielles.

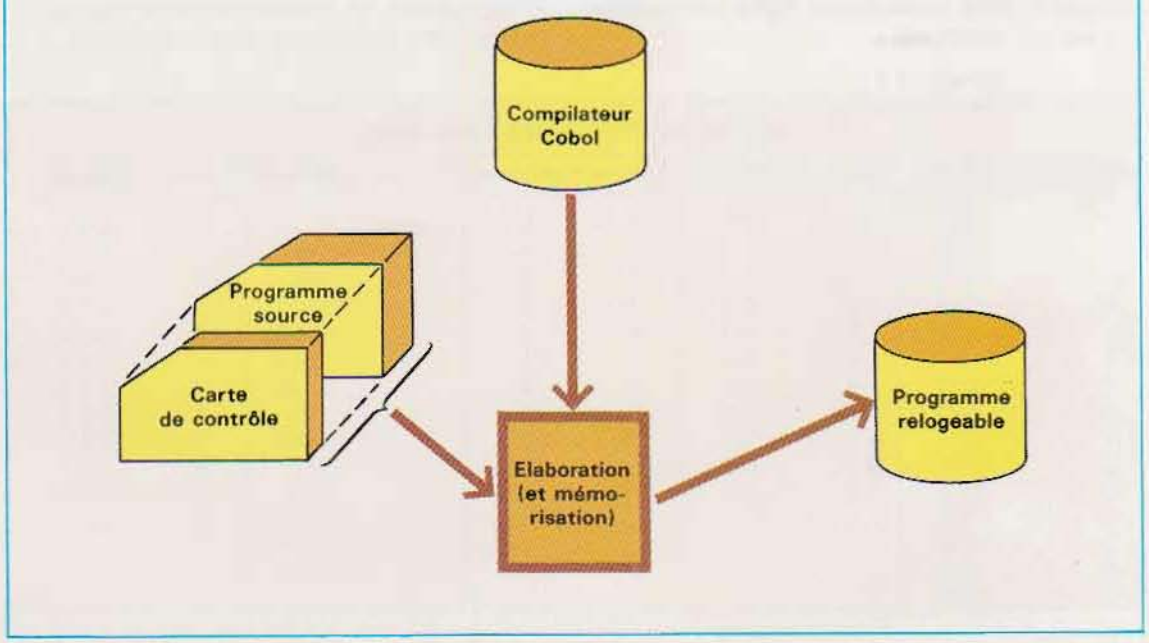
- Communiquer au système d'exploitation le nom du compilateur qui doit être chargé en mémoire (Cobol, Fortran, Basic...).
- Lancer l'opération de chargement du compilateur en faisant intervenir un programme appelé **chargeur** (loader).
- Fournir au compilateur le nom du programme source et le nom du fichier (dans le cas du Cobol sur disque) dans lequel doit être contenu le produit de la compilation (programme relogeable).

Ces opérations sont schématisées dans la figure ci-dessous. La traduction se fait grâce aux sous-programmes résidents de la bibliothèque système.

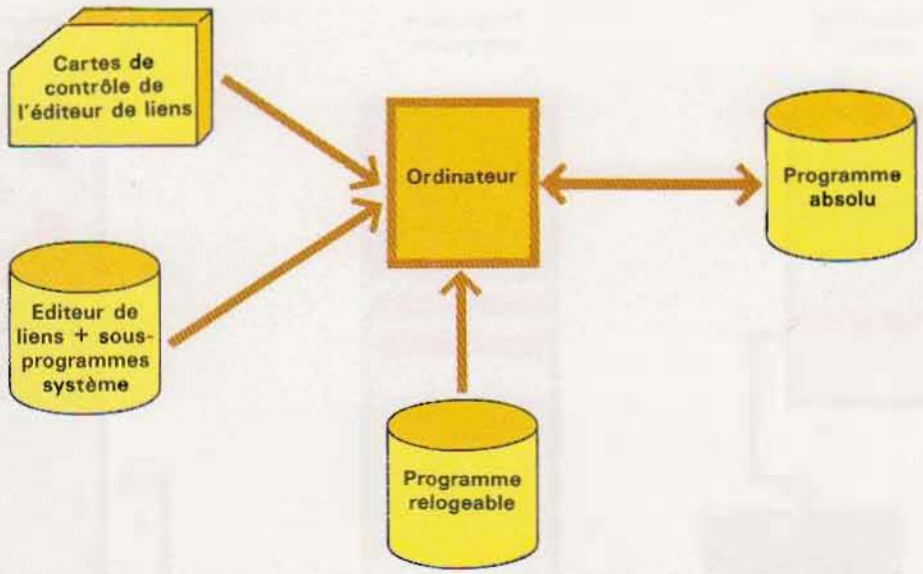
Prenons comme exemple les fonctions de lecture-écriture sur fichier : des routines d'entrée-sortie sont destinées à la réalisation de ces fonctions. Ainsi, le compilateur crée-t-il les points d'entrée nécessaires dans tous les sous-programmes appelés par les instructions du programme principal en enregistrant une série d'informations utiles à la tâche suivante (édition de liens).

Dans cette phase, seules les adresses provisoires des données définies par le programmeur sont enregistrées en fonction de la répartition générale de la mémoire. Ces adresses (relogeables) varient au moment où chaque sous-programme est réellement chargé en mémoire et assemblé de façon à adresser correctement les données du programme. De cette façon, le compilateur traite aussi bien des sous-programmes du système que des instruc-

SCHEMA DE PRINCIPE DE COMPILATION D'UN PROGRAMME



EDITION DE LIENS D'UN PROGRAMME

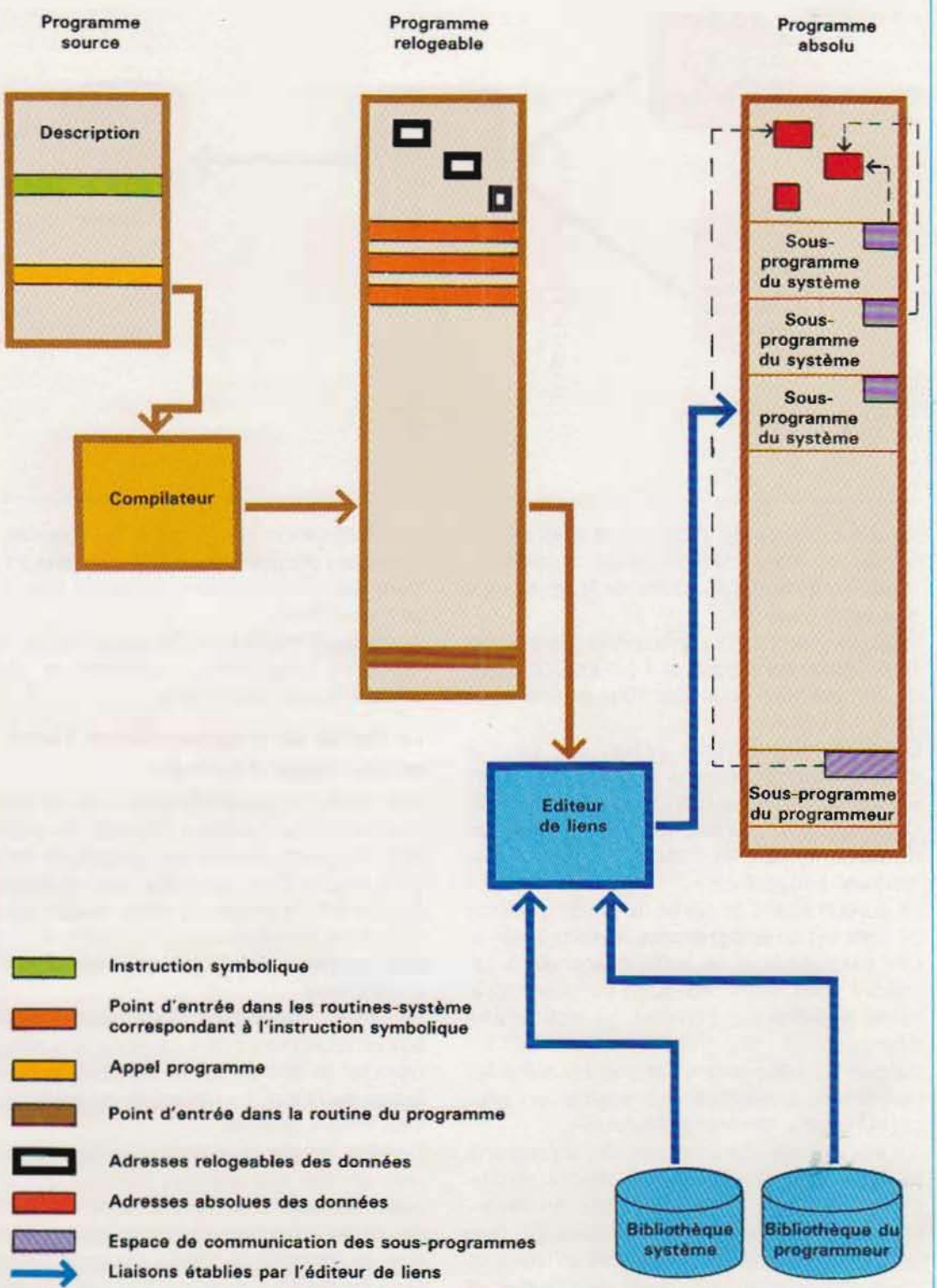


tions écrites par le programmeur et faisant partie des autres bibliothèques, à condition toutefois qu'on lui en fournisse le nom grâce à la carte contrôle.
En fin de phase de compilation, le programme objet réside sur disque et il est indispensable de lui associer tous les sous-programmes dont il a besoin.
C'est l'éditeur de liens (linker) qui, comme indiqué dans le schéma ci-dessus, exécute cette fonction. Une fois chargé en mémoire, ce programme résident (dans la bibliothèque du système) gère les liaisons entre les divers modules relogeables.
Le produit fourni en sortie de phase d'édition de liens est un **programme absolu**, c'est-à-dire exécutable (il se suffit désormais à lui-même). Le schéma de la page suivante représente le processus complet qui vient d'être décrit.
Jusque là, nous avons surtout énuméré les opérations auxquelles est soumis un programme pour devenir exécutable.
S'il est vrai que, dans l'ensemble, le programme source peut s'adapter à chaque micro-ordinateur, le compilateur Cobol, en revanche, diffère d'une machine à l'autre. Et, bien que nous n'en soyons pas encore arrivés à ce point, on comprend l'intérêt de pousser la standardisation du langage qui permet de porter les programmes d'une machine à l'autre en les compilant sur l'ordinateur chargé de leur exécution.
Après avoir décrit les différentes étapes de la vie d'un programme, analysons la phase d'écriture des instructions.

La feuille de programmation Cobol et les règles d'écriture

Une feuille de programmation est un formulaire normalisé destiné à accueillir les instructions du programmeur selon les règles du langage choisi. C'est une aide non négligeable. Cependant, le recours à cette feuille, qui est structurée en plusieurs zones distinctes, n'est pas indispensable à la rédaction correcte d'un programme.
Elle reste, cependant, fortement conseillée aux programmeurs débutants car elle oblige à reporter en clair certaines conventions spécifiques au Cobol. La rédaction du programme s'en trouve facilitée.
De plus, en cas d'utilisation de cartes perforées en tant que support d'entrée des données, la feuille de programmation accélère le travail de la personne chargée de la perforation en assurant la correspondance entre le programme écrit et sa codification sur carte.

PROCESSUS DE COMPILATION ET D'EDITION DE LIENS



FEUILLE DE PROGRAMMATION COBOL

PROGRAMMEUR _____ DONNEES _____ COBOL

PROGRAMME _____ 73 76 80 Page 1 3 de _____

PROGR.	CONT.	A	B	TEST
4	7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72	
0 1				
0 2				
0 3				
0 4				
0 5				
0 6				
0 7				
0 8				
0 9				
1 0				
1 1				
1 2				
1 3				
1 4				
1 5				
1 6				
1 7				
1 8				
1 9				

Un exemple de feuille de programmation est reproduit dans la page précédente. Remarquons qu'elle se décompose en deux parties. Le haut est destiné aux informations concernant la documentation, (nom du programmeur, nom du programme, date de sa rédaction). La partie inférieure recueille les instructions et les commentaires faisant partie du programme. En fait, à chaque ligne correspondra une carte. Les colonnes numérotées de la formule sont au nombre de 72 seulement, alors que les colonnes disponibles sur une carte sont au nombre de 80. En effet, le compilateur Cobol considère comme ne faisant partie du programme que les caractères perforés des colonnes 7 à 72.
Il est possible de marquer chaque carte perforée de façon à savoir avec certitude à quel programme elle appartient et où elle doit se trouver à l'intérieur du paquet de cartes (deck). Les colonnes 1 à 6 et 73 à 80 sont réservées à ce type d'informations. Les colonnes de 1 à 6 peuvent contenir un nombre ainsi décomposable : les trois premiers chiffres identifient le numéro de la feuille de programmation, les trois autres le numéro de la ligne à l'intérieur de la feuille. Le numéro complet perforé dans les six premières colonnes de chaque carte détermine ainsi de manière univoque la carte à l'intérieur du programme. Le nom du programme auquel chaque carte appartient peut être perforé dans les colonnes 73 à 80. Le numéro de la feuille (1 ÷ 3) et le nom du programme sont répartis dans la partie supérieure, alors que le numéro de série des lignes doit être nécessairement perforé sur la ligne correspondante (colonne 4 ÷ 6) de la partie inférieure.
La numérotation des cartes selon les conventions décrites n'est pas obligatoire en cas d'utilisation. Quand les cartes ne seront pas présentées dans un ordre correct, le compilateur Cobol signalera l'erreur.
Les colonnes de la partie inférieure (à l'exception des colonnes 4 à 6 dont l'utilisation a déjà été décrite), respectent des règles bien précises d'utilisation.

Colonne 7 — Dans cette colonne, il n'est possible de perforer que certains symboles particuliers. Un trait d'union (-) informe le compilateur Cobol que le premier caractère de la ligne correspondante est la suite du dernier mot de la ligne précédente, alors qu'un astérisque (*) permet au programmeur d'utiliser la ligne pour insérer un commentaire dans la liste du programme.

Colonne 8 ÷ 11 — Identifiée en clair par la lettre A correspondant à la colonne 8, cette zone est également appelée **espace A**. Elle est réservée à la perforation des noms des divisions, sections et paragraphes (parties d'un programme Cobol).

Colonne 12 ÷ 72 — Délimitée par un trait plus marqué avec la lettre B correspondant à la colonne 12, cette zone est réservée à la perforation des instructions du programme. Si, après avoir compilé la feuille entière, le programme a besoin d'insérer une ou plusieurs lignes, on peut éviter de réécrire le module complet en utilisant les lignes non numérotées et en adoptant une numérotation adaptée.

L'exemple ci-contre représente toutes les règles d'écriture que nous venons d'examiner, en particulier l'utilisation de la colonne 7 et l'insertion d'autres lignes dans la zone non numérotée de la feuille.
Cet exemple n'est fourni qu'à titre indicatif. Il semblera plus clair après l'exposé des syntaxes du langage Cobol.

Structure d'un programme Cobol

Chaque programme Cobol est structuré en division (DIVISION), sections (SECTION), paragraphes et périodes, hiérarchiquement organisés selon le schéma de la page 962. En d'autres termes, une division est composée de plusieurs sections, chaque section de plusieurs paragraphes et chaque paragraphe d'une ou de plusieurs périodes.

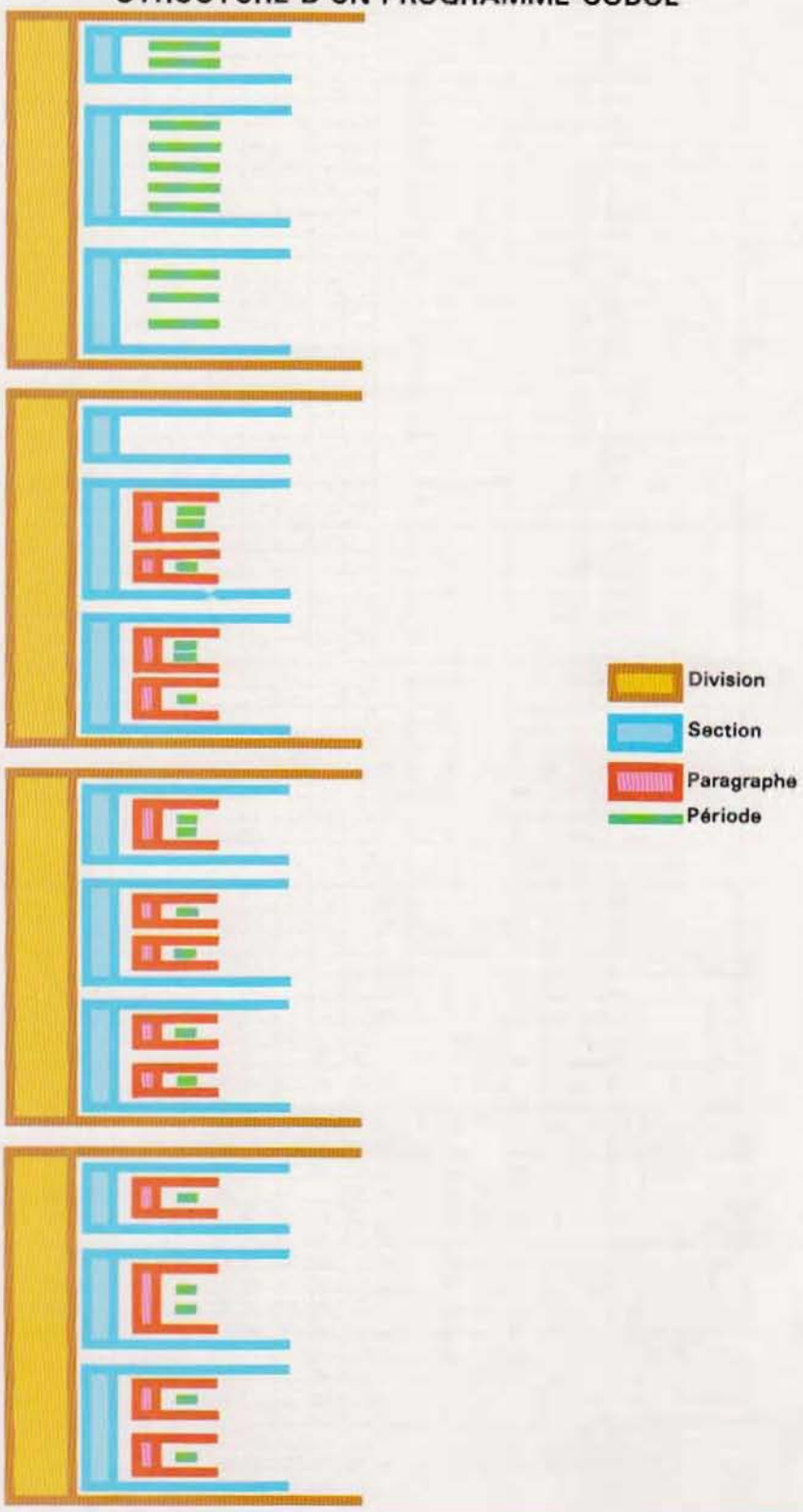
PROGRAMMEUR _____ DONNEES _____ COBOL

PROGRAMME _____ 73 76 80 Page 1 3 de _____

TEST

```
      Ø PROCÉDURE DIVISION.
      Ø DÉBUT SECTION.
      Ø OUVRE FICHIER.
      Ø*                  LE PARAGRAPHE "OUVRE FICHIER" OUVRE TOUS LES FICHIERS
      Ø*                  UTILISÉS PENDANT LE DÉROULEMENT DU PROGRAMME.
      Ø         OPEN ENTRÉE FICHIER-ANCIEN.
      Ø         OPEN SORTIE FICHIER-NOUVEAU.
      Ø LIRE ANCIEN
      Ø         READ DOSSIER-ANCIEN INTO REC-ANCIEN
      Ø                  AT END AFFICHAGE 'LECTURE DU FICHIER-ANCIEN TERMINÉE'
      Ø-      'CORRECTEMENT' SUR IMPRIMANTE
      Ø                  GOTO FIN.
      Ø TRAITEMENT ANCIEN.
      Ø         MOVE...
      Ø         ADD...
      Ø LIRE MISE A JOUR.
      Ø         READ MISE A JOUR.
      Ø                  AT END....
      Ø
     ØG5        OPEN ENTRÉE MISE A JOUR.
     155        SUBSTRACT...
```

STRUCTURE D'UN PROGRAMME COBOL



La **phrase** comporte une ou plusieurs instructions délimitées par un point.
Par exemple, l'ensemble des instructions suivantes :

```
MOVE A TO B
MOVE C TO D
COMPUTE E = B — C.
```

Le **paragraphe** se compose de plusieurs phrases.
Il est identifié par un nom choisi par le programmeur et comportant au maximum 30 caractères, c'est-à-dire : lettres de l'alphabet anglais, chiffres de 1 à 9 et signe (—) employé comme trait d'union.
Sur la carte perforée, le nom du paragraphe commence à la colonne 8 et se termine par un point. Par exemple :

```
CALCUL-TVA.
    COMPUTE TVA = MONTANT * 18/100
    .....
```

Le paragraphe va du nom qui l'identifie jusqu'au paragraphe suivant.
Plusieurs paragraphes forment une **section,** également identifiée par un nom soumis aux mêmes règles que le paragraphe. Le mot SECTION permet au compilateur de différencier les deux entités. Si on désire traiter une série d'instructions comme une section, on doit faire suivre le nom attribué à la section pas le mot SECTION.
Ainsi la série de cartes perforées

```
CALCUL-TVA SECTION.
CALCUL.
    COMPUTE TVA = MONTANT * 18/100.
```

définit la section CALCUL-TVA.
Une section commence par un nom de paragraphe et finit au début de la section suivante.
Il faut d'ailleurs préciser que, sauf instructions explicites de saut données par le programmeur, la fin d'un paragraphe ou d'une section ne signifie pas que les calculs s'arrêtent à ce niveau.
Le but de la subdivision du programme Cobol en paragraphes et sections sera expliqué par la suite. Les subdivisions sont du ressort du programmeur. En revanche, les divisions sont obligatoires dans un programme Cobol et vont toujours par quatre dans un ordre déterminé.
Le déroulement séquentiel ne peut absolument pas être modifié par le programmeur. Il permet au compilateur de traiter, selon une suite logique, toutes les informations indispensables à l'exécution correcte des instructions du programme.

IDENTIFICATION DIVISION. Cette division annonce au compilateur la nature du traitement à exécuter en décrivant les relations établies entre certains organes physiques et le système. Elle se compose de deux sections.

CONFIGURATION SECTION. Elle constitue la première section. Elle décrit le modèle d'ordinateur qui compile le programme et l'exécute.
La seconde section, **INPUT-OUTPUT SECTION**, fournit les noms de fichiers consultés par le programme et ceux des supports physiques qui les stockent.

DATA DIVISION. La fonction de cette division consiste essentiellement à réserver, pour le programme principal, des zones mémoire, dûment dimensionnées ; ensuite à les associer, de façon univoque, à des mnémoniques créés par le programme et utilisés au cours de ce travail.
On distingue schématiquement trois catégories dans les zones mémoire réservées à la DATA DIVISION (division données).

- Une première catégorie d'allocation mémoire est celle que le compilateur réserve aux opérations de lecture et/ou d'écriture sur les fichiers. La définition de ces zones est réalisée grâce à la première section de la DATA DIVISION, c'est-à-dire la **FILE SECTION** (section fichier).
- La deuxième catégorie concerne les zones mémoire réservées par le programme et entièrement gérées par lui. Les définitions de ces allocations mémoire sont regroupées dans la **WORKING STORAGE SECTION** (section zone de travail).
- La troisième catégorie d'allocation en mé-

moire centrale concerne les zones réservées aux échanges d'information entre le programme principal et d'éventuels sous-programmes externes.
Le nom de la section où sont définies ces zones est justement **LINKAGE SECTION** (section de lien).
La LINKAGE SECTION n'a d'utilité que si le programme fait référence à des programmes externes. En résumé, la structure complète de la DATA DIVISION est la suivante :

DATA DIVISION.
FILE SECTION.
    Description des fichiers
WORKING-STORAGE SECTION.
    Description des données indépendantes
LINKAGE SECTION.
    Description des données échangées avec des sous-programmes.

PROCEDURE DIVISION. Cette division, dite aussi division algorithmes, contient toutes les instructions Cobol que le programmeur écrit pour traduire les opérations à effectuer sur les données. La division est généralement composée de paragraphes et de sections avec un regroupement logique des instructions.
La description sommaire d'un programme Cobol et des fonctions assurées par chacune de ses parties donne une idée du caractère particulier de ce langage.
Examinons maintenant en détail chaque division afin de bien comprendre l'organisation et la syntaxe du Cobol. Dans cet exposé, nous emploierons les conventions décrites dans le tableau qui se trouve en bas de page.

IDENTIFICATION DIVISION

La division identification est la seule division comportant uniquement des paragraphes. Le nom de la division, comme ceux de ses paragraphes, se situe obligatoirement à partir de la colonne 8 (ZONE A).
Le tableau suivant donne le format complet de la division identification.

FORMAT DE LA DIVISION IDENTIFICATION

IDENTIFICATION DIVISION.
PROGRAM-ID. nom-programme.
[AUTHOR. nom-auteur.]
[INSTALLATION. nom-de-l'installation.]
[DATE WRITTEN. date-d'écriture.]
[DATE-COMPILED. date-de-compilation.]
[SECURITY. type-de-sécurité.]
[REMARKS. commentaires.]

En accord avec nos conventions, on voit bien que le seul paragraphe obligatoire de cette division est PROGRAM-ID (program identification), tous les autres étant optionnels et ne servant que de commentaires. Ces paragraphes, lorsqu'ils existent, doivent néanmoins respecter l'ordre indiqué. REMARKS

CONVENTIONS UTILISEES

Symbole	Description	Convention
___	Soulignement	Indique une entité obligatoire dans la syntaxe d'une instruction.
[ ]	Crochets	Referment une entité, une clause ou une série de clauses supplémentaires et optionnelles dans le format d'une instruction.
{ }	Accolades	Renferment une série d'entités alternatives dans lequel le programmeur opère une sélection.
	Caractères maj.	Tous les noms réservés du Cobol s'écrivent en lettres capitales.
	Caractères min.	Tous les domaines ou entités s'écrivent en minuscules. Les noms et le format sont librement attribués par le programmeur.
.....	Points de suspension	Indiquent que la clause (ou l'entité précédente) peut se trouver plusieurs fois dans la même instruction.

permet d'insérer un assez long commentaire qui s'achève au moment où le compilateur rencontre le premier point.
Exemple de division identification complète :

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXEMPLE-ID.
AUTHOR. MICHEL-DUPOND-LYON.
INSTALLATION. PARIS.
DATE-WRITTEN. JUIN 84.
DATE-COMPILED. 10/06/84.
SECURITY. AUCUNE.
REMARKS. CECI EST UN EXEMPLE DU FORMAT COMPLET DE LA DIVISION
         IDENTIFICATION D'UN PROGRAMME COBOL.
```

ENVIRONNEMENT DIVISION

Lorsque nous avons décrit les raisons pour lesquelles une normalisation est intervenue au niveau de certains langages et notamment du Cobol, nous avons parlé de portabilité, c'est-à-dire de la possibilité d'exécuter un même programme sur différentes machines. Cette notion est malheureusement rarement respectée. Théoriquement, la standardisation permet de structurer les instructions en conformité avec la syntaxe du Cobol, de façon à ménager une indépendance totale vis-à-vis de l'ordinateur. Mais elle ne peut imposer des notations qui obligeraient les constructeurs à modifier la philosophie même du traitement. Ainsi, le programme, qui est directement en cheville avec le système employé, fait nécessairement référence au nom du support logique contenant les données (fichier) et au nom du support physique (type d'unité) où il réside : disque, bande, lecteur de cartes, etc.
Si on désire exécuter un programme sur deux machines fonctionnant avec des systèmes différents, on sera contraint non seulement de le recompiler, mais aussi de modifier la partie décrivant le type d'environnement informatique d'accueil (ENVIRONNEMENT DIVISION).
Les différents paragraphes et sections de cette division seront analysés par la suite, en laissant au lecteur la tâche de paramétrer le programme, en accord avec les spécifications décrites dans les manuels fournis par les constructeurs de l'ordinateur utilisé. Rappelons que la division environnement doit figurer dans tout programme, que les noms de sections et de paragraphes doivent se trouver à partir de la colonne 8 et les informations associées débuter en colonne 12 et au-delà.

FORMAT DE LA DIVISION ENVIRONNEMENT

```
ENVIRONNEMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. non-ordinateur.
OBJET-COMPUTER. non-ordinateur.
[SPECIAL-NAMES.]
[IMPUT-OUTPUT SECTION.]
[FILE CONTROL.]
```

CONFIGURATION SECTION

Le nom même de cette section est très explicite. Sa fonction est d'identifier aussi bien le système qui compile le programme (SOURCE-COMPUTER) que celui qui assure son exécution (OBJECT-COMPUTER).
Le nom du système apparaissant dans ces deux paragraphes doit être conforme aux indications du constructeur.

SPECIAL NAMES. Optionnel, ce paragraphe est très utile dans la plupart des applications du Cobol. Il rend des services surtout dans le domaine de la gestion : il permet d'imprimer un nombre en lui associant automatiquement le symbole.
Nous verrons, plus tard, le détail de toutes ces opérations. Avant tout, pour l'instant, il est important de montrer comment un compilateur Cobol traite de manière analogue le symbole de toute autre monnaie.
La modification est très simple : le remplacement du $ comme symbole de devise est annoncé au compilateur dans le paragraphe SPECIAL-NAMES.
Supposons que l'on veuille traiter de francs français (symbole F). Le format du paragraphe sera :

SPECIAL-NAMES.
CURRENCY SIGN IS 'F'.

Le nouveau symbole ne peut être aucun des signes suivants :
— chiffres de 0 à 9
— caractères alphabétiques A B C D L P R S V X Z
— caractères spéciaux * + — , . ; () " / =

Puisque le Cobol (comme d'autres langages de programmation), a été conçu aux Etats-Unis, il est normal que la représentation des nombres décimaux soit conforme aux conventions américaines. C'est le point (.) qui est utilisé pour différencier les entiers et les décimaux, la virgule étant réservée à la séparation des nombre entiers en groupes de trois chiffres (centaines, milliers, etc). Ainsi, le nombre 28 432 807,57 (en notation européenne) s'écrira dans la représentation américaine :

28,432,807.57

Pour imprimer des nombres selon la notation européenne, il suffit d'insérer dans le paragraphe SPECIAL-NAMES l'information suivante :

DECIMAL POINT IS COMMA

qui signifie littéralement « le point décimal est la virgule ». Les conventions adoptées dans le paragraphe SPECIAL-NAMES sont valables de manière définitive pour tous les programmes et sans possibilité d'alternance.

FILE-CONTROL. Grâce à ce paragraphe, le programmeur indique au compilateur le nom des divers fichiers employés par le programme, ainsi que leurs supports physiques.
Supposons, par exemple, que l'on doive utiliser un fichier DONNEES ETAT CIVIL contenu sur un disque catalogué par le système sous le nom ETACIV. Si le programmeur décide de l'appeler en « interne » dans son programme FILE-ETA, il devra l'associer au nom « externe » ETACIV. Le système établira ainsi la correspondance désirée, ce qui permettra de retrouver les données à traiter en phase d'exécution.
Cette association est créée dans le paragraphe FILE-CONTROL par la notation suivante :

SELECT FILE-ETA
ASSIGN TO DISC ETACIV.

La clause SELECT, unique pour chaque fichier traité, a généralement le format spécifié dans le tableau ci-dessous.
Même pour la définition du nom du support physique d'un fichier, il faut suivre les instructions du constructeur. De ce fait, l'exemple donné n'est valable que pour certains systèmes, les différences qui apparaissent étant directement liées à la diversité du matériel mis en œuvre. Le mot OPTIONAL, qui apparaît dans le format général de la clause SELECT, annonce au compilateur que le fichier correspondant peut aussi faire défaut en phase d'exécution. Mais l'absence de fichier en phase d'exécution ne signifie nullement qu'il ne se trouve pas physiquement sur le sup-

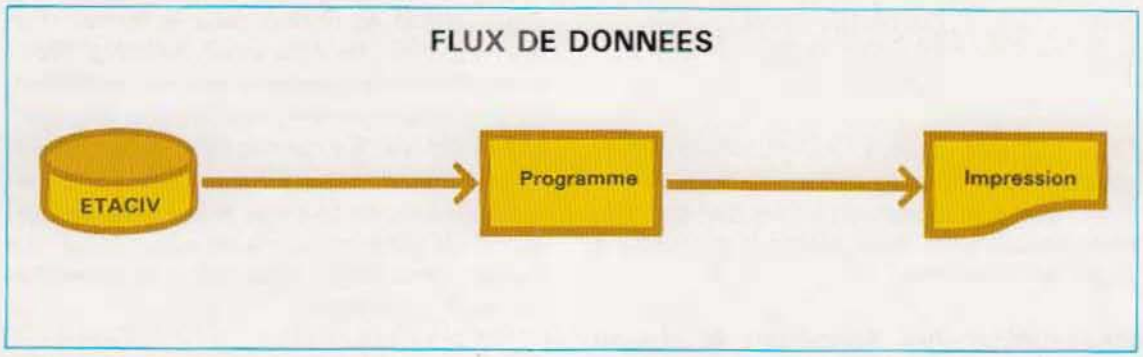
FORMAT DE LA CLAUSE SELECT

SELECT [OPTIONAL] Non-interne-fichier ASSIGN TO non support

Non-fichier-interne	C'est le nom que l'on veut donner au fichier à l'intérieur du programme.
Nom-support	C'est le nom sous lequel le fichier est connu du système et du support matériel sur lequel ce fichier se trouve.

Les supports matériels peuvent être

CARD-READER	= lecteur de cartes
CARD-PUNCH	= perforatrice de cartes
PRINTER	= imprimante
TAPE	= bande magnétique
DISC ou DISK	= disque



port matériel défini. Tout au plus n'a-t-il pas été assigné au programme. Cette assignation est réalisée, de manière externe au programme, par des ordres de contrôle en rapport avec l'ordinateur employé.
Pour clarifier les idées, prenons le cas d'un programme devant appeler le fichier de l'exemple précédent et imprimer son contenu. Le diagramme du flux d'information correspondant est représenté par le schéma ci-dessus.
L'INPUT-OUTPUT SECTION du programme contient la sélection des fichiers correspondants ainsi que leur dénomination internes :

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-ETA
    ASSIGN TO DISC ETACIV.
    SELECT IMPRESSION ASSIGN TO
    PRINTER.
```

En phase d'exécution, il faut signaler au système que le programme utilisera le fichier ETACIV, sans qu'il soit nécessaire d'assigner l'imprimante, toujours disponible puisque gérée par le système lui-même. La séquence suivante illustre l'exécution de ces opérations dans un ordre fonction du système employé :

1re carte

ordre-d'assignation-fichier ETACIV

2e carte :

ordre-d'exécution-programme
nom-programme.

DATA DIVISION

Comme déjà mentionné, la fonction de la data division (division données) est de réserver au programme des zones de mémoire centrale, correctement dimensionnées, puis de les associer de manière univoque aux mnémoniques créés par le programmeur et utilisés en cours d'exécution.
Une première catégorie d'allocations mémoire est celle que le compilateur réserve aux opérations de lecture et/ou d'écriture sur fichiers. La description de ces zones est donnée dans la première section de la division données, la **FILE-SECTION** (section-fichier). Dans la deuxième catégorie, on trouve les zones mémoire que le programmeur se réserve en fonction de ses propres besoins et qu'il gère lui-même. Les définitions de ces allocations mémoire sont regroupées dans la **WORKING-STORAGE-SECTION.**

FILE SECTION

La section FILE SECTION fournit au compilateur les caractéristiques de tous les fichiers utilisés par le programme. Naturellement, cette section est omise quand son exécution ne nécessite aucun fichier en entrée ou en sortie. Ce programme existe, certes, mais ne présente pas d'intérêt en gestion là où les traitements consistent surtout à traiter un volume important de données.
Chaque fichier est défini par l'indicateur de niveau, le nom et une série de clauses, dont certaines sont facultatives, qui précisent ses caractéristiques. L'indicateur de niveau est repéré par le mot réservé FD, abréviation de File Description (description-fichier), enregistré à partir de la colonne 8 de la phrase de description :

```
DATA DIVISION.
FILE SECTION.
FD ETACIVIL
```

Dans notre exemple, ETACIVIL désigne le fichier interne du programme.
Bien noter que le nom du fichier doit toujours se trouver à partir de la marge B, ou mieux à partir de la colonne 12.

Description des données : la clause BLOCK CONTAINS. Avant d'analyser la suite de la phase de description des fichiers, il convient de revenir sur cette notion de fichiers, en particulier sur ceux qui sont séquentiels.
Un fichier séquentiel, sur support disque, est lu enregistrement par enregistrement. Cette méthode est d'une efficacité toute relative, surtout quand il s'agit d'enregistrements nombreux et courts. En effet, chaque lecture ou écriture sur disque implique des opérations mécaniques, telles que le positionnement de la tête de lecture sur la piste ou sur le secteur approprié, avec des temps d'exécution en moyenne 1000 fois supérieurs aux opérations courantes en mémoire.
Il est donc préférable (quand cela est possible) de structurer les enregistrements de lecture ou d'écriture sur le disque de telle façon qu'on puisse traiter simultanément plusieurs enregistrements.
En d'autres termes, le Cobol transfère toujours un bloc à la fois. S'il n'existe pas de clauses relatives au « blocage » des fichiers, le compilateur réserve à toutes ces opérations une zone mémoire égale à la longueur de l'enregistrement.
Le système sera alors contraint de réaliser autant d'accès disque qu'il y a d'enregistrements. En revanche, si le programmeur détermine un blocage des enregistrements, le compilateur réservera une zone tampon susceptible de les contenir.

Ainsi l'accès au disque, pour la lecture d'un nouveau bloc, ne sera lancé qu'après traitement des enregistrements du bloc précédent. La procédure est schématisée par le graphique ci-contre. Sur cet exemple, le système lit et traite des enregistrements d'un fichier séquentiel avec un blocage = 3. La description du fichier comporte la clause suivante (dont le format général est donné dans le tableau situé en bas de page) :

```
BLOCK CONTAINS 3 RECORDS
```

Différentes clauses sont admises :

```
BLOCK CONTAINS 10 RECORDS
BLOCK CONTAINS 1 TO 10 RECORDS
BLOCK CONTAINS 100 CHARACTERS
BLOCK CONTAINS 30 TO 900 CHARACTERS
```

Avec la mention CHARACTERS, le nombre de caractères déclarés comprend aussi les caractères de contrôle.
La clause BLOCK peut être omise, ce qui peut se traduire par

```
BLOCK CONTAINS 1 RECORDS
```

La clause LABEL RECORD. Cette clause (voir tableau immédiatement ci-dessous) figure toujours dans la description d'un fichier. Son rôle est de préciser si un fichier a ou aura des étiquettes (labels).

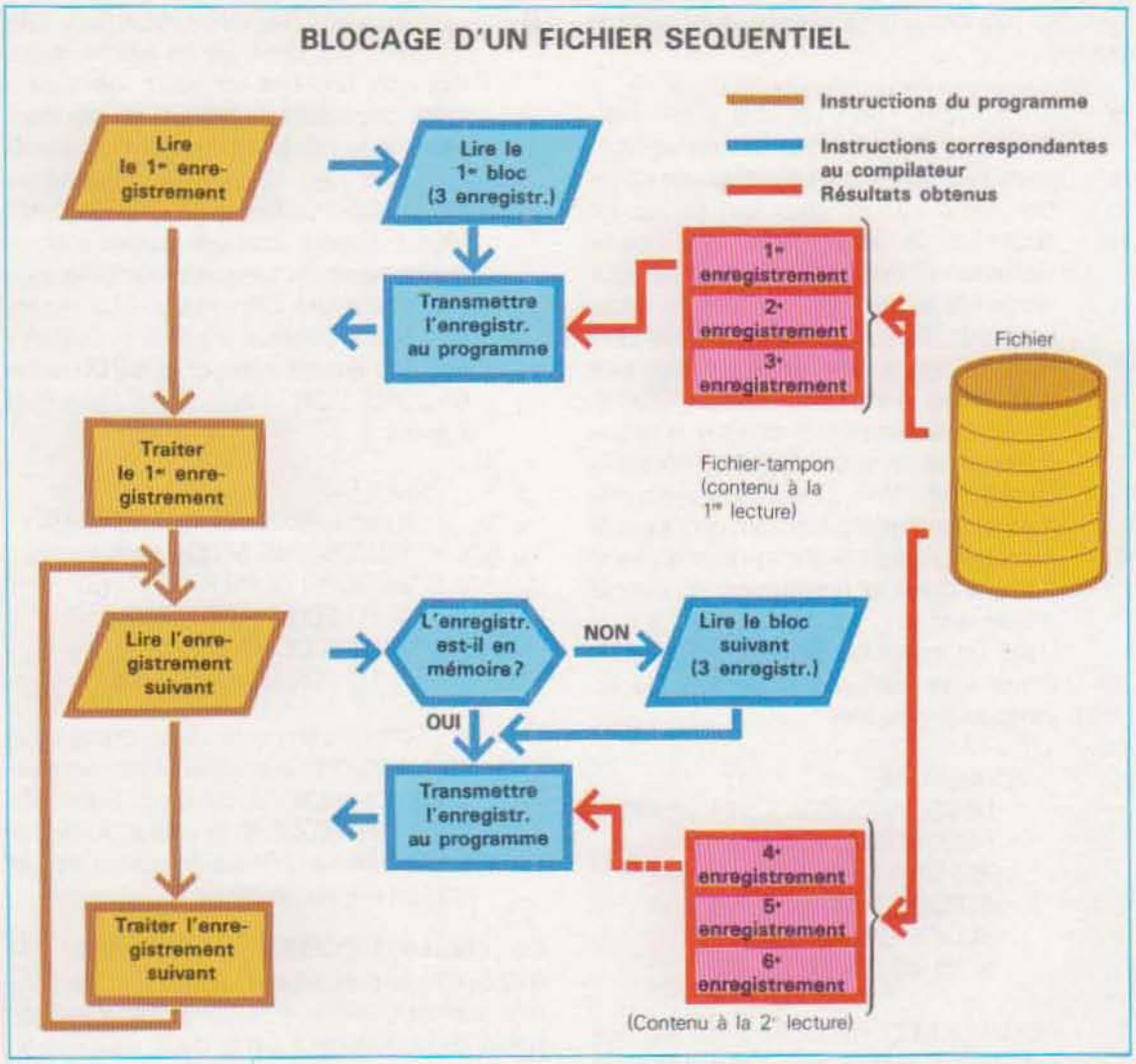
FORMAT GENERAL DE LA CLAUSE LABEL RECORD

```
LABEL { RECORD IS   } { OMITTED  }
      { RECORD ARE  } { STANDARD }
```

FORMAT DE LA CLAUSE BLOCK CONTAINS

```
BLOCK CONTAINS [entier-1 TO] entier-2 { CHARACTERS }
                                      { RECORDS    }
```

BLOCAGE D'UN FICHIER SEQUENTIEL



Comme on peut le constater, le mot OMITTED précise que le fichier n'a pas d'étiquettes ; par contre, le mot STANDARD signale que les fichiers existants, ou devant être créés, ont des étiquettes standard prévues par le système.
Les fichiers attribués au lecteur de cartes, à la perforatrice ou à l'imprimante utilisent la clause.

LABEL RECORD IS OMITTED

La clause RECORDING MODE. Les enregistrements contenus dans un fichier sont, en général, de même dimension. Dans certains cas, il est toutefois possible de traiter d'autres types de supports d'enregistrement. Le format général de la clause est le suivant :

FORMAT DE LA CLAUSE RECORDING MODE

RECORDING MODE IS { F / V / U / S }

sauter la clause revient à écrire RECORDING MODE IS F : le compilateur suppose que les enregistrements du fichier ont une longueur fixe (Fixed). Cette longueur est calculée automatiquement selon la description de l'enregistrement, comme nous le verrons plus loin.

Signification des lettres apparaissant dans la clause :

V C'est l'initiale de « variable ». Elle indique explicitement que les enregistrements du fichier ont une longueur variable. Nous venons plus loin (lorsqu'on abordera la description de l'enregistrement) comment le compilateur reconnaît le caractère variable de cette longueur. Il faut toutefois garder présent à l'esprit que, pendant l'écriture d'un fichier avec RECORDING MODE V, le système associe à chaque enregistrement un champ où l'on retrouve la longueur du bloc. Ces champs supplémentaires sont entièrement gérés par le système. Aussi ne doit-on pas en tenir compte dans la description de l'enregistrement.
Voici un exemple de description d'un fichier composé d'enregistrements de longueurs variables :

```
FD ARCHIVE
   LABEL RECORD IS STANDARD
   RECORDING MODE IS V
   RECORD CONTAINS
   1 TO 200 CHARACTERS
   BLOCK CONTAINS
   1 TO 45 RECORDS.
```

La clause RECORD CONTAINS n'a pas encore été décrite mais elle est facile à interpréter.

U C'est l'initiale de undefined (indéfini). Elle caractérise un fichier dont les enregistrements peuvent avoir une longueur quelconque sur un intervalle déterminé. Dans ce cas, la reconnaissance du type d'enregistrement traité est réalisée par le programmeur au moyen d'un critère quelconque. Dans RECORDING MODE U, il faut éliminer la clause BLOCK, comme le montre l'exemple suivant :

```
FD FILE-U
   LABEL RECORD IS STANDARD
   RECORDING MODE IS U.
   RECORD CONTAINS
   100 TO 200 CHARACTERS.
```

S La clause RECORDING MODE IS S définit un fichier dont les enregistrements logiques ont une longueur de caractères dépassant la dimension du bloc. Les enregistrements sont répartis (spanned, en anglais) sur plusieurs blocs. Un fichier avec RECORDING MODE S peut contenir autant d'enregistrements à longueur variable que fixe. C'est une information que reconnaît le compilateur d'après la description de l'enregistrement et/ou la clause RECORD CONTAINS. Dans l'exemple suivant

```
FD FILE-S
   LABEL RECORD IS STANDARD
   RECORDING MODE IS S
   RECORD CONTAINS
   1 TO 500 CHARACTERS
   BLOCK CONTAINS
   1 TO 300 CHARACTERS.
```

le fichier comporte des enregistrements de longueur variable pouvant dépasser la capacité du bloc. Dans RECORDING MODE S, la valeur du blocage doit être exprimée en caractères et non en enregistrement.

La clause RECORD CONTAINS. Elle mentionne explicitement la longueur de l'enregistrement, même si le compilateur est capable de la déduire à partir de la description de l'enregistrement ou des enregistrements contenus dans le fichier (voir format en haut de la page suivante).

Structure des enregistrements. Lors de l'analyse des diverses clauses de description de fichier, nous avons rappelé que le compilateur est en mesure de déduire, de la description de l'enregistrement, ses dimensions en caractères. Le compilateur doit réserver en mémoire autant d'allocations que nécessaire pour contenir l'enregistrement. En effet, c'est dans cette zone qu'on transfère, au programme, un enregistrement en provenance du disque, ou bien que le programme traite l'enregistrement avant de l'écrire sur le disque. En général, un enregistrement est composé d'une série de caractères groupés en champs. Alors qu'une opération sur fichier

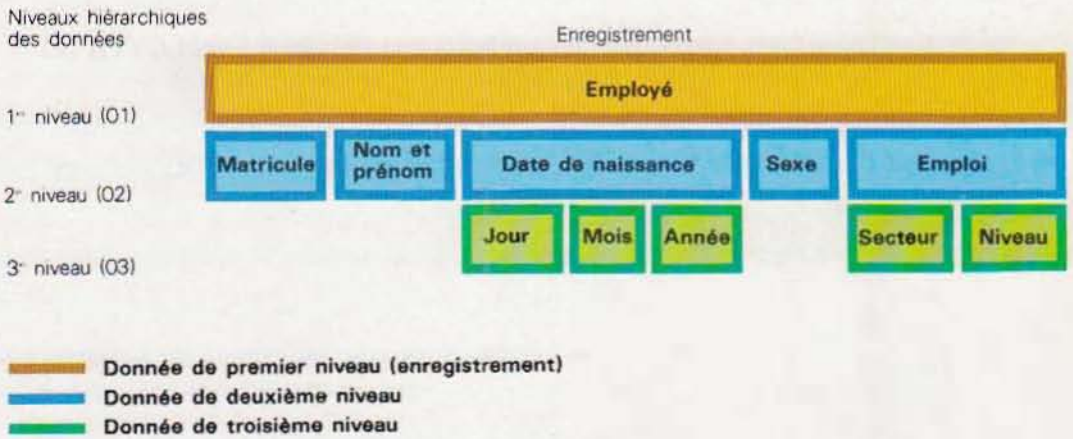
FORMAT DE LA CLAUSE RECORD CONTAINS

RECORD CONTAINS [entier-1 TO] entier-2 CHARACTERS

Entier-1 indique le nombre minimum de caractères pouvant constituer un enregistrement dans le cas de fichier dont les enregistrements sont de longueur variable, de type indéfini ou de type "spanned" (réparti)

Entier-2 indique le nombre maximum de carctères de l'enregistrement.

STRUCTURE D'UN ENREGISTREMENT



(par exemple une lecture) transfère en mémoire un enregistrement entier, le programme a la possibilité d'adresser et de traiter l'enregistrement soit dans son intégralité, soit dans les champs qui le constituent, à condition qu'ils aient été clairement définis.

Examinons, par exemple, le fichier EMPLOYES qui contient des enregistrements de longueur fixe de 100 caractères.

La structure de l'enregistrement est donnée par le schéma ci-dessus. Il établit un ordre hiérarchique qui permet d'adresser et de modifier, totalement ou en partie, la zone mémoire qui reçoit l'enregistrement EMPLOYE. A cette hiérarchie, on associe des niveaux (voir listing n° 1 de la page suivante). Chaque champ utilise une ligne (une carte) avec, au début de la carte, le numéro du niveau, compris entre 01 et 49, qui indique le niveau hiérarchique de ce champ par rapport au précédent. Le niveau 01 désigne l'enregistrement tout entier. Modifier le contenu du champ EMPLOYE équivaut, par exemple, à modifier en une seule fois les contenus de tous les champs de niveaux supérieurs.

A l'inverse, il est possible de changer le contenu du champ ANNEE-DE-NAISSANCE (niveau 03), sans qu'aucun autre champ de l'enregistrement EMPLOYE ne soit affecté.

Il n'est pas nécessaire que l'ordre des numéros de niveau soit croissant ; il est même conseillé d'attribuer à des champs contigus des numéros de niveau ayant un pas d'au moins cinq unités, ce qui permettra de décrire en détail n'importe quel champ (voir listing n° 2 page 972).

Le mot PIC apparaissant dans les listings est l'abréviation du mot réservé PICTURE. Il définit, grâce aux signes qui le suivent, la longueur et le type de champ. Le type de caractère que le champ peut recevoir est défini par le symbole qui suit immédiatement PIC :

DESCRIPTION DES ENREGISTREMENTS DU FICHIER EMPLOYES (1)

```
FD  EMPLOYES
    LABEL RECORD STANDARD
    RECORDING MODE IS F
    RECORD CONTAINS 100 CHARACTERS.
01  EMPLOYE.
    02  MATRICULE                  PIC 999.
    02  PRENOM ET NOM              PIC X(30).
    02  DATE DE NAISSANCE.
        03  JOUR                   PIC 99.
        03  MOIS                   PIC 99.
        03  ANNEE                  PIC 99.
    02  SEXE                       PIC A.
    02  EMPLOI.
        03  SECTEUR                PIC 9(3).
        03  NIVEAU                 PIC 9(3).
```

DESCRIPTION DES ENREGISTREMENTS DU FICHIER EMPLOYES (2)

```
FD  EMPLOYES
    LABEL RECORD STANDARD
    RECORDING MODE IS F

01  EMPLOYE.
    05  MATRICULE                  PIC 9(3).
    05  PRENOM ET NOM              PIC X(30).
    05  DATE DE NAISSANCE.
        10  JOUR                   PIC 9(2).
        10  MOIS                   PIC 9(2).
        10 ANNEE                   PIC 9(2).
    05  SEXE                       PIC A.
    05  EMPLOI.
        10  SECTEUR                PIC 9(3).
        10  NIVEAU                 PIC 9(3).
    05  FILLER                     PIC X(54).
```

X = caractères alphanumériques (lettres, chiffres et espaces)
A = caractères alphabétiques (lettres de A à Z et espaces).
9 = caractères numériques (chiffres de 0 à 9)

Examinons le champ NOM-ET-PRENOM : la notation PIC X (30) annonce au compilateur que la zone mémoire réservée à ce domaine contient au maximum 30 caractères alphanumériques. Remarquons que le fait d'écrire 05 MATRICULE PIC 9 (3) ou bien 05 MATRICULE PIC 999 revient exactement au même pour le compilateur. Cependant, ce second format est moins pratique pour le programmeur, surtout si le champ est très long.
Dans le cas des champs définis ci-dessous, comme par exemple DATE-DE-NAISSANCE, le compilateur n'exige pas la définition des dimensions et du type de champ, dans la mesure où il peut en calculer la longueur (égale à la somme des longueurs des champs le constituant). De plus, il considère toujours le champ de niveau supérieur comme alphanumérique. On reviendra sur ce sujet.
Bien que différents en apparence, les exemples illustrés par les deux listings ci-dessus sont tout à fait semblables. Dans le premier cas, la clause RECORD CONTAINS 100 CHARACTERS ayant été précisée de manière explicite, le compilateur réserve une zone de 100 caractères à l'enregistrement, même si le total des champs de l'enregistrement vaut 46 caractères. Nous avons alors dans l'enregistrement un champ de 54 caractères qui, n'ayant pas été défini, ne peut être employé dans le programme.

Dans le deuxième exemple, on arrive au même résultat en insérant dans la description de l'enregistrement le champ de 54 caractères qui a pour nom le mot réservé FILLER (remplisseur).
Un champ FILLER occupe en effet des espaces mémoire, mais le programme ne peut lui faire référence.
Aussi le compilateur réservera-t-il automatiquement 100 caractères après avoir additionné les différentes longueurs, ce qui justifie l'omission de la clause RECORDING MODE F devenue superflue.
En effet, si le fichier avait été de longueur variable, il aurait fallu insérer soit la clause RECORD CONTAINS nombre minimum TO nombre maximum CHARACTERS, soit la définition, au niveau 01, d'un autre enregistrement de longueur différente.

WORKING-STORAGE SECTION

C'est la section de la division données où se trouvent décrites données et zones de travail du programme. Celles-ci accueillent les valeurs fixes ou temporaires générées au cours du traitement.
Soulignons que la majorité des règles ayant trait à la description des champs dans la WORKING-STORAGE (que nous verrons par la suite), sont également applicables aux enregistrements de la FILE SECTION.
Nous avons sciemment évité de détailler, ce qui n'est pas indispensable à ce niveau (même si le Cobol le prévoit) du moment que c'est correct du point de vue de la syntaxe.
Donc, sauf indication contraire, toutes les règles décrites doivent désormais être considérées comme valables pour la description de FILE SECTION.

Le niveau 77. Comme vous le savez maintenant, on associe, à la description d'un champ, un numéro de niveau qui signale toujours sa position hiérarchique à l'intérieur d'une zone mémoire.
Un champ est **élémentaire** si on ne peut le décomposer en d'autres champs de niveau inférieur. Soit l'exemple du champ DATA-TRAITEMENT défini par :

```
01 DATA-TRAITEMENT.
   05    JOUR        PIC 9 (2).
   05    MOIS        PIC 9 (2).
   05    ANNEE       PIC 9 (2).
```

où tous les champs de niveau 05 sont élémentaires.
Supposons que l'on doive définir un champ élémentaire, par exemple DEPARTEMENT, n'appartenant à aucun autre champ et n'ayant pas été à son tour subdivisé. Deux types de définition peuvent être donnés :

```
01    DEPARTEMENT PIC X(2).
77    DEPARTEMENT PIC X(2).
```

Les deux descriptions sont équivalentes, la différence portant uniquement sur le numéro de niveau.
La première description utilise le niveau 01 et peut être écrite à n'importe quel endroit de la WORKING-STORAGE SECTION, alors que la seconde (niveau 77) devra être écrite en début de section, comme le montrent les exemples suivants :

- **premier cas**

```
. . . . . .
. . . . . .

WORKING-STORAGE SECTION.
. . . . . .
. . . . . .

01 CHAMP-1
   05 SOUS-CHAMP-1    PIC X(30).
   05 SOUS-CHAMP-2
    10 SOUS-CHAMP-21  PIC 9(3).
    10 SOUS-CHAMP-22  PIC 9(3).
. . . . . .
. . . . . .

01 DEPARTEMENT        PIC X(2).
```

- **deuxième cas** (niveau 77)

```
. . . . . .
. . . . . .

WORKING STORAGE SECTION.
77 DEPARTEMENT PIC X(2)
. . . . . .
. . . . . .
```

A noter que le numéro 77 ne peut être utilisé pour la définition d'un enregistrement dans la FILE SECTION.

Le niveau 88. Puisque le domaine DEPARTEMENT est un alphanumérique à deux caractères [PIC X(2)], il peut contenir le sigle du département lui-même. Si, au cours du programme, on doit vérifier que le département est celui de Lyon, on devrait écrire (à partir de la colonne 12)

```
IF DEPARTEMENT IS EQUAL TO' '69'
```

La même vérification peut se faire d'une façon plus parlante ; si le champ département

```
01   DEPARTEMENT      PIC X (2).
     88 LYON          VALUE '69'.
     88 MARSEILLE     VALUE '13'
     88 PARIS         VALUE '75'
     88 . . . . . .
     88 . . . . . .
```

Dans ce cas, la vérification peut se faire de la manière suivante :

```
IF LYON
. . . . . .
```

Le niveau 88 permet donc d'associer, à un champ élémentaire différent des niveaux 01 ou 77, un nom conditionnel.
Un autre exemple est illustré par le listing ci-dessous. Ainsi, pour savoir si l'employé en question est un homme ou une femme, on peut écrire IF HOMME... au lieu de IF SEXE IS EQUAL TO 'M'.

Le niveau 66 et la clause RENAMES. C'est le dernier numéro de niveau spécial utilisé par le langage Cobol. D'un emploi peu fréquent, il est exclusivement lié à la clause RENAMES qui permet de référencer soit une donnée élémentaire, soit un groupe de données ayant une dénomination autre que celle employée dans la description. Examinons par exemple l'enregistrement EMPLOYE défini par :

```
01 EMPLOYE
   05 MATRICULE              PIC 9(3).
   05 PRENOM ET NOM
      10 NOM                 PIC X(20).
      10 PRENOM              PIC X(10).
   05 DATE DE NAISSANCE
      10 JOUR                PIC 9(2).
      10 MOIS                PIC 9(2).
      10 ANNEE               PIC 9(2).
   05 SEXE                   PIC A.
```

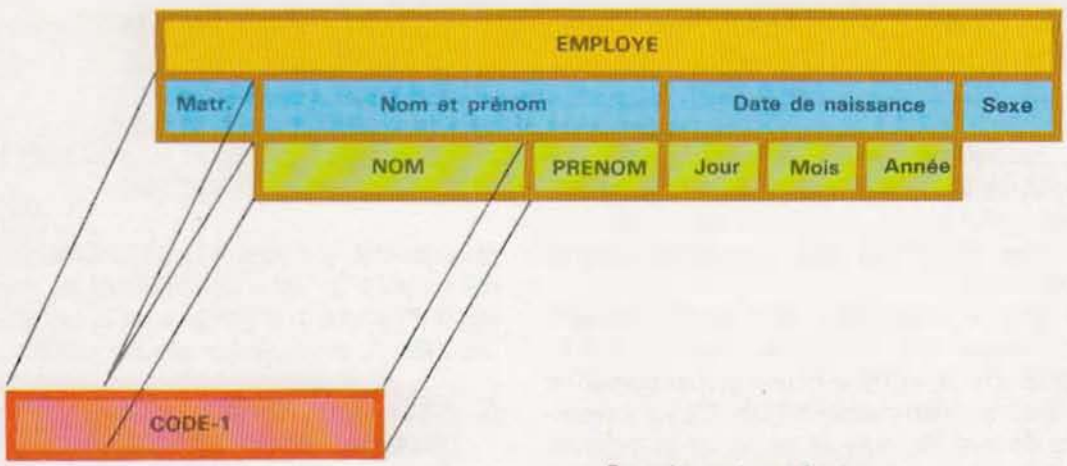
Supposons qu'il faille maintenant grouper en un seul champ (dénommé CODE-1) à la fois le numéro de matricule (MATRICULE) et le nom de l'employé (NOM) sans toutefois faire de transfert. L'opération est réalisée au moyen de la clause RENAMES, en écrivant :

```
01 EMPLOYE.
   05 MATRICULE              PIC 9(3).
   05 PRENOM ET NOM.
      10 NOM                 PIC X(20).
      10 PRENOM              PIC X(10).
   05 DATE DE NAISSANCE.
      10 JOUR                PIC 9(2).
      10 MOIS                PIC 9(2).
      10 ANNEE               PIC 9 (2).
   05 SEXE                   PIC A.
66 CODE-1 RENAMES MATRICULE THRU
NOM.
```

EMPLOI DU NIVEAU 88

```
01  EMPLOYE.
    05  MATRICULE                    PIC 9(3).
    05  PRENOM ET NOM                PIC X(30).
    05  DATE DE NAISSANCE.
        10  JOUR                     PIC 9(2).
        10  MOIS                     PIC 9(2).
        10  ANNEE                    PIC 9(2).
    05  SEXE                         PIC A.
        88  HOMME     VALUE  'M'.
        88  FEMME     VALUE  'F'.
    05  EMPLOI.
        10  SECTEUR                  PIC 9(3).
        10  NIVEAU                   PIC 9(3).
    05  FILLER                       PIC X(54).
```

SCHEMA SIMPLIFIE DES ZONES MEMOIRE SOUMISES A LA CLAUSE RENAMES



Encombrement en mémoire :
Code-1 : 33 caractères
Employé : 40 caractères

De cette manière, on informe le compilateur qui réserve une zone mémoire désignée par CODE-1 en symbole alphanumérique d'une longueur totale de 33 caractères. Elle peut donc contenir simultanément les champs de l'enregistrement allant de MATRICULE à (THRU) NOM compris (voir le schéma ci-dessus).

La clause RENAMES peut être également employée tout simplement pour attribuer à une donnée un nom différent de celui que contient sa description. Par exemple :

66 NAISSANCE RENAMES DATE-DE-NAISSANCE.

La signification de nom-donnée-1, nom-donnée-2 et nom-donnée-3 a été décrite dans les exemples précédents. Le tableau ci-dessous donne le format général de la clause RENAMES.

La clause REDEFINES. En Cobol, on a souvent besoin de définir un groupe de données contiguës autrement que dans la définition d'origine.

La clause RENAMES est certainement la méthode la moins employée pour résoudre ce problème : son inconvénient vient de ce qu'elle occupe une nouvelle zone mémoire pour l'allocation de la donnée portant un nouveau nom. En revanche, la clause REDEFINES permet de redéfinir l'espace réservé pour une donnée sans occuper d'espace supplémentaire en mémoire.

Soulignons que l'emploi de la clause REDEFINES n'est possible que lorsqu'on est absolument certain que l'une et l'autre définition seront utilisées à des instants différents. En d'autres termes, si le programme utilise à un moment donné la zone DATA-TR (traitement des données), puis la zone ART (article), on peut éviter de définir séparément les deux zones dans la WORKING-STORAGE SECTION (donc en occupant deux zones mémoire) en recourant à l'écriture suivante :

FORMAT DE LA CLAUSE RENAMES

66 nom-données-1 RENAMES nom-donnée-2 [THRU nom-donnée-3]

```
01 DATA-TR
   05 JOUR          PIC 9(2)
   05 MOIS          PIC 9(2)
   05 ANNEE         PIC 9(2)
01 ART REDEFINES DATA-TR.
   05 SERIE         PIC X.
   05 NUMERO        PIC 9(3).
   05 FILLER        PIC X(2).
```

La clause REDEFINES exigeant que la zone redéfinie et celle à redéfinir aient la même longueur, il a fallu, dans l'exemple, insérer un champs FILLER de deux caractères dans la zone ART.
Un même champ peut être défini plusieurs fois selon les modalités décrites précédemment : il faut également garder présent à l'esprit que des clauses REDEFINES successives doivent toujours se rapporter au premier champ de la chaîne. Supposons que l'on veuille utiliser de nouveau le champ DATA-TR et le redéfinir en tant que champ NUM (numéro protocole), il est alors correct d'écrire ce qui suit :

```
01 DATA-TR.
   05 JOUR          PIC 9(2).
   05 MOIS          PIC 9(2).
   05 ANNEE         PIC 9(2).
01 ART REDEFINES DATA-TR.
   05 SERIE         PIC X.
   05 NUMERO        PIC 9(3).
   05 FILLER        PIC X(2).
01 NUM REDEFINES DATA-TR PIC 9(6).
```

La clause REDEFINES ne peut s'appliquer à des phrases de description d'enregistrement dans la FILE SECTION ni à des champs ayant un numéro de niveau 66 et 88. La clause doit suivre immédiatement la description du champ auquel elle fait référence et doit avoir le même numéro de niveau. Comme nous le verrons plus loin, le compilateur peut donnner au contenu d'un champ la valeur établie par le programmeur au moyen de la clause VALUE. Soit par exemple le champ ETAB (établissement) à définir de la façon suivante :

```
01 ETAB.
   05 SIEGE         PIC X(2) VALUE 'LY'
   05 NUM-ETAB      PIC 9(4).
```

Dans ces conditions, il n'est pas permis d'utiliser ETAB pour redéfinir un autre champ. C'est donc une erreur que d'écrire :

```
01 DATA-TR.
   05 JOUR          PIC 9(2).
   05 MOIS          PIC 9(2).
   05 ANNEE         PIC 9(2).
01 ETAB REDEFINES DATA-TR.
   05 SIEGE         PIC X(2) VALUE 'LY'
   05 NUM ETAB      PIC 9(4).
```

En revanche, la clause VALUE, utilisée pour la définition d'un nom conditionnel au niveau 88, n'empêche pas l'emploi de la clause REDEFINES. L'exemple suivant est correct.

```
01 DATA-TR.
   05 JOUR          PIC 9(2).
   05 MOIS          PIC 9(2).
   05 ANNEE         PIC 9(2).
01 ETAB REDEFINES DATA-TR.
   05 SIEGE         PIC X(2).
      88 LYON       VALUE 'LY'.
      88 PARIS      VALUE 'PA'.
   05 NUM-ETAB      PIC 9(4).
```

Les notations

```
   05 SIEGE         PIC X(2) VALUE 'LY'
```

et

```
   05 SIEGE         PIC X(2).
      88 LYON       VALUE 'LY'.
```

ne sont pas équivalentes. La première impose au compilateur de places la valeur LY dans la zone SIEGE ; la seconde laisse au programmeur le soin de gérer lui-même le contenu de SIEGE, en demandant au compilateur d'associer le nom LYON au groupe de caractères LY de façon à pouvoir en vérifier l'existence dans le champ SIEGE. Quant à l'emploi du niveau 88, il a été décrit plus haut.

La clause PICTURE. Cette clause (PIC) a déjà été introduite et employée dans les exemples précédents. Elle fournit au compilateur les informations relatives au nombre et au type de caractères contenus dans une donnée élémentaire. En outre, elle permet



Recherche assistée par ordinateur de formules chimiques

d'indiquer au compilateur toute une série d'opérations à réaliser sur les caractères du champ. Le format général de la clause PICTURE (voir ci-dessous) doit être précisé pour tous les champs élémentaires — qu'ils soient du niveau 77 ou compris entre 01 et 49.

FORMAT DE LA CLAUSE PICTURE

[{ PICTURE / PIC } IS symboles.]

La clause n'est pas nécessaire pour les champs composés car le compilateur sait calculer leurs dimensions (en additionnant les dimensions des champs et en supposant, par définition, qu'un champ composé est alphanumérique, indépendamment des classes des données décrites).
Ainsi défini globalement, le champ :

```
01 DATE DE NAISSANCE.
   05 JOUR          PIC 9(2).
   05 MOIS          PIC 9(2).
   05 ANNEE         PIC 9(2).
```

est interprété comme un domaine alphanumérique à 6 caractères. Soulignons que la clause PICTURE revêt une importance fondamentale pour la fiabilité même des résultats d'un traitement. En effet, un programme, correct du point de vue de la forme et de la syntaxe, peut donner des résultats peu fiables ou se terminer de manière anormale, uniquement parce qu'il contient une définition erronée d'un champ. Cette précision a pour but d'attirer l'attention du lecteur sur l'importance de cette clause dont les implications seront expliquées ultérieurement.
Avant de poursuivre, signalons que, par la suite, nous adopterons toujours le format abrégé :

PIC symboles.

Les symboles employés dans la clause PICTURE sont :

A X 9 P X * $ B 0 + — . , / S V CR DB

S V CR DB peuvent apparaître une seule fois dans le cadre d'une même PICTURE, alors que la virgule est répétitive à condition de ne pas suivre immédiatement une autre virgule. La description d'une PICTURE contient au maximum 30 symboles ; on peut donc écrire

```
01 NOM          PIC X(35).
```

la forme suivante n'étant pas permise :

```
01 NOM  PIC XXX...(35 symboles X)...XXX.
```

dans la mesure où les symboles qui suivent le mot PIC sont supérieurs à 35.
La syntaxe et la signification de la clause sont différentes selon que les champs sont numériques, alphabétiques ou alphanumériques.

Description des champs numériques. Un champ destiné à accueillir exclusivement des valeurs numériques est décrit dans la WORKING-STORAGE SECTION par un symbole de PICTURE représentant une combinaison des caractères 9 V P S.
Parmi les symboles qui décrivent le champ, 9 repère un chiffre (en base 10). Le contenu des champs numériques est à l'image du système décimal, même si sa représentation interne est réalisée selon le code défini par la clause USAGE qui sera décrite plus loin. Par exemple, la ligne

```
01 CHAMP-1          PIC 9(10).
```

décrit un champ qui peut recevoir au maximum un nombre entier de dix chiffres.
Par définition, un champ numérique ne peut contenir d'autres caractères que des chiffres. D'après cette description, il ne peut donc même pas contenir le point décimal (ou virgule en notation européenne). Pour pouvoir indiquer la position de ce séparateur, on se sert du symbole V, qui repère la position virtuelle. En d'autres termes, le symbole V n'occupe pas d'espace en mémoire mais établit une référence pour le compilateur de manière à permettre d'aligner correctement les données traitées dans ce champ. Le domaine numérique décrit par la ligne

```
01 CHAMP-2          PIC 9(3)V9(3).
```

occupe 6 caractères en mémoire et reçoit des nombres réels avec, au maximum, trois chiffres entiers et trois décimaux. Si on attribue au champ -2 la valeur 123456, ce nombre sera enregistré et traité sous la forme 123.456.

Le caractère P permet d'étendre le nombre contenu dans le champ avec autant de zéros (0) qu'il y a de symboles P présents parmi les symboles de la PICTURE. Supposons que la donnée à transférer soit 8471, si la définition du champ est

```
01 CHAMP-3          PIC 9(4)P(5)V.
```

Le nombre sera enregistré et traité sous la forme

```
874100000
```

alors que si la définition est

```
01 CHAMP-3          PIC VP(5)9(4).
```

le nombre sera traité sous la forme

```
.000008741
```

Dans le cas où il est utilisé, le caractère S est le premier de la combinaison de symboles de la PICTURE. Il traite de la valeur numérique contenue dans le champ avec son signe algébrique. Si, parmi les symboles de la PICTURE, on omet le caractère S, le contenu du champ est considéré comme toujours positif. En revanche, le champ décrit par :

```
01 CHAMP-4          PIC S9(8).
```

reçoit aussi bien la valeur + 425 que — 425. En adoptant le symbole S, on a en outre la possibilité d'indiquer le signe algébrique de l'impression.

Il convient de souligner l'importance que prend la définition correcte d'un champ numérique. En effet, bien qu'un champ alphanumérique (PIC X) puisse aussi recevoir des chiffres, ces derniers seront traités comme n'importe quel autre caractère non numérique.
Ce n'est que si le champ est défini comme numérique (PIC 9) qu'on pourra traiter réelle-

ment son contenu en tant que nombre. L'emploi des champs numériques permet donc non seulement d'effectuer des opérations arithmétiques sur leur contenu, mais encore de les imprimer correctement par l'insertion physique du point décimal ou du signe algébrique.
De ce qui précède, on déduit que la définition d'un champ porte sur la taille de la zone qu'il occupera en mémoire ainsi que sur sa codification en conformité avec le type de caractères qu'il recevra.

Description des champs alphabétiques. Par cette expression, on entend généralement un champ apte à recevoir tous les caractères de l'alphabet anglais et eux seuls, en plus de l'espace (blank), que nous appellerons b par la suite. Pour indiquer au compilateur les allocations mémoire de type alphabétique, on introduit dans la description autant de A qu'il y a de caractères composant le champ (on peut toutefois utiliser la forme abrégée déjà décrite). Supposons qu'il faille définir le champ alphabétique de 30 caractères, NOM et PRENOM, sa description sera

```
01 NOM ET PRENOM  PIC A(30).
```

Ce champ recevra par exemple la donnée :

```
DUPONT PIERRE JEAN
```

mais non la donnée :

```
PROF. DUPONT PIERRE JEAN
```

car dans cette chaîne de caractères se trouve un point. Examinons maintenant le cas d'un programme qui lise un champ de 8 caractères alphabétiques dont les deux premiers identifient le modèle, les trois suivants la série et les trois derniers le mois de production d'un article donné. Si, pour des raisons d'impression ou pour tout autre motif, les groupes de caractères cités doivent être espacés par un ou plusieurs blancs, on peut avoir recours au symbole B de la clause PICTURE :

```
01 CODE       PIC A(2)BBA(3)BA(3).
```

Si le code à lire est AZXYWSET, le contenu de CODE sera alors

```
AZ  XYW SET
```

Le programmeur peut donc obtenir, dans certaines parties du champ, l'insertion d'autant de blancs que l'on a décidé d'insérer de B dans la clause PICTURE.

Description des champs alphanumériques. Ils sont en mesure de recevoir une chaîne de caractères composée de lettres, de chiffres, de caractères spéciaux et de blancs dans n'importe quel ordre. Le symbole qui signale cette classe de données au compilateur est la lettre X. Le compilateur considère toutefois comme alphanumérique un champ dans lequel se trouvent simultanément les symboles de définition d'au moins deux classes de caractères. Tous les champs suivants sont donc alphanumériques :

```
01 SERIE-ARTICLE     PIC AA9(2).
01 PRODUCTEUR        PIC X(30).
01 DESCRIPTION       PIC AAAXXXX.
01 DESCRIPTION       PIC A(3)X(4).
```

Les champs alphanumériques prévoient une série de symboles, dits éditing, qui, lorsqu'ils sont judicieusement insérés dans la clause PICTURE, permettent certaines opérations particulières sur les caractères. Considérons par exemple le cas où le champ

```
01 SERIE-ARTICLE     PIC AA9(2)
```

est à imprimer sous la forme Serie/Numéro. Dans ce cas, il suffit de définir le champ d'impression ainsi :

```
05 SERIE-IMPRESSION  PIC A(2)/9(2).
```

Si donc la donnée contenue dans SERIE-ARTICLE était SR12, une fois placée dans SERIE-IMPRESSION, elle prendrait la forme :

```
SR/12
```

Le symbole 0, utilisé dans une PICTURE de type alphanumérique, permet d'introduire (de la même façon que pour les symboles B et /), un ou plusieurs caractères 0 dans la position indiquée par la clause. D'autres symboles sont utilisés exclusivement en prévision de l'impression des champs numériques :

* A l'impression, remplace un éventuel zéro initial se trouvant dans la position occupée par le symbole dans la PICTURE.

Z A l'impression, remplace autant de zéros initiaux qu'il y a de Z définis par autant de blanks (espaces).
Insère une virgule à la position définie dans la PICTURE, à condition que le caractère qui la précède n'ait pas été supprimé.

$ A un double usage :
a) suppression des zéros non significatifs et remplacement du dernier zéro supprimé par un symbole monétaire.
b) insertion du symbole monétaire à la position définie.
Dans le premier cas a), il y aura nécessairement dans la PICTURE autant de symboles $ qu'il y a de zéros que l'on désire supprimer. Dans le cas b), il suffit de définir un seul caractère $.

+ — Employés de la même façon que le symbole $, comme caractère d'insertion ou de suppression.

CR DB Le suffixe CR est l'abréviation de CREDIT et DB de DEBIT.

Symbole	Donnée à imprimer	Picture du domaine à imprimer	Valeur imprimée
*	0000	* * * * *	* * * * * *
	0000	* * * *	* * * *
	1234	* * * *	1234
	0012	* * * *	* * 12
	0012	* * * 9	* * 12
	0012	* 9(3)	* 012
Z	0000	ZZZZ	
	1230	ZZZZ	1230
	0012	ZZZZ	12
	0012	ZZZ9	12
	0012	Z9(3)	012
	12345	99,999	12,345
	00123	ZZ,ZZZ	123
	00123	* * , * * *	* * * 123
	12345	9(3).9(2)	123.45
$	12345	$$$$.99	$123.45
	1234	9(4)	$1234
	0000	$Z(3)9	$ 0
	0000	$Z(4)	
+ et —	15	— 9(2)	15
	— 15	— 9(2)	— 15
	— 15	9(2) —	15 —
	00	— 9(2)	00
	15	+ 9(2)	+ 15
	— 15	+ 9(2)	— 15
	15	9(2) +	15 +
	123	— — 9(2)	123
	— 012	— — 9(2)	— 12
	000	— — — —	
	012	+ + 9(2)	+ 12
	— 001	+ + + 9	— 1
	000	+ + + +	
	123	+ + 99	+ 123
CR et DB	78912	$$$$.99CR	$789.12
	— 00478	$$$$.9(2)CR	$4.78CR
	— 00478	$$$$.9(2)DB	$4.78DB

Ils n'apparaissent qu'à la fin d'une PICTURE et permettent d'imprimer, à la droite d'un nombre, les suffixes DB (si le nombre est négatif) et CR. Dans le cas contraire, deux blancs sont imprimés.

Pour clarifier les concepts exposés dans le tableau de la page précédente, les symboles ont été illustrés par un exemple. Pour chaque cas, on peut ainsi lire :

- la valeur numérique qu'il faut transférer en impression,
- la PICTURE du champ d'impression destiné à la recevoir,
- le résultat correspondant sur papier.

Pour établir une synthèse des différents champs sur lesquels opèrent les instructions qui seront étudiées par la suite, nous avons reporté dans le tableau ci-dessous tous les symboles que l'on peut utiliser selon les trois classes de données traitées par le Cobol.

PROCEDURE DIVISION

Jusqu'à présent, nous avons exposé les fonctions et les règles de syntaxe des trois premières divisions d'un programme Cobol. Bien qu'elle constitue la quatrième et dernière division du Cobol, celle-ci est la seule où le programmeur traduit réellement l'ensemble des opérations à réaliser sur les données. Il ne faut cependant pas l'identifier au programme lui-même. Le programmeur doit analyser le problème et adopter une méthode de résolution qui tienne compte des quatre divisions.
Examinons maintenant la division algorithmes (PROCEDURE DIVISION) en cherchant tout d'abord à mettre en évidence les problèmes liés à l'organisation du programme.
Les instructions qui constituent cette division appartiennent à trois niveaux hiérarchiques décroissants.
Le niveau le plus bas du groupe est constitué par la **phrase**, c'est-à-dire par une série d'instructions s'exécutant en séquence comme s'il s'agissait d'une instruction unique. On trouvera un regroupement de ce type dans les parties du programme où il faut exécuter des branchements différents suivant qu'une condition est vérifiée ou non. Cette phase de décision est réalisée en Cobol (comme dans la plupart des autres langages) par l'instruction IF, dont nous verrons l'usage en détail.

SYMBOLES UTILISES POUR LA DESCRIPTION DES DONNEES (PICTURES)

Classe de la donnée	Définition de classe			Symboles d'opération			Symboles de préparation à l'impression (EDITING)											
	X	A	9	S	V	P	*	Z	0	B	,	.	$	+	—	DB	CR	/
ALPHANUMERIQUE	OUI	OUI*	OUI*	NON	NON	NON	NON	NON	OUI	OUI	NON	NON	NON	NON	NON	NON	NON	OUI
ALPHABETIQUE	NON	OUI	NON	NON	NON	NON	NON	NON	NON	OUI	NON	NON	NON	NON	NON	NON	NON	NON
NUMERIQUE	NON	NON	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI	OUI

Caractère admis

Caractère non autorisé

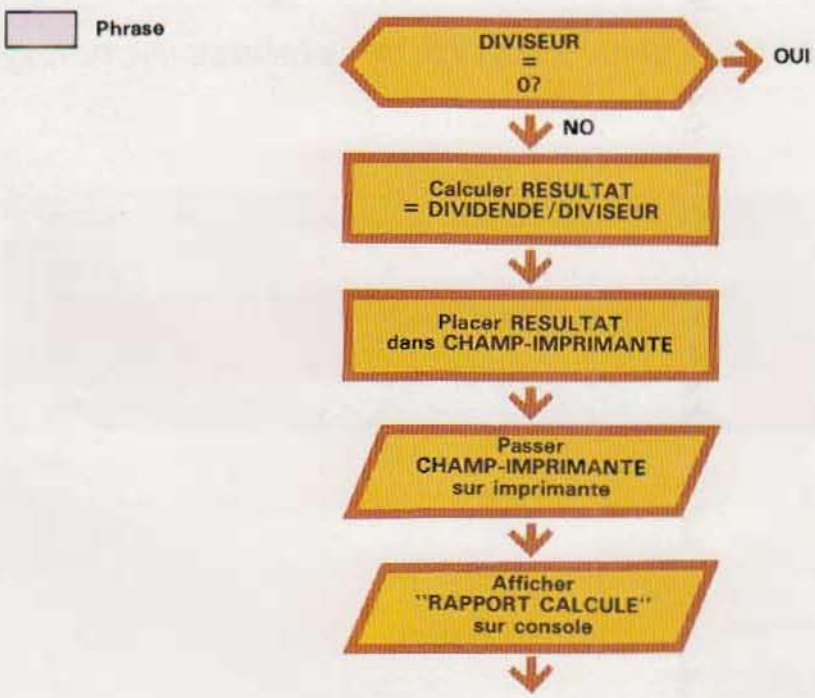
* Pour définir une donnée alphanumérique, il faut au moins un autre symbole de définition de classe (A, X, 9)

Pour approfondir le sens de la « phrase », prenons le cas, représenté ci-dessous, du calcul du rapport entre les variables DIVIDENDE et DIVISEUR.
L'opération n'est naturellement possible que si DIVISEUR est différent de 0.
On désire donc

1 / calculer le résultat,
2 / placer le résultat dans un champ destiné à l'impression,
3 / transférer ce champ à l'imprimante,
4 / éditer enfin le message « RAPPORT CALCULE » sur la console système.

Les quatre opérations correspondent à quatre instructions Cobol ; elles constituent une phrase dans la mesure où elles doivent être exécutées selon une séquence logique répondant à la réalisation d'un événement. Pour familiariser le lecteur avec les instructions du langage, nous avons reporté, dans le schéma ci-dessous, l'écriture correcte de la partie de PROCEDURE DIVISION correspondant à l'ensemble. Remarquons qu'une phrase se termine toujours par un point. Un programme Cobol, comme tout autre programme, est constitué d'une série d'instructions disposées en séquence.

```
IF    DIVISEUR NOT EQUAL  TO 0
      COMPUTE RESULTAT = DIVIDENDE / DIVISEUR
      MOVE RESULTAT TO CHAMP-IMPRIMANTE
      DISPLAY CHAMP-IMPRIMANTE UPON PRINTER
      DISPLAY  '** RAPPORT CALCULE = ' RESULTAT
               UPON CONSOLE.
```



Le diagnostic assisté par ordinateur (2)

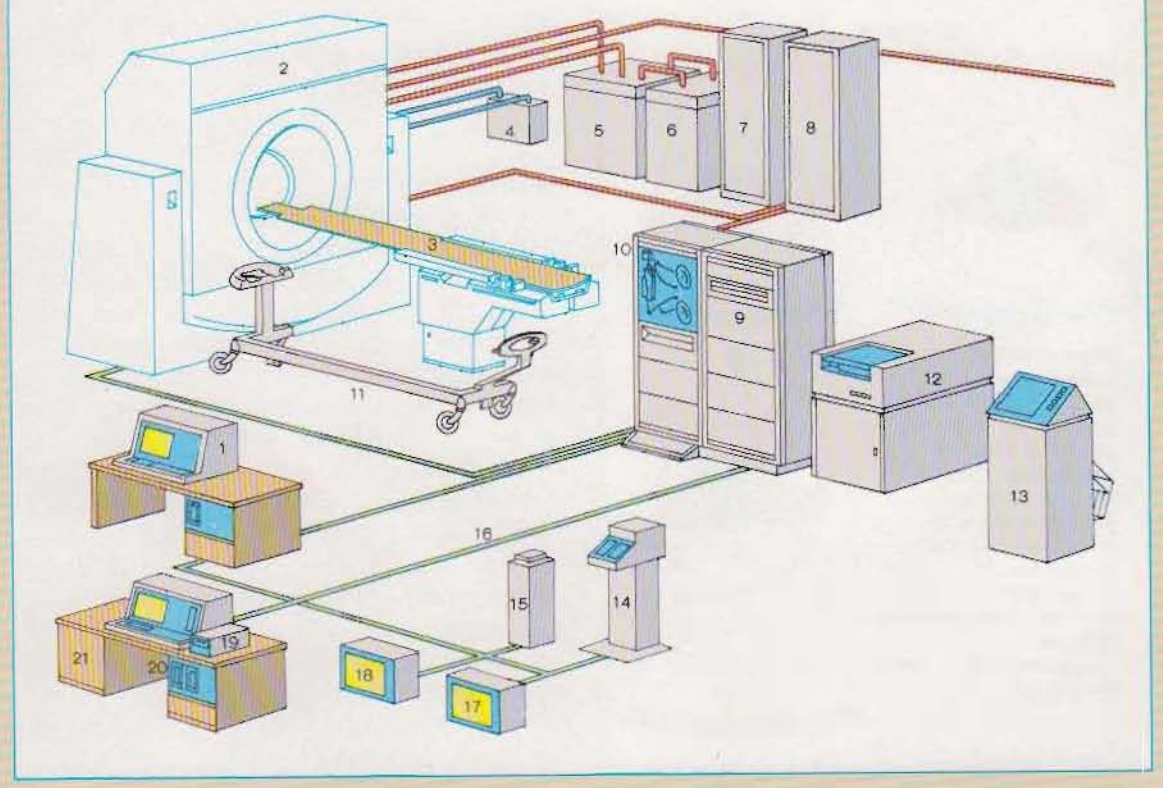
C'est vers 1970 que l'informatique a commencé à s'introduire dans le domaine biomédical et la tomographie est peut-être la technique qui a le plus bénéficié, jusqu'ici, de ses progrès.

L'introduction de la ***Tomographie Assistée par Ordinateur (TAO)*** *dans le monde du radiodiagnostic a été vécue comme un événement révolutionnaire. Pour la première fois, on a pu examiner — dans les moindres détails et avec des images d'une haute définition — n'importe quelle coupe axiale du corps d'un patient.*

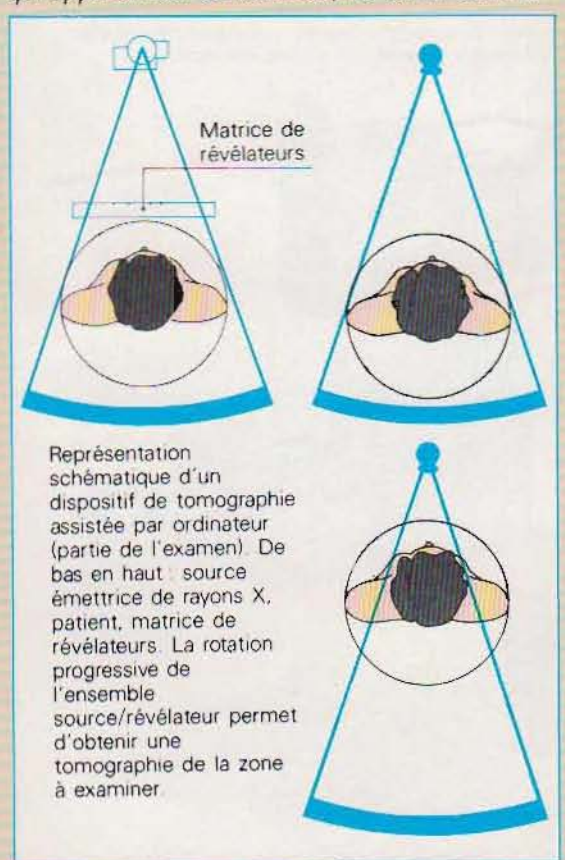
La TAO repose sur la reconstruction d'une ***image globale*** *issue des différentes images radiographiques réalisées sous plusieurs incidences. Une source émettrice, permettant une haute résolution, est positionnée le long d'une circonférence et orientée vers le centre. Au point de la circonférence diamétralement opposé se trouve, sur un arc d'une quarantaine de degrés (correspondant au champ du faisceau de rayons X émis), une matrice de détection. Lorsqu'on place le patient entre la source et le détecteur, l'amplitude du signal en sortie est déterminée par celui-ci. La sortie sous forme numérique, qui est fournie par la matrice de détection à la suite d'une émission X, est sauvegardée. Puis on réalise une rotation d'un certain angle pour l'ensemble source/détecteur et on procède à la mémorisation d'une autre image. Quand le système a réalisé une rotation de 360° en traitant de manière numérique les images mémorisées, on obtient enfin une coupe tomographique de la zone à explorer.*

Un appareil tomographique typique est le Tomoscan que l'on trouve schématisé en détail dans la page précédente. La source émettrice utilise des tensions de 100-120 kV et émet des impulsions d'une durée de 1-2 ms.

Schéma d'ensemble des divers composants d'un système *Tomoscan* (de 1 à 9 : système base) : 1/ console et moniteur opérateur ; 2/ scanner avec tube émetteur de rayons X et matrice de détection ; 3/ support pour le patient ; 4/ unité de refroidissement à eau ; 5/ armoire du tétrode ; 6/ transformateur à haute tension ; 7/ tableau de puissance ; 8/ tableau central ; 9/ ordinateur avec microprocesseur destiné à la reconstruction de l'image ; (de 10 à 21 : en option) ; 10/ unité de bande ; 11/ chariot ; 12/ disque à grande capacité ; 13/ table traçante imprimante électrostatique ; 14/ caméra multi-objectifs ; 15/ appareil photo Polaroid ; 16/ ligne de transmission des données ; 17/ moniteur noir et blanc ; 18/ moniteur couleur ; 19/ lecteur de cassettes magnétiques ; 20/ deuxième lecteur pour disquette (disponible sur la console de l'opérateur et sur la console à distance) ; 21/ console à distance.



Selon la résolution de l'image, le nombre d'impulsions émises pour une rotation complète variera de 350 à 1200. La matrice de détection (constituée d'environ 600 capteurs, assemblés dans une chambre sous vide contenant du xénon sous 20 atmosphères) couvre un arc de 43,5°. Pour produire une image à haute définition, il faut environ 21 secondes (3 à 10 sec. sont nécessaires pour compléter la rotation).
Une autre façon d'opérer permet de visualiser l'image initiale après traitement des données correspondant à la demi-rotation d'une seconde. ***La matrice de recomposition de l'image*** *est de 256 X 256 pixels (éléments graphiques), l'épaisseur des « tranches » analysées variant entre 1,5 et 12 mm.*
Naturellement, le Tomoscan informatisé peut aussi servir d'appareil radiographique classique pour des projections frontales. Cette propriété simplifie la gestion de l'ensemble du système. Cet appareil permet en effet de ***mouvoir automatiquement le sujet*** *en utilisant une image radiographique frontale initiale d'incidence sous le contrôle du système. Cette incidence, qui apparaît sur le moniteur, va servir de référence à la sélection du niveau exact de la coupe. Le Tomoscan réalise automatiquement, en 6 secondes, un scannogramme frontal sur une zone de 36 cm autour du point d'intérêt. Pendant cette opération, l'image radiographique est révélée par bandes d'une largeur de 1,5 mm. Ces bandes sont ensuite transmises au moniteur qui affiche l'image frontale. Grâce au stylo optique, l'opérateur sélectionne une coupe au vu de toutes ces informations.*
*En déplaçant de manière radiale l'équipement mobile par rapport au centre de rotation, on utilise le Tomoscan au maximum de ses capacités de révélation. Ce procédé permet d'****adapter la géométrie du système aux dimensions de la coupe désirée****, qu'il s'agisse du thorax d'un adulte ou du crâne d'un enfant, en utilisant toujours complètement la matrice de détection. Quand le tube émetteur s'approche du patient, le courant d'alimentation se réduit. L'adaptation automatique sous le contrôle du détecteur de référence permet de réaliser des tomographies très détaillées avec un minimum de pénétrance RX.*
C'est le logiciel de gestion de l'ensemble de l'appareil qui autorise une telle gamme de prestations. Le Tomoscan est contrôlé par un microprocesseur de 16 bits PC 857, doté d'une mémoire RAM de 128 Ko. Deux unités de disque de 5,6 Mo chacune sont reliées à l'ordinateur, elles peuvent stocker 30 images ; une disquette de 0,25 Mo supporte, elle, l'équivalent de 10 images. Les gros fichiers sont sauvegardés sur bande magnétique (300 images) tous les 730 mètres environ.
L'opérateur dialogue avec l'ordinateur par le ***clavier*** *et le* ***stylo optique****. Avec ce dernier, il peut sélectionner une image parmi un large éventail de détails. En le pointant sur une partie de l'écran, l'opérateur obtient un agrandissement du double ou du quadruple. Il peut aussi appeler simultanément 4, 6 ou 9 images qui serviront à établir des comparaisons. L'effet zoom fournit des coupes en plan oblique à partir d'une succession de tomographies axiales. La qualité de l'image peut être améliorée en agissant sur le contraste du moniteur.*
Les images (initiales et calculées) sont stockées en mémoire de masse. Elles peuvent être rappelées instantanément (en temps réel) par des consoles, diffusées en circuit vidéo ou bien reproduites sur papier.



Représentation schématique d'un dispositif de tomographie assistée par ordinateur (partie de l'examen). De bas en haut : source émettrice de rayons X, patient, matrice de révélateurs. La rotation progressive de l'ensemble source/révélateur permet d'obtenir une tomographie de la zone à examiner.







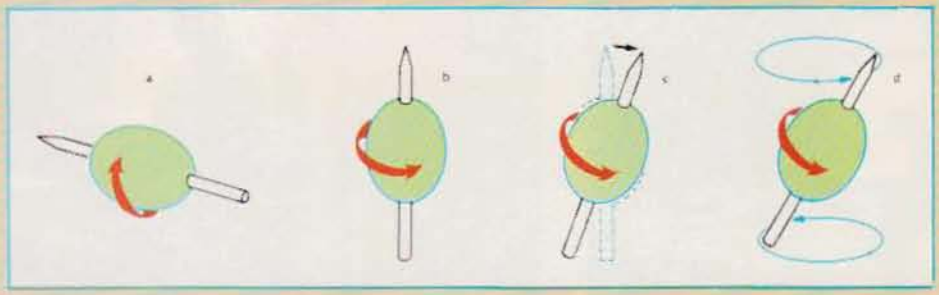
Tomographie axiale assistée par ordinateur (TAO). Photo du haut à gauche : vue d'ensemble du Tomoscan. Ci-dessus : positionnement automatique du sujet à examiner. Ci-contre : examen à distance de la coupe sériée sur une console.

L'intérêt de la TAO (Tomographie Assistée par Ordinateur) ne se limite pas à la visualisation des coupes sériées ; elle permet aussi la conduite des traitements radiothérapiques par tomodensitométrie (mesure de l'absorption densitométrique des tissus). Enfin, la TAO réduit très sensiblement les temps d'exposition aux radiations ionisantes.

*La tomographie à RMN, Résonance Magnétique Nucléaire (en anglais NMR nuclear magnetic resonance) est une nouvelle méthode d'investigation biomédicale. Elle utilise l'**effet magnétique** de certains noyaux atomiques, dont l'**hydrogène**, lequel est présent dans tous les tissus par l'intermédiaire de l'eau (H_2O).*

*Chargé positivement, le noyau d'hydrogène crée, en tournant sur lui-même, un champ magnétique orienté perpendiculairement à son plan de révolution (voir le graphique page suivante). Dans un élément ou dans un tissu quelconque, les noyaux d'hydrogène développent des champs magnétiques orientés statistiquement dans toutes les directions (a). Soumis à l'influence d'un champ magnétique extérieur, tous les noyaux ont tendance à aligner leur propre axe de rotation sur les lignes de force du champ magnétique imposé (b). Les noyaux se comportent ainsi comme de **minuscules gyroscopes**. Lorsque le champ magnétique externe est homogène, tous les noyaux oscillent à la même fréquence. Par contre, une excitation magnétique externe provoquera un désalignement des noyaux qui tendront à se disposer selon un certain angle (c). A la fin de cette impulsion, les noyaux retrouvent leur position initiale, libérant ainsi de l'énergie sous forme de faibles signaux radio (d). L'intensité et la période du signal émis renseignent sur la densité de la répartition des atomes présents dans l'espace, ce qui constitue un élément de différenciation des tissus.*

*L'image résulte des **signaux radio émis par les tissus**. La recomposition de l'image se fait par des techniques identiques à celles de*



Principe de la résonance magnétique nucléaire (RMN).

la TAO. Elle est donc en mesure de réutiliser tous les programmes de fonctionnement développés jusque là. La tomographie RMN devrait également différencier, dans le détail, même les tissus mous.

Malgré le secret qui enveloppe, pour des raisons évidentes, les industries de pointe, on connait à peu près le principe de fonctionnement d'yn système tomographique RMN. Le groupe Philips met au point une nouvelle version de RMN appelée Gyroscan comportant :

— un aimant principal, de type résistif ou supraconducteur ;
— une batterie de générateurs à fréquence radio pour produire les excitations ;
— une batterie de révélateurs à fréquence radio destinés à capter les réponses ;
— un spectromètre universel servant à l'analyse des signaux de réponse ;
— un ordinateur et ses périphériques.

Le phénomène de la réponse magnétique nucléaire a été découvert il y a environ soixante-dix ans. Comme nombre de phénomènes de cet ordre, on s'est contenté de le noter, car on n'en voyait aucune application immédiate, de tels phénomènes ayant été plutôt perçus comme une gêne ou comme dépourvus d'intérê technique, plutôt que comme un moyen d'exploration.

C'est seulement vers 1977 que la RMN connut sa première application aux Etats-Unis, dans un domaine jusqu'ici fort délicat à explorer, celui de l'estimation de la **densité osseuse**. La synthèse de l'image nécessita, à l'époque, un temps de calcul de quatre heures et demie, pour un résultat encore incertain. Depuis, l'industrie électronique s'est intéressée au phénomène et l'a révolutionné pour ce qui concerne à la fois sa fiabilité et le temps de traitement.

Même s'ils présentent encore un caractère embryonnaire, les systèmes actuels connaissent, d'ores et déjà, une très large diffusion.

Les procédés d'exploration utilisant la RMN présentent un énorme avantage par rapport aux techniques d'exploration classiques : ils ne nécessitent l'emploi d'aucune radiosource, que ce soit celle du rayonnement X ou celle des isotopes radioactifs.

On connait, bien entendu, le caractères destructif des rayonnements, utilisés précisément dans ce but-là en radiothérapie (destruction des métastases, en particulier, par curiethérapie), mais, jusqu'il y a encore peu de temps, on n'accordait qu'une importance relative à l'impact de rayonnements, même très faibles, sur les organismes qui y étaient soumis.

On savait bien que les radiologues risquaient une maladie professionnelle du fait d'une protection insuffisante due très souvent à des explorations entreprises (dans l'intérêt du malade et surtout du blessé) dans des conditions où toutes les consignes de sécurité ne pouvaient pas être appliquées. Ce dont on se rendait moins compte, c'est que des rayonnements très pénétrants (donc à haute énergie), saturent l'organisme et que celui-ci ne doit pas y être exposé au-delà d'une certaine durée (en données cumulées pour toute une vie).

Un premier progrès a été celui de la **numérisation des écrans**, permettant le stockage de l'information radiologique sans qu'il soit nécessaire de conserver le très long temps d'exposition que nécessitait une radiographie (avec obtention d'un film) alors qu'une "scopie", ne pouvant se conserver sur un support fixe, ne servait donc qu'au diagnostic immédiat.

Le second progrès a été l'élimination de toute radiosource. Les progrès de l'exploration génétique (pour les besoins de l'ingénierie génétique, en particulier) ont mis en valeur l'importance de phénomènes fins, comme ceux observés en présence de radioactivité naturelle.

Une séquence d'instruction Cobol, d'un point de vue à la fois logique et structurel, se subdivise en paragraphes ou regroupements. Chaque paragraphe est caractérisé par un nom, situé dans la marge A (colonne 8) et finissant par un point. La subdivision en paragraphes permet d'entrer, à des points déterminés du programme, par l'intermédiaire d'instructions de branchement conditionnel. Par ailleurs, on exécute ainsi toutes les instructions d'un certain paragraphe simplement en associant son nom au verbe PERFORM.

Un nom de paragraphe est créé par le programmeur à l'aide d'une combinaison de 30 caractères (maximum) alphabétiques et numériques, plus le trait d'union avec le signe moins. Le compilateur considère comme terminé un paragraphe quand il rencontre le nom d'un autre paragraphe.

Les paragraphes peuvent, à leur tour, être regroupés en une structure d'un niveau supérieur : la section. Celle-ci est identifiée selon les mêmes règles que le paragraphe. Elle est obligatoirement suivie du mot SECTION et d'un point. Chaque section s'achève soit à la fin du programme, soit au début d'une autre section.

Voici quelques exemples de noms de paragraphes et de sections :

```
DEBUT.                      (paragraphe)
CALCUL-TVA SECTION.         (section)
A10-ADDITION.               (paragraphe)
340-LECTURE-FICHIER.        (paragraphe)
ECRITURE SECTION.           (section)
```

Un nom de paragraphe ou de section comporte, impérativement, au moins un caractère alphabétique.

La PROCEDURE DIVISION d'un programme générique en Cobol est ainsi structurée :

```
PROCEDURE DIVISION.
PREMIERE SECTION.           (section)
DEBUT.                      (paragraphe)
. . . . . .
. . . . . .                 INSTRUCTION
. . . . . .
. . . . . .
OUVERTURE-FICHIER.          (paragraphe)
. . . . . .
. . . . . .
. . . . . .
. . . . . .
DEUXIEME SECTION.           (section)
LECTURE-FICHIER.            (paragraphe)
. . . . . .
. . . . . .
. . . . . .
. . . . . .
CALCUL.                     (paragraphe)
. . . . . .
. . . . . .
. . . . . .
TROISIEME SECTION.          (section)
IMPRESSION.                 (paragraphe)
. . . . . .
. . . . . .
. . . . . .
FERMETURE-FICHIER.          (paragraphe)
. . . . . .
. . . . . .
. . . . . .
. . . . . .
FIN.                        (paragraphe)
STOP RUN.
```

Une section contient au moins un paragraphe

Avertissement

La disposition des mots réservés dans les divers champs est importante dans la structure du langage Cobol. Indépendamment du périphérique d'entrée utilisé, chaque programme peut être décrit sur cartes perforées. En examinant les listings, on vérifie que les mots sont correctement regroupés par colonne et donc que la perforation des instructions sur les cartes est correcte. Les exemples du texte ne sont là que pour illustrer l'ordre d'apparition de mots-instructions simples. A noter que les contraintes typographiques ne permettent pas toujours, comme c'est le cas des sorties sur imprimantes (listings), de respecter la syntaxe du Cobol.

dont le nom est suivi du nom d'un paragraphe.
Ainsi, il serait incorrect d'écrire :

```
PREMIERE SECTION.        (section)
MOVE 0 TO A.             (instruction)
```

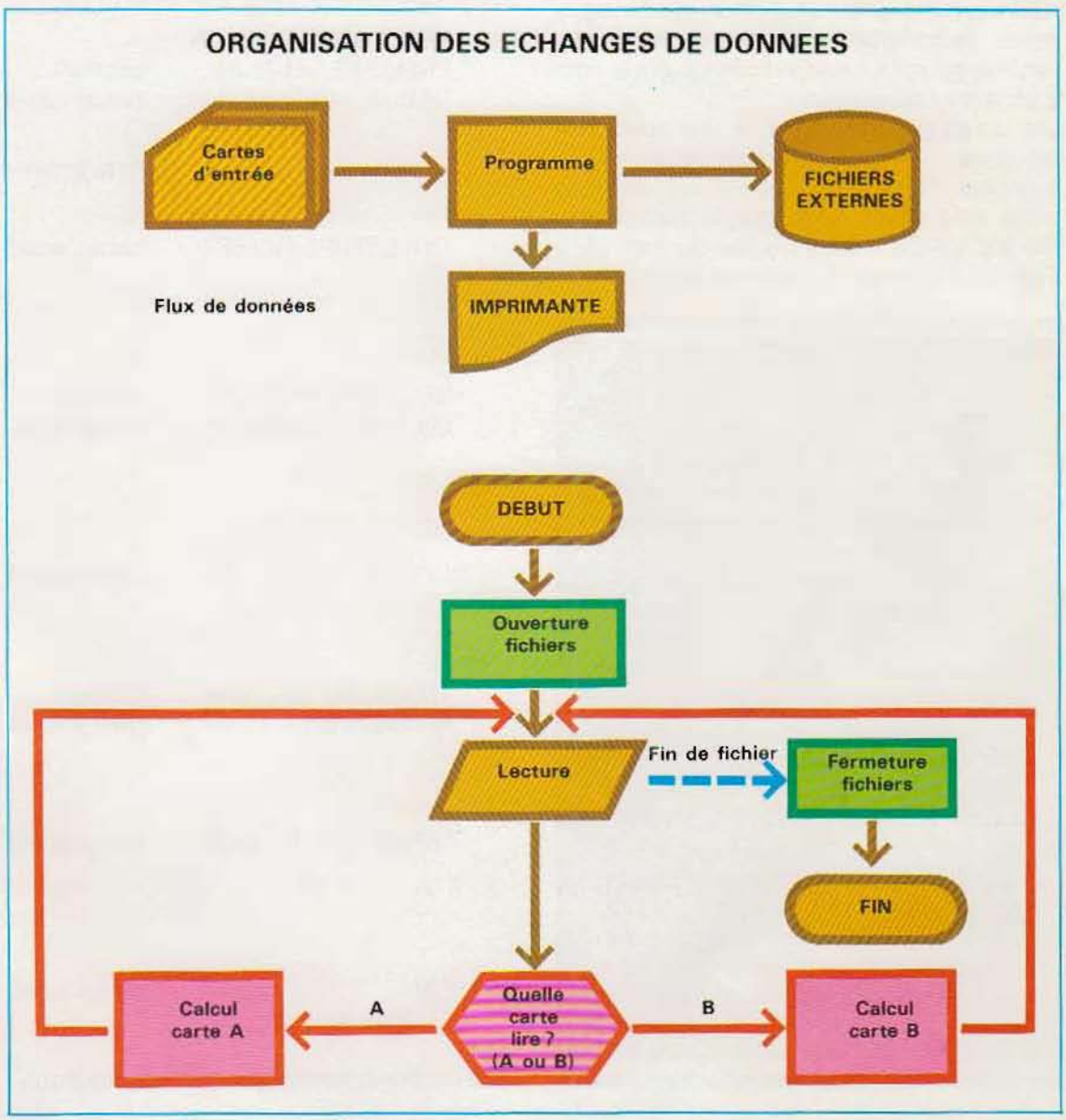
au lieu de :

```
PREMIERE SECTION.        (section)
DEBUT.                   (paragraphe)
MOVE 0 TO A.             (instruction)
```

De même, chaque paragraphe comporte au moins une instruction. Comme on peut le constater dans les exemples précédents, les noms de paragraphes et de sections sont très parlants. Attribuer des noms de cette manière permet, à l'auteur du programme ou à un autre programmeur, de comprendre intuitivement le sens de la fonction. Il ne s'agit pas là d'une règle du Cobol, mais cela contribue à faciliter la lecture et la maintenance des programmes.

Exemple de PROCEDURE DIVISION

Soit un programme qui lise dans dans un fichier deux types de cartes A et B décrites



STRUCTURE D'UNE PROCEDURE DIVISION
DESCRIPTION DES FICHIERS (CARTES D'ENTREE)

```
01  CARTE-B.
    05  FILLER.                    PIC X.
    05  PRENOM-NOM                 PIC X(30).
    05  SOLDE                      PIC 9(10).
    05  INTERET                    PIC 9(3)V9(3).
    05  FRAIS                      PIC 9(5).
    05  FILLER                     PIC X(5).

01  CARTE-A.
    05  FILLER                     PIC X.
    05  NOM                        PIC X(30).
    05  DATE-NAISSANCE.
        10  JOUR                   PIC 9(2).
        10  MOIS                   PIC 9(2).
        10  ANNEE                  PIC 9(2).
    05  DONNER                     PIC 9(10).
    05  AVOIR                      PIC 9(10).
```

DESCRIPTION DE LA LIGNE A IMPRIMER

```
01  LIGNE.
    05  FILLER                     PIC X(12) VALUE 'NOM'.
    05  NOM-LIGNE                  PIC X(30).
    05  FILLER                     PIC X(10) VALUE SPACES.
    05  FILLER                     PIC X(08) VALUE 'NE LE'.
    05  DATE-LIGNE                 PIC X(2)/X(2)/X(2).
    05  FILLER                     PIC X(10) VALUE '  DONNER'.
    05  DONNER-LIGNE               PIC **********.
    05  FILLER                     PIC X(10) VALUE ' AVOIR'.
    05  AVOIR-LIGNE                PIC **********.
    05  FILLER                     PIC X(10) VALUE ' SOLDE'.
    05  SOLDE-LIGNE                PIC **********.
```

DESCRIPTION D'UN ENREGISTREMENT

```
01  RECORD-B.
    05  NOM-RECORD                 PIC X(30).
    05  TOTAL-RECORD               PIC 9(10).
```

dans le listing du haut de cette page. Pour chaque carte A lue, on doit créer et imprimer la ligne écrite dans le listing du centre. Pour chaque carte B, on doit préparer et écrire sur fichier l'enregistrement décrit dans le listing du bas. L'organigramme du programme est décrit sur la page précédente avec le diagramme de flux des données.

Le traitement des cartes A nécessite l'exécution des opérations suivantes :

1 / Calculer, dans un champ de travail, SOLDE-DISPONIBLE, soit la différence entre DEBIT et CREDIT.

2 / Transférer la valeur ainsi obtenue dans le champ SOLDE-LIGNE

3 / Transférer, dans les champs correspondants de LIGNE, tous les autres champs de la CARTE A.

4 / Imprimer la ligne ainsi composée.

Pour le traitement des cartes B, on doit, au contraire :

1 / Calculer, dans le champ de travail TOTAL-DISPONIBLE, le résultat de la formule

SOLDE + INTERET — FRAIS

2 / Transférer la valeur obtenue dans le champs TOTAL-RECORD
3 / Transférer le champ NOM ET PRENOM dans NOM-RECORD
4 / Ecrire l'enregistrement dans le fichier FILE-OUT.

Dans cet exemple apparaissent des instructions et des mots qui ne vous ont pas encore été expliqués. En fait, nous avons surtout voulu mettre en valeur les avantages associés à l'utilisation des paragraphes et des sections dans l'écriture d'un programme. La PROCEDURE de ce programme pourrait être structurée comme le montre le listing ci-dessous. Cependant, il convient de noter qu'un programme écrit de cette manière comportera plusieurs inconvénients :

- Les phrases, écrites pour chaque branche de l'instruction IF, sont trop longues et ne pourraient pas être analysées correctement par le compilateur.
- Pour les programmes d'une plus grande complexité, le recours à un nombre élevé de sauts conditionnels ou inconditionnels (GO TO) peut ne pas rendre immédiate la mise en évidence de boucles (cycles infinis) non désirés.
- La possibilité de comprendre intuitivement et rapidement la fonction d'une branche du programme est considérablement réduite.

Pour contourner ces difficultés, la PROCEDURE DIVISION doit donc être rédigée en faisant appel aux sections, comme le montrent les listings des pages 991 et 992. La PROCEDURE DIVISION est ainsi réduite aux quelques lignes de la PREMIERE SECTION qui permettent de comprendre intuitivement les liens entre différentes branches du programme. Les autres sections, (après la PREMIERE SECTION), sont alors appelées et ensuite

STRUCTURE D'UNE PROCEDURE DIVISION (EXEMPLE 1)

```
PROCEDURE DIVISION.
DEBUT.
    OPEN INPUT CARTES.
    OPEN OUTPUT IMPRIMANTE.
    OPEN OUTPUT FILE-OUT.
LIRE-CARTE.
    READ CARTES
         AT END
         CLOSE CARTES
         CLOSE IMPRIMANTE
         CLOSE FILE-OUT
         GO TO FIN.
    IF   TYPE-CARTE IS EQUAL TO 'A'
         MOVE CARTE TO CARTE-A
         COMPUTE SOLDE-DISPONIBLE = CREDIT - DEBIT
         MOVE SOLDE-DISPONIBLE TO SOLDE-LIGNE
         MOVE NOM              TO NOM-LIGNE
         MOVE DATE-NAISSANCE   TO DATE-LIGNE
         MOVE DEBIT            TO DEBIT-LIGNE
         MOVE CREDIT           TO CREDIT-LIGNE
         WRITE LIGNE
         GO TO LIRE-CARTE.
    IF   TYPE-CARTE IS EQUAL TO 'B'
         MOVE CARTE TO CARTE-B
         COMPUTE TOTAL-DISPONIBLE = SOLDE + INTERET - FRAIS
         MOVE TOTAL-DISPONIBLE TO TOTAL-RECORD
         MOVE PRENOM-NOM       TO NOM-RECORD
         WRITE RECORD-B
         GO TO LIRE-CARTE.
FIN.
    STOP RUN.
```

STRUCTURE D'UNE PROCEDURE DIVISION (EXEMPLE 2)

```
 PROCEDURE DIVISION.
 PREMIERE SECTION.
 DEBUT.
     PERFORM OUVERTURE-FICHIERS.
     PERFORM PREMIERE-LECTURE.
 CALCUL.
     IF  TYPE-CARTE IS EQUAL TO 'A'
         PERFORM CALCUL-A
         WRITE LIGNE.
     IF  TYPE-CARTE IS EQUAL TO 'B'
         PERFORM CALCUL-B
         WRITE RECORD-B.
     PERFORM LIRE-CARTE.
     GO TO CALCUL.
*
*
*
*
*
 OUVERTURE-FICHIERS SECTION.
 OUVERTURE.
     OPEN INPUT CARTES.
     OPEN OUTPUT IMPRIMANTE.
     OPEN OUTPUT FILE-OUT.
 FIN-OUVERTURE. EXIT.
*
*
*
*
*
 FERMETURE-FICHIERS SECTION.
 FERMETURE.
     CLOSE CARTES.
     CLOSE IMPRIMANTE.
     CLOSE FILE-OUT.
 FIN-FERMETURE. EXIT.
*
*
*
*
*
 PREMIERE-LECTURE SECTION.
 PREMIERE-CARTE.
     READ CARTES
          AT END
          DISPLAY ' ** FICHIER CARTES VIDE ** '
                  UPON PRINTER
          PERFORM FIN-CALC.
 FIN-PREMIERE-CARTE. EXIT.
*
*
*
*
*
 CALCUL-A SECTION.
 CA-CARTE-A.
          MOVE CARTE TO CARTE-A
          COMPUTE SOLDE-DISPONIBLE = CREDIT - DEBIT.
          MOVE SOLDE-DISPONIBLE TO SOLDE-LIGNE.
          MOVE NOM              TO NOM-LIGNE.
          MOVE DATE-NAISSANCE   TO DATE-LIGNE.
          MOVE DEBIT            TO DEBIT-LIGNE.
          MOVE CREDIT           TO CREDIT-LIGNE.
 FIN-CA-CARTE-A. EXIT.
*
*
*
```

```
*
*
 CALCUL-B SECTION.
 CA-CARTE-B.
          MOVE CARTE TO CARTE-B.
          COMPUTE TOTAL-DISPONIBLE = SOLDE + INTERET - FRAIS.
          MOVE TOTAL-DISPONIBLE TO TOTAL-RECORD.
          MOVE PRENOM-NOM       TO PRENOM-RECORD.
 FIN-CA-CARTE-B. EXIT.
*
*
*
*
*
 LIRE-CARTE SECTION.
 AUTRES-CARTES.
     READ CARTES
          AT END
          PERFORM FIN-CALC.
 FIN-AUTRES-CARTES. EXIT.
*
*
*
*
*
 FIN-CALC SECTION.
 FIN.
     PERFORM FERMETURE-FICHIERS.
     DISPLAY ' *** FIN ELABORATION *** '
             UPON PRINTER.
     STOP RUN.
*
*
*
*
*
```

exécutées pas à pas, par l'intermédiaire du mot PERFORM.
On reviendra sur l'emploi de l'instruction PERFORM et sur ses modalités d'emploi, ainsi que sur toutes les autres instructions mentionnées dans ces exemples.

Les instructions du Cobol

Avant d'analyser en détail le langage Cobol, il est essentiel de connaître certaines conventions et restrictions imposées au programmeur dans l'écriture d'un programme. L'ensemble des caractères admis par le compilateur est constitué par toutes les lettres de l'alphabet anglais, les nombres naturels de 0 à 9 et un ensemble réduit de caractères spéciaux auxquels le compilateur associe des significations résumées dans le tableau ci-contre. Langage descriptif, le Cobol emploie des mots entiers et abrégés de la langue anglaise qui sont des actions associées au compilateur. Ces mots (environ 250) ne sauraient être utilisés par le programmeur autrement qu'à travers la syntaxe du langage et sans altération orthographique par omission ou insertion de caractères. C'est pourquoi on les appelle **mots réservés.** Ainsi, SOURCE-COMPUTER ne peut être écrit SOURCE COMPUTER ou SOURCECOMPUTER car le compilateur ne le reconnaîtrait pas.
De même, le programmeur n'est pas autorisé

à employer le mot SOURCE-COMPUTER comme nom de variable ou de fichier.
L'exemple suivant montre l'emploi erroné du mot réservé DATA comme nom de champ défini par le programmeur :

```
01 DATA.
   05 JOUR        PIC 99.
   05 MOIS        PIC 99.
   05 ANNEE       PIC 99.
```

Voici un autre exemple où sont mis en évidence certains mots réservés :

```
PROCEDURE DIVISION.

DEBUT SECTION.
DEBUTER.
    OPEN INPUT FILE-IN.
    OPEN INPUT CARTES.
    OPEN OUTPUT FICHIER-SORTIE
    PERFORM CONTROLE-CARTES.
    IF ERREUR = 1
    GO TO FIN.
CALCUL.
    READ FICHIER-ENTREE INTO FICHIER-SORTIE
        AT END
FIN.
    CLOSE FICHIER-ENTREE
          FICHIER-SORTIE
          CARTES.
    STOP RUN
```

Pour rendre le langage un peu plus familier, on a imaginé des constantes figuratives associées à certaines séries de caractères d'emploi plus fréquents. On en trouvera la signification dans le tableau suivant qui donne aussi, dans la même case, les formats d'une constante.

Constante figurative	Caractères équivalents
ZERO / ZEROS / ZEROES	Un ou plusieurs zéros numériques ou même un ou plusieurs 0 selon le contexte
SPACE / SPACES	Un ou plusieurs espaces (blanks),
LOW-VALUE / LOW-VALUES	Un ou plusieurs caractères avec le plus faible poids dans la séquence du code utilisé.
HIGH-VALUE / HIGH-VALUES	Un ou plusieurs caractères avec le plus fort poids dans la séquence du code utilisé.
QUOTE / QUOTES	Un ou plusieurs guillemets (").
ALL'caractère désiré'	Un ou plusieurs "caractères désirés" jusqu'au remplissage du champs récepteur.

Un champ générique (dénommé CHAMP) est capable d'accueillir jusqu'à six caractères. Le tableau de la page suivante illustre le

Symbole	Description	Signification pour le compilateur	Catégorie du symbole
.	Point	Fin d'une période	Ponctuation
,	Virgule	Séparateur d'une série	Ponctuation
;	Point-virgule	Séparateur d'une série clauses	Ponctuation
'	Accent	Délimitateur d'une chaîne de caractères	Ponctuation
( )	Parenthèses	Délimitateurs d'indices	Ponctuation
+	Plus	Addition	Arithmétique
—	Moins	Soustraction	Arithmétique
*	Astérisque	Multiplication	Arithmétique
/	Barre (slash)	Division	Arithmétique
**	2 astérisques	Puissance (exposant)	Arithmétique
=	Egal	Egal	Arithmétique
>	Plus grand	Plus grand	Condition
<	Plus petit	Plus petit	Condition
( )	Parenthèses	Délimitent une expression ; contrôlent aussi une série d'opérations logiques et arithmétiques	Condition

transfert dans CHAMP des constantes figuratives énumérées.

Au début, nous avons signalé, comme une caractéristique du langage Cobol, sa parenté avec le langage courant dans lequel l'unité de dialogue est le mot. Ainsi, comme en langage clair, chaque action à exécuter par l'ordinateur est commandée à l'aide de **mots.** Rencontrés par le compilateur dans le programme source, ces derniers sont traduits en une série d'instructions codées en langage machine et aptes à agir sur les composants impliqués par ces instructions. On comprend intuitivement que les mots du Cobol sont tous des mots réservés qui ne peuvent être utilisés par le programmeur à d'autres fins que celles pour lesquelles ils ont été définis.
Les mots du Cobol sont classés en cinq catégories selon leur type de fonction :

Verbes d'E/S. (ou de I/O) La notation E/S est l'abréviation d'Entrée/Sortie (I/O = Input/Output), soit les mots qui autorisent le transfert des données de et vers des unités périphériques :
OPEN (ouvrir)
CLOSE (fermer)
READ (lire)
WRITE (écrire)
ACCEPT (admettre)
DISPLAY (affecter)

Verbes arithmétiques. Ils permettent d'agir sur des constantes et des champs numériques ; ils exécutent les 4 opérations arithmétiques :
COMPUTE (calculer)
ADD (additionner)
SUBTRACT (soustraire)
MULTIPLY (multiplier)
DIVIDE (diviser)

Verbes de transfert ou de manipulation des données. Ils permettent le transfert du contenu d'un champ, élémentaire ou composé, d'une position mémoire vers une autre (transfert), la transformation de plusieurs champs en un seul champ et vice versa, ainsi que des opérations particulières sur une donnée :
MOVE (transférer)
INSPECT (vérifier)
STRING (enchaîner)
UNSTRING (séparer)

Verbes pour le contrôle de la séquence des instructions. En phase d'exécution, les mots de cette catégorie modifient le déroulement normal (c'est-à-dire strictement séquentiel) des instructions du programme :
GO TO (aller à)
PERFORM (exécuter)
EXIT (sortir)
STOP (arrêter)

Verbes conditionnels. Ils permettent d'analyser une donnée :
IF (si)
SEARCH (rechercher)

Verbes d'E/S

Ces verbes sont chargés du transfert des données entre la mémoire centrale et les unités externes (périphériques) tels que lecteurs

TRANSFERT DE CONSTANTES FIGURATIVES

MOVE ZEROS	TO CHAMPS.	0 0 0 0 0 0
MOVE SPACES	TO CHAMPS.	
MOVE LOW-VALUES	TO CHAMPS.	A A A A A A
MOVE HIGH-VALUES	TO CHAMPS.	Z Z Z Z Z Z
MOVE QUOTES	TO CHAMPS.	" " " " " "
MOVE ALL '*'	TO CHAMPS.	* * * * * *

de cartes, imprimantes, bandes et disques. Pour pouvoir opérer sur de telles unités par l'intermédiaire de mots ou d'instructions d'entrée/sortie, le programmeur doit déclarer, au compilateur, qu'au cours de calcul il accèdera aux périphériques.
On sait maintenant que, pour cela, on préfère l'INPUT-OUTPUT SECTION à l'ENVIRONNEMENT DIVISION.

```
INPUT-OUTPUT SECTION.
FICHIER-CONTROL.
    SELECT FICHIER-A ASSIGN TO DISC
    FICHIER A
```

ne dit pas si le FICHIER-A sera utilisé en lecture ou en écriture. Rappelons que l'ENVIRONNEMENT DIVISION, chargée de la définition des interactions avec l'ordinateur, peut varier d'une machine à l'autre*.
Seules exceptions à ces règles : les fichiers assignés au lecteur de cartes (CARD-READER), à l'imprimante (PRINTER) et au perforateur de cartes (CARD-PUNCH) pour lesquelles on donne le mode d'accès. En effet, un lecteur de cartes ne peut être utilisé que comme unité d'entrée, une imprimante ou un perforateur de cartes que comme unité de sortie.
L'exemple suivant met en évidence les différentes opérations à réaliser sur un fichier Cobol pour en utiliser le contenu.
Supposons que nous voulions mettre à jour notre propre agenda (FICHIER = AGENDA) qui se trouve rangé dans le tiroir du bureau (UNITE = BUREAU).
La première opération consistera naturellement à ouvrir l'agenda, l'opération suivante à y reporter (OUTPUT) les notes de la journée. Au moment où l'agenda a été ouvert, on avait déjà l'intention d'effectuer cette opération (OUVRIR POUR ECRIRE = OPEN OUTPUT). Une fois la mise à jour terminée, il faut fermer l'agenda (FERMER AGENDA = CLOSE AGENDA).
Supposons maintenant que l'ordinateur est doté d'une unité appelée BUREAU ; dans ce cas, la série des opérations décrites pourrait se traduire par les instructions contenues dans le listing de la page 996.

Ouverture et fermeture des fichiers : les verbes OPEN et CLOSE

Avant de poursuivre, il est bon d'expliquer le mécanisme activé par les opérations d'ouverture et de fermeture d'un ou de plusieurs fichiers à l'intérieur d'un programme.
Quand, en phase d'exécution, un programme rencontre une instruction OPEN, il réserve en mémoire l'espace (tampon) nécessaire aux opérations sur ce fichier, contrôle ou crée l'étiquette d'en-tête et positionne l'unité de lecture du support où réside le fichier au début du fichier lui-même.
L'opération d'ouverture d'un fichier positionne l'unité sur le premier enregistrement de ce fichier ; par contre, elle ne rend pas disponible son contenu.
Pour transférer le contenu de l'enregistrement de ou vers le fichier, il est nécessaire de le lire ou de l'écrire explicitement en faisant appel au verbe READ ou WRITE.
Généralement, on assigne à chaque programme une zone mémoire qui ne comprend pas les espaces réservés à la lecture et à l'écriture sur les fichiers déclarés dans l'ENVIRONNEMENT DIVISION et décrits dans la DATA DIVISION.
De tels espaces sont alloués en mémoire lors de l'ouverture du fichier. Pour cette raison, il peut arriver que, pendant l'exécution d'un programme, celui-ci termine sur une erreur pour violation de la capacité de mémoire dès l'opération d'ouverture d'un fichier. Puisqu'un fichier peut être ouvert et fermé un nombre quelconque de fois dans un programme, il est préférable, si aucune exigence particulière de programmation n'intervient, de fermer le fichier immédiatement après son utilisation. Comme on l'a vu, l'ouverture d'un fichier suppose aussi la déclaration du mode d'utilisation du fichier lui-même. En d'autres termes, le Cobol permet d'ouvrir un fichier en le rendant accessible à des opérations de :

Lecture	(INPUT)
Ecriture	(OUTPUT)
Lecture-Ecriture	(I/O) ou (E/S)
Ajout	(EXTEND)

*Dans ce cas, on utilise la syntaxe typique des systèmes UNIVAC de la série 1100 (Sperry Corporation), parce qu'elle est d'une interprétation plus immédiate que celle des autres systèmes.

EXEMPLE D'OUVERTURE D'UN FICHIER

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. AGG-AGENDA.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. ...........
 OBJECT-COMPUTER. ...........
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT AGENDA ASSIGN TO BUREAU.
 DATA DIVISION.
 FILE SECTION.
 FD  AGENDA
     LABEL RECORD STANDARD.
 01  NOTE.
     05  ............
         ............
         ............
         ............
         ............
 *
 *
 *
 *
 *
  PROCEDURE DIVISION.
  PREMIERE SECTION.
  OUVERTURE-FICHIER.
      OPEN OUTPUT AGENDA.
      PERFORM ECRIRE-NOTE.
      CLOSE AGENDA.
      STOP RUN.
 *
 *
 *
 *
 *
  ECRIRE-NOTE SECTION.
  ECRIRE.
        ............
        ............
        ............
        ............
    WRITE NOTE.
 FIN-ECRIRE. EXIT.
 *
 *
```

Utilisation en lecture : OPEN INPUT. Le format complet de l'instruction OPEN est donné dans le tableau de la page suivante. Les clauses optionnelles RESERVED et WITH NO REWING se réfèrent aux fichiers sur unité de bande. En écrivant simplement :

OPEN INPUT-FICHIER-A

quelle que soit l'unité à laquelle il est assigné, le fichier est ouvert et la tête de lecture est positionnée sur le premier enregistrement. Dans le cas d'un fichier assigné à une unité bande, celle-ci est rembobinée jusqu'à son point de départ (LOAD POINT) à moins que ne soit mentionnée une des deux clauses citées. En utilisant la clause RESERVED, la tête de lecture est positionnée sur le dernier enregistrement du fichier de façon à pouvoir le lire à l'envers, alors que l'emploi de la clause WITH NO REWIND empêche de rembobiner la bande.

Utilisation en écriture : OPEN OUTPUT. L'ouverture d'un fichier en écriture est obtenue avec l'instruction OPEN décrite en bas de page. La clause WITH NO REWIND conserve la même fonction qu'en lecture. Notons qu'ouvrir un fichier en OUTPUT signifie préparer le périphérique à écrire en partant du premier enregistrement du fichier : le contenu éventuel de l'enregistrement sera écrasé par les opérations suivantes d'écriture (WRITE). A titre d'exemple, supposons que nous devions lire le fichier F1-BANDE résidant sur bande pour en copier le contenu, enregistrement par enregistrement, sur le fichier F-DISQUE résidant sur disque. Ce dernier devra être copié ensuite sur bande comme F2-BANDE à la suite de F1-BANDE. Supposons de plus que le fichier F2-BANDE doive être organisé comme à la page 998 et dans le listing de la page 999. Les mots READ et WRITE sont utilisés par les sections du programme bien que n'ayant pas été encore décrits car leur fonctions est évidente.

Après avoir ouvert le fichier F1-BANDE en INPUT avec la clause Reserved, la première opération de lecture sera effectuée sur le dernier enregistrement du fichier. Cet enregistrement est lu et sauvegardé dans un espace travail (DISPONIBLE-REC-F1) d'où il sera ensuite transféré dans le fichier F2-B BANDE comme premier enregistrement.

Pour copier F1-BANDE en entier sur F-DISQUE, il est maintenant nécessaire de positionner le fichier F1-BANDE à son premier enregistrement (sur LOAD POINT). Il suffit alors de fermer le fichier et de le rouvrir sans ajouter d'autres clause (CLOSE, F1-BANDE, OPEN INPUT F1-BANDE). Quant à l'instruction OPEN OUTPUT F-DISQUE, elle ouvre le fichier F-DISQUE et le prépare à recevoir des données en écriture.

L'opération de lecture de F1-BANDE et d'écriture sur F-DISQUE est réalisée, enregistrement par enregistrement, à partir de la SECTION LIRE F1-ECRIRE-F.

Après avoir lu F1-BANDE jusqu'à la fin, l'unité est positionnée sur le caractère de fin-fichier de F1-BANDE. Puisque le programme doit maintenant copier le contenu de F-DISQUE sur F2-BANDE à la suite de F1-BANDE, il faut fermer F1-BANDE. Il suffit d'ajouter la clause WITH NO REWIND aux deux instructions citées. La dernière partie du programme se comprend immédiatement car les opérations successives sont :

- Copier le champ DISPONIBLE-REC-F1, dans lequel est sauvegardé le contenu du dernier enregistrement de F1-BANDE, dans le premier enregistrement de F2-BANDE.
- Copier le contenu de F-DISQUE dans F2-BANDE (PERFORM LIRE-F-ECRIRE-F2).
- Fermer tous les fichiers encore ouverts et rembobiner la bande (CLOSE F-DISQUE, F2-BANDE).

Utilisation en lecture-écriture : OPEN E/S. Le fichier déclaré doit nécessairement résider sur disque. On peut effectuer sur lui autant d'opérations de lecture que d'enregistrement. Le format est :

OPEN E-S nom-du-fichier

OPEN EXTEND. Grâce à cette instruction, le fichier, se trouvant sur disque ou sur bande à l'ouverture, est parcouru jusqu'à son

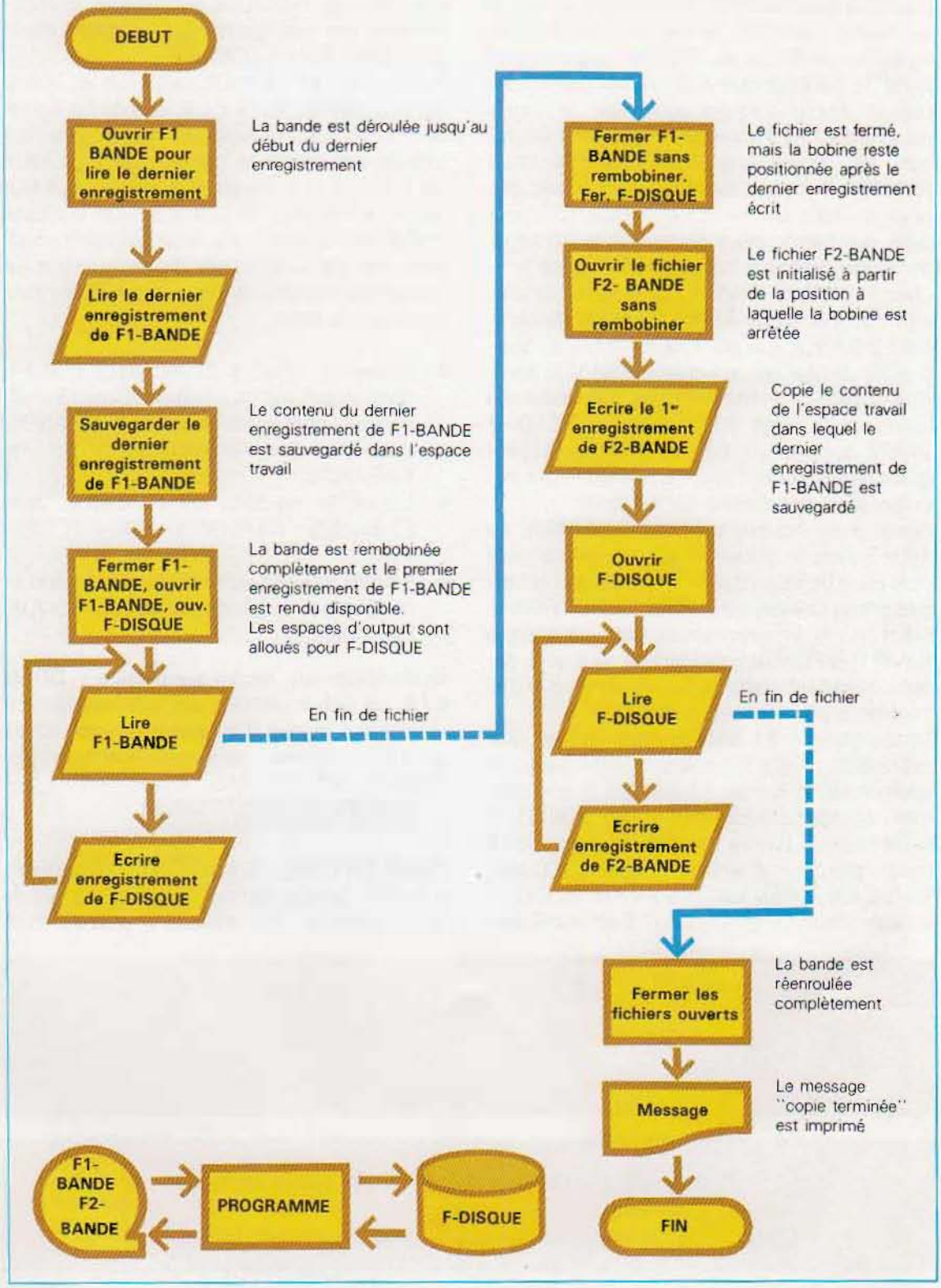
FORMAT DE L'INSTRUCTION OPEN EN LECTURE

OPEN INPUT nom du fichier { REVERSED / WITH NO REWIND }

FORMAT DE L'INSTRUCTION OPEN EN ECRITURE

OPEN OUTPUT nom du fichier [WITH NO REWIND]

EXEMPLE D'EMPLOI DE OPEN ET CLOSE



EXEMPLE D'EMPLOI DE OPEN ET CLOSE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COPIE.
REMARKS. CE PROGRAMME EST UN EXEMPLE D'EMPLOI
         DES INSTRUCTIONS OPEN ET CLOSE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT F1-BANDE   ASSIGN  TO TAPE F1.
    SELECT F2-BANDE   ASSIGN  TO TAPE F2.
    SELECT F-DISQUE   ASSIGN  TO DISC F-DISQUE.
DATA DIVISION.
FILE SECTION.
FD  F1-BANDE
    LABEL RECORD STANDARD.
01  REC-F1-BANDE.
    05  ..............
    05  ..............
FD  F2-BANDE
    LABEL RECORD STANDARD.
01  REC-F2-BANDE.
    05  ..............
    05  ..............
FD  F-DISQUE
    LABEL RECORD STANDARD.
01  REC-F-DISQUE.
    05  ..............
    05  ..............
*
 WORKING-STORAGE SECTION.
 01 REC-DISPONIBLE-F1          PIC X(...).
*
PROCEDURE DIVISION.
PREMIERE SECTION.
DEBUT.
    DISPLAY '** DEBUT COPIE **' UPON PRINTER.
    OPEN INPUT F1-BANDE REVERSED.
    PERFORM LIRE-DERNIER-REC-F1.
    CLOSE F1-BANDE.
    OPEN INPUT F1-BANDE.
    OPEN OUTPUT F-DISQUE.
    PERFORM LIRE-F1-ECRIRE-F.
    CLOSE F1-BANDE WITH NO REWIND.
    CLOSE F-DISQUE.
    OPEN  F2-BANDE WITH NO REWIND.
    OPEN INPUT F-DISQUE.
    PERFORM COPIE-DERNIER-REC-F1.
    PERFORM LIRE-F-ECRIRE-F2.
    CLOSE F-DISQUE.
    CLOSE F2-BANDE.
    DISPLAY '** COPIE TERMINEE **'
            UPON PRINTER.
    STOP RUN.
*
*  = = = = = = =    SECTIONS   = = = = = = = = = =
*
 LIRE-DERNIER-REC-F1 SECTION.
 DERNIER-REC.
        ..............
 DERNIER-REC-EX. EXIT.
*
 LIRE-F1-ECRIRE-F SECTION.
 F1-F.
        ..............
 F1-F-EX. EXIT.
*
 COPIE-DERNIER-REC-F1 SECTION.
 COPIE-DERNIER.
        ..............
COPIE-DERNIER-EX. EXIT.
*
 LIRE-F1-ECRIRE-F2 SECTION.
 F1-F2.
        ..............
 F1-F2-EX. EXIT.
```

dernier enregistrement logique et est prêt pour l'écriture d'autres enregistrement à partir de cette position. Le format est :

```
OPEN EXTEND nom-du-fichier
```

Notons l'importance de cette instruction pour effectuer des ajouts de données sur des fichiers déjà existants. Au lieu d'écrire :

```
OPEN E/S FICHIER-A.
PERFORM LIRE-TOUT-FICHIER-A.
PERFORM
ECRIRE-NOUVEAUX-RECORD.
CLOSE FICHIER-A.
```

Avec la clause EXTEND, on a recours à la séquence suivante qui effectue les mêmes opérations :

```
OPEN EXTEND FICHIER-A.
PERFORM
ECRIRE-NOUVEAUX-RECORD.
CLOSE FICHIER-A.
```

L'avantage de cette procédure est surtout évident quand le nombre d'enregistrements contenus dans les fichiers est grand. Cependant, la clause EXTEND n'est pas utilisable pour un fichier vide : le fichier ouvert avec la clause EXTEND contient au moins une donnée. Les formats complets des instructions OPEN et CLOSE sont donnés ci-dessous. En utilisant l'extension LOCK avec CLOSE, on ferme un fichier de manière à empêcher sa réouverture au cours du programme. Pour une plus grande lisibilité du programme, il est conseillé d'écrire les instructions en les disposant en colonne :

```
OPEN INPUT          FICHIER-1
                    FICHIER-2
                    FICHIER-3
        OUTPUT      FICHIER-4
                    FICHIER-5
.......................................
CLOSE FICHIER-1
      FICHIER-2 WITH LOCK
                WITH LOCK
      FICHIER-5
```

Lecture et écriture sur les fichiers : les verbes READ et WRITE

Dans les exemples donnés jusqu'à présent, une opération de lecture sur fichier s'effectue avec READ, l'écriture d'un enregistrement sur un fichier avec WRITE. Le format le plus simple de ces instructions est :

```
READ nom-du-fichier AT END
                    phrase-impérative.
WRITE nom de l'enregistrement.
```

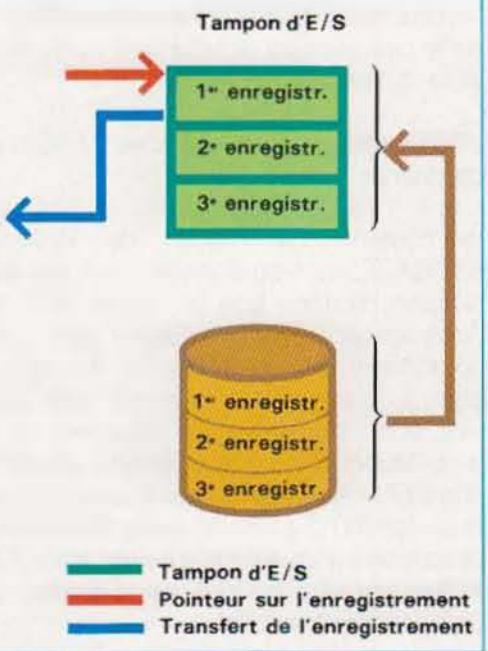
(Pour mémoire, on lit (READ) un fichier et on écrit (WRITE) un enregistrement).
Chaque instruction de lecture sur un fichier opère le transfert d'un enregistrement logique. En fichiers séquentiels, on effectue autant de lectures que d'enregistrements contenus dans le fichier, entre l'ouverture et la fermeture du fichier. Après chaque commande de lecture, un test de fin de fichier est automatiquement réalisé. Lorsque le système d'exploitation reconnait la fin du fichier, la phrase décrite dans la clause

FORMATS COMPLETS DES INSTRUTIONS OPEN ET CLOSE

```
{                                                  }
{          INPUT      { nom-du-fichier }   [{ REVERSED      }]   }
{                                          [{ WITH NO REWIND }]   }
{OPEN      OUTPUT     nom-du-fichier       [{ WITH NO REWIND }]   }
{          E-S        { nom-du-fichier                            }
{          EXTEND     nom-du-fichier                              }
{                                                                }
CLOSE      nom-du-fichier      [WITH      { NO REWIND }]
                                          { LOCK      }]
```

MECANISME DE LECTURE D'UN FICHIER

```
FD FICHIER-A
    LABEL ENREGISTREMENT STANDARD
    BLOCK CONTAINS 3 RECORDS
    ENREGISTREMENT CONTAINS 100 CHARACTERS
    01 REC-A                          PIC x (100)
    WORKING-STORAGE SECTION
    01 REC-A-W-S                      PI x (100)
    PROCEDURE DIVISION
    MAIN SECTION
    OUVRIR FICHIER
        OPEN INPUT FICHIER-A
        READ FICHIER-A INTO REC-A-W-S
            AT END
            DISPLAY 'FIN FICHIER-A'
                UPON PRINTER
            CLOSE FICHIER-A
```



AT END est exécutée. L'instruction READ est également utilisée avec le format :

READ nom-du-fichier INTO nom-de-donnée AT END phrase-impérative.

Dans le premier format, le contenu de l'enregistrement est transmis au programme par l'intermédiaire de la zone décrite dans la FILE-SECTION. Par contre, dans le second format, le même enregistrement est copié dans une zone définie dans WORKING-STORAGE SECTION.
Pour illustrer le mécanisme de lecture d'un fichier, considérons le cas simple (représenté dans le schéma ci-dessus) du fichier FICHIER-A constitué par six enregistrement de longueur fixe de 100 caractères avec un facteur de blocage égal à 3.
Le contenu de la zone REC-A est inconnu avant l'ouverture du fichier. Cette ouverture comporte l'allocation en mémoire d'une zone tampon dont la dimension permet de contenir trois enregistrement de 100 caractères chacun, c'est-à-dire le résultat de la lecture du premier bloc de trois enregistrements dans le tampon. A ce moment, le contenu de REC-A est encore indéfini. La première lecture (READ) n'effectue aucun transfert de données du tampon vers REC-A. Elle se borne à créer un pointeur entre REC-A et le premier enregistrement dans le tampon, rendant aussi disponible pour le programme le contenu de l'enregistrement. Si la clause INTO est spécifiée, le contenu du premier enregistrement est copié dans REC-A-W-S. La seconde lecture fait pointer REC-A sur ce second enregistrement et copie son contenu du tampon vers REC-A-W-S... La quatrième lecture, trouvant le tampon vide, va lire le second bloc du fichier. Elle « libère » le premier enregistrement du tampon, effectuant comme toujours le copie dans REC-A-W-S. La lecture suivante rencontre la marque de fin-fichier et exécute la phrase

```
DISPLAY 'FIN FICHIER-A'
            UPON PRINTER
CLOSE   FICHIER-A
```

La fermeture du fichier comporte l'effacement du tampon (le contenu de REC-A n'est plus disponible). Le processus est le même pour l'instruction WRITE, en sens inverse.

Dans le format

WRITE nom-de-l'enregistrement

l'instruction effectue les opérations d'écriture en se positionnant directement sur le tampon alors qu'avec la forme :

WRITE nom-de-l'enregistrement FROM nom-de-donnée

le contenu du champ de WORKING-STORAGE, nom-de-donnée, est copié dans le tampon. Notons que la clause INTO pour l'instruction READ et FROM pour WRITE constituent des transferts de données d'un point à un autre de la mémoire. Elles respectent donc les règles qui régissent ce type d'opération. Notons que, si dans une instruction READ, le nom-de-donnée accompagnant la clause INTO a une longueur inférieure à la description d'un enregistrement à lire, l'enregistrement sera tronqué sur la droite.
Soit l'exemple suivant :

```
DATA DIVISION.

FILE SECTION.
FD CARTES.
LABEL RECORD OMITTED.
01 CARTE                          PIC X(80).

WORKING-STORAGE SECTION.

01 CARTE-W-S.
   05 TYPE-CARTE                  PIC X(2).
   05 DATE-CARTE.
      10 JOUR                     PIC 9(2).
      10 MOIS                     PIC 9(2).
      10 ANNEE                    PIC 9(2).
   05 ARTICLE.
      10 SERIE                    PIC X(2).
      10 NUMERO                   PIC 9(4).
   05 FILLER                      PIC X(3).
```

(Notons que l'enregistrement CARTE du fichier CARTES est défini par 80 caractères alors que le champ de WORKING-STORAGE CARTE-W-S ne comporte que 17 caractères).

```
PROCEDURE DIVISION.

LIRE-FICHIER.
OPEN INPUT CARTES.
READ CARTES INTO-CARTE-W-S
                AT END...
```

Si le contenu de la carte lue (présente dans le tampon) est, par exemple,

AA101283XZ7411CABLETELEPHONIQUE

on a dans le champ CARTE-W-S

AA101283XZ7411CAB

Inversement, si le champ CARTE-W-S avait été

01 CARTE-W-S PIC X(150).

l'instruction

READ CARTES INTO CARTE-W-S
AT END...

aurait transféré le contenu entier de la carte lue, remplissant les positions de la 81e à la 150e avec des caractères blancs.
La règle énoncée demeure également valable pour l'instruction WRITE :

- Si la longueur du champ spécifié dans la clause FROM est inférieure à la description de l'enregistrement d'output, celui-ci chargera le champ entier de WORKING STORAGE, plus une série de blancs, jusqu'au remplissage de l'enregistrement.
- Dans le cas où le champ de FROM a, au contraire, une longueur supérieure à l'enregistrement, son contenu sera coupé aux dimensions maximales de l'enregistrement d'output.

Bien que les formats qui définissent directement les tampons entrée/sortie soient manifestement plus efficaces (car ils ne prévoient pas le transport des données d'un point à un autre de la mémoire), ils imposent une gestion des instructions plus rigoureuse. Il importe, en effet, de connaître exactement le moment où le contenu de l'enregistrement courant est fiable et quand il ne l'est plus. Voyons, à ce propos, une série d'instructions erronées qui illustrent les problèmes

auxquels nous avons fait allusion.

Premier cas :
WRITE RECORD-A.
MOVE RECORD-A TO DISPONIBLE-REC.
Chaque opération d'écriture rend le contenu de l'enregistrement courant inutilisable puisqu'elle l'écrase par une nouvelle valeur. Il en est de même pour READ.

Deuxième cas :
MOVE DISPONIBLE-REC TO RECORD-A
OPEN INPUT FICHIER-A.
L'opération d'ouverture d'un fichier ne rend pas disponible l'enregistrement qui, ailleurs, ne contient plus la valeur DISPONIBLE-REC.

Troisième cas :
READ FICHIER-A
AT END
MOVE RECORD-A TO DISPONIBLE-REC.
La condition de fin de fichier ne valide pas la valeur de l'enregistrement courant.

Ecriture sur fichiers d'imprimante. Les modalités d'utilisation de l'instruction WRITE pour les fichiers sur disque restent également valables pour les fichiers assignés à l'imprimante.
Pour ces derniers, il existe des clauses permettant de contrôler la tête d'impression. Rappelons cependant que les fichiers assignés à l'imprimante (PRINTER) doivent toujours être déclarés avec la clause LABEL RECORD OMITTED et que la longueur de l'enregistrement doit, au maximum, faire 132 caractères (taille du tampon de l'imprimante) plus 1, car une position est réservée au caractère de contrôle de la tête. Le format complet de l'instruction WRITE est développé dans le tableau ci-dessous.
Les états produits par un programme Cobol sont rarement de simples listings. En effet, on a souvent besoin de mettre en évidence les résultats des calculs sous forme de tableaux. De plus, un tableau peut aussi être très long ; aussi est-il indispensable que non seulement la première mais chaque page d'un état comporte une ou plusieurs lignes de marge expliquant la nature et la signification des informations reproduites. Examinons maintenant les clauses simples du WRITE assurant la gestion des fichiers d'imprimante.

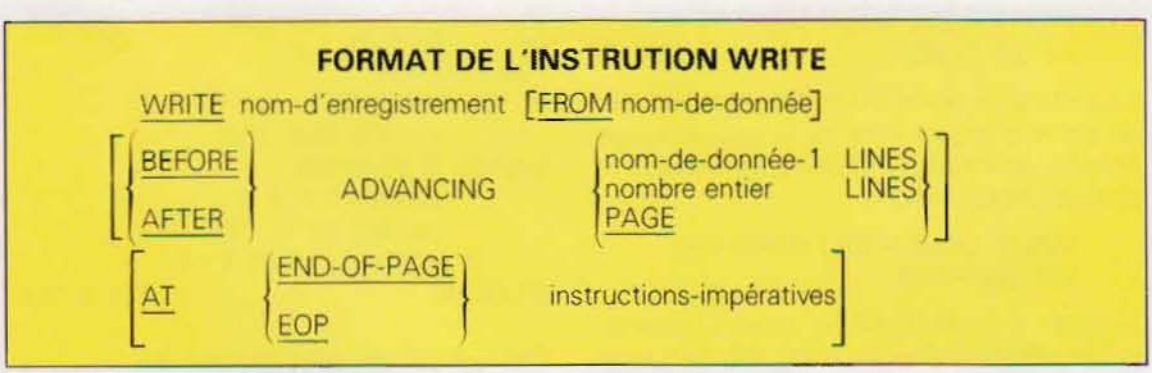
Clauses de saut de lignes. Si aucune clause n'est spécifiée dans l'instruction WRITE, celle-ci imprime le contenu de l'enregistrement dans la ligne qui suit immédiatement celle qui est imprimée. Si un autre type d'impression est demandé, le compilateur tient compte du nombre de lignes devant être sautées avant l'impression de la ligne contenue dans la mémoire.
En d'autres termes,

```
WRITE LIGNE FROM LIGNE-W-S
AFTER ADVANCING 3 LIGNES.
```

équivaut à dire : « après avoir sauté 3 lignes (AFTER ADVANCING 3 LINES) écrire (WRITE) l'enregistrement LIGNE en prélevant le contenu du champ de WORKING-STORAGE LIGNE-W-S ».
On peut d'ailleurs obtenir l'impression de la ligne en premier et seulement ensuite le saut d'un certain nombre de lignes en utilisant le format :

```
WRITE LIGNE FROM LIGNE-W-S
BEFORE ADVANCING 3 LINES.
```

FORMAT DE L'INSTRUTION WRITE



A savoir : « Ecrire l'enregistrement LIGNE en prélevant le contenu du champ LIGNE-W-S avant d'avancer de 3 lignes.

Le nombre de lignes dont on fait progresser la tête (en réalité c'est le papier qui avance) peut aussi être un nombre entier sans signe contenu dans un champ numérique élémentaire défini dans WORKING-STORAGE SECTION. Les exemples précédents sont donc équivalents à :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER ADVANCING SAUT LIGNES.
WRITE LIGNE FROM LIGNE-W-S
BEFORE ADVANCING SAUT LINES.
```

où SAUT est défini dans WORKING-STORAGE SECTION ainsi :

```
01 SAUT PIC 9 (2) VALUE 3.
```

Indépendamment de la méthode utilisée pour annoncer les lignes à sauter, leur nombre ne peut dépasser 99.
Enfin, on observe que les mots ADVANCING et LINES ne sont pas obligatoires : de ce fait, les mêmes instructions peuvent s'écrire :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER 3.
WRITE LIGNE FROM LIGNE-W-S
AFTER SAUT.
WRITE LIGNE FROM LIGNE-W-S
BEFORE 3.
WRITE LIGNE FROM LIGNE-W-S
BEFORE SAUT.
```

Clause de saut de pages. Comme pour le saut de lignes, l'instruction WRITE comporte une gestion de la page qui reçoit l'impression de la ligne. La clause PAGE permet de sauter à la page suivante :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER PAGE.
```

équivaut à : « après (AFTER) s'être positionné sur la première ligne de la nouvelle page (PAGE), écrire (WRITE) LIGNE en utilisant le champ LIGNE-W-S » ;

```
WRITE LIGNE FROM LIGNE-W-S
BEFORE PAGE.
```

Signifie : « écrire LIGNE en utilisant le contenu de LIGNE-W-S avant (BEFORE) de sauter à la première ligne utile de la nouvelle page ». Si elle n'est pas explicitement modifiée par le programmeur avec des clauses spéciales, une telle ligne est directement gérée par le système d'exploitation.

La clause LINAGE. Pour organiser de manière personnalisée les pages d'un état, le programmeur a recours à la clause LINAGE dans la FILE SECTION. Cette clause signale au compilateur les points suivants :

- Le nombre total de lignes à imprimer dans le champ de la page (LINAGE) doit être plus grand que 0.
- Le nombre de lignes vides précédant les lignes à imprimer (TOP) peut être égal à 0.
- Le nombre de lignes vides suivant les lignes à imprimer (BOTTOM) peut être égal à 0.
- Le numéro de la ligne, où doit être imprimé l'enregistrement courant avant de sauter à unr nouvelle page (FOOTING), doit se situer entre les lignes pouvant être imprimées.

Le format complet de la clause LINAGE s'écrit comme indiqué dans le tableau ci-dessous.
Tous les paramètres annoncés doivent être des nombres entiers ou des noms de données numériques élémentaires sans signe.
Ecrire donc

FORMAT DE LA CLAUSE LINAGE

```
LINAGE IS nombre-ligne-pouvant être imprimées LINES
[WITH FOOTING AT ligne-de-saut]
[LINES AT TOP marge-supérieure]
[LINES AT BOTTON marge-inférieure]
```

```
DATA DIVISION.
FILE SECTION.
FD IMPRIMANTE
LABEL RECORD OMITTED
LINAGE IS 56 LINES
        LINES AT TOP 6
        LIGNES AT BOTTOM 4
        WITH FOOTING AT 56
01 LIGNE                    PIC X(133).
```

Ces instructions sont équivalentes à :

```
DATA DIVISION.

FILE SECTION.
FD IMPRIMANTE
LABEL RECORD OMITTED

LINAGE IS NOMBRE-LIGNES LINES
   LINES AT TOP MARGE-SUPERIEURE
   LINES AT BOTTOM MARGE-
   INFERIEURE
   WITH FOOTING LIGNE-FOOTING.
01 LIGNE                          PIC X(133).

WORKING-STORAGE SECTION.
01 NOMBRES-LIGNES  PIC 9(2) VALUE 56.
01 MARGE-SUPERIEURE    PIC 9 VALUE 6.
01 MARGE-INFERIEURE    PIC 9 VALUE 4.
01 LIGNE-FOOTING    PIC 9(2) VALUE 56.
```

Les clauses de LINAGE sont optionnelles. Dans le cas où elles ne sont pas utilisées, les valeurs suivantes sont prises par défaut :

marge-supérieure = 0
marge-inférieure = 0
Ligne-de-saut (footing) = nombres de lignes pouvant être imprimées.

La clause LINAGE accélère la gestion de l'imprimante. En effet, au moment de l'ouverture d'un fichier d'imprimante auquel est associée la cause LINAGE, on assigne également un registre spécial (LINAGE-COUNTER) géré automatiquement par le Cobol pendant l'exécution des instructions WRITE. Ceci est illustré par l'exemple ci-dessous. On en déduit que, lorsqu'on utilise la clause LINAGE dans la description d'un fichier d'imprimante, le Cobol est en mesure d'identifier, par l'intermédiaire du LINAGE-COUNTER, la fin de la page délimitée par les paramètres en FILE-SECTION. En utilisant cette facilité, il est possible de déclarer, dans l'instruction WRITE, une action que le programme doit accomplir immédiatement après la fin d'une page.

IDENTIFICATION DIVISION

```
—
FILE SECTION
FD IMPRIMANTE
    LABEL RECORD OMITTED
    LINAGE IS 50.
           TOP 10
           BOTTOM 6.
01 LIGNE                          PIC X(133).
WORKING-STORAGE SECTION.
01 LIGNE-W-S.
    05
    —
    —
PROCEDURE DIVISION.
OUVERTURE-FICHIER.
    OPEN OUTPUT IMPRIMANTE
    —
    —
    —
    —
    —
WRITE LIGNE FROM LIGNE W-S.
    —
    —
WRITE LIGNE FROM LIGNE W-S AFTER 3.
    —
WRITE LIGNE FROM LIGNE-W-S.
    —
    —
    —
    —
    —
    —
WRITE LIGNE FROM LIGNE-W-S AFTER
PAGE
```

Assigne le fichier et met à 0 le contenu du LINAGE-COUNTER.

Imprimer le contenu de LIGNE-W-S sur la 11e ligne de la feuille, c'est-à-dire après une marge supérieure à 10 lignes (TOP 10). Le LINAGE-COUNTER est incrémenté de 1.

Avancer le papier de 3 lignes, imprimer le contenu de LIGNE-W-S et incrémenter de 3 le LINAGE-COUNTER.

On suppose, qu'avec l'instruction WRITE utilisée en ce point, le chariot est positionné sur la 50e ligne, à savoir sur la dernière ligne susceptible d'écrire, en accord avec la clause LINAGE. Dans ce cas, le contenu de LIGNE est imprimé sur la première ligne utilisable à la page suivante. Le LINAGE-COUNTER est mis à 1, de façon à tenir compte uniquement des lignes imprimées dans la page commencée.

L'opération exécutée est analogue à la précédente pour le WRITE, mais elle est commandée par le programmeur qui, à ce point du programme, désire changer de page.

Supposons, par exemple, que nous voulions imprimer une marge au début de chaque page. Dans ce cas, si les instructions nécessaires à l'impression de la marge sont toutes contenues dans une SECTION appelée MARGE, on écrit :

```
WRITE LIGNE FROM LIGNE-W-S
AT END-OF-PACE PERFORM MARGE
```

ou, d'une manière plus concise

```
WRITE LIGNE FROM LIGNE-W-S
EOP PERFORM MARGE.
```

Les instructions ACCEPT et DISPLAY

Ces deux verbes d'E/S sont respectivement comparables à READ et à WRITE, mais ne sont opérationnels que sur de faibles quantités de données.
Avec ACCEPT, la programmeur est en mesure d'arrêter un calcul pour saisir des données sur la console système, le lecteur de cartes ou par certains registres particuliers du calculateur. Contrairement à ce qui se passe pour l'instruction READ, ACCEPT ne demande l'attribution d'aucune carte ni à l'intérieur ni à l'extérieur du programme.
L'instruction a le format décrit ci-dessous et permet de transférer, dans le champ de WORKING-STORAGE nom-de-donnée, l'entrée provenant d'une des zones systèmes (FROM) indiquées entre les accolades. De cette manière, le programme en cours reçoit de l'extérieur des paramètres variables.
Ces derniers ont des fonctions

- de documentation,
- de paramétrage,
- de communication avec l'opérateur.

On verra en détail la signification de cet ordre dans l'exemple étudié après la description du verbe DISPLAY. Pour le moment, on note que les éléments suivants sont transférés dans le champ, nom-de-donnée :

1 / la réponse numérisée à la console de l'opérateur quand la clause FROM CONSOLE est spécifiée ;
2 / le contenu d'une carte quand on emploie la clause FROM CARD-READER ;
3 / la date du jour avec le format JJMMAA quand on utilise la clause FROM DATE ;
4 / le numéro du jour avec le format JJJAA quand on utilise la clause FROM DAY;
5 / l'heure du jour exprimée avec 8 caractères avec le format HHMMSSCC quand la clause FROM TIME est utilisée.

Pour comprendre la signification des notations utilisées dans les trois derniers formats, on suppose que le programme a été exécuté le 15 janvier 1984 à 8 heures, 12 minutes, 35 secondes et 75 centièmes.
Les trois instructions suivantes

```
ACCEPT DATE-ACTUELLE FROM DATE.
ACCEPT DATE DU JOUR FROM DAY.
ACCEPT HEURE         FROM TIME.
```

imposent les champs respectifs d'arrivée aux valeurs ci-après reportées.

```
DATE-ACTUELLE = 150184
N°-DU-JOUR    = 01584
HEURE         = 08123575
```

On se souvient que les champs d'arrivée doivent avoir été définis avec une longueur respectivement de 6,5 et 8 caractères, numériques et sans signe.

Le verbe DISPLAY permet d'envoyer un message à la console système et à l'imprimante.

FORMAT DE L'INSTRUTION ACCEPT

ACCEPT nom-de-donnée FROM { CONSOLE | CARD-READER | DATE | DAY | TIME }

Un tel message peut être indifféremment composé de constantes alphanumériques ou du contenu des champs décrits dans le WORKING-STORAGE. Le format général de l'instruction figure dans le tableau. Un exemple d'emploi des instructions ACCEPT et DISPLAY est donné dans le tableau ci-dessous, relatif à un programme simple.

FORMAT DE L'INSTRUCTION DISPLAY

```
DISPLAY { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]  UPON { CONSOLE }
        { constante-1     }   { constante-2     }          { PRINTER }
```

EXEMPLE D'EMPLOI DES INSTRUCTIONS ACCEPT ET DISPLAY

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ACCEPT DISPLAY.
       REMARKS. CE PROGRAMME EST UN EXEMPLE D'EMPLOI
                DES INSTRUCTIONS ACCEPT ET DISPLAY.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.....................
       OBJCET-COMPUTER.....................
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MOUVEMENTS ASSIGN TO DISC MOUVEMENTS.
           SELECT STATISTIQUES ASSIGN TO DISC STATISTIQUES.
       DATA DIVISION.
       FILE SECTION.
       FD  MOUVEMENTS.
           LABEL RECORD STANDARD.
       01  REC-MOUV                        PIC X(14).
       FD  STATISTIQUES
           LABEL RECORD STANDARD.
       01  REC-STA                         PIC X(10).
      *
       WORKING-STORAGE SECTION.
       01  REC-MOUV-WS.
           05  ART-MOUV                    PIC 9(3).
           05  DATE-MOUV.
               01 ANNEE-MOUV               PIC 9(2).
               01 MOIS-MOUV                PIC 9(2).
               01 JOUR-MOUV                PIC 9(2).
           05  QUANTITE                    PIC 9(5).
      *
       01  REC-STA-WS.
           05  ART-STA                     PIC 9(3).
           05  MOIS-STA                    PIC 9(2).
           05  VENDU                       PIC 9(5).
      *
       01  CARTE-PARAMETRE.
           05  TYPE-SK-PAR                 PIC XX.
           05  ART-SK-PAR                  PIC 9(3).
      *
       01  DEMANDER-MOIS                    PIC X(80).  VALUE
           'DIGITALISATION DU NOMBRE DE MOIS A CALCULER'.
      *
       01  MOIS-DE-CONSOLE                 PIC 9(2).
      *
       01  DATE-CALCUL                PIC 9(6).
       01  DATE-CA  REDEFINES DATE-CALCUL.
           05  ANNEE-CA                    PIC 9(2).
           05  MOIS-CA                     PIC 9(2).
           05  JOUR-CA                     PIC 9(2).
      *
       01  HEURE-CALCUL               PIC 9(8).
       01  HEURE-CA  REDEFINES HEURE-CALCUL.
           05  HEURE                       PIC 9(2).
           05  MINUTE                      PIC 9(2).
```

```
           05  SECONDES                        PIC 9(2).
           05  CENTIEMES                       PIC 9(2).
       *
       01  CALCUL.
           05  FILLER                          PIC X(17)  VALUE
               'CALCUL DU'.
           05  JOUR-DISPLAY                    PIC 99.
           05  FILLER                          PIC X      VALUE '/'.
           05  MOIS-DISPLAY                    PIC 99.
           05  FILLER                          PIC X      VALUE '/'.
           05  ANNEE-DISPLAY                   PIC 99.
           05  FILLER                          PIC X(10)  VALUE
               '    AUX'.
           05  HEURES-DISPLAY                  PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  MINUTES-DISPLAY                 PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  SECONDES-DISPLAY                PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  CENTIEMES-DISPLAY               PIC 99.
       *
       01  LECTURE.
           05  FILLER                          PIC X(42)  VALUE
               'LECTURE ET CODAGE DE L'ARTICLE'
           05  ART-DISPLAY                     PIC 9(3).
       PROCEDURE DIVISION.
       DEBUT SECTION.
       ACCEPTE.
           ACCEPT DATE-CALCUL FROM DATE.
           ACCEPT HEURE-CALCUL FROM TIME.
           DISPLAY DEMANDER-MOIS
                   UPON CONSOLE.
           ACCEPT MOIS-DE-CONSOLE FROM CONSOLE.
       DECIDE-OPEN.
           IF   MOIS-DE-CONSOLE = 1
                OPEN OUTPUT STATISTIQUES
           ELSE
                OPEN EXTEND STATISTIQUES.
       ACCEPTE-SK-PAR.
           ACCEPT CARTE-PARAMETRE FROM CARD-READER.
           OPEN INPUT MOUVEMENTS.
       LIS-MOUVEMENTS.
           READ MOUVEMENTS INTO REC-MOUV-WS
                AT END
                PERFORM FERMETURE
                STOP RUN.
           IF   ART-MOUV NOT = ART-SK-PAR
                GO TO LIRE-MOUVEMENTS.
           PERFORM CALCUL-RECORD.
           WRITE REC-STA FROM REC-STA-WS.
           GO TO LIRE-MOUVEMENTS.
       *
       CALCUL-RECORD SECTION.
       CA-REC.
           ......
       CA-REC-EX. EXIT.
       FERMETURE SECTION.
       FERME.
           CLOSE MOUVEMENTS.
           CLOSE STATISTIQUES.
           MOVE ART-SK-PAR TO ART-DISPLAY.
           MOVE ANNEE-CA   TO ANNEE-DISPLAY.
           MOVE MOIS-CA    TO MOIS-DISPLAY.
           MOVE JOUR-CA    TO JOUR-DISPLAY.
           MOVE HEURES     TO HEURES-DISPLAY.
           MOVE MINUTES    TO MINUTES-DISPLAY.
           MOVE SECONDES   TO SECONDES-DISPLAY.
           MOVE CENTIEMES  TO CENTIEMES-DISPLAY.
           DISPLAY CALCUL UPON PRINTER.
           DISPLAY LECTURE UPON PRINTER.
           DISPLAY '** FIN CALCUL **'
                   UPON PRINTER.
       FERMETURE-EX. EXIT.
       *
```

Le programme des instructions ACCEPT et DISPLAY doit :

1/ lire un fichier sur le disque qui stocke les mouvements mensuels de tous les articles vendus dans un magasin,

2/ extraire uniquement les données d'un article codé sur carte perforée,

3/ mettre en file d'attente les données traitées derrière le fichier de statistiques,

4/ créer ou ouvrir en extension le fichier STATISTIQUES et demander au pupitreur à quel mois (janvier ou autre) correspondent les données en cours de traitement,

5/ imprimer en fin de traitement le message suivant :

TRAITEMENT DU (date) A (heure)
EXTRACTION ET MISE EN FILE D'ATTENTE POUR L'ARTICLE (code-article)
FIN TRAITEMENT.

Ces diverses opérations sont schématisées dans l'organigramme de la page suivante.

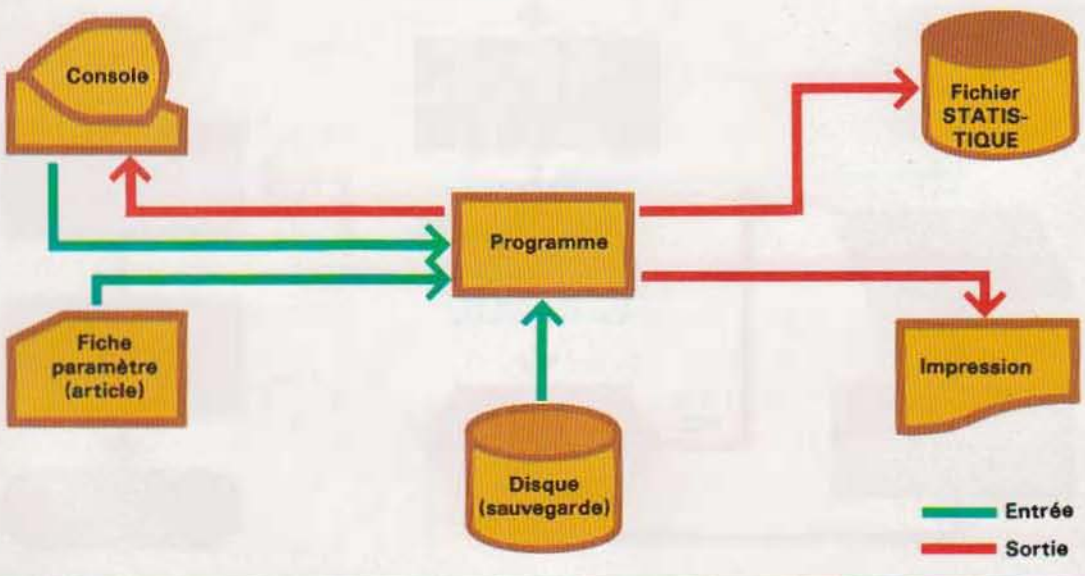
Verbes arithmétiques

Ils permettent de traiter les données rangées en mémoire pour le calcul de certaines formules mettant en jeu ces mêmes données.
Un langage de programmation doit être apte à traiter sans difficulté les informations provenant du type d'application pour lequel il a été conçu. S'agissant du Cobol, qui comporte un ensemble limité d'opérations mathématiques, ce critère est totalement pris en compte. Il répond parfaitement aux exigences de la gestion commerciale.
Avant d'examiner de près les instructions arithmétiques, il est indispensable de connaître la représentation des données numériques dans la mémoire. Nous décrirons ensuite certaines clauses qui permettent de choisir un type de représentation plutôt qu'un autre, de manière à structurer au mieux le traitement des données numériques.

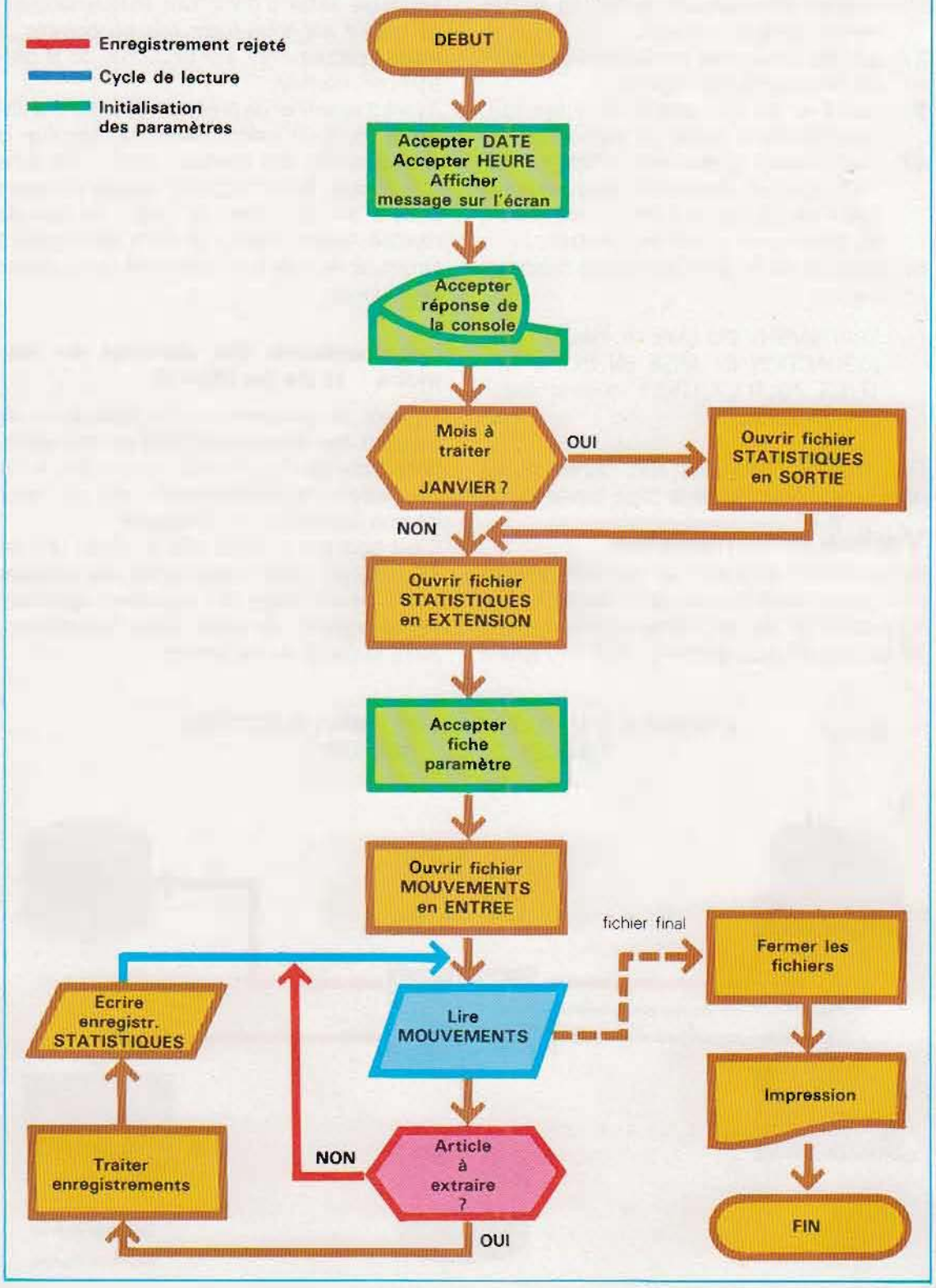
Représentation des données en mémoire : la clause USAGE

Pendant le déroulement des opérations de transfert des données (MOVE) ou des opérations arithmétiques, l'ordinateur procède à des conversions de représentation, afin de traiter les données de façon homogène.
C'est pourquoi, adopter dès le départ une représentation interne appropriée des données décharge le système des opérations répétitives de conversion, réduisant ainsi considérablement la durée du traitement.

EXEMPLE D'UTILISATION DES INSTRUCTIONS "ACCEPT" ET "DISPLAY"



EXEMPLE D'UTILISATION DES INSTRUCTIONS "ACCEPT" ET "DISPLAY"



En ANS COBOL, deux modes de représentation ont été prévus :

DISPLAY (affichage)
(codification hexadécimale)
COMPUTATIONAL (calcul)
(codification binaire)

Pour définir une donnée codée binaire, on peut écrire indifféremment :

```
01 CHAMP    PIC S9 (5)
            USAGE IS COMPUTATIONAL.
```

ou encore

```
01 CHAMP    PIC S9 (5) COMP.
```

Si la clause USAGE (dans sa forme entière ou abrégée) est omise, le compilateur opte par défaut pour une représentation type DISPLAY. Pour lui, les 3 descriptions :

```
01 CHAMP    PIC S9 (5)
01 CHAMP    PIC S9 (5)
            USAGE IS DISPLAY.
01 CHAMP    PIC S9 (5) DISPLAY.
```

sont totalement équivalentes.

USAGE DISPLAY. Dans ce mode, chaque chiffre occupe un caractère en mémoire.
Ainsi, la clause USAGE n'ayant pas été spécifiée dans la description :

```
01 CHAMP    PIC 9 (4) VALUE 6.
```

le compilateur choisit DISPLAY.
Le champ ayant été initialisé à 6, sa représentation interne est 0006.

USAGE COMPUTATIONAL. Cette clause (COMP) indique au compilateur que le contenu du champ examiné doit être codé en binaire. Le recours à ce code n'oblige pas le compilateur à effectuer des opérations de conversion puisqu'il reconnaît ce mode de représentation. La clause USAGE COMP réduit les temps de traitement et occupe un emplacement mémoire restreint. En effet, alors qu'un champ avec USAGE DISPLAY prend autant d'octets qu'il y a de caractères dans la PICTURE, avec la clause USAGE COMP, les mêmes nombres sont représentés mais occupent un espace mémoire inférieur.
Considérons la définition d'un champ devant contenir jusqu'à 10 chiffres.
Si sa description était

```
01 CHAMP    PIC S9 (10).
```

le compilateur réserverait 10 octets en mémoire.
Ce même champ décrit autrement :

```
01 CHAMP    PIC S9 (10)
            USAGE COMPUTATIONAL.
```

ne prendrait que 8 octets.
Cet avantage est plus parlant pour des champs d'une longueur supérieure à 10 chiffres. Dans les limites maximales autorisées par le Cobol (18 chiffres),

```
01 CHAMP    PIC S9 (18) COMP.
```

l'espace mémoire occupé sera toujours égal à 8 octets.
Le tableau des équivalences ci-dessous donne l'occupation effective en mémoire pour différents champs COMPUTATIONAL affichés dans la clause PICTURE

Nombre de chiffres (9) dans la PICTURE	Espace mémoire
de 1 à 4	2 octets
de 5 à 9	4 octets
de 10 à 18	8 octets

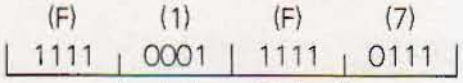
Pour bien saisir la différence entre les deux représentations analysées ici, considérons deux champs, CHAMP-1 et CHAMP-2, déclarés respectivement DISPLAY et COMPUTATIONAL :

```
01 CHAMP-1    PIC S9(2).
01 CHAMP-2    PIC S9(2) COMP.
```

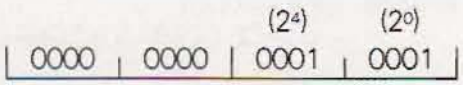
L'un et l'autre occupent 2 octets chacun : CHAMP-1 parce que la clause PICTURE le déclare expressément et CHAMP-2 d'après le tableau des équivalences.

Supposons que les deux champs contiennent le nombre 17, leur représentation s'écrit :

CHAMP-1



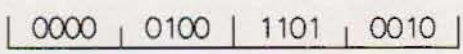
CHAMP-2



Comme on peut le constater, le contenu de CHAMP-2 est le codage binaire normal de 17 sur un mot de 16 bits.
Pour mémoire, le premier bit à gauche du champ est réservé au signe. Si ce bit est mis à 0, le nombre est positif (les nombres négatifs étant calculés par complémentation à 2). Dans le champ :

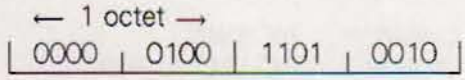
01 CHAMP-3 PIC S9(4) COMP.

le contenu vaudrait - 1234 à un certain moment. Si ce nombre avait été positif, sa représentation interne aurait été

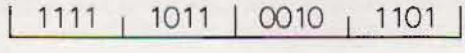


Le nombre - 1234 devant être rangé en mémoire, la machine se charge de représenter le même nombre que celui de la configuration précédente, comme complément à 2, en procédant selon différentes étapes :

1/ Représentation du nombre (+ 1234)

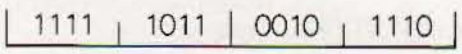


2/ Complément à 1



3/ Addition de 1

4/ Représentation du nombre (- 1234)



De nombreux compilateurs comportent d'autres modes de représentation numérique qui ne sont pas composés dans le Cobol standard. Ces modes sont affichés selon les formats suivants de la clause USAGE :

```
USAGE IS COMPUTATIONAL-3.
USAGE IS COMPUTATIONAL-1.
USAGE IS COMPUTATIONAL-2.
```

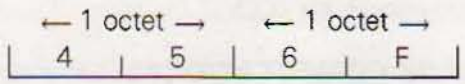
formats équivalents à :

```
COMP-3.
COMP-1.
COMP-2.
```

USAGE COMP-3. Cette représentation, dite aussi **packed** (abrégée), réserve un octet pour tous les deux chiffres du champ décrit, à l'exception du dernier octet à droite où un seul chiffre est mémorisé, plus l'emplacement du signe.
Considérons le champ :

01 CHAMP-4 PIC 09(3) COMP-3.

dont la valeur serait, à un moment donné, 456. Etant donné qu'avec USAGE COMP-3 chaque chiffre occupe un demi-octet (le dernier étant réservé au signe), on aura la représentation interne suivante :

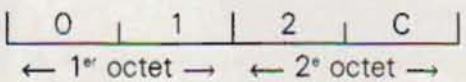


où F tient compte du fait que CHAMP-4 ne prévoit pas le signe (le symbole S manquant dans la PICTURE).
Observons que l'assignation en mémoire des octets nécessaires à la PICTURE est telle qu'elle occupe toujours un nombre entier d'octets. Autrement dit, un champ de deux chiffres avec le signe [(PIC S9(2)] nécessiterait l'utilisation de 3 demi-octets : 2 pour les chiffres et 1 pour

le signe. Dans ce cas, le compilateur assigne le 4e demi-octet, en remplissant la partie gauche non utilisée avec un 0.
Soit le champ :

```
01 CHAMP-5      PIC S9(2) COMP-3.
```

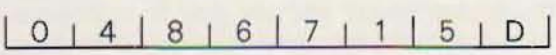
et supposons qu'à un certain moment CHAMP-5 contienne la valeur (+12). Sa représentation interne sera :



C indique que la valeur contenue dans le champ est positive.
Pour compléter la description de la clause COMP-3, considérons le champ :

```
01 CHAMP-6      PIC S9(4)V9(2)COMP-3.
```

L'occupation mémoire est de 6 demi-octets (soit le nombre de chiffres déclarés par le PICTURE) plus un demi-octet pour le signe, plus un demi-octet imposé par le compilateur pour rendre entier le nombre d'octets ; soit au total, 4 octets.
Prenons un exemple. Si une des valeurs temporaires de CHAMP-6 est (-4867,15), sa représentation sera :



D indiquant le signe négatif.

USAGE COMP-1 et **COMP-2.** Ces deux modes, dits aussi floating point (virgule flottante), sont rarement employés dans le cadre du Cobol, sauf lorsqu'on doit manipuler de très grands nombres, avec toutefois une précision relative.
Grâce à ces représentations internes, on peut traiter des nombres allant de 10^{-78} à 10^{+75}, avec un degré de précision allant jusqu'à la 6e décimale pour le COMP-1, et la 16e décimale pour le COMP-2.
Ceci explique que l'usage de tels nombres soit limité en informatique de gestion, terrain privilégié du Cobol.

Le format de l'USAGE COMP-1 ou COMP-2 est différent de celui des représentations étudiées précédemment. En effet, si un champ est déclaré en virgule flottante, le compilateur lui assigne automatiquement un mot (pour COMP-1) ou deux mots (pour COMP-2) en mémoire.
Par conséquent, un champ déclaré en virgule flottante et en simple précision est décrit par :

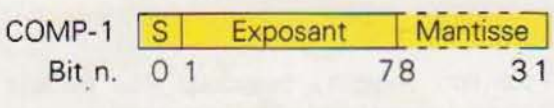
```
01 CHAMP-7      COMP-1.
```

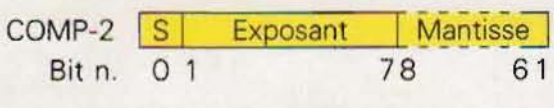
Cela donne, en double précision :

```
01 CHAMP-8      COMP-2.
```

Le codage interne d'un nombre en virgule flottante comporte :
- la **mantisse** ou partie entière pour les chiffres significatifs du nombre,
- l'**exposant,** partie qui détermine la valeur réelle du nombre.

Les ordinateurs sont généralement dotés d'un circuit spécial capable d'effectuer les opérations de « normalisation ». Nous ne mentionnerons donc que les deux modes de présentations COMP-1 et COMP-2 pour un 32 bits ou moins :





Une bonne connaissance des représentations numériques internes constitue une base solide pour la mise en place d'un programme rapide et fiable.
N'oublions pas que le Cobol permet d'utiliser dans le cadre d'une même opération des champs numériques ayant des représentations différentes. Cela est rendu possible par des systèmes de conversion directement gérés par le compilateur. Ils sont donc liés à une hiérarchie qui établit un ordre décroissant de priorité :



Litton PRC

Le microprocesseur se retrouve même dans une caisse enregistreuse.

COMP-2
COMP-1
COMP-3
DISPLAY
COMP

Tout ceci semblera beaucoup plus évident quand on parlera des verbes de type arithématique.

Les expressions arithmétiques

En arithématique, de telles formules permettent d'effectuer une série d'opérations généralement complexes sur des opérandes ou sur des constantes ; le résultat peut être associé à une variable.

$$A = (B + C)^2 + \frac{D}{(E + F)} - (G + H)^P + 100$$

constitue une expression où la valeur de la variable A (à gauche du signe égal) est équivalente au résultat du traitement de la seconde partie dont les valeurs sont, par définition, différentes de 0. En Cobol, le concept d'expression arithmétique est à peu près identique. Le programmeur qui désire calculer la valeur d'une expression dispose des 5 opérations mathématiques fondamentales de l'ordinateur plus les parenthèses, avec lesquelles il peut modifier l'ordre d'exécution des opérations

\+ addition
— soustraction
* multiplication
/ division
** élévation à une puissance.

A l'aide de ces signes, l'expression précédente sera réécrite en Cobol :

```
A = (B+C)**2+D/(E+F) — (G+H) ** P+100
```

Le calcul Cobol respecte toutes les règles de

l'arithmétique, à savoir l'ordre d'exécution des opérations : résolution des élévations à une puissance, ensuite multiplications et divisions et enfin additions et soustractions.
Prenons un exemple : le calcul de l'expression

```
A = 7 + 5*4 - 6**2
```

passe par les étapes suivantes :

```
A = 7 + 5 * 4 + 36
A = 7 + 20 + 36
A = 63
```

Les opérations de même niveau de priorité sont résolues en fonction de leur ordre d'arrivée dans l'expression, de gauche à droite s'il s'agit de + - * /, de droite à gauche pour les élévations à une puissance (**).
Dans l'expression :

```
A = B*C/D + E - F + G **2 + H **3
```

Le déroulement des opérations est le suivant :

1/ H ** 3
2/ G ** 2
3/ B * C
4/ (résultat de B * C) /D
5/ (résultat de B *C /D) + E
6/ (résultat de B * C/D + E) - F
7/ ...

Le programmeur dispose de 9 niveaux de parenthèses pour modifier l'ordre des calculs. De cette façon, il peut regrouper de n'importe quelle façon les données à traiter. Exemple :

```
A = (( B + (C*D)) ** 2 + E/ (F-G) ** 3
```

Les groupements de premier niveau (ici C*D et F-G) sont d'abord traités, en respectant à l'intérieur de ceux-ci les règles générales déjà décrites. Ainsi, dans ce même niveau, on calculera (C*D) avant (F-G).

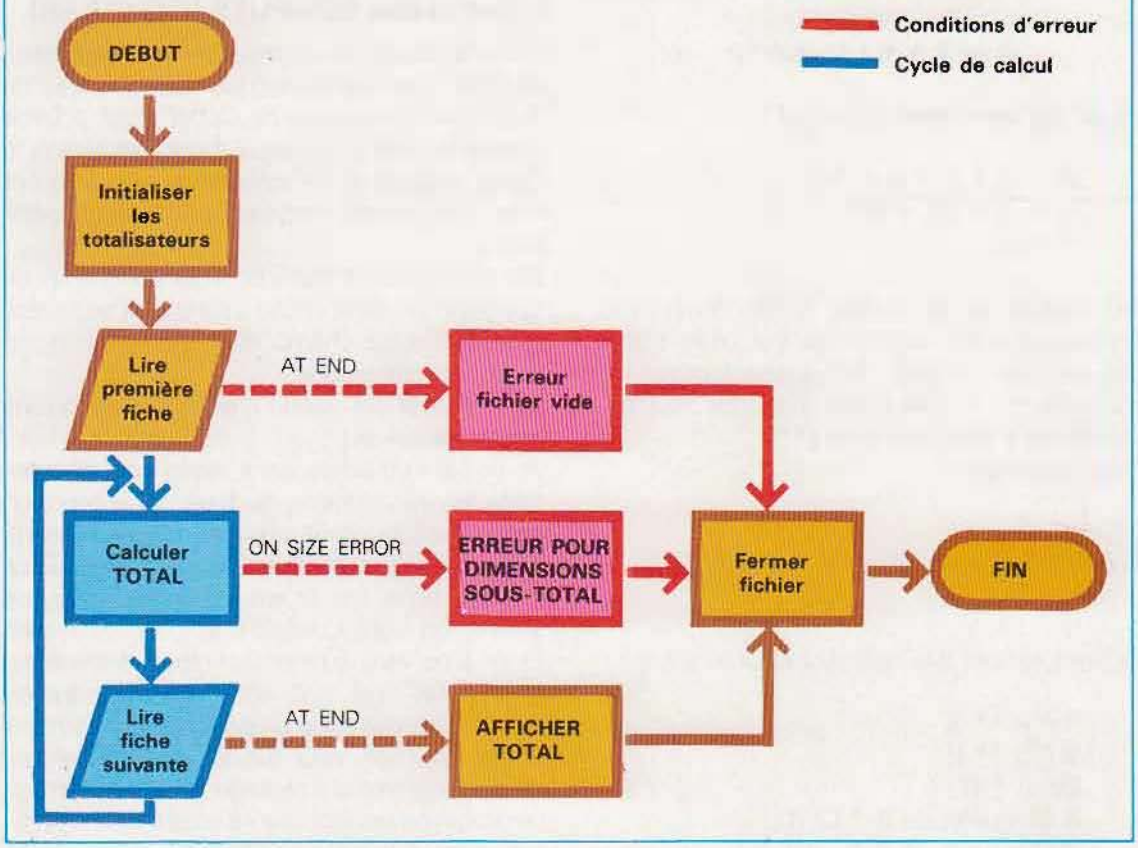
L'instruction COMPUTE (CALCULER)

Jusqu'à présent, les expressions ont été écrites dans leur forme arithmétique classique, tout en respectant la syntaxe du Cobol. Pour pouvoir obtenir le résultat du calcul d'une expression, le Cobol dispose d'une instruction prévue à cet effet (voir format ci-dessous en bas de cette page).
Elle permet de transférer, vers un champ de données, la valeur d'une expression arithmétique d'un autre champ de données ou d'une constante numérique.
Pour illustrer son mode d'emploi, considérons l'exemple suivant :
on doit lire un fichier dans lequel sont enregistrées les informations du type : quantité d'un certain article, coût unitaire, rabais consenti (pourcentage), frais d'expédition à l'unité. Pour chaque fiche lue (c'est-à-dire pour chaque achat), on veut connaître le coût du lot de façon à parvenir au montant global des achats. Le résultat final doit également prendre en compte les résultats précédents, directement entrés par l'opérateur sur console à la demande du programme (message affiché à l'écran). Le problème est représenté sous forme d'organigramme à la page suivante. On a également reproduit le flux des données et leur enregistrement sur une carte perforée (page 1017), ainsi que le programme correspondant (pages 1018 et 1019). Dans cet exemple, il faut remarquer l'insertion d'un compteur de fiches lues (COMPTE-FICHES), qui n'est pas explicitement requis par les spécifications Cobol. En effet, il est bon de toujours prévoir un compteur d'enregistrement (lus ou écrits) de manière à avoir une indication sur les données mouvementées par le programme.

FORMAT GENERAL DE L'INSTRUCTION

```
COMPUTE nom-de-donnée [ROUNDED] { expression arithmétique
                                  nom-de-donnée-1
                                  constante numérique }
    [ON SIZE ERROR phrase impérative]
```

EXEMPLE D'APPLICATION DE L'INSTRUCTION "COMPUTE"



Dans cet exemple, on a examiné tous les formats et toutes les clauses de l'instruction COMPUTE.

COMPUTE COMPTE-FICHES = 0

met à 0 (initialise) le compteur de fiches lues ;

COMPUTE TOTAL = TOTAL-PRECEDENT

range, dans le champ TOTAL, la valeur du champ TOTAL-PRECEDENT
(attention : le signe - est un séparateur inséré dans le nom du champ).

Les expressions :

```
COMPUTE ACHAT-NET ROUNDED =
      (QUANTITE * COUT-UNITAIRE) -
      (QUANTITE * COUT-UNITAIRE)*
      (RABAIS/100)+
      (QUANTITE * FRAIS-UNITAIRES)
```

calculent la valeur de l'expression à droite du signe et les rangent dans le champ ACHAT-NET en respectant l'alignement par rapport à la décimale dans la position prévue par la PICTURE.
Avec ce format, la valeur calculée n'est pas simplement tronquée à la 3e décimale ; si elle est supérieure à 5, elle est arrondie à la décimale immédiatement supérieure. En d'autres termes, si la valeur de l'expression était :

785 642, 3896 (notation européenne),

on lirait, dans ACHAT-NET : 785 642, 390 grâce à la clause ROUNDED. Sans elle, le contenu du champ aurait été :

785 642, 389

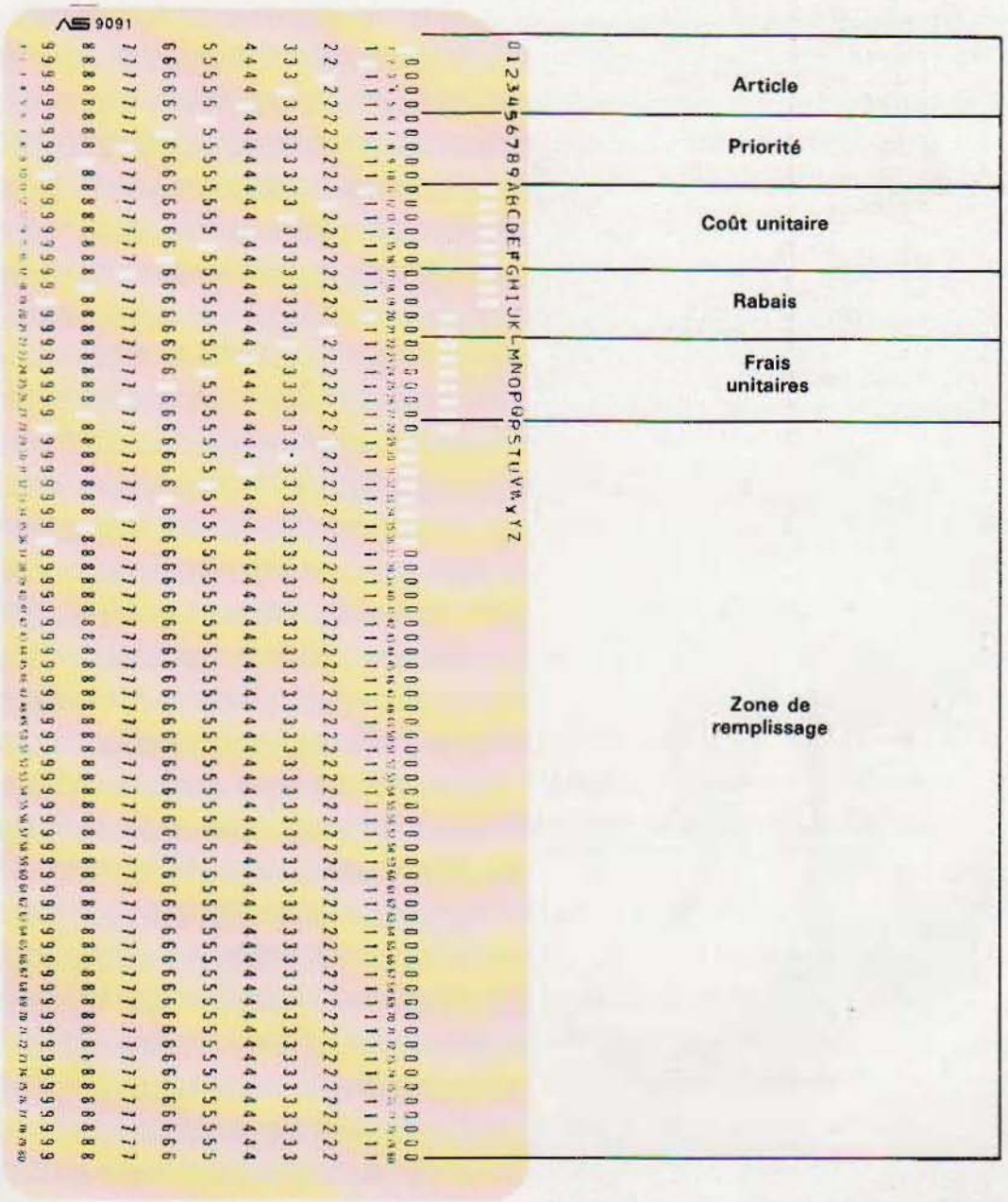
(toujours en notation européenne).

Après avoir atteint le nombre maximum qu'il peut représenter (quelle qu'en soit la base), un compteur se remet automatiquement à 0.
On dit alors qu'il « sature », tel un compteur

FLUX DES DONNEES



LECTURE D'UNE CARTE



EXEMPLE D'APPLICATION DE COMPUTE

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CALCUL.
       REMARKS. CE PROGRAMME EST UN EXEMPLE
                D'APPLICATION DE L'INSTRUCTION
                COMPUTE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. ..............
       OBJECT-COMPUTER. ..............
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FICHES ASSIGN TO CARD-READER FICHES.
       DATA DIVISION.
       FILE SECTION.
       FD  FICHES
           LABEL RECORD OMITTED.
       01  FICHE                          PIC X(80).
      *
      *
      *
       WORKING-STORAGE SECTION.
       01  FICHE-WS.
           05  ARTICLE                    PIC X(5).
           05  QUANTITE                   PIC X9(5).
           05  COUT UNITAIRE              PIC S9(3)V9(3)  COMP.
           05  RABAIS                     PIC S9(2)V9(3)  COMP.
           05  FRAIS UNITAIRES            PIC S9(3)V9(3)  COMP.
           05  ZONE DE REMPLISSAGE        PIC X(53).
      *
       01  TOTAL PRECEDENT                PIC S9(15)      COMP.
      *
       01  TOTAL                          PIC S9(15)      COMP.
      *
       01  NET-ACHAT                      PIC S9(8)V9(3)  COMP.
      *
       01  SOUS-TOTAL                     PIC S9(15)V9(3) COMP.
      *
       01  COMPTE-FICHES                  PIC S9(5)       COMP.
      *
      *
      *
      *
      *
       PROCEDURE DIVISION.
       MAIN SECTION.
       INITIALISER.
           COMPUTE COMPTE-FICHES = 0
           COMPUTE SOUS-TOTAL = 0
           DISPLAY ' = CONVERTIR LE TOTAL PRECEDENT  :
                   UPON CONSOLE.
           ACCEPT TOTAL-PRECEDENT FROM CONSOLE.
           COMPUTE TOTAL = TOTAL-PRECEDENT.
      *
      *
      *
       OUVRIR-FICHIER.
            OPEN INPUT FICHES.
       PREMIERE-LECTURE.
            READ FICHES INTO FICHES-WS
                 AT END
                 DISPLAY '** FICHIER VIDE **'
                         UPON CONSOLE
                 PERFORM FERMETURE
       CALCUL.
            COMPUTE COMPTE-FICHES = COMPTE-FICHES+1.
            COMPUTE NET-ACHAT ROUNDED = (QUANTITE * COUT-UNITAIRE)-
                                        ((QUANTITE * COUT-UNITAIRE) *
```

```
                                        RABAIS / 100) +
                                       (QUANTITE * FRAIS-UNITAIRES).
        COMPUTE SOUS-TOTAL = SOUS-TOTAL + NET-ACHAT
                             ON SIZE ERROR
                             DISPLAY '* CHAMP SOUS-TOTAL INSUFFISANT *
                         'AVERTIR LE RESPONSABLE *'
                                    UPON CONSOLE
                             PERFORM FERMETURE.
    AUTRES-LECTURES.
        READ FICHES INTO FICHES-WS
                    AT END
                    PERFORM FERMETURE.
        GO TO CALCUL.
  *
  *
  *
  *
  *
    FERMER-SECTION.
    FERMER-FICHIER.
        CLOSE FICHES.
        DISPLAY '** FICHES LUES = ' COMPTE-FICHES
                UPON CONSOLE.
        COMPUTE TOTAL ROUNDED = TOTAL + SOUS-TOTAL.
        DISPLAY ' '
                UPON CONSOLE.
        DISPLAY '** TOTAL ACTUEL = ' TOTAL
                UPON CONSOLE.
        STOP RUN.

  *
```

tel un compteur kilométrique d'automobile. En général, ce type de compteur est prévu pour 5 chiffres ; il est donc en mesure de compter jusqu'à 99 999 kilomètres ; le 100 000e kilomètre sature le compteur qui se remet à 00000. Il en est de même des registres d'ordinateurs comme des champs décrits dans un programme Cobol.

Toujours dans le but d'assurer l'intégrité des informations contenues dans un compteur, le compilateur applique la clause ON SIZE ERROR à toutes les instructions arithmétiques. Grâce à elle, le programme est en mesure de signaler la saturation d'un compteur au moment où son contenu est incrémenté. S'agissant d'une condition anormale, de nature à fausser les résultats, une phrase impérative est associée à cet événement, permettant au programmeur de bien le gérer.

Dans l'exemple cité, la saturation du compteur SUBTOTAL (sous-total) entraîne un blocage de la situation ; il est donc nécessaire que le système signale cette anomalie à l'écran avant de procéder à l'interruption du traitement.

De l'exemple décrit, on peut tirer des règles générales valables aussi bien pour COMPUTE que pour toutes les instructions arithmétiques qui seront analysées par la suite.

1/ Tous les noms utilisés dans les expressions numériques doivent représenter des données numériques définies dans la division données (DATA DIVISION).

2/ Toutes les constantes utilisées dans les instructions de type arithmétique doivent être numériques.

3/ La longueur prévue pour chaque opérande d'une expression arithmétique est de 18 caractères au maximum.

4/ Les PICTURE et la clause USAGE des différentes opérandes d'une expression arithmétique ne doivent pas nécessairement être uniformes, dans la mesure où, au cours des opérations, aussi bien l'alignement sur le point décimal.(V) que la conversion du format de chaque opérande sont automatiquement réalisés. Cette conversion a lieu selon un ordre hiérarchique précis (voir page 1014).

Dans les champs :

```
01 CHAMP-1      PIC S9(8)V9(37).
01 CHAMP-2      COMP-1.
01 CHAMP-3      COMP-2.
```

l'instruction :

```
COMPUTE CHAMP-3 =
            CHAMP-1 * CHAMP-2
```

déclenche les opérations suivantes :

- CHAMP-1 est transformé en COMP-1 par DISPLAY (en fait COMP-1 est la représentation la plus prioritaire),
- calcul de CHAMP-1 * CHAMP-2 en codage COMP-1,
- conversion du résultat précédent de COMP-1 en COMP-1 en COMP-2.

5/ Le format des données utilisées dans une expression arithmétique ne contient aucun symbole associé à l'impression.

Le verbe COMPUTE peut lancer les 5 opérations sur une série d'opérandes groupées dans une expression unique. Les verbes qui seront ultérieurement analysés n'autorisent, en revanche, qu'un seul type d'opération sur les données décrites.

Le verbe ADD

Cette instruction additionne, selon troix modalités, deux ou plusieurs nombres et range le résultat dans un champ d'arrivée.
Malgré sa complexité apparente (voir ci-dessous) le premier format est très proche de la structure classique de l'addition. Considérons en effet les champs suivants :

```
01 CUMULATEUR-1  PIC S9(4)V9(2) COMP.
01 CUMULATEUR-2  PIC S9(4)V9(2) COMP.
01 CUMULATEUR-3  PIC S9(4)V9(2) COMP.
```

On veut additionner leur contenu, ainsi qu'une constante fixe, égale à 3 000, au contenu d'un champ somme défini par

```
01 SOMME      PIC S9(4)COMP.
```

L'opération demandée peut être ainsi codifiée :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000                    TO SOMME.
```

Les descriptions des différents cumulateurs (seconds termes de l'addition), comportent 2 décimales qui seront tronquées dans la dernière phase de l'addition dans le champ-SOMME (celui-ci ne prévoit pas de décimales). Il suffit, pour obtenir un résultat arrondi et non tronqué, de coder de la manière suivante :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000              TO SOMME ROUNDED
```

Comme le champ SOMME prévoit par ailleurs le même nombre de caractères que les cumulateurs, il faut s'attendre à une saturation rapide dans son contenu, avec perte consécutive de l'intégrité de la donnée.
La logique de l'instruction COMPUTE permet aussi de contrôler le dépassement de capacité (overflow) de SOMME : grâce à la clause ON SIZE ERROR et à une phrase impérative indiquant le type d'opération à entreprendre dans une telle éventualité.

Exemple :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000      TO SOMME ROUNDED
              ON SIZE ERROR
              DISPLAY « CHAMP SOMME
                  EN DEPASSEMENT DE
                  CAPACITE »
                  UPON CONSOLE
              GO TO FINE.
```

PREMIER FORMAT DE L'INSTRUCTION ADD



TO nom-de-donnée-n [ROUNDED] nom-de-donnée-p [ROUNDED]
[ON SIZE ERROR phrase impérative].

Précisons que le champ d'arrivée, SOMME, doit être un champ numérique ne contenant pas de symboles d'impression dans la PICTURE, à moins d'adopter le 2e format de l'instruction ADD.
Dans ce dernier cas, une fois la somme calculée, le résultat est transféré selon des modalités précises dans le champ d'arrivée nom-de donnée-n.
Ce champ est réservé aux opérations de transfert et ignore les calculs. Il peut donc contenir des symboles d'impression.
Le dernier format de l'instruction ADD exploite la possibilité qu'a le Cobol de décrire les sous-ensembles de deux champs différents à l'aide des mêmes noms.

Considérons les descriptions :

```
01 BLOC-1.
   05 MEMBRE-1.
      10 MEMBRE-11      PIC S9(3) COMP.
      10 MEMBRE-12      PIC S9(2) COMP.
   05 MEMBRE-2          PIC S9(4) COMP.
   05 MEMBRE-3          PIC S9(6) COMP.

01 BLOC-2.
   05 MEMBRE-1.
      10 MEMBRE-11      PIC S9(3) COMP.
      10 MEMBRE-12      PIC S9(2) COMP.
   05 MEMBRE-2          PIC S9(4) COMP.
   05 MEMBRE-3          PIC S9(6) COMP.
```

Pour additionner terme à terme les sous-champs de BLOC-1 et de BLOC-2 à l'aide du 1er format de l'instruction ADD, on doit écrire de la manière suivante :

```
ADD MEMBRE-11      OF BLOC-1
TO MEMBRE-11       OF BLOC-2.
ADD MEMBRE-12      OF BLOC-1
TO MEMBRE-12       OF BLOC-2.
ADD MEMBRE-2       OF BLOC-1
TO MEMBRE-2        OF BLOC-2.
ADD MEMBRE-3       OF BLOC-1
TO MEMBRE-3        OF BLOC-2.
```

En revanche, si on opte pour le 3e format, on obtient le même résultat en écrivant seulement :

```
ADD CORRESPONDING BLOC-1
TO BLOC-2.
```

Malgré son avantage dans ce cas particulier, il vaut mieux éviter de définir des champs avec des noms de sous-ensembles semblables.
Dans un souci de clarté, il est, de loin, préférable d'affecter des noms différents aux champs. On saisit mieux et plus facilement le déroulement de la fonction logique (d'après son nom), de même que ses anomalies en cours de traitement, grâce au message CROSS REFERENCE envoyé par le compilateur.

DEUXIEME FORMAT DE L'INSTRUCTION ADD

```
ADD { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
    { constante-1     }   { constante-2     }

GIVING nom-de-donnée-n [ROUNDED] [ON SIZE ERROR phrase impérative].
```

TROISIEME FORMAT DE L'INSTRUCTION ADD

```
ADD { CORRESPONDING } nom-de-donnée-1 TO nom-de-donnée-2
    { CORR          }

[ROUNDED] [ON SIZE ERROR phrase-impérative]
```

Le verbe SUBTRACT

Très proche de l'addition, cette instruction comporte également 3 formats. Elle permet de soustraire, du contenu d'un champ, celui d'un ou de plusieurs autres. Les clauses respectent les mêmes règles déjà décrites dans ADD.
Pour illustrer le premier format, considérons les champs :

```
01 VALEUR-INITIALE-1      PIC S9(4) COMP.
   VALUE 3500.
01 VALEUR INITIALE-2       PIC S9(4) COMP.
   VALUE 1000.
01 VALEUR-INITIALE-3       PIC S9(4) COMP.
   VALUE 0.
01 EN SOUSTRAYANT-1        PIC S9(3) COMP.
   VALUE 630.
01 EN SOUSTRAYANT-2        PIC S9(3) COMP.
   VALUE 280.
01 EN SOUSTRAYANT-3        PIC S9(3) COMP.
   VALUE 2.
```

L'instruction :

```
SUBTRACT 180
         SOUSTRAIRE-1
         SOUSTRAIRE-2
         SOUSTRAIRE-3

    FROM VALEUR-INITIALE-1
         VALEUR-INITIALE-2.
```

effectue les sommes de 180, procède à SOUSTRAIRE-1, SOUSTRAIRE-2, SOUSTRAIRE-3 et enfin soustrait le résultat du contenu des champs VALEUR-INITIALE-1 et VALEUR-INITIALE-2.
Sur la base des valeurs données plus haut, le résultat de l'opération sera :

```
VALEUR-INITIALE-1 =  2408
VALEUR-INITIALE-2 =   -92
```

Si l'on souhaite ne pas modifier le contenu du champ dans lequel les soustractions sont effectuées et obtenir le résultat dans un autre champ, il faut adopter le 2e format.

PREMIER FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
         { constante-1     }   [ { constante-2     } ]

    FROM nom-de-donnée-m [ROUNDED]
         nom-de-donnée-n [ROUNDED]

    [ON SIZE ERROR phrase impérative]
```

DEUXIEME FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
         { constante-1     }   [ { constante-2     } ]

    FROM { nom-de-donnée-m }
         { constante-m     }

   GIVING nom-de-donnée-n [ROUNDED]

    [ON SIZE ERROR phrase-impérative]
```

En reprenant les champs de l'exemple précédent, on peut écrire :

```
SUBTRACT 180
         SOUSTRAIRE-1
         SOUSTRAIRE-2
         SOUSTRAIRE-3

    FROM  VALEUR INITIALE-1
    GIVING VALEUR INITIALE-3.
```

Le résultat de l'opération sera :

```
VALEUR INITIALE-1 = 3500
VALEUR INITIALE-3 = 2048
```

Le 3e format est, en tous points, équivalent à celui de ADD. Si on se réfère aux groupes de données BLOC-1 et BLOC-2 déjà décrits dans l'addition, l'instruction :

```
SUBTRACT CORR BLOC-1
FROM BLOC-2
```

équivaut aux instructions suivantes :

```
SUBTRACT MEMBRE-11 OF BLOC-1
FROM MEMBRE-11 OF BLOC-2.

SUBTRACT MEMBRE-12 OF BLOC-1
FROM MEMBRE-12 OF BLOC-2.

SUBTRACT MEMBRE-2 OF BLOC-1
FROM MEMBRE-2 OF BLOC-2.

SUBTRACT MEMBRE-3 OF BLOC-1
FROM MEMBRE-3 OF BLOC-2.
```

Prenons comme valeurs des différents sous-ensembles avant l'exécution de l'instruction :

	BLOC-1	BLOC-2
MEMBRE-11	210	500
MEMBRE-12	31	63
MEMBRE-2	4780	1200
MEMBRE-3	370	25612

Les résultats seraient :

	BLOC-1	BLOC-2
MEMBRE-11	210	290
MEMBRE-12	31	32
MEMBRE-2	4780	-3580
MEMBRE-3	370	25242

Le verbe MULTIPLY

Dans le format le plus usité (voir page suivante), le résultat de la multiplicande (nom-de-champ-1 ou constante-1) et le multiplicateur (nom-de-champ-2 ou constante-2) est rangé dans le champ non-de-donnée-3.

Le verbe DIVIDE

Le premier format de la division est décrit dans la page suivante. Il agit comme MULTIPLY. Le quotient de la division, entre le dividende (nom-de-donnée 1 ou constante-1) et le diviseur (nom-de-donnée-2 ou constante-2), est rangé dans un champ d'arrivée (nom-de-donnée-3). Ce quotient est calculé dans un registre spécial de l'unité centrale avec un nombre constant de chiffres décimaux. De ce fait, si le résultat demandé est rond, l'opération, destinée à arrondir le quotient, se produit après le calcul proprement dit (avec un nombre fixe de chiffres décimaux).
N'oublions pas que la division d'un nombre par zéro donne lieu à un dépassement de capacité,

TROISIEME FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { CORRESPONDING } nom-de-donnée-1 FROM nom-de-donnée-2
         { CORR          }

[ROUNDED] [ON SIZE ERROR] phrase impérative]
```

FORMAT DE L'INSTRUCTION MULTIPLY

```
MULTIPLY { nom-de-donnée-1 }      BY { nom-de-donnée-2 }
         { constante-1     }         { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
        [ON SIZE ERROR phrase-impérative]
```

FORMAT DE L'INSTRUCTION DIVIDE

```
DIVIDE { nom-de-donnée-1 }      BY { nom-de-donnée-2 }
       { constante-1     }         { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
        [ON SIZE ERROR phrase-impérative]
```

géré par le programmeur avec l'introduction de la clause ON SIZE ERROR.
Il est superflu de noter que le champ qui recueille le résultat devra être décrit avec exactitude.
Si le résultat de l'opération est un nombre décimal, alors que le champ d'arrivée est défini en entier, il y aura perte de toutes les décimales.

Ainsi :

DIVIDE 10 BY 4 GIVING QUOTIENT

donnera comme résultat :

QUOTIENT = 2,5

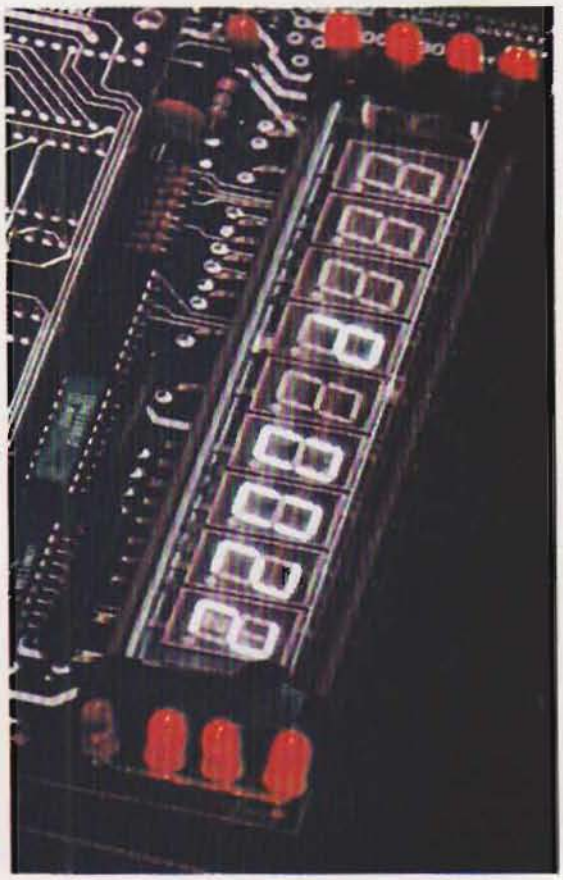
si le champ d'arrivée est décrit par

01 QUOTIENT PIC S9V9.

Le résultat sera 2 si le quotient est décrit par :

01 QUOTIENT PIC S9

Dans certaines applications, on peut trouver très utile d'avoir, pour résultat, les deux parties entière et décimale du quotient. Cela correspond au 2e format de DIVIDE.



Litton BEI

Affichage LED (Light Emission Diode) appliqué à un convertisseur analogique numérique.

DEUXIEME FORMAT DE L'INSTRUCTION DIVIDE

```
DIVIDE { nom-de-donnée-1 }  BY { nom-de-donnée-2 }
       { constante-1     }     { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
    REMAINDER nom-de-donnée 4
        [ON SIZE ERROR phrase-impérative]
```

A titre d'exemple, supposons que l'on veuille savoir si le nombre contenu dans le champ NOMBRE est pair ou impair. On sait que tous les nombres pairs sont divisibles par 2 avec un reste égal à 0, ce qui facilite l'opération de vérification :

```
DIVIDE NOMBRE BY 2
       GIVING QUOTIENT
       REMAINDER RESTE.

IF RESTE = 0
       DISPLAY « NOMBRE PAIR »
           UPON CONSOLE

ELSE
       DISPLAY « NOMBRE IMPAIR »
           UPON CONSOLE.
```

Verbes de transfert et de manipulation des données

Une des instructions les plus usitées en Cobol est MOVE. Elle procède au transfert des données d'un champ vers un autre parmi ceux décrits dans la division données du programme. Compte tenu de la signification explicite du verbe MOVE (déplacer), cette instruction a déjà été utilisée dans certaines exemples dans son format le plus simple.
Avant de l'examiner en détail, il convient de souligner que le compilateur détermine le type de transfert à effectuer d'après les caractéristiques du champ d'arrivée. En d'autres termes, selon qu'il est alphanumérique ou numérique, l'instruction MOVE sera également alphanumérique ou numérique. Il existe un 3e type de transfert dit **MOVE par état** destiné au transfert de données vers des champs dont les PICTURE renferment des caractères d'impression **(symboles d'édition)**. Ces derniers permettent d'insérer ou de masquer certains caractères du contenu du champ en fonction d'exigences esthétiques, fonctionnelles ou tout simplement de sécurité.
Considérons, par exemple, le cas où le champ TOTAL décrit par

```
01 TOTAL      PIC 9(5)V9(4).
```

doit être envoyé sur une ligne d'impression. Sachant que le caractère V ne représente qu'une position virtuelle de la virgule des décimales, si le champ en impression avait la même PICTURE, il ne serait pas en mesure d'établir la valeur réelle de TOTAL.
En effet, si cette valeur était 45 781,2478 (notation européenne) on aurait à l'impression :

```
457812478
```

En transférant TOTAL dans le champ TOTAL-IMPRESSION :

```
01 TOTAL-IMPRESSION      PIC 9(5).9(4).
```

on obtient la représentation réelle du nombre contenu dans TOTAL.
Le caractère . (point) est donc un caractère de préparation à l'impression. Notons au passage que, contrairement au symbole V, le point (la virgule pour nous) occupe une adresse mémoire, ce qui fait, pour TOTAL-IMPRESSION, 10 caractères.

Le verbe MOVE

L'instruction MOVE a deux formats valables tant pour les MOVE alphanumériques que pour

FORMAT DE L'INSTRUCTION MOVE

```
MOVE { nom-de-donnée-1 } TO nom-de-donnée-2 [nom de donnée-3] . . . . . .
     { constante       }
```

SECOND FORMAT DE L'INSTRUCTION MOVE

```
MOVE { CORRESPONDING } nom-de-donnée-1 TO nom-de-donnée-2.
     { CORR          }
```

les MOVE numériques ou par état. La nature de l'instruction est déterminée à partir de la description des champs d'arrivée.
D'une façon générale, MOVE transfère le contenu de nom-de-donnée-1 ou d'une constante vers un ou plusieurs champs d'arrivée.
Soit le champ :

```
01 DEPART      PIC X(20).
```

On désire transférer son contenu vers :

```
01 ARRIVEE-1   PIC X(20).
01 ARRIVEE-2   PIC X(20).
01 ARRIVEE-3   PIC X(20).
```

On a le choix entre 2 formulations équivalentes :

```
MOVE DEPART TO ARRIVEE-1.
MOVE DEPART TO ARRIVEE-2.
MOVE DEPART TO ARRIVEE-3.
```

ou bien

```
MOVE DEPART TO ARRIVEE-1
               ARRIVEE-2
               ARRIVEE-3.
```

La forme contractée est conseillée car elle rend plus évidente l'opération de MOVE simultanée du même champ vers plusieurs champs d'arrivée.

Dans le format :

```
MOVE constante TO  nom-de-donnée-2
                   nom-de-donnée-3
                   . . . . . .
```

« constante » doit être alphanumérique ou numérique selon le champ d'arrivée. Sur la base de ce qui a été dit à propos des MOVE multiples, il est évident que la compatibilité doit être respectée pour tout champ d'arrivée.

L'exemple suivant :

```
WORKING-STORAGE SECTION.
01 ARRIVEE-1                PIC X(20).
01 ARRIVEE-2.
   05 CHAMP-1               PIC 9(10).
   05 FILLER                PIC X(10).
PROCEDURE DIVISION.
DEBUT.
MOVE
'DEBUT TRAITEMENT' TO  ARRIVEE-1.
                       CHAMP-1.
```

est erroné, dans la mesure où la constante alphabétique DEBUT TRAITEMENT ne peut être déplacée dans un domaine numérique tel que CHAMP-1. En revanche il est correct de la déplacer vers ARRIVEE-1. Le transfert des données vers un ou plusieurs champs ne détruit pas le contenu du champ départ qui reste inchangé jusqu'à ce qu'on le modifie volontairement. En général, les champs d'arrivée et de départ peuvent être indifféremment élémentaires ou composés.
Rappelons à ce sujet qu'une donnée composée est prise dans son ensemble comme alphanumérique, indépendemment des descriptions des champs composants.

Analysons maintenant l'exemple suivant :

```
DATA DIVISION.
FILE SECTION.
FD FICHES
   LABEL RECORD OMITTED.
01 FICHE                          PIC X(3).
   05 TYPE-FICHE                  PIC X(3).
   05 CLASSEMENT NOMINATIF.
      10 PRENOM                   PIC X(10).
      10 NOM                      PIC X(15).
   05 DATE-DE-NAISSANCE.
      10 JOUR                     PIC 9(2).
      10 MOIS                     PIC 9(2).
      10 ANNEE                    PIC 9(2).
   05 SEXE                          PIC X.
   05 FILLER                     PIC X(45).
*
*
WORKING-STORAGE SECTION.
77 BAC-A-FICHE                    PIC X(80).
77 BAC-A-NOM                      PIC X(25).
01. . . . . .
   . . . . . .
*
*
PROCEDURE DIVISION
DEBUT.
   . . . . . .
   . . . . . .
*
*** EXEMPLES DE TRANSFERT
*
*
MOVE SPACES TO          TO BAC-A-FICHE.
MOVE FICHE              TO BAC-A-FICHE
MOVE BAC-A-NOM                TO NOM.
MOVE 'ABC'              TO FICHE-TYPE.
MOVE 83                       TO ANNEE.
   . . . . . .
   . . . . . .
```

Comme on peut le remarquer :

MOVE SPACES TO BAC-A-FICHE

déplace la constante figurative SPACES dans un champ élémentaire.

MOVE FICHE TO BAC-A-FICHE

déplace le champ composé FICHE dans un champ élémentaire.

MOVE BAC-A-NOM TO NOM

déplace un champ élémentaire dans un composé,

MOVE 'ABC' TO TYPE-FICHE

transfère la constante alphanumérique ABC dans un champ élémentaire.

MOVE 83 TO ANNEE

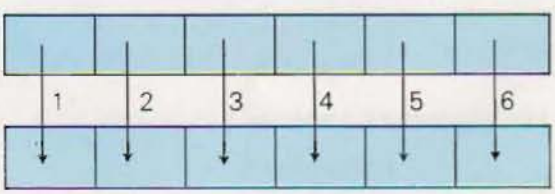
situe la constante numérique 83 dans le champ ANNEE.

Pour analyser en détail l'instruction MOVE, il est nécessaire d'apporter une précision importante quant aux constantes figuratives. Il s'agit de :

SPACE	(SPACES)
ZERO	(ZEROS ou ZEROES)
LOW-VALUE	(LOW-VALUES)
HIGH-VALUE	(HIGH-VALUES)
QUOTE	(QUOTES)
ALL	

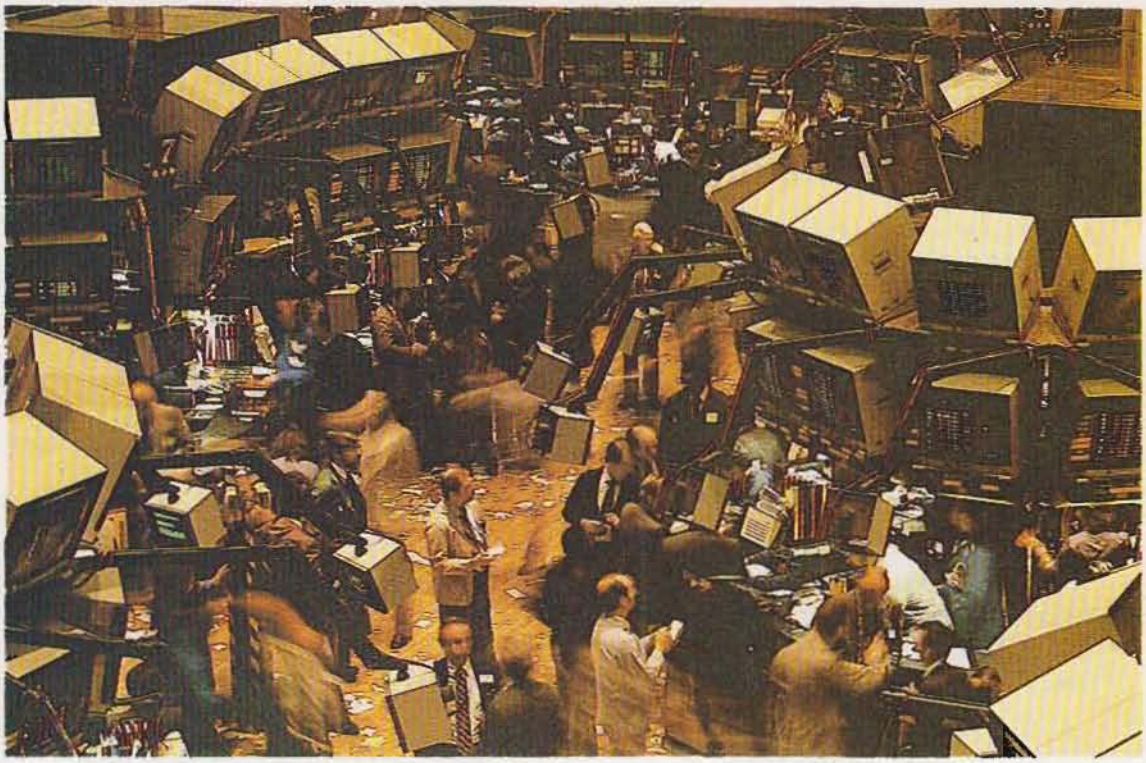
SPACE (ou SPACES) est considéré comme alphabétique, ZERO (ou ZEROS, OU ZEROES) comme numérique, alors que LOW-VALUE et HIGH-VALUE sont assimilés à des données alphanumérique.

MOVE alphanumérique. Il effectue le transfert des données dans la séquence indiquée par le schéma suivant :



Le champ d'arrivée est rempli de gauche à droite en copiant le contenu du champ de départ, caractère par caractère.

Si la longueur du champ départ est inférieure à celle de l'arrivée, il est comblé avec des blancs.



A la Bourse, les cambistes sont équipés de terminaux.

Par exemple :

```
01 DEPART      PIC X (3) VALUE 'ABC
01 ARRIVEE     PIC X (6) VALUE 'UVXYW'.
PROCEDURE DIVISION.
A.
      MOVE DEPART TO ARRIVEE.
```

Avant l'instruction MOVE, la situation était :

DEPART | A | B | C |

ARRIVEE | U | V | Z | X | Y | W |

Immédiatement après, on a :

DEPART | A | B | C |

ARRIVEE | A | B | C | | | |

Comme on peut le constater, le transfert de caractères a provoqué le remplissage du champ avec des blancs sur la droite.
Pour inverser cette modalité de transfert (c'est-à-dire remplir le champ ARRIVEE avec le contenu de DEPART et par conséquent être amené à combler à gauche avec des blancs) il faut insérer, dans la propre description du champ d'arrivée, la clause JUSTIFIED RIGHT (JUST).

Toujours en utilisant les champs de l'exemple précédent, on a :

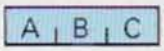
```
01 DEPART      PIC X (3) VALUE 'ABC'.
01 ARRIVEE     PIC X (6) JUST VALUE
               'UVZXYW'.
PROCEDURE DIVISION.
A.
      MOVE DEPART TO ARRIVEE.
```

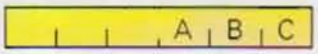
Situation avant MOVE

DEPART | A | B | C |

ARRIVEE | U | V | Z | X | Y | W |

Situation après MOVE :

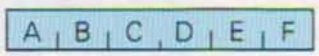
DEPART 

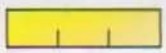
ARRIVEE 

Considérons maintenant le cas où le champ départ aurait une longueur supérieure à celle du champ d'arrivée :

```
01 DEPART     PIC X (6) VALUE 'ABCDEF'.
01 ARRIVEE    PIC X (3) VALUE SPACES.
```

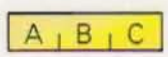
La situation précédant l'exécution de l'instruction MOVE DEPART TO ARRIVEE est alors :

DEPART 

ARRIVEE 

Situation après :

DEPART 

ARRIVEE 

Ainsi, tous les caractères en trop situés à droite du champ ARRIVEE ont été tronqués.

Inversement, si la définition du champ ARRIVEE avait été :

```
01 ARRIVEE PIC X (3) JUST VALUE SPACES.
```

La situation finale aurait été :

DEPART 

ARRIVEE 

En d'autres termes, la réduction des caractères en trop se serait produite à gauche, après le remplissage (à partir de la droite) de l'espace disponible. Le tableau ci-dessous présente une synthèse des opérations MOVE de type alphanumérique décrites jusqu'ici.
On y constate qu'un champ composé est en mesure d'accueillir des données en provenance de tout autre type de champ.
Exception à cette règle générale : les nombres en virgule flottante (USAGE COMP-1 et COMP-2). Cette exception n'apparaît pas dans le tableau ; il convient donc d'illustrer le cas à l'aide d'un exemple.

Considérons deux champs :

```
01 NOMBRE                    COMP-1.
01 CHAMP.
   05 SOUS-CHAMP-1           PIC 9 (4).
   05 SOUS-CHAMP-2           PIC X (3).
   05 FILLER                 PIC X.
```

CHAMP-DEPART	CHAMP-ARRIVEE			
Type	Composé	Elément alphanumérique	Elément alphabétique	Elément numérique
Composé	oui	oui	oui	NON
Elément alphanulérique	oui	oui	oui	NON
Elément alphabétique	oui	oui	oui	NON
Elément numérique	oui	oui*	NON	oui

*Seulement si le champ de départ est entier.

L'instruction

```
MOVE NOMBRE TO CHAMP
```

est erronée dans la mesure où le champ NOMBRE a un codage interne en virgule flottante en simple précision. On observera, en outre, que l'astérisque apparaissant sur le tableau indique que le transfert d'un champ élémentaire numérique vers un champ alphanumérique n'est possible que si celui de DEPART est entier.
L'instruction USAGE de ce champ peut être indifféremment COMP ou DISPLAY ; cependant, même alors il reste impossible d'utiliser un champ en virgule flottante comme départ.
Au cours du transfert d'un champ numérique vers un champ alphanumérique, la conversion se passe dans le champ ARRIVEE, selon les modalités décrites pour ce type de champ. L'exemple suivant permettra de mieux comprendre cette règle.
Considérons le champ :

```
01 ARRIVEE          PIC X (3).
```

et les instructions :

```
MOVE 38 TO ARRIVEE.
MOVE 7425 TO ARRIVEE.
```

Après le premier MOVE, le contenu d'arrivée sera :

3	8	

alors qu'après le deuxième MOVE, le champ aura la configuration suivante :

7	4	2

Remarquons que, si l'on souhaite vérifier, par exemple, que le champ ARRIVEE contient le nombre 38, cette opération devra être réalisée sur les trois caractères du champ.
En dépit de son apparence correcte, la séquence d'instructions :

```
MOVE 38 TO ARRIVEE.
IF ARRIVEE = '38'
   PERFORM EGAL-38.
```

n'aboutira pas à l'exécution de la SECTION



J. Pickerell/Marka

Test des plaques du circuit de commande pour disques souples.

EGAL-38. En effet, après le déroulement de l'instruction MOVE 38 TO ARRIVEE, le contenu du champ ARRIVEE est la configuration de caractères

3	8	b

et non pas 38 ; la forme correcte de l'exemple précédent est :

```
MOVE 38 TO ARRIVEE.
IF ARRIVEE = '38'
   PERFORM EGAL-38.
```

Notons par ailleurs que ce qui vient d'être exposé demeure valable quand le champ ARRIVEE est justifié à droite (JUST RIGHT) ; en d'autres termes, ' 38' n'est pas égal à '38'.

La clause ALL. Si on doit remplir tout un champ avec une série de caractères, il existe un format de l'instruction MOVE :

MOVE ALL caractères TO nom-champ.

Le diagnostic assisté par ordinateur (3)

L'échographie est une méthode de diagnostic fondée sur la production d'images des détails anatomiques au moyen de sondes spéciales qui émettent des ultrasons. Les échos émis par les différentes structures anatomiques sont traités par un microprocesseur et projetés sur un écran. L'image visualisée reproduit fidèlement les caractéristiques morphologiques et structurelles de la partie ou de l'organe examiné.

Les caractéristiques particulières qui différencient l'échographie des autres méthodes d'investigation la rendent pratiquement irremplaçable, surtout du fait de son inocuité absolue. Comme elle ne fait pas appel aux radiations ionisantes, on peut la répéter indéfiniment sans aucun risque. Elle fit son apparition aux alentours des années 70 mais ce n'est que vers la moitié de la dernière décennie que ce procédé se développa réellement en Europe et aux Etats-Unis, pour aboutir à nos installations actuelles. Celles-ci sont capables de traiter en ***temps réel*** *des images d'organes en mouvement. Autre avantage : son coût bien moindre en comparaison des autres systèmes. Enfin, ses dimensions réduites et sa facilité d'emploi le font largement apprécier.*

Les unités échographiques multidisciplinaires du type SDU 3000 sont aujourd'hui dotées d'une vaste gamme de transducteurs destinés à des examens statiques et dynamiques en pédiatrie, en ophtalmologie, en obstétrique, et à l'examen des organes profonds ou superficiels en général.

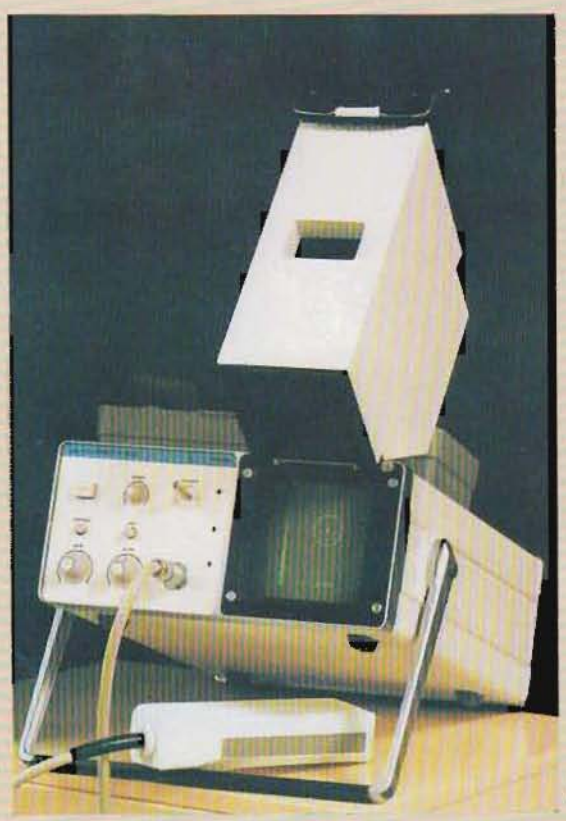
Il existe fondamentalement deux types de sondes pour l'investigation dynamique : la sonde linéaire à double focale (géométrique et électronique) pour des examens panoramiques de l'abdomen et pour l'obstétrique et la sonde à secteur qui permet d'éliminer les difficultés d'accès à certains organes.

Les systèmes multidisciplinaires de base sont dotés d'une vidéo incorporée, d'objectifs pour la production de documents imprimés, de sondes bioptiques et de perforateurs pour l'enregistrement des données sur papier. Le tout est intégré dans un petit meuble de la taille d'un réfrigérateur, sur roulettes, facile à transporter. Ces appareils présentent en outre des caractéristiques de résolution et de pénétration qui paraissaient utopiques au regard des imposantes installations que l'on construisait il y a moins d'une décennie. Une sonde sectorielle a une pénétration pouvant atteindre 30 cm et une résolutation angulaire d'à peine 12 pouces (soit 30 cm).

Malgré tous ces résultats déjà brillants, il ne faudrait pas croire que la ***miniaturisation*** *aurait déjà atteint ses limites. Le système Sono Diagnost R 1000 en est la démonstration la plus évidente : il pèse moins de 12 kg, caméra comprise (photo ci-dessous). L'écran de contrôle fait apparaître des images 10 × 8 cm produites par le traitement des sondages prélevés 25 fois à la seconde par une sonde linéaire qui émet un faisceau* ***d'ultrasons*** *à 2,2-3 MHz. Cette installation peut fonctionner en pleine campagne en se branchant sur une batterie de voiture. Leur prix et leur simplicité d'emploi la met à la portée de tous les cabinets médicaux.*

Les techniques de sondage aux rayons X et celles qui utilisent les ultrasons donnent une

Le Sono Diagnost R 1000, équipement portable pour échographies.





Philips



Philips

Appareil d'échographie transportable : le Sono Diagnost Universal 3000 (SDU 3000).

image morphologique des organes examinés se limitant à la reproduction, certes précise, d'un détail anatomique mais sans pour autant fournir la moindre information sur son état fonctionnel.

C'est justement pour répondre à cette exigence qu'est née la technique de diagnostic utilisant la radioactivité, c'est-à-dire les isotopes radioactifs de certains éléments naturellement présents dans le corps humain. (Attention ! Ceci n'a rien à voir avec la résonance magnétique nucléaire, la RMN, dont il a été question précédemment). Le but principal des ***isotopes radioactifs*** *est de permettre l'étude de la fonction physiologique des organes sans se borner à la production d'une image anatomique statique.*

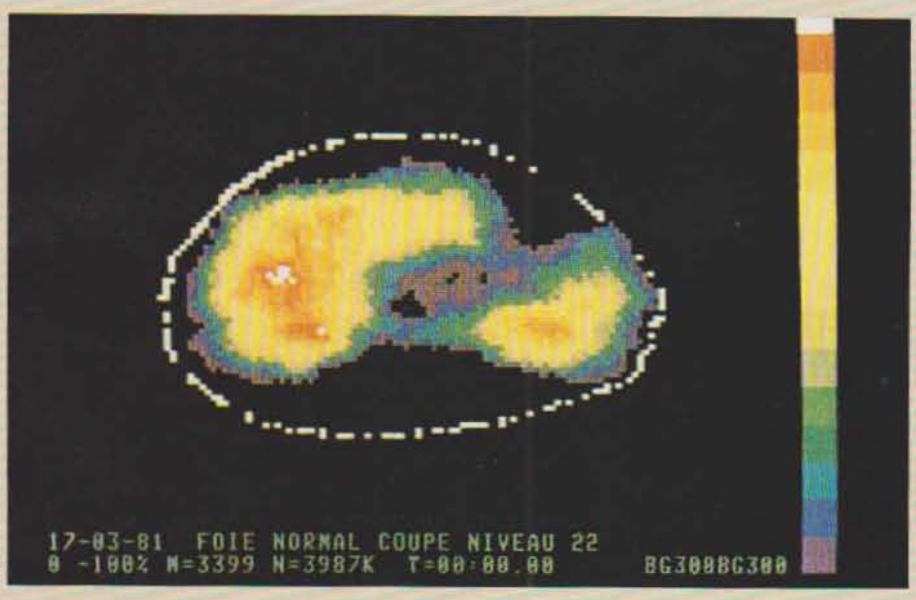
Les techniques utilisant les isotopes radioactifs nécessitent l'injection, dans l'organisme, de produits émettant un ***rayonnement gamma*** *(c'est le cas des produits radioactifs) et qui, de plus, sont métabolisés dans l'organe ou dans le système anatomique à examiner. On procède ensuite au relevé et à la mesure d'intensité des radiations, ce qui permet de suivre le processus de métabolisation et d'établir ainsi une carte des concentrations du produit traçant dans les différents organes.*

Dans ce secteur, les nouveautés doivent beaucoup aux possibilités offertes par ***l'automatisation*** *des appareils. Depuis vingt-cinq ans, l'instrument de base d'un laboratoire de médecine nucléaire est la gamma-caméra, mais c'est seulement au cours des cinq dernières années que l'introduction du microprocesseur, pour piloter l'exploration, a transformé cet instrument en un puissant système de diagnostic, grâce auquel on peut relever jusqu'aux images tomographiques axiales.*

Dans le système Gamma Diagnost Tomo, grâce auquel on obtient l'image complète du corps par la tomographie axiale digitalisée, le révélateur gamma proprement dit est suspendu à un bras articulé. Il est, en outre, muni d'un verre à scintillation de 40 cm de diamètre et de 9 mm d'épaisseur, derrière lequel se trouve une batterie de 61 photo-amplificateurs transformant les signaux lumineux (produits dans le verre par la révélation des rayons gamma), en signaux électriques qui sont traités pour fournir l'image désirée.



Philips



Philips

Ci-dessus : positionnement du révélateur du « Gamma Diagnost Tomo ».
Ci-contre : image tomographique digitalisée de la coupe d'un thorax (l'échelle de concentration du traçant est représenté à droite de l'image).

Ceux-ci sont, à leur tour, traités pour fournir l'image désirée. La vitesse d'acquisition des données est de 40 images à la seconde, avec possibilité de coupler deux Gamma-caméras. La console de contrôle et de traitement des données est munie d'un moniteur d'une résolution de 512 × 512 pixels, (extensible à 640 × 256 pour le corps entier) et 256 niveaux de couleurs sont disponibles. Le système de calcul comporte 2 microprocesseurs spécialisés fonctionnant en parallèle. Ils sont dotés de 400 à 528 Koctets de mémoire vive, programmables en Fortran, en RTL 2, en Assembleur et en PMCL (un langage dédié aux applications biomédicales). Le système est complété par un disque dur de 24 Moctets et une disquette de 1,2 Moctets.

Le calculateur rend donc possible la construction de ***l'image tomographique sur un écran couleur.*** *La possibilité de préciser les contours de la section examinée permet également de corriger les données en tenant compte de l'absorption des rayons Gamma par les tissus. Les images antérieures peuvent être rappelées simultanément pour, par exemple, vérifier d'une manière simple l'évolution du processus de concentration de la* ***substance traçante*** *dans un organe. En outre, on peut réaliser des copies d'images sur papier ou sur transparent.*

De tels systèmes ont profondément contribué à éliminer les risques liés aux procédures classiques, en particulier dans le domaine du diagnostic des affections pulmonaires rénales, cardiaques, du foie, du pancréas, du système osseux et du système endocrinien.



Philips

Tableau de commande et de contrôle d'un thermographe à image digitalisée. A droite : manettes de positionnement et curseurs de délimitation du champ d'observation.

D'un usage aujourd'hui courant, la thermographie est pourtant une technique d'investigation relativement récente. Les thermographes de première génération utilisaient des plaques photographiques sensibles à l'infrarouge directement exposées aux radiations corporelles du patient. On exploitait ainsi la propriété, commune à de nombreux phénomènes inflammatoires et pathologiques, d'altérer localement la température du corps. L'application correcte de cette technique nécessitait, en ce temps-là, la production de plaques photographiques ***hautement sensibles,*** *même à des variations de quelques fractions de degré.*

Aujourd'hui, le procédé de balayage a recours à des cellules photovoltaïques sensibles à l'infra-rouge et à des circuits de traitement numérique des signaux émis par elles. Des images en fausses couleurs (c.a.d. en couleurs symboliques) sur grand écran, qu'on peut également obtenir en sortie d'imprimante ou stocker sous forme numérique dans une mémoire de masse, remplacent les anciennes photographies.

Le document ci-dessus reproduit le tableau de contrôle d'un ***thermographe à affichage numérique.*** *Le tube thermographique est doté d'une cellule photovoltaïque à indio-antimoine baignant dans de l'azote liquide. Des moteurs électriques contrôlent les mouvements transversaux (+ 25°), latéraux (+ 100°) et de niveau (50 ÷ 180 cm).*

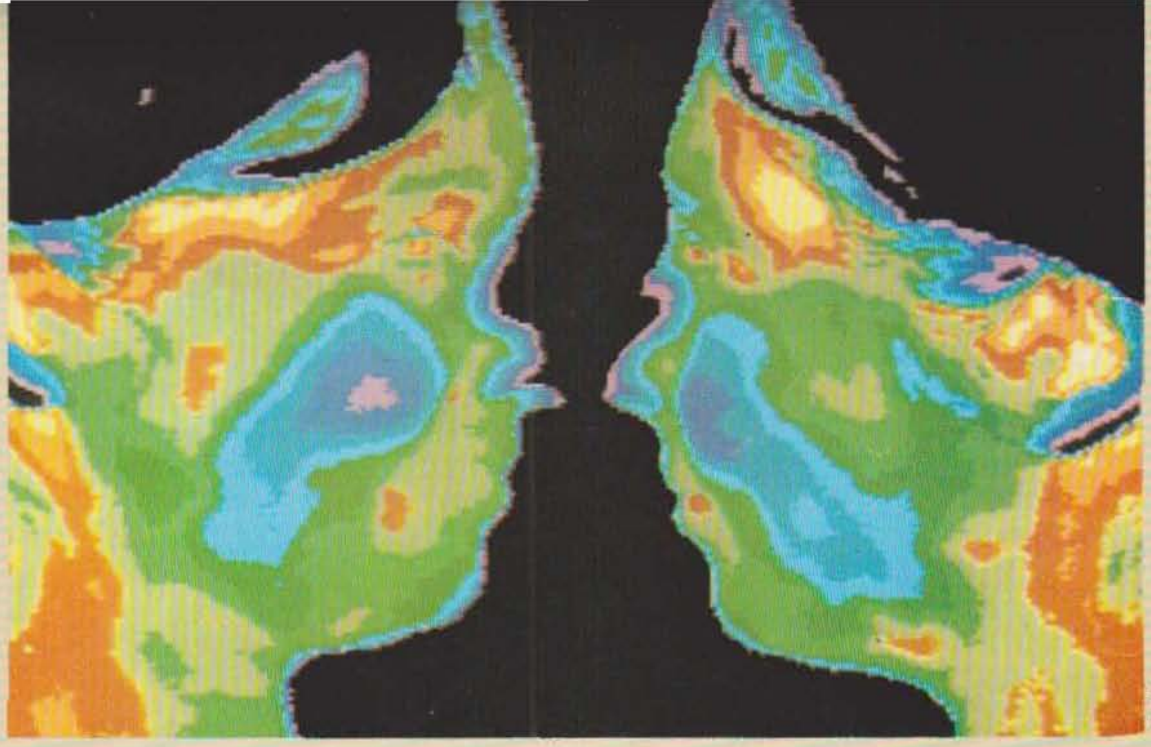
Grâce à sa fine résolution, la cellule permet de distinguer des détails sous un angle inférieur à 0,002 radians, alors qu'en résolution thermique la température varie seulement de 0,08°C. Le temps d'exposition nécessaire, pour obtenir une réponse complète, est d'une seconde.

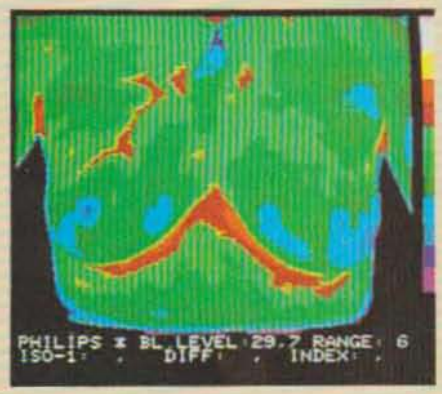
Les signaux émis sont enregistrés par le système qui élabore une matrice numérique du champ d'investigation qui est envoyée sur un moniteur à dix couleurs de 30 cm. Sur ce dernier, une matrice de 312 × 240 pixels, bien en évidence, complète par une échelle des couleurs certaines indications, permettant de repérer avec précision l'échelle des températures.

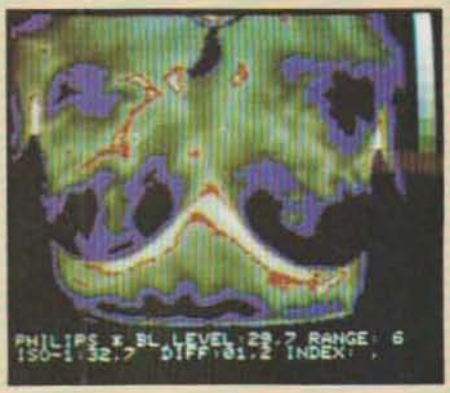
La température du seuil est réglée entre 15 et 40°C. Toutes les données indiquant que la température est inférieure à ce niveau sont visualisées en noir, d'où son nom de « black level ». L'étendue de la zone au-dessus du black level (range) est, quant à elle, réglée aux valeurs, 2, 3, 4, 5, 8, 10, 15°C.

Une fois visualisée, l'image thermographique est ***archivée*** *et, si besoin,* ***rappelée*** *sur une partie de l'écran. Ce qui permet de mettre en évidence simultanément des zones anatomiques particulières qui ne pourraient entrer en même temps dans le champ visuel de la caméra, comme par exemple les deux côtés d'une tête ou les deux côtés d'une main (photos de la page 1035).*

Il existe plusieurs manières de numériser l'image avant de la transcrire en fausses couleurs. Grâce à la première fonction du circuit électronique, on met en évidence deux isothermes d'une amplitude égale à 3% de la gamme sur deux couleurs. Une fois le niveau de la première isotherme choisi (ISO-1), celui de la seconde isotherme peut varier à volonté dans la gamme supérieure ; la différence de niveau est alors automatiquement affichée à l'écran (DIFF). Des curseurs du tableau de











Après calculs, l'image thermographique d'une comparaison opérée entre les deux côtés du visage est visualisée sur écran vidéo et, le cas échéant, enregistrée sur support magnétique. Ci-contre : deux représentations d'un thorax féminin et, dessous, la comparaison thermographique des mains d'un patient. La vascularisation réduite de la main gauche peut être quantifiée par l'intermédiaire du profil thermique visible à droite de l'écran.

contrôle permettent de délimiter, sur le moniteur, une fenêtre sur une région avec calculs en ***temps réel*** *de l'indice thermographique (INDEX). Mais surtout, l'ordinateur est en mesure de mettre en évidence le profil thermique d'une ligne horizontale sélectionnée par l'opérateur sur le moniteur (photo ci-dessus). La courbe de température est établie d'après les signaux thermiques traités en fonction du nombre de pixels « allumés ».*

Outre la traditionnelle application au ***diagnostic du cancer*** *du sein, la possibilité d'obtenir des thermogrammes, en temps réel dans un intervalle d'une seconde, permet accessoirement de quantifier les variations thermiques associées à l'administration de remèdes.*

Soit le champ :

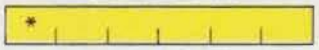
```
01 ARRIVEE          PIC X(6).
```

Dans les exemples suivants, on a mis en évidence les zones ARRIVEE après chacun des deux transferts (MOVE ALL et MOVE simple) :

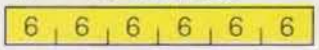
MOVE ALL ' * ' TO ARRIVEE



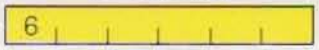
MOVE ' * ' TO ARRIVEE



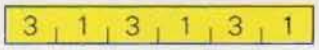
MOVE ALL ' 6 ' TO ARRIVEE



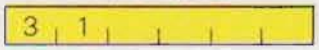
MOVE ' * ' TO ARRIVEE



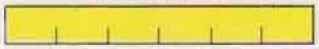
MOVE ALL ' 31 ' TO ARRIVEE



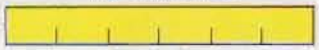
MOVE ' 31 ' TO ARRIVEE



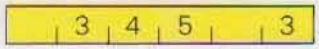
MOVE ALL ' ' TO ARRIVEE



MOVE ' ' TO ARRIVEE



MOVE ALL ' 345 TO ARRIVEE



La dernière instruction montre comment le compilateur respecte la répétition de la rangée de caractères indiquée jusqu'au remplissage du champ, tout en retranchant les caractères en trop.
De plus, les espaces sont traités comme des caractères à part entière, quelle que soit leur position.

MOVE numérique. L'opération de transfert des données nécessite, comme on l'a vu précédemment, une bonne connaissance des mécanismes internes. La validité des résultats est donc directement liée à l'utilisation correcte des instructions et au dimensionnement des champs. Ainsi, le MOVE alphanumérique ne se préoccupe pas de la signification des données, alors que le MOVE numérique tient compte de l'USAGE du champ d'arrivée et d'autres facteurs comme le signe et la position du point décimal.
Le transfert de type numérique est effectué en 5 étapes :

1/ Positionnement du point décimal
2/ Remplissage du champ ARRIVEE
3/ Troncature des chiffres excédentaires ou complémentation du champ d'arrivée,
4/ Conversion du format de départ en format d'arrivée
5/ Traitement du signe.

1/ Positionnement du point décimal.
Pour transférer un nombre décimal, les chiffres entiers devront être correctement positionnés à gauche et les décimaux à droite du point.
Dans le transfert suivant :

NOM-DEPART vers NOM-ARRIVEE, où :

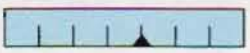
```
01 NOM-DEPART       PIC S9(4)V9(3).
01 NOM-ARRIVEE      PIC S9(6)V9(3).
```

L'instruction :

```
MOVE NOM-DEPART TO NOM-ARRIVEE.
```

effectue le positionnement de la façon suivante :

NOM-DEPART



NOM-ARRIVEE



(A noter que le symbole ▲ représente la position virtuelle en mémoire du point décimal).
C'est une opération qui est réalisée même quand la PICTURE des champs départ et arrivée ne comporte par le symbole V. Pour le compilateur, les descriptions :

```
01 NOMBRE-1         PIC S9(4).
01 NOMBRE-2         PIC SP(3)9(5).
01 NOMBRE-3         PIC S99P(5).
```

sont équivalentes aux suivantes :

```
01 NOMBRE-1     PIC S9(4)V.
01 NOMBRE-2     PIC SVP(3)9(5).
01 NOMBRE-3     PIC S99P(5)V.
```

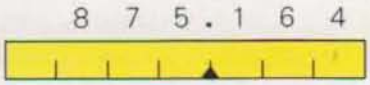
Les modalités de positionnement déjà décrites sont donc applicables aux champs de l'exemple. De plus, dans une constante numérique, le point décimal est assimilé à une position virtuelle.
Par exemple, si le champ ARRIVEE est défini par :

```
01 ARRIVEE      PIC S9(4)V9(3).
```

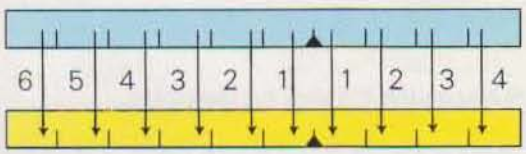
L'instruction :

```
MOVE 875.164 TO ARRIVEE.
```

comporte le positionnement du schéma suivant :



2/ Remplissage du champ d'arrivée. Après avoir positionné le point décimal des deux champs, la partie entière (arrivée) est remplie de la droite vers la gauche selon la séquence :



3/ Troncature ou complémentation du champ d'arrivée. Comme pour le MOVE alphanumérique, quand les champs n'ont pas la même longueur, on procède à la vérification du remplissage jusqu'aux limites du champ. De la même manière, on retranchera les chiffres excédentaires.

Soient les champs :



J. Pickerell/Marka

Tableau de contrôle de l'ordinateur Univac 1100/8.

```
01 NOM-DEPART      PIC S9(3)V9
                   VALUE 436.2.

01 NOM-ARRIVEE     PIC S9(6)V9(3).
                   VALUE 123456.789.
```

Voici la configuration des deux champs avant et après l'exécution de :

```
MOVE NOM-DEPART TO NOM-ARRIVEE
```

NOM-DEPART	**NOM-ARRIVEE**
Avant :	
4 3 6 2	1 2 3 4 5 6 7 8 9
Après :	
4 3 6 2	1 2 3 4 5 6 7 8 9

Puisque, dans le cas que nous venons d'examiner, nous avons une longueur de champ départ inférieure à celle du champ d'arrivée, on doit remplir les espaces libres avec des zéros non significatifs, à gauche et à droite des chiffres transférés.
Dans le cas contraire, il sera procédé à une troncature de la valeur transférée, aussi bien pour les entiers que pour les décimaux.
Pour clarifier le concept, considérons les champs :

```
01 NOM-DEPART          PIC S9(4)V9(2)
                       VALUE 1436.78.

01 ARRIVEE.
   05 NOM-ARRIVEE-1    PIC S9(3)V9(2).
   05 NOM-ARRIVEE-2    PIC S9(3)V9(4).
   05 NOM-ARRIVEE-3    PIC S9(4).
   05 NOM-ARRIVEE-4    PIC SV9.
   05 NOM-ARRIVEE-5    PIC S9(6).
```

On voit, à travers ces exemples, comment une opération de transfert vers un champ d'arrivée qui n'a pas été correctement dimensionné conduit à des résultats erronés.
En fait, si la troncature opérée par le MOVE est appliquée à la partie entière, on perd les chiffres les plus significatifs. En particulier, dans le cas du dernier exemple, où la valeur 1000000.796 devient 0 après transfert.

4/ Conversion de format. Au cours du transfert des données numériques, le compilateur traite les champs avec une codification interne différente.
Connaissant l'USAGE implicite ou explicite des champs de départ et d'arrivée, il est capable de transformer correctement les données dans le format demandé.
Cette conversion est faite, sur la valeur éventuellement retranchée de la donnée, à l'intérieur du compilateur. Notons que cela revient à le pénaliser lourdement quand il s'agit de convertir une grande masse de données.
L'exécution de routines spéciales et la mobilisation d'une mémoire additionnelle engendrent, en effet, un accroissement du temps de calcul et une extension de l'occupation en mémoire centrale.
Il est donc conseillé de décrire les champs numériques d'une manière homogène, en ne faisant appel à la conversion de format qu'en cas de force majeure.

5/ Traitement du signe algébrique. Dernière opération du compilateur sur le champ d'arrivée : le traitement du signe. Si le champ contient, dans sa propre PICTURE, le symbole S, le signe de la valeur transférée est bien codifié, conformément à la codification interne employée.
Inversement, l'instruction MOVE transfère uniquement la valeur absolue de la donnée en l'absence du symbole S.
Il est donc recommandé de recourir systématiquement au symbole S dans la description des champs numériques afin d'éviter une transcription tronquée toujours possible, en dehors de quelques rares exceptions qui seront signalées au fur et à mesure de leur apparition.
Quand elle ne sont pas respectées, ces ex-

RETRANCHEMENTS DANS LE TRANSFERT DES DONNEES

Instructions de transfert	Contenus des champs d'arrivée après le MOVE
MOVE NOM-DEPART TO NOM-ARRIVEE 1	4 3 6 7 8
MOVE NOM-DEPART TO NOM-ARRIVEE 2	4 3 6 7 8 0 0
MOVE NOM-DEPART TO NOM-ARRIVEE 3	1 4 3 6
MOVE NOM-DEPART TO NOM-ARRIVEE 4	7
MOVE 1000000.796 TO NOM-ARRIVEE 5	0 0 0 0 0 0

INSTRUCTION MOVE

CHAMP-DEPART \ CHAMP-ARRIVEE				Composé	Elémentaire								
					Alpha-numérique	Alphabétique	Numérique					Numérique à l'impression	Alphanum. à l'impression
							DISPLAY	COMP	COMP-3	COMP-1	COMP-2		
Composé				A	A	A						P	P
Elémentaire	Alphanumérique			A	A	A						P	P
	Alphabétique			A	A	A							P
	Constante	Alpha-numérique		A	A							P	P
		Alphabétique		A	A	A							P
		Figurative	SPACES	A	A	A							P
			HIGH-VALUES LOW-VALUES	A	A	A							P
			ZEROES	A	A		N	N	N	N	N	P	P
		Numérique		A	A		N	N	N	N	N	P	P
	Alphanumérique d'impression			A	A								P
	Numérique d'impression			A	A								P
	Numérique	DISPLAY		A	A*		N	N	N	N	N	P	P
		COMP		A	A*		N	N	N	N	N	P	P
		COMP-3		A	A*		N	N	N	N	N	P	P
		COMP-1					N	N	N	N	N	P	
		COMP-2					N	N	N	N	N	P	

* Uniquement pour les nombres entiers

MOVE permis

MOVE non permis

ceptions sont traitées par le compilateur comme des erreurs. Cette méthode permet d'éviter de perdre du temps à la recherche d'erreurs difficiles à analyser.
L'exemple suivant illustre le cas d'une description erronée d'une donnée numérique qui aboutit au résultat : - 1 + 1 = 2.

```
01 COMPTEUR          PIC 9.
—
—
—
—
—
PROCEDURE DIVISION.
COMPTER.
        MOVE-1 TO COMPTEUR.
        ADD  1 TO COMPTEUR.
```

Le COMPTEUR ne comportant pas de symbole S, le transfert de - 1 engendre la perte du signe algébrique. Après le MOVE, le contenu est, de ce fait, 1 et non - 1, et la valeur obtenue en additionnant 1 est 2 et non 0.
Le tableau de la page précédente résume toutes les combinaisons possibles des MOVE alphanumériques et numériques. On a utilisé le symbole A pour indiquer un MOVE alphanumérique, N pour un MOVE numérique et P (Print), pour un MOVE d'impression.

L'instruction Inspect

Pendant son déroulement, le programme peut avoir besoin de gérer le contenu d'un champ : analyser le nombre de caractères de blanc qu'il contient, vérifier si une certaine combinaison de caractères est présente... Les instructions décrites jusqu'à présent ne peuvent répondre de manière bienfaisante à ces questions. C'est pourquoi on a recours à une instruction spécifique, INSPECT.
Dans son format général, cette instruction est capable de décompter et de remplacer les répétitions d'un groupe de caractères par un ou plusieurs autres caractères d'un champ déterminé.
Les deux fonctions (décompter et substituer) peuvent cependant être exécutées séparément, grâce à deux sous-formats.

La fonction de comptage. Pour réaliser cette fonction, l'instruction a besoin de savoir :

1/ Quel champ numérique servira au compteur (**compteur**)

2/ Le nom du champ contenant la chaîne de caractères à analyser **(chaîne)**

3/ et les objets qu'elle doit compter (**combinaison**)

Le tableau ci-contre (page 1041) fournit le format de l'instruction où « combinaison » définit la série des caractères à rechercher dans la chaîne analysée.
En réalité, pour effectivement opérer, l'instruction doit aussi savoir selon quelle modalité cette recherche doit être effectuée.

La clause ALL. Considérons, par exemple, le cas où l'on veut connaître le nombre de caractères A présents dans le champ :

```
01 COULEUR PICX (20) VALUE 'AMARANTE'.
```

Le calcul peut être effectué en utilisant, comme compteur, le champ :

```
01 COMP PIC S9(5)    COMP VALUE 0.
```

Notons que, pendant le comptage, l'instruction INSPECT incrémente la valeur du compteur pour chaque combinaison valable rencontrée. Il est donc nécessaire d'initialiser le compteur, sauf si l'on ne désire pas un calcul total des différentes chaînes analysées.
Dans notre cas, INSPECT aura le format suivant :

```
INSPECT COULEUR TALLYING COMP
                    FOR ALL ' A '.
```

La « combinaison » requise est alors ALL ' A ' (« tous les A présents dans la chaîne »). Après exécution, le résultat sera 3, mémorisé dans le champ COMP.
On aurait obtenu le même résultat avec les instructions :

```
MOVE ' A ' TO FLAG.
INSPECT COULEUR TALLYING COMP
                    FOR ALL FLAG.
```

FORMAT DE L'INSTRUCTION INSPECT POUR COMPTAGE

INSPECT chaîne TALLYING compteur FOR combinaison

Dans la dernière des expression, FLAG indique le champ de la WORKING-STORAGE SECTION:

```
01 FLAG   PIC X.
```

INSPECT aide également à rechercher en même temps, dans un champ donné, plusieurs combinaisons de caractères.

Considérons encore le champ couleur précédemment défini (CHAMP) pour lequel on désire calculer simultanément combien de groupes AMA, R, ANTE sont présents.
On procède ainsi :

```
01 COULEUR     PICX(20)  VALUE
                         'AMARANTE'.
01 COMP-AMA  PIC 9(2)   COMP VALUE 0.
01 COMP-R    PIC 9(2)   COMP VALUE 0.
01 COMP-ANTE PIC 9(2)   COMP VALUE 0.
PROCEDURE DIVISION.
COMPTER.
    INSPECT COULEUR
        TALLYING
        COMP-AMA FOR ALL ' AMA ',
        COMP-R FOR ALL ' R ',
        COMP-ANTE FOR ALL ' ANTE '.
```

A présent, voyons les valeurs des compteurs après l'opération :

```
COMP-AMA  = 1
COMP-R    = 1
COMP-ANTE = 1
```

La clause LEADING. Avec cette clause, l'instruction INSPECT vérifie si une certaine combinaison de caractères se retrouve en début de champ.
Pour savoir si le contenu du champ COULEUR commence par les caractères AM, on écrit :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR LEADING ' AM '.
```

Dans le cas que nous sommes en train d'examiner, AMARANTE commence par le bloc de caractères ' AM ', l'instruction incrémente donc le champ COMPTEUR.

Pour le même champ, l'instruction :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR LEADING ' MAR '.
```

n'aurait pas modifié le contenu de COMPTEUR, simplement parce que le bloc MAR ne constitue pas la partie initiale du mot AMARANTE.

La clause CHARACTERS. Elle fournit le nombre de caractères que contient un champ déterminé. La valeur donnée par l'instruction comprend aussi les blancs.

L'instruction :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR CHARACTERS.
```

rétablit la valeur 20, dans le compteur correspondant au nombre de caractères déclarés dans la PICTURE du champ COULEUR, et non 8 (nombre de caractères différents de blanc constituant le mot AMARANTE).

La clause BEFORE. Dans toutes les clauses de l'instruction INSPECT que nous avons examinées jusqu'à présent, le champ à analyser a toujours été balayé à partir du premier caractère et de gauche à droite.
Cette contrainte peut être éliminée grâce à deux clauses qui permettent d'effectuer toutes les recherches décrites précédemment en partant de n'importe quelle position.
Ce point de départ, qui n'est pas fixé de manière rigide par le programmeur, est lié au contenu du champ.
Il est donc permis de prendre, pour point de

EMPLOI DE LA CLAUSE BEFORE

```
 01  ARTICLE.
     05  NUMERO-SERIES                  PIC X(10).
     05  DESCRIPTION                    PIC X(20).
     05  .........
     05  .........
     .
     .
     .
 01  K-SERIES-CAR                       PIC 99(2)  COMP VALUE 0.
     88  TROIS-CARACTERES          VALUE 3.
     88  DEUX-CARACTERES           VALUE 2.
     88  UN-CARACTERE              VALUE 1.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
     .
     .
     .
     .
 REGARDER-SERIES.
     INSPECT NUMERO-SERIES
             TALLYING K-SERIES-CAR FOR CHARACTERS BEFORE '/'.
     IF  TROIS-CARACTERES
         PERFORM CALCUL-SERIE-3.
     IF  DEUX-CARACTERES
         PERFORM CALCUL-SERIE-2.
     IF  UN-CARACTERE
         PERFORM CALCUL-SERIE-1.
     .
     .
     .
```

départ, la position de la chaîne où a été vérifiée l'existence d'une combinaison de caractères. Si la clause CHARACTERS précédemment décrite n'est pas très courante, elle peut avoir des applications très intéressantes dès qu'elle est utilisée en même temps que BEFORE.
Un exemple très parlant concerne l'application de BEFORE à la gestion d'un magasin, où le code de série de chaque article est composé d'un nombre quelconque de caractères toujours séparé du nombre d'articles, par le caractère /.
Celle-ci est ainsi programmée :

ABC/1234
ZX/789
U/045

Supposons que nous ayions à calculer autrement chaque article selon que la série est constituée de 1, 2, 3..., N caractères.

La seule manière de connaître exactement la série à laquelle appartient l'article est de procéder comme il est indiqué dans le listing ci-dessus.
En reprenant l'exemple illustrant la première forme de la clause CHARACTERS, on aurait pu dénombrer les caractères du mot AMARANTE en utilisant indifféremment une des formes suivantes :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS BEFORE.
```

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CARACTERS BEFORE
        SPACES
```

Le résultat aurait été 8 dans les deux cas.

On notera que, dans le second format, la constante figurative SPACES a été utilisée à la

place du caractère ' '.
Ceci reste valable pour n'importe quel format INSPECT. Dans ce contexte, SPACES équivaut à un seul espace blanc, tout comme ZEROS est interprété comme un seul 0.

La clause AFTER. Alors qu'avec la clause BEFORE l'instruction INSPECT s'exécute à partir du premier caractère à gauche et s'arrête au moment où l'existence du caractère déclaré dans BEFORE a été vérifiée, on a le comportement inverse avec AFTER.
En effet, le champ est balayé à partir de la gauche jusqu'à détection du caractère déclaré. Le contrôle des autres conditions indiquées dans l'instruction commence après cette opération de vérification.
Si, en considérant le champ :

```
01 COULEUR PICX(20) VALUE ' AMARANTE '.
```

on désire augmenter le nombre de caractères après le premier N rencontré dans le champ, on emploie l'instruction :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS AFTER 'N'.
```

Par contre, si on désire connaître le nombre de caractères compris entre le groupe AMA et le groupe TE, l'instruction :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS
                BEFORE 'TO'
                AFTER 'AMA'.
```

n'est pas admise.
Il faut recourir à plusieurs INSPECT et finalement calculer le nombre recherché par différence entre les valeurs obtenues par chaque instruction.
Le format général de l'instruction INSPECT utilisée pour compter les caractères est schématisé dans le tableau ci-dessous.

La fonction de substitution. Cette fonction nécessite d'examiner les trois points suivants :

1/ Champ à analyser par INSPECT
2/ Elément à remplacer
3/ Caractères de substitution

La fonction se présente ainsi :

```
INSPECT champ
        REPLACING combinaison-existante
        BY
        combinaison-nouvelle.
```

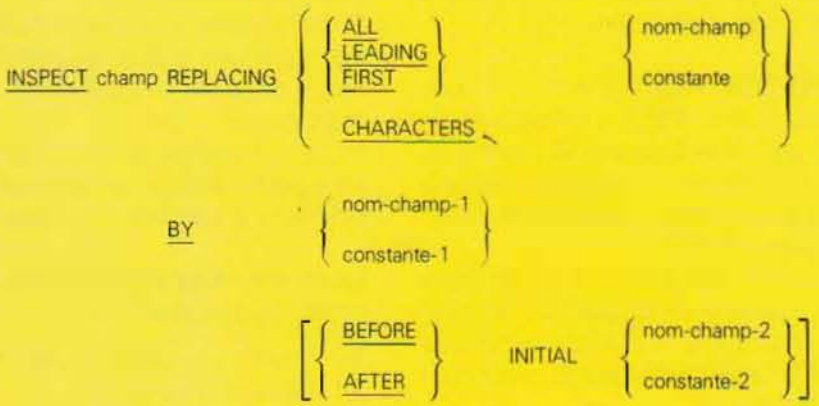
Cela revient à dire :

Balayer (INSPECT) le contenu du **champ** en remplaçant (REPLACING) la combinaison de caractères **combinaison-existante** par (BY) la nouvelle combinaison **combinaison-nouvelle**".
Comme pour la fonction de comptage, le terme "combinaison-existante" comprendra également, et d'une manière concise, la clause de description des caractères à substituer. Celle-ci

FORMAT GENERAL DE L'INSTRUCTION INSPECT AVEC FONCTION DE COMPTAGE

```
INSPECT champ TALLYING compteur FOR { { ALL       } { nom-champ          } }
                                    { { LEADING   } { constante          } }
                                    { { CHARACTERS} { constante-figurative} }

[ { BEFORE } INITIAL { nom-donnée } ]
[ { AFTER  }         { constante   } ]
```

FORMAT GENERAL DE L'INSTRUCTION INSPECT AVEC FONCTION DE SUBSTITUTION



est développée dans le format qui se trouve en haut de cette page.

Illustrons nos propos avec quelques exemples d'INSPECT.

Soit le champ NOMBRE-EN-LETTRES suivant :

```
01 NOMBRE-EN-LETTRES        PICX(30)
VALUE ' MILLESEPTCENTVINGTTROIS '.
```

Les configurations du champ, avant et après l'exécution des instructions INSPECT, sont données dans la page ci-contre.

Format général de l'instruction INSPECT.

Avec l'instruction INSPECT, on peut lancer, en même temps, des opérations de comptage et des opérations de substitution.
A noter qu'on peut même effectuer, avec cette instruction, plusieurs opérations de différent type sur le même champ.
Si on voulait effectuer sur le champ initial NOMBRE-EN-LETTRE :

1/ le décompte de tous les caractères différents de blanc, en utilisant K-CARACTERES comme compteur,
2/ le décompte de tous les caractères ' T ' (K-T comme compteur),
3/ la substitution de la chaîne ' WWWW ' au premier groupe ' CENT ',
4/ le remplacement de toutes les lettres ' T ' suivant le groupe ' SEPT ' par ' N ',

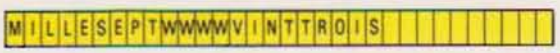
INSPECT s'écrira :

```
INSPECT NOMBRE-EN-LETTRES
        TALLYING K-CARACTERES
                 FOR CHARACTERS
                   BEFORE ' '
                 K-T
                 FOR ALL 'T'
      REPLACING FIRST ' CENT '
                 BY 'WWWW'
                 ALL 'T' BY 'N'
                 AFTER 'SEPT'.
```

Après l'exécution de l'instruction, on aura :

```
K-CARACTERES = 23
K-T          =  5
```

et la configuration du champ deviendra :



L'instruction INSPECT, (format ci-contre, page 1045), est également applicable aux champs numériques, à condition d'utiliser, implicitement ou explicitement, USAGE DISPLAY.

APPLICATION D'UNE INSTRUCTION INSPECT

CONFIGURATION INITIALE DU CHAMP NOMBRE-EN-LETTRES

| M | I | L | L | E | S | E | P | T | C | E | N | T | V | I | N | G | T | T | R | O | I | S | | |

L'instruction STRING

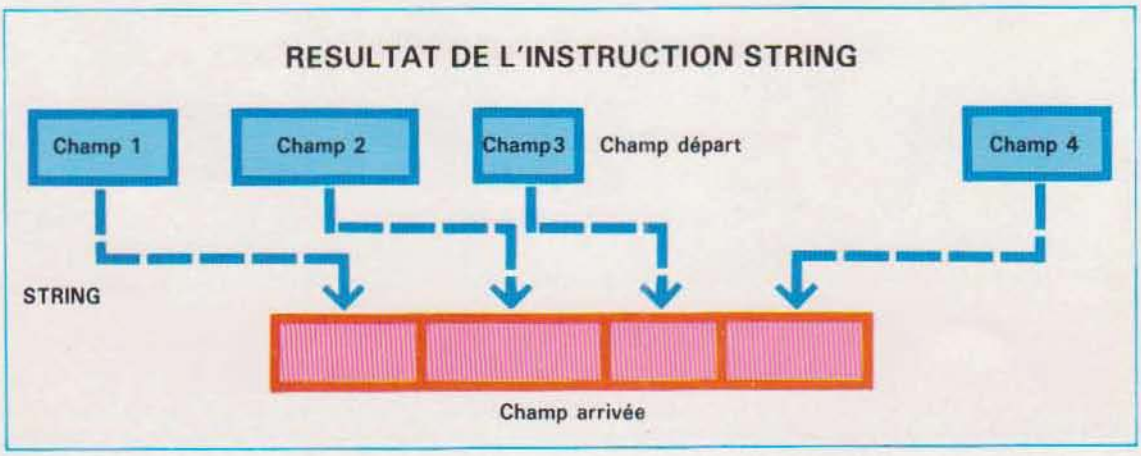
Pour compléter l'examen des instructions spécialisées dans la manipulation des données, analysons maintenant deux instructions très importantes : STRING et UNSTRING. Dans un certain sens, ces instructions regroupent les possibilités de MOVE et d'INSPECT.
Nous savons désormais que MOVE est capable de transférer des données d'un champ de départ vers un champ d'arrivée à condition d'être préalablement décrits dans la division données. Deux cas se présentent : soit l'instruction MOVE n'entre pas dans le contenu du champ (MOVE alphanumérique), soit elle le fait (MOVE numérique), mais alors il s'agit de respecter le type de champ d'arrivée. Pour MOVE, la valeur de la donnée transférée est complètement transparente et, en général, il n'est pas permis de ne déplacer qu'une partie de la donnée, à moins que le champ ne soit volontairement défini de cette manière.
Le champ CHAINE décrit comme suit :

```
01 CHAINE
   05 CHAINE-1          PIC X(3).
   05 CHAINE-2          PIC X(5).
      10 CHAINE-31       PIC X.
      10 CHAINE-32       PIC XX.
      10 CHAINE-33       PIC X(4).
```

contient la valeur « EXEMPLE DE MOVE », avec, comme contenu des différents sous-champs :

```
CHAINE-1  = 'EXE'
CHAINE-2  = 'MPLE'
CHAINE-31 = 'D'
CHAINE-32 = 'E'
CHAINE-33 = 'MOVE'
```

Dans ce champ, il est possible de déplacer, par exemple, la préposition 'DE' puisque les caractères la composant appartiennent à deux sous-champs différents.
Contrairement à l'instruction MOVE, l'instruction INSPECT fait abstraction d'éventuelles sous-descriptions du champ analysé. Elle « entre » dans son contenu et le modifie le cas échéant.
Les instructions STRING et UNSTRING constituent la seule manière d'analyser un contenu et d'effectuer le déplacement de certaines de ses parties. STRING permet de transférer le contenu complet ou partiel d'un ou de plusieurs champs vers un champ unique d'arrivée en les disposant l'un à la suite de l'autre selon l'ordre déclaré par le programmeur (voir schéma ci-dessous).
Supposons que l'on doive contrôler si une date saisie à la console sous la forme JJ/MM/AA (jour/mois/année) est antérieure ou non à celle de l'ordinateur.
Grâce à l'instruction ACCEPT, on peut prélever la date du jour sur le calendrier de la machine. Cette date est fournie dans le champ déclaré sous la forme AAMMJJ (année/mois/jour), plus appropriée aux opérations de contrôle. Ainsi le 13 décembre 1983 est facilement vérifiable car 831213 (représentation du 13/12/83) est plus grand que 831111 (représentation du 11/11/83). Il est évident que la date fournie sous la forme JJ/MM/AA



CONVERSION D'UNE DATE

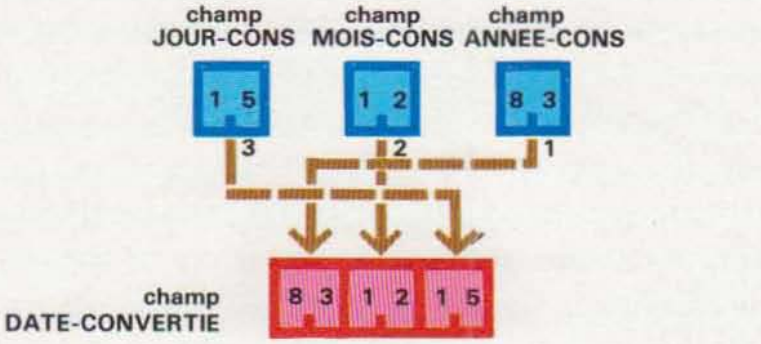
```
 01  DATE-CONSOLE.
     05  JOUR-CONS                PIC 9(2).
     05  FILLER                   PIC X.
     05  MOIS-CONS                PIC 9(2).
     05  FILLER                   PIC X.
     05  ANNEE-CONS               PIC 9(2).
     05  FILLER                   PIC X.
*
*
 01  DATE-CALCUL                 PIC 9(6).
 01  DATE-CALCUL    REDEFINES DATE-CALCUL.
     05  ANNEE-CALCUL             PIC 9(2).
     05  MOIS-CALCUL              PIC 9(2).
     05  JOUR-CALCUL              PIC 9(2).
*
*
 01  DATE-CONVERTIE.
     05  ANNEE-CONV               PIC 9(2).
     05  MOIS-CONV                PIC 9(2).
     05  JOUR-CONV                PIC 9(2).
*
*
 PROCEDURE DIVISION.
 ACCEPTER-DATES.
     DISPLAY '** DIGITALISER DATE (JJ,MM,AA)'
             UPON CONSOLE.
     ACCEPT  DATE-CONSOLE FROM CONSOLE.
     STRING  ANNEE-CONS
             MOIS-CONS
             JOUR-CONS
                        DELIMITED BY SIZE
                        INTO      DATE-CONVERTIE.
     ACCEPT  DATE-CALCUL FROM DATES.
     IF  DATE-CONVERTIE IS GREATER THAN DATE-CALCUL
         PERFORM .........
         ..................
         ..................
*
*
```

doit être opportunément convertie en AAMMJJ.
Le problème peut être résolu par le programme ci-contre (voir listing ci-dessus).
L'instruction :

```
STRING  ANNEE-CONS
        MOIS-CONS
        JOUR-CONS
        DELIMITED BY SIZE
INTO DATE-CONVERTIE
```

génère les opérations schématisées dans la page suivante où l'on suppose que la date du 15/12/83 a été saisie à la console.
En définitive, l'instruction STRING transfère, dans le champ arrivée (INTO DATE-CONVERTIE), et dans l'ordre indiqué, le contenu entier des champs ANNEE-CONS, MOIS-CONS, JOUR-CONS.
La clause DELIMITED BY SIZE indique que le transfert concerne le champ dans ses dimensions maximales, telles qu'elles ont été déclarées dans division données. On peut, en effet, noter dans cet exemple que les champs transférés étaient déclarés chacun pour 2 caractères : PIC 9(2).
Pour obtenir le même résultat, on aurait également pu utiliser trois MOVE à la place de STRING.
Connaissant toutes les clauses prévues par l'instruction STRING, son emploi paraît beaucoup plus simple. Les exemples suivants donnent une idée de sa puissance. Elle agit sur des champs alphanumériques et peut créer, dans un champ arrivée, une combinaison de chaînes

CONVERSION D'UN FORMAT DATE AVEC L'INSTRUCTION STRING



de caractères en provenance de plusieurs champs départ.
Les caractères à prélever sont déterminés en fonction d'un ou de plusieurs caractères déclarés comme bornes.
Soit, par exemple, le champ :

```
01 DEPART          PIC X(4).
```

dont le contenu, à un certain moment, vaut

```
ABCD
```

On veut composer dans le champ :

```
01 ARRIVEE         PIC X(40).
```

une chaîne de caractères contenant :

1/ la constante alphanumérique : EXEMPLE DE STRING,
2/ un blanc,
3/ le caractère : (deux points),
4/ un blanc,
5/ toutes les lettres qui précèdent la lettre C dans le contenu du champ DEPART.

L'instruction STRING (voir format général en page 1049) sera alors :

```
STRING 'EXEMPLE DE STRING' DELIMITED
                                BY SIZE
       ' '                      DELIMITED
                                BY SIZE
       ' : '                    DELIMITED
                                BY SIZE
       SPACE                    DELIMITED
                                BY SIZE
       DEPART                   DELIMITED
                                BY 'C'
INTO
       ARRIVEE.
```

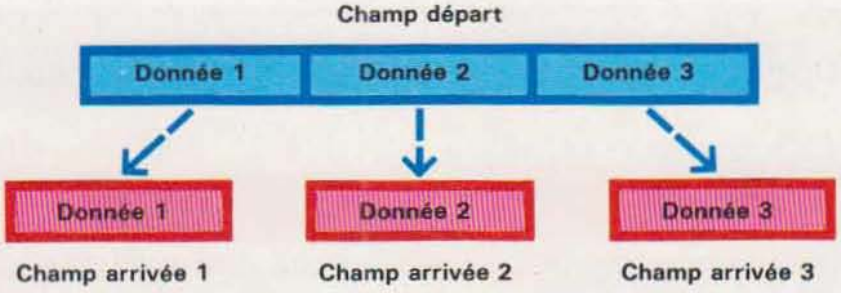
Dans ce cas, le champ arrivée commencera à partir du premier caractère à gauche. Il est possible d'établir un autre point de départ en utilisant un pointeur.
Ainsi, la chaîne précédente serait composée à partir de la septième position du champ ARRIVEE. Elle utiliserait alors les instructions suivantes :

```
MOVE 7 TO POSITION.
STRING 'EXEMPLE DE STRING'
       ' '
       ' : '
       SPACE      DELIMITED BY SIZE
       DEPART     DELIMITED BY 'C'
INTO
       ARRIVEE WITH POINTER POSITION
ON OVERFLOW
       DISPLAY
```

FORMAT GENERAL DE L'INSTRUCTION STRING

```
STRING   { nom-donnée-1 }  [ { nom-donnée-2 } ]
         { constante-1  }  [ { constante-2  } ]  ....

         DELIMITED BY  { nom-donnée-3 }
                       { constante-3  }
                       { SIZE         }

[        { nom-donnée-4 }  [ { nom-donnée-5 } ]
         { constante-4  }  [ { constante-5  } ]  ....

         DELIMITED BY  { nom-donnée-6 }
                       { constante-6  }
                       { SIZE         }  ]

         INTO nom-donnée-7
              [WITH POINTER nom-donnée-8]
              [ON OVERFLOW phrase-impérative]
```

DEROULEMENT DE L'INSTRUCTION UNSTRING



'CHAMP ARRIVEE INSUFFISANT'
UPON CONSOLE.

Dans notre exemple, a été insérée la clause OVERFLOW qui permet de contrôler si le champ d'arrivée est capable de contenir tous les caractères déclarés.

L'instruction UNSTRING

UNSTRING est l'inverse de STRING. L'instruction prélève des données d'un champ alphanumérique unique et les transfère dans un ou plusieurs champs d'arrivée, selon le schéma ci-dessus. La détermination des caractères à prélever est effectuée selon la même logique utilisée par l'instruction STRING, en spécifiant le caractère-limite (DELIMITED BY...).
De plus, UNSTRING comporte des clauses sans équivalents dans STRING (voir format général page 1050).
La clause ALL permet au compilateur d'interpréter un groupe de caractères identiques comme borne unique. Dans la chaîne :

BELLE

par exemple :

DELIMITED BY ALL 'L'

FORMAT GENERAL DE L'INSTRUCTION UNSTRING

```
UNSTRING nom-donnée-1
    [ DELIMITED BY [ALL]  { nom-donnée 2 | constante 1 }
          [ OR [ALL]      { nom-donnée 3 | constante 2 } ] ]
INTO                                  nom-donnée 4
     [DELIMITER IN                    nom-donnée 5]
     [COUNT IN                        nom-donnée 6]
    [ nom-donnée 7                                    ]
    [ [DELIMITER IN                   nom-donnée 8]   ] .....
    [ [COUNT IN                       nom-donnée 9]   ]
[WITH POINTER                         nom-donnée 10]
[TALLYING IN                          nom-donnée 11]
[ON OVERFLOW                          phrase-impérative].
```

EXEMPLE D'APPLICATION DE L'INSTRUCTION UNSTRING

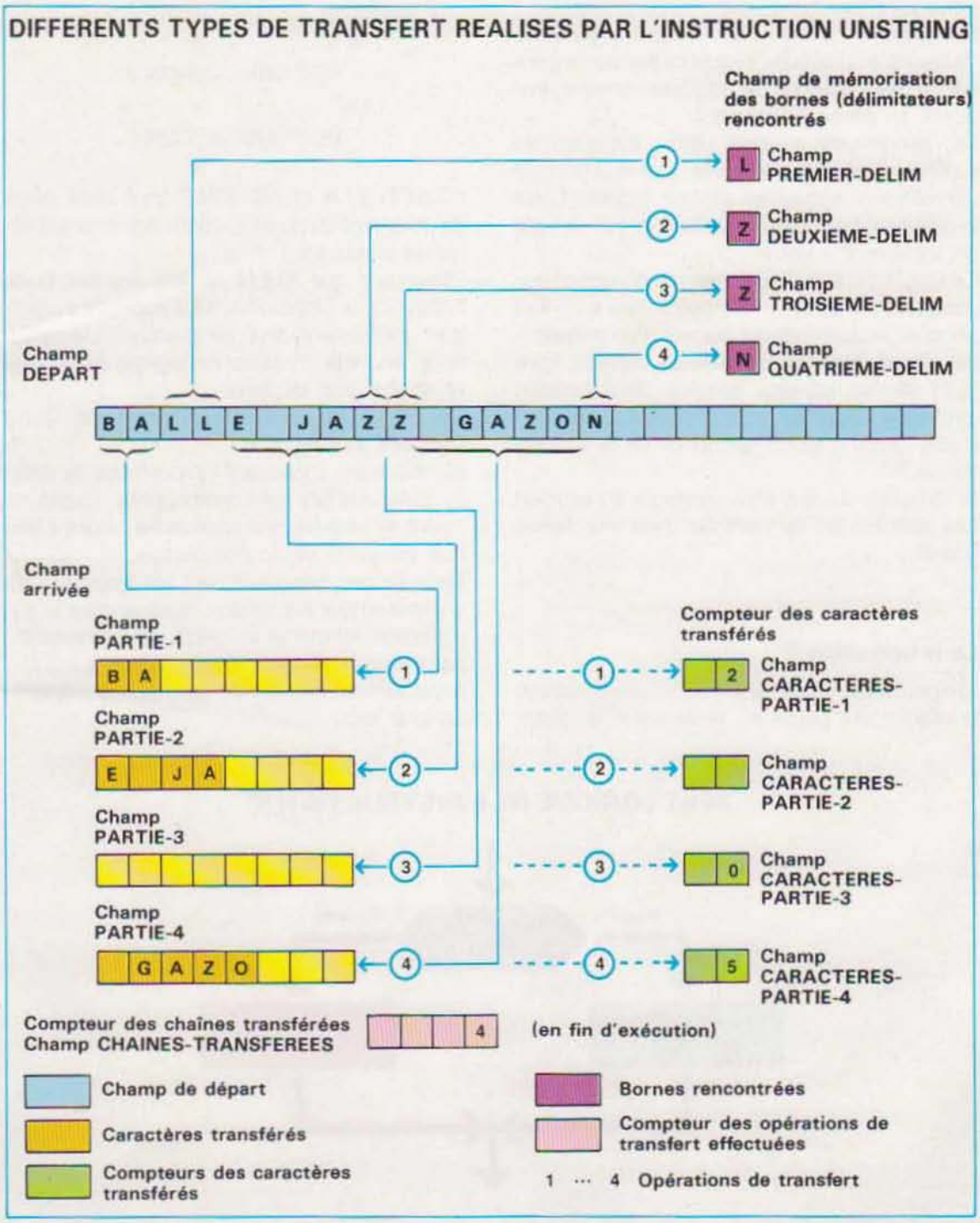
```
MOVE 'BALLE JAZZ GAZON' TO DEPART
UNSTRING DEPART
         DELIMITED BY Z,                          Définition des
                      ALL 'L'                     bornes (DELIM)
INTO PARTIE-1
     DELIMITER  IN  PREMIER-DELIM
     COUNT      IN  CARACTERES-PARTIE 1           1er transfert
*
     PARTIE-2
     DELIMITER  IN  SECOND-DELIM
     COUNT      IN  CARACTERES-PARTIE 2           2e transfert
*
     PARTIE-3
     DELIMITER  IN  TROISIEME-DELIM
     COUNT      IN  CARACTERES-PARTIE 3           3e transfert
*
     PARTIE-4
     DELIMITER  IN  QUATRIEME-DELIM
     COUNT      IN  CARACTERES-PARTIE 4           4e transfert
*
     TALLYING   CHAINE-TRANSFEREES                Compteur des opérations
     ON OVERFLOW                                  de transfert effectuées
          DISPLAY 'OVERFLOW UNSTRING'
               UPON CONSOLE.
```

fera ressortir le groupe 'LL' comme borne unique. Sans cette spécification de la clause ALL, seul le premier "L" aurait été traité comme délimiteur, (le second "L" aurait été identifié comme appartenant à la chaîne suivante).

Nous avons représenté graphiquement (tableau ci-dessous) les définitions des principales clauses de l'instruction UNSTRING.

Verbes de contrôle

Ce qui a été développé jusqu'à présent a per-



mis au lecteur de connaître toutes les opérations sur les données, celles d'entrée/sortie (INPUT/OUTPUT) pour le traitement des fichiers séquentiels en lecture ou en écriture, ainsi que les opérations qui sont spécifiques aux manipulations de données en mémoire. Elles constituent les outils essentiels au programmeur en Cobol.

Un programme est une structure logique de décisions et d'actions séquentielles correspondant à des situations particulières mises en évidence en phase d'analyse.

Le programmeur doit être capable de « débrancher », à n'importe quel moment, le déroulement séquentiel de test logique. C'est la réponse à ce test qui va déterminer le type de traitement à lancer.

Ce Cobol dispose d'un ensemble d'instructions destinées au contrôle des opérations en cours pendant le déroulement normal d'un programme. Ces instructions ont déjà été utilisées, sous leurs formes les plus simples, dans certains exemples présentés, que le lecteur a pu interpréter grâce à sa connaissance de la syntaxe du BASIC.

Le chapitre suivant sera consacré à l'examen des instructions de contrôle dans leur forme générale.

La proposition IF

Elle permet de tester le produit d'une condition préalablement posée et, en fonction du résultat, de choisir entre deux types d'actions. IF équivaut à la phrase suivante = "si (IF) une certaine condition est vérifiée, alors (THEN) exécuter l'ACTION-A, sinon (ELSE) exécuter l'ACTION-B". En utilisant le verbe PERFORM, on a (voir organigramme ci-dessous) :

```
IF condition
    THEN
        PERFORM ACTION-A
    ELSE
        PERFORM ACTION-B.
```

où ACTION-A et ACTION-B sont deux noms de paragraphes ou de sections contenant différentes instructions.

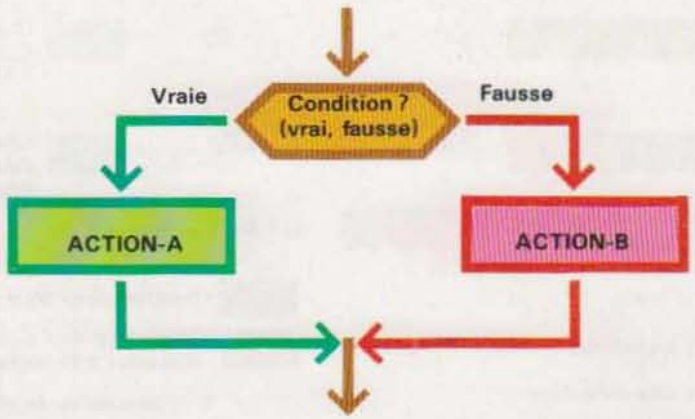
Observons que THEN ne fait pas partie de l'ANS Cobol. Donc nous nous bornerons à l'utiliser uniquement dans les premiers exemples, pour rendre la construction logique de la phrase entière plus évidente.

Nous suivrons, par la suite, les règles du Cobol standard de plus près.

Mentionnons également l'importance du point (.) dans une phrase conditionnelle. Oublié ou inséré de manière non appropriée, le point sera mal interprété par le compilateur.

Dans ce cas, non seulement les résultats sont complètement inattendus, mais encore le déroulement lui-même du calcul est imprévisible ; par exemple fin pour erreur, boucles infinies, dépassement des limites de l'espace mémoire assigné, etc.

TEST LOGIQUE DE L'INSTRUCTION IF



FORMAT GENERAL DE L'INSTRUCTION IF

```
IF condition [THEN]  { NEXT SENTENCE       }  [ ELSE  { NEXT SENTENCE       } ]
                     { phrase-impérative-1 }         { phrase-impérative-2 }
```

Avant de procéder à son analyse, il est bon de rappeler combien, en langage naturel, les phrases conditionnelles constituent des points de repère indispensables à la compréhension d'un organigramme.
Lorsque vous relisez un programme écrit par vous-même ou par une autre personne, l'étape essentielle consiste à repérer les points décisionnels, les points de branchement et les conditions posées.
En effet, si le contexte logique dans lequel une certaine instruction s'exécute n'est pas bien analysé, il devient, par avance, inutile d'entrer dans le détail du programme.

Lire un programme équivaut donc à lire les IF qu'il contient.

De ce fait, les points de décision du programme doivent être rédigés de manière à accrocher le regard dès la première lecture.
Le Cobol permet d'écrire la plupart des instructions tout en respectant ses règles syntaxiques. Et la seule disposition des opérandes devrait donner une idée de leur fonction logique.
Tel qu'il est analysé à la page précédente, et tel qu'il est explicité dans l'encadré ci-dessus, le format de l'instruction IF est difficile à interpréter. Il est donc vivement recommandé de ne l'utiliser qu'avec des clauses détaillées :
De cette façon :

```
[THEN]  { NEXT SENTENCE       }
        { phrase-impérative-1 }

[ ELSE  { NEXT SENTENCE       } ]
        { phrase-impérative-2 }
```

- on fait ressortir la condition dans le contexte du programme,
- on isole clairement les instructions qui représentent l'alternative offerte par le test.

Dans l'exemple suivant, les instructions IF ont une syntaxe correcte. Pourtant, à cause de leur disposition, l'alternative ne saute pas aux yeux.

```
IF TYPE-CARTE = 'A' MOVE CARTE
TO CARTE-DISPONIBLE
COMPUTE SOLDE = CREDIT-DEBIT
ADD SOLDE TO SOLDE-TOTAL
ADD 1 TO CARTES-LUES
GO TO LIRE-CARTES ELSE
ADD 1 TO CARTES-LUES
ADD 1 TO CARTES-DECOUVERTES
GO TO LIRE-CARTES.
```

Voici le même exemple programmé comme nous vous le conseillons :

```
IF TYPE-CARTE       = 'A'

   MOVE CARTE TO BAC-DISPONIBLE
   COMPUTE SOLDE = CREDIT-DEBIT
   ADD SOLDE      TO SOLDE-TOTAL
   ADD 1          TO CARTES-LUES
   GO TO          LIRE-CARTE

ELSE
   ADD 1          TO CARTES-LUES
                  CARTES-DECOUVERTES
   GO TO          LIRE-CARTES.
```

Sous cette forme, on peut facilement remonter à l'organigramme et, par conséquent, avoir une compréhension immédiate des fonctions dans leur déroulement.

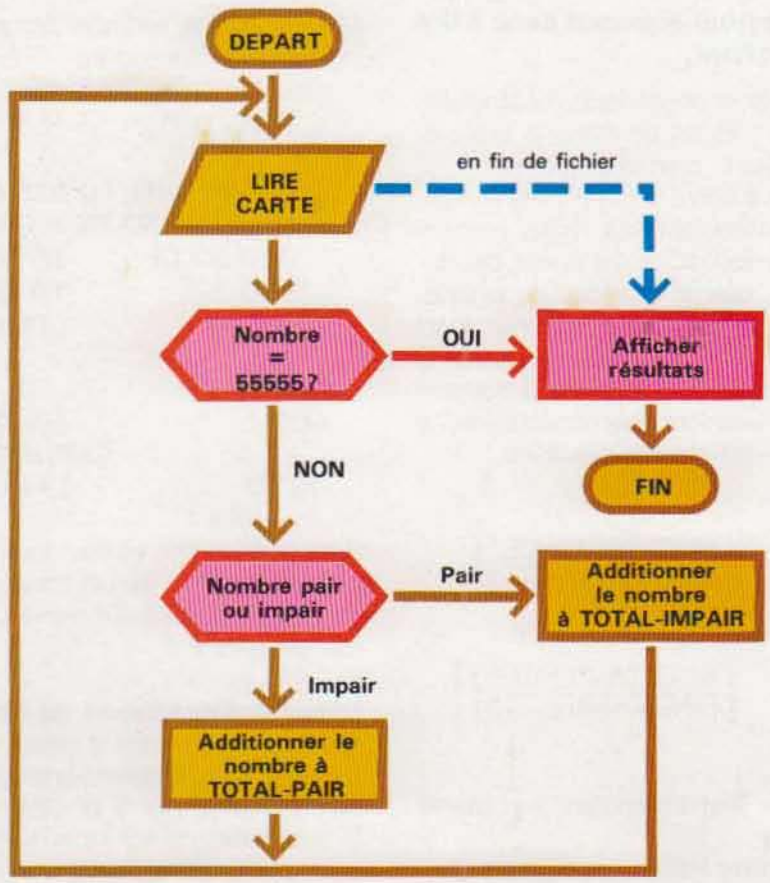
Logique d'exécution de l'instruction IF.
Avant d'entrer dans le détail des différentes analyses accompagnant la proposition IF, il est bon d'en examiner le déroulement.
Si la condition spécifiée immédiatement après la proposition IF se révèle vraie, la phrase qui la suit est exécutée jusqu'à ce qu'elle rencontre

le mot ELSE ou le premier point. Après exécution de la phrase, le contrôle passe à la deuxième phrase (la phrase après le 1er point). Si, au contraire, la condition est fausse, c'est la phrase qui suit immédiatement la proposition ELSE (quand elle existe) qui est exécutée et le contrôle passe ensuite à la première instruction suivant le point de fermeture de la phrase conditionnelle.

Voici quelques exemples :
Nous voulons calculer les sommes des nombres entiers, pairs et impairs, perforés (un nombre par carte) sur un fichier. Le calcul terminé les résultats obtenus sont affichés à l'écran dès que l'on rencontre, dans le fichier, une carte avec le nombre 55555. Il est évident que, pour effectuer la somme des nombres impairs de l'autre, il est nécessaire de reconnaître la nature du nombre qu'on vient de lire sur la carte.
Par ailleurs, avant de traiter le nombre lu, il faut vérifier qu'il n'est pas égal à 55555. A première vue, le programme suit l'organigramme ci-dessous. On divise un nombre par 2 pour connaître sa parité : si le reste de la division est 0, le nombre est pair, sinon impair. Le programme décrit peut donc être détaillé tel qu'il appartient dans le tableau ci-contre. A sa lecture, on observe que si la condition NOMBRE = 55555 s'avère vraie, on exécute, l'instruction GO TO FIN-CALCUL, alors que, toutes les fois où elle est fausse, le programme passe à l'instruction suivant le point :

```
DIVIDE    NOMBRE BY 2
          GIVING QUOTIENT
          REMAINDER RESTE
```

EXEMPLE DE PHRASES CONDITIONNELLES



APPLICATION DE L'INSTRUCTION IF

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ESSAI-IF.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. ..............
 OBJECT-COMPUTER. ..............
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT CARTES ASSIGN TO CARD-READER CARTES.
*
*
*
 DATE DIVISION.
 FILE SECTION.
 FD  CARTES
     LABEL RECORD OMITTED.
 01  CARTE                    PIC X(80).
*
*
 WORKING STORAGE SECTION.
 01  CARTE-WS.
     05  NUMERO               PIC 9(5) COMP.
     05  FILLER               PIC X(75).
*
 01  QUOTIENT                 PIC 9(5)V99 COMP.
 01  RESTE                    PIC 9(2)    COMP.
 01  ADDITIONNER-PAIRS        PIC 9(10)   COMP VALUE 0.
 01  ADDITIONNER-IMPAIRS      PIC 9(10)   COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT SECTION.
 OUVRIR-FICHIER.
     OPEN INPUT CARTES.
 LIRE-CARTE.
     READ CARTES INTO CARTE-WS
                 AT END
                 GO TO FIN-CALCUL.
     IF  NOMBRE = 55555
         GO TO FIN-CALCUL.
     DIVISE NOMBRE BY 2
         GIVING QUOTIENT
         REMAINDER RESTE.
     IF  RESTE = 0
         ADD NOMBRE TO ADDITIONNER-PAIRS
     ELSE
         ADD NOMBRE TO ADDITIONNER-IMPAIRS.
     GO TO LIRE-CARTE.
*
*
*
 FIN-CALCUL.
     DISPLAY '** ADDITIONNER NOMBRES PAIRS = ' ADDITIONNER-PAIRS
             UPON CONSOLE.
     DISPLAY ' '
             UPON CONSOLE.
     DISPLAY '** ADDITIONNER NOMBRES IMPAIRS = ' ADDITIONNER-IMPAIRS
             UPON CONSOLE.
 FERMER-FICHIER.
     CLOSE CARTES.
 FIN.
     DISPLAY '*** CALCUL TERMINE ***'
             UPON CONSOLE.
     STOP RUN.
```

Après la division du nombre par 2, le reste est égal à 0 ou ne l'est pas. Dans le 1er cas, le nombre (pair) est ajouté à ADDITION-PAIR, sinon (ELSE) il est ajouté à ADDITION-IMPAIR. La phrase conditionnelle se termine avec le point de fin d'instruction :

```
ADD NOMBRE TO ADDITION-IMPAIR
```

Quelle que soit la décision prise par le programme dans le test conditionnel, la main est toujours cédée à l'instruction suivant immédiatement la phrase conditionnelle elle-même. Une fois l'analyse sur la valeur du reste (RESTE) terminée et l'instruction correspondante exécutée, le programme retourne au paragraphe LIRE-CARTE.
Ce cycle est exécuté jusqu'à épuisement du fichier (AT END) ou jusqu'à lecture d'une carte contenant 55555.
A propos de l'importance du point dans l'exacte définition d'une phrase conditionnelle, on observe que si le second IF avait été mal écrit, nous aurions :

```
IF RESTE = 0
    ADD NOMBRE TO ADDITION-PAIR
ELSE
    ADD NOMBRE TO ADDITION-IMPAIR
GO TO LIRE-CARTE.
```

(autrement dit si le point avait été omis dans l'instruction ADD NOMBRE TO ADDITION-IMPAIR), le compilateur aurait interprété la séquence des opérations comme page 1057.
Le programme n'aurait relu les cartes que dans le cas où le nombre traité précédemment avait été impair. Une fois le premier nombre pair rencontré, et après l'incrémentation de ADDITION-PAIR, le programme serait passé au paragraphe FIN-CALCUL avec comme conséquence l'arrêt des calculs !

L'option NEXT SENTENCE. Elle permet de transférer le contrôle des opérations à la première instruction suivant immédiatement le point de fermeture de la phrase conditionnelle. Pour éclairer ce concept, considérons à nouveau le premier IF de l'exemple précédent :

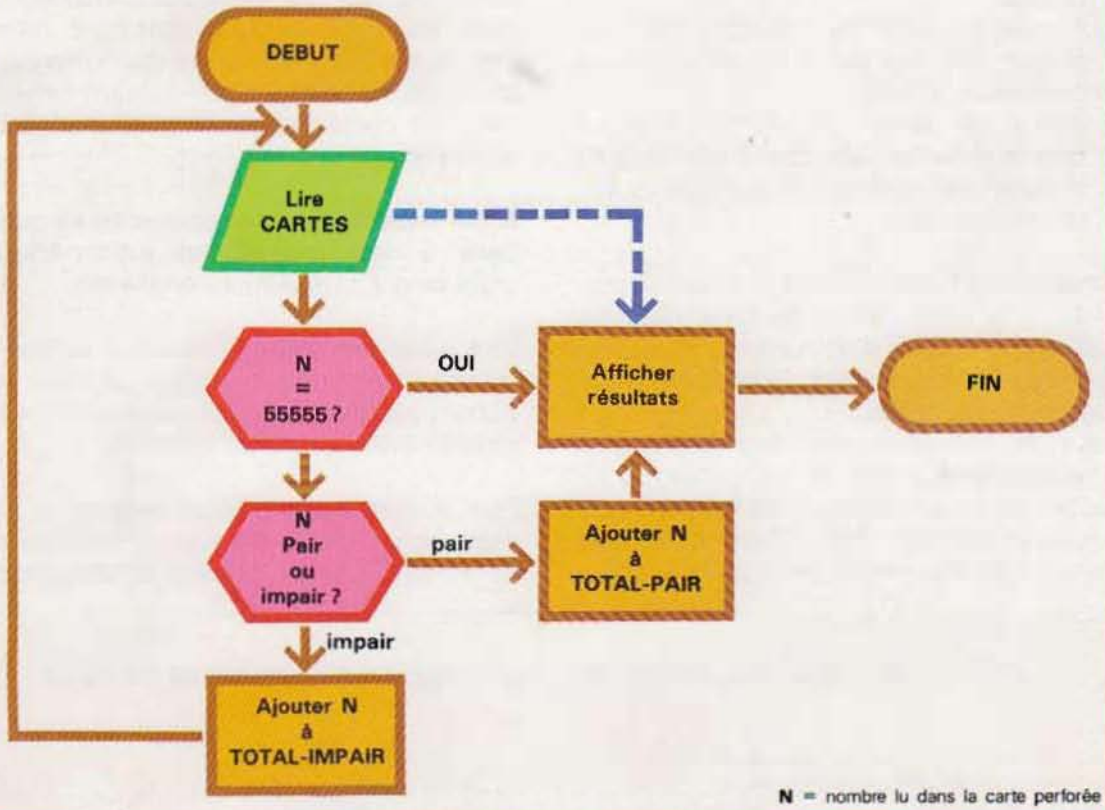
```
IF NOMBRE = 55555
  GO TO FIN-CALCUL.
DIVIDE . . . . . .
. . . . . .
```

Test numérique d'un afficheur à cristaux liquides LCD (Liquid Cristal Display).



M. Wolf/Graza Neri-Black Star

INTERPRETATION ERRONEE D'UNE SEQUENCE CONDITIONNELLE



On aurait obtenu le même résultat en écrivant :

```
IF NOMBRE NOT = 55555
   NEXT SENTENCE
ELSE
   GO TO FIN-TRAITEMENT.
DIVIDE .....
```

Chaque fois que le contenu du champ NOMBRE est différent de 55555, la main est donnée à l'instruction immédiatement après le point, à savoir :

```
DIVIDE NOMBRE BY 2
     GIVING QUOTIENT
     REMAINDER RESTE.
```

Dans le cas contraire, c'est GO TO FIN-TRAITEMENT qui est exécutée.

Bien que les deux formes soient équivalentes quant au résultat, la seconde est d'une lecture moins immédiate que la première.
De là l'avantage de structurer les IF (si) d'une manière simple et linéaire.
Cela rend plus facile la rédaction et la lecture du programme et surtout la maintenance.

Conditions liées à l'instruction IF. Dans le format général de l'instruction IF, la nature des conditions posées n'a pas encore été précisée pas plus que la phrase conditionnelle.

Il existe, en effet, plusieurs tests conditionnels applicables au IF qui peuvent être résumées en 4 types :

- Analyse de relation : elle met une condition à l'exécution d'une section déterminée du programme portant sur la comparaison (plus grand, égal, plus petit, etc.). Elle revêt un caractère particulier quand il s'agit de données alphabétiques ou alphanumériques.

- Analyse de signe : elle subordonne un traitement à la nature du signe (+ ou -) de la donnée.
- Analyse de condition : l'exécution de l'instruction est soumise à la vérification du contenu du champ.
- Analyse de classe : elle vérifie la classe à laquelle appartient une donnée (numérique / alphabétique) avant de lancer le type de traitement approprié.

Analyse de relation. Dans ce mode, on procède à la comparaison de deux quantités (opérandes) à l'aide d'un opérateur relationnel spécifique (voir ci-dessous, 1er encadré, le format de la proposition IF).
Les opérateurs relationnels (résumés en bas de page) qualifient le type de comparaison à effectuer sur les opérandes. Tous les opérateurs s'écrivent indifféremment à l'aide du symbole correspondant ou en clair (en anglais). Les opérandes (objets de la comparaison) sont des champs décrits dans la division donnée, des constantes ou des expressions mathématiques. Toutes les analyses sont donc autorisées, quelle que soit la nature des opérandes : un champ peut être comparé à un autre champ avec une constante ou avec une expression arithmétique.

Mais attention : une constante se compare à une expression arithmétique mais non à une autre constante.

Lorsqu'une des deux opérandes à comparer est une expression arithmétique, sa valeur est d'abord calculée, et ensuite seulement mise en parallèle avec la seconde opérande.

Pour illustrer notre propos, prenons le cas d'une lecture. Nous disposons d'un fichier de cartes dont la forme de l'enregistrement est :

FORMAT DE L'INSTRUCTION IF DANS UNE ANALYSE DE RELATION

```
IF   { nom-donnée-1 }   opérateur   { nom-donnée-2 }
     { constante-1  }               { constante-2  }
     { expression-1 }               { expression-2 }

[THEN]
     { NEXT SENTENCE }
     { phrase-1      }

[ ELSE
     { NEXT SENTENCE }
     { phrase-2      }
```

OPERATEURS RATIONNELS

Symbole Cobol	Equivalent mathématique	Forme complète	Signification
=	=	IS EQUAL [TO]	... égal ...
NOT =	≠	IS NOT EQUAL [TO]	... différent ...
>	>	IS GREATER [THAN]	... supérieur ...
<	<	IS LESS [THAN]	... inférieur ...
NOT >	⩽	IS NOT GREATER [THAN]	... inférieur ou égal ...
NOT <	⩾	IS NOT LESS [THAN]	... supérieur ou égal ...



Finsiel

Système IBM 370 dans une application de gestion d'archives stockées sur bandes magnétiques.

```
01 CARTE-WS.
   05 TYPE-CARTE          PIC X(3).
   05 ARTICLE.
      10 SERIE            PIC X(3).
      10 NUMERO           PIC 9(5).
      10 DESCRIPTION      PIC X(30).
   05 STOCK               PIC 9(5).
   05 SEUIL-MINIMUM       PIC 9(3).
   05 VENTE               PIC 9(5).
   05 VENTES-MOIS         PIC 9(2).
```

Supposons que chaque carte lue doive être soumise aux contrôles suivants :

1/ La carte est égale à MAG. Sinon la rejeter en incrémentant un compteur de cartes lues (à afficher en fin de traitement).

2/ Les descriptions de l'article ne sont pas vides. Sinon rejeter la carte sans oublier d'afficher, après totalisation, le nombre de cartes écartées en fin de traitement.

3/ Le stock (quantité réelle) n'est pas inférieur au seuil d'approvisionnement minimum (pour éviter d'être en rupture de stock »). Sinon, signaler l'événement par un message approprié (STOCK D'ALARME).

4/ Totaliser les articles mouvementés dans le mois de référence indiqué par l'opérateur de saisie.

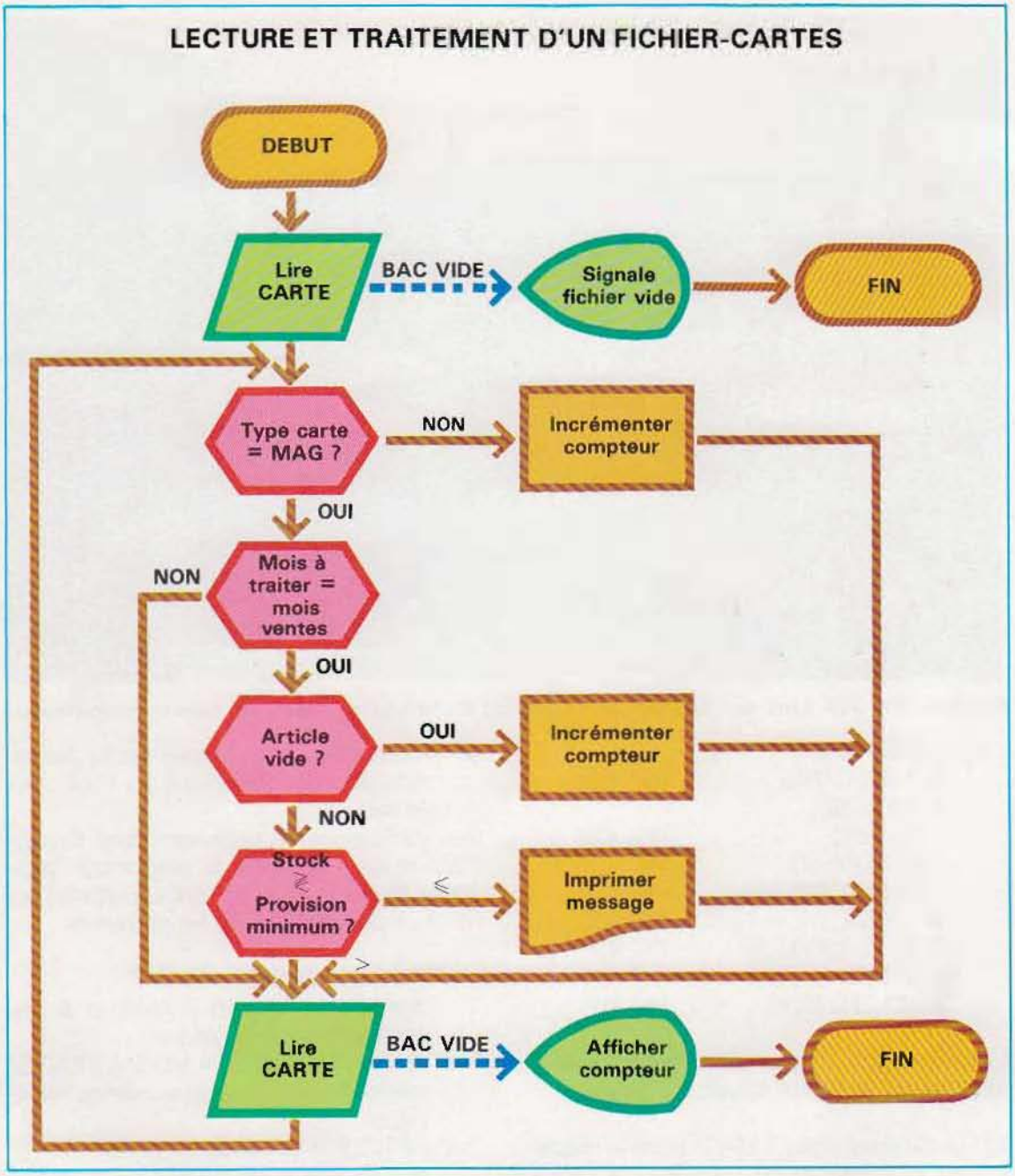
Ces vérifications se retrouvent dans la page 1060 et dans le listing du programme (page 1061). Plusieurs types d'IF ont été utilisés pour l'analyse de relation entre les opérandes :

IF TYPE-CARTE NOT = 'MAG'
compare un champ numérique à une constante alphanumérique.

IF VENTES-MOIS NOT = MOIS-A-TRAITER
vérifie l'égalité entre deux champs numériques.

IF ARTICLE = SPACES
met en relation et vérifie l'égalité entre un champ numérique et la constante figurative SPACES. Cela équivaut à vérifier si le contenu entier du champ est composé d'espaces.

IF STOCK-VENTES < SEUIL-MINIMUM
compare le résultat de l'expression arithmétique STOCK-VENTES avec le contenu du champ numérique SEUIL-MINIMUM (stock d'alarme, dans l'organigramme).



L'analyse de relation s'effectue aussi bien entre champs de même classe (alphanumériques, numériques) qu'entre champs différents.
Une bonne connaissance des règles de comparaison est essentielle pour écrire un programme, et ainsi éviter certaines erreurs qui n'apparaissent, étant donné le nombre de comparaisons, qu'en phase de compilation.
A ce niveau, ou bien les résultats sont faux, ou bien le calcul s'achève d'une manière inattendue.

Soit la phrase conditionnelle :

```
IF SW-ACCORD = '0'
   PERFORM TRAITEMENT
ELSE
   PERFORM FIN-TRAITEMENT.
```

LECTURE ET TRAITEMENT D'UN FICHIER CARTES

```
01  CARTE-WS.
    05  TYPE-CARTE                      PIC X(3).
    05  ARTICLE.
        10  SERIES                      PIC X(3).
        10  NUMERO                      PIC 9(5).
        10  DESCRIPTION                 PIC X(30).
    05  STOCK                           PIC 9(5).
    05  SEUIL-MINIMUN                   PIC 9(3).
    05  VENTES                          PIC 9(5).
    05  VENTES-MOIS                     PIC 9(2).
*
01  CALCUL-MOIS                         PIC 9(2).
*
01  CARTES-LUES                         PIC 9(3) COMP VALUE 0.
01  CARTES-INCORRECTES                  PIC 9(3) COMP VALUE 0.
01  CARTES-ECARTEES                     PIC 9(3) COMP VALUE 0.

PROCEDURE DIVISION.
DEBUT.
    MOVE 0 TO CARTES-ECARTEES
              CARTES-INCORRECTES
              CARTES-LUES.
    DISPLAY '** ENTRER MOIS A TRAITER **'
            UPON CONSOLE.
    ACCEPT CALCUL-MOIS FROM CONSOLE.
OUVRIR-FICHIER.
    OPEN INPUT CARTES.
PREMIERE-LECTURE.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '** FICHIER CARTES VIDE **'
                        UPON CONSOLE
                GO TO FIN-CALCUL.
CONTROLE.
    ADD 1       TO CARTES-LUES.
    IF  TYPE-CARTE NOT = 'MAG'
        ADD 1       TO CARTES-ECARTEES
        GO TO LIRE-CARTES.
    IF  VENTES-MOIS NOT = CALCUL-MOIS
        GO TO LIRE-CARTES.
    IF  ARTICLE = SPACES
        ADD 1       TO CARTES-INCORRECTES
        GO TO LIRE-CARTES.
    IF  STOCK - VENTES LESS THAN SEUIL-MINIMUM
        DISPLAY '** ARTICLE : '
                DESCRIPTION
                '  * SERIE  : '
                SERIE
                '  * NUMERO : '
                NUMERO
                '  * SOUS SEUIL **'
                UPON PRINTER.
LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '***      CARTES LUES        ='
                        CARTES-LUES
                        UPON PRINTER
                DISPLAY '***      CARTES ECARTEES ='
                        CARTES-ECARTEES
                        UPON PRINTER
                DISPLAY '***      CARTES INCORRECTES ='
                        CARTES-INCORRECTES
                        UPON PRINTER
                GO TO FIN-CALCUL.
    GO TO CONTROLE.
*
FIN-CALCUL.
    CLOSE CARTES.
    STOP RUN.
```

avec SW-ACCORD décrit ainsi dans la WORKING-STORAGE SECTION :

```
01 SW-ACCORD*          PIC X(3)
```

Si, à un certain moment, SW-ACCORD contient la valeur non numérique :

0	0	0

l'analyse de relation IF SW-ACCORD = '0' sera fausse, et le programme effectuera l'appel à la procédure FIN-TRAITEMENT.
Ce résultat ne s'explique que si l'on a bien compris le mécanisme qui règle la comparaison. La proposition IF permet de mettre en relation des champs non numériques ou des champs numériques et non numériques, quelle que soit leur longueur.
Considérons, pour le moment, des champs de même longueur.
Dans ce cas, la comparaison est faite caractère par caractère en partant de la gauche jusqu'à l'épuisement des caractères à analyser ou jusqu'au rejet de la condition exigée (non satisfaite).
Soit, par exemple, CHAMP-1 et CHAMP-2 décrits :

```
01 CHAMP-1          PIC X(3).
02 CHAMP-2          PIC X(3).
```

Posons les valeurs :

CHAMP 1 =

A	B	C

CHAMP 2 =

A	B	D

Nous voulons vérifier la condition d'égalité suivante :

```
IF CHAMP-1 = CHAMP-2
   DISPLAY'LES CHAMPS SONT EGAUX'
           UPON CONSOLE
ELSE
   DISPLAY'LES CHAMPS NE SONT PAS
   EGAUX'
           UPON CONSOLE
   STOP RUN.
```

Le mécanisme de comparaison est schématisé dans le graphique de la page ci-contre.
Examinons maintenant le cas où l'on veut vérifier si le contenu du CHAMP-1 est plus petit que le contenu du CHAMP-2.
Nous entreprendrons, par conséquent, les actions décrites dans la phrase suivante :

```
IF CHAMP-1   CHAMP-2
   DISPLAY 'CHAMP-1 PLUS PETIT
           QUE CHAMP-2'
           UPON CONSOLE
ELSE
   DISPLAY 'CHAMP-1 PLUS GRAND OU
           EGAL AU CHAMP-2'
           UPON CONSOLE
STOP RUN.
```

Dans ce cas, le programme s'assure avant tout que la longueur des deux champs est égale ; il procède ensuite à la comparaison caractère par caractère en partant de la gauche (comme dans l'exemple précédent).
Le critère, qui permet de comparer les valeurs respectives de deux caractères, dépend de la manière dont ils sont hiérarchisés à l'intérieur du code utilisé par la machine (ASCII ou EBCDIC).
Ainsi, le résultat de la comparaison entre les deux valeurs A B Z et A B 9 est :

ABZ AB9 en EBCDIC
et
ABZ AB9 en ASCII.

Dans le premier langage, les représentations des lettres de l'alphabet ont toutes des valeurs inférieures à celles des nombres, contrairement au code ASCII (voir tableau ci-contre).
Quand les opérandes d'une proposition IF n'ont pas la même longueur, la comparaison s'effectue comme si l'opérande inférieure était remplie de blancs sur sa droite.
A ce propos, considérons à nouveau l'exemple déjà étudié :

```
IF SW-ACCORD = '0'
   PERFORM TRAITEMENT
ELSE
   PERFORM FIN-TRAITEMENT.
```

avec, pour contenu de SW-ACCORD, au moment de l'analyse :

COMPARAISON ENTRE DEUX CHAMPS ALPHANUMERIQUES

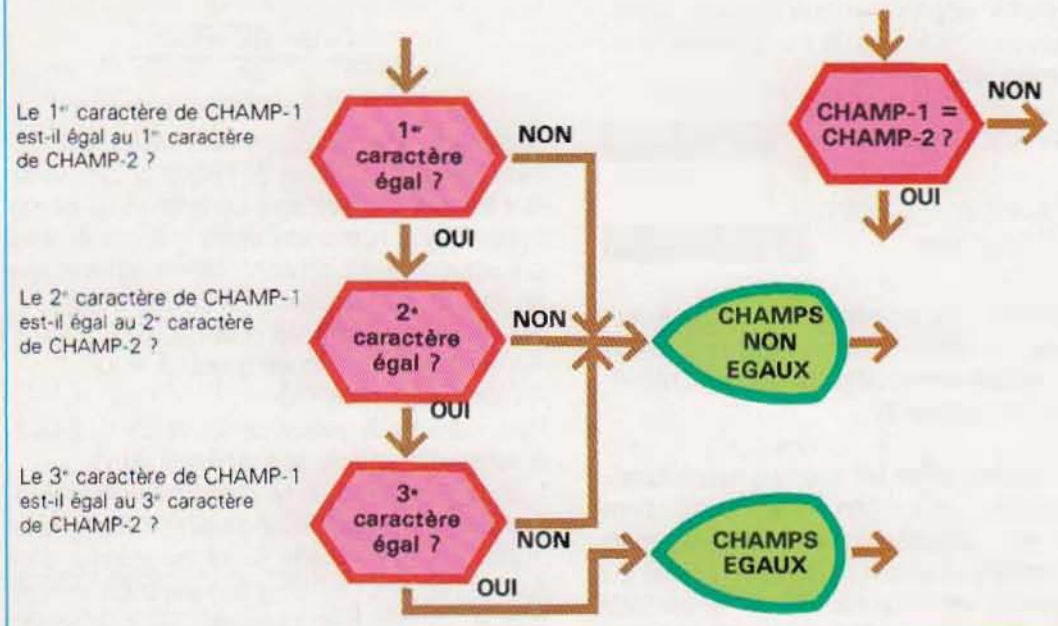


TABLEAU COMPARATIF DES CODES EBCDIC ET ASCII

Caractères EBCDIC		Caractères ASCII	
(valeurs inférieures)		**(valeurs inférieures)**	
b	blanc	b	blanc
.	point ou point décimal	"	guillemets
<	inférieur	$	dollar
(	parenthèse ouvrante	'	apostrophe
+	plus	(	parenthèse ouvrante
$	dollar	)	parenthèse fermante
*	astérisque	*	astérisque
)	parenthèse fermante	+	plus
;	point-virgule	,	virgule
–	moins ou trait d'union	–	moins ou trait d'union
/	barre de fraction	.	point ou point décimal
,	virgule	/	barre de fraction
>	supérieur	0-9	chiffres de 0 à 9
'	apostrophe	;	point-virgule
=	égal	<	inférieur
"	guillemets	=	égal
A-Z	lettres de A à Z	>	supérieur
0-9	chiffres de 0 à 9	A-Z	lettres de A à Z
(valeurs supérieures)		**(valeurs supérieures)**	

0	0	0

La constante alphanumérique 0 étant constituée d'un seul caractère, la comparaison s'effectue entre les champs :

SW-ACCORD	0	0	0

et constante '0' étendue à 3 caractères

0	0	0

Comme on le voit, et indépendamment du code utilisé, les deux valeurs ne seront jamais égales. Le programme appellera donc la procédure FIN-TRAITEMENT.

La comparaison entre les champs numériques, contrairement aux non-numériques, tient compte non des champs eux-mêmes, mais seulement de leur valeur algébrique. Toutes les règles mathématiques sont donc respectées dans la comparaison entre champs numériques. Il en résulte, par exemple (notation américaine), que :

+0000005.0000		vaut 5
- 425	est plus grand que	- 728
+12	est plus grand que	- 65

Analyse de signe. La proposition IF est capable d'analyser le signe du contenu d'un champ numérique ou du résultat d'une expression arithmétique en utilisant le format représenté dans le tableau ci-dessous.

Soit l'équation du second degré :

$$Ax^2+Bx+C=0$$

dont il faut extraire les racines. Les solutions de l'équation sont :

$$x_1=\frac{-B-\sqrt{B^2-4AC}}{2A}$$

$$x_2=\frac{-B+\sqrt{B^2-4AC}}{2A}$$

Supposons que nous voulions seulement calculer les racines réelles de l'équation, en signalant les cas où elles sont complexes et en désignant par Δ **(discriminant)** le terme B^2-4AC qui apparaît sous la racine carrée, on sait que les racines de l'équation du 2e degré sont :
— réelles et distinctes quand $\Delta>0$
— réelles et identiques quand $\Delta = 0$
— complexes si $\Delta<0$
Pour calculer la valeur de x_1 et de x_2 il faut, d'autre part, que A soit différent de 0.
L'organigramme d'un tel programme est représenté dans la page ci-contre et son listing page 1066. Rappelons que la racine carrée d'un nombre équivaut à l'élévation à la puissance du même nombre avec exposant 1/2 (0,5), soit :

$$\sqrt{B^2-4AC}=\sqrt{DELTA}=(DELTA)^{1/2}=$$
$$=(DELTA)^{0.5}=DELTA**0.5$$

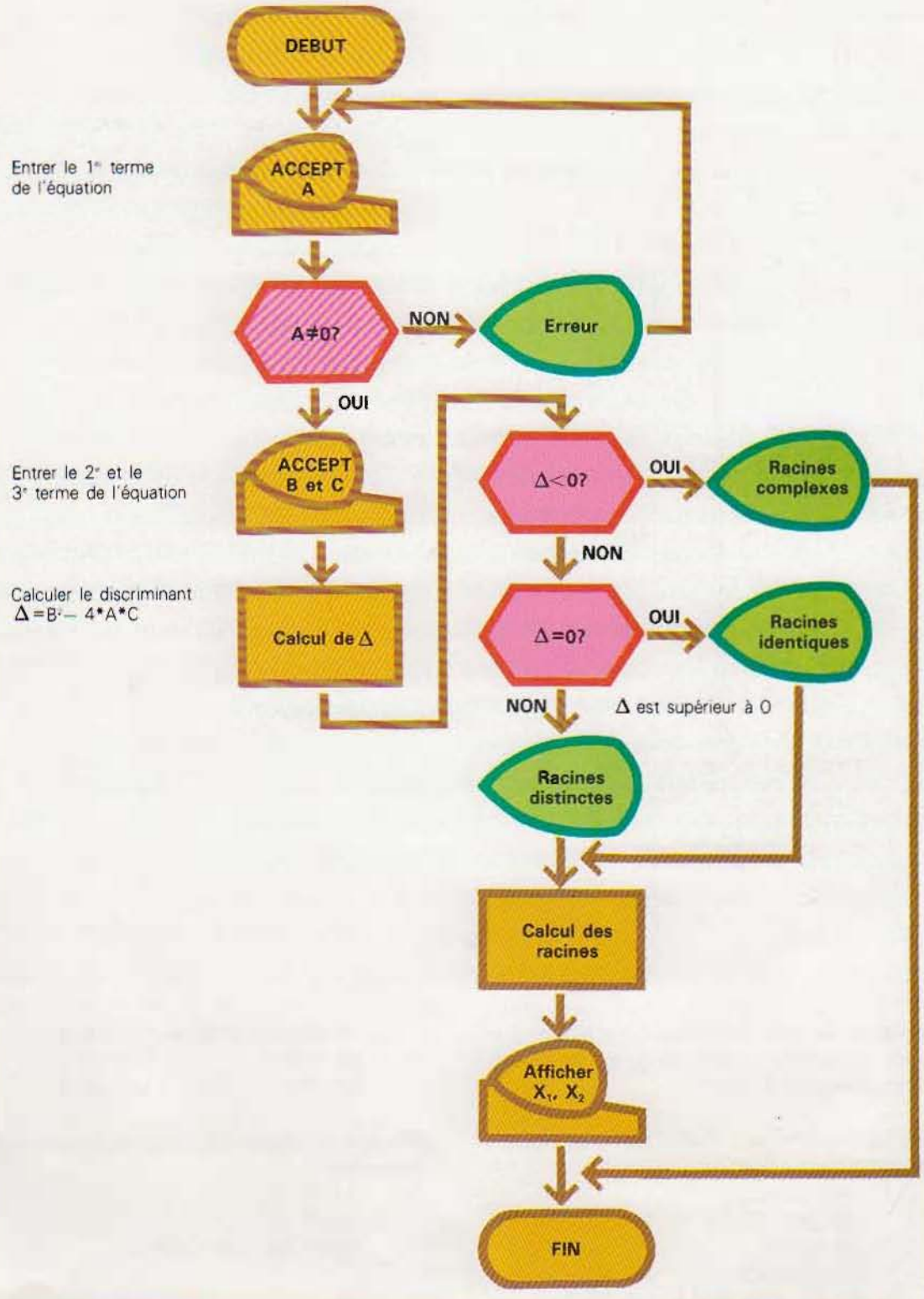
Notons que la valeur de DELTA est calculée en dehors de la proposition pour des raisons de rapidité (un calcul au lieu de trois) ; l'analyse de signe vient après.
Pour conclure, notons qu'une analyse de signe peut se ramener à une analyse de relation en comparant la valeur du champ ou de l'expression à 0.

Analyse de condition. La fonction du niveau 88 a été étudiée lors de la description des noms des champs de la division données. Rappelons-nous, dans ce contexte, que si un champ a été décrit avec une ou plusieurs définitions de niveau 88, le contenu peut être analysé en se référant directement à elles.

FORMAT DE L'INSTRUCTION IF POUR L'ANALYSE DU SIGNE

IF { nom-donnée / expression-arithmétique } IS [NOT] { POSITIVE / NEGATIVE / ZERO }

CALCUL DES RACINES D'UNE EQUATION DU SECOND DEGRE



CALCUL DES RACINES D'UNE EQUATION DU SECOND DEGRE

```
  01  A                          PIC S9(6)V9(3)  COMP.
  01  B                          PIC S9(6)V9(3)  COMP.
  01  C                          PIC S9(6)V9(3)  COMP.
  01  DELTA                      PIC S9(6)V9(3)  COMP.
  01  X1                         PIC S9(6)V9(3)  COMP.
  01  X2                         PIC S9(6)V9(3)  COMP.
*
*
  PROCEDURE DIVISION.
  DEBUT.
      DISPLAY ' CALCUL EQUATION 2.EME DEGRE'
              UPON CONSOLE.
  ACCEPTER-A.
      DISPLAY '* ENTRER PREMIER COEFFICIENT A = '
               UPON CONSOLE.
      ACCEPT A FROM CONSOLE.
      IF  A  IS EQUAL ZERO
          DISPLAY '*** ERREUR ***'
                  ' COEFFICIENT A DOIT ETRE DIFFERENT DE 0'
                  UPON CONSOLE.
                  GO TO ACCEPTER-A.
      DISPLAY '* ENTRER SECOND COEFFICIENT B = '
              UPON CONSOLE.
      ACCEPT B FROM CONSOLE.
      DISPLAY '* ENTRER TROISIEME COEFFICIENT C ='
           UPON CONSOLE.
  ACCEPT C FROM CONSOLE.
  COMPUTE DELTA = B ** 2 - 4 * A * C.
  IF  DELTA IS NEGATIVE
      DISPLAY '* DELTA NEGATIF *
              '* RACINES COMPLEXES *'
              UPON CONSOLE
      GO TO FIN.
  IF  DELTA IS ZERO
      DISPLAY '* DELTA NUL *'
              '* RACINES IDENTIQUES *'
              UPON CONSOLE.
  IF  DELTA IS POSITIVE
      DISPLAY '* DELTA POSITIF *'
              '* RACINES REELLES ET DISTINCTES *'
               UPON CONSOLE.
    COMPUTE X1 = (0 - B - DELTA ** .5) / 2 * A.
    COMPUTE X2 = (0 - B + DELTA ** .5) / 2 * A.
    DISPLAY '** X1 = ' X1
            '     '
            '** X2 = ' X2
            UPON CONSOLE.
FIN.
    DISPLAY 'FIN CALCUL'
            UPON CONSOLE.
    STOP RUN.
```

Les deux exemples qui suivent mettent en évidence l'avantage qui découle du recours à cette procédure du Cobol.

Soit l'enregistrement EMPLOYE ainsi décrit :

```
01 EMPLOYE.
   08 MATRICULE        PIC 9(5).
   05 NOM              PIC X(30).
   05 ETAT-CIVIL       PIC 9.
   88 CELIBATAIRE-F    VALUE 1.
   88 CELIBATAIRE-H    VALUE 2.
   88 MARIE            VALUE 3.
   88 VEUF             VALUE 4.
```

A l'évidence, dans le contexte du programme, l'instruction :

```
IF VEUF
   PERFORM TRAITEMENT.
```

s'interprète plus rapidement que la suivante :

```
IF ETAT-CIVIL = 4
   PERFORM TRAITEMENT
```

A travers les exemples cités, on voit que l'analyse de condition équivaut à vérifier, dans une forme familière, la signification logique de l'événement lié à un contenu déterminé du champ.
L'avantage est encore plus frappant quand ces conditions sont très nombreuses et chaînées en séquence.

Considérons l'opération de lecture d'un fichier de cartes.
Les articles décrits sur chaque fiche sont codés au moyen de deux caractères, une lettre suivie d'un chiffre compris entre 4 et 9.
La validité de chaque carte lue passe par la vérification de toutes les combinaisons possibles.
La solution devient, en revanche, plus simple si on adopte la description suivante, dans laquelle la liste de tous les caractères alphabétiques (VALUE 'A' THRU 'Z') est associée à SERIE-VALABLE et les chiffres allant de 4 à 9 (VALUE 4 THRU 9) à NOMBRE-VALABLE :

```
01 CARTE-ARTICLE.
   05 CODE-ARTICLE.
      10 SERIE-ARTICLE          PIC X.
         88 SERIE-VALABLE       VALUE 'A'
                               THRU 'Z'.
      10 NOMBRE-ARTICLE         PIC 9.
         88 NOMBRE-VALABLE      VALUE 4
                               THRU 9.
```

L'instruction, pour le contrôle du CODE-ARTICLE, s'écrit donc :

```
IF NOT SERIE-VALABLE
   PERFORM ERREUR-SERIE
   GO TO LIRE-CARTES.
IF NOT NOMBRE-VALABLE
   PERFORM ERREUR-NOMBRE
   GO TO LIRE-CARTES.
```

Analyse de classe. Grâce à ce dernier type d'analyse associé à l'instruction IF, il est possible d'examiner si le contenu d'un certain champ est, à un moment déterminé, de classe numérique ou alphanumérique (voir format ci-dessous).

L'analyse nécessite le respect de certaines règles intuitives :
- Dans un champ alphanumérique (PIC X), effectuer un test pour évaluer si son contenu est numérique ou alphabétique et a un sens.

- Inversement, vérifier si un champ alphabétique (PIC A), contenant des nombres, est une erreur logique et syntaxique.

Ces tests sont résumés dans le tableau qui se trouve en bas de cette page.

FORME DE L'INSTRUCTION IF POUR L'ANALYSE DE CLASSE

```
IF nom-donnée  IS [NOT] { NUMERIC    }
                        { ALPHABETIC }
```

ANALYSE DE CLASSE

PICTURE	[NOT] NUMERIC	[NOT] ALPHABETIC
A (Alphabétique)	NON	OUI
9 (Numérique)	OUI	NON
X (Alphanumérique)	OUI	OUI

N'oublions pas que le champ examiné doit être implicitement ou explicitement USAGE DISPLAY.

Emploi des opérateurs logiques. Tous les tests que nous avons examinés jusqu'à présent portent sur un événement unique. Il s'agissait de conditions simples. En fait, l'action à déclencher est souvent liée au résultat de plusieurs analyses subordonnées de différents types et réalisées par un nombre quelconque de variables.
L'enchaînement logique des conditions est obtenu au moyen d'opérateurs logiques propres à l'algèbre booléenne : AND, OR, NOT, (ET, OU, NON).

Quand une décision est subordonnée au test simultané de deux ou de plusieurs événements, l'analyse peut être conduite en utilisant AND avec un seul IF.

La phrase conditionnelle est vraie quand les deux conditions s'avèrent également vraies.
Dans l'exemple :

```
01 CHAMP-1      PIC 9 VALUE 1.
02 CHAMP-2      PIC 9 VALUE 0.
```

Pour vérifier, en cours d'exécution, si le contenu des deux champs est resté le même qu'en entrée, on peut écrire :

```
IF      CHAMP-1 = 1
        AND
        CHAMP-2 = 0
        PERFORM CHAMPS-OK
ELSE
```

L'appel à la procédure CHAMPS-OK est activé seulement si les deux champs vérifient simultanément les conditions demandées. Bien noter qu'il suffit qu'une condition ne soit pas validée pour que le contrôle passe aux instructions subordonnées à la clause ELSE.
On emploie AND très fréquemment pour constituer un fichier des enregistrements d'entrée sur la base de l'appartenance d'une donnée à un intervalle préalablement établi.

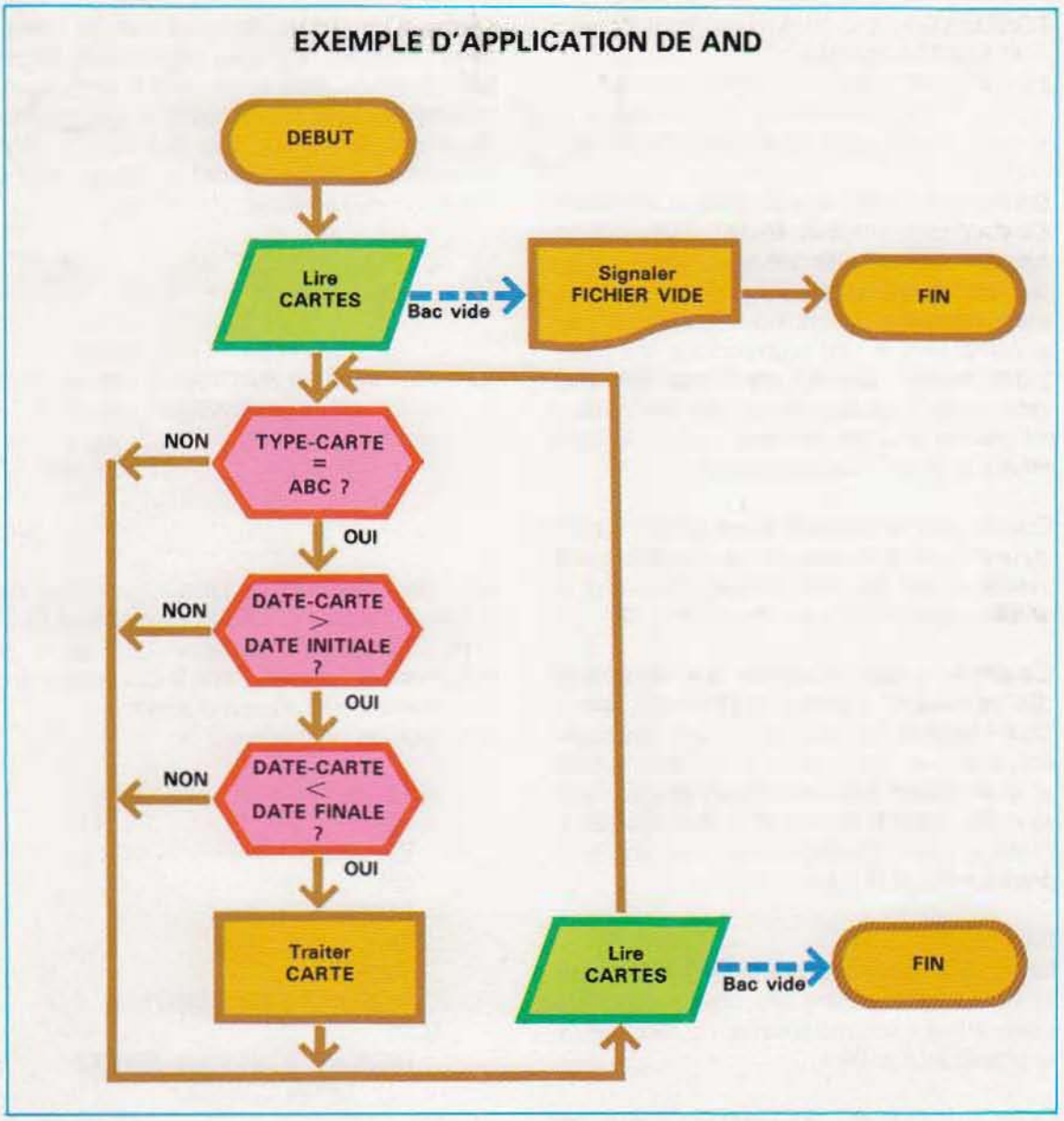
Soit le fichier CARTES suivant :

```
01 CARTE-WS.
   05 TYPE-CARTE              PIC X(3).
      88 CARTES-VALABLE VALUE 'ABC'.
   05 DATE-CARTE.
      10 ANNEE-CARTE          PIC 9(2).
      10 MOIS-CARTE           PIC 9(2).
      10 JOUR-CARTE           PIC 9(2).
   05
```

EXEMPLE D'APPLICATION DE AND

```
PROCEDURE DIVISION.
DEBUT.
    OPEN INPUT CARTES.
PREMIERE-LECTURE.
    READ  CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 CLOSE CARTES
                 STOP RUN.
CONTROLE.
    IF   CARTE-VALABLE
         AND
         DATE-CARTE > DATE-INITIALE
         AND
         DATE-CARTE < DATE-FINALE
         PERFORM CALCUL-CARTE.
    READ  CARTES INTO CARTE-WS
                  AT END
                  CLOSE CARTES
                  STOP RUN.
    GO TO CONTROLE.
*
*
*
```

EXEMPLE D'APPLICATION DE AND



Nous voulons ne traiter que les cartes de type 'ABC' et relatives à la période comprise (extrêmes exclus) entre :

```
01 DATE-INITIALE.
   05 ANNEE-INIT          PIC 99.
   05 MOIS-INIT           PIC 99.
   05 JOUR-INIT           PIC 99.
*
*
01 DATE-FINALE.
   05 ANNEE-FIN           PIC 99.
   05 MOIS-FIN            PIC 99.
   05 JOUR-FIN            PIC 99.
```

La phase de choix et de traitement est détaillée par le programme ci-contre (page 1068) et par l'organigramme ci-dessus.

Il suffit que la carte ne soit pas de type ABC, ou que l'une des deux limites de l'intervalle d'extraction ne soit pas respectée pour que la main soit donnée à l'instruction suivant immédiatement le point qui ferme IF.

Quel que soit le résultat de IF, on passera quand même par cette instruction, car l'opération READ n'est pas assujettie à la clause ELSE. La validité et la clarté de la solution proposée apparaissent mieux dans le détail du paragraphe suivant :

```
CONTROLE.
   IF CARTE-VALABLE
      IF DATE CARTE > DATE-INITIALE
         IF DATE-CARTE < DATE-FINALE
            PERFORM TRAITEMENT-CARTE.
```

Dans ce cas, nous avons eu recours à une série d'instructions IF imbriquées.
Adopter une telle codification, surtout dans des cas moins simples, comporte généralement un risque de moindre lisibilité du programme, ainsi qu'une augmentation des possibilités d'erreur. En effet, les IF imbriqués peuvent mener le programmeur vers des niveaux de plus en plus hiérarchisés, jusqu'à lui faire perdre le fil de l'analyse logique.

Quand, pour entreprendre une action, il suffit qu'une seule parmi une série de conditions, soit vérifiée, il est dès lors possible d'exprimer la phrase conditionnelle en recourant à OR.

L'exemple suivant constitue une application des opérateurs logiques AND et OR utilisés pour recenser les employés d'une entreprise non mariés et nés entre le 31 décembre 1940 et le 1er janvier 1960 inclus. Les données sont stockées dans le fichier de cartes ANAGRAPHIQUE (voir l'organigramme ci-contre et le programme de la page 1072).

Bien noter :
La signification de l'opérateur NOT est intuitive et son utilisation, dans une phrase conditionnelle, entraîne une modification du sens réel de la phrase tout entière.

Cette altération est pratiquement impossible dans le cas de conditions simples, n'indiquant la vérification que d'un seul événement.
Ainsi, la phrase :

```
IF A NOT = B
   PERFORM PREMIERE-ACTION
ELSE
   PERFORM SECONDE-ACTION.
```

ne comporte aucune ambiguïté et sa lecture est immédiate.
A l'inverse, la compréhension d'une phrase, avec une ou plusieurs conditions ne dépendant pas d'autres opérateurs logiques AND et OR, n'est pas immédiate. D'autant que ces opérateurs respectent les règles générales de l'algèbre booléenne. Sans entrer dans le vif du sujet, considérons tout de même l'exemple suivant. Nous devons lire le fichier EMPLOYES dans lequel chaque carte contient le champ relatif à l'état-civil de l'employé.

```
01 EMPLOYE.
   05 .....
   05 .....
   05 ETAT-CIVIL          PIC 9.
      88 CELIBATAIRE-H  VALUE 1.
      88 CELIBATAIRE-F  VALUE 2.
      88 MARIE          VALUE 3.
      88 VEUF           VALUE 4.
   05 .....
      .....
```

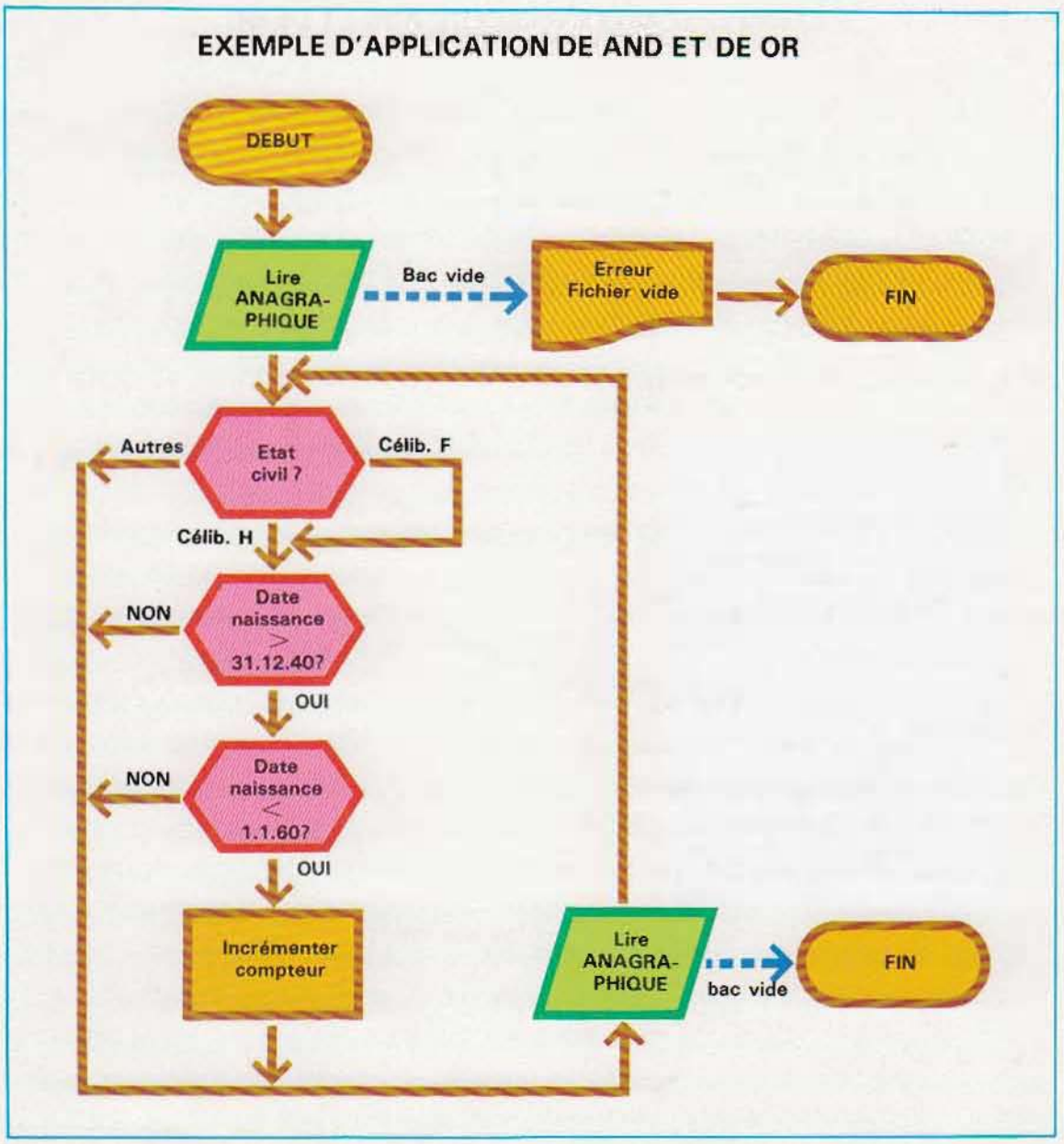
Nous voulons contrôler, pendant la lecture du fichier de cartes, la validité du champ ETAT-CIVIL. S'il contient le chiffre 1,2,3 ou 4, le traitement se poursuit. Dans le cas contraire, il faut émettre un message d'erreur.
En Cobol, on peut écrire :

```
IF CELIB H
   OR
   CELIB F
   OR
   MARIE
   OR
   VEUF
   PERFORM TRAITEMENT
ELSE
   DISPLAY 'ETAT CIVIL ERRONE'
      UPON CONSOLE.
```

Si, pour des raisons de structuration d'un programme, on doit analyser en premier le cas d'erreur, (c'est-à-dire si l'état-civil n'est ni célibataire homme, ni célibataire femme, ni marié, ni veuf), on peut intuitivement penser à codifier les instructions de la façon suivante :

```
IF NOT CELIB H
      OR
   NOT CELIB F
      OR
   NOT MARIE
      OR
   NOT VEUF
```

EXEMPLE D'APPLICATION DE AND ET DE OR




```
        DISPLAY 'ETAT CIVIL ERRONE'
                UPON CONSOLE
    ELSE
        PERFORM TRAITEMENT.
```

En réalité, en adoptant les instructions décrites, toutes les cartes seront signalées comme erronées.
Supposons, en effet, que l'employé décrit dans la carte à traiter, soit un homme célibataire. Dans ce cas, la première condition, pour écarter la carte, n'est pas vérifiée puisqu'il est vrai que l'employé est un jeune homme ; c'est pourquoi, en procédant à l'analyse de la phrase, l'employé ne peut être une célibataire femme et la condition d'erreur est donc vérifiée.
L'analyse de la phrase se poursuit jusqu'au test veuf, mais à ce niveau, trois possibilités d'erreur ont déjà été vérifiés. C'est pourquoi le message 'ETAT CIVIL ERRONE' est anormalement affiché.
Ce qui résume la formule :

NOT (A OR B) = (NOT A) AND (NOT B)

La négation globale d'une série de conditions

EXEMPLE D'APPLICATION DE AND ET DE OR

```
 01 EMPLOYE-WS.
    05  MATRICULE                    PIC 9(3).
    05  NOM                          PIC X(30).
    05  ETAT-CIVIL                   PIC 9.
        88  CELIBATAIRE-F            VALUE 1.
        88  CELIBATAIRE-H            VALUE 2.
        88  MARIE                    VALUE 3.
        88  VEUF                     VALUE 4.
    05  DATE-NAISSANCE.
        10  ANNEE                    PIC 9(2).
        10  MOIS                     PIC 9(2).
        10  JOUR                     PIC 9(2).
    05  ..................
    05  ..................
        10  ..............
*
*
 01 COMPTEUR                         PIC 9(4) COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
    OPEN INPUT ANAGRAPHIQUE.
 PREMIERE-LECTURE.
    READ ANAGRAPHIQUE INTO EMPLOYE-WS
                      AT END
                      DISPLAY '** FICHIER ANAGRAPHIQUE VIDE' **'
                              UPON PRINTER
                      CLOSE ANAGRAPHIQUE
                      STOP RUN.
 CONTROLE.
    IF  (CELIBATAIRE-F OR CELIBATAIRE-H)
        AND
        DATE-NAISSANCE > 401231
        AND
        DATE-NAISSANCE < 600101
        ADD 1 TO COMPTEUR.
    READ ANAGRAPHIQUE INTO EMPLOYE-WS
                      AT END
                      CLOSE ANAGRAPHIQUE
                      DISPLAY '** EMPLOYES NON MARIES '
                              'DEMANDES = '
                              COMPTEUR
                              UPON PRINTER
                      STOP RUN.
*
*
```

alternatives (OR) peut être exprimée **seulement** comme un produit logique (AND) de négations des conditions composantes.
L'écriture correcte de la seconde formule est donc :

```
IF NOT CELIBATAIRE-H
        AND
   NOT CELIBATAIRE-F
        AND
   NOT MARIE
        AND
   NOT VEUF
        DISPLAY 'ETAT CIVIL ERRONE'
                UPON CONSOLE
ELSE
   PERFORM TRAITEMENT.
```

En conclusion :
Quand plusieurs conditions sont analysées en même temps, il est bon de contrôler leur déroulement en mettant entre parenthèses les groupes simples.
De cette façon, on a le double avantage d'éviter des erreurs et de préserver l'ordre logique de l'analyse.

L'instruction GO TO

Les instructions d'un programme Cobol s'exécutent dans l'ordre de leur écriture par le programmeur, sauf indication contraire de celui-ci, par recours à l'instruction de saut GO TO.
En théorie, un programme Cobol pourrait être construit selon une cascade d'instructions appartenant toutes à un paragraphe unique :

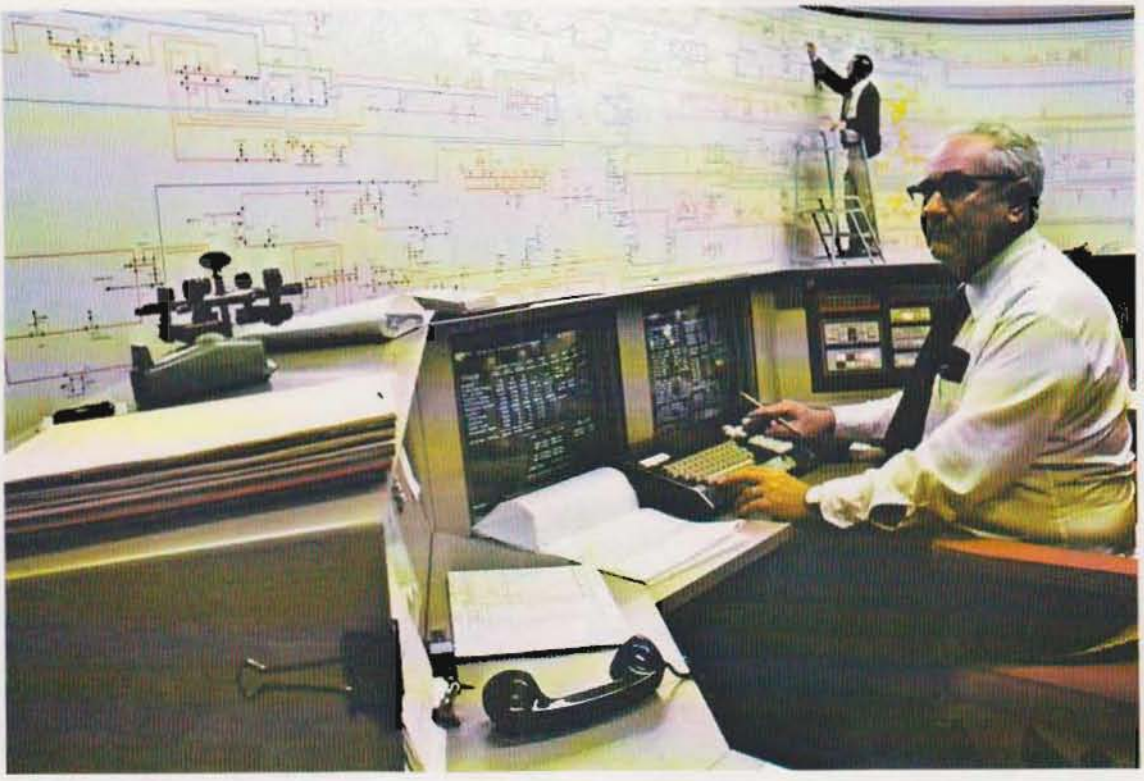
```
PROCEDURE DIVISION.
DEBUT.
    OPEN.....
    READ..... INTO.....
    ADD.....
    MOVE.....
    WRITE.....
    .....
    STOP RUN.
```

Un tel programme ne peut remplir que des fonctions très simples. La répétition d'une ou de plusieurs opérations identiques, comme la lecture d'un fichier et le traitement d'enregistrements, oblige à modifier le déroulement séquentiel du programme en créant des phases itératives sur un certain nombre d'instructions spécialement groupées.
Nous savons qu'un programme Cobol s'organise en paragraphes et en sections.
Cette organisation est particulièrement fonctionnelle quand, associée à l'emploi du verbe PERFORM, elle permet de créer des points de branchement auxquels le contrôle peut être cédé par l'instruction GO TO.
Pour le compilateur, les noms de paragraphes et de sections sont complètement transparents à moins qu'ils ne soient appelés par l'instruction GO TO. La forme la plus simple de l'instruction est :

```
GO TO nom-procédure
```

où nom-procédure comporte au maximum 30 caractères alphanumériques perforés à partir de la marge E (colonne 8) sur la carte.
Le saut du programme d'un point à un autre peut être conditionnel (lors de la vérification d'un événement prévisible) ou inconditionnel.

Tableau de contrôle des circuits de distribution dans une grande centrale électrique.



K. Reese/Marka

Situations qu'on retrouve dans les exemples suivants :

Premier cas

```
ADD 1 TO COMPUTER
IF COMPTEUR > 100
   GO TO FIN
ELSE
   .....
   .....
```

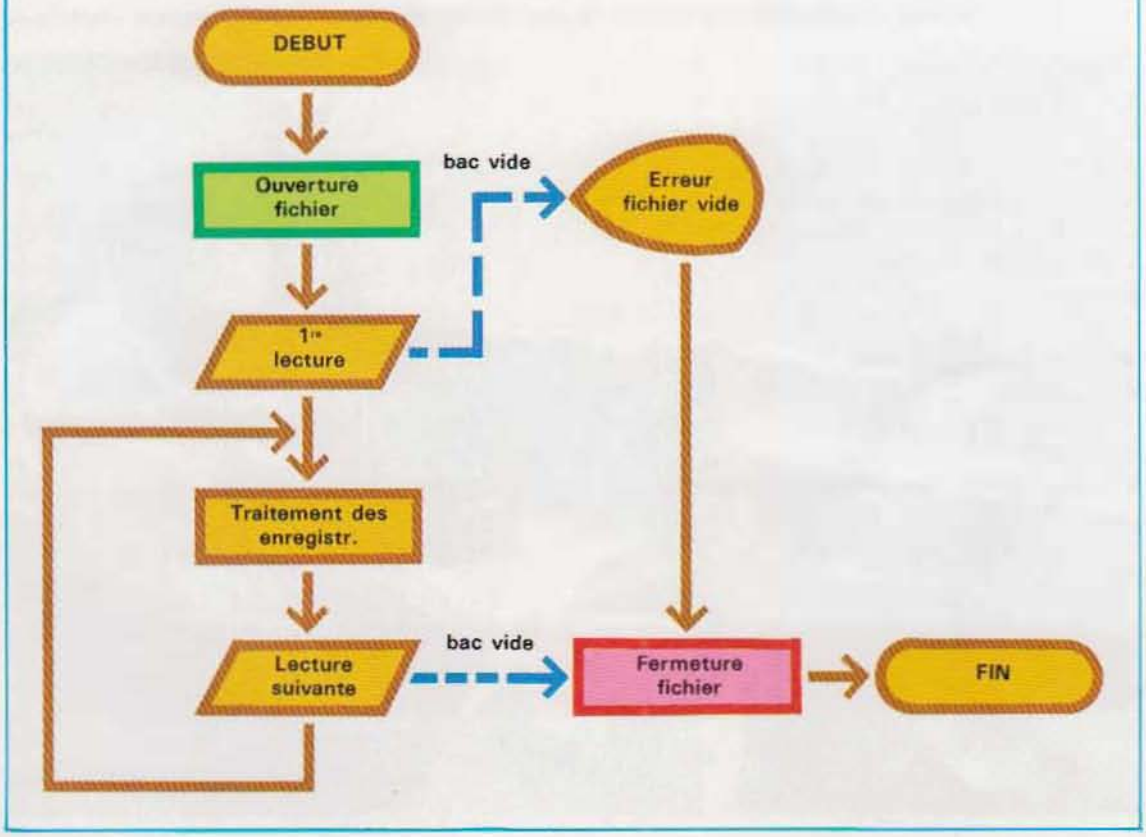
Second cas

```
MOVE A TO B.
ADD 100 TO K.
GO TO PROCEDURE-1.
```

Dans le premier cas, le saut à la procédure FIN n'est effectif que si le contenu de COMPTEUR dépasse 100, alors, que dans le second cas, le saut à la PROCEDURE-1 est réalisé après avoir additionné 100 à la variable K.
Le programmeur est libre d'utiliser les instructions de saut à n'importe quel point du programme, en sachant toutefois qu'elles altèrent sa lisibilité.
Inversement, un recours judicieux et limité de GO TO met en évidence la logique des opérations. C'est le cas des applications de lecture d'un fichier et de contrôle de la condition de fin de fichier.

Le déroulement des opérations de lecture d'un fichier séquentiel, par exemple, respecte le schéma de la figure ci-dessous.
La première lecture (également appelée « lecture hors cycle »), est exécutée une seule fois, immédiatement après l'ouverture du fichier.
Il est évident que, dans ce contexte, le test de la condition « fin de fichier » déclare que le fichier est vide. Evidemment, une telle condition est anormale et l'arrêt du traitement est donc justifiée.
Si le fichier n'est pas vide,le contrôle passe à la première instruction exécutable après le READ et l'enregistrement traité.

CYCLE DE LECTURE D'UN FICHIER SEQUENTIEL



CYCLE DE LECTURE D'UN FICHIER SEQUENTIEL

```
      ..................
      OPEN INPUT FICHIER-ENTREE.
  PREMIERE-LECTURE.
      READ FICHIER-ENTREE INTO ENR-LU
                  AT END
                  DISPLAY '** FICHIER FICHIER-ENTREE VIDE **'
                          UPON CONSOLE
                  GO TO FIN-TRAITEMENT.
  TRAITEMENT-ENREGISTREMENT
      MOVE ..........  TO ..........
      ADD  ..........  TO ..........
                          ..........
                          ..........
      IF   .......... = ............
           ..........
           COMPUTE ....... = ......... + .........
           MOVE    ....... TO.........
      ELSE
           ..........
           ..........
           ..........
      MOVE ENR-LU      TO .........
*
  LIRE-FICHIER-ENTREE.
      READ FICHIER-ENTREE INTO ENR-LU
                  AT END
                  DISPLAY '** FIN LECTURE FICHIER-ENTREE **'
                          UPON CONSOLE
                  GO TO FIN-TRAITEMENT.
*
  FIN-TRAITEMENT.
      CLOSE FICHIER-ENTREE.
      STOP RUN.
```

Cette phase terminée, c'est au tour de la seconde instruction de lecture d'être exécutée autant de fois qu'il y a d'enregistrements dans le fichier.
Le détail de la lecture d'un fichier séquentiel se trouve sous forme de programme en haut de cette page et ci-contre sous forme d'organigramme.

La clause DEPENDING ON. Le GO TO est généralement employé en aval du test de validité d'une événement : selon le résultat, la main est donnée à telle ou telle procédure.
Le cas le plus courant est décrit dans l'exemple suivant :

```
IF A = 1
   ..............
   ..............
   GO TO A-EGAL-1
ELSE
   ..............
GO TO A-DIFFERENT-1.
```

Ainsi, le contrôle est cédé, selon le résultat du test, à l'une des procédures A-EGAL-1 ou A-DIFFERENT-1.
Dans le cas de plusieurs conditions, il serait inélégant d'écrire :

```
IF A = 1
   GO TO PROCEDURE-1.
IF A = 2
   GO TO PROCEDURE-2.
IF A = 3
   GO TO PROCEDURE-3.
   ..............
   ..............
IF A = N
   GO TO PROCEDURE-N.
```

La clause DEPENDING ON, associée à l'instruction GO TO, permet d'aboutir au même résultat, comme il apparait ci-après :

```
GO TO PROCEDURE-1
      PROCEDURE-2
      PROCEDURE-3
      ..........................
      PROCEDURE-N DEPENDING ON A.
```

La séquence d'opérations lancée par le compilateur se déroule de la même manière que dans la formule précédente.
Il faut préciser que :

- le champ spécifié dans la clause DEPENDING ON (A dans notre exemple) doit être numérique ;
- si le contenu du champ spécifié est un nombre différent de 1, 2, ..., N (nombre des procédures énumérées), le contrôle passe à l'ordre suivant immédiatement l'instruction de saut ;
- les noms des procédures doivent être énumérés de telle manière que le nombre contenu dans le champ spécifié (dans la clause DEPENDING ON) détermine leur position dans la liste.

Par exemple, la phrase :

```
GO TO A
      B
      C
      D
      E DEPENDING ON NOMBRE.
```

passera le contrôle aux procédures A si NOMBRE=1, B si NOMBRE=2, et ainsi de suite.

Le verbe PERFORM

Déjà introduite dans certains des exemples présentés, cette instruction permet, avec une seule commande, de traiter toutes celles d'un paragraphe ou d'une section. Cela permet de subdiviser le programme en blocs fonctionnels pouvant être rappelés depuis n'importe quel point du programme. Son emploi, en accord avec la description du Cobol, permet de structurer le programme de manière concise et auto-documentée.
Naturellement, cette dernière caractéristique est liée à la possibilité qu'a le programmeur d'utiliser, pour chaque procédure appelée par PERFORM, des noms désignant directement la fonction.
Si, par exemple, une section du programme calcule la TVA, il est conseillé de la nommer CALCUL-TVA, ce qui est plus explicite qu'une abréviation du type C-T (Calcul TVA).

L'instruction PERFORM est dotée de nombreuses clauses qui seront analysées par la suite. Elle se présente ainsi :

PERFORM nom-procédure

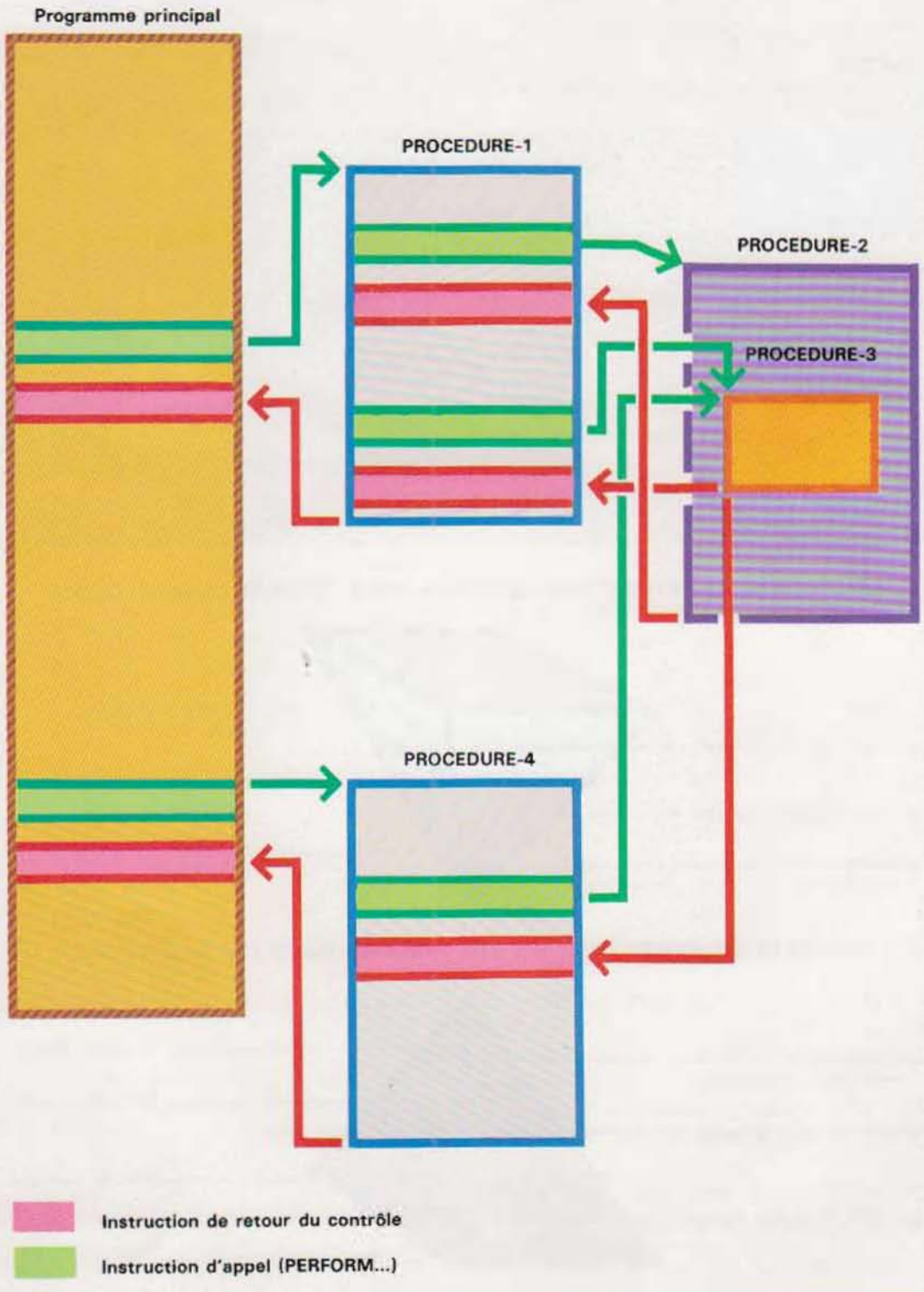
où « nom-procédure » désigne un paragraphe ou une section.
Le diagramme ci-contre montre comment le programme exécute toutes les instructions contenues dans une procédure quand celle-ci est appelée par un PERFORM.
On y voit également comment le contrôle des opérations passe du programme principal aux procédures appelées avec retour à la première instruction à exécuter immédiatement après le verbe d'appel.
Notons qu'une procédure peut contenir l'appel à une autre procédure à condition que celle-ci soit entièrement externe ou interne à la procédure appelante.
De plus, une procédure peut être appelée autant de fois que nécessaire et depuis un point quelconque du programme.
Dans le diagramme fonctionnel, la PROCEDURE-3 faisant partie de PROCEDURE-2 apparaît néanmoins dans d'autres procédures. Supposons, pour simplifier, que la fonction de PROCEDURE-3 soit remplie par l'ensemble des instructions énumérées ci-dessous et comprises entre le nom de paragraphe PARAGRAPHE-1 et le nom de paragraphe PARAGRAPHE-4 :

```
PROCEDURE-1.
        MOVE A TO B.
        ..............
        ..............
PARAGRAPHE-2.
        ..............
        ..............
PARAGRAPHE-3.
        ..............
        ..............
        ADD 1 TO K.
PARAGRAPHE-4.
        ..............
        ..............
```

PROCEDURE-3

Il est possible d'appeler, avec une seule instruction, toutes celles indiquées par l'accolade en écrivant simplement :

APPEL DE PROCEDURE PAR L'INTERMEDIAIRE DE PERFORM

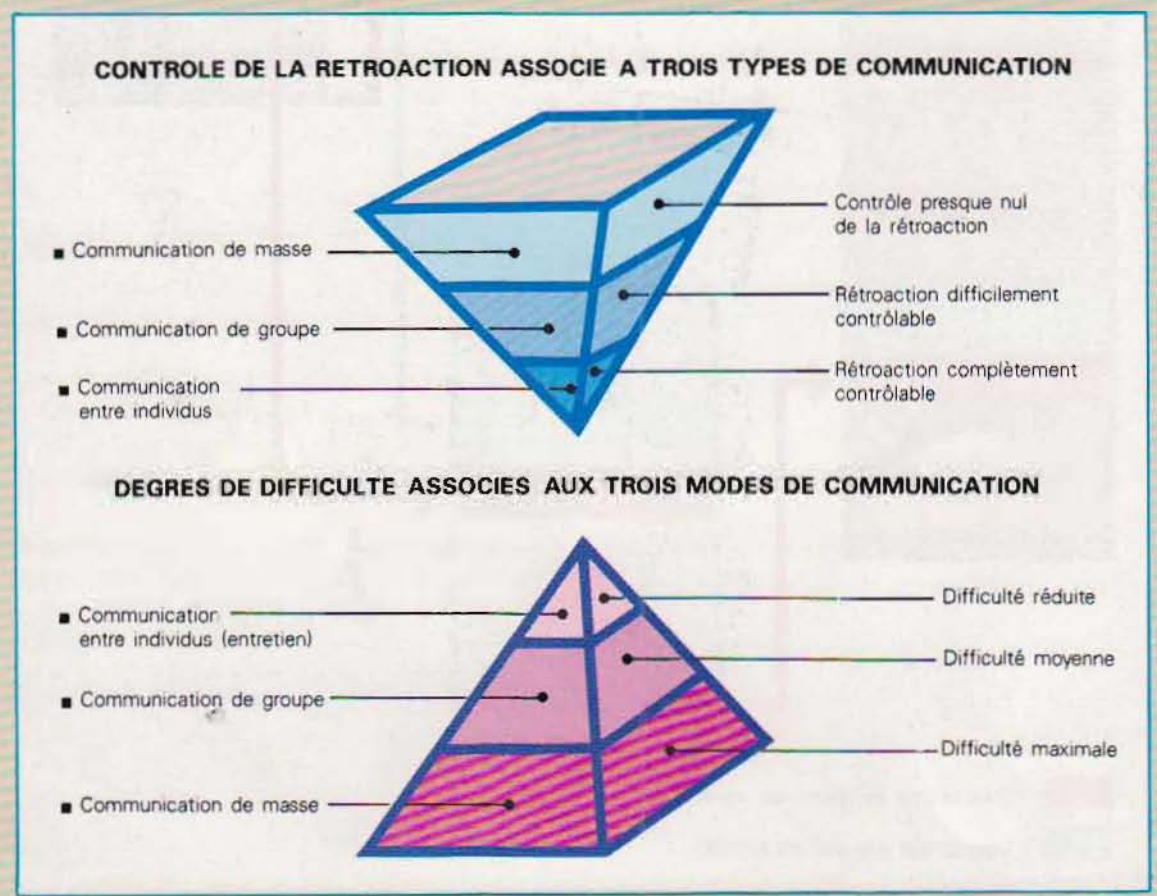


Le dialogue homme-machine

Pensez à l'une de vos journées et analysez le nombre de fois où vous avez eu l'occasion du matin au soir de :

- *parler avec un simple interlocuteur* (communication entre individus) ;
- *parler à un groupe de personnes* (communication de groupe) ;
- *parler face à un média, comme la radio ou la télévision* (communication de masse).

Evidemment, à moins d'être un super président, la plupart de vos rencontres seront de type interpersonnel.
Cet aspect des choses fait ressortir un ordre de fréquence d'utilisation de la communication nettement en faveur des entretiens entre simples personnes.
On peut classer les difficultés de communication selon une échelle de valeurs progressives (voir schéma ci-dessous).
Les gens ont donc, pour la plupart, ***l'habitude du dialogue avec des individus.*** *Par contre, ils se sentent extrêmement démunis face à un groupe et, pire encore, quand il s'agit de passer par l'intermédiaire d'un dispositif audio ou vidéo (enregistrement, passage à la radio ou à la télévision).*
La panique éprouvée par les sens entraîne une prestation de mauvaise qualité, bien moins bonne que si cela s'était passé en petit comité et, surtout, hors de la présence d'appareils intimidants.
Mais pourquoi est-il difficile de communiquer en groupe ?
La raison en semble la réduction progressive du contrôle sur la communication de retour (rétroaction ou feedback).
Le phénomène s'explique par ce que l'on peut appeler la théorie de la chauve-souris. Cet animal aveugle possède une sorte de radar qui lui sert à repérer les obstacles situés sur son passage.
Il s'agit là d'un système d'analyse de signaux de retour.



L'homme se comporte de la même manière au cours d'un entretien.
Observez attentivement votre interlocuteur : vous percevrez toute une série de signaux (mouvements des yeux, de la bouche, des mains) qui révèlent l'attention, l'ennui, la hâte, la stupéfaction.
Par contre, on se rend tout de suite compte que quelqu'un écoute : ses yeux sont bien fixes et sa tête immobile, contrairement à celui qui s'ennuie et qui essaie de fixer son attention en tambourinant des doigts sur la table, en regardant souvent sa montre, et qui s'oublie même à bailler...
Ce type de **contrôle des signaux de retour** peut aussi être effectué dans la communication de groupe, même si le degré de difficulté augmente à cause du nombre des interlocuteurs.
En revanche, le phénomène de **feed-back** disparaît complètement face à un micro ou à une caméra. C'est pourquoi il est souvent si angoissant de se retrouver devant un appareil sans réaction.
Le même phénomène se produit face au clavier et à l'écran d'un ordinateur.
A travers ces signaux, nous recherchons d'abord des « renforts » et un réconfort. Finalement, nous demandons inconsciemment de l'aide.
Une des fonctions fondamentales de la communication est, en effet, de soulager de l'anxiété. En effet, en plus de plaire, l'homme a besoin de communiquer.
Richard Stevens répertorie huit fonctions de base autour desquelles l'activité de relation humaine se développe. Nous communiquons :

- pour accomplir ou pour atteindre quelque chose (fonction instrumentale),
- pour dicter un comportement (fonction de contrôle),
- pour découvrir ou expliquer quelque chose (fonction d'information),
- pour transmettre notre intérêt pour quelque chose (fonction de stimulation),
- parce que la situation l'exige (fonction liée au rôle),
- pour exprimer nos sentiments ou pour nous imposer d'une certaine manière (fonction d'expression),
- pour montrer notre plaisir à être en compagnie (fonction de contact social),
- pour approfondir un problème ou pour nous soulager d'une préoccupation (fonction de soulagement de l'anxiété).

Trois, au moins, des fonctions décrites (expression, contact social et soulagement de l'anxiété) mettent clairement en évidence la nécessité des échanges avec les autres. Voyons maintenant comment Stevens illustre ces trois fonctions.

Fonction d'expression. Une femme se confie à une amie, un homme relève avec colère l'erreur d'un collègue, un amoureux chuchote des paroles d'amour... La communication est un procédé qui permet d'exprimer notre façon d'être : elle est spontanée et authentique ou soigneusement construite pour atteindre un but désiré, par exemple, pour faire impression sur un employeur potentiel pendant un entretien. La fonction auto-expressive de la communication peut opérer de façon dissimulée. Ce qui pourrait n'être qu'un échange fortuit d'informations, pendant une réception, devient dans la réalité une tentative de briller par l'esprit et l'intelligence. Une discussion politique peut être, face à un interlocuteur, l'occasion de manifester sa supériorité sur l'autre.
Le style de la communication, le langage employé, les paroles utilisées constituent une manière d'affirmer **l'appartenance à un groupe.** De nombreuses sous-cultures (groupes professionnels par exemple) ont leur jargon spécifique qu'il faut connaître pour être accepté par le groupe.

Fonction de contrat social. La communication peut être une fin en soi. Comme pour toutes les caractéristiques humaines, il y a, même dans ce cas, des différences d'un individu à un autre.
La plupart des gens éprouvent (plus ou moins) le désir d'être en compagnie d'autres personnes pour le simple plaisir d'être ensemble. Nous apprécions et recherchons davantage le contact avec des personnes qui ont des comportements et des expériences semblables aux nôtres avec lesquelles une interaction s'établit.
Les expériences psychologiques, d'une part, ainsi que les récits de naufragés ou d'ermites,

de l'autre, montrent combien l'isolement social produit souvent des résultats désastreux.

Fonction thérapeutique. Il est amplement démontré que, lorsqu'un individu se trouve dans une situation qui provoque **l'anxiété,** il tente d'établir le contact avec les autres. Le psychologue américain Stanley Schachter a trouvé, par exemple, que certaines étudiantes qui croyaient devoir se soumettre à des tests psychologiques à base de « secousses » électriques douloureuses, ont préféré (contrairement à un autre groupe qui ignorait la nature du remède) attendre leur tour en même temps que d'autres. Ceci ne se produisait, naturellement, que si les camarades en attente se trouvaient dans la même situation. Schachter (1959) la résuma ainsi :
« Le malheur ne se contente pas de n'importe quelle compagnie, il a besoin de la compagnie d'autres malheureux ».
Il est clair que la communication nous est utile pour échanger une série complexe de messages dont beaucoup sont riches en implications secondaires. Ce n'est pas par hasard que Lévi-Strauss définit la société comme un système d'individus et de groupes humains qui communiquent entre eux. La **communication** n'est pas seulement un système de langage, elle est aussi un **procès de personnes.**
Nous nous sommes rendu compte combien la communication est utile aux contacts humains pour survivre sans trop d'angoisse. Nous avons enfin vu comment ces rapports, même pour les habituels échanges interpersonnels qui privilégient les contacts individuels (plutôt que les contacts à travers un média), altèrent en fait le dialogue entre individus.
Mais quand le **dialogue** n'a pas lieu entre deux personnes mais **entre un homme et une machine**, que se passe-t-il ?
Un nouveau type de dialogue naît. Un dialogue spécial fait d'un côté d'humanité et de l'autre de rapidité d'exécution quand la machine est un ordinateur. Tout va bien quand l'homme, face à l'ordinateur, demande des solutions rapides sur la base des programmes que lui-même à écrit. Cependant, les choses se gâtent quand l'homme ne demande pas uniquement de la rapidité mais également un autre type d'aide. Bref un dialogue, c'est-à-dire un processus d'échange, de relation, de sentiments, un moyen de **fuir la solitude et la peur...**
Il n'est pas légitime d'attendre de l'aide, au sens psychologique du terme, de la part de l'ordinateur, mais cela se produit souvent. De là, les questions angoissées de nombreux parents : passer un long temps face à la machine, n'est-ce pas nuisible au développement intellectuel de l'enfant ? Ne va-t-il pas se rendre dépendant de la machine ? Ne s'agit-il pas d'une aide apparente qui masque un frein au développement de ses capacités mentales ?
Tout d'abord, chaque génération devrait avoir le bon sens de ne pas trop intervenir dans le vécu quotidien de la génération suivante et de ne pas trop chercher à la gérer avec son propre étalon mental. Pour une raison simple : cette génération **N'A PAS** les mêmes besoins que nous.

Examinons d'abord un premier élément : LE TYPE DE RAPPORT DANS LES DIALOGUES.
La quantité de dialogue entre personnes sera en voie de diminution au profit des dialogues entre l'homme et l'ordinateur. Celui qui prétend obtenir le même type de communication avec un ordinateur que dans le dialogue humain vit en dehors du temps. Aujourd'hui, l'homme doit apprendre à dialoguer avec la machine sans se sentir frustré.

L'un des exemples les plus simples est celui du téléphone. Ajourd'hui, on fait tout avec le téléphone ; on a presque l'impression de ne plus pouvoir vivre sans lui. Et pourtant, il y a quelques décennies seulement, le téléphone était considéré comme une espèce de monstre dont il fallait se méfier. Aujourd'hui, beaucoup d'organisations d'assistance téléphonique sauvent des vies humaines ou, à tout le moins, libèrent les dépressifs de l'angoisse.
Mais on dira qu'il se trouve, à l'autre bout, une **présence humaine.** C'est vrai. Et le programme avec lequel nous dialoguons ou jouons aux échecs n'est pas, en termes psychologiques, de même nature. Mais, dans ce rapport, lequel doit changer d'entre la machine et l'homme ? Si, à la machine, nous demandons chaleur humaine, sentiments, aide psychologique, est-ce de sa faute si nous ne les obtenons pas ? Non, n'est-ce pas ? On ne devrait, logiquement, jamais attendre d'une chose ce qu'elle ne contient pas par nature.

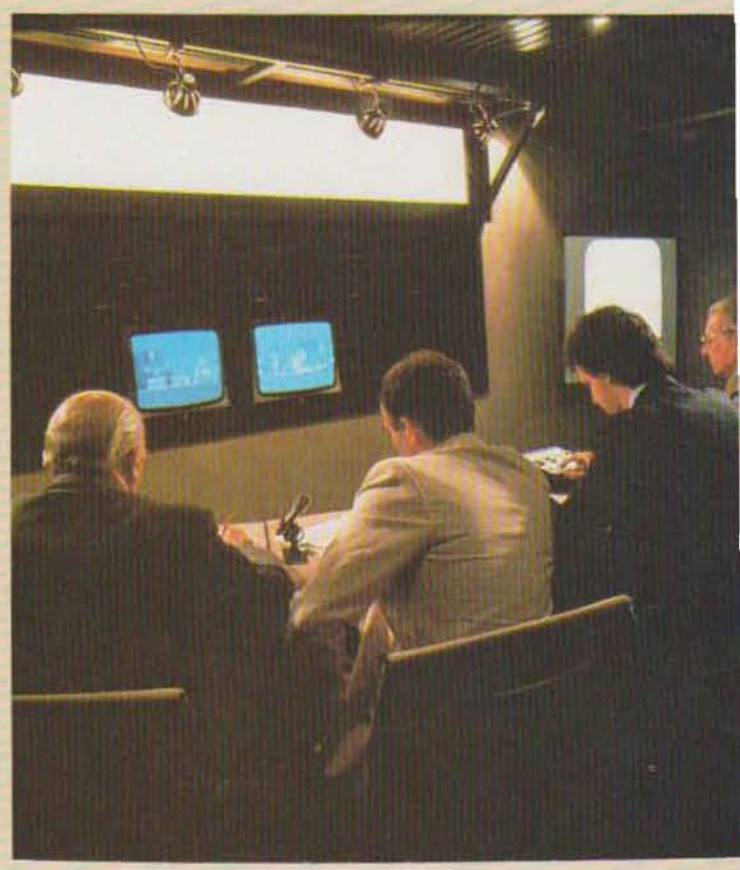
On a vu, dans le précédent numéro, comment le dialogue avec l'ordinateur était effectivement chargé et comment s'opéraient des phénomènes de transfert.

L'ordinateur finit donc par devenir un test d'équilibre psychologique. Si nous ne nous sentons pas bien avec les machines, peut-être est-ce parce que nous ne le sommes pas avec nous-mêmes. Peut-être aussi avons-nous besoin de relations d'aide, mais cela ne concerne pas l'ordinateur.

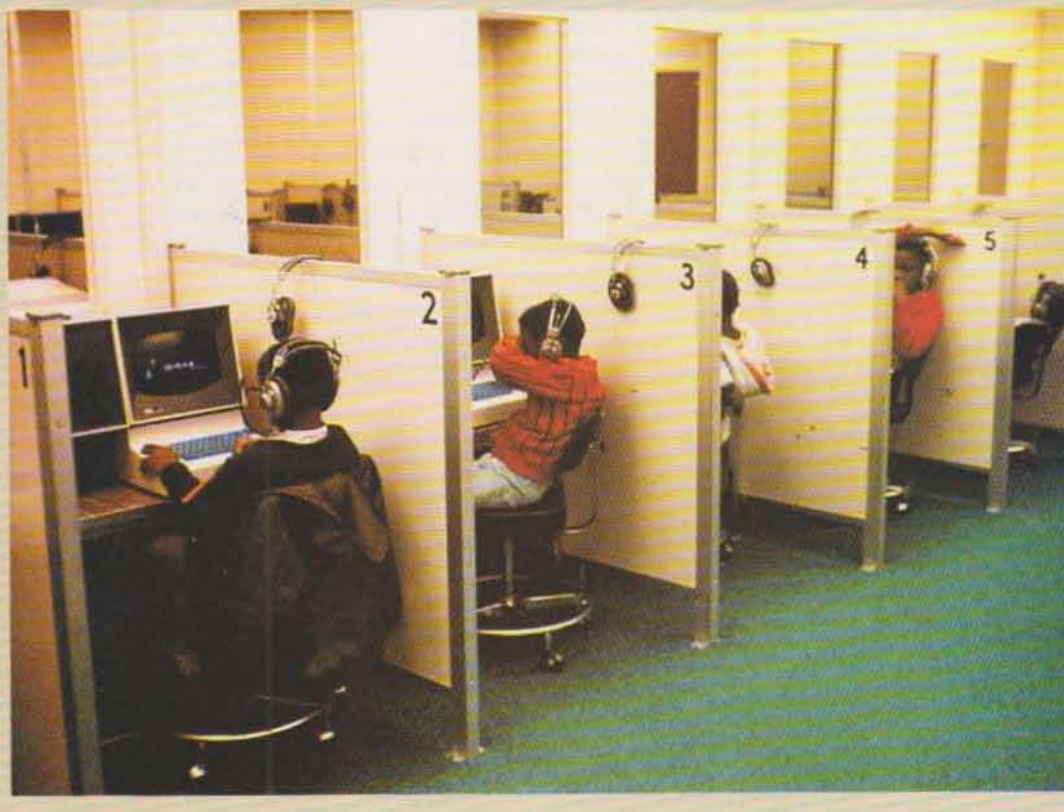
La machine est un ***instrument*** *qui, par définition est* ***neutre.*** *Le couteau sert à couper le pain et à tuer les adversaires : toutes choses ne dépendant pas de l'instrument mais de l'usage que l'on en fait. Nous pourrions donc affirmer qu'une personne équilibrée, indépendante d'une aide extérieure, peut sereinement « dialoguer » avec l'ordinateur.*

Les ***psychanalystes*** *freudiens savent combien la « neutralité » de l'analyse apparaît comme une forme d'hostilité au cours des premières séances. « Vous ne participez pas... Vous restez là à m'écouter parler... Vous moquez-vous de moi ?... »*

Nous demandons donc à nos interlocuteurs une intervention active. La neutralité (mais la



SIP



J. Launois/Grazia Neri-Black Star

La difficulté de communiquer réduit souvent la capacité des étudiants. L'enseignement assisté par ordinateur (EAO) aide à surmonter la barrière du rapport étudiant-enseignement (photo ci-contre). Dans ces conditions, la machine étant un moyen neutre, le contrôle de la rétroaction est total. Les solutions novatrices comme la téléconférence (en haut) introduisent une interface életronique qui abolit les distances.

psychanalyse n'est pas un dialogue) nous irrite jusqu'à ce que nous comprenions que la réponse à nos problèmes n'est pas dans l'autre mais en nous.
Ce n'est ni chez l'analyste ni dans l'ordinateur que se trouve la solution, même s'il est facile de ***transférer notre angoisse sur ces « passifs ».***
Le type de langage adopté à une grande importance pour l'individu qui utilise l'ordinateur avec une approche psychologique correcte et équilibrée.
Il semble aujourd'hui normal que les machines s'adaptent à l'homme en termes ergonomiques mais aussi en termes de « capacité expressive ». Par cette expression, on ne doit pas seulement entendre amélioration linguistique (pourtant indispensable dans le dialogue homme-machine), mais aussi une forme de communication directe avec l'homme, et surtout, un échange fait sur un mode neutre et évitant toute occasion d'irritation.
Les couleurs et les sons constituent, avec les mots, un univers qui doit encore être complètement exploré dans la nouvelle approche homme-machine.
En général, la tendance à assimiler la communication à des faits visuels et sonores, confère une plus grande importance à l'aspect linguistique et à la langue écrite.
L'avènement de ***l'écriture informatique*** *provoque déjà une révolution comparable à celle de Gutenberg avec l'utilisation des caractères mobiles d'impression, ce qui a développé la communication entre des personnes éloignées à un point inimaginable auparavant.*
La communication écrite devra être plus concrète, plus concise. Il sera nécessaire d'apprendre à transmettre l'émotion sans nécessairement recourir à des flots de termes lyriques.
Comment ? La voie est déjà tracée par les techniciens des communications écrites, qu'il s'agisse des ***journalistes*** *ou des rédacteurs de textes* ***publicitaires.***

Voici quelques conseils pratiques pour rendre la communication écrite plus concrète, plus concise et de ce fait, mieux adaptée à l'ère électronique :

- Recourir à des mots courts. *Selon Rudolph Flesh, un texte de 100 mots écrits en anglais ne devrait pas dépasser 200 syllabes.*
- Construire des phrases brèves : *sujet-verbe-attribut avec, toujours selon la règle de lisibilité de Flesh, une moyenne d'environ 15 mots par phrase.*
- Donner des informations utiles *en se demandant constamment ce que le lecteur retient de cette affirmation.*
- Ne pas essayer de dire trop de choses *en même temps. Quand on en dit trop, rien ne passe.*
- Expliquer avec des exemples bien choisis : *les exemples éclairent beaucoup mieux que les définitions détaillées.*
- Placer l'exemple avant la définition. *Si l'on doit expliquer l'effet* ***synergique,*** *on peut commencer avec un exemple : « Deux plus deux égal cinq. La synergie... » Dans la communication écrite, on procède généralement de manière inverse : définition et ensuite exemple ; il s'ensuit qu'il faut tout relire du début puisqu'habituellement la définition est comprise après avoir lu l'exemple.*
- Anticiper les conclusions *à la manière d'un compte-rendu journalistique : d'abord la conclusion du fait, ensuite sa chronologie.*
- Répéter pour augmenter la mémorisation. *Afin d'éviter de créer une réaction négative (« je l'ai déjà lu... »), il est nécessaire de répéter selon une autre formulation. On laisse ainsi le temps au lecteur de comprendre, tout en continuant à éveiller son attention et, éventuellement, en attirant sa réflexion sur un point particulier.*

Et si, maintenant, nous ***récapitulions*** *tout ce que nous avons appris ?*

Comme outil, l'ordinateur est neutre. Ne lui demandons pas une aide psychologique, comme nous faisons avec des personnes. Le dialogue homme-machine a d'autres finalités. Acceptons donc la nouvelle réalité : moins de communication interpersonnelle pour fuir son angoisse et plus de dialogue orienté vers l'échange d'information.

Enrico Cogno

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3
```

Dans cette forme, PERFORM passe la main à la première instruction exécutable de PARAGRAPHE-1 (MOVE A TO B) immédiatement après l'exécution de la dernière instruction de PARAGRAPHE-3 (ADD 1 TO K.).
Le recours à une telle forme de PERFORM présente deux inconvénients majeurs qui poussent à en déconseiller l'utilisation.
La première erreur serait d'écrire :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-4
```

en croyant n'envoyer, en exécution, que les instructions comprises entre les mots PARAGRAPHE-1 et PARAGRAPHE-4. En rélité, seront également concernées toutes les instructions de PARAGRAPHE-4. On peut cependant éviter ce problème avec un peu d'expérience.
Le second inconvénient a trait à la manipulation du programme, ce qui est bien plus grave.
Supposons, en effet, qu'au cours de mises à jour successives, il faille modifier le programme de la façon suivante :

```
PARAGRAPHE-1.
     MOVE A TO B
     .......
     .......
PARAGRAPHE-2.
     .......
     .......
PARAGRAPHE-3.
     .......
     .......
RETOUR.
     .......
     .......
     IF A NOT=C
        .......
        .......
          GO TO RETOUR.
     .......
     .......
     ADD 1 TO K.
PARAGRAPHE-4.
     .......
     .......
```

On a créé, entre PARAGRAPHE-3 et PARAGRAPHE-4, le nouveau paragraphe

Le portable HP110 est un vrai micro-ordinateur et ne pèse que 4,10 kg.



Hewlett Packard

RETOUR, ce qui est justifié par la nécessité d'effectuer un contrôle sur la variable A.
Dans ce cas, l'instruction :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3
```

restitue le contrôle au programme principal dès que l'on rencontre le paragraphe RETOUR, rétablissant le déroulement du traitement. Il faut donc corriger toutes les instructions d'appel en les modifiant ainsi :

```
PERFORM PARAGRAPHE-1 THRU RETOUR.
```

Les corrections demandées peuvent être importantes, selon le nombre de modifications implémentées et la complexité du programme.
En conclusion, il est recommandé de structurer le programme en sections prenant complètement en charge la fonction à éxécuter.
Les appels doivent toujours être écrits selon la forme suivante :

```
PERFORM nom-section.
```

Dans le cas où l'on désire adopter l'appel de plusieurs procédures avec la proposition THRU, on peut recourir à l'instruction EXIT.

L'instruction EXIT. Elle n'a, en réalité, aucun effet, si ce n'est de pallier, en partie, le second inconvénient de PERFORM. Il suffit que le programme soit structuré ainsi :

```
PARAGRAPHE-1.
      MOVE A TO B.
      . . . . . .
      . . . . . .
PARAGRAPHE-2.
      . . . . . .
      . . . . . .
PARAGRAPHE-3.
      . . . . . .
      . . . . . .
      ADD 1 TO K.
PARAGRAPHE-3-EX.
      EXIT.
PARAGRAPHE-4.
```

L'instruction :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3-EX.
```

envoie alors EXIT en exécution, comme dernière instruction.
Le contrôle est ensuite restitué à la partie appelante.
De ce fait, n'importe quel paragraphe inséré à l'intérieur de la procédure appelée est exécuté sans interruption.

Exécutions itératives de la même procédure. Grâce à la clause TIMES, l'instruction PERFORM permet de répéter l'exécution de la même procédure ou du même groupe de procédures (THRU).
Le tout se trouve résumé dans le format de PERFORM (voir ci-contre, haut de la page 1085), complété par les clauses que l'on vient de décrire.
Soit le calcul de la somme des nombres entiers de 1 à 1000. Le programme s'écrit :

```
PROCEDURE DIVISION.
DEBUT SECTION.
MISE-A-ZERO.
     MOVE ZEROES TO NOMBRE
                    TOTAL.
INCREMENTER-NOMBRE.
     ADD 1 TO NOMBRE.
     IF NOMBRE GREATER 1000
       DISPLAY'*** TOTAL='
               TOTAL
               '***'
               UPON CONSOLE
       STOP RUN.
     ADD NOMBRE TO TOTAL.
     GO TO INCREMENTER-NOMBRE.
*
```

On parvient au même résultat avec l'instruction suivante :

```
PROCEDURE DIVISION.
DEBUT SECTION.
MISE-A-ZERO.
     MOVE ZEROES TO NOMBRE
                    TOTAL.
     PERFORM SOMME 1000 TIMES.
     DISPLAY'*** TOTAL='
             TOTAL
             '***'
             UPON CONSOLE.
```

PREMIER FORMAT DE L'INSTRUCTION PERFORM.

```
PERFORM nom-procédure-1
        [THRU nom-procédure-2]

[ { nombre-entier-1 }      TIMES ]
  { nom-donnée—1    }
```

```
        STOP RUN.
*
   SOMME-SECTION.
   SOMME-PARAGR.
        ADD 1 TO NOMBRE.
      ADD NOMBRE TO TOTAL.
   SOMME-EX. EXIT.
```

La clause UNTIL. La possibilité de connaître a priori le nombre de fois qu'un groupe d'instructions doit être exécuté est assez rare. Surtout que le cycle d'itération utilise un champ numérique imposé par le programme et non une constante fixe comme dans le dernier exemple. Il est plus fréquent de faire intervenir l'arrêt de l'itération au moment où un certain événement se vérifie.
Nous voulons, par exemple, additionner les premiers entiers jusqu'à obtenir un résultat supérieur à 3425.
Notons que la valeur, pour stopper l'itération, peut aussi être contenue dans un champ numérique défini dans WORKING-STORAGE.
Le résultat désiré s'obtient avec les instructions suivantes, qui intègrent l'utilisation de la clause UNTIL (jusqu'à ce que) :

```
   PROCEDURE DIVISION.
   DEBUT SECTION.
   MISE A ZERO.
      MOVE ZEROES TO NOMBRE.
                     TOTAL
                     NOMBRE-SOMMES.
      PERFORM SOMME
              UNTIL
              TOTAL GREATER 3425.
      DISPLAY '*** NOMBRES-SOMMES='
              NOMBRES-SOMMES
              UPON CONSOLE.
      DISPLAY'***TOTAL FINAL='
              TOTAL
              UPON CONSOLE.
         STOP RUN.
*
   SOMME SECTION.
   SOMME-PARAGR.
      ADD 1 TO NOMBRE
               NOMBRES-SOMMES.
      ADD NOMBRE TO TOTAL
   SOMME-EX. EXIT.
```

La clause VARYING... FROM... BY. Cette clause étend la puissance de la précédente en conditionnant les itérations lors de la vérification d'une condition (UNTIL).
Son intérêt vient de ce qu'elle est dotée d'une plus grande flexibilité.
Une telle souplesse est due à la possibilité de varier (VARYING) la valeur d'une donnée quelconque pendant tout le déroulement de la procédure.
Cette valeur est incrémentée à chaque itération d'un pas déterminé (BY) en partant d'une valeur initiale (FROM) (voir format, haut de la page suivante).
Une application typique de ce format concerne la gestion des tableaux, surtout en phase de chargement et de recherche séquentielle.
On étudiera plus longuement cet aspect par la suite.

L'instruction STOP. La forme de cette instruction, déjà rencontrée dans les exemples précédemment étudiés, est la suivante :

```
STOP RUN.
```

A noter qu'il existe une seconde forme qui permet de suspendre momentanément le traitement.

SECOND FORMAT DE L'INSTRUCTION PERFORM

```
PERFORM     nom-procédure-1
            [THRU nom-procédure-2]
            VARYING     nom-donnée-1
            FROM    { constante numérique-1 }
                    { nom-donnée-2          }
            BY      { constante numérique-2 }
                    { nom-donnée-3          }
            UNTIL       condition
```

On envoie, à ce moment-là, un message à la console du système. L'interruption est alors gérée par l'opérateur et le traitement redémarre en fonction de la réponse fournie, à partir de la première instruction suivant l'instruction STOP.

La ligne :

STOP'ERREUR DE CALCUL

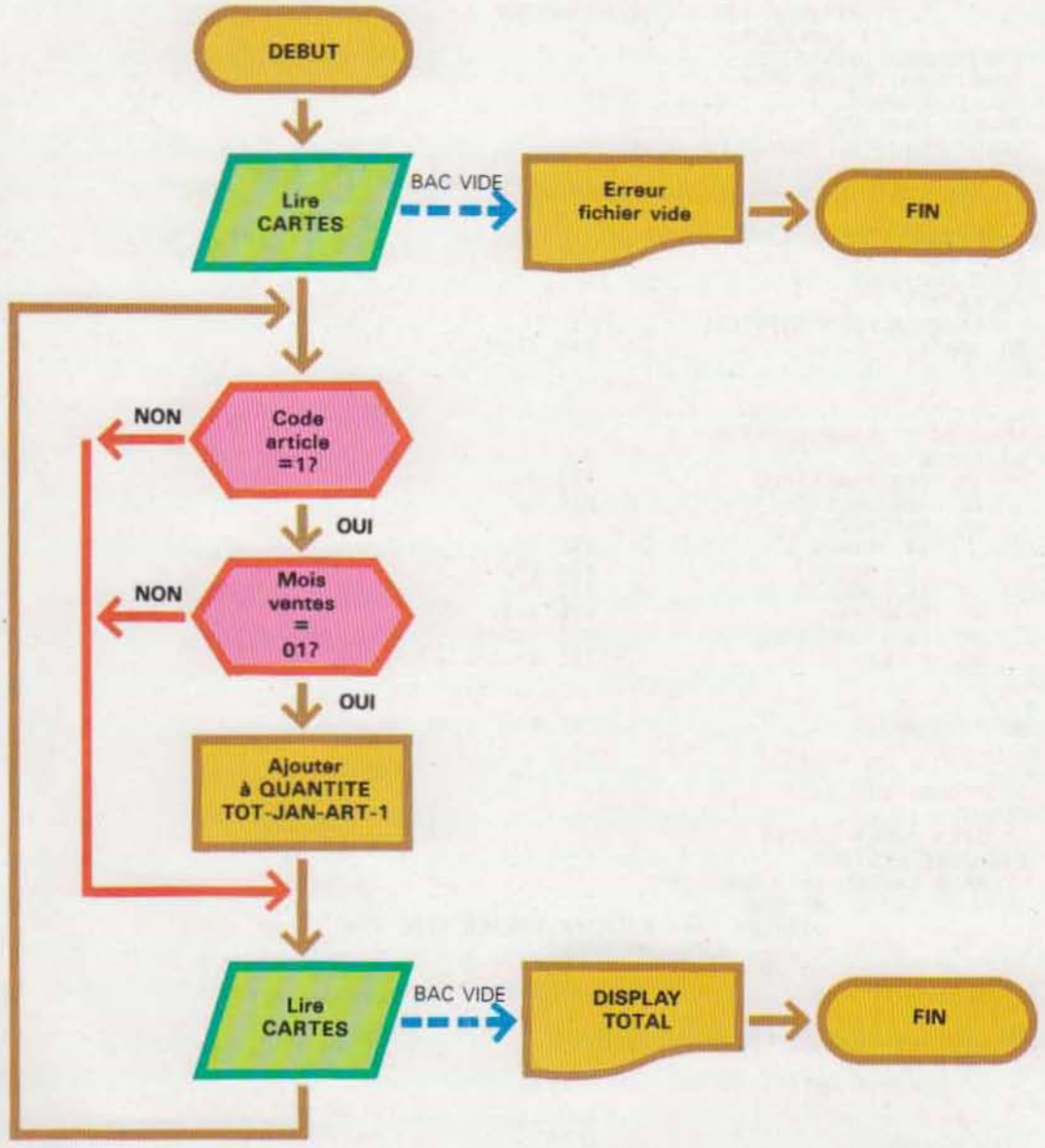
envoie le message ERREUR DE CALCUL à la console.

Gestion des tableaux en Cobol

En Cobol, un tableau est constitué d'un ensemble de champs contigus destinés à contenir des informations homogènes. Les tableaux constituent un des meilleurs moyens dont le Cobol dispose pour gérer des données en mémoire. En effet, si le recours à cette technique pour les petits volumes de données (à gérer directement en mémoire) peut être un choix du programmateur, il est pratiquement impossible de s'en passer lorsqu'on doit manipuler de nombreuses données.
Soit l'exemple d'un fichier de cartes à lire où chaque carte décrit la quantité des articles vendus dans un magasin.
A la fin de la lecture du fichier entier, le total des quantités vendues au mois de janvier du seul article de code 1 doit être imprimé. Un programme très simple suffirait à répondre au problème.
On peut observer comment, après avoir vérifié que la carte lue se réfère à l'article de code-1 et aux ventes de janvier, la somme de la quantité vendue est calculée dans l'unique totalisateur défini : TOT-JAN-ART-1. Ce programme est détaillé page 1088.
Pour s'appuyer sur une représentation visuelle qui nous sera utile par la suite, imaginons la mémoire réservée aux données comme une feuille où le totalisateur TOT-JAN-ART-2 occupe la partie indiquée dans l'encadré du haut de la page 1089.

Considérons maintenant le cas où l'on veut calculer la somme des quantités vendues, toujours de l'article de code 1, mais pour tous les autres mois de l'année.
Dans ce cas, l'organigramme du programme se complique considérablement, bien que le nombre d'informations à gérer soit encore limité (voir page 1089). Le programme Cobol se trouve aux pages 1090 et 1091.
Il n'est pas très pratique d'utiliser des totalisateurs séparés, comme nous venons de le faire, quand le nombre des totalisations devient supérieur à 3 ou 4 ; il devient donc quasiment impossible de la faire quand ce nombre avoisine la centaine ou le millier ce qui est courant en matière de gestion.

LECTURE ET TRAITEMENT D'UN FICHIER (1)



FLUX DES DONNEES



LECTURE ET TRAITEMENT D'UN FICHIER (1)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TABLEAUX.
 REMARKS. CE PROGRAMME EST UN EXEMPLE
          D'INTRODUCTION A L'UTILISATION
          DES TABLEAUX.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  ..........
 OBJECT-COMPUTER.  ..........
 INPUT-OUPUT SECTION.
 FILE-CONTROL.
     SELECT CARTES ASSIGN TO CARD-READER.
*
*
 DATA DIVISION.
 FILE SECTION.
 FD CARTES
    LABEL RECORD OMITTED.
 01 CARTE                          PIC X(80).
*
*
*
 WORKING-STORAGE SECTION.
 01 CARTE-WS.
    05  SERIE-ARTICLE              PIC 9.
    05  CODE-ARTICLE               PIC 9.
    05  DATE-VENTE.
        10  ANNEE                  PIC 99.
        10  MOIS                   PIC 99.
        10  JOUR                   PIC 99.
    05  QUANTITE                   PIC 9(3).
    05  COUT-UNITAIRE              PIC 9(3).
    05  FILLER                     PIC X(66).
*
*
 01 TOT-JAN-ART-1                  PIC 9(6) COMP VALUE 0.
*
*
 PROCEDURE DIVISION.
 DEBUT.
    OPEN INPUT CARTES.
 PREMIERE LECTURE.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '*** FICHIER CARTES VIDE ***'
                        UPON PRINTER
                GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE NOT = 1
         GO TO LIRE-CARTES.
     IF  MOIS = 1
         ADD QUANTITE TO TOT-JAN-ART-1.
*
*
 LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY ' TOTAL VENTES ART.1 JANVIER = '
                        TOT-JAN-ART-1
                        UPON PRINTER
                GO TO FIN-CALCUL.
     GO TO CONTROLE.
*
*
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
```

REPRESENTATION VISUELLE DE LA MEMOIRE AVEC UN TOTALISATEUR UNIQUE

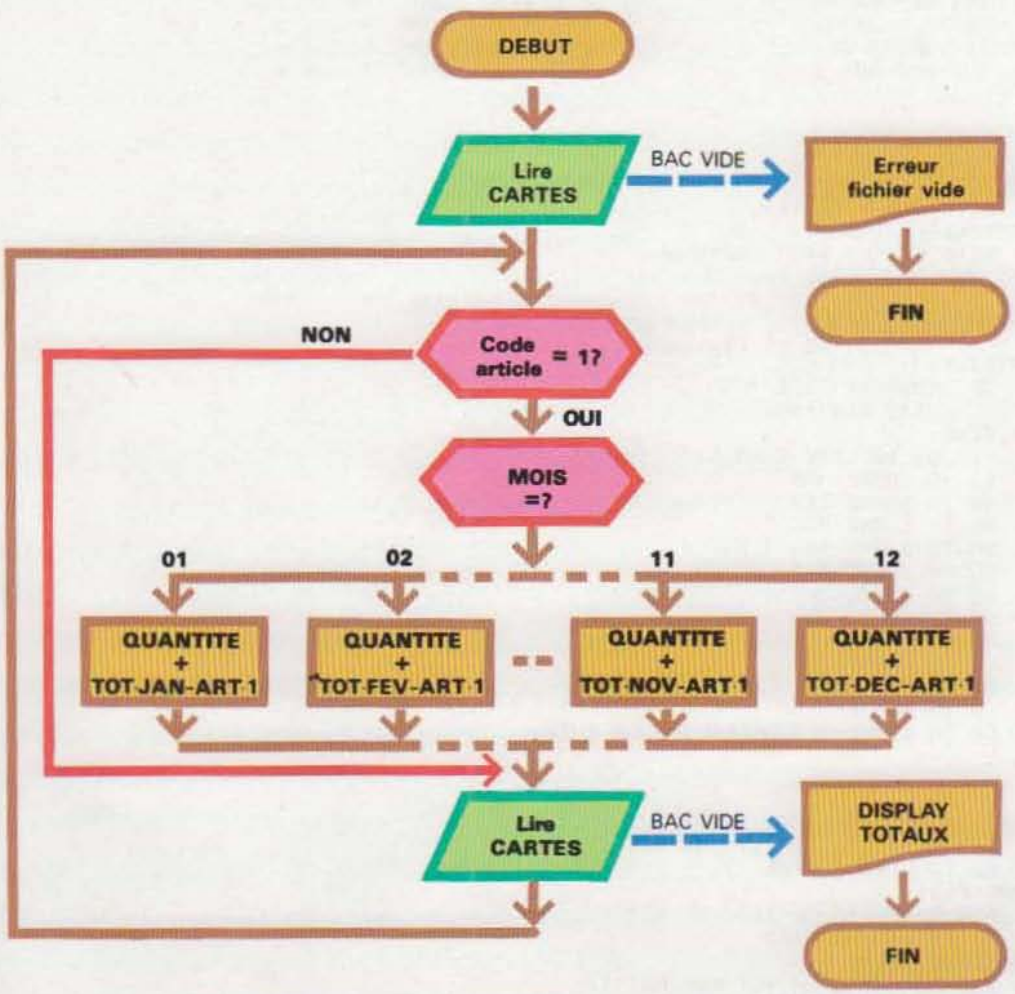


Revenons à la mémoire des données disposée comme une feuille ; la partie occupée par les compteurs réservés aux quantités vendues d'articles du code-1 chaque mois est schématisée comme nous l'avons indiqué en haut de la page 1092.

Tableaux à une dimension

Les problèmes dont nous venons de parler ont une solution très simple. Les totalisateurs ont été déclarés au compilateur comme les 12 éléments d'un tableau. Chacun d'eux peut être adressé avec un nom générique d'élément et son positionnement est exprimé par un nombre dans le tableau.
Autrement dit, si l'on crée un tableau des

LECTURE ET TRAITEMENT D'UN FICHIER (2)



LECTURE ET TRAITEMENT D'UN FICHIER (2)

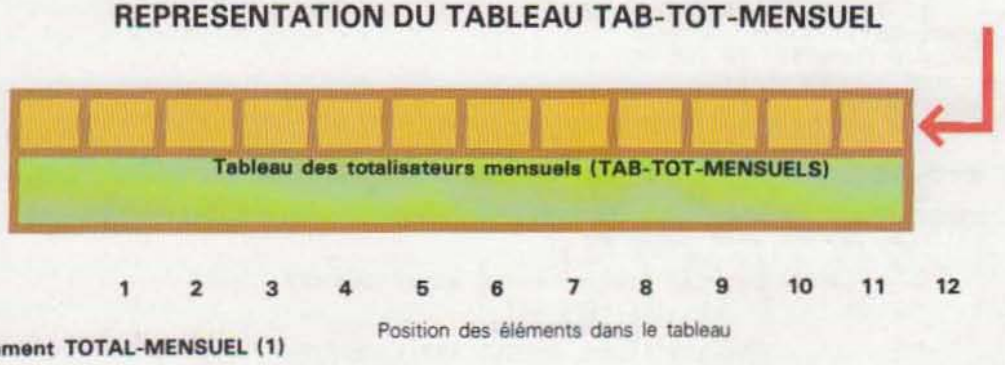
```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE              PIC 9.
     05  CODE-ARTICLE               PIC 9.
     05  DATE-VENTE.
         10  ANNEE                  PIC 9(2).
         10  MOIS                   PIC 9(2).
         10  JOUR                   PIC 9(2).
     05  QUANTITE                   PIC 9(3).
     05  COUT-UNITAIRE              PIC 9(3).
     05  FILLER                     PIC X(66).
*
*
*    ---------- T O T A L I S A T E U R S ----------
*
*
*
 01  TOT-JAN-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-FEV-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAR-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-AVR-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAI-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUI-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUL-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-AOU-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-SEP-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-OCT-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-NOV-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-DEC-ART-1                  PIC 9(6)  COMP  VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE = 1
         NEXT SENTENCE
     ELSE
         GO TO LIRE-CARTES.
     GO TO SOMME-JAN
     GO TO SOMME-FEV
     GO TO SOMME-MAR
     GO TO SOMME-AVR
     GO TO SOMME-MAI
     GO TO SOMME-JUI
     GO TO SOMME-JUL
     GO TO SOMME-AOU
     GO TO SOMME-SEP
     GO TO SOMME-OCT
     GO TO SOMME-NOV
     GO TO SOMME-DEC DEPENDING ON MOIS.
*
*
*
 SOMME-JAN.
     ADD QUANTITE TO TOT-JAN-ART-1.
     GO TO LIRE-CARTES.
 SOMME-FEV.
     ADD QUANTITE TO TOT-FEV-ART-1.
     GO TO LIRE-CARTES.
 SOMME-MAR.
     ADD QUANTITE TO TOT-MAR-ART-1.
     GO TO LIRE-CARTES.
```

```
SOMME-AVR.

    ADD QUANTITE TO TOT-AVR-ART-1.
    GO TO LIRE-CARTES.
SOMME-MAI.
    ADD QUANTITE TO TOT-MAI-ART-1.
    GO TO LIRE-CARTES.
SOMME-JUI.
    ADD QUANTITE TO TOT-JUI-ART-1.
    GO TO LIRE-CARTES.
SOMME-JUL.
    ADD QUANTITE TO TOT-JUL-ART-1.
    GO TO LIRE-CARTES.
SOMME-AOU.
    ADD QUANTITE TO TOT-AOU-ART-1.
    GO TO LIRE-CARTES.
SOMME-SEP.
    ADD QUANTITE TO TOT-SEP-ART-1.
    GO TO LIRE-CARTES.
SOMME-OCT.
    ADD QUANTITE TO TOT-OCT-ART-1.
    GO TO LIRE-CARTES.
SOMME-NOV.
    ADD QUANTITE TO TOT-NOV-ART-1.
    GO TO LIRE-CARTES.
SOMME-DEC.
    ADD QUANTITE TO TOT-DEC-ART-1.
LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY 'TOTAL VENTES ART.1 JANVIER   = '
                        TOT-JAN-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 FEVRIER   = '
                        TOT-FEV-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 MARS      = '
                        TOT-MAR-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 AVRIL     = '
                        TOT-AVR-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 MAI       = '
                        TOT-MAI-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 JUIN      = '
                        TOT-JUI-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 JUILLET   = '
                        TOT-JUL-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 AOUT      = '
                        TOT-AOU-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 SEPTEMBRE = '
                        TOT-SEP-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 OCTOBRE   = '
                        TOT-OCT-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 NOVEMBRE  = '
                        TOT-NOV-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 DECEMBRE  = '
                        TOT-DEC-ART-1
                        UPON PRINTER
                GO TO FIN-CALCUL.
    GO TO CONTROLE.
FIN-CALCUL.
    CLOSE CARTES.
    STOP RUN.
```

REPRESENTATION FIGUREE DE LA MEMOIRE AVEC 12 TOTALISATEURS SEPARES



REPRESENTATION DU TABLEAU TAB-TOT-MENSUEL



DESCRIPTION D'UN TABLEAU EN DATA-DIVISION

```
01 nom-tableau.
   05 nom-élément    PIC... USAGE... OCCURS N TIMES
```

totaux mensuels contenant 12 éléments, de nom générique TOTAL-MENSUEL, le compteur du mois d'avril sera adressé comme TOTAL-MENSUEL (4) c'est-à-dire comme le champ TOTAL-MENSUEL qui occupe la 4e position dans le tableau.

Le tableau est une donnée composée (ou structurée) dans laquelle les données composantes, (qui ne sont pas nécessairement élémentaires), ont toutes le même nom et peuvent être adressées seulement en fonction de leur position (voir 2e schéma de cette page).

Le compilateur doit évidemment savoir qu'une certaine donnée constitue un tableau. Cette information lui est fournie par le programmeur dans la DATA-DIVISION, lors de la description du champ, selon le format ci-dessus.

A l'aide de cette description, le compilateur réserve un espace « nom tableau » constitué par la série d'éléments « nom-élément ». L'occupation mémoire est identique à celle que l'on aurait obtenue si N champs avaient été définis séparément. Différence essentielle avec le tableau TAB-TOT-MENSUELS : au lieu des 12 compteurs TOT-JAN-ART-1... TOT-FEV-ART-1, le programme de l'exemple précédent est certainement moins fastidieux à détailler et plus aisé à lire grâce à la réduction draconienne du nombre d'instructions présentes (voir listing en pages 1093 et 1094).

L'exemple décrit dans ce programme contient certaines notions importantes qu'il est bon de souligner.

Pour éviter d'écrire, en fin de traitement,

LECTURE ET TRAITEMENT D'UN FICHIER (3)

```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE              PIC 9.
     05  CODE-ARTICLE               PIC 9.
     05  DATE-VENTE.
         10  ANNEE                  PIC 9(2).
         10  MOIS                   PIC 9(2).
         10  JOUR                   PIC 9(2).
     05  QUANTITE                   PIC 9(3).
     05  COUT-UNITAIRE              PIC 9(3).
     05  FILLER                     PIC X(66).
*
*
*    ------------  T A B L E A U X  -----------
*
**** TABLEAU TOTALISATEURS DES QUANTITES MENSUELLES ART.1 ****
*
*
 01  TAB-TOT-MENSUELS.
     05  TOTAL-MENSUEL              PIC 9(6)  COMP OCCURS 12.
**** TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS.
     05  FILLER                     PIC X(9)  VALUE 'JANVIER   '.
     05  FILLER                     PIC X(9)  VALUE 'FEVRIER   '.
     05  FILLER                     PIC X(9)  VALUE 'MARS      '.
     05  FILLER                     PIC X(9)  VALUE 'AVRIL     '.
     05  FILLER                     PIC X(9)  VALUE 'MAI       '.
     05  FILLER                     PIC X(9)  VALUE 'JUIN      '.
     05  FILLER                     PIC X(9)  VALUE 'JUILLET   '.
     05  FILLER                     PIC X(9)  VALUE 'AOUT      '.
     05  FILLER                     PIC X(9)  VALUE 'SEPTEMBRE '.
     05  FILLER                     PIC X(9)  VALUE 'OCTOBRE   '.
     05  FILLER                     PIC X(9)  VALUE 'NOVEMBRE  '.
     05  FILLER                     PIC X(9)  VALUE 'DECEMBRE  '.
 01  TABLEAU-MOIS  REDEFINES   MOIS.
     05  NOM-MOIS                   PIC X(9)  OCCURS 12.
*
 01  INDICE-MOIS                    PIC 9(2)  COMP.
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 DEBUT.
     MOVE LOW-VALUES TO TAB-TOT-MENSUELS.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER


                 GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE = 1
         ADD QUANTITE TO TOTAL-MENSUEL (MOIS).
     READ CARTES INTO CARTE-WS
                 AT END
                 PERFORM AFFICHAGE-TOTAUX
                 GO TO FIN-CALCUL.
     GO TO CONTROLE.
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
```

```
*
*============  S E C T I O N S  ============
*
*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
     MOVE ZEROES TO INDICE-MOIS.
 BOUCLE-AFFICHAGE.
     ADD 1 TO INDICE-MOIS.
     IF  INDICE-MOIS > 12
         GO TO BOUCLE-AFFICHAGE-EX.
     DISPLAY ' TOTAL VENTES ART.1 '
             NOM-MOIS (INDICE-MOIS)
             ' = '
             TOTAL-MENSUEL (INDICE-MOIS)
             UPON PRINTER.
     GO TO BOUCLE-AFFICHAGE.
 BOUCLE-AFFICHAGE-EX. EXIT.
*
```

12 instructions DISPLAY contenant les messages :

TOTAL-VENTES ART. 1 mois = total-mois

on fait appel à un autre tableau dans lequel chaque élément contient, en clair, le nom d'un des douze mois de l'année. De cette façon, le message à imprimer devient paramétré et il peut être répété 12 fois en reportant, à chaque fois, le nom du mois et le nombre des ventes correspondantes.
La description d'un tableau ne peut contenir la clause VALUE. Quand on a besoin de mémoriser des valeurs constantes et prédéfinies dans les éléments d'un tableau, on recourt à la technique utilisée dans l'exemple, à savoir :

1 / On commence par définir le champ composé d'autant de sous-champs qu'il y a d'éléments dans le tableau (dans ce cas, 12). Il n'est pas nécessaire d'utiliser des noms définis par le programmeur pour nommer les sous-champs ; le nom réservé FILLER est suffisant. Les sous-champs utilisés, tous de même longueur, commencent à la valeur qui nous intéresse. Dans notre exemple, la longueur d'un champ est égale à celle du nom le plus long (SEPTEMBRE).

2 / Avec la clause REDEFINES du champ précédent, on décrit un champ-tableau et on précise le nombre et le nom des éléments dans le tableau (clause OCCURS).

Avec la technique des tableaux, on peut ainsi adresser un élément générique. Celui-ci, grâce à la clause REDEFINES, partage la partie mémoire du champ redéfini avec la valeur imposée. Par rapport au champ MOIS et au TABLEAU-MOIS qui le définit, la partition de la mémoire fournit le schéma ci-contre. La section AFFICHAGE-TOTAUX, qui intervient à la fin de la lecture du fichier CARTES par l'intermédiaire du verbe PERFORM, utilise le champ INDICE-MOIS pour se positionner simultanément sur les éléments des tableaux TABLEAU-MOIS et TAB-TOT-MENSUELS.
La valeur d'INDICE-MOIS est incrémentée de 1 jusqu'à 12. Pour chaque valeur prise par l'indice, le nom du mois correspondant (par exemple 1 = JANVIER) et le contenu de la quantité totalisée dans ce mois sont prélevés de façon à composer le message désiré (voir ci-contre le diagramme de construction des 12 messages).
Une attention particulière doit être accordée aux cycles de lecture ou d'écriture des données dans le tableau.
Le compilateur n'est pas capable de se rendre

compte si un indice assume une valeur supérieure ou inférieure à celles accordées aux dimensions du tableau. Si, par exemple, en lecture de TABLEAU-MOIS, l'indice est imposé de façon erronée à la valeur 13 et si la situation en mémoire est celle décrite en haut de la page 1096, le message composé s'écrit :

TOTAL VENTES ART. 1 ? <A47Y/&=S78)(5

Ici, l'erreur saute aux yeux, mais ce n'est pas toujours le cas : parfois, la gestion erronée d'un indice peut entraîner une anomalie totale des résultats ou la fin anormale du programme.

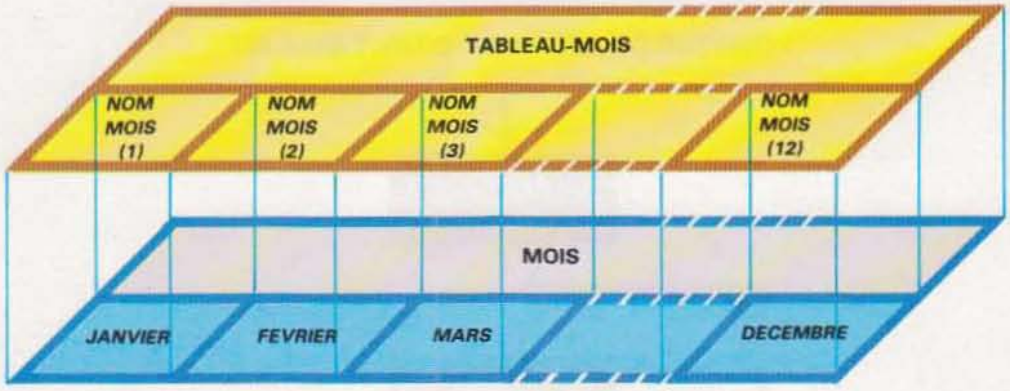
Nous reviendrons plus longuement sur ce sujet lorsque nous parlerons du chargement d'un tableau. Pour le moment, le lecteur doit observer la section AFFICHAGE-TOTAUX de l'exemple. Son organigramme (page 1096) est très simple. Quand il est adopté pour la lecture de toutes les données d'un tableau, il permet d'éviter les erreurs précédemment décrites.

Tableaux à deux dimensions

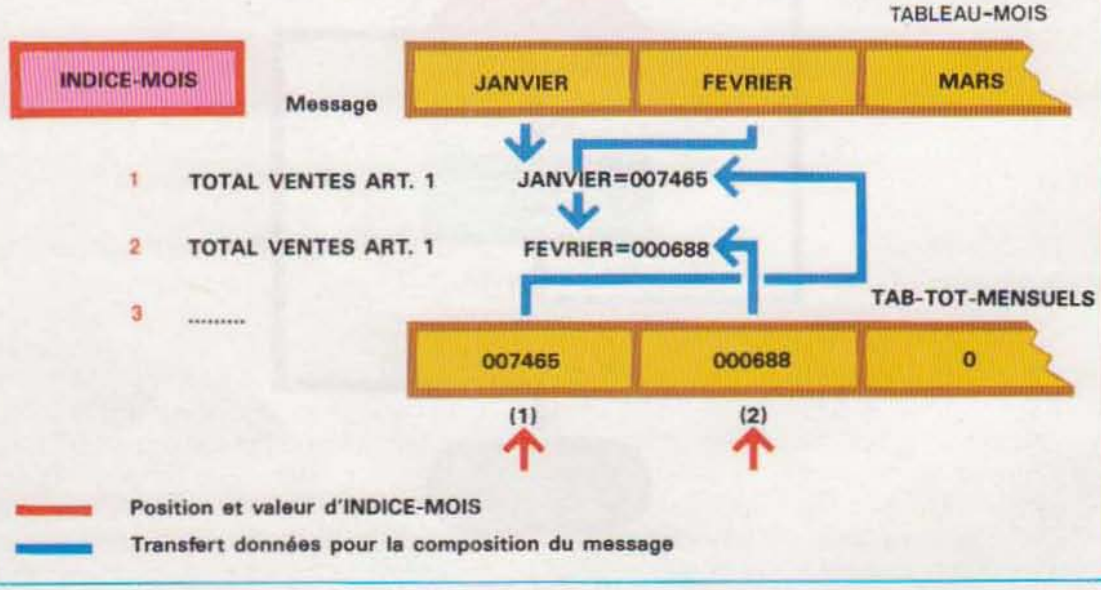
Considérons à nouveau l'exemple exposé au début.

On remarque qu'un programme capable de traiter le total des ventes mensuelles d'un

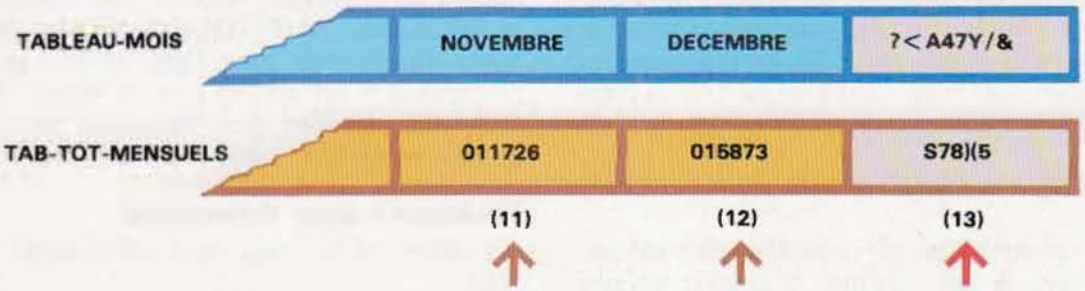
ATTRIBUTION DE VALEURS CONSTANTES A UN TABLEAU



COMPOSITION DU CHAMP D'IMPRESSION



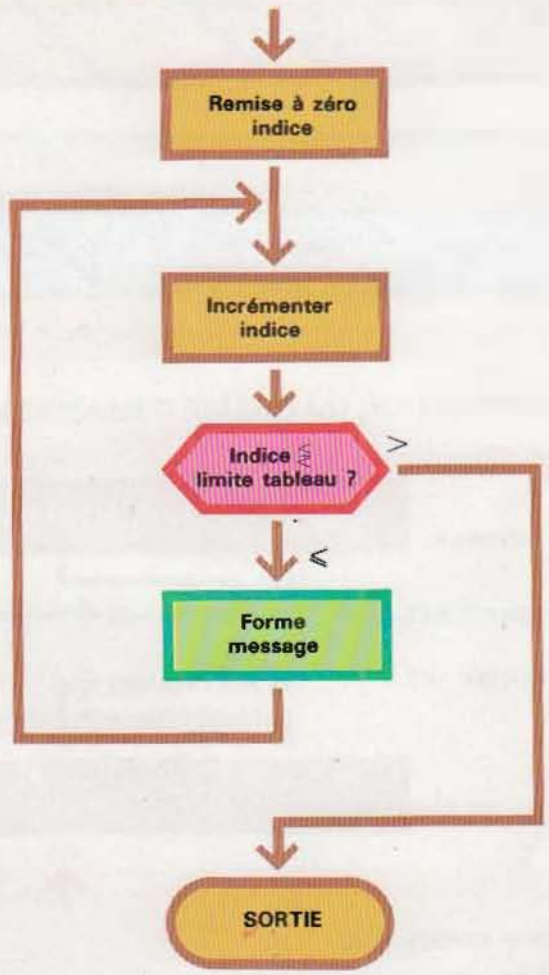
TAILLE D'UN TABLEAU AVEC DES VALEURS D'INDICE NON PERMISES



Champs adjacents aux tableaux mais n'appartenant pas aux tableaux mêmes

Position du pointeur avec INDICE-MOIS=13

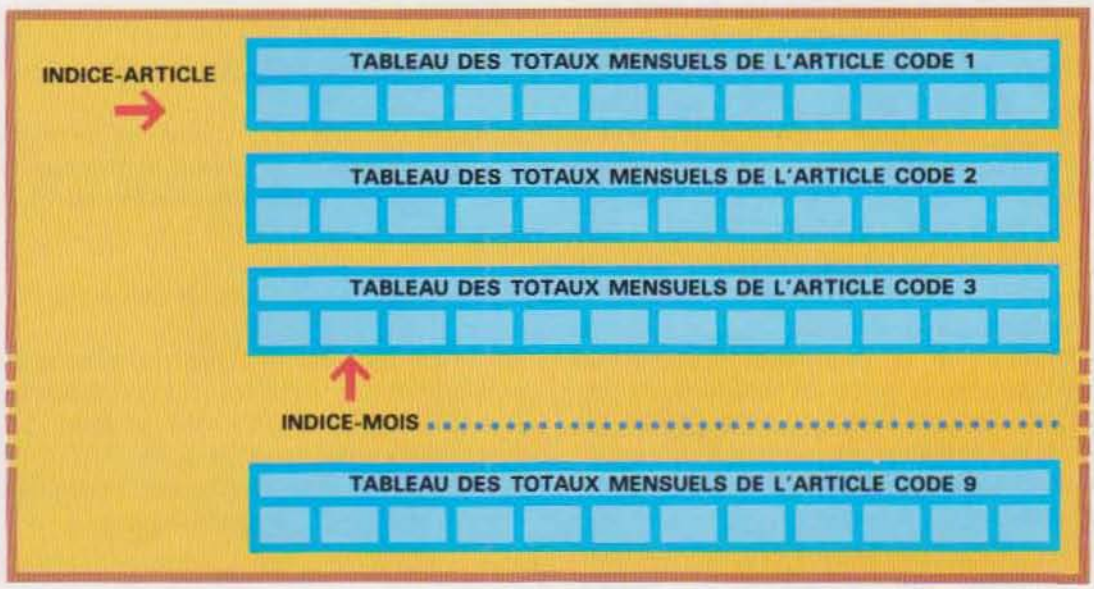
LOGIQUE DE LECTURE D'UN TABLEAU



article unique n'a que peu d'intérêt. Pour simplifier, convenons que le code article est représenté par un seul caractère numérique et extrapolons le type de traitement décrit plus haut (article de code 1) à la gestion de tous les articles codés de 1 à 9. Poursuivant cette généralisation, on peut également gérer la partie de mémoire dédiée aux totalisateurs des quantités mensuelles (voir schéma ci-dessous).
La « feuille mémoire » tout entière est alors assimilée à un tableau dans lequel chaque élément se comporte comme un tableau. Pour lire ou pour modifier le totalisateur du mois de janvier de l'article de code 2, il faudra d'une part se positionner sur la ligne numéro 2 (INDICE-ARTICLE) et d'autre part sur la première position (JANVIER=1) du tableau des totalisateurs mensuels (INDICE-MOIS).
La situation se rapproche beaucoup du jeu de la « bataille navale » dans lequel on communique à l'adversaire sa propre manœuvre en fournissant les codes de la ligne et de la colonne pour « cibler » le bateau ennemi.
Avant de fournir l'organigramme de la nouvelle version du programme, arrêtons-nous un instant sur la manière d'écrire le « tableau des tableaux ». Puisque on a prévu de ne traiter que 9 articles, chacun d'entre eux devra correspondre à un élément du tableau, lui-même ensuite défini comme un tableau de 12 compteurs dans lequel est totalisée la quantité mensuelle vendue par article.
Finalement, l'occupation mémoire sera de 9 lignes × 12 colonnes × 4 octets = 432 octets. Rappelons-nous, en effet, qu'un champ numérique de longueur déclarée entre 4 et 9 caractères occupe en réalité 4 octets en USAGE-COMP. Dans notre cas, chaque compteur est décrit PIC 9(6) COMP, occupant par conséquent 4 octets. Le tableau des totalisateurs des articles TAB-TOT-ART sera décrit :

```
01 TAB-TOT-ART.
   05 TOTAUX-ARTICLE          OCCURS 9.
      10 TAB-TOT-MENSUELS.
         15 TOTAL-MENSUEL PIC 9(6)
                         COMP
                         OCCURS 12.
```

SCHEMA D'UN TABLEAU A DEUX DIMENSIONS TAB-TOT-ART.





Finsiel

Le stylo optique relié à une caisse enregistreuse permet de lire les informations sur l'emballage.

Le programme détaillé se trouve en 1099 et 1100. Bien que, dans cette nouvelle version, le traitement concerne 9 × 12 = 108 totalisateurs et non plus seulement 12, on peut voir que le nombre d'instructions dans la MAIN SECTION n'a pratiquement pas changé.
La seconde remarque concerne la possibilité, offerte par les tableaux, d'initialiser chacun de leurs éléments avec une seule instruction quand ils sont constitués de champs homogènes. La ligne MOVE LOW-VALUES TO TAB-TOT-ART, qui renvoie au plus haut niveau du tableau (01), initialise à LOW-VALUES tous les éléments de niveau inférieur.
La fonction de la procédure AFFICHAGE-TOTAUX, activée en fin de lecture du fichier CARTES, consiste à lire le contenu entier du tableau des totalisateurs, mettant ainsi en évidence les totaux des ventes de chaque mois en face de chaque article. On se reportera au diagramme de lecture page 1101 qui met bien en évidence les instructions de lecture « horizontale » (celle des compteurs mensuels se rapportant au même article) et les instructions de pointage « vertical », (le positionnement sur la ligne). On notera, par exemple, que le positionnement sur le totalisateur de mai pour l'article de code 9 est effectué par l'élément :

TOTAL-MENSUEL (9, 5)

La façon correcte, pour fournir les références d'un élément dans un tableau de tableaux (ou mieux, dans un tableau à deux dimensions), est la suivante :

nom-élément (indice-ligne, ␣ indice-colonne)

Un blanc (espace) doit nécessairement être présent entre la virgule et l'indice de colonne. Dans le listing de la page 1102, seuls les résultats relatifs aux articles de codes 1, 2, 3 et 4 ont été portés, pour des raisons évidentes de compréhension du programme.

Tableaux à trois dimensions

De notre exposé sur les tableaux, il ressort qu'un seul indice suffit dans le TABLEAU-MOIS pour adresser complètement l'élément

UTILISATION D'UN TABLEAU A DEUX DIMENSIONS

```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE            PIC 9.
     05  CODE-ARTICLE             PIC 9.
     05  DATE-VENTE.
         10  ANNEE                PIC 9(2).
         10  MOIS                 PIC 9(2).
         10  JOUR                 PIC 9(2).
     05  QUANTITE                 PIC 9(3).
     05  COUT-UNITAIRE            PIC 9(3).
     05  FILLER                   PIC X(66).
*
*
*    ------------  T A B L E A U X  -----------
*
**** TABLEAU TOTALISATEURS DES QUANTITES MENSUELLES ART.1 ****
*
*
 01  TAB-TOT-ART.
     05  TOTAUX-ARTICLES          OCCURS 9.
         10  TAB-TOT-MENSUELS.
             15  TOTAL-MENSUEL    PIC 9(6) COMP OCCURS 12.
*
*
**** TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS.
     05  FILLER                    PIC X(9)  VALUE 'JANVIER   '.
     05  FILLER                    PIC X(9)  VALUE 'FEVRIER   '.
     05  FILLER                    PIC X(9)  VALUE 'MARS      '.
     05  FILLER                    PIC X(9)  VALUE 'AVRIL     '.
     05  FILLER                    PIC X(9)  VALUE 'MAI       '.
     05  FILLER                    PIC X(9)  VALUE 'JUIN      '.
     05  FILLER                    PIC X(9)  VALUE 'JUILLET   '.
     05  FILLER                    PIC X(9)  VALUE 'AOUT      '.
     05  FILLER                    PIC X(9)  VALUE 'SEPTEMBRE '.
     05  FILLER                    PIC X(9)  VALUE 'OCTOBRE   '.
     05  FILLER                    PIC X(9)  VALUE 'NOVEMBRE  '.
     05  FILLER                    PIC X(9)  VALUE 'DECEMBRE  '.
 01  TABLEAU-MOIS  REDEFINES   MOIS.
     05  NOM-MOIS                  PIC X(9)  OCCURS 12.
*
*
 01  INDICE-ARTICLE                PIC 9  COMP.
 01  INDICE-MOIS                   PIC 9(2)  COMP.
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 DEBUT.
     MOVE LOW-VALUES TO TAB-TOT-ART.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 GO TO FIN-CALCUL.
 SOMME.
     ADD QUANTITE TO TOTAL-MENSUEL (CODE-ARTICLE, MOIS).
     READ CARTES INTO CARTE-WS
                 AT END
                 PERFORM AFFICHAGE-TOTAUX
                 GO TO FIN-CALCUL.
     GO TO SOMME.
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
*
```

```
* ============  S E C T I O N S  ============

*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
     MOVE ZEROES TO INDICE-ARTICLE.
 AFFICHAGE-ARTICLE.
     ADD 1 TO INDICE-ARTICLE.
     IF  INDICE-ARTICLE > 9
         GO TO AFFICHAGE-ARTICLE-EX.
     DISPLAY SPACES UPON PRINTER.
     DISPLAY ' ARTICLE COD. '
             INDICE-ARTICLE
             '     '
             '* * * TOTAUX VENTES MENSUELLES * * *'
             UPON PRINTER.
     DISPLAY SPACES UPON PRINTER.
     MOVE ZEROES TO INDICE-MOIS.
 AFFICHAGE-VENTES.
     ADD 1 TO INDICE-MOIS.
     IF  INDICE-MOIS > 12
         GO TO AFFICHAGE-ARTICLE.
     DISPLAY '
             NOM-MOIS (INDICE-MOIS)
             ' = '
             TOTAL-MENSUEL (INDICE-ARTICLE, INDICE-MOIS)
             UPON PRINTER.
     GO TO AFFICHAGE-VENTES.
 AFFICHAGE-ARTICLE-EX. EXIT.
```

contenant la description du mois. Inversement, pour adresser complètement le totalisateur de la quantité vendue de l'article de code 2 au mois de mai, il faut deux indices. Ils permettent le double positionnement, d'abord sur la ligne correspondante à l'article (2) et, ensuite, au milieu de celle-ci, sur la colonne se rapportant au mois de mai (5).

Dans le premier cas (TABLEAU-MOIS), nous avons parlé d'un tableau à une dimension alors que dans l'exemple précédent, il s'agissait d'un tableau à deux dimensions.

Bien noter que le Cobol autorise une définition de tableaux avec un maximum de trois dimensions.

Poursuivons avec le même exemple : nous continuons à totaliser les quantités vendues mensuellement pour chaque article et pouvant appartenir à 9 série différentes.

Pour la clarté de la démonstration, supposons qu'une carte contient les valeurs :

SERIE = 2
ARTICLE = 5
MOIS = 12
QUANTITE = 007215

et une autre, ces valeurs :

SERIE = 4
ARTICLE = 5
MOIS = 12
QUANTITE = 000036

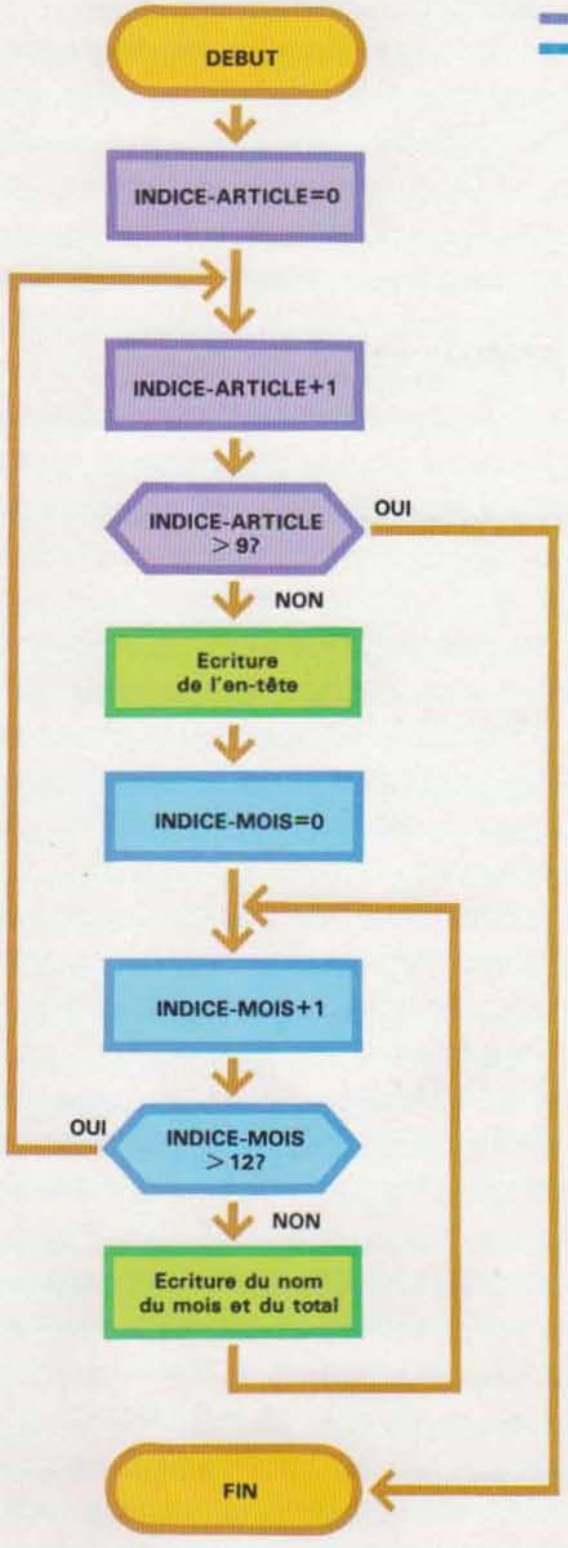
Malgré l'identité des valeurs respectives d'ARTICLES et de MOIS, il est évident que les quantités doivent être totalisées dans deux compteurs distincts.

Nous retrouvons ce découpage de la mémoire réservée aux totalisateurs mensuels page 1103. La mémoire est représentée par une série de « feuilles » adressées par le nombre de SERIE. Chaque feuille est ensuite traitée et adressée comme dans l'exemple précédent : une ligne correspond à un article et un champ de la ligne au contenu (totalisateur) des ventes mensuelles.

En WORKING-STORAGE SECTION, le tableau s'écrit :

```
01 TAB-TOT-VENTES
   05 TOTAUX-SERIE              OCCURS 9.
      10 TOTAUX-ARTICLES        OCCURS 9.
         15 TOTAL-MENSUEL PIC 9 (6)
                         COMP
                         OCCURS 12.
```

ORGANIGRAMME DE LA PROCEDURE DISPLAY-TOTAUX



Instructions pour lecture "verticale"
Instructions pour lecture "horizontale"

Remise à zéro de l'indice de ligne INDICE-ARTICLE

Incrémentation de l'indice de ligne

Contrôle de la position maximale "verticale" atteinte

En-tête
ARTICLE COD. N. *** TOTAUX VENTES MENSUELLES ** où N est le code générique en cours de traitement

Remise à zéro de l'indice de colonne INDICE-MOIS pour la lecture des compteurs mensuels

Incrémentation de l'indice de colonne

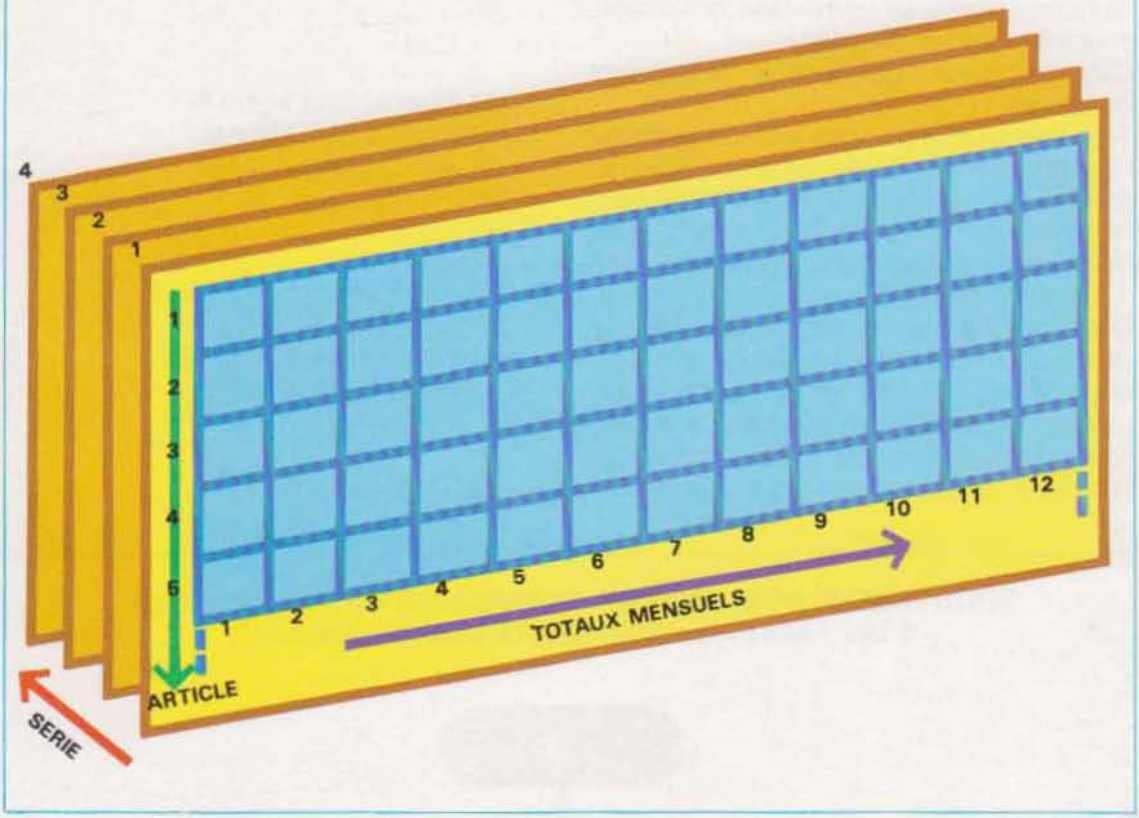
Contrôle de la position maximale "horizontale" atteinte

Le nom du mois et le total des ventes sont affichés pour l'article traité

EXEMPLE D'IMPRESSION DES DONNEES D'UN TABLEAU A DEUX DIMENSIONS

```
ARTICLE COD.1   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006001
                MAI         = 003564
                JUIN        = 001234
                JUILLET     = 001234
                AOUT        = 006574
                SEPTEMBRE   = 012564
                OCTOBRE     = 010084
                NOVEMBRE    = 011284
                DECEMBRE    = 010064

ARTICLE COD.2   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006002
                MAI         = 003564
                JUIN        = 002234
                JUILLET     = 002234
                AOUT        = 006574
                SEPTEMBRE   = 022564
                OCTOBRE     = 020084
                NOVEMBRE    = 021284
                DECEMBRE    = 020064

ARTICLE COD.3   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006001
                MAI         = 003564
                JUIN        = 001334
                JUILLET     = 001334
                AOUT        = 006574
                SEPTEMBRE   = 013564
                OCTOBRE     = 010084
                NOVEMBRE    = 011384
                DECEMBRE    = 010064

ARTICLE COD.4   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006004
                MAI         = 003564
                JUIN        = 004334
                JUILLET     = 041334
                AOUT        = 006574
                SEPTEMBRE   = 012564
                OCTOBRE     = 010084
                NOVEMBRE    = 011284
                DECEMBRE    = 010064
```

SCHEMA D'UN TABLEAU A TROIS DIMENSIONS
TAB-TOT-VENTES



Chargement d'un tableau

Dans le listing de la page 1102, on remarque que le mot ARTICLE COD. 1 prévoit, côté utilisateur, la consultation ultérieure d'un indice qui mette bien en évidence la description de l'article de code 1.
En réalité, la consultation de l'indice (qui a pour but de vérifier la correspondance entre le code et la description) doit être effectuée par le même programme, ce qui nécessite la production de l'en-tête suivant :

ARTICLE : description-article

La correspondance peut être établie en utilisant un tableau dans lequel chaque élément contient :

1 / le nombre de série (1 caractère)
2 / le code article (1 caractère)
3 / la description de l'article (30 caractères).

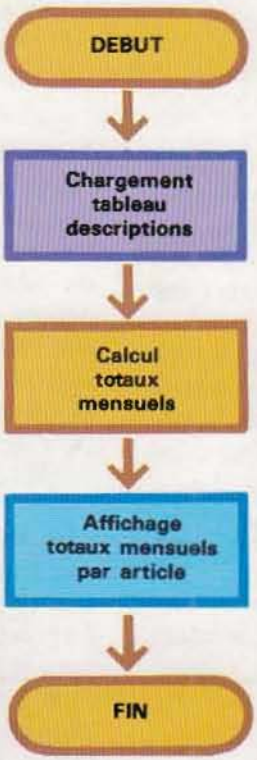
Puisqu'il est logique de penser que de nouveaux articles seront insérés et que certains des paramètres déjà catalogués devront être modifiés, il est préférable que le chargement du tableau de description soit effectué en-dehors de chaque exécution du programme, par l'intermédiaire d'un fichier de cartes, par exemple.
Si l'on adoptait la technique déjà employée pour le tableau des noms de mois, chaque modification du contenu du tableau serait suivie de la correction de la description correspondante dans WORKING-STORAGE-SECTION et de la recompilation du programme.
La procédure est analogue à celle déjà utilisée pour le calcul des totaux des ventes.
Avec une différence, toutefois : alors que, dans ce dernier cas, on pouvait adresser directement le compteur d'article (grâce aux codes de série, article et mois), entreprendre le char-

gement d'un tableau impose un autre type de gestion de l'indice.

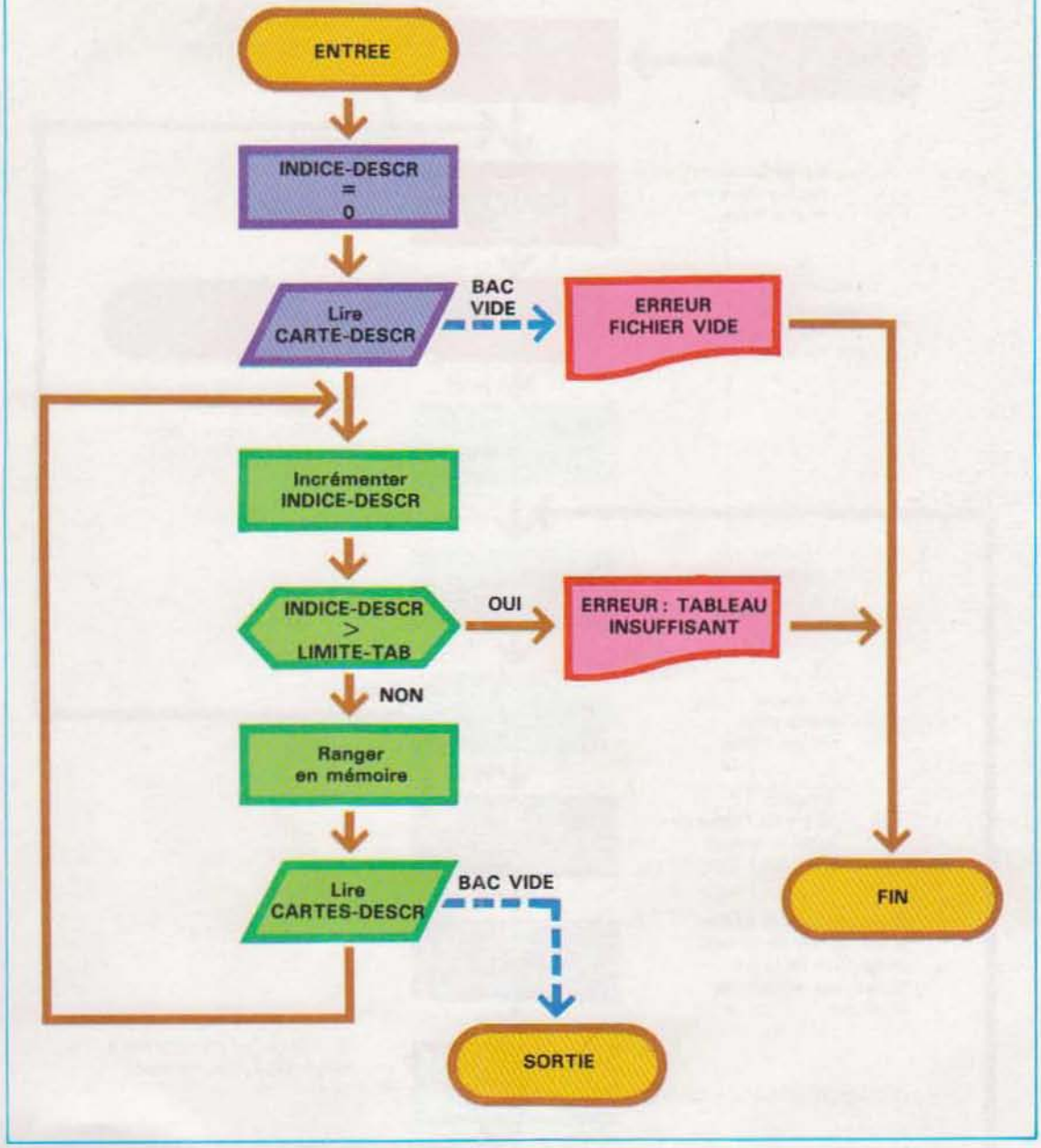
L'indice ou les indices du tableau à charger doivent être gérés et contrôlés à chaque pas, de façon à obtenir un remplissage sans discontinuité de l'espace disponible.
Ceci afin d'éviter notamment un débordement du tableau.
Tout cela paraîtra plus clair après avoir étudié l'exemple suivant. Dans l'organigramme ci-dessous, nous avons représenté le chargement du tableau de description des articles. Chaque bloc peut être, à son tour, détaillé (voir page 1105, 1106 et 1107).
On suppose que les cartes décrivant les articles sont disposées à l'intérieur du fichier par ordre croissant de série et d'article. Dans l'hypothèse où les descriptions des 9 articles (vêtements par exemple) et des 9 series sont présentes, le fichier des cartes-description peut être organisé comme suit :

11 CHEMISES HOMME
12 CHEMISES FEMME
13 CHEMISES ENFANT
14 .....
15 .....
..
..
21 PANTALONS HOMME
22 PANTALONS FEMME
23 PANTALONS ENFANT
24 .....
25 .....
..
..
..
91 CHAUSSURES HOMME
92 CHAUSSURES FEMME
..
..
99 SANDALES ENFANT

PROGRAMME UTILISANT LE CHARGEMENT ET LA GESTION D'UN TABLEAU (ORGANIGRAMME SOMMAIRE)



PHASE DE CHARGEMENT DU TABLEAU DES DESCRIPTIONS

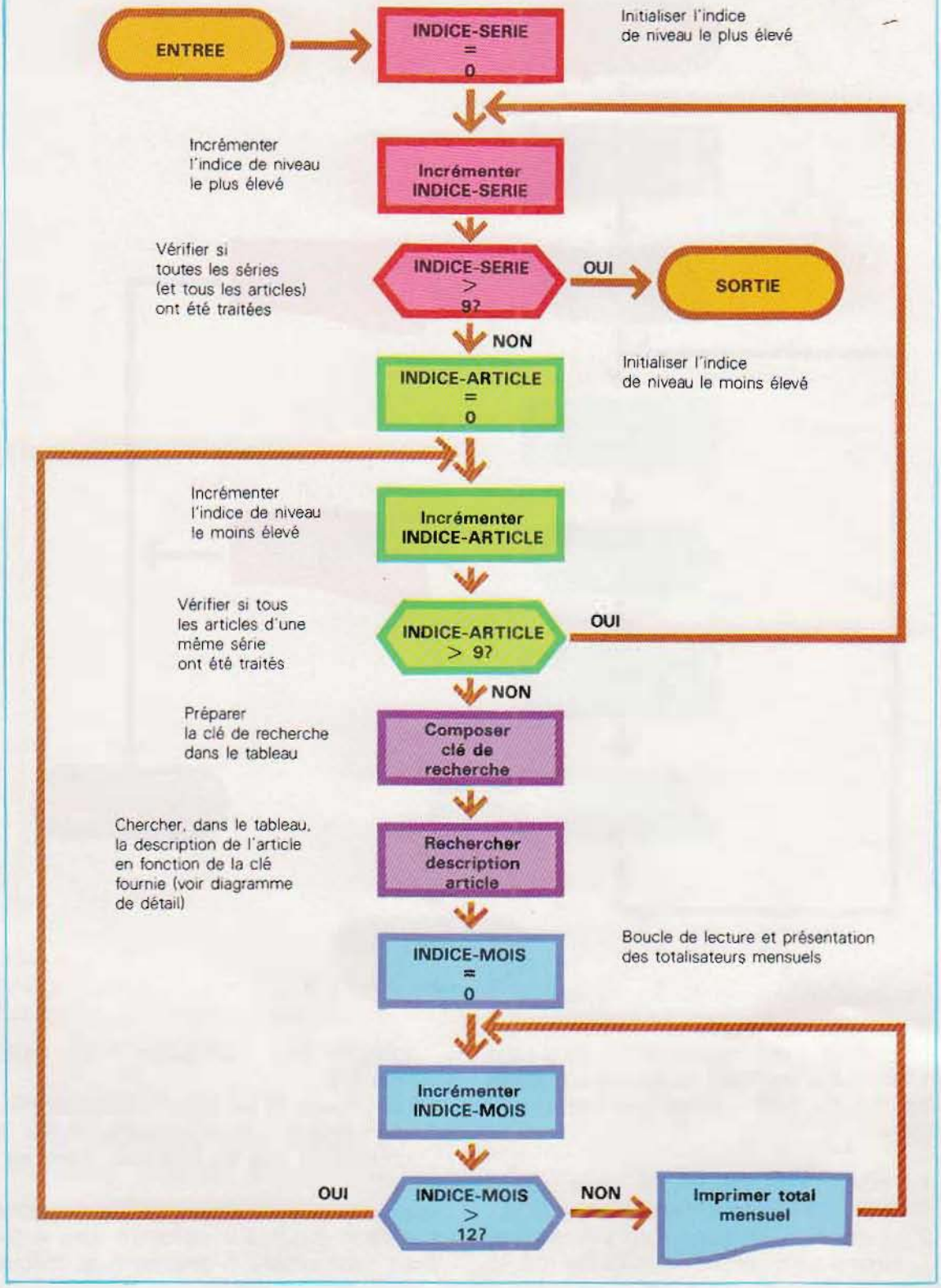


On constate, dans l'organigramme de la page 1106, que la technique de lecture du tableau est encore celle utilisée précédement, à savoir :

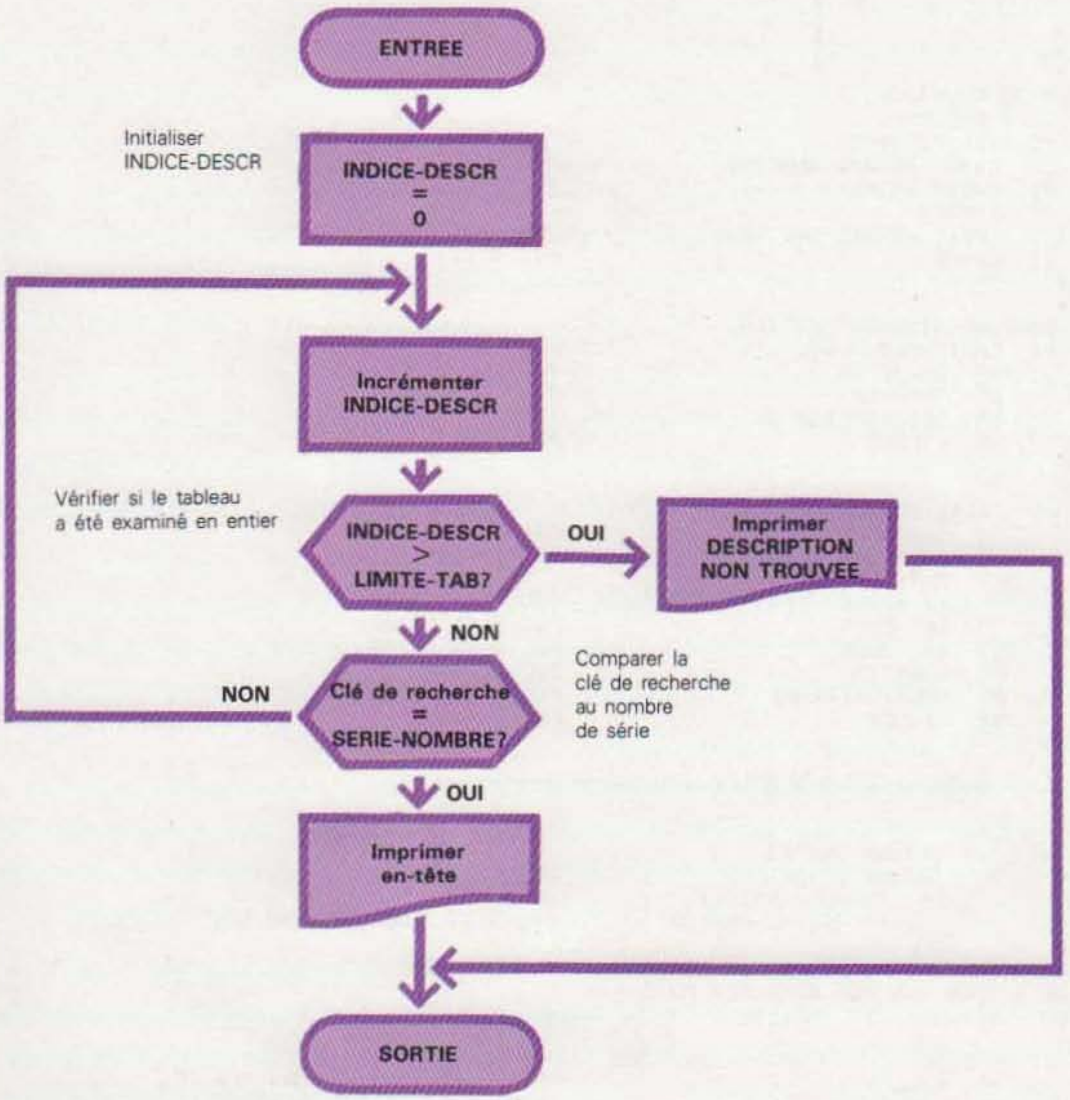
1 / l'indice de niveau le plus élevé (INDICE-SERIE) est incrémenté et contrôlé ;
2 / la valeur de cet indice n'est pas modifiée avant le traitement de la totalité des articles (contrôle pour INDICE-CONTROLE supérieur à 9) ;
3 / Une boucle de test est effectuée pour chaque valeur de l'indice pour lire et afficher le contenu de tous les totaliseurs mensuels.

Une importance particulière doit être accordée à la partie dédiée à la recherche dans le tableau, pour extraire la description de chaque

PHASE DE LECTURE DES CARTES-DESCRIPTION



PHASE DE RECHERCHE DE LA DESCRIPTION D'UN ARTICLE



article (page 1107). Puisqu'il s'agit de mettre en évidence la description de tous les aspects d'une gestion commerciale, un champ utilitaire (clé de recherche), indiquant les valeurs du nombre de série et du code des articles est composé pour chacun d'eux.

Le champ-clé est donc itérativement comparé aux deux premiers caractères de chaque élément du tableau des descriptions.

La boucle s'interrompt lorsque les deux champs comparés s'avèrent égaux : dans ce cas, on prélève la partie DESCRIPTION de l'élément pour composer la ligne du titre.

Il est alors prudent d'envisager le cas où l'égalité décrite n'est pas vérifiée ; dans cette éventualité, il est alors prévu de remplacer la description de l'article par ses codes indentificateurs (série et nombre) accompagnés du commentaire DESCRIPTION NON TROUVEE. On verra le programme détaillé pages 1108 à 1110. Les descriptions de certaines parties, pour lesquelles nous renvoyons le lecteur aux exemples précédents, ont été volontairement omises.

PROGRAMME POUR LE CHARGEMENT ET LA GESTION D'UN TABLEAU

```
*
*
*
 DATE DIVISION.
 FILE SECTION.
 FD  CARTES-DESCR
     LABEL RECORD OMITTED.
 01  CARTE-DESCR                          PIC X(80).
 FD  CARTES
     LABEL RECORD OMITTED.
 01  CARTE                                PIC X(80).
*
*
 WORKING-STORAGE SECTION.
 01  CARTE-DESCR-WS.
     05  SERIE-SK                         PIC 9.
     05  CODE-SK                          PIC 9.
     05  DESCRIPTION-SK                   PIC X(30).
     05  FILLER                           PIC X(48).
*
*
 01  CARTE-WS.
     05  SERIE-ARTICLE                    PIC 9.
     05  CODE-ARTICLE                     PIC 9.
     05  DATE-VENTE.
         10 ANNEE                         PIC 9(2)  COMP.
         10  MOIS                         PIC 9(2)  COMP.
         10  JOUR                         PIC 9(2)  COMP.
     05  QUANTITE                         PIC 9(3).
     05  COUT-UNITAIRE                    PIC 9(3).
     05  FILLER                           PIC X(66).
*
*
* ------------  T A B L E A U  ------------
*
*
 01  TAB-TOTAUX-VENTES.
     05  TOTAUX-SERIE                     OCCURS 9.
         10  TOTAUX-ARTICLE               OCCURS 9.
             15  TOTAL-MENSUEL            PIC 9(6)  COMP OCCURS 12.
*
*
****  TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS .
     05  FILLER                           PIC X(9)  VALUE 'JANVIER  '.
     05  FILLER                           PIC X(9)  VALUE 'FEVRIER  '.
     05  FILLER                           PIC X(9)  VALUE 'MARS     '.
     05  FILLER                           PIC X(9)  VALUE 'AVRIL    '.
     05  FILLER                           PIC X(9)  VALUE 'MAI      '.
     05  FILLER                           PIC X(9)  VALUE 'JUIN     '.
     05  FILLER                           PIC X(9)  VALUE 'JUILLET  '.
     05  FILLER                           PIC X(9)  VALUE 'AOUT     '.
     05  FILLER                           PIC X(9)  VALUE 'SEPTEMBRE'.
     05  FILLER                           PIC X(9)  VALUE 'OCTOBRE  '.
     05  FILLER                           PIC X(9)  VALUE 'NOVEMBRE '.
     05  FILLER                           PIC X(9)  VALUE 'DECEMBRE '.
 01  TABLEAU-MOIS  REDEFINES  MOIS.
     05  NOM-MOIS                         PIC X(9)  OCCURS 12.
*
*
*        TABLEAU DES DESCRIPTIONS DES ARTICLES
*
*
 01  TABLEAU-DESCR.
     05  DESCR-ART                        OCCURS 100.
```

```
          10  CLE-ELEMENT.
              15  SERIE-ELEM          PIC 9.
              15  CODE-ELEM           PIC 9.
          10  DESCRIPTION             PIC X(30).
*
*                  = = =  I N D I C E S  = = =
*
*
 01  INDICE-ARTICLE                   PIC 9      COMP VALUE  0.
 01  INDICE-SERIE                     PIC 9      COMP VALUE  0.
 01  INDICE-MOIS                      PIC 9(2)   COMP VALUE  0.
 01  INDICE-DESCR                     PIC 9(2)   COMP VALUE  0.
 01  LIMITE-TAB-DESCR                 PIC 9(3)   COMP VALUE  100.
*
*
*                  = = =  U T I L I T A I R E S  = = =
*
 01  CLE.
     05  SERIE-CLE                    PIC 9.
     05  CODE-CLE                     PIC 9.
*
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 OUVRIR-FICHIER-DESCR.
     OPEN INPUT CARTES-DESCR.
 PREMIERE-LECTURE-DESCR.
     READ CARTE-DESCR INTO CARTE-DESCR-WS
                      AT END
                      DISPLAY '** FICHIER DESCRIPTIONS VIDE **'
                              UPON PRINTER
                      CLOSE CARTES-DESCR
                      GO TO FIN-CALCUL.
 BOUCLE-DESCRIPTIONS
     ADD 1 TO INDICE-DESCR.
     IF  INDICE-DESCR > LIMITE-TAB-DESCR
         CLOSE CARTES-DESCR
         DISPLAY '** TABLEAU DESCRIPTIONS INSUFFISANT **'
                 UPON PRINTER
         DISPLAY '** NOMBRE MAXIMUM DE DESCRIPTIONS PREVU = 100'
                 UPON PRINTER
         GO TO FIN-CALCUL.
     MOVE CARTE-DESCR-WS TO DESCR-ART (INDICE-DESCR).
     READ CARTES-DESCR INTO CARTE-DESCR-WS
                       AT END
                       CLOSE CARTES-DESCR
                       DISPLAY '** DESCRIPTIONS CHARGEES = '
                               INDICE-DESCR
                               UPON PRINTER
                       GO TO CHARGER-DONNEES.
     GO TO BOUCLE-DESCRIPTIONS.
 CHARGER-DONNEES.
     OPEN INPUT CARTES.
     MOVE LOW-VALUES TO TAB-TOTAUX-VENTES.
 PREMIERE-LECTURE-DONNEES.
     READ CARTES INTO CARTE-WS
                 AT END
                 CLOSE CARTES
                 DISPLAY '** FICHIER CARTES DONNEES VIDE **'
                         UPON PRINTER
                         FIN-CALCUL.
 SOMME.
     MOVE SERIE-ARTICLE     TO INDICE-SERIE.
     MOVE CODE-ARTICLE      TO INDICE-ARTICLE.
     MOVE MOIS              TO INDICE-MOIS.
     ADD QUANTITE TO TOTAL-MENSUEL (INDICE-SERIE, INDICE-ARTICLE
      , INDICE-MOIS).
 DONNEES SUIVANTES.
     READ CARTES INTO CARTE-WS
                 AT END
```

```
                    CLOSE CARTES
                    PERFORM AFFICHAGE-TOTAUX
                    GO TO FIN-CALCUL.
      GO TO SOMME.
*
*
*
 FIN-CALCUL.
      STOP RUN.
*
*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
      MOVE ZEROES TO INDICE-SERIE.
 AFFICHAGE SERIE.
      ADD 1 TO INDICE-SERIE.
      IF  INDICE-SERIE > 9
          GO TO AFFI-TOT-EX.
      MOVE ZEROES TO INDICE-ARTICLE.
 AFFICHAGE ARTICLE.
      ADD 1 TO INDICE-ARTICLE.
      IF  INDICE-ARTICLE > 9
          GO TO AFFICHAGE-SERIE.
      MOVE ZEROES TO INDICE-DESCR.
      MOVE INDICE-SERIE   TO SERIE-CLE.
      MOVE INDICE-ARTICLE TO CODE-CLE
 RECHERCHER-DESCRIPTION.
      ADD 1 TO INDICE-DESCR.
      IF  INDICE-DESCR > LIMITE-TAB-DESCR
          DISPLAY 'ARTICLE SERIE    : '
                  INDICE-SERIE
                  ' CODE            : '
                  INDICE-ARTICLE
                  ' ** DESCRIPTION NON TROUVEE **'
                  ' * * * TOTAUX VENTES * * *'
                  UPON PRINTER
          GO TO ECRIRE-TOTAUX.
      MOVE INDICE-SERIE TO SERIE-CLE.
      MOVE CODE-ARTICLE TO CODE-CLE.
      IF CLE NOT = CLE-ELEMENT (INDICE-DESCR)
          GO TO RECHERCHER-DESCRIPTION.
      DISPLAY 'ARTICLE : '
              DESCRIPTION (INDICE-DESCR)
              ' * * *  TOTAUX VENTES * * * '
              UPON PRINTER.
 ECRIRE-TOTAUX.
      MOVE ZEROES TO INDICE-MOIS;
 AFFICHAGE-VENTES.
      ADD 1 TO INDICE-MOIS.
      IF  INDICE-MOIS > 12
          GO TO AFFICHAGE-ARTICLE.
      DISPLAY '
              NOM-MOIS (INDICE-MOIS)
              ' = '
              TOTAL-MENSUEL (INDICE-SEZRIE, INDICE-ARTICLE
     , INDICE-MOIS)
              UPON PRINTER.
      GO TO AFFICHAGE-VENTES.
 AFFI-TOT-EX. EXIT.
*
*
*
```

Les indices des tableaux

Pour adresser un champ dans un tableau, nous avons, jusqu'à présent, utilisé des champs définis dans la WORKING-STORAGE SECTION : pensons, par exemple, à NOM-MOIS et à INDICE-MOIS (tableau des pages 1108 à 1110).Le compilateur traite ainsi les champs en question en fonction de leurs caractéristiques et non en se fondant sur la fonction d'indice que le programmeur lui réserve. Autrement dit, le programmeur est en droit d'utiliser le champs INDICE-MOIS pour n'importe quelle opération autorisée par le Cobol, sans pour autant entraîner d'erreurs de compilation. En effet, la fonction d'indice associée à ce champ ne lui a pas été déclarée.
INDICE-MOIS, INDICE-ARTICLE décrits dans la WORKING-STORAGE SECTION et utilisés par le programmeur comme tels sont appelés **champs souscrits**.
Le Cobol prévoit la possibilité d'associer, à un tableau, un ou plusieurs indices reconnus comme tels par le compilateur et gérés par lui de manière spécifique : **un indice,** en effet, **n'a pas besoin de définition dans la WORKING-STORAGE SECTION**. Sa déclaration au compilateur est réalisée pendant la prescription du tableau auquel l'indice est lié. Considérons le TABLEAU-MOIS de l'exemple précédent. Pour pouvoir lui associer un indice considéré comme tel, il est nécessaire d'adopter la notation indiquée dans le listing ci-dessous.
Le champ IND-MOIS, associé de manière univoque au tableau par la clause INDEXED BY, est utilisable seulement pour adresser les champs de TABLEAU-MOIS. Chacun d'eux peut, du reste, être adressé non seulement par IND-MOIS mais aussi par n'importe quel subscript (champ réservé).
En résumé, quand TABLEAU-MOIS est décrit comme ci-dessus, les trois instructions :

MOVE NOM-MOIS (3) TO .....
MOVE NOM-MOIS (IND-MOIS) TO .....
MOVE NOM-MOIS (INDICE-MOIS) TO .....

sont valables.
L'adressage est obtenu dans la première instruction avec une constante (3), dans la deuxième avec l'indice du tableau (IND-MOIS) et dans la troisième avc un subscript (INDICE-MOIS).
Notons, enfin, que plusieurs indices peuvent être associés à un tableau, les limitations décrites restant bloquées pour chacun d'eux.
On pourrait ainsi écrire :

```
.
01 TABLEAU-MOIS REDEFINES MOIS.
   05 NOM-MOIS  PIC X (9) OCCURS 12
                INDEXED BY
                IND-MOIS
                NOMBRE-MOIS
                POINTEUR
                .....
```

Le verbe SET pour la gestion d'un indice.
L'indice associé à un tableau par la clause INDEXED BY ne demande pas de description dans la WORKING-STORAGE parce que le compilateur réserve, à cet effet, un de ses registres internes. Ce registre assure une gestion automatique lorsque apparaissent des

DESCRIPTION D'UN TABLEAU AVEC INDICE GERE PAR LE COMPILATEUR

```
01  MOIS .
    05  FILLER                  PIC X(9)  VALUE 'JANVIER  '.
    05  FILLER                  PIC X(9)  VALUE 'FEVRIER  '.
    05  FILLER                  PIC X(9)  VALUE 'MARS     '.
    05  FILLER                  PIC X(9)  VALUE 'AVRIL    '.
    05  FILLER                  PIC X(9)  VALUE 'MAI      '.
    05  FILLER                  PIC X(9)  VALUE 'JUIN     '.
    05  FILLER                  PIC X(9)  VALUE 'JUILLET  '.
    05  FILLER                  PIC X(9)  VALUE 'AOUT     '.
    05  FILLER                  PIC X(9)  VALUE 'SEPTEMBRE'.
    05  FILLER                  PIC X(9)  VALUE 'OCTOBRE  '.
    05  FILLER                  PIC X(9)  VALUE 'NOVEMBRE '.
    05  FILLER                  PIC X(9)  VALUE 'DECEMBRE '.
01  TABLEAU-MOIS  REDEFINES  MOIS.
    05  NOM-MOIS                PIC X(9)  OCCURS 12
                                INDEXED BY IND-MOI.
```

instructions dont on parlera un peu plus loin. A cause de sa particularité, le programme ne peut intervenir sur son contenu avec les instructions Cobol employées pour les subscripts (MOVE, ADD...).
Ainsi recourt-il à l'instruction SET qui permet les opérations suivantes :

- positionner l'indice sur une valeur déterminée,
- incrémenter d'un pas la valeur de l'indice,
- décrémenter l'actuelle valeur de l'indice.

Les 3 fonctions décrites sont assumées par les formats du tableau du bas de cette page.
Notons que les valeurs attribuées par un indice dépendent des dimensions du tableau. Dans TABLEAU-MOIS, l'indice IND-MOIS n'est assumé que par les valeurs de 1 à 12, en fonction du nombre d'éléments déclarés dans le tableau. Celles qui se trouvent en dehors de cet intervalle rendent imprévisible le contenu de l'indice.
Sur la base de ce que l'on a vu jusqu'à présent, le verbe SET et les indices qui lui sont associés pourraient paraître comme inutilement redondants dans la gestion d'un tableau. En réalité, il a son utilité.

Le verbe SEARCH pour la recherche dans les tableaux

Considérons à nouveau l'exemple développé jusqu'à présent pour le traitement des tableaux. Une clé de recherche, avec les nombres de série et d'article, a été composée pour pouvoir personnaliser la description d'un article.
La clé a été successivement comparée aux deux premiers caractères des 100 éléments du tableau des descriptions. Si elle avait été égale à la clé de l'élément, la partie description aurait été prélevée ; sinon on augmenterait l'indice en vue d'une nouvelle comparaison.
Ce processus itératif s'interrompt une fois l'élément trouvé, ou bien lorsque l'indice dépasse le nombre d'éléments admis par le tableau même.
Notre exemple peut comporter jusqu'à 9 séries de 9 types d'articles, soit 81 descriptions. Dès lors, les comparaisons du 82e ou 100e sont sans objet ; elles ne feraient que rallonger bien inutilement les temps de calcul.
D'autre part, il a été imposé que le chargement des descriptions de tous les articles et séries possibles soit effectué d'office.
A défaut, le nombre de descriptions aurait été inférieur à 81 et les comparaisons excessivement plus nombreuses que nécessaires.
Parvenir à limiter la recherche aux seuls éléments effectivement chargés (sans balayer le tableau dans sa totalité) est particulièrement avantageux lorsqu'on a affaire à des tableaux volumineux, surtout s'ils ne sont que très partiellement remplis.
Pour limiter la recherche aux seuls éléments chargés, il suffit d'enregistrer leur nombre dans un champs utilitaire approprié au moment du chargement. Le tableau sera alors considéré comme complet, lors de la phase suivante de recherche (quand l'indice aura dépassé le contenu du champ utilitaire).
On peut ainsi réécrire le paragraphe BOUCLE-

FORMATS DE L'INSTRUCTION SET

SET nom-indice TO { constante | nom-donnée | nom-indice-1 }

SET nom-indice UP BY { constante | nom-donnée }

SET nom-indice DOWN BY { constante | nom-donnée }

DESCRIPTIONS de l'exemple précédent dans lequel le chargement des descriptions des articles était effectué par le fichier de cartes CARTES-DESCR. :

```
BOUCLE-DESCRIPTIONS.
   ADD 1 TO INDICE-DESCR.
   IF INDICE-DESCR    LIMITE-TAB-DESCR
      CLOSE CARTES-DESCR
      DISPLAY 'TABLEAU DESCRIPTIONS
                  INSUFFISANT'
                  UPON PRINTER
                  GO TO FIN-TRAITEMENT.
   MOVE CARTE-DESCR-WS
   TO DESCR-ART (INDICE-DESCR).
   READ CARTES-DESCR
   INTO CARTES-DESCR-WS
         AT END
         MOVE INDICE-DESCR
                  TO COMPTER-ELEMENTS
                  CLOSE CARTES-DESCR
                  GO TO CHARGER-DONNEES
   GO TO BOUCLE-DESCRIPTIONS.
CHARGER-DONNEES.
   .....
```

A la fin de la lecture du fichier CARTES-DESCR, INDICE-DESCR (qui contient encore le numéro de position du dernier élément chargé) est sauvegardé dans le champ utilitaire COMPTER-ELEMENTS supposé décrit ainsi :

```
01 COMPTER-ELEMENTS    PIC 9(3) COMP.
```

Avec ces préliminaires, le paragraphe RECHERCHER-DESCRIPTION est écrit ainsi dans la section DISPLAY-TOTAUX :

```
RECHERCHER-DESCRIPTION.
   ADD 1 TO INDICE-DESCR.
   IF INDICE-DESCR    COMPTER-ELEMENTS
      DISPLAY 'ARTICLE SERIE'
                  INDICE-SERIE
                  'CODE'
                  INDICE-ARTICLE
                  '*** DESCRIPTION
                  NON TROUVEE ***'
                  UPON PRINTER
                  GO TO ECRIRE-TOTAUX.
.....
.....
```

La description est considérée comme « non trouvée » dans le tableau soit quand tous les éléments réellements présents ont été balayés, soit quand l'indice a dépassé la valeur contenue dans COMPTER-ELEMENTS. Tout cela constitue un préliminaire à l'examen de l'instruction SEARCH et des tableaux à dimension variable.

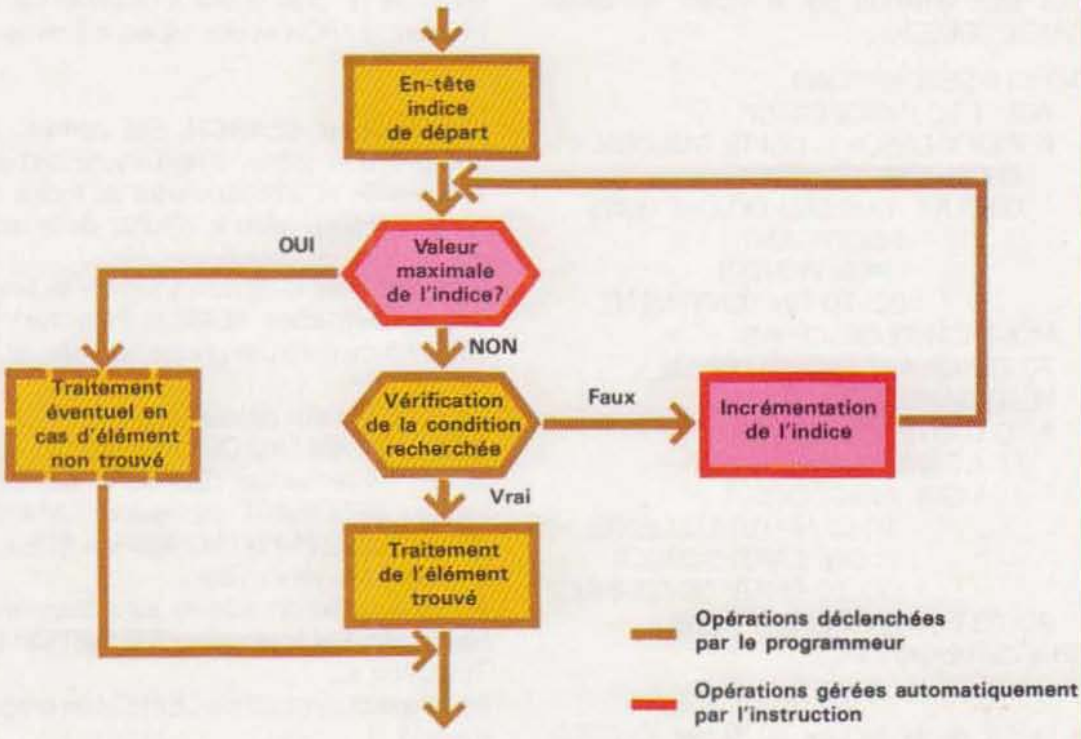
L'instruction SEARCH. Elle permet, dans une première forme, d'opérer une recherche séquentielle et d'entreprendre au moins deux types d'actions selon le résultat de la recherche (négatif ou positif).
En accord avec le caractère familier du langage Cobol, l'instruction SEARCH (chercher) peut être traduite dans une phrase normale, de type humain.
La recherche des descriptions de l'article dans le tableau TABLEAU-DESCR équivaut à l'ordre suivant : « rechercher (SEARCH) l'élément du tableau DESCR-ART et, quand (WHEN) le champ CLE-ELEMENT est égal à la CLE, écrire la description de l'article.
Arrivé à la fin du tableau sans avoir trouvé égalité (AT END), signaler DESCRIPTION NON TROUVEE ».
En adoptant l'instruction SEARCH, le programme doit :

1 / imposer la valeur de l'indice correspondant à la position de départ de la recherche. Normalement, on balaie le tableau à partir du premier élément (SET indice TO 1) ;
2 / indiquer le type d'action à entreprendre quand la condition de recherche est validée ;
3 / spécifier l'action à lancer dans le cas où l'élément recherché est absent du tableau.

Le mécanisme de fonctionnement du SEARCH séquentiel est illustré par le schéma de la page 1114.
Pour éclairer l'emploi de SEARCH, il convient d'examiner les deux versions du paragraphe RECHERCHER-DESCRIPTION (voir listings pages 1114 et 1115) avec, pour définition de TABLEAU-DESCR :

```
01 TABLEAU-DESCR.
   05 DESCR-ART    OCCURS 100
                  INDEXED BY IND-DES.
      10 CLE-ELEMENT.
         15 SERIE-ELEM          PIC 9.
         15 CODE-ELEM           PIC 9.
      10 DESCRIPTION            PIC X(30).
```

EXECUTION D'UN INSTRUCTION SEARCH SEQUENTIELLE



RECHERCHE DANS UN TABLEAU AU MOYEN D'UN SUBSCRIPT

```
      .
      .
      .
      MOVE ZEROES TO INDICE-DESCR.
  RECHERCHER-DESCRIPTION.
      ADD 1 TO INDICE-DESCR.
      IF  INDICE-DESCR GREATER THAN LIMITE-TAB-DESCR
          DISPLAY 'ARTICLE SERIE '
                  INDICE-SERIE
                  'CODE '
                  INDICE-ARTICLE
                  ' *** DESCRIPTION NON TROUVEE ***'
                  UPON PRINTER
                  GO TO ECRIRE-TOTAUX.
  *
  *
   COMPOSER-CLE.
      MOVE INDICE-SERIE    TO SERIE-CLE.
      MOVE INDICE-ARTICLE  TO CODE-CLE.
  COMPARER-CLE.
      IF  CLE NOT = CLE-ELMENT (INDICE-DESCR)
          GO TO RECHERCHER-DESCRIPTION.
      DISPLAY 'ARTICLE : '
              DESCRIPTION (INDICE-DESCR)
              '*** TOTAUX VENTES ***'
              UPON PRINTER.
  *
  *
   ECRIRE-TOTAUX.
      ..................
```

RECHERCHE DANS UN TABLEAU AU MOYEN DE SEARCH SEQUENTIEL

```
 RECHERCHER-DESCRIPTION.
     SET IND-DES TO 1.
     MOVE INDICE-SERIE    TO SERIE-CLE.
     MOVE INDICE-ARTICLE  TO CODE-CLE.
*
*
     SEARCH DESCR-ART
            AT END DISPLAY 'ARTICLE SERIE '
                           INDICE-SERIE
                           ' CODE '
                           INDICE-ARTICLE
                           ' *** DESCRIPTION NON TROUVEE ***'
                           UPON PRINTER
            WHEN CLE-ELEMENT (IND-DES) = CLE
                 DISPLAY 'ARTICLE : '
                           DESCRIPTION (IND-DES)
                           ' *** TOTAUX VENTES ***'
                           UPON PRINTER.
*
*
 ECRIRE-TOTAUX.
     ..................
```

FORMAT GENERAL DE L'INSTRUCTION SEARCH

```
SEARCH nom-élément [ VARYING { nom-indice    } ] [ AT END { phrase-impérative-1 } ]
                             { nom-donnée    }             { NEXT SENTENCE     }

     WHEN condition-1 { phrase-impérative-2 }
                      { NEXT SENTENCE     }

   [ WHEN condition-2 { phrase-impérative-3 } ]  .....
                      { NEXT SENTENCE     }
```

Dans la version recherche gérée par le programmeur (par l'intermédiaire de INDICE-DESCR), on utilise la limite maximale des dimensions du tableau et non pas celle de remplissage (COMPTER-ELEMENTS). De cette manière, et d'un point de vue fonctionnel, les deux versions du paragraphe sont complètement équivalentes.

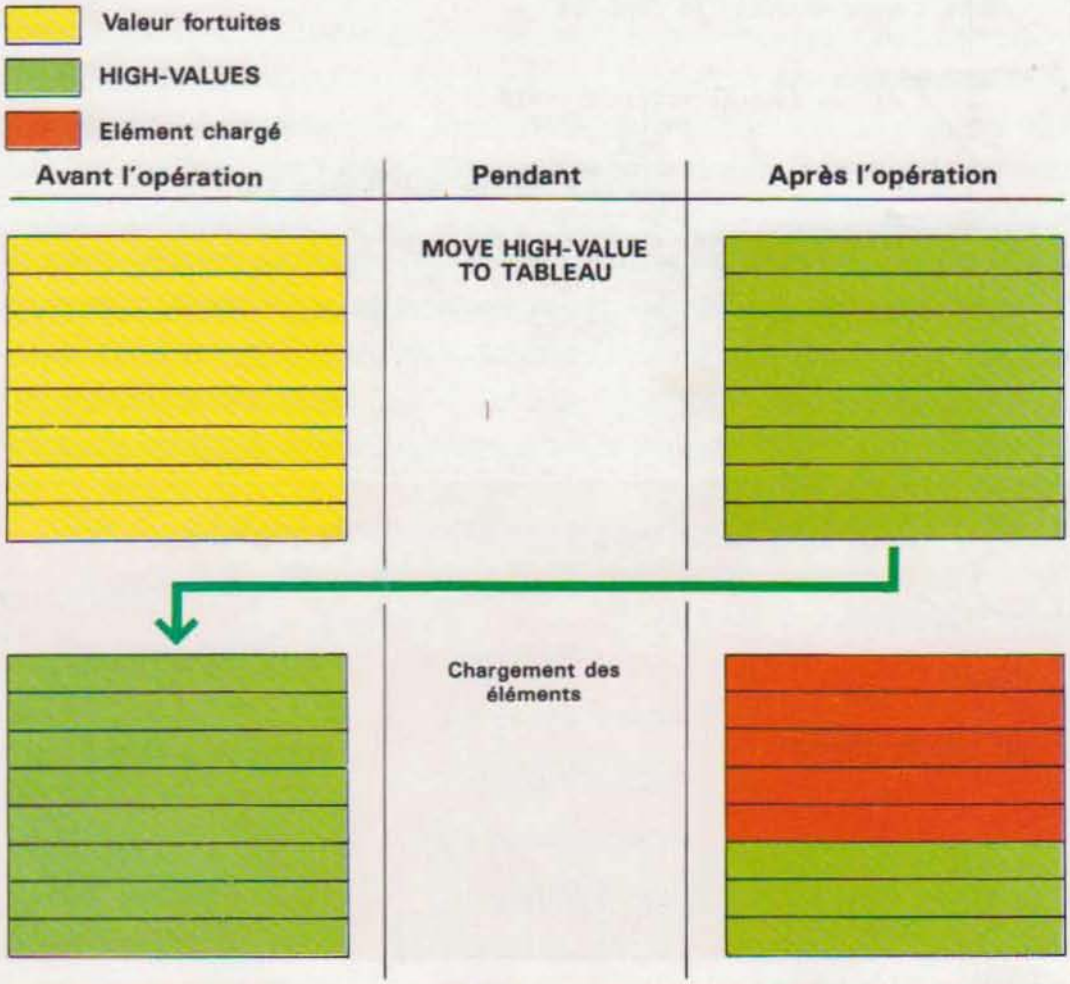
Le format général de l'instruction séquentielle SEARCH est donné en page 1115.

Nom-élément est le terme générique de l'élément dans le tableau où s'effectue la recherche. Quand la clause AT END n'est pas spécifiée et si la condition de l'élément trouvé n'a jamais été vérifiée, le contrôle passe à la première instruction après le point de fermeture du SEARCH à exécuter. La clause NEXT SENTENCE se comporte exactement comme la clause analogue de l'instruction IF, tandis que VARYING permet à chaque hausse de l'indice d'incrémenter automatiquement un autre indice ou nom-donnée.

Emploi du SEARCH séquentiel. Le problème créé par l'arrêt du balayage du tableau au-delà des limites de son remplissage réel (problème déjà affronté dans la recherche par l'intermédiaire de subscripts), peut être résolu en adoptant le SEARCH.

Dans la première méthode (page 1116), on impose HIGH-VALUES à tous les éléments du tableau ; les éléments réels sont ensuite chargés. La condition de fin de recherche sera vérifiée si le tableau est réellement terminé ou si le

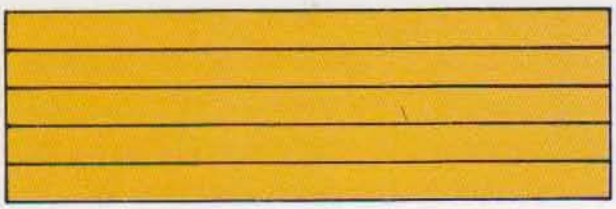
SCHEMATISATION DU TABLEAU MEMOIRE PENDANT LE CHARGEMENT




```
PROCEDURE DIVISION.
    .
    .
    .
     MOVE HIGH-VALUES  TO  TABLEAU.
     PERFORM CHARGER-TABLEAU.
        .
        .
        .
CALCULER.
        .
        .
        .
RECHERCHER-ELEMENT.
     SEARCH ELEMENT
            AT END
            PERFORM NON-TROUVE
     WHEN ELEMENT (INDICE) = HIGH-VALUES
            PERFORM NON-TROUVE
     WHEN CLE-ELEMENT (INDICE) = CLE-RECHERCHE
            PERFORM CALCULER-ELEMENT.
```

DESCRIPTION D'UN TABLEAU A DIMENSION VARIABLE DANS WORKING-STORAGE SECTION

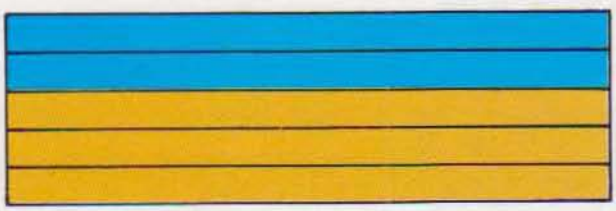
```
 01  DIMENSION                    PIC 9  COMP.
*
 01  TABLEAU.
     05  ELEMENT                  PIC X(10)
                                  OCCURS 1 TO 5
                                  DEPENDING ON DIMENSION
                                  INDEXED BY IND-TAB.
```



Représentation schématique de l'espace physique assigné au tableau

```
MOVE  2  TO  DIMENSIONE.
```

Limitation à 2 éléments de la dimension du tableau



Représentation schématique de l'espace physique assigné au tableau et de la partie (établie par DIMENSION=2) sur laquelle les instructions (par exemple SEARCH) s'activent

dernier élément examiné est égal à HIGH-VALUE.
Les pas à effectuer au cœur du programme pour limiter la recherche aux seuls éléments présents dans le tableau sont réduits, comme indiqué page 1116.
Une seconde méthode, pour minimiser les temps de recherche séquentielle, exploite les possibilités prévues par le Cobol de déclarer au compilateur le nombre d'éléments concernés par la recherche.
Cette dernière méthode exige de définir le tableau comme un tableau à dimensions variables. Le nombre de ses éléments est spécifié par la valeur d'un champ numérique attribuée par le programmeur.
La description de ce type de tableau nécessite les paramètres suivants :

- le nombre minimal d'éléments contenus par le tableau,
- le nombre maximal d'éléments contenus par le tableau,
- le nom du champ numérique dont la valeur établit les dimensions réelles du tableau.

La description d'un tableau à dimensions variables est conforme à la forme :

01 TABLEAU.
05 ELEMENT OCCURS minimum TO maximum TIMES DEPENDING ON nom-champ.

minimum C'est une constante supérieure à 0 sans signe et inférieure à « maximum »

nom-champ C'est le nom d'un champ numérique ne pouvant assumer que des valeurs comprises entre « minimum » et « maximum ».

L'expression « tableau à dimensions variables » ne doit pas être interprétée comme la possibilité d'étendre l'espace occupé en mémoire pendant le chargement.

RECHERCHE SEQUENTIELLE DANS UN TABLEAU

```
.
WORKING-STORAGE SECTION.
.
.
.
01  COMPTEUR-ELEMENTS                    PIC 9(3)  COMP VALUE 0.
01  TABLEAU-DESCR.
    05  DESCR-ART                        OCCURS 1 TO 100
                                         DEPENDING ON COMPTEUR-ELEMENTS
                                         INDEXED BY IND-DES.
        10  CLE-ELEMENT.
            15  SERIE-ELEM               PIC 9.
            15  CODE-ELEM                PIC 9.
        10  DESCRIPTION                  PIC X(30).
01  INDICE-DESCR                         PIC 9(3)  COMP VALUE 0.
01  LIMITE-TAB-DESCR                     PIC 9(3)  COMP VALUE 100.
01  CLE.
    05  SERIE-CLE                        PIC 9.
    05  CODE-CLE                         PIC 9.
*
*
*
PROCEDURE DIVISION.
.
.
CHARGER-DESCRIPTIONS.
    ADD  1  TO  INDICE-DESCR.
    IF  INDICE-DESCR > LIMITE-TAB-DESCR
        CLOSE CARTES-DESCR
        DISPLAY 'TABLEAU DESCRIPTIONS INSUFFISANT'
                UPON PRINTER
        GO TO FIN-CALCUL.
    MOVE CARTE-DESCR-WS TO DESCR-ART (INDICE-DESCR).
    READ CARTES-DESCR INTO CARTE-DESCR-WS
                      AT END
                      MOVE INDICE-DESCR TO COMPTEUR-ELEMENTS
                      CLOSE CARTES-DESCR
                      GO TO CHARGER-DONNEES.
    GO TO CHARGER-DESCRIPTIONS.
.
.
.
.
*
*
*
CHARGER-DONNEES.
.....................
.....................
.....................
.
.
.
*
*
*
RECHERCHER-DESCRIPTION.
    SET   IND-DES          TO 1.
    MOVE INDICE-SERIE      TO SERIE-CLE.
    MOVE SERIE-ARTICLE     TO CODE-CLE.
    SEARCH DESCR-ART
           AT END
                DISPLAY 'ARTICLE SERIE '
                        INDICE SERIE
                        ' CODE '
                        INDICE-ARTICLE
                        '*** DESCRIPTION NON TROUVEE ***'
                        UPON PRINTER
           WHEN
                CLE-ELEMENT (IND-DES) = CLE
```

```
                    DISPLAY 'ARTICLE : '
                            DESCRIPTION (IND-DES)
                            ' *** TOTAUX VENTES ***'
                            UPON PRINTER.
      .
      .
      .
    *
    *
    *
     ECRIRE-TOTAUX.
     ..................
```

En effet, le compilateur assigne un espace suffisant pour contenir le tableau dimensionné au maximum.
Bien noter que la variabilité des dimensions du tableau (en fonction du contenu de nom-champ) est purement logique, en ce sens que n'importe quelle instruction traitée par le tableau, à un moment déterminé, ne voit que la partie déclarée par la valeur de nom-champ. (Voir schéma ci-dessus).
Reprenons maintenant le programme de l'exemple.
A l'évidence, il reproduit la méthode la plus rapide pour rechercher séquentiellement la description d'un article. A noter que, dans le listing, les parties non concernées par l'application ont été omises.

L'instruction SEARCH ALL (recherche binaire). Nous venons de montrer que l'utilisation du SEARCH séquentiel comporte des avantages relativement modestes par rapport à une recherche avec subscripts entièrement gérée par le programmeur.
Nous allons examiner une autre forme de SEARCH ALL.
Le classement ascendant ou descendant des éléments du tableau est établi en fonction d'un champ clé déclaré.
Reprenons le tableau des descriptions des articles.
Les spécificités du programme prévoient que le fichier concernant les cartes de descriptions (CARTES-DESCR) soit rangé en ordre croissant de série et d'article :

Clé de comparaison

Série	Article	Description
11	CHEMISES	HOMME
12	CHEMISES	FEMME
13	CHEMISES	ENFANT
.		
21	PANTALONS	HOMME
22	PANTALONS	FEMME
23	PANTALONS	ENFANT
.		
91	CHAUSSURES	HOMMES
92	CHAUSSURES	FEMME
.		
99	SANDALES	ENFANT

Après chargement, TABLEAU-DESCR reflétera donc ce classement.
Dans une recherche séquentielle, (par exemple sur un article de clé 23), on procède ainsi :
— la clé de recherche CLE est d'abord comparée avec celle du 1er élément (11),
— ensuite avec celle du 2e (12)
— et ainsi de suite jusqu'au 12e élément dont la clé est 23.
Le nombre d'accès à effectuer dans un tableau pour vérifier l'existence d'un élément est en moyenne égal à N/2 avec N représentant le nombre d'éléments à examiner.
Il est clair que le nombre d'accès composés dans le tableau dépend de la position effective de l'élément recherché : une seule fois si l'élément en question est en 1ère position et n fois s'il se trouve en dernière position du tableau.
La logique de recherche du SEARCH ALL s'apparente au jeu dichotomique du « haut et bas » (voir représentation page 1120 et 1121). Le joueur n° 1 doit deviner un nombre (47, par exemple) compris entre 1 et 100 écrit par le joueur n° 2 sur une feuille de papier.
Le joueur n° 1 peut effectuer des tentatives auxquelles le joueur n° 2 ne peut répondre qu'avec les phrases « trop grand » ou « trop petit » ce qui permet d'éliminer des zones de classement par pans entiers. On en devine immédiatement l'avantage. D'ailleurs les chiffres expérimentaux sont là pour le prouver.
Dans notre exemple, alors qu'une recherche

Joueur N° 1	Commentaire	Joueur N° 2
50	Le joueur n° 1 cherche immédiatement à savoir dans quelle moitié de la série 1 à 100 se trouve le nombre, en effectuant une première tentative sur le numéro central.	Trop grand
25	La réponse est "trop grand". Le nombre recherché est donc dans les 50 premiers. On n'effectue alors plus de tentatives entre 51 et 100. En reprenant le raisonnement précédent, on propose 25, de façon à savoir dans quelle moitié de 1-50 se trouve le nombre.	Trop petit
37	Le joueur n° 1 sait maintenant que le nombre recherché est compris entre 25 et 50, c'est pourquoi il divise encore cet intervalle par deux en proposant 37.	Trop petit
44	La dernière réponse du joueur n° 2 permet de déduire que le nombre est compris entre 37 et 50. C'est pourquoi le joueur n° 1, ajoutant 7 à la moitié de l'intervalle 37-50 à 37, divise en deux ce dernier intervalle.	Trop petit
47	Le nombre doit être compris entre 44 et 50. Toujours en suivant le même critère, on ajoute 3 (50-40) à 44 pour proposer 47.	Exact

séquentielle aurait comporté 47 tentatives avant d'arriver au résultat, la **recherche binaire** a nécessité seulement 5 accès. C'est ce type de recherche appelée également dichotomique (du grec, couper en deux) que réalise le SEARCH ALL.

On comprend, du même coup, pourquoi il est indispensable d'ordonnancer le tableau en sens ascendant ou descendant, en fonction de la clé de recherche. On doit aussi préciser que le classement du tableau se trouve entièrement aux mains du programmeur, de façon à permettre au compilateur d'exclure les parties du tableau devenues inopérantes après chaque tentative.

Par conséquent, la forme décrivant un tableau à soumettre au SEARCH ALL est importante. Une telle forme prévoit, en effet, la déclaration sur le type de classement appliqué au tableau par le programmeur (voir ci-dessous).

La clause ASCENDING/DESCENDING KEY IS, clé-recherche, déclare au compilateur le type de classement appliqué aux tableaux et la partie de l'élément à utiliser comme clé de recherche.

Pour fixer les idées, considérons à nouveau l'exemple du tableau TABLEAU-DESCR contenant les descriptions des articles. Pour appliquer l'instruction SEARCH ALL en phase de recherche des descriptions, la définition du tableau et le paragraphe RECHERCHER-DESCRIPTION peuvent être détaillés comme indiqué dans le listing de la page 1122.

Dans le format de l'instruction SEARCH ALL (haut de la page 1121), nous observons que, contrairement au séquentiel, le SEARCH binaire ne nécessite pas de connaître la position initiale de l'indice (SET indice TO...), puisqu'il est directement géré par l'instruction.

De plus, avec le SEARCH ALL, la seule compa-

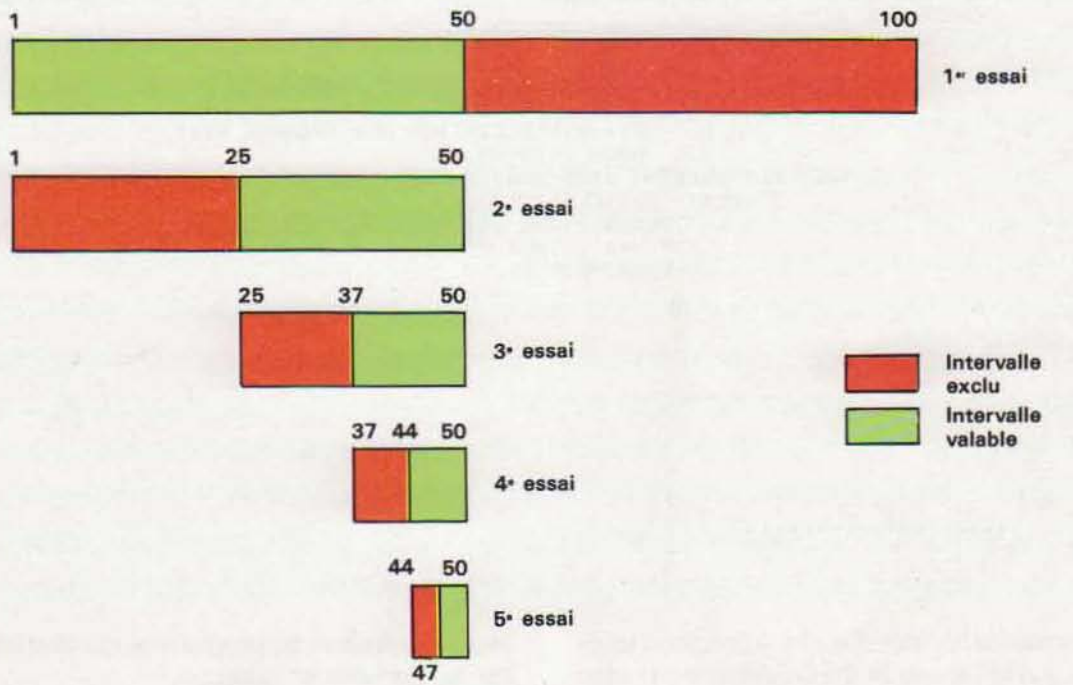
DESCRIPTION D'UN TABLEAU A SOUMETTRE A SEARCH ALL

```
01 nom-tableau
   05 nom-élément    OCCURS minimum TO maximum
                     DEPENDING ON dimensions
                    { ASCENDING  }
                    { DESCENDING } KEY IS clé-recherche
                     INDEXED BY nom-indice
```

FORMAT DE L'INSTRUCTION SEARCH ALL

```
SEARCH ALL nom-élément
           AT END
                  { phrase-impérative-1 }
                  { NEXT-SENTENCE       }
           WHEN   clé-recherche  { constante  }
                                 { nom-champ  }
                                 { expression }
                  { phrase-impérative }
                  { NEXT SENTENCE     }
```

RECHERCHE DICHOTOMIQUE DU NOMBRE 47 DANS L'INTERVALLE 1 ÷ 100



raison admise concerne l'égalité entre la clé de recherche et la clé de l'élément dans le tableau. Une telle limitation n'existe pas en SEARCH séquentiel puisque ce dernier admet également des comparaisons pour supérieur et inférieur.

Classement d'un tableau

En Cobol, la consultation d'un tableau est courante, pour des motifs très variés.
L'exposé sur la recherche dans les tableaux et sur les outils mis en œuvre (SEARCH et SEARCH ALL) a fait ressortir les avantages associés à la manipulation de données par l'intermédiaire de tableaux.
Laissons de côté les cas simples de tableaux contenant des valeurs fixes, comme les noms de mois et de jours de la semaine.

L'utilisation d'un tableau, surtout de grandes dimensions, doit être attentivement conduite, en particulier quand l'ordinateur n'est pas très puissant.

Le SEARCH ALL permet en effet d'accélérer les procédures de recherche en réduisant au

RECHERCHE DICHOTOMIQUE DANS UN TABLEAU

```
 01  TABLEAU-DESCR.
     05  DESCR-ART                      OCCURS 1 TO 100
                                        DEPENDING ON COMPTE-ELEMENTS
                                        ASCENDING KEY IS CLE-ELEMENT
                                        INDEXED BY IND-DES.
         10  CLE-ELEMENT.
             15  SERIE-ELEM            PIC 9.
             15  CODE-ELEM             PIC 9.
         10  DESCRIPTION               PIC X(30).
*
*
*
      .
      .
      .
*
*
 RECHERCHER-DESCRIPTION.
     MOVE INDICE-SERIE    TO SERIE-CLE.
     MOVE INDICE-ARTICLE  TO CODE-CLE.
     SEARCH ALL DESCR-ART
                AT END DISPLAY 'ARTICLE SERIE '
                               INDICE-SERIE
                               ' CODE '
                               INDICE-ARTICLE
                               ' *** DESCRIPTION NON TROUVEE ***'
                               UPON PRINTER
                WHEN CLE-ELEMENT (IND-DES) = CLE
                     DISPLAY 'ARTICLE : '
                             DESCRIPTION (IND-DES)
                             ' *** TOTAUX VENTES ***'
                             UPON PRINTER.
*
*
 ECRIRE-TOTAUX.
     ..................
```

minimum le nombre d'accès, à condition toutefois que le tableau ait été préalablement ordonné, selon une clé déterminée.
Si les données à charger dans le tableau ne sont pas déjà ordonnées selon un certain ordre croissant ou décroissant, le problème de savoir si l'on adopte une recherche de type séquentiel ou binaire se pose.
Avant de passer à l'utilisation de ce dernier mode, il faut en effet prévoir une phase de classement des données, qui peut être conduite de trois manières différentes.

— On peut ordonner le fichier de données selon une clé, grâce à des programmes de tri. L'opération de classement doit être effectuée avant l'exécution du programme qui doit utiliser les données en question.
Pour fixer les idées, prenons l'exemple du fichier de cartes CARTES-DESCR, contenant les descriptions des articles.
Si l'on adopte une sélection externe, l'organisation des données sera celle illustrée dans la page suivante.
— Alternativement, on peut ordonner le fichier données selon la clé désirée, en recourant à la sélection (tri) interne des données (si le compilateur ne l'a pas).
Ce mode de sélection sera plus largement décrit dans le chapitre suivant (voir organigramme page 1124).
— Comme dernière solution, on peut ordonner

directement le tableau en mémoire, tout de suite après son chargement par l'intermédiaire d'un algorithme quelconque géré par le programmeur (voir organigramme page 1125).

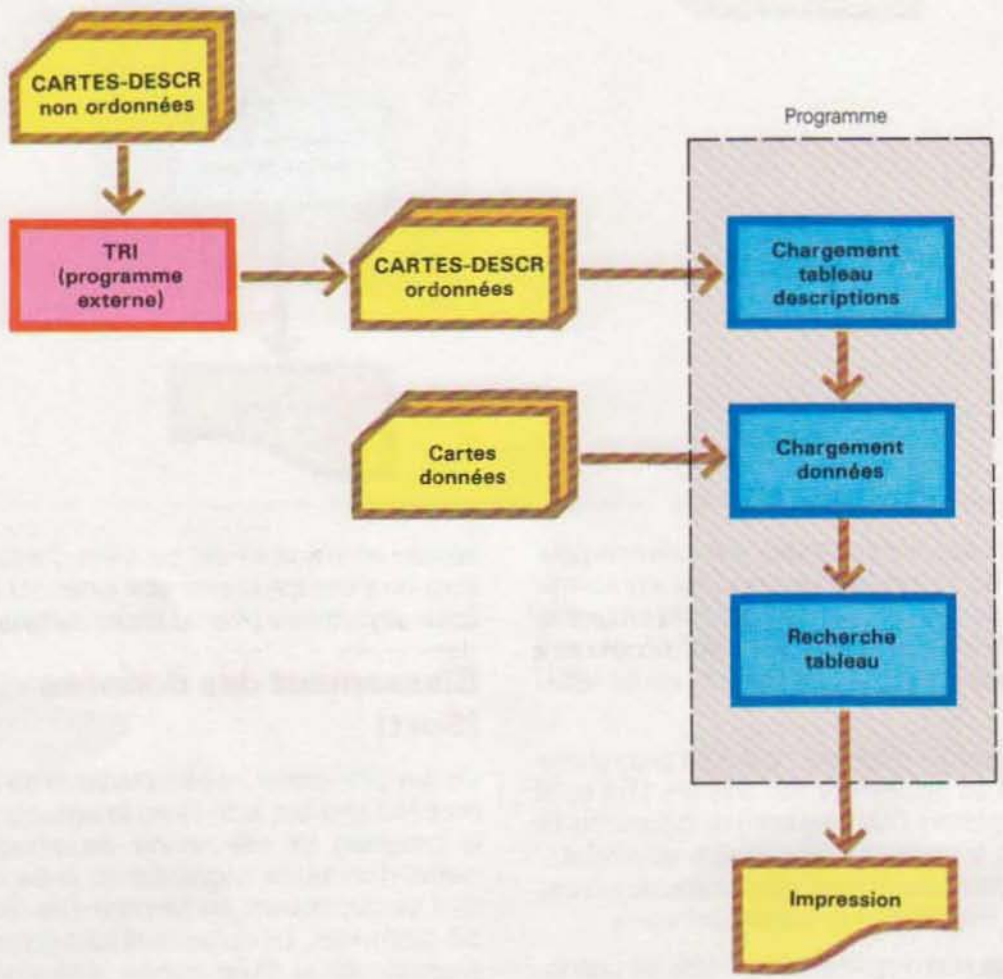
Les trois méthodes décrites présentent des avantages et des inconvénients pour ce qui concerne le temps utilisé et les ressources de calcul demandées (mémoire centrale et mémoire de masse). Indépendamment de la méthode choisie, il est normal de se demander : « est-il juste de pénaliser en temps le programme employé pour sélectionner les données dans l'intention d'utiliser le SEARCH ALL ; ne convient-il pas plutôt d'adopter une recherche plus lente sans être obligé d'ordonner les données d'entrée ? »

Il n'existe pas de réponse univoque à cette question puisque le choix dépend du type de traitement.

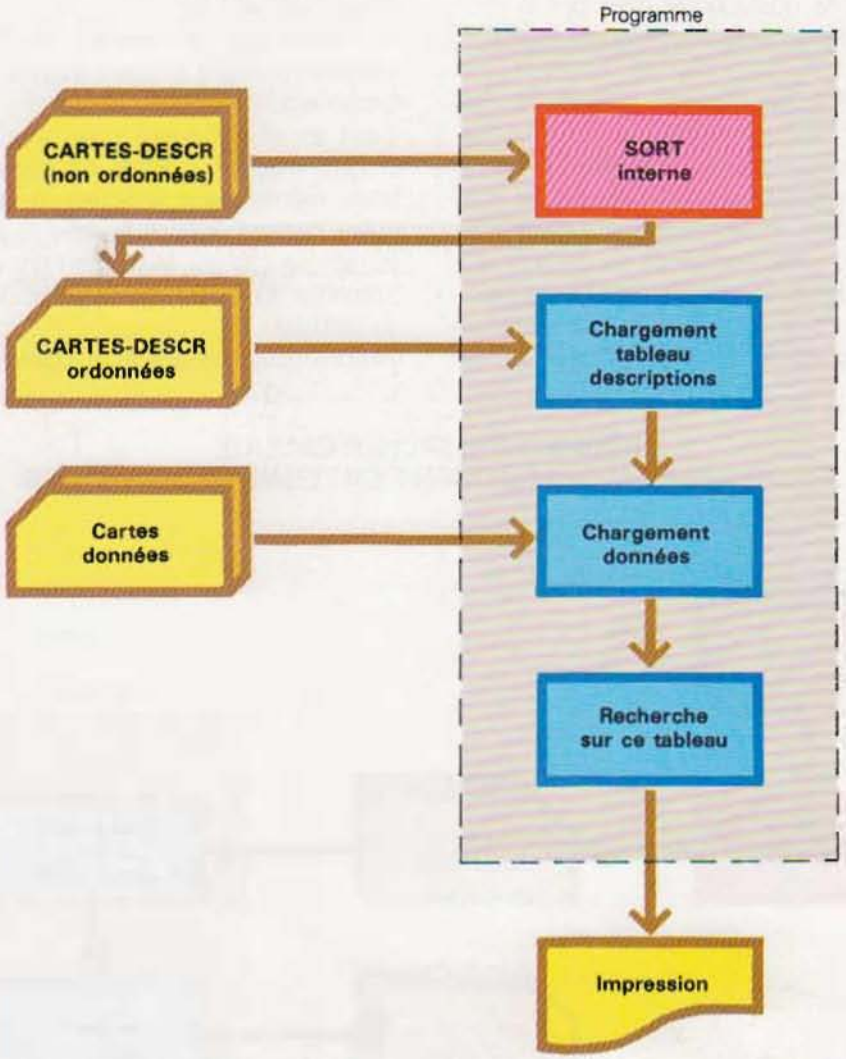
Il est en effet évident que, si au cours d'un programme, les besoins de consulter un tableau, même grand, sont peu nombreux, il vaut mieux recourir à la recherche séquentielle, en évitant de charger le classement des enregistrements du fichier d'entrée ou des éléments du tableau.

Inversement, si le traitement demande d'accé-

RECHERCHE DICHOTOMIQUE
PREALABLE AU CLASSEMENT EXTERNE DES DONNEES



RECHERCHE DICHOTOMIQUE PREALABLE AU CLASSEMENT INTERNE DES DONNEES



der très souvent au tableau (par exemple pour contrôles la validité de tous les enregistrements lus par un fichier d'entrée), la recherche séquentielle s'avère extrêmement pénible et il est donc opportun de recourir au SEARCH ALL.

Si le système n'est pas doté d'un programme destiné au classement des données (Tri) ou si le compilateur Cobol est privé de la fonction Tri interne, la seule solution qui reste, au programmeur, pour sélectionner les données du tableau est de réaliser seul le classement voulu.

Dans les micro-ordinateurs et dans les ordinateurs personnels, le champ du classement d'un tableau en mémoire est peut-être l'argument dont on s'occupe le plus et il existe de nombreux algorithmes plus ou moins rapides.

Classement des données (Sort)

Un des instruments les plus puissants dont dispose le Cobol est le tri (Sort) interne qui offre la possibilité de sélectionner les enregistrements d'un fichier séquentiel en ordre croissant ou décroissant, en fonction des champs clé déterminés. Le classement peut porter sur plusieurs clés et d'une manière différente.
Soit un fichier cartes décrit ainsi :

```
01 SK.
   05 LOCALITE        PIC X(20).
   05 DEPARTEMENT     PIC X(20).
   05 REGION          PIC X(20).
   05 NB-HABIT        PIC 9(8).
   05 POSITION        PIC X(4).
   05 FILLER          PIC X(8).
```

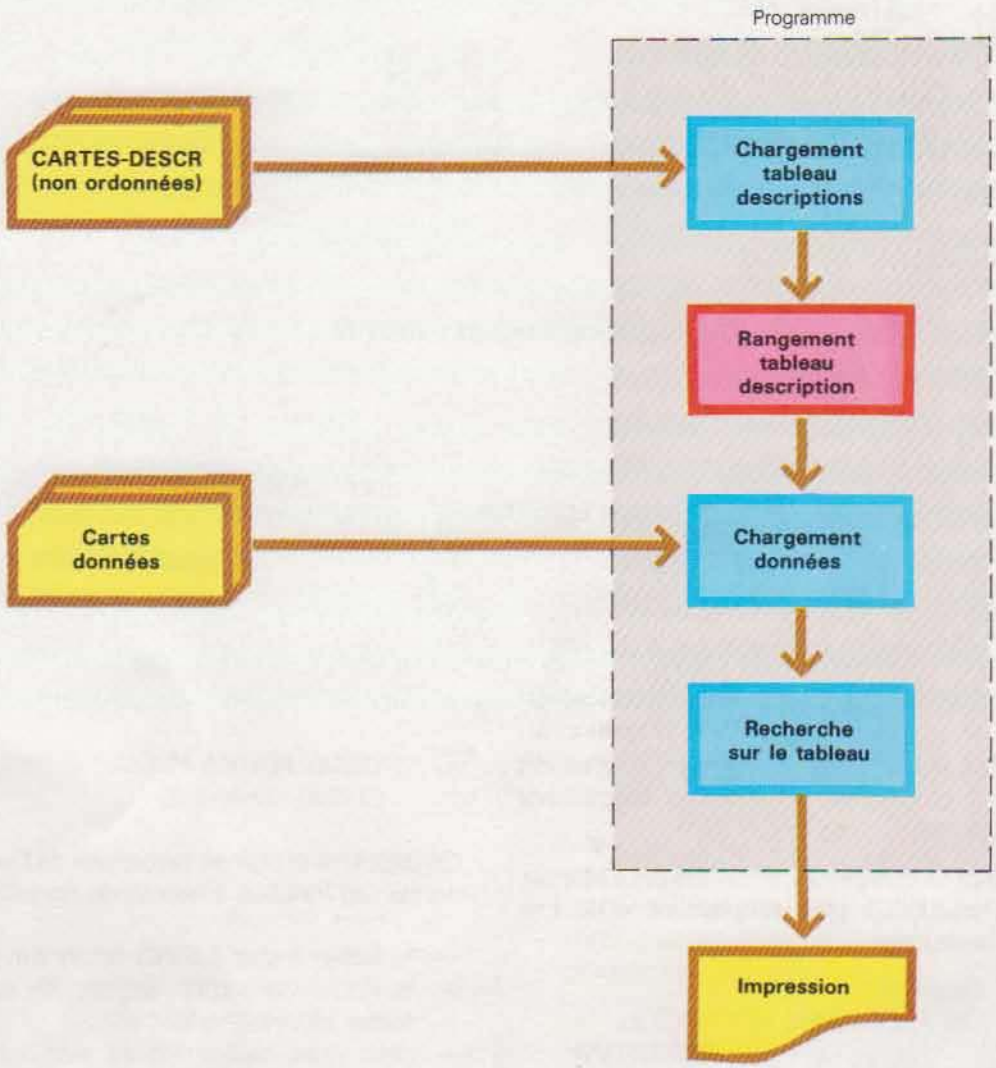
Dans chaque cadre sont reportés :

1 / le nom d'une localité française,
2 / le département d'appartenance,
3 / la région d'appartenance,
4 / le nombre d'habitants de la localité,
5 / la position (Nord ou Sud) dans laquelle on se situe par rapport à Paris.

Nous voulons sélectionner les cartes pour que :

- les régions soient ordonnées par nom ascendant,
- tous les départements de la même région soient ordonnés en série ascendante,

RECHERCHE DICHOTOMIQUE AVEC CLASSEMENT GERE PAR LE PROGRAMMEUR



EXEMPLE DE CLASSEMENT ET DE SELECTION DES DONNEES

```
IDENTIFICATION DIVISION.
.
.
.
ENVIRONMENT DIVISION.
.
.
FILE-CONTROL.
    SELECT FICHIER-DE-TRI  ASSIGN TO DISC.
    SELECT CARTES          ASSIGN TO CARD-READER.
    SELECT CARTES-ORD      ASSIGN TO DISC.
*
*
DATA DIVISION.
FILE SECTION.
FD   CARTES
         LABEL RECORD OMITTED.
01  SK                              PIC X(80).


SD  FICHIER-DE-TRI.
01  SK-TRI.
    05  LOCALITE                    PIC X(20).
    05  CHEF-LIEU                   PIC X(20).
    05  REGION                      PIC X(20).
    05  NOMBRE-HABIT                PIC 9(8).
    05  POSITION                    PIC X(4).
    05  FILLER                      PIC X(8).
*
*
*
PROCEDURE DIVISION.
SELECTIONNE.
    SORT FICHIER-DE-TRI
                    ON ASCENDING KEY REGION




                                          CHEF-LIEU
                    ON DESCENDING KEY NOMBRE-HABIT
         USING  CARTES
         GIVING CARTES-ORD.
*
*
FIN.
    STOP RUN.
```

- toutes les localités du même département soient ordonnées par nombre décroissant d'habitants.

La sélection complexe demandée peut être obtenue beaucoup plus simplement avec une seule instruction :

SORT fichier-tri
    ON ASCENDING KEY REGION
                DEPARTEM.
    ON DESCENDING KEY NB-HABIT
    USING fichier-à-trier
    GIVING fichier-trié.

On peut tirer certaines indications de l'exemple donné : la fonction a besoin de connaître :

— le fichier à trier (USING fichier-à-trier),
— le fichier où seront rangées les données triées (GIVING fichier-trié),
— dans quel espace travail sera effectuée l'opération de tri (fichier-tri),

— les champs de l'enregistrement et l'ordre de sélection (ON ASCENDING KEY...) adoptés.

Analysons maintenant chacune de ces entités.

Fichier-tri. Cet espace peut être affecté tant à DISC qu'à TAPE en utilisant la clause normale SELECT dans l'ENVIRONMENT DIVISION.
Les particularités dans la définition du fichier de tri concernent essentiellement la DATA DIVISION où :

1 / l'indicateur de niveau SD (Sort Definition) **doit** être utilisé au lieu de FD,
2 / la clause LABEL RECORD **ne doit pas** être utilisée.

Ces deux exceptions précisées, toutes les autres clauses examinées dans la description des fichiers restent valables.
La description, au niveau SD, du fichier de tri ne difinit par un fichier physique. Mais il permet au compilateur de personnaliser les champs de l'enregistrement en fonction duquel s'opère la sélection. De ce fait, on ne décrira que les champs de l'enregistrement qui participent à la sélection. On obtient ainsi des traitements simplifiés.

Fichier à trier. C'est un fichier séquentiel quelconque utilisé en entrée. Il n'exige pas d'ouverture (OPEN) ou de fermeture (CLOSE) de la part du programmeur si la clause USING est utilisée.

Fichier trié. Egalement séquentiel, il est soumis aux opérations normales de définition de fichiers : il est entièrement géré en ouverture et en fermeture par le compilateur avec la clause GIVING.

Ordre de sélection. Chaque préposition ON rencontrée dans l'instruction SORT (tri) spécifie le type de classement à appliquer aux champs-clés classés.
La séquence, selon laquelle ces clés sont disposées à l'intérieur de l'instruction, détermine leur niveau hiérarchique.

Dans l'exemple rapporté, la clé REGION s'avère d'une importance supérieure à celle de DEPARTEMENT, elle-même prioritaire par rapport à NB-HABITANTS.
En guise de conclusion à cette première description du classement des données, nous donnons le programme détaillé qui sélectionne les cartes décrites selon les demandes spécifiques (voir listing ci-contre).

L'INPUT PROCEDURE et le verbe RELEASE

La fonction TRI nécessite généralement un temps de traitement très long. Aussi est-il de bonne règle de ne sélectionner que les enregistrements indispensables au calcul.
Pour permettre au programmeur de choisir ces enregistrements, le Cobol prévoit une procédure entrée (INPUT PROCEDURE) qui représente une alternative à la clause USING.
Dans cette SECTION, l'enregistrement lu par le fichier d'entrée est soumis à des traitements et à des contrôles avant d'être écarté au moment du tri. L'opération TRI consiste à écrire l'enregistrement dans le fichier-tri par l'intermédiaire de l'instruction RELEASE (tableau ci-dessous), analogue au WRITE.
Pour illustrer la fonction introduite, prenons l'exemple d'une sélection selon les critères connus des seules cartes correspondant aux localités de plus de 20 000 habitants.
Nous obtenons le programme de la page suivante, qui ne tient pas compte des parties communes à l'exemple précédent.

L'OUTPUT PROCEDURE et le verbe RETURN

En dehors de la clause GIVING, le Cobol permet de déclarer, au compilateur, le nom d'une SECTION (OUTPUT PROCEDURE) dans laquelle le programmeur soumet les enregistrements sélectionnés (directement lus par le fichier-tri) à n'importe quel type de traitement.
Comme pour la procédure d'entrée, le fichier-tri n'est accessible qu'avec l'instruction RETURN (voir format général page 1128), et non avec l'instruction READ.

FORMAT DE L'INSTRUCTION RELEASE

RELEASE enregistrement-tri [FROM nom-données]

EXEMPLE D'UTILISATION DE L'INSTRUCTION RELEASE

```
      .
      .
      WORKING-STORAGE SECTION.
      01  SK-WS.
          05  LOCALITE-WS                PIC X(20).
          05  CHEF-LIEU-WS               PIC X(20).
          05  REGION-WS                  PIC X(20).
          05  NOMBRE-HABIT-WS            PIC 9(8).
          05  POSITION-WS                PIC X(4).
          05  FILLER                     PIC X(8).
     **
     **
     **
       PROCEDURE DIVISION.
       SELECTIONNE SECTION.
       DEBUT.
            OPEN INPUT CARTES.
            SORT FICHIER-DE-TRI
                              ON ASCENDING KEY REGION
                                               CHEF-LIEU
                              ON DESCENDING KEY NOMBRE-HABIT
                 INPUT PROCEDURE ARRETEE
                 GIVING CARTES-ORD.
       FIN.
            STOP RUN.
       *
       *
       *
        FIN SECTION.
        LIRE-CARTES.
            READ CARTES INTO SK-WS
                        AT END
                        CLOSE CARTES
                        GO TO FIN-EX.
            IF  NOMBRE-HABIT-WS GREATER THAN 20000
                RELEASE SK-TRI FROM SK-WS.




            GO TO LIRE-CARTES.
       FIN-EX.
            EXIT.
```

FORMAT GENERAL DE L'INSTRUCTION RETURN

RETURN fichier-tri [INTO nom-données]
AT END phrase-impérative.

Reprenons le même exemple et supposons que l'on doive lister sur deux fichiers distincts toujours les villes situées respectivement au nord et au sud de Paris.
Le programme convenable se trouve en page 1130 et 1131.

Gestion des fichiers annexes

Tous les fichiers abordés jusqu'ici présentaient une structure de type séquentiel. Passons maintenant aux fichiers indexés. A la différence des premiers (où il est nécessaire de lire linéairement tous les enregistrements précédant l'information recherchée) il suffit, dans les fichiers indexés, d'indiquer la valeur recherchée dans le champ clé de l'enregistrement.
On obtient immédiatement l'enregistrement lui-même.
L'organisation indexée d'un fichier permet donc d'accéder directement à un enregistrement grâce à une clé symbolique.
Supposons par exemple que l'on doive rechercher des informations relatives à un salarié dont on connaît le code (matricule). Toutes les données sur le personnel se trouvent dans un fichier ordonné par matricule. Dans un fichier séquentiel, on aurait été obligé d'employer la procédure exposée dans l'organigramme de la page 1131 ; on lirait donc **tous** les enregistrements du fichier jusqu'au résultat (matricule) recherché. Avec cette méthode, il faudrait, en moyenne, un nombre d'accès égal à la moitié du volume d'enregistrement dans le fichier.
En revanche, dans un fichier indexé, dans lequel le matricule a été défini comme clé d'accès, il suffit d'introduire le code (matricule) dans la clé pour déclencher la lecture. L'enregistrement est immédiatement disponible, sans passer par une boucle de lecture (voir organigramme page 1132).
Cette dernière procédure est opérationnelle grâce à une organisation appropriée du fichier, schématiquement représentée page 1133.
Outre les données, ce fichier contient un index **(tableau des adresses)** où sont répertoriées toutes les clés, avec l'adresse de l'enregistrement correspondant.

Opération de test et d'entretien d'une imprimante connectée à un gros ordinateur.

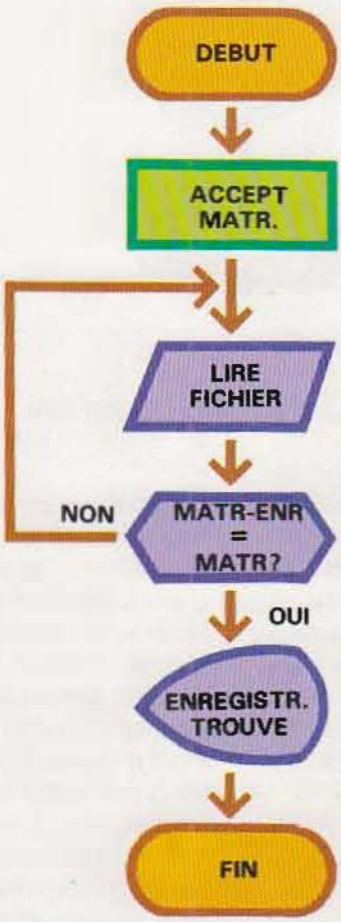


EXEMPLE D'UTILISATION DE LA COMMANDE SORT

```
  IDENTIFICATION DIVISION.
  .
  .
  ENVIRONMENT DIVISION.
  .
  .
  FILE-CONTROL.
    SELECT FIC-TRI       ASSIGN TO DISC.
    SELECT CARTE-ENTREE ASSIGN TO CARD-READER.
      SELECT CARTE-NORD   ASSIGN TO DISC.
      SELECT CARTE-SUD    ASSIGN TO DISC.
*
  DATA DIVISION.
    FILE SECTION.
*   ---- fichier d'entrée
    FD CARTE-ENTREE
         LABEL RECORD OMITTED.
  01 C-ENTREE              PIC X(80).
*  ---- premier fichier de sortie (villes du nord > 20000 hab.)
  FD CARTE-NORD
       LABEL RECORD STANDARD.
  01 C-NORD                PIC X(80).
*  ---- second fichier de sortie (villes du sud > 20000 hab.)
  FD CARTE-SUD
       LABEL RECORD STANDARD.
  01 C-SUD                 PIC X(80).
*  ---- fichier servant au tri
  SD FIC-TRI.
  01 C-TRI.
      05 LOCALITE          PIC X(20).
      05 CHEF-LIEU         PIC X(20).
      05 REGION            PIC X(20).
      05 NB-HABIT          PIC 9(8).
      05 POSIT-GEO-WS      PIC X(4).
      05 FILLER            PIC X(8).
*
*
   WORKING-STORAGE SECTION.
   01 VILLE-WS.
      05 LOCALITE-WS       PIC X(20).
      05 CHEF-LIEU-WS      PIC X(20).
      05 REGION-WS         PIC X(20).
      05 NB-HABIT-WS       PIC 9(8).
      05 POSIT-GEO         PIC X(4).
      05 FILLER            PIC X(8).
*
*
*
  PROCEDURE DIVISION.
  PRINCIPALE SECTION.
    DEBUT-PRINCIPALE.
      OPEN INPUT CARTE-ENTREE.
      SORT FIC-TRI
          ON ASCENDING KEY  REGION
                            CHEF-LIEU
          ON DESCENDING KEY NB-HABIT
      INPUT PROCEDURE  ENTREE-TRI
      OUTPUT PROCEDURE SORTIE-TRI.
    FIN-PRINCIPALE.
      STOP RUN.
*
*           sections d'entrée / sortie du tri
*
   ENTREE-TRI SECTION.
     LECTURE-CARTE.
       READ CARTE-ENTREE INTO VILLE-WS
            AT END
               CLOSE CARTE-ENTREE
               GO TO LECTURE-FIN.
```

```
       IF NB-HABIT-WS GREATER THAN 20000
           RELEASE C-TRI FROM VILLE-WS.
       GO TO LECTURE-CARTE.
     LECTURE-FIN.
       EXIT.
*
   SORTIE-TRI SECTION.
     OUV-FIC-SORTIE.
       OPEN OUTPUT CARTE-NORD
                   CARTE-SUD.
     SEPARE-GEO.
       RETURN FIC-TRI INTO VILLE-WS
           AT END
              CLOSE CARTE-NORD
              CLOSE CARTE-SUD
              GO TO SEPARE-FIN.
       IF POSIT-GEO-WS = 'NORD' WRITE CARTE-NORD FROM VILLE-WS.
       IF POSIT-GEO-WS = 'SUD'  WRITE CARTE-SUD  FROM VILLE-WS.
       GO TO SEPARE-GEO.
     SEPARE-FIN.
       EXIT.
```

EXEMPLE DE RECHERCHE DANS UN FICHIER SEQUENTIEL



EXEMPLE DE RECHERCHE DANS UN FICHIER INDEXE



Lorsqu'on demande une lecture d'enregistrement par l'intermédiaire d'une clé, le programme lance une ou plusieurs lectures dans l'index jusqu'à ce qu'il ait trouvé la clé recherchée.
Dans un deuxième temps, en se servant de l'adresse définie dans l'index, il accède directement à l'enregistrement. Donc, pour lire un enregistrement, il faut non pas une, mais deux lectures, quelle que soit la position de cet enregistrement.
C'est une procédure particulièrement avantageuse quand il s'agit d'une recherche aléatoire dans un fichier d'assez grande taille.
Il existe 3 méthodes d'accès à un fichier indexé :

1/ aléatoire (Random)
2/ séquentielle (Sequential)
3/ mixte ou dynamique (Dynamic)

En mode séquentiel, le fichier obéit aux règles habituelles du traitement des fichiers séquentiels : les données sont traitées en séquence physique du premier au dernier.
Nous avons déjà indiqué les caractéristiques de l'accès aléatoire (Random). Quant au mode d'accès dynamique (Dynamic) il s'agit d'une synthèse des deux autres : il permet un traitement mixte de fichier. On peut, en effet, adopter la procédure qui semble la plus appropriée aux besoins du moment : par exemple lire de manière séquentielle une série de données, passer à l'aléatoire pour une autre série et revenir au séquentiel, etc.

REPRESENTATION SIMPLIFIEE DE LA STRUCTURE D'UN FICHIER INDEXE

Index

Clé (matricule	Adresse de l'enregistrement
00015	000001
00038	000002
00088	000003
00133	000004
00134	000005

Enregistrement des données

Matricule	Nom et prénom
00015	DURAND PAUL
00038	DUPONT JEAN
00088	MERCIER JACQUES
00133	PERRIN NATHALIE
00134	LOISEAU CHARLES

C'est pourquoi on parle parfois de fichiers **séquentiel-indexé** ou **IS** (pour Indexed-Sequential).

Voyons maintenant les clauses servant à définir un fichier IS dans les divisions environnement et données, ainsi que les instructions de la division algorithmes (Procédure Division) nécessaires à sa gestion.

Description des fichiers IS dans ENVIRONNEMENT DIVISION

Si un programme utilise un fichier IS au niveau Select, il faut nécessairement assigner le fichier à Disc et en préciser la clause :

ORGANIZATION IS INDEXED

Elle doit être suivie d'une deuxième clause qui définit le mode d'accès :

ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }

Il faut enfin écrire la clause :

RECORD KEY IS nom-donnée

où « nom-donnée » désigne un champ de l'enregistrement défini dans le Fichier-descriptions (qui contient la clé symbolique de l'enregistrement). Ce champ, élémentaire ou composé, ne dépasse généralement pas 60 caractères. Le champ-clé ne peut contenir une valeur équivalente à LOW-VALUES et doit être déclaré avant toute opération d'accès aléatoire.

Description des fichiers IS dans DATA DIVISION

Dans le Fichier-descriptions, on emploie la clause

LABEL RECORD STANDARD

d'après les règles précédemment établies pour les fichiers séquentiels assignés à DISC.
Toujours au niveau FD, dans la description de l'enregistrement, il faut définir le champ contenant la clé de l'enregistrement au moyen du nom indiqué dans la clause RECORD KEY (au niveau SELECT).

Les instructions dans PROCEDURE DIVISION

Un fichier indexé est ouvert selon l'une des trois méthodes étudiées pour les fichiers à accès séquentiel, à l'exception de la modalité EXTEND.
Le format de l'instruction d'ouverture est donc :

FORMAT DE L'INSTRUCTION OPEN

OPEN { INPUT / OUTPUT / I/O } nom-fichier

Le verbe READ. L'instruction de lecture n'est possible qu'à deux conditions : le fichier a préalablement été ouvert en entrée (INPUT) ou en entrée-sortie (I/O) ; le fichier comporte deux formats en fonction du mode d'accès défini au

1er FORMAT DE L'INSTRUCTION READ APPLIQUEE AUX FICHIERS IS

READ nom-fichier [NEXT] RECORD [INTO nom-donnée-1]

AT END phrase-impérative.

2e FORMAT DE L'INSTRUCTION READ APPLIQUEE AUX FICHIERS IS

READ nom-fichier [INTO nom-donnée-2]

INVALID KEY phrase-impérative.

niveau SELECT (voir page 1134).
Plus précisément, les règles à respecter sont :

1/ Pour tout fichier à accès séquentiel (ACCESS MODE IS SEQUENTIAL), employer le 1er format sans la clause NEXT :

READ nom-fichier RECORD
[INTO nom-donnée-1]
AT END phrase-impérative.

2/ Pour des lectures de type séquentiel sur des fichiers à accès mixte (ACCESS MODE IS DYNAMIC), c'est le 1er format, avec la clause NEXT, que l'on doit employer :

READ nom-fichier NEXT RECORD
[INTO nom-donnée-1]
AT END phrase-impérative.

3/ Pour toutes les opérations de lecture aléatoire, employer le deuxième format lorsqu'au niveau SELECT on a déclaré ACCESS MODE IS RANDOM ou ACCESS MODE IS DYNAMIC.

4/ Avant l'opération de lecture, on assigne, au champ de RECORD KEY, une valeur correspondante à celle de l'enregistrement recherché. S'il n'existe pas, on exécutera les instructions de la « phrase-impérative » qui suit la clause INVALID KEY.

Le verbe WRITE. Cette instruction n'est opérationnelle que sur des fichiers préalablement ouverts en sortie ou en E/S et selon le format ci-dessous.
Si le fichier est ouvert en entrée (phase de création), les enregistrements doivent être écrits en ordre croissant de clé. Autrement dit, on accède au fichier de manière séquentielle, quel que soit le mode d'accès choisi.
Avant toute écriture, il faut organiser le champ du domaine contenant la clé d'enregistrement.
La phrase impérative, spécifiée dans la clause INVALID KEY, est exécutée dans les cas suivants :

- les valeurs des clés ne sont pas en ordre croissant d'une clause WRITE à la suivante
- la valeur de la clé n'est pas unique dans le cadre du fichier
- on a rempli tout l'espace alloué sur le disque.

Instructions spécifiques aux fichiers IS
Trois verbes n'ont pas d'équivalents dans les

FORMAT DE L'INSTRUCTION WRITE APPLIQUEE AUX FICHIERS IS

WRITE nom-enregistrement [FROM nom-donnée]

INVALID KEY phrase-impérative.

fichiers séquentiels :

REWRITE pour mettre à jour des données existantes

DELETE pour écraser des données déjà présentes sur le fichier

START pour pointer un enregistrement donné.

Le verbe REWRITE. L'opération de réécriture n'est autorisée que sur des fichiers ouverts en ES, selon le premier format indiqué en bas de page.

Son emploi est subordonné au respect de quelques règles fondamentales :

— L'instruction n'est exécutable que si la modification d'une donnée n'entraîne pas celle de la clé d'accès.
Sinon, la phrase impérative, spécifiée dans la clause INVALID KEY, serait exécutée.

— En fichiers séquentiels, REWRITE n'est lancé qu'après un READ fructueux.

— En fichiers à accès aléatoire ou dynamique, il n'est pas nécessaire d'exécuter un READ avant un REWRITE.

— La phrase impérative de la clause INVALID KEY s'exécute seulement si la valeur de la clé, attribuée dans RECORD KEY (avant REWRITE), est différente de celle des autres enregistrements du fichier.
Il n'est donc pas possible de réécrire un enregistrement non encore créé.

Le verbe DELETE. Comme pour REWRITE, l'instruction d'effacement (voir le second format ci-dessous) n'opère que sur des enregistrements déjà existants.

Elle suit les règles suivantes :

— Le fichier doit être ouvert en ES.

— En fichiers séquentiels, l'instruction READ doit précéder l'effacement.

— En fichiers à accès aléatoire ou dynamique, il faut réorganiser RECORD KEY avant l'opération d'effacement (...).

— Si la clé définie ne correspond à aucun enregistrement, c'est la phrase impérative de la clause INVALID KEY qui s'exécute.



Phase de test de terminaux dans une unité de production.

FORMAT DE L'INSTRUCTION REWRITE

REWRITE nom-enregistrement [FROM nom-donnée]
INVALID KEY phrase impérative.

FORMAT DE L'INSTRUCTION DELETE

DELETE nom-fichier RECORD
INVALIDE KEY phrase-impérative.

Le verbe START. Il initialise la lecture séquentielle en pointant sur un enregistrement du fichier.
Supposons que l'on ait à imprimer tous les enregistrements du fichier du personnel défini plus haut, de telle sorte que le code matricule, donc la clé d'enregistrement (RECORD KEY), soit supérieur à une valeur donnée.
Grâce à l'instruction START, on peut se positionner directement sur le premier matricule valable.
On peut aussi sortir tous les enregistrements suivants pour une lecture séquentielle du fichier.
La comparaison entre deux organigrammes, l'un avec un test et l'autre avec START, montre que cette dernière instruction ne rend pas effectivement disponible l'enregistrement mais qu'elle réalise un pointage à la position voulue.

L'instruction START (voir format page 1137) doit être suivi d'un :

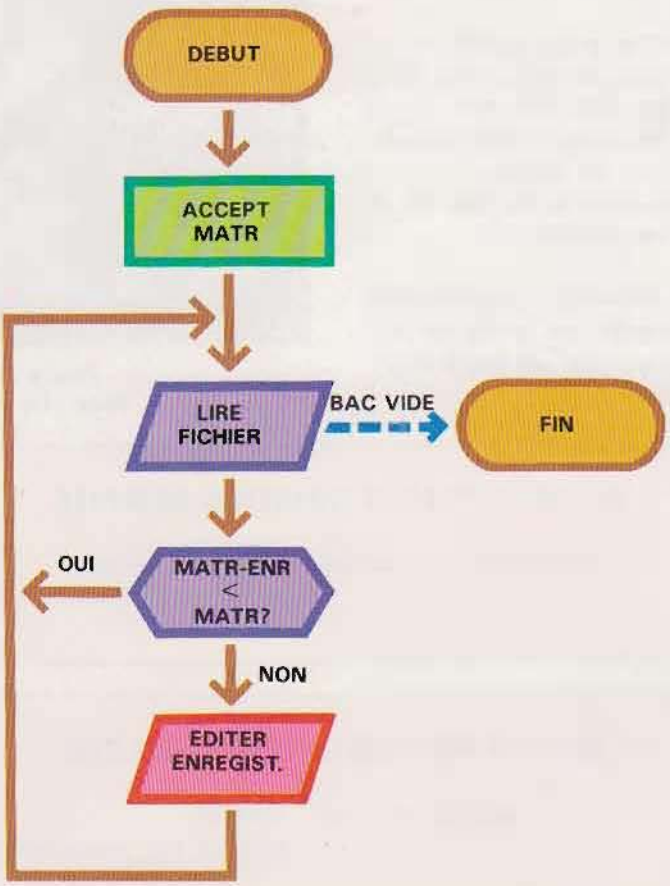
READ nom-fichier AT END...

ou d'un :

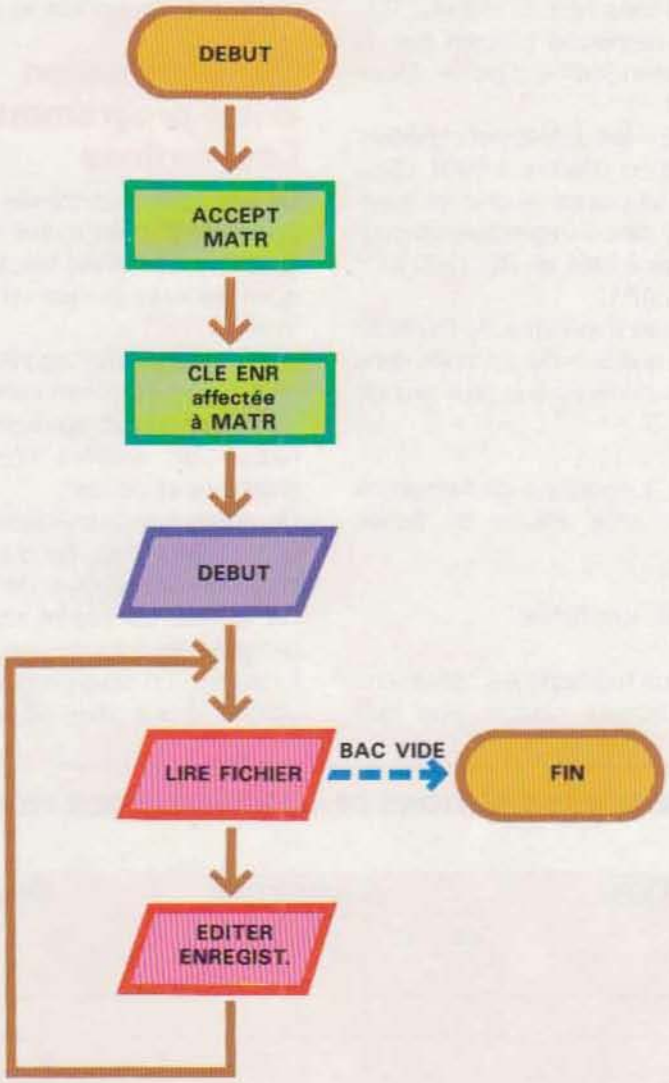
READ nom-fichier NEXT AT END...

selon le type de déclaration du fichier, respectivement en fichier séquentiel ou dynamique.
L'opération de positionnement n'est réalisable que sur des fichiers ouverts en entrée ou en E/S et après avoir défini l'accès séquentiel ou dynamique.
Le format général de l'instruction se trouve en bas de la page ci-contre.
Le « nom-donnée » désigne la clé d'enregistrement auquel il faut attribuer une valeur avant

POSITIONNEMENT SUR UN ENREGISTREMENT DU FICHIER IS SANS START



POSITIONNEMENT SUR UN ENREGISTREMENT DU FICHIER IS AVEC START



FORMAT GENERAL DE L'INSTRUCTION START

START nom-fichier KEY IS { EQUAL TO / = / GREATER THAN / > / NOT LESS THAN / NOT < } nom-donnée

INVALIDE KEY phrase-impérative.

l'exécution de START. En cas d'omission de la clause KEY, le compilateur supposera que l'enregistrement recherché possède la même clé que RECORD KEY. On reviendrait alors à la spécification de la forme KEY IS EQUAL TO.
Si l'enregistrement recherché n'existe pas, la phrase spécifiée est exécutée dans la clause INVALID KEY.
Lorsque la clause KEY est définie par supérieur à (GREATER THAN) ou inférieur à (NOT LESS THAN), le système va pointer le premier enregistrement de valeur de clé respectivement supérieure ou inférieure à celle de RECORD KEY avant l'opération START.
Dans ce cas, la phrase impérative de INVALID KEY n'est exécutée que si la clé attribuée dans RECORD KEY est supérieure à la plus grande clé du fichier.

Le verbe CLOSE. L'opération de fermeture d'un fichier IS est celle utilisée en fichier séquentiel :

CLOSE nom-fichier

Le tableau ci-dessous regroupe les opérations possibles sur des fichiers indexés ainsi que leurs conditions d'ouverture et d'accès.
Le tableau ci-contre dresse la liste des causes de sortie de fichier par INVALID KEY confrontées aux opérations exécutées en fonction des modalités d'ouverture et d'accès.

Communication entre programmes. Les routines

Un programme en format exécutable peut être composé d'un ensemble de sous-programmes (anglais subroutines) liés, logiquement et physiquement entre eux par un programme principal (main).
Avantage de cette organisation : chaque sous-programme peut être compilé et testé de manière tout à fait autonome. Cela permet de réduire de manière considérable le temps d'écriture et de test.
Un autre avantage, résultant de l'emploi des sous-programmes, est que l'on peut également créer une bibliothèque de fonctions simples ou complexes, qui seront réutilisées par d'autres programmes.
Exemple : un sous-programme qui contrôle la validité d'une date (CONTRDATE), ou qui

RESUME DES OPERATIONS REALISEES SUR DES FICHIERS IS

OPERATION	OUVERTURE			MODE D'ACCES		
	ENTREE	SORTIE	E/S	SEQ	DYN	ALE
OPEN INPUT ...	*			*	*	*
OPEN OUTPUT ...		*		*		
OPEN I-O ...			*	*	*	*
READ ... AT END ...	*		*	*		
READ ... NEXT AT END ...	*		*		*	
READ ... INVALIDE KEY ...	*		*			*
WRITE ... INVALIDE KEY ...		*		*		
WRITE ... INVALIDE KEY ...			*	*	*	*
REWRITE ... INVALIDE KEY ...			*	*	*	*
DELETE ...			*	*		
DELETE ... INVALIDE KEY ...			*		*	*
START ... INVALIDE KEY ...	*		*	*	*	

SEQ = SEQUENTIEL, DYN = DYNAMIQUE, ALE = ALEATOIRE

LISTE DES CAUSES D'INVALIDATION DES CLES DANS UN FICHIER IS

OPERATION	OUVERTURE	MODE D'ACCES	CLE INVALIDEE
READ	E-E/S	ALE	Pas de donnée identique à celle définie.
WRITE	S	SEQ	Manque espace sur disque.
WRITE	E/S	—	Donnée avec même clef déjà existante. Manque espace sur disque.
REWRITE	E/S	SEQ	Clef correspondant à celle de la donnée précédemment lue.
REWRITE	E/S	DYN/ALE	Il n'existe pas de donnée avec une clef égale à celle définie.
DELETE	E/S	DYN/ALE	Il n'existe pas de donnée avec une clef égale à celle définie.
START	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef égale à celle définie.
START ... KEY EQUAL ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef égale à celle définie.
START ... KEY GREATER ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef supérieure à celle définie.
START ... KEY NOT LESS ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef supérieure ou égale à celle définie.

SEQ = SEQUENTIEL DYN = DYNAMIQUE ALE = ALEATOIRE
E = ENTREE (Input) S = SORTIE (Output) E/S = ENTREE/SORTIE (I/O)

donne le jour de semaine d'une date (calendrier).
Ces deux sous-programmes (CONTRDATE ET CALENDRIER) sont étudiés en détail plus loin (voir listings des pages 1148 à 1151).
Rappelons que le programme principal et les sous-programmes sont reliés physiquement pendant l'opération de connexion (linking). De ce fait, l'occupation en mémoire réelle est bien égale à la somme des programmes.
Lorsqu'un programme, principal ou routine, appelle un sous-programme pour exécuter une fonction, il y a en général transfert des données à traiter.
Prenons un exemple.
Dans le cas du sous-programme de contrôle de date, le programme « appelant » doit fournir la date pour permettre au programme appelé de lancer ses opérations de vérification. Il transmettra ensuite, au programme principal, le résultat du contrôle effectué.

Cet échange d'information est obtenu en utilisant des champs de mémoire, identifiés par des noms symboliques décrits au niveau 01 ou 77 de la WORKING-STORAGE SECTION du programme d'appel.
Les champs correspondants du sous-programme sont définis, en respectant les règles, dans une section spéciale de la division données. Il s'agit de la LINKAGE SECTION, décrite à la fin de la WORKING-STORAGE SECTION.

Dans la LINKAGE SECTION, la clause VALUE n'est admise que si elle correspond aux niveaux 88.

Il n'est pas nécessaire que les champs de passage soient identiques dans le programme appelant et dans le programme appelé, mais ils doivent globalement avoir les mêmes PICTURE et les mêmes USAGE.

Les prévisions de l'ordinateur

Voilà plus de dix ans déjà que les météorologues emploient des satellites. Incontestablement, les données transmises au sol ont beaucoup contribué à faire évoluer la météorologie comme science théorico-expérimentale. Et surtout, elles ont beaucoup aidé à la formulation de prévisions plus fiables et à un degré de finesse insoupçonné il y a quelques années.

Ces résultats sont indéniablement dûs au rôle joué par les satellites. Mais aussi, et en grande partie, aux ordinateurs et aux techniques modernes de traitement de l'information.

Prenons un exemple européen, celui du nouveau centre italien pour l'acquisition et le traitement des données recueillies par les ***satellites météorologiques.*** *Il fait partie du Centre de Météorologie nationale et de climatologie de l'aviation (CNMCA) dépendant de l'Armée de l'air. Sa mission est de traiter les informations en provenance du satellite géostationnaire européen Meteosat.*

Le réseau national de saisie et de traitement des données est composé d'une station réceptrice primaire, de cinq stations secondaires, d'un site informatique et de vingt stations d'enregistrement automatiques. La station primaire (PDUS pour Primary Data User Station) reçoit les données numériques transmises par Meteosat ; les cinq autres stations, dites secondaires (SDUS pour Secondary Data User Station), réceptionnent les informations analogiques.

En effet, le satellite envoie vers la terre deux types de signaux : analogique et numérique. Le premier est une entité qui varie de manière pratiquement continue en se maintenant rigoureusement proportionnel à la grandeur mesurée ; le signal numérique, lui, est une série d'impulsions de type allumé/éteint comme celles traitées par les ordinateurs. Dans le second cas, la proportionnalité avec la grandeur observée se traduit en termes numériques.

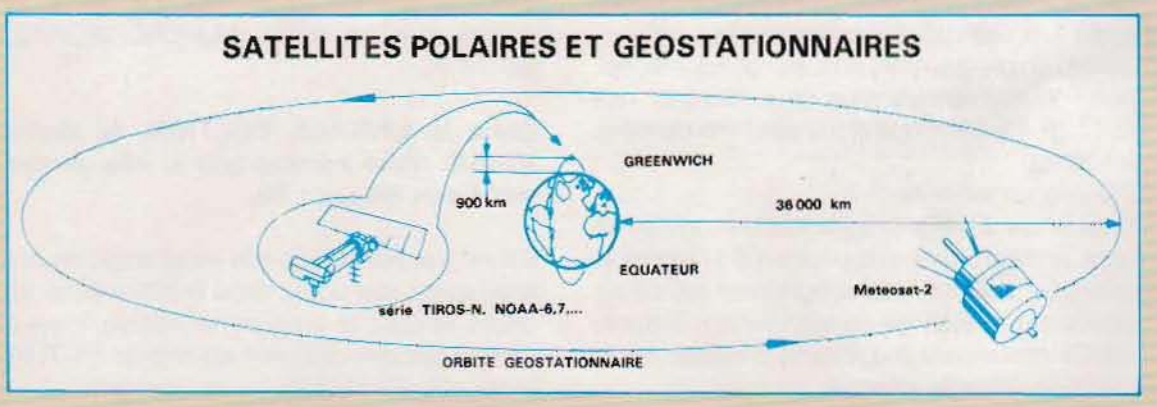
*La variable observée par Meteosat est la « **radiance** » de la surface terrestre. C'est la quantité d'énergie irradiée vers l'espace par la croûte terrestre, par les surfaces liquides et par le système nuageux. Le sous-système de calcul en mode autonome (off-line) traite localement les données contenues dans les signaux numériques ou analogiques les traduit en une représentation graphique.*

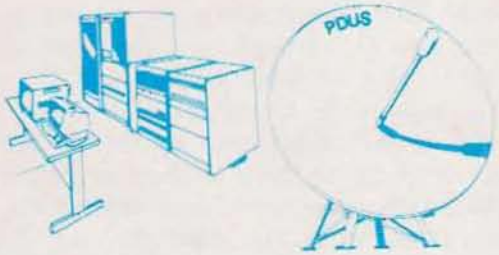
Les vingt stations automatiques de mesure des données de type DCP (Data Collection Platform) utilisent le ***Meteosat*** *comme plateforme de retransmission des données vers le* ***Réseau mondial des télécommunications météorologiques.***

La PDUS est composée d'un sous-système de réception, équipé d'une antenne parabolique d'environ 5 mètres de diamètre, d'un sous-système de calcul, d'équipements sophistiqués pour la représentation graphique et alphanumérique des images et des données.

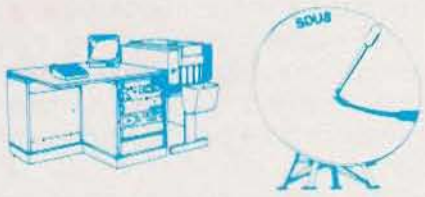
L'antenne reçoit, directement de Meteosat 2, des images à haute résolution. Après un prétraitement, les signaux sont renvoyés de Darmstadt (où se trouve le Centre opérationnel qui gère le satellite) à Meteosat.

Le cœur de la station primaire PDUS est composé d'un ordinateur de 2 Mo de mémoire RAM avec une mémoire de masse (disques magnétiques) d'environ 250 Mo. Pour la sauvegarde, deux unités de bandes magnétiques permettent la conservation, provisoire et historique, des données alors que la gestion opérationnelle des procédures de traitement des données est réalisée sur terminaux interactifs.

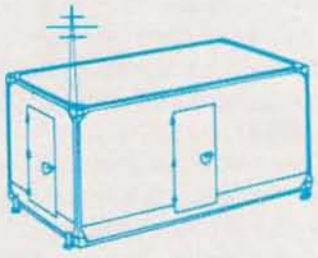




PDUS
PRIMARY DATA USER STATION



SDUS
SECONDARY DATA USER STATION



DCP
DATA COLLECTION PLATFORM





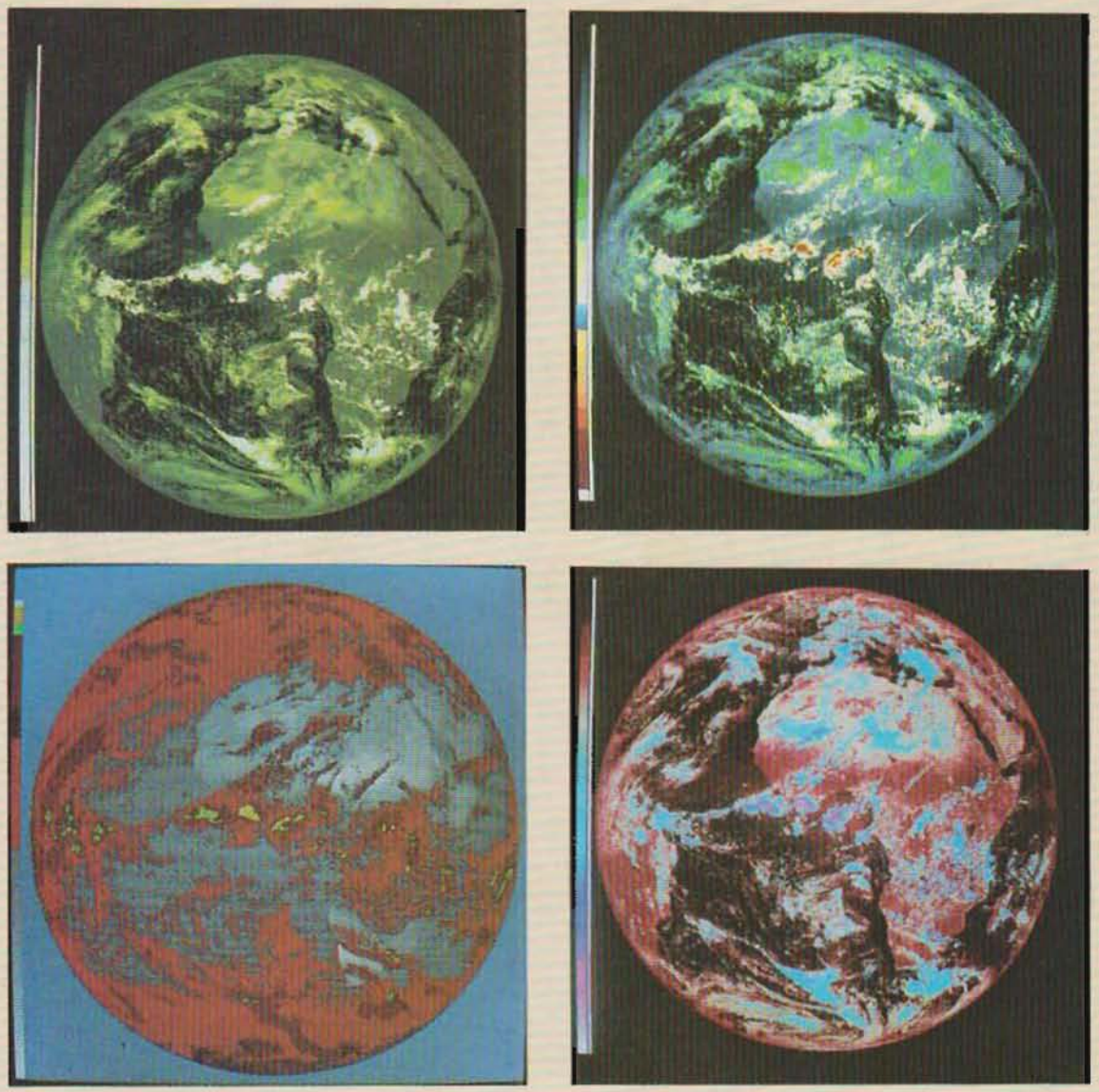


Représentation schématique de sous-systèmes de saisie et de traitement de données transmises par le satellite Meteosat.
En haut : station de travail du système ISIDE.
Au milieu : poste de travail dans une station réceptrice secondaire (SDUS) ; en bas, l'ordinateur central d'une station réceptrice primaire PDUS.

*Reliés à l'ordinateur, des appareils de photoreproduction numérique et graphique permettent également l'**impression de détails** d'une image (zoom) présentant un intérêt météorologique particulier.*

Un réseau de micro-ordinateurs (travaillant parallèlement à l'UC), d'interfaces et d'écrans graphiques permet de conserver en continu jusqu'à 32 images résolution pour chacune des trois bandes (visible, infrarouge et vapeur

d'eau) sur lesquelles le radiomètre de Meteosat travaille.
Les résultats peuvent également être des films élaborés par un système d'animation complexe reposant sur une structure de multiprocesseurs et sur une mémoire de 2 Mo.
Ainsi, les masses nuageuses et les perturbations défilent-elles sous les yeux du météorologue, reconstituant un historique du mouvement et de l'évolution sur une période de 16 heures (et plus) entre deux images, avec des intervalles de 30 minutes. De tels films sont également réalisables sur bande ou sur vidéocassette et reproductibles à loisir.
On peut employer le zoom interactif et la technique de la **fausse couleur**, fondée sur l'attribution d'une couleur à des valeurs données de la radiance, mesurée par Meteosat. Cela permet d'affiner sensiblement les possibilités d'interprétation physique des phénomènes atmosphériques en cours. La couleur est également utilisée en amont d'un traitement des données de **radiance** grâce à des techniques **multispectrales**. En effet, chacune des 32 images visualisées se compose d'un nombre considérable d'éléments d'image (pixels) : 1250 lignes × 2500 pixels par ligne dans la bande de la lumière visible et 650 × 1250 pour l'infrarouge et la vapeur d'eau. Le système mémorise la valeur numérique de la radiance correspondante à chaque pixel. Le traitement des données numériques, recueillies par la station primaire, fournit des mesures météorologiques très utiles à la prévision du temps.
L'une d'elles est la mesure de la vitesse des vents en altitude, déduite du déplacement des nuages et mesurable à partir de l'observation d'images successives d'une même zone. L'information est très importante puisqu'elle comble une lacune des observations traditionnelles en altitude au moyen de radio-sondes au-dessus des zones désertiques et des océans. La direction et la vitesse des vents, tirées des données de Meteosat, sont introduites tous les jours pour construire des **modèles numériques de prévision** du temps à court et à moyen terme. Une deuxième série importante d'informations concerne la mesure des températures à la surface des mers. Elle est obtenue par un traitement des données de la radiance dans la bande de l'infra-rouge (qui est proportionnelle aux températures du corps radiant selon le principe de Stefan-Boltzmann). Il faut également tenir compte de certaines corrections dûes à l'atténuation de la radiation quand elle traverse l'atmosphère et de l'angle d'inclinaison de la lumière solaire.
Ces valeurs de température de surface des océans sont couramment employées dans les modèles prévisionnels et pour les **bilans de radiation** du système Terre-Atmosphère.
L'analyse des nuages détermine, sur d'énormes surfaces, le pourcentage de ciel couvert et les différents types de nuages formant la nébulosité globale. Effectuée toutes les 6 heures, elle est directement exploitable pour l'assistance à la navigation aérienne. Des techniques statistiques permettent de classer les divers types de nuages et de fournir, par temps clair, des informations très utiles sur les courants marins.
L'altitude du plafond des nuages est une autre donnée que l'on peut extrapoler de l'analyse des radiances pour des zones importantes de l'Europe, de la Méditerranée et de l'Afrique. La température du plafond des corps nuageux étant inversement proportionnelle à leur altitude, on peut appliquer une couleur à des intervalles de température et afficher sur écran de télévision une carte représentant la surface externe des systèmes nuageux. Il s'agit bien d'une information précieuse pour l'assistance à la navigation aérienne, civile et militaire. Les données les plus intéressantes pour la climatologie sont principalement les **cartes de couverture du ciel**, de l'enneigement et des valeurs de radiation pour les océans et les déserts. Avec ces informations, on peut calculer la différence entre le total d'énergie qui entre et qui sort du système Terre-Atmosphère. Naturellement, ce « solde » détermine la température moyenne de la terre et conditionne le climat à l'échelle de la planète.
Les stations secondaires SDUS sont installées au CNMCA de Rome, dans les aéroports de Milan, de Cagliari, de Brindisi et de Palerme. Elles reçoivent les images analogiques des transmissions WEFAX (Weather Facsimile) de Meteosat dans les trois bandes de la lumière visible, de l'infra-rouge et de la vapeur d'eau, selon des formats et un programme définis par le centre principal de Darmstadt, en Allemagne Fédérale.
Les images sont composées de 800 lignes ×



Images en fausses couleurs fournies par une station PDUS.

800 pixels et couramment utilisées pour l'analyse du temps sur 24 heures. Contrairement à celles reçues par PDUS, ces images sont destinées à des usages de type non numérique.

Chaque station est munie d'un module récepteur (antenne, down-converter, récepteur) qui capte les signaux analogiques envoyés sur deux canaux de télécommunication (dans la bande S à 1691 et 1694,5 MHz) par un micro-ordinateur qui règle le flux des informations et qui accepte un ***dialogue interactif*** *avec l'utilisateur. Elle dispose également d'un système de visualisation avec des écrans en noir et blanc, d'un système d'enregistrement sur bande magnétique et d'un système de reproduction sur papier photographique (un reproducteur photo à rayon laser).*

Pourtant, le système de contrôle et de commande de la station est très simple. Ainsi, grâce à des ***procédures élémentaires de dialogue****, l'opérateur peut programmer le fonctionnement automatique de la station entière. Les signaux, captés par chacune des cinq SDUS, sont envoyés, par téléphone, à des terminaux de type facsimile ou vidéo (un maximum de dix par station). Ainsi, 50 organismes, répartis sur le territoire italien reçoivent-ils, en temps réel, les images Meteosat.*



Image, fournie par une SDUS, d'un système nuageux au-dessus de l'Europe.

La possibilité d'enregistrement des images sur bande magnétique (cassette) constitue un excellent moyen de créer un système d'archives locales, peu encombrant et relativement bon marché. Le système de traitement en mode local (off-line) a d'abord été utilisé par le service météorologique de l'aviation militaire, pour étudier les caractéristiques des images de Meteosat et développer des programmes de traitement des informations numériques en provenance du satellite. Ces programmes, après mise au point, sont progressivement intégrés à la PDUS.

Le système traite les images reçues presque en temps réel par la station primaire, grâce à une liaison à grande vitesse et à un accès direct à la mémoire des deux ordinateurs. Le système, qui s'appelle ISIDE (Interactive System for Image Data Elaboration) se compose d'un ordinateur analogue à celui de la PDUS, avec une mémoire centrale d'une capacité de 1 Mo et une mémoire de masse sur disques de 20 Mo. Il est complété par deux unités de bandes magnétiques, un terminal de contrôle, une table traçante et un écran graphique couleur.

Naturellement, le système ISIDE fournit des images en provenance de Meteosat, mais aussi d'autres satellites. Il s'agit d'images radar numériques ou de cartes météorologiques que l'on peut superposer à d'autres images visualisées sur écran.

Chacune des 20 stations automatiques DCP est munie de 8 capteurs qui contrôlent et mesurent les variables fondamentales au sol : intensité du vent, pression, température de l'air, humidité relative, précipitations, température au sol, radiation solaire.

Les stations, entièrement automatisées, déterminent les valeurs des paramètres reçus chaque heure, convertissent les signaux électriques des capteurs en paramètres physiques, au moyen de tableaux de correspondance et codifient les informations en leur attribuant le format adéquat.

Les messages sont enregistrés au niveau local et transmis à Meteosat. Le satellite renvoie le message de chaque DCP au centre opérationnel de Darmstadt qui le rediffuse sur le réseau mondial de télécommunications météorologiques. Les stations sont également reliées au CNMCA de Rome au moyen de lignes téléphoniques commutées qui permettent la transmission du message en cas de panne du satellite.

Le nouveau centre du Service météorologique pour le traitement de données provenant du satellite sera complété par des équipements destinés à la réception de données numériques ou sous forme d'images envoyées à Terre par des satellites polaires de la série TIROS-N.

Munis d'une résolution spatiale de l'ordre de 1,2 km (environ quatre fois celle de Meteosat), ces derniers fournissent des images de qualité supérieure. De plus, disposant d'un nombre élevé de capteurs radiométriques dans la bande de l'infra-rouge et dans celle des micro-ondes, ils sont en mesure d'effectuer des sondages sur les profils verticaux de température, avec une résolution spatiale d'environ 50 km.

Le Centre que nous venons de décrire est le résultat de la collaboration entre la DATAMAT, une entreprise italienne d'ingénierie des systèmes qui a fourni la station primaire PDUS, les stations secondaires SDUS et le système autonome, et la SIAP, une autre entreprise italienne d'équipements de précision qui a construit les stations automatiques DCP. Le Service météorologique de l'Armée de l'air d'Italie emploie le système de traitement des informations provenant du satellite en mode continu (24 heures sur 24). Les premiers résultats montrent que les objectifs fixés seront bientôt atteints.

D'après le Service Météorologique de l'Armée de l'air et de la Datamat Ingegnieria dei Sistemi S.p.A., en Italie.

En d'autres termes, le champ d'échange du programme principal et son équivalent du sous-programme peuvent avoir des descriptions différentes, à condition que la longueur globale et le type d'USAGE employé restent inchangés. Si, dans le programme appelant, on définit ainsi un champ :

```
01 A.
   05 A1        PIC X(2).
   05 A2        PIC X(4).
```

dans la LINKAGE SECTION du programme appelé, le même champ est décrit :

```
01 B
   05 B1        PIC X.
   05 B2        PIC X(3).
   05 B3        PIC X(2).
```

La définition des champs, dans la LINKAGE SECTION, n'implique pas d'allocation de la mémoire correspondante. En effet, le sous-programme prévoit que les champs seront déjà physiquement définis dans la mémoire du programme appelant. Le champs B de l'exemple précédent, défini dans la LINKAGE SECTION du programme appelé, occupe physiquement la même zone mémoire que le champ A défini dans la WORKING-STORAGE SECTION du programme appelant.

En résumé, le champ de LINKAGE SECTION d'un sous-programme peut être comparé à une REDEFINES du champ de WORKING-STORAGE SECTION du programme appelant.

Le verbe CALL

L'appel d'un sous-programme est réalisé par l'instruction CALL (premier format ci-dessous).

« Literal » représente une chaîne de caractères, entre guillemets, correspondant au nom défini dans la clause PROGRAM-ID du programme appelé ; « nom-donnée-1, nom-donnée-2... » ; ils désignent les champs de passage définis dans la WORKING-STORAGE SECTION.
Le nom du programme appelé est nécessairement unique : un programme ne peut appeler deux sous-programmes différents avec la même clause PROGRAM-ID.

PROCEDURE DIVISION d'un sous-programme

Pour compléter la description des conditions d'emploi d'un sous-programme, il faut se rappeler, qu'outre la LINKAGE SECTION, il est indispensable d'introduire la clause USING dans l'instruction PROCEDURE DIVISION selon le second format ci-dessous.
Les « nom-donnée-3, nom-donnée-4... » désignent des champs de passage définis dans la LINKAGE SECTION. L'ordre dans lequel ces noms sont définis doit refléter fidèlement l'ordre de définition des données dans la clause USING du CALL correspondant.
L'association entre les champs de passage du programme appelant et ceux définis dans la LINKAGE SECTION du programme appelé n'est en effet accomplie qu'au moyen de cette correspondance.

L'instruction EXIT PROGRAM

Cette instruction transfère le contrôle du sous-programme au programme appelant.
Elle a le format suivant :

```
EXIT PROGRAM
```

Un sous-programme peut comporter plusieurs

FORMAT DE L'INSTRUCTION CALL

```
CALL literal USING nom-donnée-1 nom-donnée-2 ...
```

PROCEDURE DIVISION D'UN SOUS-PROGRAMME

```
PROCEDURE DIVISION USING nom-donnée-3 nom-donnée-4 ...
```

STRUCTURE D'UN PROGRAMME APPELANT

```
IDENTIFICATION DIVISION
.
.
.
DATA DIVISION.
.
.
  WORKING-STORAGE SECTION.
  01 A            PIC X(8).
  01 B.
     05 B1        PIC 9(6).
     05 B2        PIC X(10).
.
.
.
*
*
PROCEDURE DIVISION.

PRINCIPALE SECTION.
  EXEMPLE-APPELANT.
     .
     .
     .
     CALL 'ROUTINE' USING B, A.
     .
     .
     .
```

STRUCTURE D'UN PROGRAMME APPELE

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   ROUTINE.
.
.
.
DATA DIVISION.
.
.
  WORKING-STORAGE SECTION.
  .
  .
  LINKAGE SECTION.
  01 C            PIC X(8).
  01 D.
     05 D1        PIC 9(6).
     05 D2        PIC X(4).
     05 D3        PIC X(6).
*
*
PROCEDURE DIVISION USING D, C.

PRINCIPALE SECTION.
  DEBUT.
  .
  .
  .
  SORTIE.
   EXIT PROGRAM.
```



Opération de chargement d'une bande magnétique dans un dérouleur.

EXIT PROGRAM en fonction des diverses sorties prévues. La structure schématique d'un programme principal (APPELANT) utilisant un sous-programme de PROGRAM-ID VERIFICATION se trouve sur le listing commençant en page 1148. Le sous-programme VERIFICATION, lui, se trouve dans le second listing commençant en page 1151. En suivant l'énoncé du listing, on peut vérifier l'application de certaines règles précédemment exposées :

— Les champs A et B, décrits dans la WORKING-STORAGE SECTION du programme appelant, sont passés au sous-programme au moyen de l'instruction CALL'VERIFICATION'USING B A.
— Le LITERAL employé dans l'instruction appelante contient le PROGRAM-ID du sous-programme (VERIFICATION).
— Au moyen de la ligne symbolique du sous-programme 'PROCEDURE DIVISION USING D C', on établit une correspondance entre les champs des programmes appelant et appelé.

Remarquons l'ordre de définition des champs. Aussi bien dans l'instruction d'appel :

CALL'VERIFICATION' USING B A

que dans la :

PROCEDURE DIVISION USING D C

on fait d'abord référence au champ de 16 caractères, puis à celui de 8 caractères.

Exemples d'application

Pour conclure cet exposé, nous présentons deux exemples de sous-programmes d'un emploi très fréquent :

1 / CONTRDATE, pour le contrôle et la conversion d'une date,
2 / CALENDRIER, pour le calcul du jour de la semaine correspondant à une date donnée.

Les fonctions accomplies par les deux programmes sont explicitées au niveau des REMARKS...
Observons qu'avant de réaliser le calcul demandé, la routine CALENDRIER appelle CONTRDATE et lui demande de contrôler la date transmise par le programme principal.

EXEMPLE D'APPLICATION : CONTROLE DU FORMATAGE D'UNE DATE

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID.   CONTRDATE.
   REMARKS. ================================================
            ==     SOUS ROUTINE POUR CONVERTIR UNE DATE     ==
            ================================================
            LE FORMAT DE PRESENTATION D'UNE DATE EST CONTROLE
            PAR LA VARIABLE OPTION :
            - OPTION = 1 : CONTROLE DE LA VALIDITE D'UNE DATE
            - OPTION = 2 : CONTROLE ET CONVERSION
                           JJMMAA -> AAMMJJ
            - OPTION = 3 : CONTROLE ET CONVERSION
                           JJMMAA -> AAJJJ (DATE JULIENNE)
                            ====
            LA DATE EVENTUELLEMENT CONVERTIE REMPLACE LA DATE
            D'ENTREE (VARIABLE DATE-ENTREE)
            SI LA DATE EST CORRECTE ERREUR =0 SINON ERREUR =1.
 *
 *
 *
  ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
 *
 *
    WORKING-STORAGE SECTION.
 *
    01 DATE-JJMMAA.
       05 JOUR.
          10 JJ            PIC 99.
       05 MOIS.
          10 MM            PIC 99.
       05 ANNEE.
          10 AA            PIC 99.
 *
    01 DATE-JJAAA REDEFINES DATE-JJMMAA.
       05 ALIGN-JULIEN     PIC 9(1).
       05 AA-JULIEN        PIC 9(2).
       05 JJJ-JULIEN       PIC 9(3).
 *
    01 TAMPON-99           PIC 99.
    01 RESTE               PIC 9 COMP.
       88 BISSEXTILE  VALUE 0.
 *
 *      tableaux de conversion.
```

```
*
    01 JOURS-DES-MOIS.
       05 FILLER           PIC X(24)       VALUE
          '312831303130313130313031'.
    01 TAB-JOURS-MOIS  REDEFINES JOURS-DES-MOIS.
       05 NB-JOURS         PIC 99  OCCURS 12.
    01 BASE-JULIEN.
       05 FILLER           PIC X(36)       VALUE
          '000031059090120151181212243273304334'.
    01 TAB-BASE-JUL REDEFINES BASE-JULIEN
       05 BASE             PIC 9(3) OCCURS 12.
*
*
    LINKAGE SECTION.
    01 OPTION              PIC 9.
    01 DATE-ENTREE.
       05 JJ-ENTREE        PIC 99.
       05 MM-ENTREE        PIC 99.
       05 AA-ENTREE        PIC 99.
    01 ERREUR              PIC 9.
*
*
*
 PROCEDURE DIVISION USING OPTION
                          DATE-ENTREE
                          ERREUR.
*
    INITIALISATION.
      MOVE 0 TO ERREUR.
    CONTROLE-OPTION.
      IF OPTION NOT = 1
                AND   2
                AND   3
          MOVE 1 TO ERREUR
          DISPLAY '*** OPTION', OPTION, 'NON PREVUE' UPON PRINTER
          GO TO FIN-PROGRAMME.
      MOVE DATE-ENTREE TO DATE-JJMMAA.
    CONTROLE-DATE-ENTREE.
      IF DATE-ENTREE = SPACES
          MOVE 1 TO ERREUR
          DISPLAY '*** DATE NON INTRODUITE' UPON PRINTER
          GO TO FIN-PROGRAMME.
       INSPECT JOUR   REPLACING ALL LEADING SPACES BY ZEROES.
       INSPECT MOIS   REPLACING ALL LEADING SPACES BY ZEROES.
       IF    JOUR  NOT NUMERIC
```

```
          OR  MOIS  NOT NUMERIC
          OR  ANNEE NOT NUMERIC
            MOVE 1 TO ERREUR
            DISPLAY '*** DATE NON NUMERIQUE' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONTROLE-MOIS.
        IF MM GREATER THAN 12 OR LESS THAN 1
            MOVE 1 TO ERREUR
            DISPLAY '*** NUMERO DU MOIS IMPOSSIBLE ' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONTROLE-BISSEXTILE.
        DIVIDE AA BY 4
                  GIVING TAMPON-99
                  REMAINDER RESTE.
        IF BISSEXTILE MOVE 29 TO NB-JOURS (2).
      CONTROLE-JOUR.
        IF JJ LESS THAN 1 OR GREATER THAN NB-JOURS (MM)
            MOVE 1 TO ERREUR
            DISPLAY '*** JOUR IMPOSSIBLE POUR CE MOIS' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONVERSION.
        IF OPTION = 1
            GO TO FIN-PROGRAMME.
        IF OPTION = 2
            MOVE JJ TO TAMPON-99
            MOVE AA TO JJ
            MOVE TAMPON-99 TO AA
            MOVE DATE-JJMMAA TO DATE-ENTREE
            GO TO FIN-PROGRAMME.
        MOVE 0 TO ALIGN-JULIEN, JJ-JULIEN.
        MOVE AA-ENTREE TO AA-JULIEN.
        IF BISSEXTILE AND MM-ENTREE GREATER THAN 2
            MOVE 1 TO JJ-JULIEN.
        ADD JJ-ENTREE
            JJ-JULIEN
            BASE (MM-ENTREE)   GIVING JJ-JULIEN.
        MOVE DATE-JULIEN TO DATE-ENTREE.
    *
    *
      FIN-PROGRAMME.
        EXIT PROGRAM.
```

EXEMPLE D'APPLICATION : CALCUL DU JOUR DE LA SEMAINE

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID.   CALENDRIER.
   REMARKS.  ==========================================
             ==      SOUS ROUTINE  CALENDRIER        ==
             ==========================================
               CETTE ROUTINE ACCEPTE EN ENTREE UNE DATE
             (DATE-ENTREE) DANS LE FORMAT JJMMAA; ELLE
             RESTITUE LE JOUR DE LA SEMAINE CORRESPONDANT
               UTILISE LA SOUS ROUTINE CONTRDATE
               LE CALCUL EST EFFECTUE EN PRENANT COMME
             BASE LE 1 JANVIER 1900 (LUNDI)
               EN CAS D'ERREUR LE CALCUL EST INTERROMPU
             ==========================================
*
*
*
 ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
*
*
   WORKING-STORAGE SECTION.
*
   01 OPTION                PIC 9.
   01 ERREUR                PIC 9.
   01 DATE-JJMMAA.
      05 JOUR.
         10 JJ              PIC 99.
      05 MOIS.
         10 MM              PIC 99.
      05 ANNEE.
         10 AA              PIC 99.
*
   01 DATE-JJAAA REDEFINES DATE-JJMMAA.
      05 ALIGN-JUL          PIC 9.
      05 AA-JULIEN          PIC 9(2).
      05 JJJ-JULIEN         PIC 9(3).
*
   01 QUOTIENT              PIC 9(5).
   01 RESTE                 PIC 9(5).
   01 NB-JOURS-BASE         PIC 9(5).
   01 INDICE                PIC 9.
*
*         TABLEAUX DES JOURS DE LA SEMAINE
```

```
*
   01 TABLE-NOM.
      05 FILLER          PIC X(3)        VALUE 'DIM'.
      05 FILLER          PIC X(3)        VALUE 'LUN'.
      05 FILLER          PIC X(3)        VALUE 'MAR'.
      05 FILLER          PIC X(3)        VALUE 'MER'.
      05 FILLER          PIC X(3)        VALUE 'JEU'.
      05 FILLER          PIC X(3)        VALUE 'VEN'.
      05 FILLER          PIC X(3)        VALUE 'SAM'.
   01 NOM REDEFINES TABLE-NOM.
      05 NOM-JOUR        PIC X(3) OCCURS 7 TIMES.
*
*
   LINKAGE SECTION.
   01 DATE-ENTREE        PIC X(6).
   01 J-SEM              PIC X(3).
*
*
 PROCEDURE DIVISION USING DATE-ENTREE, J-SEM.
*
 INITIALISATION.
   MOVE DATE-ENTREE TO DATE-JJMMAA.
   MOVE 3 TO OPTION.
   CALL 'CONTRDATE' USING OPTION, DATE-JJMMAA, ERREUR.
   IF ERREUR NOT = 0
       DISPLAY '*** ERREUR DATE ENTREE' UPON PRINTER
       STOP RUN.
*
*
 CALCUL.
   DIVIDE AA-JULIEN BY 4 GIVING QUOTIENT REMAINDER RESTE.
   IF RESTE = 0 SUBTRACT 1 FROM QUOTIENT.
   COMPUTE NB-JOURS-BASE = AA-JULIEN  * 365 +
                           JJJ-JULIEN +
                           QUOTIENT.
   DIVIDE NB-JOURS-BASE BY 7 GIVING QUOTIENT REMAINDER RESTE.
   COMPUTE INDICE = RESTE + 1.
   MOVE NOM-JOUR (IND) TO J-SEM.
*
*
 FIN-PROGRAMME.
   EXIT PROGRAM.
```

Le langage Fortran

Le Fortran (**FOR**mula **TRAN**slator) a été un des premiers langages de haut niveau tout spécialement destiné à l'usage scientifique. La première version officielle remonte à 1957 et, depuis cette date, de nombreuses modifications et adjonctions ont abouti au Fortran ANSI 77, version moderne assez proche du Basic. Le sigle ANSI vient des initiales de American National Standard Institute, l'équivalent de notre Afnor pour la normalisation.

En 1977, ANSI a mis la main aux dernières modifications. La version ANSI 77 est, de ce fait, un standard qui garde une certaine souplesse, car il existe d'autres formes de Fortran qui s'en écartent plus ou moins.

A l'époque où ont été mis au point le Fortran et le Cobol, l'ordinateur recevait les instructions au moyen d'un seul support physique : la carte perforée. En périphérique d'entrée, la machine disposait d'un lecteur de cartes et non d'un clavier.

Chaque instruction était tapée sur le clavier d'une perforatrice de cartes (sans aucune connexion à l'ordinateur), en respectant un format précis. Ainsi, le lecteur de cartes, puis le compilateur pouvaient interpréter de manière univoque le sens de l'instruction.

Le format de perforation des instructions sur les cartes, qui sont encore de nos jours utilisées mais plus rarement, est donc pris en considération par les compilateurs Fortran.

Durant les dix dernières années, l'arrivée du support magnétique a bouleversé les techniques de saisie, éliminant peu à peu les cartes perforées au profit des stockages des programmes sur disque et sur bande. Cependant, le format des instructions reconnu par les compilateurs n'a pas changé. Les diverses parties (champs) composant une instruction sont toujours tapées sur une longueur totale de 80 colonnes, puis découpées dans le format des cartes perforées.

Un lecteur de cartes IBM 3504. Il y a quelques années, les lecteurs de cartes étaient encore les seuls organes d'entrée pour tous les ordinateurs.



IBM

Pour cette raison, et comme pour le Cobol, nous ferons référence aux différents caractères d'une instruction en indiquant leur position sur une carte (colonne 1, colonne 2, etc.), même s'il s'agit d'une saisie au clavier. Nous présenterons le Fortran en nous appuyant largement sur les analogies qu'il présente avec le Basic, quitte à signaler leurs différences

Caractéristiques fondamentales du Fortran comparé au Basic

Tout d'abord, il s'agit d'un langage compilé. Il n'offre donc pas la facilité d'emploi et de mise au point du Basic interprété. Deuxième différence importante : la numérotation des lignes. En Basic, chaque ligne est numérotée dans un ordre croissant ; en Fortran, la numérotation est facultative : elle permet au programmeur d'identifier une ligne particulière (emploi de l'instruction GO TO, par exemple).
Conséquence : les numéros de ligne n'étant pas obligatoires, les sous-programmes Fortran seront identifiés différemment du Basic. Alors qu'en Basic, les sous-programmes sont adressés en se référant au numéro de la ligne initiale (par exemple GOSUB 1000), en Fortran ils seront « marqués » d'un symbole choisi par le programmeur.
Exemple : un sous-programme calculant la somme A = B + C sera en Basic :

```
1000 REM Somme
1010 A = B + C
1020 RETURN
```

et en Fortran :

```
SUBROUTINE SOMME
A = B + C
RETURN
```

En Fortran, la première ligne du sous-programme doit contenir le mot SUBROUTINE, suivi du nom symbolique (ici : SOMME) qui sera employé pour appeler le sous-programme lui-même.
En Basic standard, les instructions (y compris celles des sous-programmes) accèdent à toutes les variables. On peut ainsi attribuer des valeurs aux variables B et C dans n'importe quelle ligne du programme, calculer leur somme dans un sous-programme (le 1000) et utiliser le résultat (A) dans une autre zone du programme.
En Fortran, cela n'est possible qu'avec l'aide d'instruction spéciales :
chaque sous-programme utilise localement ses propres variables qui sont donc inconnues ailleurs.
Le transfert des variables entre sous-programmes se réalise de deux manières :

1 / en transférant les valeurs comme paramètre,
2 / en faisant une déclaration COMMON.

L'appel d'un sous-programme Fortran utilise l'instruction CALL... à la place de GOSUB... en Basic.
Par exemple, l'instruction d'appel du sous-programme précédent est CALL SOMME (en Basic on aurait eu GOSUB 1000). Pour transférer des variables en tant que paramètres, il faut en spécifier le nom symbolique, aussi bien à l'appel qu'à la première instruction du sous-programme qui s'en sert.
Dans l'exemple précédent, l'appel devient :

```
CALL SOMME (A, B, C)
```

et le sous-programme (qui s'appelle SOMME) doit commencer par l'instruction :

```
SUBROUTINE SOMME (A, B, C).
```

La comparaison entre les versions Basic et Fortran d'un même programme est présentée dans la tableau ci-contre.
En Fortran, tous les numéros de ligne ont disparu, sauf pour la première (ligne 10) qui est le point d'arrivée du GO TO (ligne 40 Basic et équivalent Fortran). Pour être identifiée, la ligne d'arrivée doit avoir un nom ou étiquette (label).
En Fortran, l'éventuelle numérotation des lignes n'a qu'une signification d'étiquette et ne suit aucune logique particulière.
La ligne 10 peut précéder la 5 et se trouver après la 1000. En Basic aussi, il est possible d'utiliser une numérotation avec des intervalles variables, mais l'ordre des valeurs est toujours croissant.
La seconde méthode de transfert des valeurs consiste à déclarer « commune » les para-

COMPARAISON DES VERSIONS FORTRAN ET BASIC D'UN MEME PROGRAMME

Version Basic

```
10 C=10
20 B=4.7
30 GOSUB 1000
40 GOTO 10
1000 REM Somme
1010 A=B+C
1020 RETURN
```

Version Fortran

```
10 C=10
   B=4.7
   CALL SOMME (A,B,C)
   GOTO 10
   SUBROUTINE SOMME (A,B,C)
   A=B+C
   RETURN
```

mètres à transférer grâce à l'instruction COMMON, analogue à celle du Basic. Le programme précédent devient (en Fortran) :

```
      COMMON A, B, C
10    C = 10
      B = 4.7
      CALL SOMME
      GOTO 10
      SUBROUTINE SOMME
      COMMON A, B, C
      A = B + C
      RETURN
```

En réalité, ce programme contient une erreur. Le compilateur est informé que le point de départ du sous-programme (SUBROUTINE SOMME) et sa fin, mais il ignore le début et la fin du programme.
En Basic, ce début correspond à la ligne ayant la numérotation la plus basse. En Fortran, il faut spécifier le point de départ avec l'instruction PROGRAM suivie d'un nom symbolique affecté au programme :

```
1       PROGRAM ESSAI
2     C EXEMPLE
3       COMMON A, B, C
4    10 C = 10
5       B = 4.7
6       CALL SOMME
7       GOTO 10
8       STOP
9       END
10      SUBROUTINE SOMME
11      COMMON A, B, C
12      A = B + C
13      RETURN
14      END
```

La numérotation croissante, qui apparaît à gauche des instructions, ne fait pas partie du programme et ne sert qu'à repérer les lignes pour d'éventuelles corrections.
La programmation commence avec l'instruction PROGRAM... (ligne 1) ; la ligne 2 contient un commentaire signalé de diverses manières. Les symboles les plus employés (comme le REM ou le symbole du Basic) sont C ! * tapés comme premier caractère de l'instruction.
Le Basic n'a pas de format fixe pour introduire les instructions : chaque ligne commence par son propre numéro et on poursuit avec l'instruction.
Par exemple, les instructions suivantes sont valables :

```
   10 A = B + C
10    A = B + C
1000  A = B + C
```

Noter : en Fortran, les instructions commencent à la colonne 7 (d'une carte hypothétique). Les commentaires doivent comporter le symbole indicatif lu en colonne 1, les lignes de suite (quand une instruction dépasse le cadre d'une seule ligne), un caractère quelconque (excepté le zéro et le blanc) en colonne 6. Les colonnes 1 à 5 sont réservées aux numéros de ligne.

Opérateurs arithmétiques, logiques et relationnels

S'agissant des opérateurs arithmétiques, le calcul des expressions s'effectue comme en Basic, sauf pour l'élévation à la puissance indiquée

QUELQUES DIFFERENCES ENTRE FORTRAN ET BASIC

	FORTRAN	BASIC
Numéro de ligne	Optionnels ; ne suivent aucun ordre précis	Nécessaires et en ordre croissant
Début programme	Nécessaire et identifié par un nom avec l'instruction PROGRAM nom	Commence toujours par la ligne affectée du plus petit numéro
Sous-programmes	Identifiés par un nom symbolique précédé d'un mot réservé SUBROUTINE. Finissent sur les instrutions RETURN et END	Identifiés par un numéro de ligne. S'arrêtent à l'instruction RETURN
Appel des routines	Instruction CALL suivie par le nom symbolique : CALL ESSAI	Instruction GOSUB suivie du n° de ligne : GOSUB 1000
Transferts de valeurs	Comme paramètre CALL ESSAI (A,B,C). Comme déclaration commune COMMON A,B,C	Pas nécessaire en termes explicités. Chaque point du programme accède à toutes les variables

par le symbole ** (comme en Cobol) : A ** 2 équivaut à A ↑ 2 ou à A ∧ 2.
Les opérateurs de relation sont également les mêmes, mises à part quelques différences de syntaxe avec le Basic.
Alors qu'en Basic, ces opérateurs sont indiqués symboliquement (>, <, =), en Fortran on les présente par un sigle normalement composé de deux lettres, précédé et suivi par un point.
Même constatation, enfin, pour les opérateurs logiques, sauf que la syntaxe du Fortran prévoit des points à droite et à gauche du code mnémonique (voir tableau du bas de la page 1157).

Variables et constantes en Fortran

Le langage Fortran accepte au moins les types de données suivantes :

— Entiers
— Réels en simple précision
— Réels en double précision
— Complexes en simple précision
— Complexes en double précision
— Logiques (booléennes)
— Caractères

D'autres formes prévoient des « entiers condensés » (short integer) et des « logiques condensés » (short logical) occupant un nombre de bits inférieur et économisant de l'espace mémoire.
Leur signification est proche du Basic, à l'exception des nombres complexes (qui n'ont pas d'équivalents en Basic standard) et des entiers (pour lesquels le Fortran prévoit une déclaration implicite). En Fortran, toutes les variables commençant par I, J, K, L, M, N sont supposées entières par défaut.
Cette différence, par rapport au Basic, n'a qu'une importance relative. Le programmeur acquiert rapidement l'habitude d'utiliser les lettres mentionnées pour des variables entières et, souvent, il les emploie aussi en Basic. Ceci explique pourquoi de nombreux programmes Basic font usage de variables tels que I, J, comme indices de boucles, même si ce langage ne les reconnaît pas, a priori, comme des entiers.
Les variables complexes constituent, elles, une particularité du Fortran (même si certaines formes de Basic les prévoient), en raison surtout de son usage scientifique. Limité à des applications bien précises, l'emploi de variables complexes implique d'importantes connaissances techniques.
Les données de type logique ont les mêmes fonctions qu'en Basic ; elles prennent les

valeurs **vrai** (TRUE) et **faux** (FALSE) représentées par les valeurs 1 et 0, comme en Basic. Cependant la syntaxe est différente puisque, en Fortran, TRUE et FALSE doivent être précédés et suivis d'un point : .TRUE. et .FALSE. En plus des variables et des constantes numériques, le Fortran prévoit l'emploi de constantes :

— hexadécimales
— octales
— Hollerith

Les constantes octales et hexadécimales ont le même sens qu'en Basic tout en employant une symbolique différente.
La notation la plus employée pour les octales utilise la lettre B après le nombre :

10B = 8 décimal

et, pour les hexadécimales :

Z'valeur'

où (valeur) indique la valeur hexadécimale de la constante (Z'A' = 10 décimal).
Les constantes d'Hollerith sont une forme particulière de chaîne, avec une syntaxe :

nH XXXX

où n indique le nombre de caractères de la chaîne (XXXX) et H (abréviation de Hollerith) le symbole indiquant le type de constante.

Par exemple :

19HCECI EST UN ESSAI

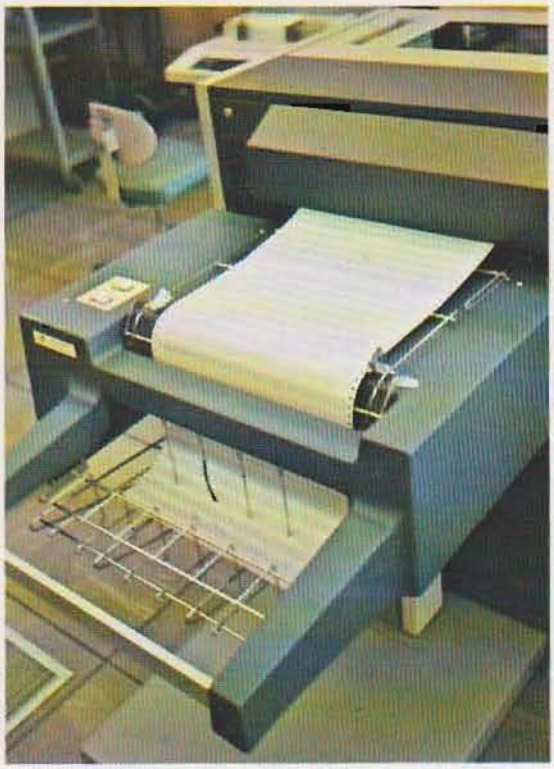
est une forme valable de constante de Hollerith.
Remarquons que les espaces entre les mots sont comptés.
Le traitement des chaînes en Fortran est assez différent du Basic. Précisons qu'il n'existe pas en Fortran de variable de type chaîne ; on n'utilisera donc pas de noms avec le symbole $. Les caractères d'une chaîne sont mémorisés en variables entières, occupant 8 bits chacun. Dans une variable entière (normalement 16 bits), on peut ranger 2 caractères, l'un dans la moitié gauche (bits 0 à 7) et l'autre dans la moitié droite (bits 8 à 15). Pour mémoriser plus de 2 caractères, il faut dimensionner un tableau composé d'autant d'éléments qu'il existe de couples de caractères.
Outre les entiers, d'autres types de tableaux (par exemple en double précision) sont parfois utilisés. Dans ce cas, le nombre de caractères contenu dans un élément est supérieur. Certains compilateurs admettent une méthode typique du Basic qui consiste à désigner un tableau avec un nom de variable. La gestion des tableaux est alors analogue à celle du Basic, sans passer par $.
Par exemple, sont identiques* les attributions :

NOM = « Essai » (Basic).
NOM : = 'Essai' (Fortran)

(*) En Fortran, on utilise normalement les guillemets simples.

OPERATIONS LOGIQUES ET DE RELATION

Fortran	Basic	Abrév. angl.	Signification	Exemple Fortran	Exemple Basic
.EQ.	=	Equal	égal	IF (A.EQ.B)	IF A=B
.NE.	<>	Not equal	différent	IF (A.NE.B)	IF A<>B...
.LT.	<	Less than	inférieur	IF (A.LT.B)	IF A < B...
.GT.	>	Greater than	supérieur	IF (A.GT.B)	IF A > =B...
.LE.	< =	Less-equal	inférieur ou égal	IF (A.LE.B)	IF A < =B...
.GE.	> =	Greater-equal	supérieur ou égal	IF (A.GE.B)	IF A > =B...
.NOT.	NOT		négation	.NOT.B	NOT B
.AND.	AND		ET logique	A.AND.B	A AND B
.OR.	OR		OU logique	A.OR.B	A OR B



Marka/Infophoto

Alimentation en continu d'une imprimante.

L'opération d'extraction d'une partie de la chaîne exige d'en spécifier les limites (caractères extrêmes).
Dans NOM = « Essai », la ligne :

```
NOM 1 = NOM (2 :3)
```

extrait et attribue à NOM 1 la valeur « ss », elle équivaut en Basic à NOM 1$ = MID$(NOM$ (2,2)).
En outre, on peut concaténer 2 chaînes avec le symbole. Ainsi, les lignes.

```
N1 = 'Essai'.
N2 = 'de concaténation'
N = N1 / / N2
```

attribuent à la variable N la valeur « Essai de concaténation ». L'équivalent Basic emploie le symbole + (N1$ + N2$) ou &.
En Fortran, si on attribue à une variable la valeur résultant d'un calcul quelconque, le compilateur lui affecte un type lié aux opérateurs. Exemple : on définit A et B comme des entiers ; le calcul A = B + C mémorise le résultat dans la variable entière A.
Le tableau ci-contre liste les différentes combinaisons possibles et permet d'éviter les erreurs de troncage, provoquées par des déclarations erronées. Exemple : si on multiplie un réel en simple précision par un entier, le résultat sera un réel en simple précision. Mémorisé dans un entier, le résultat induirait des erreurs de troncage. Aussi est-il recommandé d'utiliser une variable d'un type plus long que celui des différents opérateurs du calcul.

Déclaration du type de variables

En Fortran, le type est défini par le programmeur (entier, réel...), avec une syntaxe de déclaration :

```
type nom-1, nom-2, ...
```

où « type » désigne un type de variables et « nom-1, nom-2, ... » des noms symboliques de variables auxquelles on désire attribuer le type précisé.
Les 6 types principaux prévus en Fortran sont :

INTEGER	Entier
REAL	Réel en simple précision
DOUBLE PRECISION	Réel en double précision
COMPLEX	Complexe
LOGICAL	Logique
CHARACTER	Caractère

Ainsi, les lignes :

```
INTEGER A, NOM, K
REAL V
```

définissent comme entières les variables A, NOM et K et comme réelle la variable V.
Ce type de déclaration présente des formes et des fonctions analogues en Basic ;

```
INTEGER A
DOUBLE PRECISION Z
```

équivaut à :

```
DEFINT A
DEFDBL Z
```

Notons, qu'en Basic, la déclaration de type réel n'est pas nécessaire (elle existe toutefois avec

pour syntaxe DEFSNF) dans la mesure où les variables sont supposées réelles par défaut, sauf indication contraire. En Fortran, il se produit la même chose mais la déclaration de type REAL est à faire, ne serait-ce que pour définir comme réelles des variables dont le nom commence par I, J, K, L, M, N.
Dans certaines formes de Fortran, il est possible de définir un type en précision étendue. Ainsi, la déclaration INTEGER devient INTEGER * 2 (précision double) ou INTEGER * 4 (précision quadruple). INTEGER * 2 déclare toujours un nombre entier, mais avec une précision double par rapport à celle que donne INTEGER (à ne pas confondre avec DOUBLE PRECISION qui concerne les réels).
La déclaration CHARACTER définit une variable de chaîne avec la syntaxe :

CHARACTER * n NOM

La variable appelée nom contiendra n caractères. Si on omet n, le compilateur supposera qu'il est égal à un.
Le résultat est semblable à celui obtenu en Basic en écrivant le nom de la variable suivi du symbole S ; mais dans ce cas, il faut définir la longueur de la variable comme nombre de caractères. En Basic, c'est le système qui adapte l'espace réservé à la chaîne, au fur et à mesure que le contenu varie ; en Fortran, la longueur est posée à priori.
La déclaration CHARACTER concerne également un tableau. Ainsi, la déclaration :

CHARACTER * 3V(10)

définit 10 chaînes (V) des caractères chacune.
L'instruction DIMENSION (dimensionnement des tableaux) est alors inutile, dans la mesure où l'espace mémoire est déjà réservé par la déclaration de type.
Les déclarations de type sont « explicites »

TYPES DE VARIABLES RESULTANT DES CALCULS

PREMIER TERME

SECOND TERME	**Entier réduit**	**Entier**	**Réel en simple précision**	**Réel en double précision**	**Complexe en simple précision**	**Complexe en double précision**
Entier réduit	ER	E	R	DP	C	CD
Entier	E	E	R	DP	C	CD
Réel en simple précision	R	R	R	DP	C	CD
Réel en double précision	DP	DP	DP	DP	CD	CD
Complexe en simple précision	C	C	C	CD	C	CD
Complexe en double précision	CD	CD	CD	CD	CD	CD

ER = entier réduit
E = entier
R = réel en simple précision
DP = réel en double précision
C = complexe en simple précision
CD = complexe en double précision

dans la mesure où chaque variable est indiquée avec son propre nom. On peut définir en même temps le type de plusieurs variables regroupées autour de l'initiale du nom : on parle alors de déclaration « implicite ».
La syntaxe est la suivante :

```
IMPLICIT type (X-Y)
```

où X et Y indiquent l'intervalle considéré. La déclaration implicite :

```
IMPLICIT INTEGER (A-L)
```

définit comme **entières** toutes les variables comprises entre A et L. Sa signification est identique au Basic :

```
DEFINT A-L
```

Certains compilateurs Fortran possèdent l'option IMPLICIT NONE, qui élimine toutes les déclarations implicites.

Dimensionnement des variables structurées

Les instructions qui réservent des zones de mémoire à des variables dimensionnées sont les mêmes qu'en Basic : DIMENSION (en Basic DIM) et COMMON.
DIMENSION est employée de la même manière que DIM Basic. Mais COMMON a une autre signification, elle sert aussi à déclarer commun un tableau et, en même temps, à le dimensionner. La déclaration de type offre également cette opportunité puisqu'il est possible de dimensionner un tableau.

Les instruction COMMON et EQUIVALENCE

Les programmes Fortran et Basic ont la même structure : un programme principal appelle des sous-programmes accomplissant chacun sa propre tâche.
Ils ne se différencient que par le transfert des variables. En Fortran, chaque routine a ses propres variables, dans son sein, et qui restent, à moins qu'elles n'aient été déclarées communes par COMMON. Dans la déclaration COMMON, comme dans celle du type, des variables structurées peuvent être dimensionnées avec, pour double effet, de rendre communs certains tableaux et d'en permettre le dimensionnement.
Exemple : si la variable A(20) doit être utilisée à divers endroit du programme, avec l'instruction :

```
COMMON A(20)
```

on définit la dimension et, simultanément, on impose l'usage commun du tableau.
Dans le graphique de la page 1161 on trouve, représentées schématiquement, certaines situations typiques de l'emploi des instructions COMMON et DIMENSION. L'instruction COMMON peut être employée pour transférer simultanément un bloc de variables désigné par un nom-étiquette **(labelled common).**
Dans :

```
COMMON/ESSAI/V1,R,K,M
```

ESSAI nomme le bloc de variables V1,R,K,M. L'usage du bloc permet de modifier les noms des variables d'un sous-programme à l'autre. Si l'instruction précédente est insérée dans le programme principal et dans un sous-programme, par exemple, nous aurons :

```
COMMON/ESSAI/A1,A2,B(2)
```

qui mettaient en correspondance les significations suivantes :

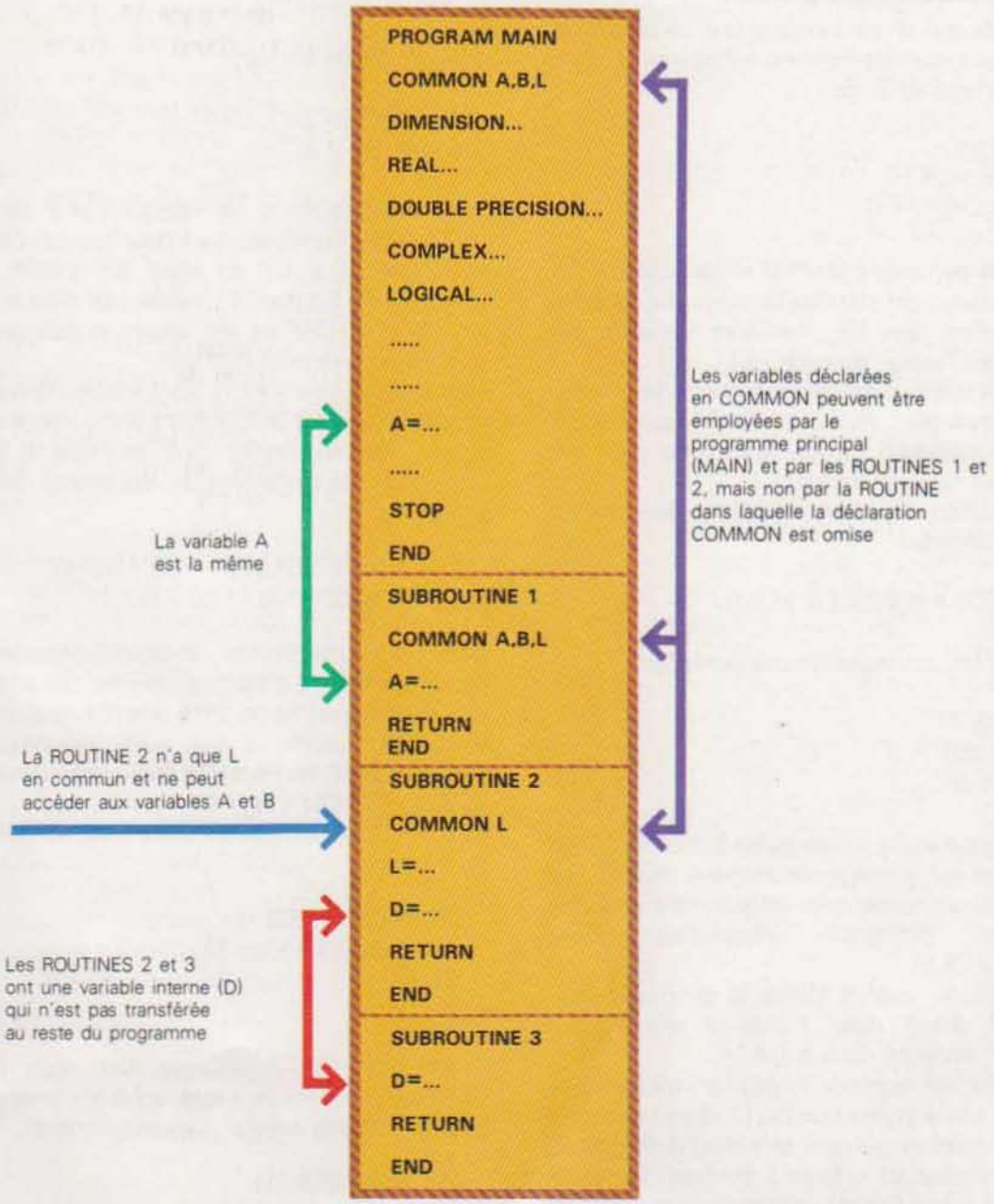
```
A1=V1, A2=R, B(1)=K, B(2)=M
```

Faire appel à une étiquette commune revient à assigner une zone de mémoire aux valeurs d'un bloc de variables. Quand, dans l'instruction, le nom de bloc n'apparaît pas, on parle alors de **blank common.** L'instruction COMMON A(20) est un exemple. Contrairement au Basic, en Fortran, on peut déclarer « équivalents » des noms de variables (même dimensionnées). L'instruction EQUIVALENCE (A,B) assigne, aux deux noms A et B, la même zone mémoire. Pour des variables dimensionnées, il faut s'assurer que l'EQUIVALENCE concerne les éléments contigus. Les instructions :

```
DIMENSION    C(3)
EQUIVALENCE (C(1)A), (C(2)B)
```

sont correctes, dans la mesure où les éléments C(1) et C(2) sont contigus.
En revanche, l'instruction :

TRANSFERTS DE VALEURS AVEC "COMMON"



```
EQUIVALENCE (C(1),A),(C(3),D)
```

comporte une erreur : C(1) et C(3) appartiennent au même tableau sans être contigus.

Instructions d'assignation

L'attribution d'une valeur à une variable s'obtient automatiquement en utilisant le symbole =, comme en Basic.

```
V=3.5
A=7+V*2
B=A*3/(2+1)
```

Ces formes sont correctes et identiques à celles du Basic dans ses versions les plus récentes (rappelons que les premières versions prévoyaient l'usage du code LET).
Un deuxième type d'assignation des valeurs est donné par l'instruction DATA, qui a la même signification qu'en Basic, mais avec une syntaxe bien différente.
L'instruction Fortran (notation américaine du point décimal) :

```
DATA A,R,V / 3.7,4.21,5.9 /
```

établit les correspondances suivantes :

```
A=3.7
R=4.21
V=5.9
```

La logique est la même qu'en Basic : à chaque variable est affectée, de manière linéaire, une des valeurs numériques listées, mais selon des formes distinctes (l'équivalent Basic 3.7,4.21,5.9).
Au-delà de l'aspect formel, la principale différence réside dans l'absence d'instruction READ, implicite dans la DATA.
En Basic, on regroupe toutes les variables assignées à une instruction DATA dans un endroit du programme qui peut être éloigné de celui où sont rangées les valeurs à attribuer. L'instruction READ incrémente le pointeur et va extraire les valeurs dans la zone DATA.
Telle qu'elle est structurée, l'instruction Fortran n'autorise pas cet éclatement géographique : variable et valeur à attribuer doivent apparaître dans la même instruction.

Ainsi les instructions Basic :

```
10 READ A,B,C
20 READ D
30 RESTORE
40 READ E,F
50 DATA 1,2,3,4
```

deviendront en Fortran :

```
DATA A,B,C,D/1,2,3,4/
DATA E,F/1,2/
```

Dans la forme Basic, les valeurs 1 et 2 de la DATA sont employées tant pour les variables A, B que pour E,F (la ligne 30 rétablit le pointeur). En Fortran, il n'existe pas d'équivalent de RESTORE et les valeurs numériques sont nécessairement répétées.
Pour assigner des valeurs aux variables dimensionnées (tableau) on a recours à une forme de boucle implicite comme dans l'exemple du tableau V(3) (de dimension 3), qui admet deux expressions :

```
DATA V(1), V(2),V(3) / 11.6,7.9,3.1 /
DATA (V(I),I=1,3) / 11.6,7.9,3.1 /
```

Observons qu'en Fortran, le dimensionnement implicite en phase d'assignation n'est pas autorisé. Alors qu'en Basic notre exemple est sans erreur (la variable V est automatiquement dimensionnée), en Fortran il exige une instruction DIMENSION V(3).
La forme complète serait alors (avec le point décimal) :

```
DIMENSION V(3)
DATA (V(I),1=1,3) / 11.6,7.9,3.1 /
STOP
END
```

L'instruction DATA accepte tout type de constante, pourvu qu'il soit cohérent avec le type de variable associé. Les instructions :

```
CHARACTERS CH
DATA CH/'A'/
```

définissent CH comme variable contenant des caractères et lui attribuent le caractère A (notons que, dans la DATA, les caractères sont entre guillemets).

Option fréquemment rencontrée dans l'instruction DATA : le symbole * qui signifie répétition.
Par exemple, la ligne :

```
DATA (V(I),1=1,3) / 3*5 /
```

assigne la valeur 5 aux trois éléments du tableau V(3) : la formule 3* signifie « trois fois la valeur qui suit ».
Pour certains compilateurs Fortran, la boucle implicite peut être omise ; ainsi l'instruction précédente s'écrira aussi :

```
DATA V / 3*5 /
```

la variable V ayant été préalablement définie comme un tableau à 3 éléments (DIMENSION V(3).
Les formes principales de DATA sont résumées dans le tableau ci-dessous. La dernière méthode d'assignation des valeurs numériques typique du Fortran consiste à utiliser l'instruction PARAMETER, dont la syntaxe est :

```
PARAMETER (nom-1=const-1, nom-2=...)
```

Celle-ci définit des constantes avec un nom symbolique. Les constantes peuvent appartenir à n'importe quel type, à condition de rester homogènes avec le type associé au nom (par exemple, constantes entières avec noms commençant par I,J,K,L,M,N).

L'instruction :

```
PARAMETER (PI=3.14)
```

assigne, au nom symbolique PI, la valeur 3.14. Dans ce cas, on n'a besoin d'aucune déclaration de type, dans la mesure où PI est un nom auquel on peut associer une valeur réelle (3.14). Si la compatibilité n'existe pas, il faut définir le type de variable avant PARAMETER. Par exemple :

```
CHARACTER*5 D
PARAMETER (D='ESSAI')
```

définit D comme chaîne de 5 caractères et lui attribue la valeur ESSAI.

Les instructions Fortran

Examinons maintenant les principales instructions Fortran, en insistant sur les différences avec le Basic. En effet, la remarquable similitude entre les deux langages permet de simplifier l'exposé en se limitant à l'énumération des différences fondamentales.

L'instruction GOTO

En Fortran, il existe 3 types de GOTO(*) :

- GOTO inconditionnel
- GOTO calculé
- GOTO assigné

(*) La présence d'un espace entre les mots GO et TO n'est pas significatif, comme c'est le cas en Basic.

FORMAT DE L'INSTRUCTION DATA

CHARACTER CH DATA CH/'A'/	Déclaration de type DATA variable, variable,... / valeur, valeur,.../
DIMENSION V(5) DATA (V(I),I=1,3)/4,2,7,1,8,3/	Dimensionnement DATA boucle/valeurs/
DIMENSION V(5) DATA (V(I),I=1,3)/3*5,1/	Dimensionnement DATA boucle/facteur de répétition*valeur/
DIMENSION V(5) DATA V/5*3,7/	Dimensionnement DATA variable dimensionnée/facteur de rép.*valeur/

Le mode inconditionnel est commun au Fortran et au Basic : même spécification d'adresse du saut (ligne identifiée par un numéro) et même syntaxe :

GOTO numéro de ligne

Bien qu'il ait son équivalent Basic, le GOTO calculé a un format différent. En Basic, l'instruction est ON K GOTO... ; en Fortran la syntaxe est :

GOTO (n° -ligne-1, n°-ligne-2,...), K

Signification : pour K=1, sauter au premier numéro de ligne indiqué (n°-ligne-1), pour K=2 au deuxième, etc. La forme exacte prévoit une virgule avant le paramètre, mais elle est généralement omise. L'indice (paramètre K) peut également être le résultat d'un calcul, ainsi :

GOTO (100, 10, 1240), R+4

déclenchera un saut à une des instructions 100, 10 ou 1240, selon le résultat du calcul R+4 (ou mieux : selon la valeur de la partie entière de ce résultat).
Le GOTO assigné, enfin, n'a pas d'équivalent Basic. Il permet d'employer un symbole à la place du numéro de ligne et doit être précédé d'une instruction. Celle-ci attribue le nom symbolique à la ligne.
L'instruction GOTO 150 devient ainsi GOTO ARRIVEE après avoir assigné, à la ligne 150, le nom ARRIVEE. L'instruction effectuant cette assignation est :

ASSIGN n°-ligne TO nom

Pour l'exemple précédent, on aurait :

```
ASSIGN 150 TO ARRIVEE
. . . .
GOTO ARRIVEE
```

Au moment d'exécuter l'instruction GOTO ARRIVEE, le programme transfère le contrôle à la ligne n° 150.

Les boucles

Le déroulement d'une boucle suit les règles décrites dans la présentation du Basic, avec une syntaxe Fortran particulière :

DO n indice=valeur initiale, valeur finale, pas

DO est le code qui déclenche la boucle (équivalent au FOR) ; « n » est le n° de ligne fin du bloc d'instructions à exécuter itérativement dans la boucle ; « indice » est l'indice de la boucle qui varie de la « valeur-initiale » à la « valeur-finale » avec des progressions égales au « pas » (intervalle constant entre deux indices).
Si la valeur du pas n'est pas donnée, le compilateur la lira par défaut comme égale à 1.
L'instruction :

```
DO 100 I=1,22,3
```

active l'itération de la partie du programme comprise entre cette ligne et celle numérotée 100, en faisant varier l'indice entre les valeurs 1 et 22 avec un pas égal à 3. La ligne d'arrêt de la boucle ne doit contenir aucune référence à l'indice (en Basic, on doit écrire l'instruction NEXT I) et il pourrait s'agir de n'importe quelle instruction Fortran. En réalité, on termine généralement par l'instruction CONTINUE « poursuivre ». Celle-ci n'a aucun effet sur le programme ; son seul but est d'identifier un n° de ligne marquant la fin d'une boucle.
La boucle précédente, complétée, devient :

```
    DO 100 I=1,22,3
    . . . .
    . . . .      Instruction exécutées
    . . . .      de manière itérative
    . . . .
100 CONTINUE
```

En outre, certaines formes de Fortran ont adopté un code plus explicite : END DO. Le programme devient alors :

```
    DO 100 I=1,22,3
    (itération)
100 END DO
```

Boucles implicites. En Fortran, il est possible d'activer une boucle de manière implicite en omettant l'instruction DO. Un exemple illustre le mode d'attribution avec une DATA appliquée à un tableau. En général, la boucle impli-



IBM

Perforatrice de cartes IBM 3325, un des périphétiques standard.

cite (appelée aussi DO implicite) a le format :

XX (tableau indice=val-init, val-fin, pas)...

La partie centrale de l'instruction (entre parenthèses) est la boucle proprement dite ; XX représente une instruction admettant ce type de boucle et les points, une suite éventuelle. Dans le cas d'une assignation avec DATA, la partie XX est remplacée par DATA, et la suite de l'instruction est composée des valeurs numériques qu'il faut attribuer (11.6,7.9,.../)
Il n'existe que 2 types d'instructions admettant la boucle implicite :

— DATA
— Instructions d'E/S

Pour éditer le contenu d'un tableau avec 10 éléments, on peut adopter l'instruction :

```
WRITE (p,n)(V(I),I=1,10)
```

WRITE (équivalent du PRINT Basic) est le code de l'instruction ; « p » et « n » sont des paramètres qui indiquent où (sur quel périphérique) et comment (avec quel format) écrire ; l'expression entre parenthèses indique les variables à écrire.
Le DO implicite est formellement semblable à un FOR... NEXT... exprimé sur une seule ligne. Par exemple, pour écrire les éléments du tableau précédent en Basic, on pourrait avoir :

```
FOR I=1 TO 10 : PRINT V(I) : NEXT I
```

équivalente à l'instruction Fortran :

```
WRITE (p,n)(V(I),I=1,10)
```

Les DO implicites peuvent être emboités, c'est-à-dire contenus l'un dans l'autre.
L'impression d'un tableau à deux dimensions s'obtient avec les instructions :

```
DIMENSION A(5,3)
. . . . . .
WRITE (p,n) ((A(I,J),J=1,3),I=1,5)
```

équivalentes au Basic :

```
10 FOR I=1 TO 5
20 FOR J=1 TO 3
30 PRINT A(I,J)
40 NEXT J
50 NEXT I
```

La forme DO... WHILE... (FAIRE... JUSQU'A CE QUE...)
Ce type de boucle n'est pas prévu dans le Fortran ANSI 77 ; il n'existe que sur certaines machines. Il est semblable au Basic WHILE... END, avec la syntaxe :

DO n WHILE (condition)

où « n » est le n° de ligne où s'arrête le DO et « condition » une condition quelconque qui provoque la répétition du cycle si elle se vérifie. Dans l'instruction :

```
DO 100 WHILE (A.GT.B)
```

la boucle est répétée jusqu'à ce que (WHILE) la valeur de la variable A soit supérieure (.GT.) à la valeur B. Normalement, le n° de ligne est facultatif et la fin du DO est indiqué par END DO. Dans ce cas, l'exemple précédent devient :

```
DO WHILE (A.GT.B)
. . . .
. . . .
END DO
```

sans aucun n° de ligne.

Instructions conditionnelles

Comme en Basic, la condition est obtenue avec IF. En Fortran, il existe trois types de IF :

- IF arithmétique
- IF logique
- IF par blocs

Les deux premiers types sont communs à toutes les versions du langage, alors que le dernier est caractéristique des versions les plus récentes.
La syntaxe de l'IF arithmétique est la suivante :

IF (expression) ligne-1, ligne-2, ligne-3

L'expression est évaluée et le contrôle est transféré à la ligne correspondante en fonction du signe du résultat. Par exemple, si on exécute l'instruction :

```
IF (A+B) 10,100,1000
```

le programme saute à la ligne 10, 100 ou 1000 selon le résultat (négatif, nul, positif) de la somme A+B.
Le contenu de l'instruction est limité à 3 adresses seulement, puisque la sélection s'opère selon le signe : le contrôle passe à la 1ère ligne (10) si le résultat est négatif, à la 2e ligne (100), s'il est égal à zéro et à la 3e (1000) s'il est positif (voir graphique ci-contre).
L'IF arithmétique du Fortran équivaut à 3 IF logiques en série. Par exemple, l'instruction

```
IF (K) 10,20,30
```

signifie

```
IF (K.LT.0) GOTO 10
IF (K.EQ.0) GOTO 20
IF (K.GT.0) GOTO 30
```

Hormis quelques écarts symboliques (.LT.,.EQ.), ces IF correspondent à ceux du Basic (IF K=0 GOTO 10, etc.).
Dans l'IF arithmétique, les nos de lignes peuvent être répétés :

IF (K) 10,10,10 transfère dans tous les cas le contrôle à la ligne 10
IF (K) 10,20,10 transfère le contrôle à la ligne 20 si K=0
IF (K) 10,10,20 transfère le contrôle à la ligne 20 si K > 0

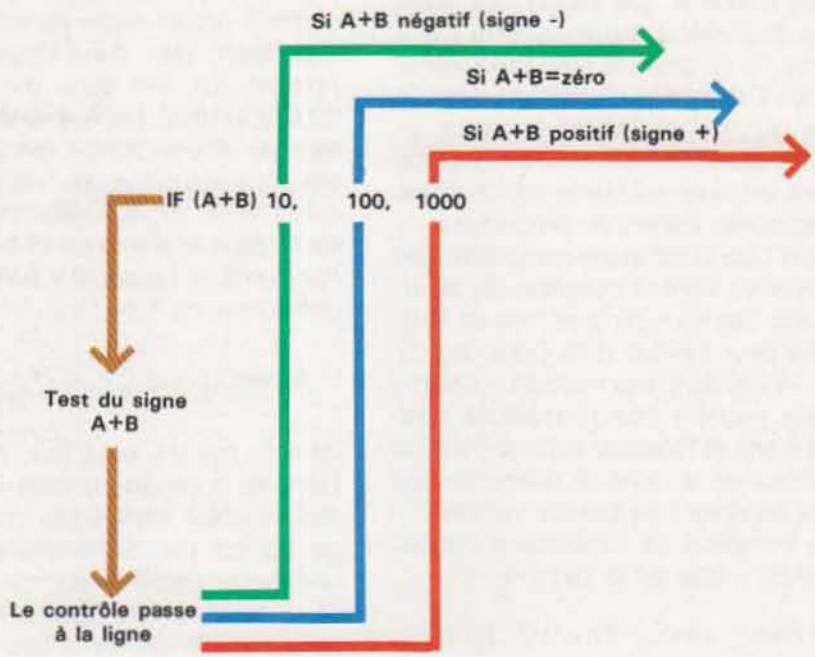
A la différence de l'arithmétique, l'IF logique a les mêmes fonctions que son homologue Basic.
Sa syntaxe est :

IF (condition) instruction

« Instruction » est exécutée seulement si « condition » est vraie. Mais en Fortran, il n'y a pas de THEN. Alors, l'instruction Basic :

```
IF A > B THEN K=1
```

DIAGRAMME SYNTAXIQUE DE UN IF ARITHMETIQUE



devient

```
IF (A.GT.B) K=1
```

« L'instruction » prend la forme d'un type quelconque, à l'exception de DO,END ou d'un autre IF logique.
L'IF par blocs est une extension de l'IF logique : il active des blocs entiers d'instructions au lieu d'une seule. La syntaxe est :

```
IF (expression) THEN
. . . . . .
(bloc d'instructions à exécuter si l'expression est vraie)
. . . . . .
END IF (ou ENDIF)
```

Dans l'IF à bloc apparaît (comme en Basic) le mot THEN et on peut employer la précision ELSE (sinon).
Par exemple, la séquence d'instructions :

```
IF (A.RQ.B) THEN
C=3*K
D=C+2
ELSE IF (A.G.T.B) THEN
C=0
D=1
ENDIF
```

attribue les valeurs C=3*K et D=C+2 à la condition que A=B, alors que pour A B, elle attribue C=0 et D=1. Le nombre de précisions ELSE dans un même IF est à la discrétion des programmeurs. On dispose donc d'un moyen de sélection bien plus puissant, même s'il n'est pas simple à appliquer.
Le langage Fortran, surtout dans sa version ANSI 77, est bien plus structuré que le Basic mais demande une période d'adaptation de la part du programmeur. Le programme est plus complet et souvent plus aisé à modifier.
Le thème de la programmation structurée sera abordé plus loin, à l'occasion de l'exposé sur le langage Pascal qui a justement été conçu pour mieux programmer.
L'emploi de la clause ELSE, à l'intérieur d'un IF, est également prévu dans certaines formes de Basic mais, sauf cas particulier, il n'est pas aussi simple qu'en Fortran.

L'emploi des sous-programmes (ou routines)

On retrouve, en Fortran, la notion de sous-programmes même si, par rapport au Basic standard, ils découlent d'une philosophie tout à fait différente. Si, en Basic, le sous-programme fait partie de l'ensemble et partage avec le programme principal toutes les variables, en Fortran, en revanche, chaque sous-programme construit une entité en soi qui ne communique pas avec les autres parties du programme.
Dans certains cas, la structure particulière des sous-programmes Fortran constitue un atout. Le découpage des sous-programmes en entités distinctes peut s'avérer utile. Dans un programme très long, donc très morcelé, les noms des variables peuvent être réemployés avec des significations différentes, soulageant ainsi le programmeur de la tâche de mémoriser les noms et les conditions de chaque variable.
La syntaxe complète de l'instruction d'appel d'un sous-programme est la suivante :

CALL nom (var-1, var-2,... *n1,*n2...)

où « nom » est le nom symbolique du sous-programme (point d'entrée équivalent au n° de ligne en Basic) ; « var-1, var-2... » les variables éventuelles à transférer au sous-programme appelant ; « n1, n2,... » les points de retour alternatifs.
Les versions Fortran les plus complètes prévoient, en effet, la possibilité d'effectuer un saut de la routine vers un point du programme suivant l'instruction appelante.
Par exemple, l'instruction :

CALL ESSAI (*100,*200,*300)

active le processus de retour paramétré. Selon la sortie sélectionnée dans le sous-programme ESSAI, le contrôle est transféré à l'une des trois lignes indiquées (100,200,300) du programme principal. Naturellement, le paramètre sera mentionné dans les 3 instructions RETURN insérées dans la routine.
Dans le programme ESSAI, au moins trois retours seront prévus : RETURN 1 pour la ligne 100, RETURN 2 pour la ligne 200, RETURN 3 pour la ligne 300.
Ce type de RETURN est paramétrable. Ainsi, en écrivant RETURN K, on déclenche le retour à 100, 200, 300 selon la valeur de K, qui peut être calculée dans le même sous-programme. Si elle n'est pas comprise dans les limites imposées, le contrôle reviendra à la ligne consécutive à l'appel (cette caractéristique n'existe cependant pas dans toutes les versions Fortran). La 1ère ligne du sous-programme (SUBROUTINE...) contient nécessairement autant de références de retour qu'il existe de solutions alternatives en fonction du résultat du calcul lancé (logique déterminée par la valeur de K, dans le graphique ci-contre).
Par exemple, l'appel précédent exige une première ligne du type :

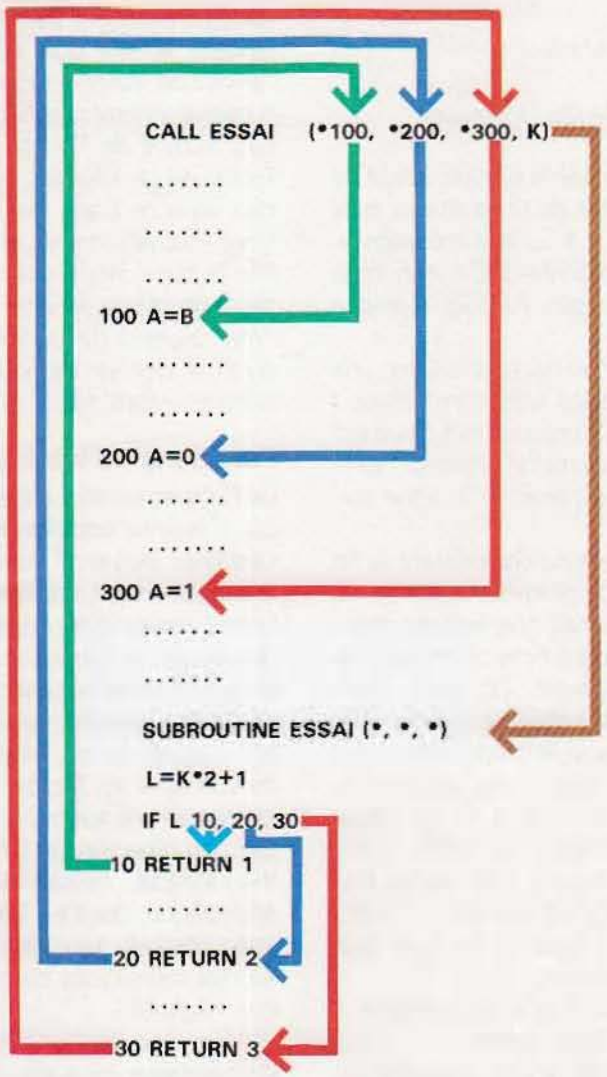
SUBROUTINE ESSAI (*,*,*)

dans la mesure où il faut prévoir 3 retours.
Dans les sous-programmes Fortran, les noms des variables transférées comme paramètres ne doivent pas nécessairement coïncider. Si une routine calcule la somme de deux nombres (A + B) et qu'elle place le résultat dans une troisième (C), elle devra être appelée en citant trois paramètres : les deux nombres à additionner et le résultat. Sa première ligne sera du genre SUBROUTINE SOMME (X, Y, R) en définissant X et Y comme nombres à additionner et R comme résultat.
Dans l'appel, on peut utiliser d'autres noms, à la seule condition de respecter le même ordre : les deux premiers pour les termes de la somme et le troisième pour le résultat.
Dans le programme :

```
      PROGRAM ESSAI
      A = 1
      B = 2
   10 CALL SOMME (A,B,S)
       .......
       .......
      X = 3
      Y = 5
   20 CALL SOMME (X,Y,T)
       .......
      SUBROUTINE SOMME (X,Y,R)
      R = X + Y
      RETURN
```

La ligne 10 (la numérotation 10, 20 n'a été adoptée que pour faciliter la lecture) appelle la

DIAGRAMME SYNTAXIQUE DES RETOURS ALTERNATIFS



routine SOMME avec les paramètres A, B, S ; au retour, la somme est contenue dans S. La ligne 20 appelle la même routine avec les paramètres X, Y, T et, au retour, la somme est contenue dans T.

En d'autres termes, la signification de chaque paramètre dépend essentiellement de sa position.

Dans la routine, on exécute le calcul R = X + Y, donc la calcul du 3e paramètre, de même que la somme des 2 autres. Quel que soit le nom symbolique des paramètres, au retour du sous-programme la 3e variable contient toujours la somme des deux premières.

Instructions de fin et d'arrêt de l'exécution

Dans ce paragraphe, nous allons examiner trois instructions : une sans équivalent Basic, les deux autres avec un sens différent :

PAUSE
STOP
END

PAUSE suspend momentanément l'exécution du programme pour, par exemple, donner le temps à l'opérateur de commander certaines opérations externes, comme de placer le papier sur l'imprimante.
La syntaxe de l'instruction est :

PAUSE n ou PAUSE 'message'

Dans les deux cas, le système signale, avant de suspendre le déroulement du programme, qu'il a rencontré la pause « n » ou le « message ». L'exécution de l'instruction PAUSE « Aligner le papier » génère le message « PAUSE Aligner le papier » à l'écran.
Le programme interrompu est réactivé par une instruction adéquate codée selon la machine ; la plus fréquente est GO. Lorsque cet ordre est introduit au clavier, le système reprend l'exécution du programme à partir de la ligne suivant PAUSE.
L'instruction **END** signale au compilateur la fin de chaque « unité de programme ».
Le Fortran est plus structuré que le Basic standard et chaque sous-programme constitue une entité compilée séparément. On peut donc rencontrer plusieurs END dans le même programme. Un seul concerne le programme principal et indique la fin logique du traitement, alors que les autres signalent la fin de sous-programmes. Normalement, le code END, dans une routine, a le même effet qu'un RETURN mais le contraire n'est pas vrai : si END est omis à la fin d'une routine, on aura une erreur lors de la compilation.
Le **STOP** produit le même effet qu'en Basic : il arrête l'exécution du programme.
En Fortran, on peut l'associer à un message ou à une valeur numérique ; l'un ou l'autre apparaissent sur l'écran lors d'un STOP.
Remarquons que PAUSE et STOP peuvent contenir un message mais doivent utiliser au moins une valeur numérique ; en général, on n'admet pas l'usage de ces instructions sans d'autres éléments.

Les fonctions du Fortran

En Fortran comme en Basic, il existe un ensemble de fonctions particulières (appelé « bibliothèque ») et des fonctions définies par l'utilisateur.

La bibliothèque du Fortran est plus riche en fonctions mathématiques que le Basic. inversement, elle dispose de très peu de fonctions pour le traitement des chaînes. Cette spécialisation s'explique aisément par l'origine délibérément scientifique du Fortran. Cependant, l'évolution parallèle des deux langages les a notablement rapprochés, à tel point que certaines formes de Fortran disposent des mêmes fonctions de chaînes que le Basic, tandis que des versions Basic possèdent toutes les fonctions mathématiques du Fortran.
Par la suite, nous examinerons quelques-unes des principales fonctions du Fortran. Leur variété dépend de la version du compilateur ; aussi ne donnerons-nous que la liste des fonctions standard ANSI 77.

Fonctions de bibliothèque

Le Fortran est très riche en fonctions de calcul car c'était sa vocation première. Les dernières versions incluent aussi certaines fonctions d'enchaînement qui rendent le Fortran apte à toutes sortes d'applications. Par la suite, nous diviserons les fonctions de la version ANSI 77 en groupes homogènes. Cette subdivision n'a pas, à proprement parler, la valeur d'une véritable classification ; elle ne constitue qu'un moyen pour en faciliter la lecture et certaines comparaisons avec Basic.
La compréhension et l'emploi de fonctions mathématiques nécessitant une bonne base scientifique, nous ne les évoquerons que pour être complets. Leur assimilation ne fait pas partie des techniques de programmation et peut être négligée.

Conversion de type. A ce groupe appartiennent les fonctions qui permettent de changer le type d'une variable. Par la suite, on opèrera une subdivision selon le type de variable obtenue.

Les fonctions de **conversion en entier** transforment en entier la valeur de variables réelles ou en double précision, selon la syntaxe :

```
I = INT (R)
I = IFIX (R)
```

(R = Réel, E = entier soit INTEGER en anglais)

Pour transformer en entier une valeur en dou-



B. Coleman/Marka

Micro-ordinateur à base de microprocesseur Intel capable d'exécuter les ordres oralement.

ble précision, il faut écrire :

I = IDINT (B)

A noter que les fonctions équivalentes en Basic sont INT (ou CINT) et FIX.

Les fonctions de **conversion en réel** (simple précision) ont comme syntaxe :

d'entier vers réel R = FLOAT (I)
de double précision vers réel R = SNGL (D)
(R = Réel, I = entier, D = Double précision)

L'équivalent Basic, est CSNG.

La conversion en **double précision** a une syntaxe unique, quel qu'en soit le type :

D = DBLE (X) (X = entier, réel, complexe)

L'équivalent Basic, sauf pour le X complexe, est CDBL (X).

La conversion caractère-nombre (et inversement nombre-caractère) s'obtient avec la syntaxe suivante :

de caractère vers entier I = ICHAR (C)
d'entier vers caractère C = CHAR (I)
(I = entier, C = caractère)

La valeur de E est celle qui est associée au caractère dans le code ASCII.
Les fonctions équivalentes en Basic sont ASC et CHR$.

Pour la **conversion d'un réel en complexe,** la fonction à utiliser (sans équivalent Basic) est CMPLX (R). Elle génère un nombre complexe, avec une partie réelle égale à R et une partie imaginaire égale à zéro.

Fonctions de calcul. Elles sont destinées à certains calculs arithmétiques immédiats.

Pour le **troncage dans la partie entière** de la valeur d'une variable, on utilise les fonctions :

A=AINT(B) A et B réels en simple précision

C=DINT(D) C et D en double précision.

Pour calculer le **le plus grand entier** contenu dans la valeur d'une variable, on utilise :

A=ANINT(B) A et B réels en simple précision
C=DNINT(D) C et D en double précision
I=NINT(B) I entier, B réel en simple précision
I=IDNINT(D) I entier, D en double précision

La **valeur absolue** d'une variable (sans le signe), ou la partie réelle d'un nombre complexe, s'obtient avec les fonctions :

I=IABS(K) I et K entier
A=ABS(B) A et B réels en simple précision
C=DABS(D) C et D en double précision
C=CABS(Z) C réel, Z complexe

La fonction de **transfert de signe** travaille sur deux nombres (A et B) et restitue (-A) si B inférieur à zéro, (A) si B supérieur ou égal à zéro. Les nombres sont nécessairement du même type, avec une syntaxe différente, suivant le type :

ISIGN(A,B) entiers
SIGN(A,B) réels en simple précision
DSIGN(A,B) nombres en double précision

La fonction **différence positive** exige également deux nombres (A, B) et restitue (A-B) si A supérieur à B et 0 si A inférieur ou égal à B :

IDIM(A,B) entiers
DIM(A,B) réels en simple précision
DDIM(A,B) nombres en double précision

La fonction **produit en double précision** calcule le produit de deux nombres réels en tranformant le résultat en double précision :

D=DPROD(A,B) A,B réels en simple précision
D en double précision

La fonction **reste** effectue le calcul :

A-INT(A/B)*B

En donnant ainsi le reste de la division entre A et B.
Son équivalent Basic présente quelques différences de noms :

MOD pour les entiers
AMOD pour les réels
DMOD pour les nombres en double précision

Les fonctions **maximum** et **minimum** permettent d'extraire, respectivement, la valeur maximum et la valeur minimum d'une série de nombres :

R=MAX 0 (A,B,C...) entiers
R=AMAX (A,B,C...) réels en simple précision
R=DMAX (A,B,C...) nombres en double précision
R=MIN 0 (A,B,C...) entiers
R=AMIN (A,B,C...) réels en simple précision
R=DMIN (A,B,C...) nombres en double précision

D'autres formes permettent un usage de type mixte.

Fonctions de chaîne. Le Fortran ANSI 77 ne comporte que deux fonctions pour le traitement des chaînes :

LEN fournit la longueur (en caractères) d'une chaîne
INDEX restitue la position initiale d'une sous-chaîne à l'intérieur d'une chaîne

Fonctions lexicales. Elles permettent de comparer deux chaînes (en suivant l'ordre l'alphabétique) avec un résultat « vrai » ou « faux » selon que la condition de comparaison est vérifiée ou non.
Les fonctions prévues sont :

C=LGE (A,B) vérifie si A = B
C=LGT (A,B) vérifie si A B
C=LLE (A,B) vérifie si A = B
C=LLT (A,B) vérifie si A B

En retour, on aura, dans C, la valeur logique TRUE et FALSE.

Fonctions mathématiques. Pour des raisons de clarté, nous ne présentons, dans ce

paragraphe, que celles qui se rapportent aux réels. Pour la double précision, le plus souvent, il suffit de remplacer la première lettre de la fonction par D. Mais il ne s'agit pas d'une règle générale : pour connaître la syntaxe exacte de la fonction, il suffit de se rapporter au manuel d'utilisation de votre version Fortran.

Description	Fonction	Exemple
Racine carrée	SQRT	B = SQRT(R)
Expotentielle	EXP	B = EXP(R)
Logarithmes	LOG	B = LOG(R)
Logarithmes décimaux	LOG 10	B = LOG10(R)
Sinus	SIN	B = SIN(A)
Cosinus	COS	B = COS(A)
Tangente	TAN	B = TAN(A)
Arcsinus	ASIN	B = ASIN(R)
Arcosinus	ACOS	B = ACOS(R)
Arctangente	ATAN	B = ATAN(R)

Les fonctions trigonométriques utilisent des angles (dans les exemples identifiés par la variable A) exprimés en radians. En Fortran, on prévoit, en outre, les fonctions hyperboliques, dont les noms s'obtiennent en ajoutant la lettre H au terme trigonométrique correspondant (par exemple SINH indique le sinus hyperbolique).

Fonctions programmables par l'utilisateur

Les fonctions définies par l'usager suivent, d'une certaine manière, des règles analogues à celles que nous avons vues en Basic. Reste la différence de fond liée au fait, qu'en Fortran, chaque module (sous-programme ou fonction) est logiquement séparé du reste du programme.
La syntaxe qui permet de définit une fonction particulière est la suivante :

```
type FUNCTION nom (liste des arguments)
```

type	Il définit le type de fonction (réelle, entière, etc.) et prend une des valeurs décrites dans les déclarations (REAL, INTEGER, etc.)
FUNCTION	Mot réservé indiquant la fonction.
nom	Nom attribué à la fonction ; contrairement au Basic, où il n'est composé que d'un caractère (DEF FN A), en Fortran on peut utiliser n'importe quel nom (comme pour les variables)
listes d'arguments	Comme en Basic, elle est composée des arguments que la fonction doit utiliser

Par exemple, la ligne :

```
INTEGER FUNCTION ESSAI ( )
```

définit la fonction entière ESSAI, sans arguments ; la ligne :

```
FUNCTION X (A,B,C)
```

définit la fonction X, qui utilise les arguments A,B,C ; et enfin la ligne :

```
CHARACTER* 4 (L)
```

définit une fonction de chaîne A, de 4 caractères (longueur) avec l'argument L.
En Fortran, les fonctions définies par l'usager présentent une structure semblable à celle d'un sous-programme ; elles peuvent, par exemple, comporter plusieurs lignes, en incluant plusieurs calculs, contrairement au Basic où une seule ligne est utilisée, avec les limites qui en résultent. Une fonction Fortran commence avec la déclaration FUNCTION et se poursuit sur un nombre quelconque de lignes pour se terminer avec RETURN.
Ainsi, la fonction :

```
FUNCTION HYPO (A,B)
HYPO-SQRT((A**2) + (B**2))
RETURN
END
```

calcule l'hypoténuse d'un triangle rectangle de côtés A et B. Pour l'utiliser, il suffit de l'appeler avec le nom symbolique (HYPO).

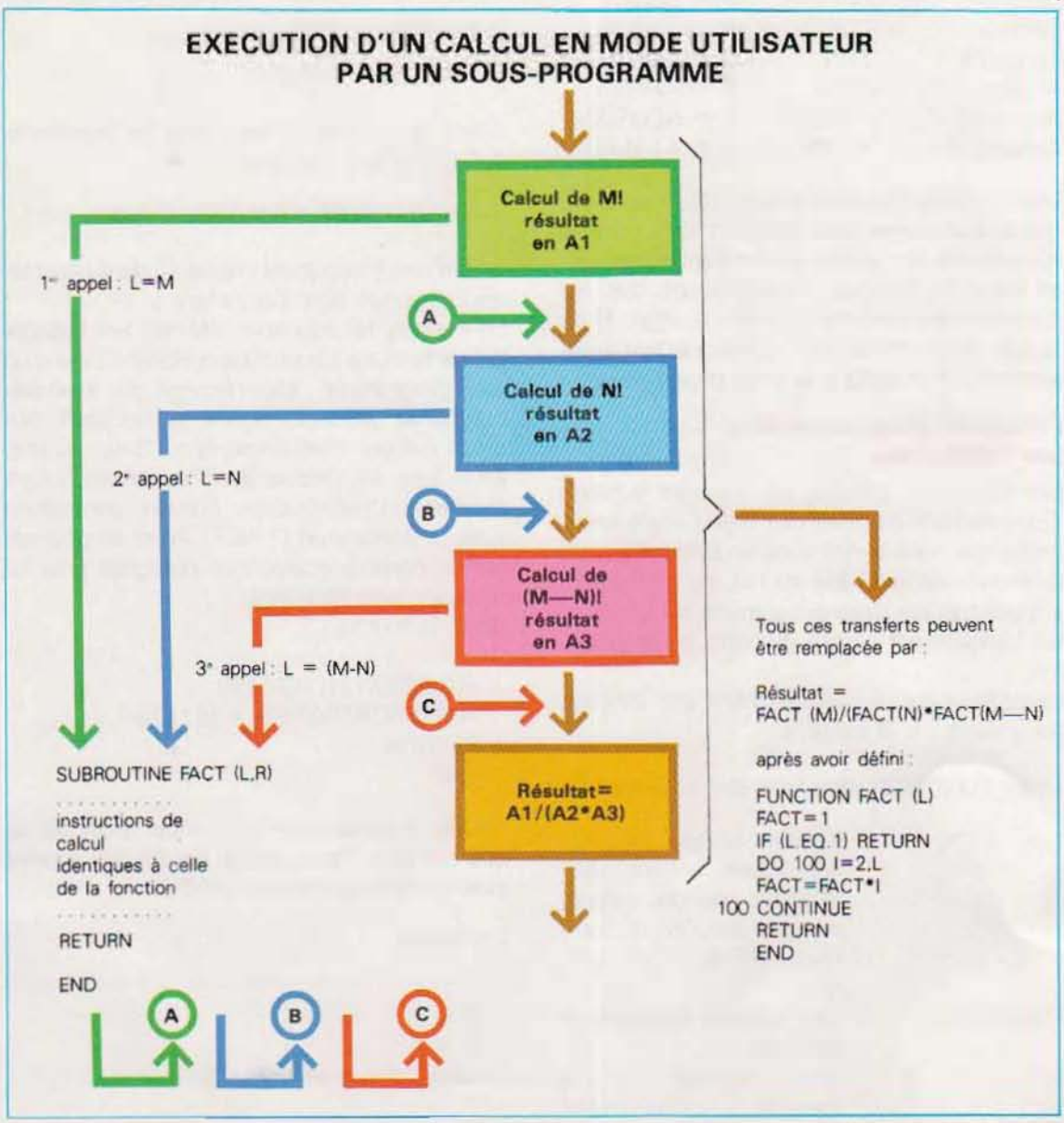
Les lignes :

```
. . . .
A = 3
B = 4
HYPOTEN = HYPO (A,B)
. . . .
```

appellent la fonction précédemment définie et transfèrent le résultat (hypoténuse) dans la variable HYPOTEN.
On aurait pu obtenir le même résultat en définissant HYPO comme sous-programme (dans ce cas il faudrait mentionner 3 paramètres : les côtés, A,B comme paramètres d'entrée et HYPOTEN comme paramètre de sortie).
Le graphique ci-dessous compare une fonction programmée par l'utilisateur à un sous-programme réalisant le même calcul.
Le programme en question (présenté sans le début et sans la fin) calcule les combinaisons de M éléments de classe N (c'est-à-dire pris par groupes de N). La formule est :

$$\text{Combinaisons} = \frac{M!}{N! \times (M-N)!}$$

où M ! (factorielle M) désigne le produit de tous les nombres entiers depuis 1 jusqu'à M (par exemple pour M × 5, on a M ! 1X2X3X4X5).
Si, pour calculer les 3 factorielles, n recourt à un sous-programme, ce dernier devra être appelé trois fois, en mettant en mémoire le résultat de chaque calcul (voir organigramme).
Le sous-programme doit posséder deux para-



paramètres : L (nombre dont on veut calculer la factorielle) et R (résultat du calcul). Au premier appel, on calcul M ! (rangé en mémoire, par exemple en A1), au deuxième, N ! (rangé en A2) et au troisième (M-N) ! (rangé en A3). La dernière étape est le calcul des combinaisons : [A1 / (A2*A3)].
En remplaçant le sous-programme par une fonction, on réduit le tout à une seule expression. Le déroulement de la fonction est identique à celui du sous-programme, mais on élimine les appels et les stockages en mémoire.
Analysons le mode d'appel.
Dans la première ligne de la fonction [FUNCTION FACT(L)] l'argument L apparaît. Comme pour les sous-programmes, son but est de signaler l'existence d'un paramètre. En remplaçant, lors de l'appel, le symbole L par le nom d'une variable quelconque ou par une valeur numérique, on lance le calcul correspondant.
Ainsi, pour calculer M !, il suffit d'appeler la fonction en appliquant M [FACT (M)] à N !.
Outre sa présentation plus complète, le Fortran fournit aussi des fonctions sur une seule ligne. Elles s'apparentent davantage à celles du Basic, mais ont pour limite de n'être utilisables que dans le contexte de leur définition.
Syntaxe de ce type de fonction :

nom (argument) = expression

où :

nom Symbolique de la fonction
arguments Paramètres utilisés dans la définition
expression Description du calcul à effectuer.

Par exemple, le calcul de l'hypoténuse d'un triangle rectangle peut s'effectuer avec :

```
HYPO (A,B) SQRT((A**2) + (B**2))
```

Il suffit, alors, de l'appeler comme pour les autres fonctions :

```
HYPOTEN = HYPO (A,B)
```

Les instructions et les formats d'E/S

En Fortran, la gestion des fonctions d'E/S est très différente de celle du Basic. Dans le Basic standard, il existe un seul périphérique d'entrée : le clavier (nous ne parlerons pas ici des fonctions E/S sur disque), et deux unités de sortie : l'écran et l'imprimante. A chaque unité correspond une instruction et il n'est donc pas nécessaire de préciser l'unité d'E/S concernée. S'il s'agit d'une instruction de sortie, le code PRINT l'adresse à l'écran, le code LPRINT à l'imprimante.

En Fortran, en revanche, il faut préciser l'unité concernée pour chaque opération d'E/S.
Cette différence provient des origines du langage. Le Fortran est né sur de gros ordinateurs qui commandent, en général, plusieurs imprimantes et plusieurs systèmes d'entrée (lecteurs de cartes et de bandes, claviers). Il a conservé ses caractéristiques au cours de son évolution.
Une deuxième différence importante concerne la possibilité, pour le Fortran, de définir un **format** pour les données, tant en entrée qu'en sortie.
Dans les fonctions d'impression du Basic, il n'existe pas une grande marge de manœuvre, sauf à recourir à l'option PRINT USING... qui, dans certaines limites, permet d'employer des formats d'impression définis par l'utilisateur.
Pour les instructions d'entrée, les limites sont encore plus grandes. Le langage n'accepte que des valeurs homogènes avec les variables, sans que l'on puisse définir de format. En fait, dans les micro-ordinateurs, il n'existe aucune raison de disposer de formats d'entrée. La saisie est réalisée uniquement au clavier et l'utilisateur ne tirerait aucun bénéfice de l'existence de formats particuliers. Au contraire, le Fortran prévoit une entrée à partir de divers périphériques, comme par exemple le lecteur de cartes perforées. Il est utile, dans ce cas, de pouvoir disposer d'instructions particulières de lecture qui permettent de reconnaître un champ, en fonction de sa position. Cette souplesse permet de perforer des cartes laissant des blancs entre une valeur et une autre, avec une lisibilité certainement supérieure.
En Fortran, chaque instruction d'E/S doit être complétée par deux paramètres : le code correspondant à l'unité physique d'E/S et un numéro de ligne qui définit le format utilisé en saisie ou en présentation de données.

L'unité doit être précisée dans la mesure où le Fortran ne dispose pas d'instruction différentes pour l'écran et pour l'imprimante. Le format de présentation ou de lecture des données peut être omis et on dira, dans ce cas, que l'instruction est **non formatée.** Les instructions d'E/S, prévues par le Fortran, sont :

READ	pour la lecture
PRINT ET WRITE	pour l'écriture

Les mots READ et WRITE sont employés aussi pour les fonctions d'E/S sur disque, mais cet aspect sera étudié ultérieurement dans la partie traitant de la gestion de fichiers. Le format général des instructions READ et WRITE est :

READ (N,M) liste de variables
WRITE (K,L) liste de variables

N et K indiquant les périphériques, respectivement d'entrée et de sortie : M et L sont les numéros de ligne où se trouve le format de lecture ou d'écriture.
L'instruction :

READ (5,270) A,B,C

signifie : « lire sur le périphérique 5 selon le format défini à la ligne 270 et transférer les valeurs dans les variables A,B,C ».
De même :

WRITE (6,300)A,B,C

signifie : « écrire les valeurs des variables A,B,C, sur le périphérique 6 selon le format défini à la ligne 300 ».

L'instruction PRINT constitue une extension de la version Fortran ANSI 77 qui ne rentre pas dans le cadre du format général du langage. Alors que les formats précédents (READ, WRITE) sont communs à toutes les versions, PRINT n'est pas prévu, par exemple, sur des systèmes plus grands utilisant des compilateurs anciens.
Observons que le n° attribué aux divers périphériques dépend de chaque machine. Normalement, le périphérique d'entrée est numéroté 5 (clavier ou lecteur de cartes) alors que celui de sortie (imprimante) est le 6 ; mais rien n'interdit d'avoir d'autres assignations.
Nous exposerons par la suite les fonctions d'E/S par rapport à leur développement sur les micro-ordinateurs. On laissera donc de côté les options pour le lecteur de cartes qui constitue une unité caractéristique des gros systèmes ; l'idée est cependant simple : il suffit de modifier le code numérique de l'unité.

PRINT

C'est l'instruction de sortie pour le périphérique standard (écran ou imprimante selon la configuration du système). Syntaxe :

PRINT N, données

N	N° de ligne de format de sortie correspondant
données	Valeurs à écrire. Il peut s'agir de variables de constantes, de calculs ou de chaînes.

L'indication du n° de ligne du format peut être omise et les valeurs seront alors écrites selon les indications contenues dans l'instruction elle-même. Examinons quelques exemples :

PRINT 10,A,B,A+B

écrit, selon le format défini à la ligne 10. Les variables A,B et leur somme :

PRINT *, 'somme=', A+B

n'ont pas de format (symbole *). La sortie est alors une chaîne « Somme= » suivie de la valeur A+B.

ASSIGN 10 TO F
PRINT F,A,B,A+B

sont une forme différente du premier exemple. A la ligne 10, où le format est défini, on a attribué le nom symbolique F(ASSIGN 10 TO F). Grâce à ce nom, la fonction peut être appelée à n'importe quel point du programme.

WRITE

L'instruction WRITE est utilisée pour transférer des données soit sur un périphérique de sortie soit sur un disque.
Format général :
WRITE (unité, format, statut, étiquette, enregistrement) données...

où

unité	Unité de sortie
format	Numéro de ligne décrivant le format. Parfois, il peut s'agir d'une définition de format entre guillemets
statut	Message d'erreur ; différent de zéro, le code renvoie à une erreur de programmation en phase d'écriture (les valeurs ne sont pas standard, chaque système disposant de son propre tableau)
étiquette	Numéro de ligne où le contrôle est transféré si erreur
enregistrement	Numéro d'enregistrement pour l'écriture sur des fichiers à accès direct
données	Liste de variables, de constantes,... à écrire

Syntaxiquement, chaque paramètre est précédé d'un mot-clé qui l'identifie :

UNIT	Unité de sortie. Exemple : UNIT=6 sélectionne l'unité 6
FMT	Format. FMT=100 indique le format décrit à la ligne 100
IOSTAT	Assigne la variable de statut. Si IOSTAT=K, au retour le code d'une erreur sera contenu dans la variable K
ERR	Ligne de contrôle en cas d'erreur, ERR=500 transfère le contrôle à la ligne 500
REC	Numéro d'enregistrement. Si REC=5, l'instruction est écrite sur l'enregistrement 5 du fichier (UNIT) n° 6

Si les deux premiers mots-clé (UNIT et FMT) peuvent être omis, les autres sont obligatoires quand les paramètres correspondants existent. Voyons les exemples suivants :

WRITE (3,10) A,B	Ecrit sur l'unité 3 selon le format 10 ; il n'y a pas d'autres paramètres et donc pas de mots-clé.
WRITE (3,10,ERR=100)	Ecrit sur la 3 dans le format 10 ; en cas

Perforatrice de ruban Facit modèle 4070 capable d'imprimer jusqu'à 75 caractères par seconde.



Facit

d'erreur, le contrôle est transféré à la ligne 100.

WRITE (3,10,IOSTAT=I,ERR=100,REC=5)A
Ecrit sur la 3 dans le format 10. La variable I contient l'éventuel code d'erreur et, dans ce cas, le contrôle passe à la ligne 100. Les données (contenu de la variable A) sont écrites dans l'enregistrement 5.

READ

L'instruction READ s'écrit de deux façons selon qu'elle adresse l'unité d'entrée standard ou un fichier. Dans le premier cas, la syntaxe est :

READ M, liste

où M est le numéro de ligne contenant les spécifications de format ; « liste » est une liste de variables où seront mémorisées les valeurs lues. Par exemple :

READ 10,A,B	lit les variables A et B selon le format défini à la ligne 10
READ*,(V(I),I=1,5)	lit, comme un DO implicite, la variable dimensionnée V(I), sans format
READ 20,(V(I),I=1,5)	même définition (DO implicite) avec format de lecture défini à la ligne 20

L'accès aux fichiers avec l'instruction READ fait appel aux mêmes paramètres que WRITE. Le format général se résume ainsi :

READ (unité, format, statut, étiquette, enregistrement) données

L'instruction :

READ (2,1,IOSTAT=KR,ERR=2,REC=6)A,B
lit les variables A et B sur l'enregistrement 6 du fichier 2 selon le format défini à la ligne 1. Le code d'erreur est dans KR, et le cas échéant, le contrôle est transféré à la ligne 2.
Cette instruction comporte en outre un nouveau paramètre qui provoque un saut en cas de fin de fichier EOF (End Of File) sur les fichiers séquentiels. Le paramètre est END=n° ligne.
Par exemple, la ligne :

READ (3,150,END=1000,REC=J)V

lit l'enregistrement J du fichier séquentiel n° 3 selon le format 150 ; en cas d'EOF, le contrôle passe à l'instruction : 1000 (END=1000).

En Fortran, ces opérations sont gérées par l'instruction READ avec l'option END=...
En Basic, par contre, il faut tester EOF pour savoir si le fichier est fini et on peut donc décider, avec un IF, de poursuivre ou non la lecture.

Formats d'E/S

Les fonctions d'E/S du Fortran peuvent faire référence à un numéro de ligne où seront données les spécifications de lecture ou d'écriture des données. Cette ligne particulière (FORMAT codes) contient le format de saisie ou de restitution. Normalement, ce type d'instruction n'est pas vérifié par le compilateur ; les erreurs de format n'apparaissent donc qu'en phase d'exécution du programme.
Voici les spécifications de traitement des variables :

Entiers	I	(INTEGER)
Réels	F	en virgule (fixed-point)
	E,D	en virgule flottante (floating-point)
	G	en virgule fixe et flottante
Caractères	A,R	
Logiques	L	
Octals	K,O	

Formats pour les entiers. La lettre d'identification (I) est suivie du nombre de chiffres qui composent la valeur à lire ou à écrire ; 13, par exemple, indique une valeur entière de 3 chiffres. Le code peut aussi être précédé d'une autre valeur numérique qui a le sens d'un **facteur de répétition** et qui indique le nombre de fois à appliquer le format défini.
Voyons quelques exemples :

READ (6,10)	Lecture de 2 valeurs dans le format 10
WRITE (5,20)I,K,L	Ecriture de 3 valeurs dans le format 20
10 FORMAT (12,15)	Peut être une erreur, car A et B ne sont pas entiers par défaut
20 FORMAT (312)	Dans ce cas, le format entier est valable ; les variables I,K,L sont entières, il faut seulement vérifier que chacune d'elles se compose de 2 chiffres (312=3 fois 12)

Le format peut s'intégrer dans la ligne qui exprime la fonction ; ainsi :

WRITE (5,'(312)') I,K,L

a la même signification que la ligne précédente. Dans certaines versions de Fortran, on prévoit l'extension :

In.m

où :
n indique le nombre maximum de chiffres, m le minimum (en OUTPUT seulement car en instruction d'entrée, m est ignoré). Exemple : avec le format 15.3, la variable 4 s'écrit :

bb004

soit avec 2 espaces pour remplir le champ maximum prévu (longueur 5) et 3 caractères apparents. Puisque la valeur numérique ne comporte qu'un chiffre (4), les chiffres manquants, dans la définition du champ minimum (longueur 3), seront remplacés par des zéros.

Formats pour les réels. La lettre F est obligatoirement suivie de deux valeurs numériques, la première pour la longueur totale du champ en caractères (point décimal compris) et la seconde pour le nombre de chiffres décimaux après la virgule).
Les lettres E et D sont employées pour les représentations en virgule flottante (voir bas de page).
Le code E réserve 2 positions à l'exposant, le code D en réserve 3. La longueur du champ (10 dans le second exemple) comprend 3 positions pour l'exposant (option D) et 1 pour le point décimal : un champ de longueur 7 en format D ne dispose donc que de 3 positions utiles pour la valeur.
Le format E fonctionne exactement comme le format D ; la seule différence est dans l'exposant, auquel on ne réserve que 2 chiffres. Le format G, s'adaptant à la longueur de la valeur, est utilisable pour représenter des valeurs en virgule flottante ou en virgule fixe.

SPECIFICATIONS DE FORMAT POUR LES NOMBRES REELS EN VIRGULE FIXE

Valeur en mémoire	Identification	Sortie
9.7584	F 4.2	9.76
9.7584	F 6.4	9.7584
15	F 4.1	15.0
128.2	F 3.1	Erreur : la valeur n'entre pas dans le champ

SPECIFICATIONS DE FORMAT POUR LES NOMBRES REELS EN VIRGULE FLOTTANTE

Valeur en mémoire	Identification	Sortie
21.412	D 7.4	Erreur : le champ d'une longueur 7 ne peut contenir la donnée
21.412	D 10.4	2.1412+01

Format pour les caractères. C'est la lettre A qui spécifie le format pour l'E/S des caractères prévus. Par exemple, A3 définit une chaîne de 3 caractères.
Si les caractères composant une chaîne sont plus nombreux que prévus par la description, on ne prend en compte que ceux décrits dans le format, en partant de la gauche.
Soit la chaîne ABCDEFG : en sortie, le format A4 générera l'impression de ABCD. De même, ce format employé en lecture n'en transfèrera que les 4 premiers caractères.
Le format R (extension du Fortran ANSI 77) s'apparente au précédent mais avec un cadrage à droite dans le champ d'E/S.

Formats pour les constantes logiques. Une constante logique (TRUE/FALSE) est indiquée par L, suivi d'un nombre représentant la longueur du champ à examiner. Celui-ci doit contenir, comme premier caractère (sauf l'espace), la lettre T (TRUE) pour VRAI ou F (FALSE) pour FAUX.
Par exemple, en entrée, le format L3 examine un champ de 3 caractères et assigne à la variable la valeur TRUE s'il trouve T, et FALSE s'il trouve F. En introduisant une des chaînes :

T,Tx, Txy

le résultat dans la variable est toujours T (VRAIE) ; les lettres qui suivent le code T ne sont pas prises en considération.
De la même manière, les chaînes F, FZ, FKL transfèrent dans la variable la valeur F (FAUX).
En sortie, on obtient la lettre T ou F, précédée d'autant d'espaces que nécessaire pour compléter le champ défini dans le format.
Par exemple, si la variable contient TRUE, le format L4 génère ␣ ␣ ␣ T, alors que le format L1 fournit T.

Descripteurs de mise en page

La gestion complète des fonctions d'E/S nécessite l'emploi de descripteurs spéciaux appelés descripteurs de mise en page. Ces éléments sont employés pour insérer des commentaires, des espaces ou des zéros dans les champs de données et ils se subdivisent en trois catégories :

- descripteurs par fonctions d'entrée
- descripteurs par fonctions de sortie
- descripteurs par fonctions d'entrée et de sortie.

Mise en page en entrée. Il s'agit de BN et de BZ. Avec les descripteurs numériques déjà décrits, les éventuels espaces (blancs) entre les valeurs numériques introduites étaient ignorés ; avec BZ, ils sont convertis en zéros.
Exemple : en lisant la valeur 3␣ 7␣ 5 avec le format 15 on obtient le nombre 375 et les espaces entre les chiffres (␣) seront ignorés. Si on utilise le format BZ, on a 30705 : les espaces sont transformés en zéros.

Mise en page en sortie. Ces descripteurs ont essentiellement deux types de fonctions :

- impression de chaînes de caractères, commentaires, descriptions.
- présentation du signe algébrique.

L'impression de chaînes s'obtient de deux manières : la première consiste à mettre entre guillemets les caractères à écrire, la seconde à les décrire comme des constantes de Hollerith (nH). Par exemple, les formats :

14, 'Total =', 13
14,9 HTotal = , 13

génèrent l'impression d'une valeur numérique (14), suivie du texte Total = et d'une autre valeur numérique (13).
Normalement, le signe + n'est pas explicité, contrairement à ce qui se produit avec les valeurs négatives (signe - toujours indiqué). SP provoque l'impression du signe +.

Descripteurs communs. Les principaux descripteurs employés, aussi bien en entrée qu'en sortie, sont :

nX : saute n positions
Tn : tabule en colonne n
/ : indique la fin d'un enregistrement

Facteur d'échelle. Il est représenté par nP, où n représente un facteur multiplicateur, P étant l'indicateur positif ou négatif. Le facteur d'échelle ne s'applique qu'aux valeurs numériques. Par exemple, avec le format F10.4 la

valeur 135.79817 est écrite comme 135.7981 (4 décimales). En appliquant un facteur d'échelle, par exemple - 2, le format devient : - 2PF10.4 et l'impression 1.357981.

Répétitions de format. N'importe quel format peut être répété autant de fois qu'on le désire et le nombre de répétitions doit précéder le descripteur. Exemple : la spécification 3F5.2 répète 3 fois le format F5.2. Le facteur de répétition s'applique aussi à des formats qui contiennent plusieurs descripteurs ; on utilise alors des parenthèses.
Par exemple, la spécification :

(312,2(14,X,F8.2))

décrit la sortie suivante :

312 Trois valeurs entières de 2 chiffres
2( ) Répétition d'un groupe composé de :
- 14 Une valeur entière de 4 chiffres
- X Un espace
- F8.2 Une valeur réelle avec un champ de 8 caractères et de 2 décimales.

Remarquons que les trois valeurs entières (312) sont écrites de manière contiguë ce qui ne permet pas de les distinguer. De même, les groupes (14,X,F8.2) seront écrits à la suite. Pour séparer les divers champs, on utilise le descripteur X. Le format précédent écrit :

(3(X,12),2(X,14,3X,F8.2))

insère un espace entre les trois premières variables entières (X,12) et un espace entre les groupes 14, etc.

Instructions d'E/S non formatées

Les instructions d'E/S sans formatage ont une syntaxe et une logique semblables aux commandes Basic correspondantes, pour l'écran, l'imprimante et le clavier alors qu'elles sont très différentes pour la gestion du disque.

Instructions d'entrée. En lecture, elles sont analogues à celles décrites pour les entrées avec format, avec en plus le symbole * :

READ *, variables

ou bien :

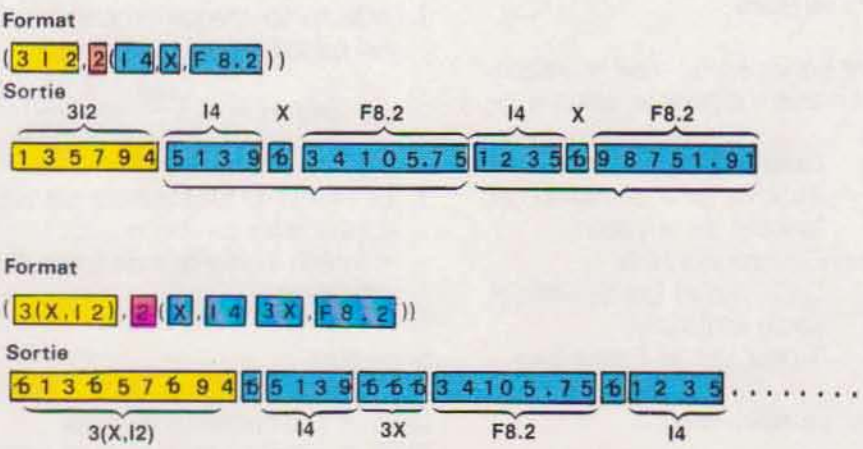
READ (U, *) variable

Dans la première, l'unité adressée (clavier) est implicite. Dans la seconde, c'est une unité d'entrée quelconque codée numérique par (U). En lecture, les valeurs, rangées en mémoire dans les variables, doivent être séparées par un ou plusieurs espaces, par le symbole / (slash) ou par une virgule. Exemple, avec l'instruction :

READ *,A,B,C

les valeurs A=3, B=5, C=11 seront entrées

EXEMPLES DE FORMATS ET SORTIES CORRESPONDANTES



par l'un des 3 formats :

```
3  5  11
3 / 5 / 11
3,5,11
```

Pour attribuer la même valeur numérique à toutes les variables on utilise un facteur de répétition. Soit la valeur 7 assignée aux variables A,B,C, on retrouve :

```
7  7  7
7 / 7 / 7
7,7,7
3*7 (signifie 3 fois la valeur 7)
```

Si besoin est, on peut spécifier l'exposant (avec un des symboles admis : E, D, etc). Par exemple, la donnée 586 se lit aussi : 5.86E2 (E2 signifie × 10^2).
Les espaces blancs ne sont pas pris en considération et la variable conserve sa valeur en fonction de l'instruction READ. Avec :

```
READ *, A, B, C
```

et les valeurs d'entrée :

```
b, 20, b
```

on transfère la valeur 20 dans la variable, B ; alors que A et C gardent leur valeur d'origine.

Instructions de sortie. L'instruction PRINT non formatée s'écrit :

```
PRINT *, variables
PRINT (U, *) variables
```

avec les mêmes significations, mais les valeurs sont écrites de manière différente, selon le type de variable :

Entiers	Comme des nombres entiers
Réels	Avec ou sans exposants, en fonction de la valeur
Double précision	Comme des réels
Complexes	Deux valeurs (partie réelle et partie imaginaire)
Logiques	T pour vrai et F pour faux

Soit les couples variables-valeur :

I=256	entier
R=25.72	réel
D=0.1173D2	double précision
C=(12,5)	complexe
L=.TRUE.	logique
CH='TEXTE'	caractères

Les sorties sont :

Instruction	**Sortie**
PRINT *,I,CH	256 TEXTE
PRINT *,C,L	(12.,5.)T
PRINT *,R,D	25.72 1.173D1

On peut aussi appeler l'instruction WRITE en précisant l'unité de sortie. Si on prend le code 6 pour le périphérique standard (6 est l'assignation la plus commune) les instructions précédentes deviennent :

```
WRITE (6,*)I,CH
WRITE (6,*)C,L
WRITE (6,*)R,D
```

avec le même résultat.

Exemple de programmation en Fortran

L'organigramme ci-contre exécute quelques instructions Fortran.
Le problème à résoudre est le suivant : étant donné un montant initial, investi avec une capitalisation mensuelle, effectuer le calcul mensuel de l'intérêt. Le taux varie selon les mois ; il faut donc prévoir 12 valeurs (une par mois).
Déroulement du calcul :

1 / A la fin de chaque période (mois) l'intérêt est calculé par :

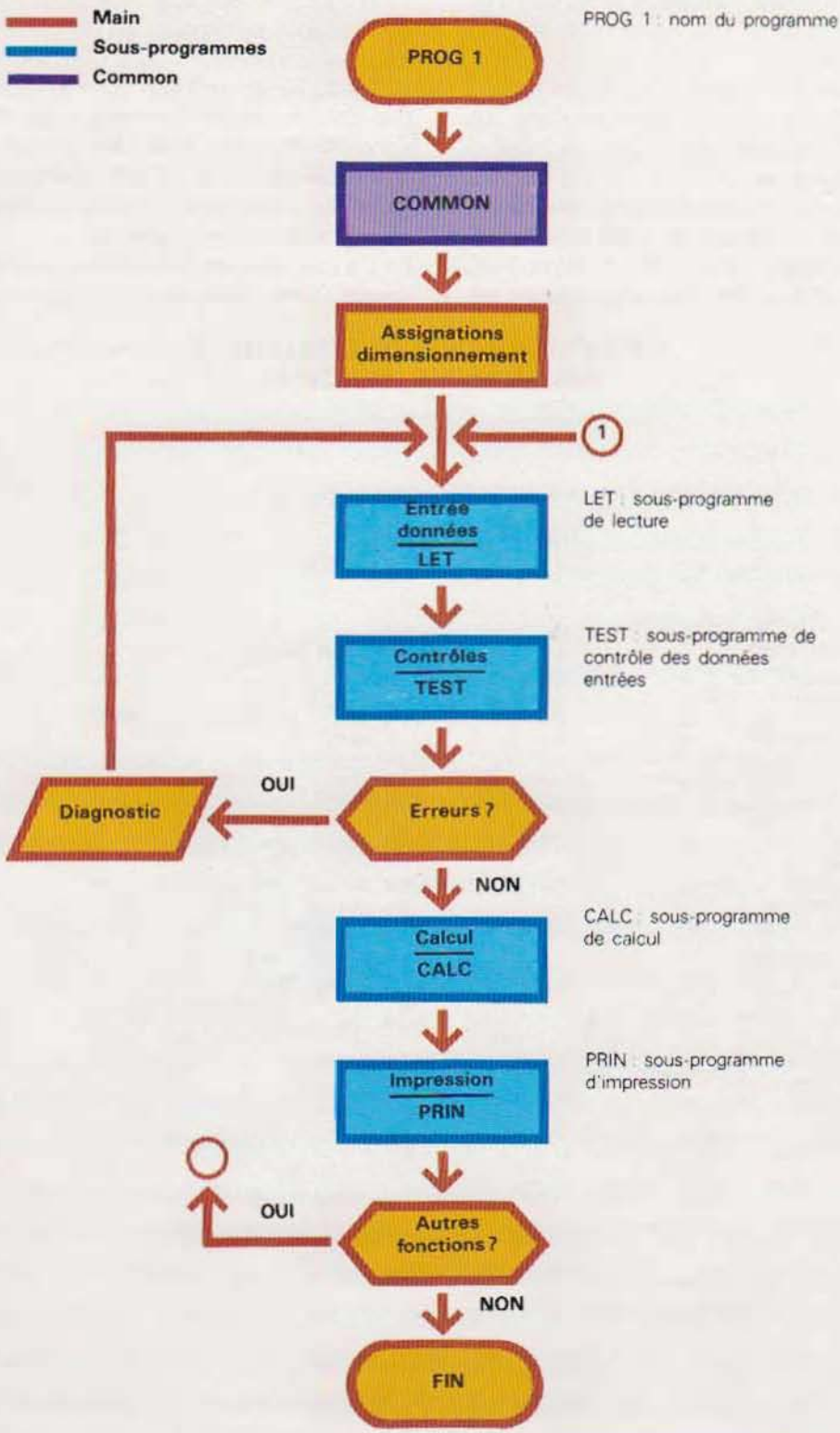
$$\text{Intérêt} = \frac{\text{Capital} \times \text{Taux (mensuel)}}{100}$$

2 / Le capital du mois suivant est calculé par :
Capital mois suivant =
= Intérêt à la fin du mois (point 1) + Capital précédent.

Si on pose :

C(12) = Capital début du mois
S(12) = Intérêt cumulé dans le mois
T(12) = Taux à appliquer pour chaque mois

ORGANIGRAMME POUR LE CALCUL DES INTERETS



pour un mois I, quelconque, on aura les expressions :

S(I)=(C(I)*T(I))/100 (Intérêt à la fin du mois)
C(I+1)=S(I)+C(I) (Capital mois suivant)

Pour effectuer le calcul sur les douze mois, il suffit d'appliquer les deux formules dans une boucle avec I variable entre 1 (premier mois) et 11 ; la dernière valeur (mois 12) est calculée séparément, car elle représente la valeur finale (total annuel) et n'a pas de mois suivant. On trouvera en pages 1185, 1187, 1188 et 1190 les organigrammes des sous-programmes appelés. Les programmes correspondants en Basic et en Fortran sont donnés page 1184, 1186, 1189 et 1191.

Le programme principal ci-dessous met en relief certaines différences fondamentales entre les deux langages. En particulier, les lignes 10 et 20 du Basic sont des commentaires, alors que dans la version Fortran il s'agit d'instructions P indispensables. La première ligne (correspondant à la 10) définit le nom du programme, la deuxième certaines variables communes aux sous-programmes.

En Fortran, ne sont numérotées que les lignes devant être adressées, en particulier les

EXEMPLE DU CALCUL D'INTERETS
PROGRAMME PRINCIPAL

Version Fortran

```
      PROGRAM PROG1
C     ************************
C     ** PROGRAMME PRINCIPAL **
C     ************************
      COMMON C(12), S(12), T(12)
30    CALL INIT
      CALL TEST (KE)
      IF (KE .EQ. 0) GOTO 80
      WRITE (*, 900)
      GOTO 30
80    CALL CALC (TA)
      CALL IMPR (TA)
      WRITE (*, 910)
      READ (*, 920) N
      IF (N .EQ. 1) GOTO 30
C     ** FORMAT DE LECTURE-ECRITURE **
900   FORMAT (3X, 6HERREUR)
910   FORMAT (3X, 'CONTINUE = 1')
920   FORMAT (1X, I1)
      STOP
      END
```

Version Basic

```
 1 REM : PROG1
 5 REM
10 REM : ************************
15 REM : ** PROGRAMME PRINCIPAL **
20 REM : ************************
25 DIM C(12), S(12), T(12)
30 GOSUB 1000
40 GOSUB 2000
50 IF PI - 0 GOTO 80
60 PRINT "ERREUR"
70 GOTO 30
80 GOSUB 3000
90 GOSUB 4000
100 PRINT "CONTINUE = 1"
110 INPUT N
120 IF N = 1 GOTO 30
170 END
```

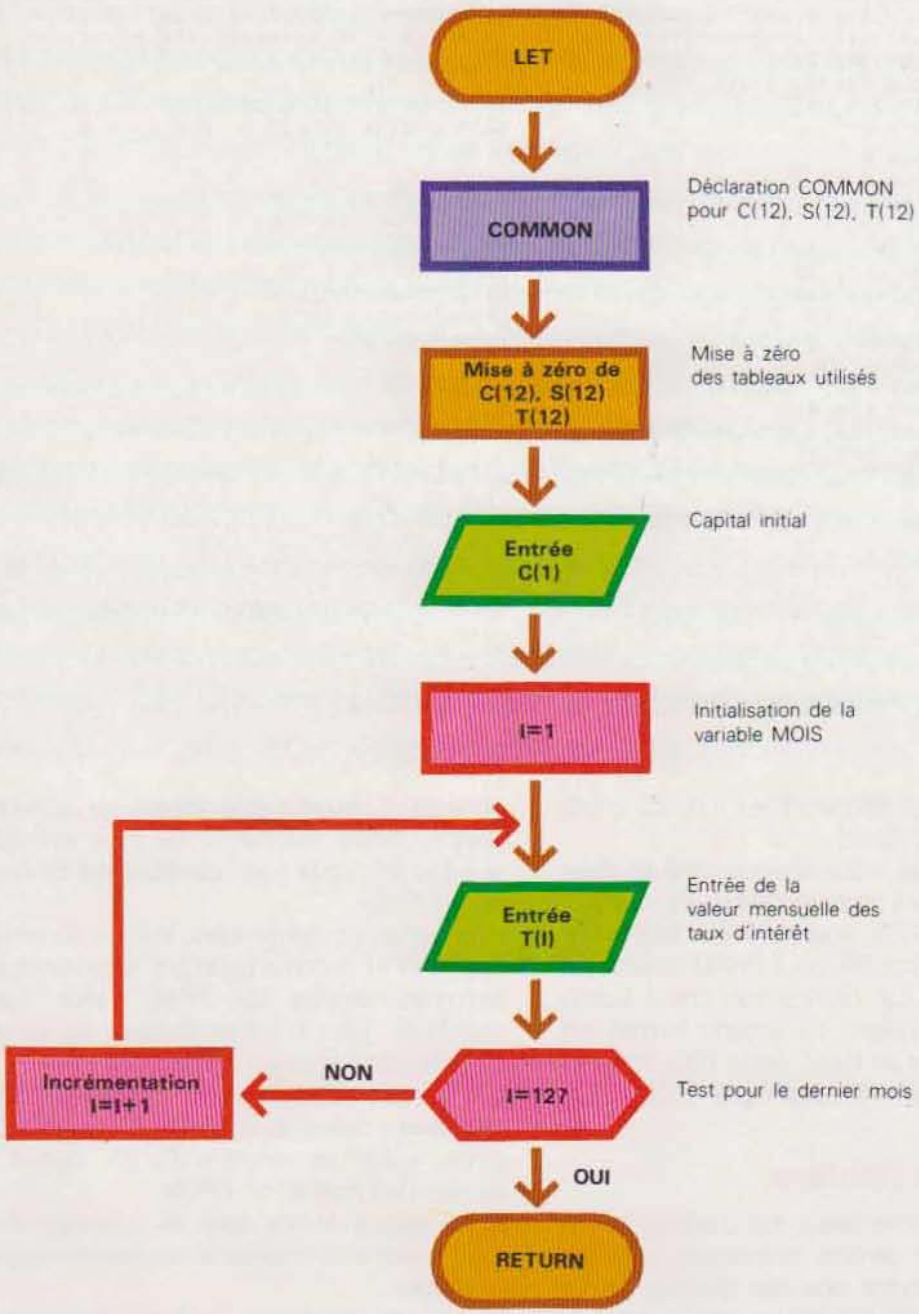
SOUS-PROGRAMME D'ENTREE DES DONNEES

Common
Fonction d'E/S
Boucle

Entrées Aucune

Fonction Mise à zéro des variables
Lecture du capital initial
Lecture du taux d'intérêt
du 12e mois

Sorties Capital initial [C1(1), début-mois 1]
Taux d'intérêt [T(12), 12 tableaux]



SOUS-PROGRAMME D'ENTREE DES DONNEES

Version Fortran

```
C       ******************************
C       ** ROUTINE D'INITIALISATION **
C       ******************************
        SUBROUTINE INIT
        COMMON C(12), S(12), T(12)
        DO 10 I=1, 12
        C(I) = 0
        S(I) = 0
        T(I) = 0
10      CONTINUE
        WRITE (*, 90)
        READ (*, 92) C(1)
        DO 20 I=1, 12
        WRITE (*, 94) I
        READ (*, 96)  T(I)
20      CONTINUE
C       ** FORMAT DE LECTURE/ECRITURE **
90      FORMAT (1X, 'CAPITAL = ')
92      FORMAT (F8.0)
94      FORMAT (1X, 'MOIS = ', 1X, I2)
96      FORMAT (F5.2)
        RETURN
        END
```

Version Basic

```
1000 REM : ************************
1001 REM : ** ROUTINE D'INITIALIS **
1010 REM : ************************

1020 FOR I = 1 TO 12
1030 C(I) = 0:S(I) =  0:T(I) = 0

1040 NEXT I
1050 INPUT "CAPITAL = "; C(1)

1060 FOR I = 1 TO 12
1070 PRINT "MOIS = "; I
1080 INPUT T(I)
1090 NEXT I

1100 RETURN
```

900,910,920 qui décrivent les formats d'E/S (sans équivalent Basic).

Dans cet exemple, nous avons adopté les deux formes possibles d'instructions de sortie : WRITE et PRINT. Si, pour WRITE, il faut préciser l'unité de sortie (6), pour PRINT, cela n'est pas nécessaire car l'instruction prend l'unité standard du système. Le second format est dont très proche du Basic (ligne 100), mais en Fortran, on y fait référence (ligne 910).

Gestion de fichiers

Le Fortran, comme beaucoup d'autres langages conçus pour de gros ordinateurs, considère le disque comme une des diverses unités d'entrée/sortie. En général, toutes les instructions d'E/S peuvent être dirigées sur des unités dont le disque fait partie. Le choix est opéré une fois précisé le code identificateur de l'unité ou du fichier.

Pour certaines d'entre elles, le code numérique est défini et connu a priori par le système (par exemple, emploi de PRINT vers l'unité standard) ; pour d'autres (fichiers sur disque) c'est le programmeur qui doit l'affecter.

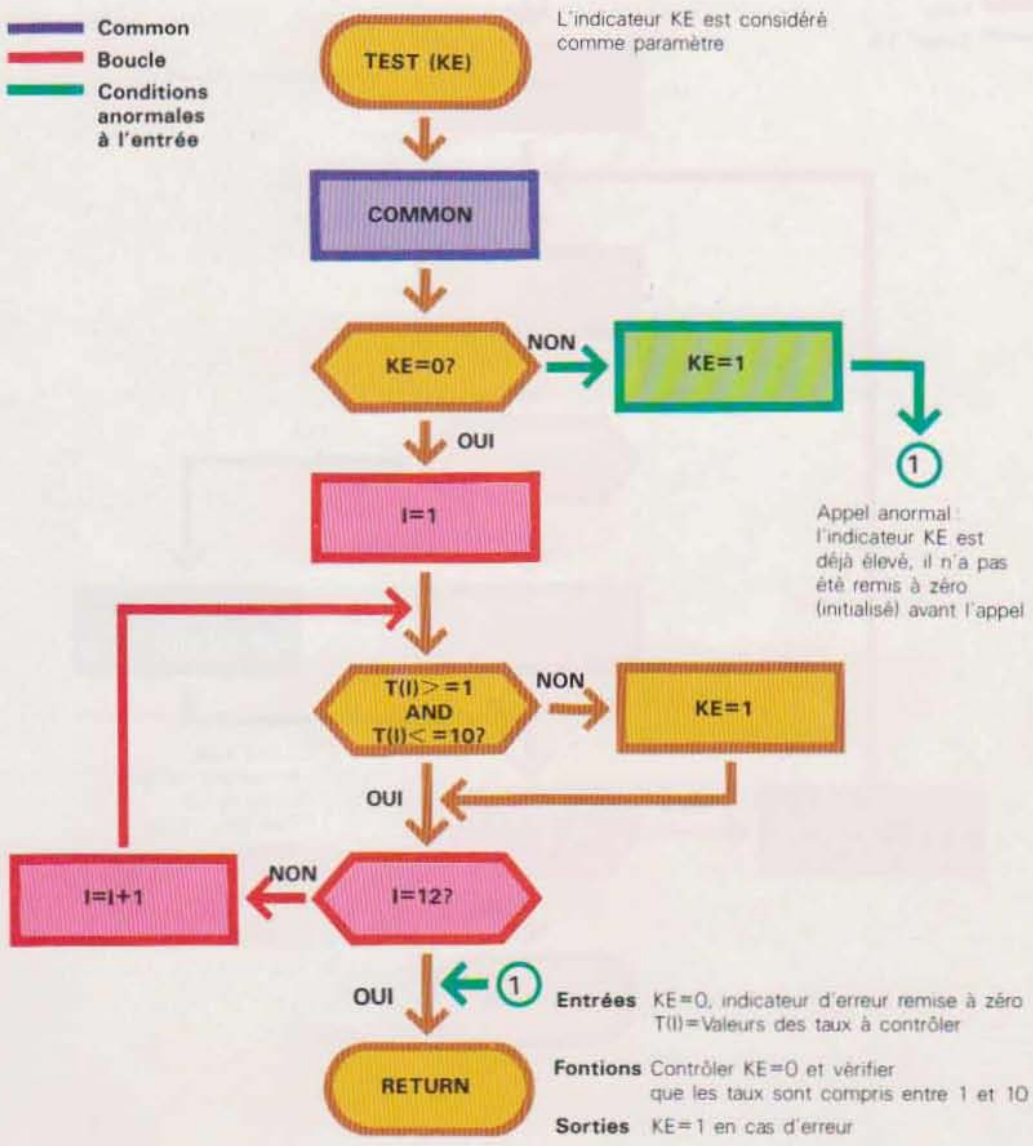
Les unités du premier type s'appellent **préliées :** celles du second type doivent être « liées » par un numéro d'unité logique au moyen de l'instruction OPEN.

Nous nous trouvons donc en présence d'une conception différente dans l'utilisation des périphériques.

En dehors du disque, le Basic standard ne

connaît pas d'autres dispositifs de gestion de fichiers. En Fortran, (comme en Cobol d'ailleurs) ces unités sont aussi considérées comme des fichiers. On peut ainsi effectuer un fichier à l'écran, à l'imprimante ou au clavier et, dans les grands systèmes, à l'unité bande. Chacune d'elles doit être activée avec des ordres et des instructions explicites, à moins d'être effectuée directement par le système.
De plus, le Fortran prévoit un type de fichier particulier, appelé **fichier interne,** composé d'une zone mémoire à laquelle on fait référence comme s'il s'agissait d'un quelconque fichier « externe », sur disque, par exemple. Chaque variable de cette zone de mémoire, ou chaque élément d'un tableau défini comme partie d'un fichier interne) est considéré comme un enregistrement.
Pour effectuer les opérations d'E/S il faut remplacer, dans les instructions respectives, le nom symbolique de la variable ou du tableau, par le numéro d'unité logique (fichier).

SOUS-PROGRAMME DE CONTROLE

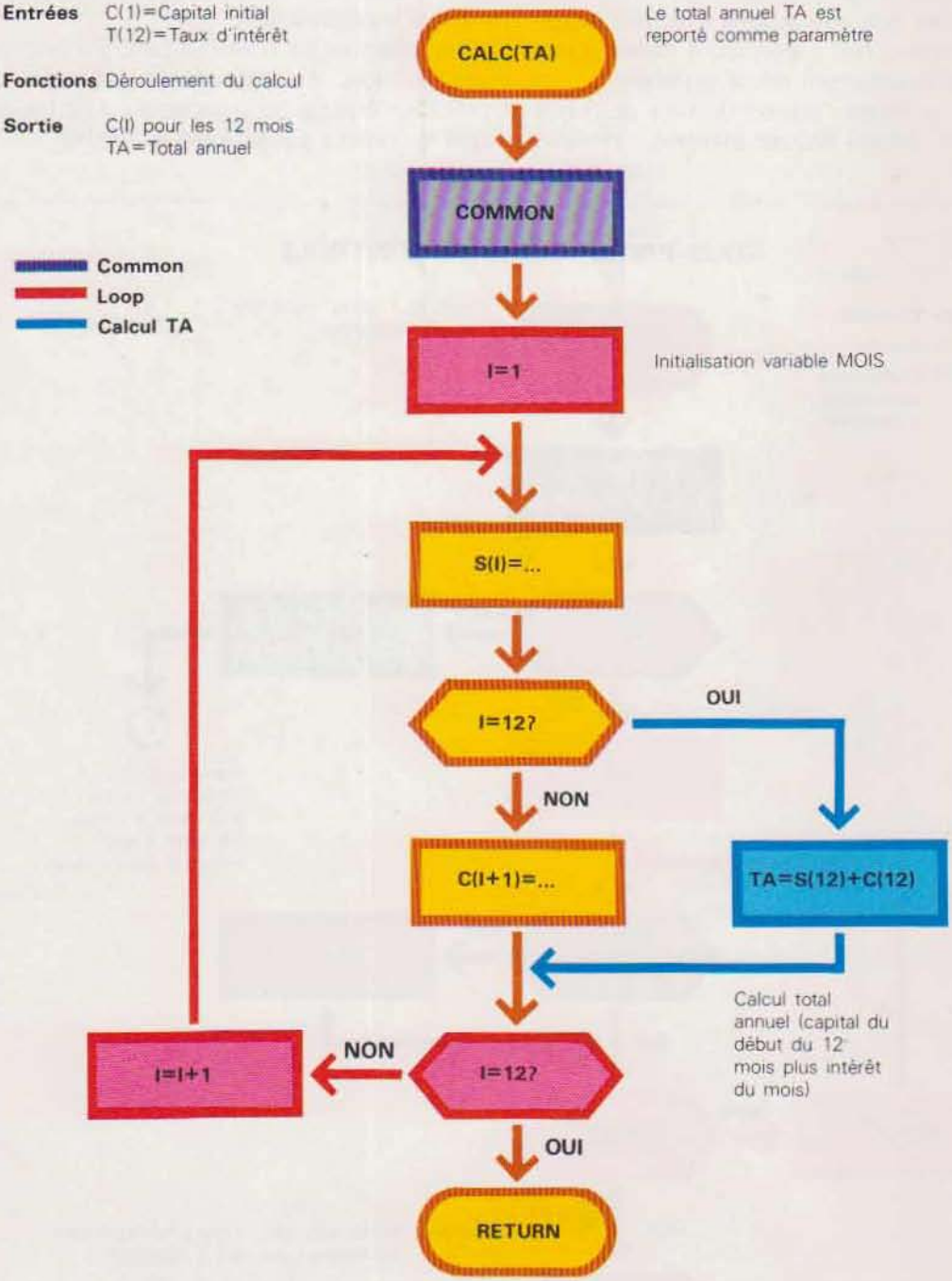


SOUS-PROGRAMME DE CALCUL

Entrées C(1)=Capital initial
T(12)=Taux d'intérêt

Fonctions Déroulement du calcul

Sortie C(I) pour les 12 mois
TA=Total annuel



SOUS-PROGRAMME DE CONTROLE ET DE CALCUL

Version Fortran

```
C     ********************************
C     ** ROUTINE DE VERIFICATION **
C     ********************************
      SUBROUTINE TEST (KE)
      COMMON C(12), S(12), T(12)
      KE = 0
      DO 10 I=1, 12
      IF (T(I) .LT. 1) KE = 1
      IF (T(I) .GT. 10) KE = 1
10    CONTINUE
      RETURN
      END

C     ********************************
C     ** ROUTINE DE CALCUL **
C     ********************************
      SUBROUTINE CALC (TA)
      COMMON C(12), S(12), T(12)
      DO 10 I=1, 12
      S(I) = (C(I) + T(I)) / 100.
      IF (I .LO. 12) GOTO 10
      C(I+1) = S(I) + C(I)
10    CONTINUE
      TA = S(12) + C(12)
      RETURN
      END
```

Version Basic

```
2000 REM : ********************************
2005 REM : ** ROUTINE VERIFICAT **
2010 REM : ********************************
2020 KE = 0
2030 FOR I = 1 TO 12
2040 IF T(I) < 1 THEN KE = 1
2050 IF T(I) > 10 THEN KE = 1
2060 NEXT I
2070 RETURN

3000 REM : ********************************
3005 REM : ** ROUTINE DE CALCUL **
3010 REM : ********************************
3020 FOR I = 1 TO 12
3030 S(I) = (C(I) + T(I)) / 100
3040 IF C(I) = 12 GOTO 3060
3050 C(I+1) = S(I) + C(I)
3060 NEXT I
3070 TA = S(12) + C(12)
3080 RETURN
```

Par exemple, l'instruction :

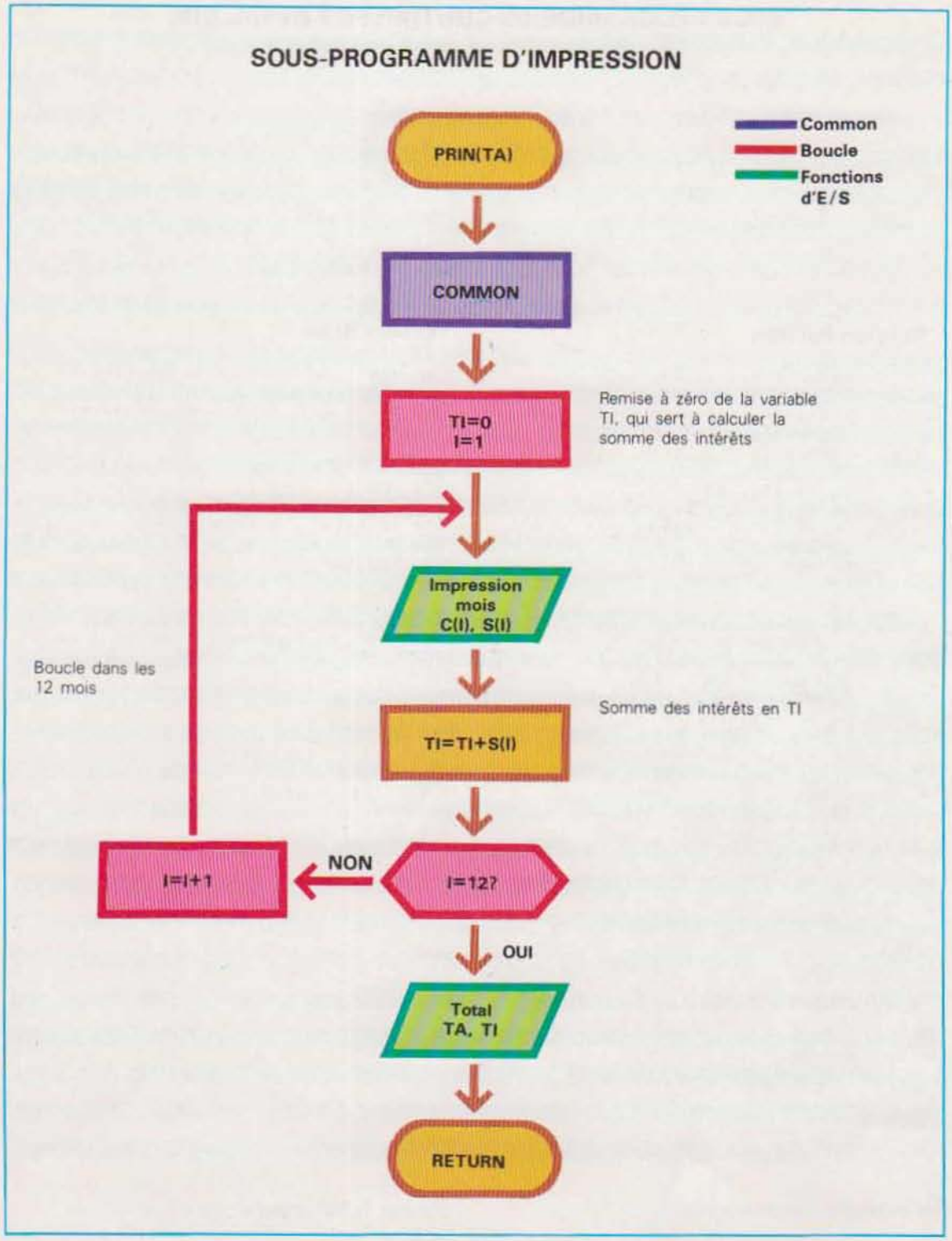
WRITE (6,30)...

se réfère, dans son usage normal, à l'unité 6 (avec le format 30). Pour l'adresser comme un fichier interne, il faut remplacer la valeur 6 par le nom de la variable (ou du tableau) considérée comme fichier interne.

La ligne :

WRITE (ESSAI,30)V

transfère le contenu de la variable V selon le format défini à la ligne 30, dans la variable ESSAI.



Gestion des fichiers sur disque

De nombreuses instructions prévues pour le disque restent également valables pour tous les autres fichiers. Par la suite, nous ferons référence implicitement à l'unité disque, car il s'agit de l'unité la plus importante quand on parle de fichiers. Le Fortran gère deux types de fichiers : ceux qui sont à accès séquentiel et

SOUS-PROGRAMME D'IMPRESSION

Version Fortran

```
C       ****************************
C       ** ROUTINE D'IMPRESSION **
C       ****************************
        SUBROUTINE IMPR (TA)
        COMMON C(12), S(12), T(12)
        TI = 0.
        DO 20 I=1, 12
        WRITE (*, 30) I, C(I), S(I)

        TI = TI + S(I)
20      CONTINUE
        WRITE (*, *) TA, TI
C       ** FORMAT D'IMPRESSION **
30      FORMAT (1X,'MOIS =', 1X,I2,
                8X,'CAPITAL =' F9.1,
                3X,'INTERET =' F8.1)
        RETURN
        END
```

Version Basic

```
' 4000 REM : ***************************
' 4005 REM : ** ROUTINE IMPRESSION **
' 4010 REM : ***************************
'
'
' 4020 TI = 0
' 4030 FOR I = 1 TO 12
' 4040 PRINT "MOIS ="; I,
'            "CAPITAL ="; C(I),
'            "INTERET ="; S(I)
' 4050 TI = TI + S(I)
' 4060 NEXT I
' 4070 PRINT TA, TI
'
'
'
'
' 4080 RETURN
'
```

ceux qui sont à accès direct, au sens donné à ces expressions en Basic.
Trois types d'enregistrements sont prévus :

- Enregistrements formatés — Ils sont écrits et lus selon des formats précis
- Enregistrements non formatés — Ils contiennent une série de données telle qu'elle se présente en mémoire
- Enregistrement Fin-de-fichier (EOF) — Il ne contient pas de données et indique la fin d'un fichier séquentiel. Il est noté par l'instruction ENDFILE.

Principales instructions liées aux fichiers :

OPEN	Relier à un fichier (ouverture)
READ	Lire données
WRITE	Ecrire données
BACKPACE	Déplacer le pointeur d'un fichier séquentiel d'un enregistrement vers l'arrière
REWIND	Placer le pointeur d'un fichier séquentiel sur le premier enregistrement
ENDFILE	Ecrire l'enregistrement End-Of-File, qui marque la fin d'un fichier séquentiel
INQUIRE	Fournir des informations relatives à un fichier
CLOSE	Fermer un fichier

OPEN. Cette instruction « relie » le fichier et le système en lui affectant un numéro d'unité logique.
Elle contient quelques définitions des caractéristiques du fichier, semblables à celles employées en Basic.

Les paramètres à fournir sont :

UNIT	numéro d'unité logique associé. Si on pose UNIT=3 le fichier est reconnu comme unité 3

IOSTAT Spécifie la variable (entière) où il doit transférer des codes d'erreurs.

ERR Numéro de ligne où est réalisé le transfert de contrôle en cas d'erreur : ERR= 130 signifie « GO TO 130 si erreur »

FILE Spécifie le nom du fichier. S'il n'existe pas, il est créé par le système lorsqu'il exécute OPEN.

STATUS Vérifie l'existence du fichier. Valeurs admises :

STATUS='OLD' Le fichier doit exister ; sinon erreur

STATUS='NEW' : le fichier ne doit pas exister ; sinon erreur

STATUS='SCRATCH' : On ne doit fournir aucun nom. Un fichier de travail vient d'être créé

STATUS='UNKNOWN' lance la recherche ; si le fichier n'existe pas, il est créé

Les deux premières options (OLD, NEW) servent à éviter les erreurs suivantes : ouvrir un fichier déjà existant ou qui ne doit pas être présent ; la dernière (UNKNOWN) gère automatiquement la sélection et relie le fichier défini, s'il existe, ou le crée (puis le relie) s'il n'existe pas. UNKNOWN est affectée par défaut.

ACCESS Moyen d'accès ; il peut être de divers types :

DIRECT Accès direct ; dans les E/S suivantes il faut spécifier le numéro d'enregistrement (REC=...)

SEQUENTIAL Fichier séquentiel ; c'est le type choisi par défaut

FORM Précise si les données doivent ou non être formatées ; elle prend la valeur FORMATTED ou UNFORMATTED

RECL Longueur des enregistrements en octets par accès direct

BLANK Peut prendre les valeurs NULL ou ZERO. Dans le premier cas, d'éventuels espaces « blanc » entre les valeurs numériques sont ignorés, dans le deuxième, ils sont convertis en zéros. Par exemple 15b79 devient avec l'option NULL, 1579 ; et 15079 avec l'option ZERO. Par défaut le système opte pour NULL.

Tableau de contrôle de la Fairchild.



Marka

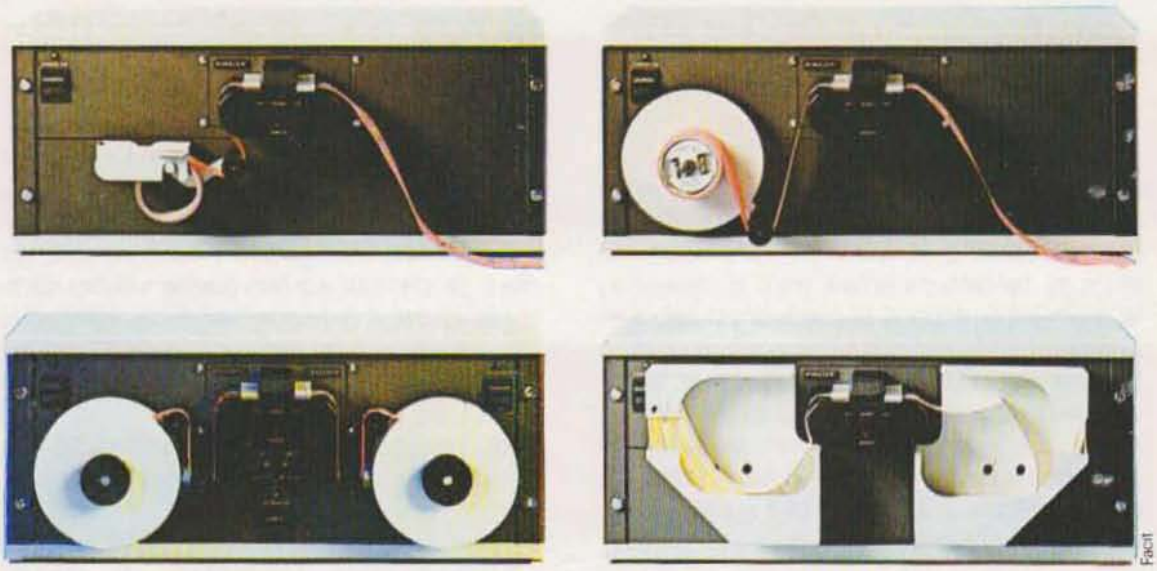
Un exemple d'emploi des spécifications associées à OPEN pourrait être le suivant :

```
OPEN (2,IOSTAT=K,ERR=1,FILE='F1')
```

définit le fichier F1 comme un fichier séquentiel (par défaut) relié à l'unité logique 2. K est la variable d'erreur et 1 le numéro de ligne pour la gestion des conditions d'erreur.
La syntaxe complète pour définir l'unité logique est UNIT=2, mais l'option UNIT peut être omise. La ligne :

```
OPEN (2,FILE='F1',ACCESS='DIRECT',
RECL=1)
```

ouvre le fichier F1 à accès direct, avec des enregistrements d'un octet de longueur, sur l'unité logique 2. La ligne :



Modèles de lecteurs de bandes perforées Facit 4030. De gauche à droite et de haut en bas : modèle pour la lecture d'une bande de 20 m de long (l'équivalent de 8 000 caractères, de 50 m (20 000 caractères), 100 m (40 000 caractères). En bas à droite, lecteur de bandes pliées.

```
OPEN (2,FILE='TEST',STATUS='OLD'
ACCESS='DIRECT,RECL=100)
```

assure les mêmes fonctions que l'instruction précédente, mais elle vérifie que le fichier existe déjà (STATUS='OLD'). Si tel n'est pas le cas, elle génère un message d'erreur. L'autre forme aurait, en revanche, créé le fichier en cas d'erreur ; en effet, par défaut, on a STATUS = 'UNKNOWN'.

READ ET WRITE. Les instructions READ et WRITE, appliquées à des fichiers sur disque, ont des formats identiques à ceux des autres périphériques.
Après l'ouverture du fichier et l'affectation d'un numéro d'unité logique, les opérations d'E/S sont identiques à celles des autres périphériques.
Naturellement, pour les fichiers directs, il faut préciser le numéro d'enregistrement à employer. Par exemple, la ligne :

```
READ  (2,130,REC=12)variable1,variable2,...
```

lit, selon le format 130, l'enregistrement numéro 12 du fichier numéro 2 et transfère les valeurs dans les variables spécifiées. De la même manière, en écriture, avec la ligne :

```
WRITE (2,130,REC=7)variables
```

Les valeurs des variables sont transférées dans l'enregistrement n° 7.

BACKSPACE. Dans les fichiers séquentiels, à la suite de chaque opération d'E/S, un pointeur (géré par le système) sera incrémenté ; il indique le numéro du premier enregistrement disponible. L'opération suivante traite l'enregistrement suivant, et ainsi de suite. L'instruction BACKSPACE décrémente ce pointeur (-1), en le positionnant à nouveau sur le dernier enregistrement traité. La syntaxe de l'instruction est la suivante :

```
BACKSPACE n
```

avec n numéro du fichier.

REWIND. Positionne le pointeur au début du fichier. Les paramètres sont UNIT, IOSTAT ERR, chacun avec la signification décrite dans OPEN. Par exemple, la ligne :

```
REWIND(2,IOSTAT=K,ERR=100)
```

positionne le pointeur au début du fichier 2 ; la variable d'erreur est K, et en cas d'erreur, le contrôle passe à l'instruction 100.

ENDFILE. Ecrit un enregistrement à la fin du

fichier ; le numéro d'unité est le seul paramètre. La ligne :

ENDFILE 2

ferme le fichier précédemment ouvert avec le numéro 2. Les instructions REWIND et ENDFILE, exécutées dans cet ordre, provoquent la perte du contenu du fichier, dans la mesure où la première postionne le pointeur au début et que la seconde marque cette position comme Fin-de-Fichier (EOF). Le résultat est un fichier avec un seul enregistrement contenant la fin du fichier. Les données qui suivent ne sont plus accessibles, car le système traite le fichier de manière séquentielle, trouve EOF sur le premier enregistrement et s'arrête.

INQUIRE. Fournit les caractéristiques d'un fichier. Les paramètres sont ceux employés pour OPEN, avec des valeurs dépendant de l'état du fichier. Par exemple, l'instruction :

```
INQUIRE (FILE='ESSAI',OPENED=K,
NUMBER=N,ACCESS=A)
```

restituera :

K = TRUE si le fichier est ouvert (sinon FALSE)
N = numéro d'unité logique associée
A = Moyen d'accès

CLOSE. Ferme le fichier spécifié et le déconnecte du numéro d'unité logique. Les paramètres prévus sont :

UNIT	Numéro de l'unité
IOSTAT	Etat d'erreur
ERR	Numéro de ligne pour les erreurs
STATUS	Prend deux valeurs : KEEP (fermeture fichier), ou DELETE (fichier écrasé). KEEP est assumé par défaut.

Dans l'ordinogramme ci-contre, le programme lit un fichier séquentiel et calcule la moyenne des valeurs (numériques) qu'il contient.
Chaque valeur (variable V) occupe un enregistrement composé d'un réel (trois chiffres entiers) et d'un chiffre décimal (format F5.1).
Dans l'ordinogramme, apparaît le contrôle de fin de fichier (qui n'existe pas sur le listing de la page 1194, car il est implicite dans l'instruction READ, ligne 100) avec l'option END = 200.
Le fichier à ouvrir s'appelle ESSAI et il est associé à l'unité logique 1. Observons la ligne 100. Pour le calcul de la moyenne des valeurs on ne peut utiliser le nom MOYENNE car il commence par une des lettres qui définissent les variables entières (I,J,K,L,M,N) ; dans ce cas, en effet, on perdrait les éventuelles valeurs décimales. Comme on le constate sur le listing, des formats peuvent se trouver à un endroit quelconque du programme, à condition que ce soit avant END.
Cette instruction doit être la dernière, car elle agit comme un signal de fin pour le compilateur.
De plus, en Basic, le contrôle d'EOF ne peut s'effectuer dans l'instruction de lecture même ; il faut donc le prévoir séparément.

Instructions particulières au Fortran

A ce groupe, appartiennent des instructions propres aux formes de programmation les plus évoluées. Elles ne seront mentionnées qu'à titre indicatif car leur existence est liée au type de machine et de compilateur employés.
Les principales sont :

EXTERNAL
ENTRY
SAVE
BLOCK DATA

EXTERNAL

Définit un nom de sous-programme et l'utilise comme argument dans un appel.

La syntaxe est la suivante :

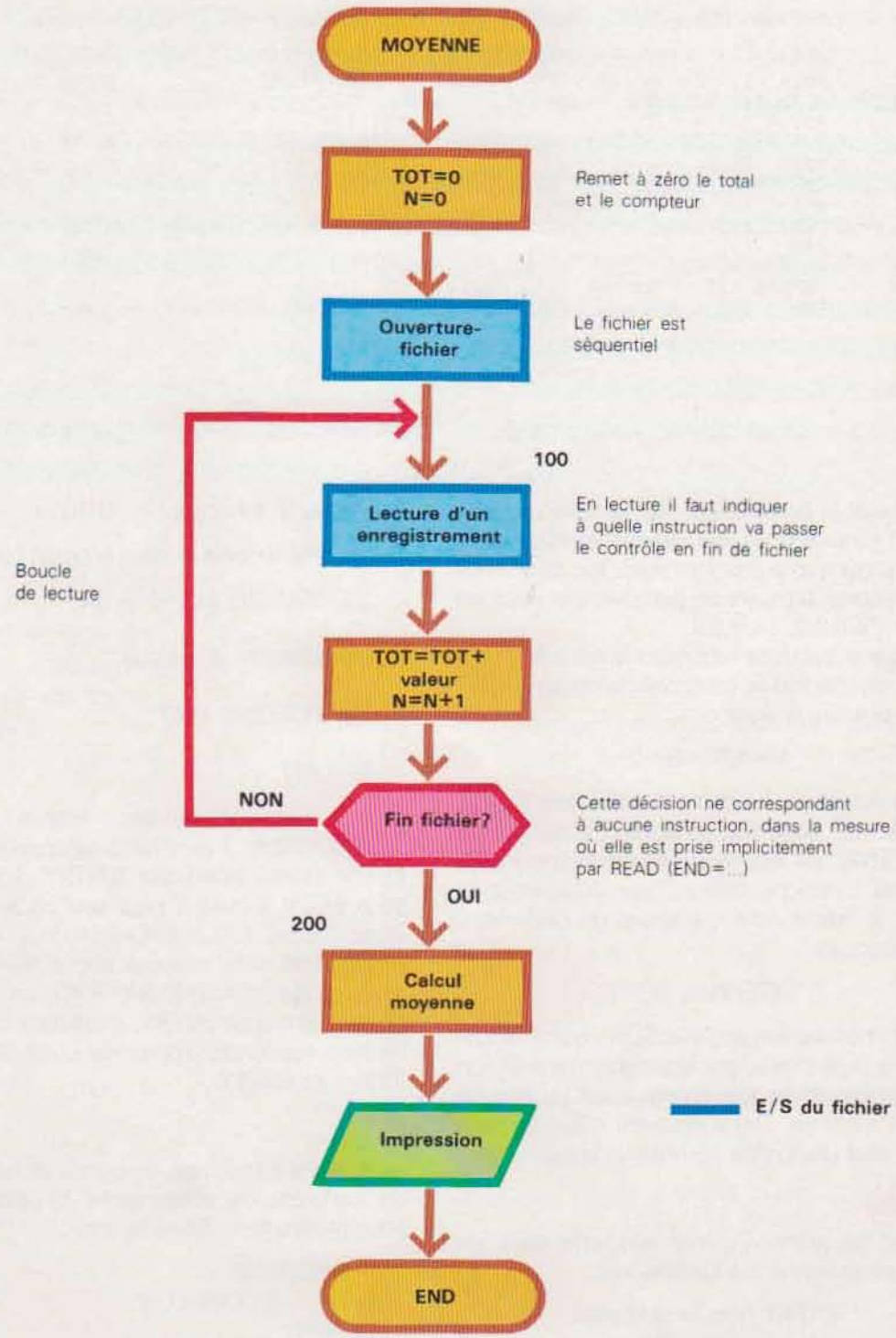
EXTERNAL nom 1, nom 2, etc.

Où nom 1, nom 2... sont les noms des sous-programmes à transférer.

Par exemple, les instructions :

```
EXTERNAL A
CALL SOMME (A,B,C)
SUBROUTINE SOMME (A,B,C)
X = A (N) + B + C
RETURN
```

CALCUL DE LA VALEUR MOYENNE DES DONNEES CONTENUES DANS UN FICHIER



LECTURE D'UN FICHIER ET CALCUL DE LA MOYENNE DES VALEURS

```
      PROGRAM MEDIA
      TOT = 0
      N = 0
      OPEN (1, FILE = 'ENTREE', ACCESS = 'SEQUENTIAL')
100   READ (1, 110, END = 200) V
      TOT = TOT + V
      N = N + 1
      GOTO 100
200   RMOYENNE = TOT / N
      WRITE (6, 210) RMOYENNE
      CLOSE (1)
      STOP
110   FORMAT (F5.1)
210   FORMAT (1X, 'MOYENNE = ', F9.1)
      END
```

informent le compilateur que le nom symbolique A n'indique pas une variable, mais un autre sous-programme (ou fonction), qui peut avoir ses propres arguments, par exemple dans un appel [SOMME (A,B,C)].
Lorsque le système rencontre le nom A, il utilise la routine (ou la fonction) correspondante.
Par exemple, la ligne :

X=A(N)+B+C

utilise A comme fonction du paramètre N alors que B et C sont des variables. La déclaration EXTERNAL est également employée pour définir des fonctions définies par l'utilisateur et ayant le même nom que celles du système.
Par exemple :

EXTERNAL SQRT

décrit une fonction appelée SQRT qui effectuera les calculs prévus par le programmeur et non une racine carrée telle qu'elle aurait pu être lue par le système. Cette dernière n'est évidemment plus disponible au niveau du système.

ENTRY

Définit un point d'entrée alternatif dans un sous-programme. La syntaxe est :

ENTRY nom (arguments)

où nom est le nom symbolique de cette nouvelle entrée (entry-point) et arguments les grandeurs à échanger.

Par exemple, dans le sous-programme :

```
SUBROUTINE ESSAI (A,B,C)
. . . . . .
(déroulement du calcul)
. . . . . .
ENTRY ESSAI 1 (C)
. . . . . .
RETURN
```

on a bien prévu une entrée normale (SUBROUTINE...) avec les 3 paramètres A,B,C et une entrée alternative (ENTRY...) qui s'appelle ESSAI 1 avec C pour seul paramètre.
Avec l'appel CALL ESSAI (A,B,C), le sous-programme sera exécuté depuis son début, alors qu'avec CALL ESSAI 1 (C), on passe le contrôle à la ligne ENTRY, en sautant toutes les lignes précédentes, comprises entre SUBROUTINE... et ENTRY...

SAVE

Avec cette instruction, on conserve les valeurs de certaines variables après le retour d'un sous-programme. Dans la liste :

```
SUBROUTINE xy
SAVE A,B/DONNEES/
RETURN
SUBROUTINE Z
SAVE
```

La première instruction (SAVE A,B...) préserve les valeurs des variables A et B dans le bloc DONNEES, la seconde (SAVE) conserve les valeurs de toutes les variables utilisées dans le sous-programme Z.

BLOCKDATA

En Fortran, on peut appeler des sous-programmes spéciaux qui ont pour seule fonction de définir et d'initialiser des variables intégrées à un COMMON étiqueté (labelled common).
Outre les instructions de déclaration, ces sous-programmes ne peuvent contenir que des instructions d'initialisation. Par exemple :

```
BLOCKDATA UN
COMMON/B/V (5), C,X
DATA V (5*3.1)
END
```

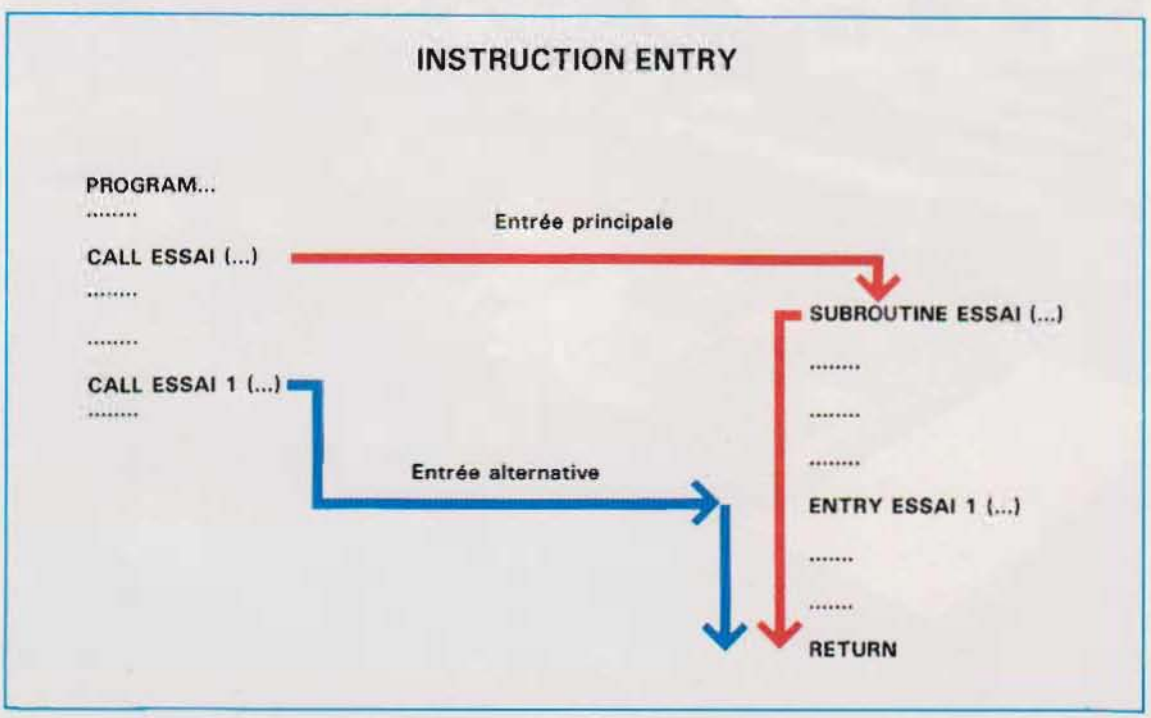
La routine, qui s'appelle UN, définit un zone common de nom B, à laquelle appartiennent les variables V(5), C et X ; de plus, cette routine initialise le tableau V(5), avec des valeurs égales, soit 3.1.

Améliorations spéciales

Par rapport au Fortran standard, certains compilateurs comportent des améliorations obéissant aux normes MIL-STD-1753 (Military Standard) et qui permettent d'agir directement sur les bits de chaque variable.

SHIFT	Déplace les bits d'un nombre déterminé de positions
EXTRACTION	Prélève un certain nombre de bits d'une position de mémoire, (extraction partielle)
TEST	Effectue un contrôle d'état de chaque bit (ON/OFF)
SET	Place n'importe quel bit sur ON
CLEAR	Met en OFF (zéro) le bit indiqué
MOVE	Déplace un ou plusieurs bits
PRODUIT LOGIQUE	fonction AND sur les bits d'un mot (ET)
SOMME LOGIQUE	fonction OR (OU)
OU EXCLUSIF	fonction XOR (OUX)
COMPLEMENT	fonction NOT (NON)

Ces fonctions, qui sont caractéristiques des compilateurs les plus avancés, sont très



utilisées en gestion de périphériques, en particulier des servomécanismes et des appareils d'acquisition de données. Elle confèrent, au Fortran, à la fois une souplesse et une variété d'emploi équivalente sinon supérieure au Basic, la seule différence restant dans la nécessité de compiler, ce qui rend plus difficile le développement et la mise au point de programmes.

Format normal des instructions

La syntaxe des instructions de n'importe quel langage peut se représenter graphiquement, sous forme de description synthétique de ses principales caractéristiques.
Cette représentation, appelée **Railroad Normal Form** (RNF) n'est pas très pédagogique, surtout pour un néophyte. En revanche, elle est très utile comme aide-mémoire pour ceux qui ont une connaissance même superficielle des instructions. En principe, la représentation en format normal s'applique à tout langage symbolique. Elle consiste à illustrer graphiquement les alternatives pour atteindre un certain but ou pour définir une fonction.
Le seul symbole particulier est représenté par un cercle associé à un chiffre. Il signifie que l'instruction correspondante doit figurer un nombre de fois au moins égal au chiffre du cercle.
Les instructions indiquant un programme ou une de ses parties sont :

— Main ou programme principal (PROGRAM) ; l'instruction apparaît au moins une fois (elle est signalée par le symbole I)
— Function (fonction)
— Subroutine (sous-programme)
— Block (bloc de données)

Elles ont toutes des significations analogues et seront donc représentées de manière « parallèle » dans la RNF.
Un programme principal contient des instructions (program statement) et les fonctions function statement... Des étiquettes (n° de ligne) leur sont affectées et l'ensemble doit s'achever avec le mot END.

Terminal d'un Hewlett-Packard.

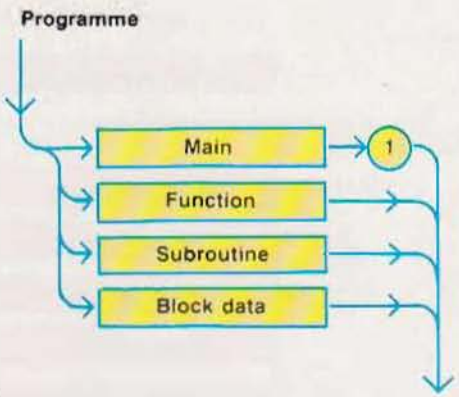


D. Bonecchi

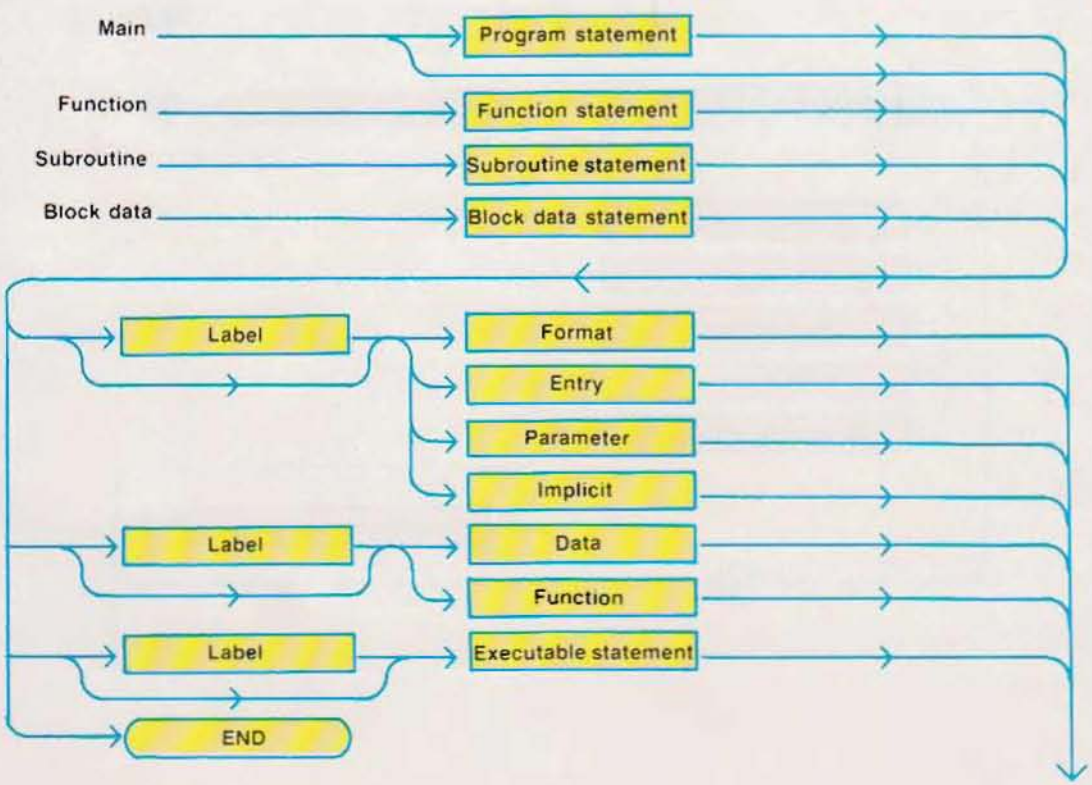
Ainsi, pour chacune des lignes précédentes, il existe de nouvelles possibilités, à un niveau de détail plus poussé.
Le graphique ci-dessous illustre cette hiérarchie sur 2 niveaux.
En poussant encore vers le détail, on aboutit à la description individuelle de chaque instruction (voir exemple, pages suivantes).

REPRESENTATION "RAILROAD NORMAL FORM"

1er niveau



2e niveau



REPRESENTATION DE LA SPECIFICATION DE TYPE

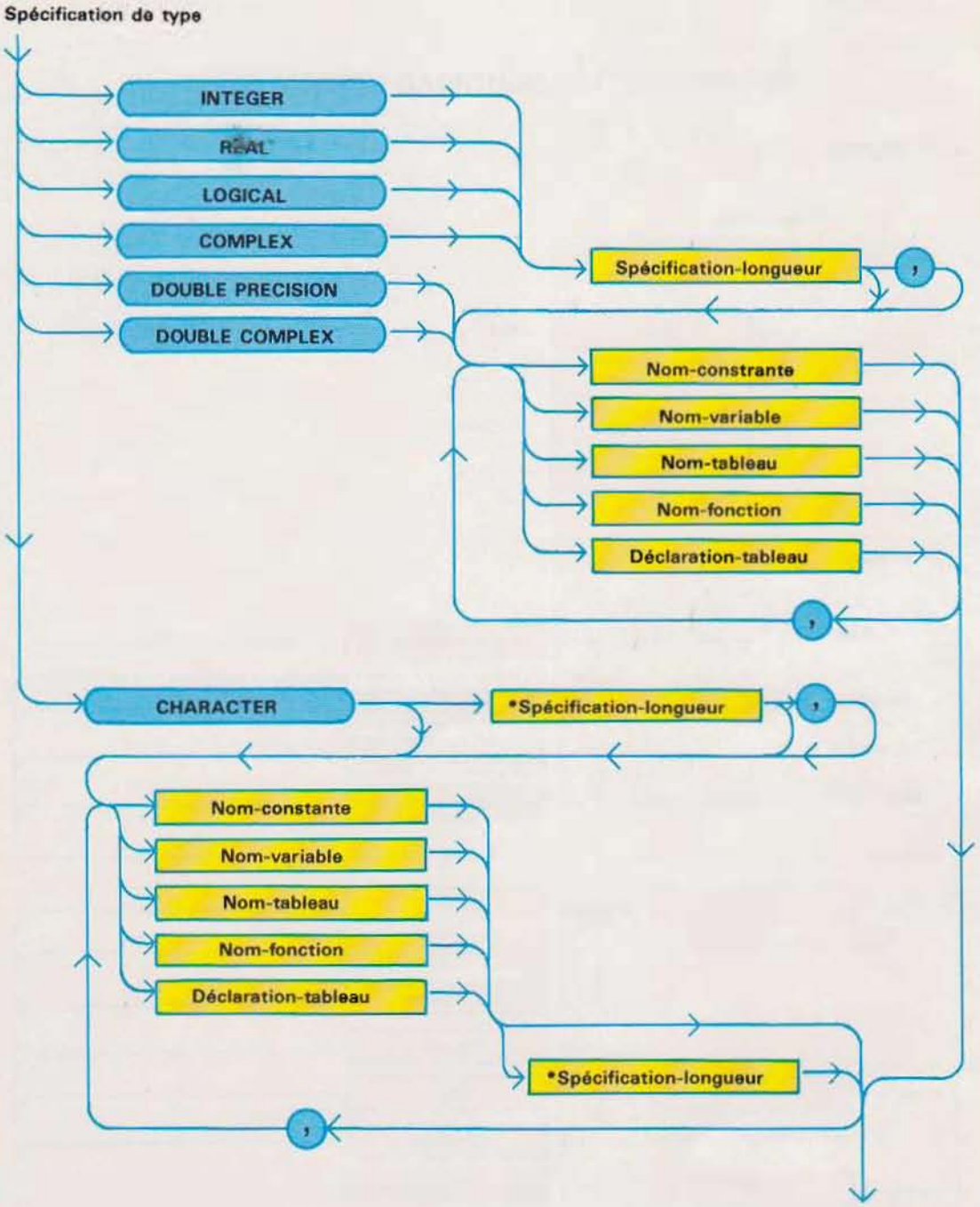
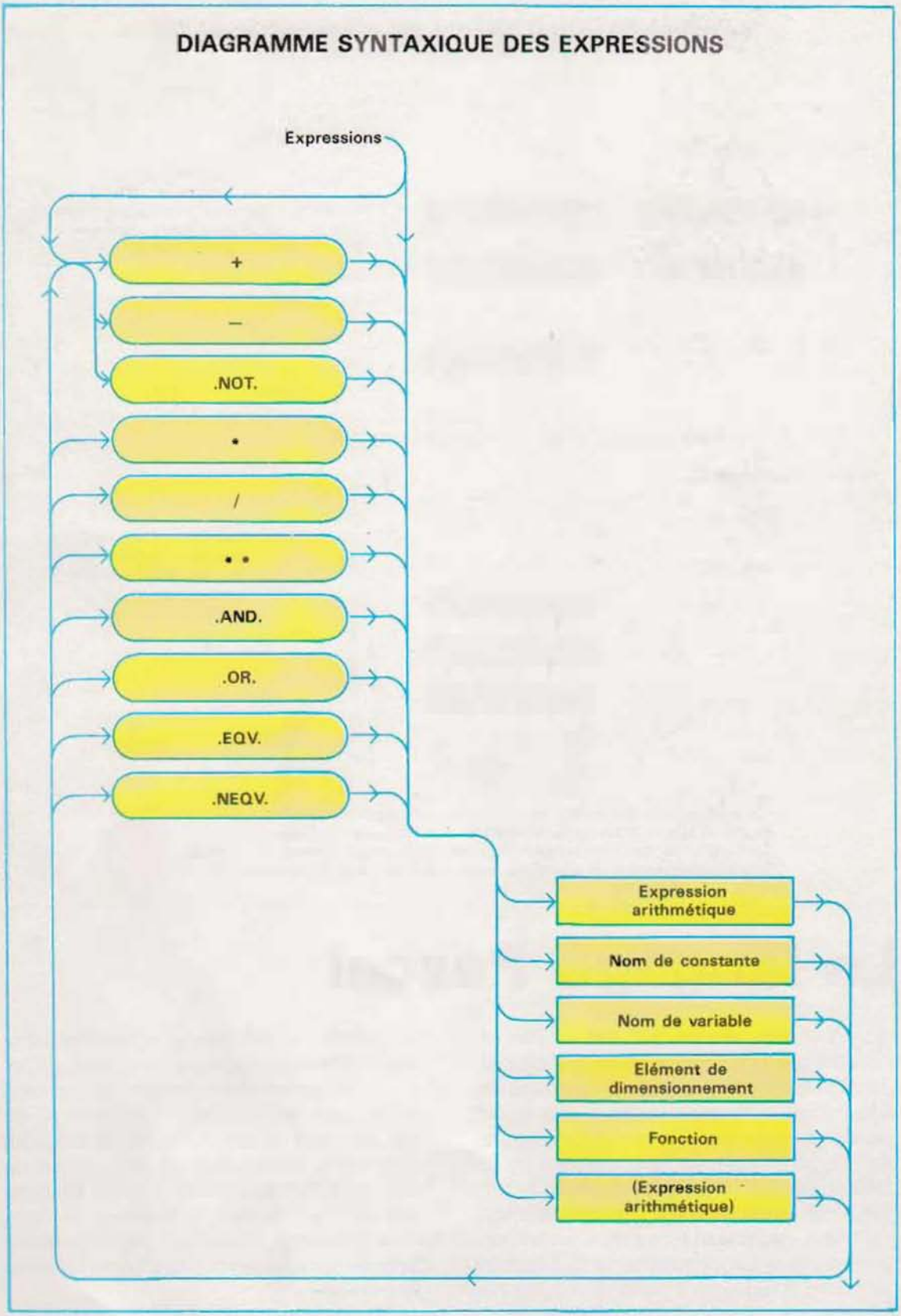
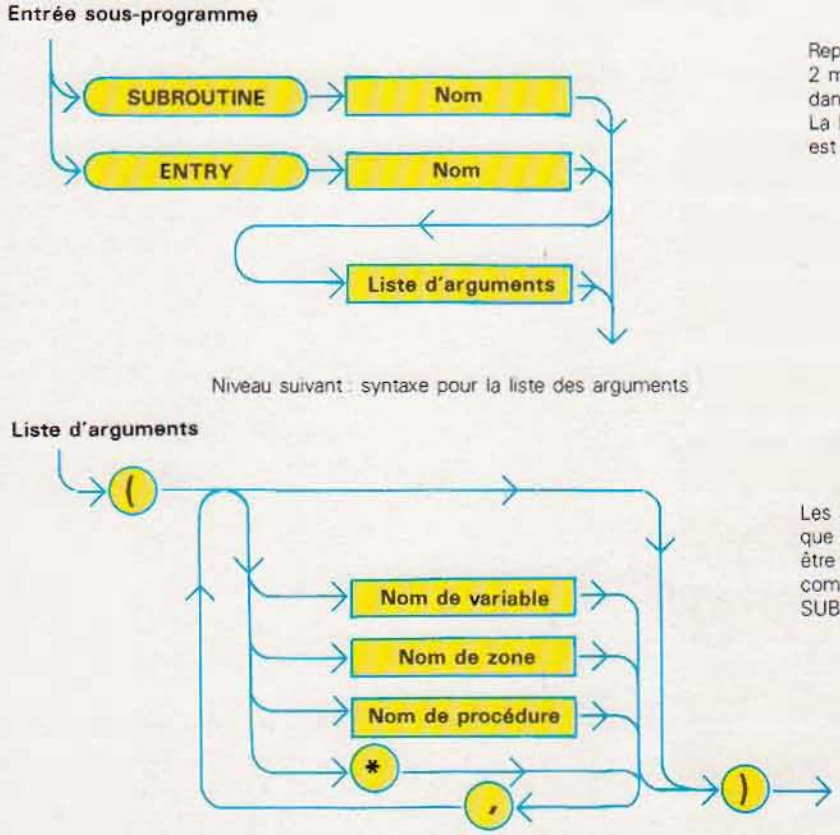


DIAGRAMME SYNTAXIQUE DES EXPRESSIONS



REPRESENTATION D'ENTREE EN SOUS-PROGRAMME ET STRUCTURE DES PARAMETRES



Le langage Pascal

Au début des années 60, les ordinateurs n'avaient pas la puissance qu'on leur connaît. Et quand le programmeur devait construire des algorithmes en langage machine, ces limites tenaient évidemment à la technologie employée : capacité maximale de la mémoire ne dépassant pas les 16-32 Koctets. Aussi le programmeur était-il amené à améliorer le rendement de la machine en recourant à des astuces particulières qui, malheureusement, finissaient par rendre pratiquement illisible le programme.

Les progrès réalisés depuis, aussi bien sur la machine elle-même que sur les logiciels de base, ont inconstestablement donné naissance à une nouvelle génération d'ordinateurs. C'est ainsi que sont apparus d'abord les langages symboliques assemblés, puis les langages de haut niveau (de plus en plus orientés langages naturels). Ces derniers comportent de nombreux sous-programmes (routines d'utilisation générale absentes des machines des premières générations.

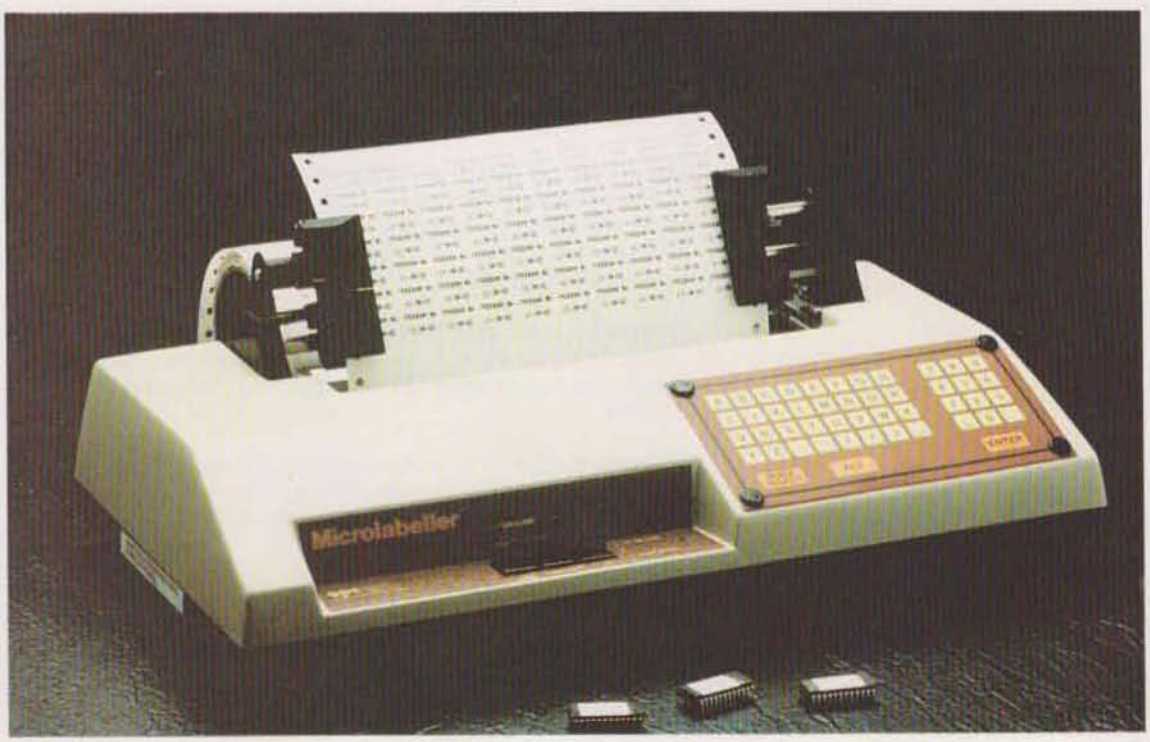
Une telle évolution a rendu inopérante l'emploi, par les programmeurs, des astuces qui faisaient précisément leurs compétences. Comme on peut l'imaginer, il n'a pas été facile pour les créateurs du Fortran (introduit vers 1957) de combattre la résistance et les préjugés des programmeurs dont l'efficacité se mesurait à la « technologie des trucs » (trickology, en anglais), devenue un art. Après le Fortran, en 1960, vint l'Algol (Algorithmic language), fondamentalement proche du Fortran mais disposant d'un atout supplémentaire. Au niveau des instructions, on avait la possibilité de créer des « structures » de procédures en regroupant les instructions de manière à former une opération complète, avantage dont le Fortran disposait seulement au niveau des expressions arithmétiques. On pouvait, dès lors, composer des instructions en structures de contrôle ; les opérations définies par un programme étaient donc vues comme une répétition de blocs et non pas comme de simples instructions.

Malheureusement, l'Algol, malgré ses innovations, n'a pas eu le succès escompté. Il reste aujourd'hui un langage de chercheurs et d'universitaires. Son absence, dans des applications scientifiques ou de gestion, est certainement due à la complexité du concept de structure, incompatible avec l'art d'utiliser les trucs et les sauts inconditionnels à l'intérieur des instructions.

Cette inertie est analogue à celle qui a créé un obstacle dans le passage de l'Assembleur vers le Fortran. Aujourd'hui, il ne s'agit plus tant de pousser la technologie dite « hard » pour gagner quelques microsecondes dans la puissance de traitement, encore que, dans le domaine de l'intégration à très haute échelle (VLSI), on soit encore loin des limites. Pour beaucoup, l'effort doit se porter sur la fiabilité des programmes et sur les possibilités de les modifier sans encombre.

C'est précisément la philosophie de la **programmation structurée.** Elle consiste à fournir au programmeur une méthode pour résoudre les problèmes de manière systématique sans recourir à son « atelier de trucs ». Parmi les langages les plus courants (voir page 1204), seul le Pascal (et demain peut-être l'Ada) revient à la structuration proposée par l'Algol. Pour que la structuration ne soit plus liée à un langage particulier, on tente de plus en plus d'y introduire des **structures de contrôle.** C'est une tendance qui devrait se confirmer.

Imprimante programmable pour l'édition d'étiquettes autocollantes.



B. Coleman/Matra

LANGAGES DE PROGRAMMATION

Date création	Nom du langage	Description
1957	Fortran **For**mula **Tran**slator	Très employé. Utilise des notations très proches de l'algèbre. L'unique structure de données est le tableau. Possibilité de définir des sous-programmes comme bibliothèques. L'efficacité des premiers compilateurs a rendu acceptable le Fortran dans toutes les applications scientifiques, mathématiques et statistiques.
1960	Algol **Algo**rithmic **l**anguage	Peu utilisé en dehors des milieux de recherche sur le software. Plus connu en Europe qu'aux Etats-Unis, il introduit les concepts de base de la programmation structurée.
1960	Cobol **Co**mmon **b**usiness **o**riented **l**angage	Très courant. Par définition, le langage de gestion utilise une notation "English like" (c'est-à-dire très proche du langage naturel anglais).
1961	Lisp **List p**rocessing. Langage de programmation en intelligence artificielle.	Très utilisé dans les recherches sur l'intelligence artificielle. La structure de base et la "liste" dans laquelle les éléments sont des données élémentaires ou des listes.
1962	Snobol	Langage de programmation pour la manipulation de chaînes de caractères. Il inclut de puissantes instructions pour la recherche d'ensembles de caractères ou chaînes. Il est, par exemple, utilisé dans la gestion de bibliothèques.
1965	Basic **B**eginner's **a**ll-purpose **s**ymbolic **i**nstruction **c**ode	Conçu comme un simple langage d'apprentissage, il est aujourd'hui le plus utilisé en micro-informatique. Malheureusement, il n'existe par un standard mais d'innombrables variétés.
1965	PL/1 **P**rogramming **l**anguage	Développé par IBM, il devait être le langage qui rassemblerait les caractéristiques du Fortran, du Cobol et de l'Algol. Il en est résulté un langage extrêmement complexe et excessivement étendu. On le trouve uniquement sur les machines IBM (peu utilisé).
1967	APL **A p**rogramming **l**anguage	Regroupe une vaste panoplie d'opérations mathématiques. L'usager peut, en plus, définir ses propres opérateurs. Les programmes APL sont très concis mais impénétrables.
1971	Pascal	Considéré comme un instrument pour l'enseignement de la programmation structurée. Il intègre les structures de contrôle de cette programmation, très efficaces dans la rédaction des programmes.
1980	Ada	Langage temps réel, utilisable pour le contrôle de processus. Il intègre de nouvelles idées sur les structure modulaires et les compilations séparées pour assurer le développement de grands projets.

La programmation structurée

La programmation structurée est fondée sur un concept essentiel : quelle que soit la tâche à programmer, il est toujours possible de la décrire dans un ensemble d'unités fondamentales, en recourant au langage naturel.
La seconde étape consiste à décomposer chaque unité en plusieurs sous-unités et ainsi de suite, jusqu'à arriver au niveau d'abstraction le plus élémentaire.
Ce travail d'affinement débouche sur un ensemble d'instructions interprétables par un calculateur, qu'elles appartiennent à un langage de haut niveau comme le Fortran ou qu'il s'agisse d'instructions en langage machine.
Ainsi, pour lancer la tâche de « préparer un repas », on peut isoler la suite des opérations fondamentales : acheter les ingrédients nécessaires/préparer les plats/servir. Chacun de ces blocs est ensuite décomposé en une succession de petites tâches. La préparation des plats pourra être ultérieurement spécifiée comme un travail sur un ensemble de services (entrée, plat, garniture, dessert, fruits). La décomposition d'un bloc au niveau du détail nous amènera au stade des opérations élémentaires à accomplir, comme mettre une casserole d'eau sur le feu et attendre l'ébullition.
Il est important d'utiliser, pour chaque niveau d'abstraction, des **blocs connexes de manière simple,** et conformément au schéma général adopté. Principe important : un bloc n'a **qu'un point de départ** (entrée) et **un point final** (sortie). De cette manière, on gagne nettement en clarté sur les inter-relations des divers blocs.
La programmation structurée peut donc être entendue comme un ensemble de techniques de planification et de codification qui guident le programmeur dans la préparation de programmes organisés, autodocumentés et vraiment fiables.

L'organisation des programmes

La technique consistant à partir du problème général et à le subdiviser en sous-problèmes plus petits, à travers des pas successifs, est appelée **top-down** (du haut vers le bas). La seconde technique, **bottom-up** (du bas vers le haut), a une approche inverse : commencer à partir du niveau de détail le plus élémentaire en constituant de petites unités, qui, à leur tour, formeront des blocs plus complexes. Dans le cas de la préparation du repas, résoudre le problème avec la technique bottom-up consisterait à préparer un plat particulier cherchant les ingrédients qui sont donc nécessaires et à composer les différents plats jusqu'au repas complet.
Le développement de programmes comporte deux phases en méthode structurée : **planification et implémentation .** Pendant la planification, la technique de développement la plus usitée est le top-down alors que dans l'implémentation c'est le bottom-up.
En fait, lors de l'analyse d'un problème, le système top-down est sans aucun doute irremplaçable. Par contre, dans le cas de la synthèse (nécessaire pour transférer le programme sur une machine ou dans un langage particulier) la meilleure approche est le bottom-up. En général, pour développer un programme de manière ordonnée, quatre niveaux d'abstraction au moins seront nécessaires avant d'arriver au détail dans le langage de programmation choisi :

1 / objectif à atteindre
2 / remarques détaillées
3 / pseudocode
4 / code

Lors de l'écriture d'un programme, un test simple permettra de vérifier la validité de l'organisation : décrire par des phrases le résultat visé. Si ce pas s'avère trop difficile, c'est probablement parce que la finalité du programme n'apparaît pas encore assez clairement ; on doit alors procéder à un nouveau contrôle des descriptions détaillées.
Il convient ensuite d'examiner directement si ce qui a été décrit est réalisable avec les ressources disponibles, qu'elles soient matérielles ou logicielles, par exemple, en vérifiant que l'ordinateur disponible est bien adapté en termes de rapidité et de capacité mémoire. Une fois l'objectif défini de manière satisfaisante, on procède à la mise au point des notes détaillées. Celle-ci peuvent être subdivisées en au moins trois parties :

1 / données d'entrée
2 / calculs à effectuer
3 / données de sortie

La description complète des données d'entrée et de sortie facilite la phase intermédiaire, c'est-à-dire la description des actions à accomplir à l'intérieur du programme. Cette description (calculs à effectuer) faite en langage naturel aide à faire ressortir d'éventuelles lacunes. Dans la rédaction du pseudocode, les blocs d'entrée, de calcul et de sortie précédemment définis peuvent être segmentés en d'autres blocs dans lesquels sont utilisées des structures contenant des actions précises. Le pseudocode ne définit pas le langage final d'écriture du programme ; cependant, l'emploi de structures du type Algol ou Pascal aide à rendre plus clair et lisible ce que l'on désire réaliser.
Un test important peut être effectué au terme de la préparation du pseudocode : faire lire le programme à une personne extérieure au projet. On est sûrement sur la bonne voie pour obtenir un programme clair et facile à manipuler si l'objectif visé est correctement interprété.
La dernière étape consiste alors à traduire le pseudocode en code, en utilisant le langage le plus adapté à cette application. Certains langages, comme le Pascal, rendent cette opération de traduction très facile puisqu'ils définissent des procédures, fonctions et variables booléennes et permettent, en outre, d'ajouter des commentaires afin de rendre le code semblable au pseudocode descriptif. A ce niveau, il est conseillé de créer de véritables bibliothèques de procédures ou des fonctions qui, le cas échéant, seraient rappelées en divers points du programme.
Si le processus s'est correctement déroulé, le programme principal (main) sera exclusivement formé de quelques appels de procédure ou de fonctions qui, à leur tour, contiendront d'autres procédures, et ainsi de suite... jusqu'à ce que l'on arrive aux blocs finaux (qui ne contiendront que peu de lignes de code).
Le programme ainsi obtenu n'aura pas besoin d'une documentation ultérieure. En effet, la description du but du programme (pas 1) est un excellent résumé de ses caractéristiques ; la définition des notes détaillées en donne une bonne description technique et un pseudocode bien présenté remplace l'organigramme, devenu inutile.

Le pseudocode

Pour illustrer la façon d'obtenir un pseudocode et sa forme, prenons le problème suivant : lire un nombre entier non négatif ; si le nombre est égal à zéro, le programme est terminé, sinon il doit calculer et éditer séparément les sommes de tous les nombres pairs et impairs lus.
Première tâche : il faut définir clairement le but du programme. Le problème est ainsi résumé : supposons une valeur entière positive ; on analyse si ce nombre est zéro, pair ou impair. Si le nombre lu est zéro, le programme s'achève par l'impression :

— du nombre des nombres pairs introduits,
— de leur somme,
— du nombre des nombres impairs introduits,
— de leur somme.

Si le nombre n'est pas zéro, le compteur et la somme du groupe correspondant sont mis à jour et une nouvelle donnée est ensuite demandée. Cela pour le but.
Nous passons, ensuite, aux notes détaillées du programme, en examinant quelles doivent être les données d'entrée, les calcules demandés et les données de sortie.

Données d'entrée Un nombre entier, de valeur comprise entre zéro et la valeur maximum (entière), pouvant être représenté sur le calculateur (32767 pour une machine à 16 bits).

Elaboration Si le nombre vaut zéro, imprimer les résultats et terminer le calcul, sinon identifier si le nombre est pair ou impair (en le divisant par deux et en contrôlant si le reste vaut zéro). En fonction du test, on augmente le compteur des nombres et on effectue la somme du groupe correspondant. Le calcul se poursuit par la demande d'une nouvelle donnée.

Données de sortie Impression du nombre des nombres impairs émis et de leur somme.

Dans cette phase, nous avons défini les données d'entrée, les calculs nécessaires et les données de sortie.
L'écriture du pseudocode (niveau 3) entre dans le détail du problème en utilisant le langage naturel.
Dans le tableau ci-contre, remarquons la manière non habituelle d'écrire les variables dont le nom est constitué de deux parties reliées par un trait d'union (NOM-IMPA)
Cette symbolique permet d'utiliser une première partie commune et une seconde spécifique. Par exemple, les deux variables NOM-IMPA et NON-PAIR se réfèrent toutes deux à des valeurs numériques (NOM) mais l'un aux impairs (IMPA), l'autre aux pairs (PAIR).
Il ne s'agit cependant que d'une méthode d'identification et non d'une règle.

Les organigrammes structurés

Le but du pseudocode est analogue à celui de l'organigramme (figure de la page 1208) et il est facile de suivre le parcours du programme en utilisant l'indentation des écritures *. Avec des phrases explicites comme « début-fin »,« tant-que-faire », « si-alors-sinon » on identifie mieux les blocs du programme.
Le pseudocode sera ultérieurement affiné par extension de la partie chargée de déterminer si le nombre est pair ou impair. Normalement, ceci n'est pas nécessaire puisqu'on peut aller directement à la phase de détail. Cette solution permet même d'améliorer le programme (voir pages 1209, 1210 et 1211, le programme écrit en Fortran 77). Le code présenté ne résout pas complètement le problème. On pourrait, par exemple, introduire un traitement des erreurs en données d'entrée. Mais les modifications sont aisées étant donné la modularité du programme.

En programmation structurée, toute représentation graphique est, en principe, évitée. Pourtant Nossi et Shneiderman proposent l'**organigramme N-S** (voir p. 1212) qui met en évidence l'articulation des structures (ce qui est moins évident dans un organigramme normal). Pour cette raison, les organigrammes N-S sont définis comme structurés.

* On entend par indentation les marges des diverses phrases de description d'un pseudocode.

Exemple de pseudocode

```
Définitions constantes
    MAXINT      nombre entier maximum
                accepté par la machine
    ZERO        valeur zéro

Définitions variables et leur type
    NOMBRE      variable contenant le
                nombre lu, de type entier
    NOM-PAIR    nombre des valeurs pai-
                res lues, de type entier
    NOM-IMPA    nombre des valeurs im-
                paires lues, de type
                entier
    SOMME-PAIR  somme des valeurs pai-
                res lues, de type entier
    SOMME-IMPA  somme des valeurs im-
                paires lues, de type
                entier

Début
  Initialiser à zéro les variables NOM-PAIR,
  NOM-IMPA, SOMME-PAIR, SOMME-IMPA
  Lire première valeur d'entrée —→ NOMBRE.
  Tant que NOMBRE est différent de ZERO
  faire :
    Début
      Si NOMBRE pair
      alors
        Début
          Incrémenter NOM-PAIR
          Somme NOMBRE à SOMME-PAIR
        Fin
      Sinon
        Début
          Incrémenter NOM-IMPA
          Somme NOMBRE à SOMME-IMPA
        Fin
      Lire —→ NOMBRE
    Fin
  Un zéro lu, imprime NOM-PAIR, SOMME-
  PAIR, NOM-IMPA, SOMME-IMPA
Fin
```

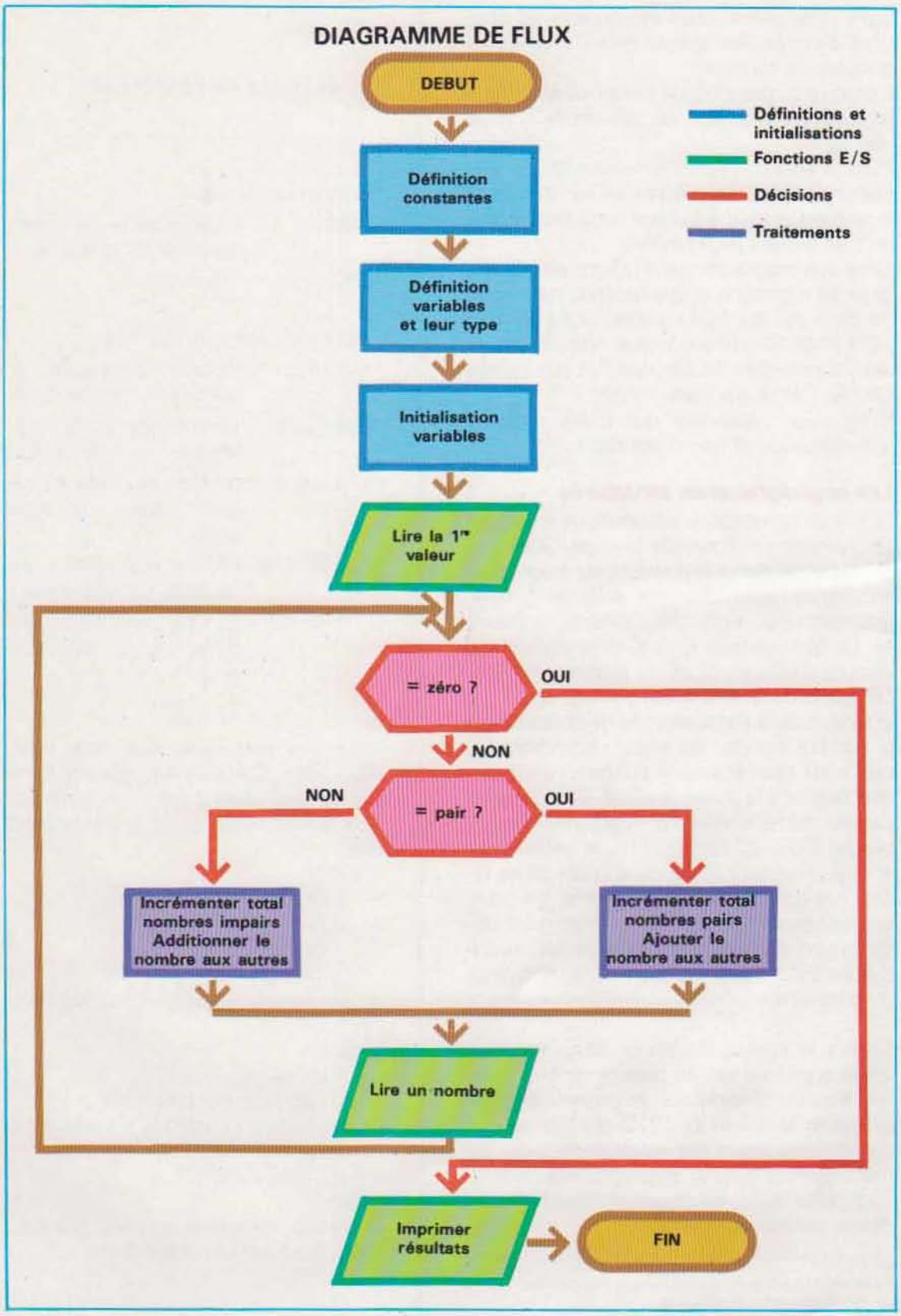


DIAGRAMME DE FLUX
DEBUT
Définition constantes
Définition variables et leur type
Initialisation variables
Lire la 1re valeur
= zéro ?
OUI
NON
= pair ?
NON
OUI
Incrémenter total nombres impairs Additionner le nombre aux autres
Incrémenter total nombres pairs Ajouter le nombre aux autres
Lire un nombre
Imprimer résultats
FIN
Définitions et initialisations
Fonctions E/S
Décisions
Traitements

N.B. La codification en FORTRAN ANSI 77 ne permet pas d'utiliser le tiret à l'intérieur des noms de variables ; NOM-IMPA est donc codifié NOMIMPA

EXEMPLE DE PROGRAMMATION STRUCTUREE EN FORTRAN

PROGRAMMEUR JACQUES RIME DATE 31 MAI 84

FORTRAN

PROGRAMME PAIR OU IMPAIR ?


```
      PROGRAM  PAIRIMPAIR
      IMPLICIT NOM
C     DEFINITION VARIABLES ET TYPE
C     NOMBRE --> VALEUR LUE                          ENTIER
C     NOMPAIR --> NOMBRE DES VALEURS PAIRES          ENTIER
C     NOMIMPA --> NOMBRE DES VALEURS IMPAIRES        ENTIER
C     SOMMEPAIR --> SOMME DES VALEURS PAIRES         ENTIER
C     SOMMEIMPA --> SOMME DES VALEURS IMPAIRES ENTIER
C
      INTEGER*2  NOMBRE, NOMPAIR, NOMIMPA, SOMMEPAIR, SOMMEIMPA
C
C        MISE A ZERO DES VARIABLES
         NOMPAIR=Ø
         NOMIMPA=Ø
         SOMMEPAIR=Ø
         SOMMEIMPA=Ø
C
C        LIRE UN NOMBRE
         PRINT *,' IMPRIMER UN NOMBRE ENTIER POSITIF OU NUL'
         READ  *, NOMBRE
```

1210

PROGRAMMEUR JACQUES RIME DATE 31 MAI 84 FORTRAN

PROGRAMME PAIR OU IMPAIR ? Page 2 de 3

```
C     TANT QUE LE NOMBRE N'EST PAS EGAL A ZERO FAIRE
C
      DO WHILE (NOMBRE.NE.Ø)
         PRINT *,'LE NOMBRE VAUT',NOMBRE
C
C        SI LA VALEUR EST PAIRE: ANALYSE AVEC LA FONCTION ONE MOD QUI
C                                DONNE LE RESTE D'UNE DIVISION ENTRE
C                                ENTIERS
         IF (MOD(NOMBRE,2).EQ.Ø) THEN
C
C        SI LE RESTE DE LA DIVISION ENTRE LA VALEUR ET 2 VAUT ZERO
C                                ALORS LE NOMBRE EST PAIR
C
            NOMPAIR=NOMPAIR + 1
            SOMMEPAIR=SOMMEPAIR + NOMBRE
C
C                                SINON LE NOMBRE EST IMPAIR
C
         ELSE
            NOMIMPA=NOMIMPA + 1
            SOMMEIMPA=SOMMEIMPA + NOMBRE
```

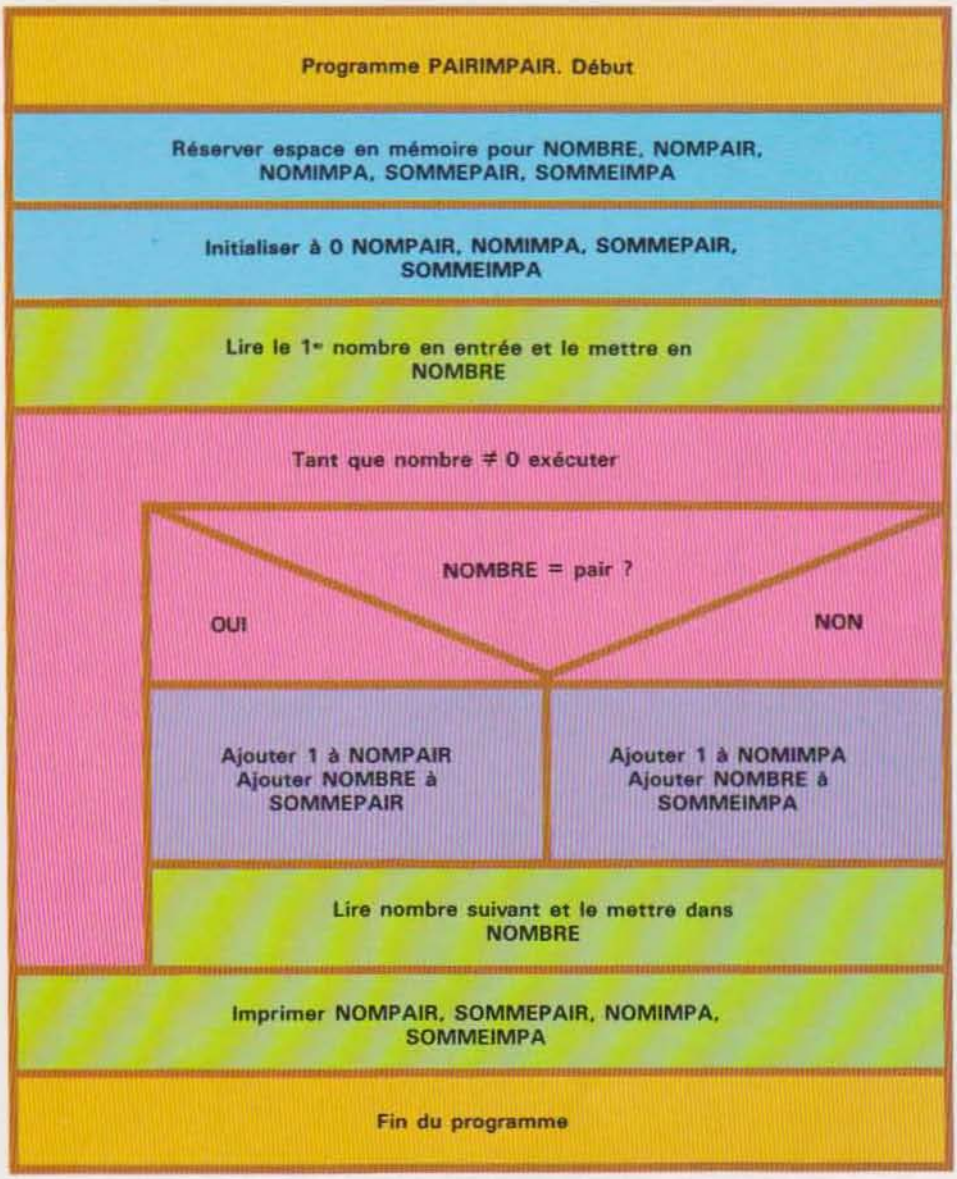
PROGRAMMEUR JACQUES RIME DATE 31 MAI 84 FORTRAN

PROGRAMME PAIR OU IMPAIR ? Page 3 de 3

```
            ENDIF
C
            PRINT *,'TAPER UN AUTRE NOMBRE OU ZERO (Ø) POUR SORTIR'
            READ (*,END=99) NOMBRE
C
         END DO
C
C        LIRE UN ZERO, IMPRIMER RESULTATS
C
   99    PRINT *, 'NOMBRE DES VALEURS PAIRES=',NOMPAIR
         PRINT *, 'SOMME DES VALEURS PAIRES=',SOMMEPAIR
         PRINT *, 'NOMBRE DES VALEURS IMPAIRES=',NOMIMPA
         PRINT *, 'SOMME DES VALEURS IMPAIRES=',SOMMEIMPA
C
         STOP
C
      END
C
C     FIN DU PROGRAMME PAIR OU IMPAIR
```

1211

ORGANIGRAMME "STRUCTURE" (DIAGRAMME N-S)



Dans la figure ci-dessus, notons bien que chaque bloc est représenté par un symbole particulier ainsi que les structures « tant-que-faire » et « si-alors-sinon ».
Par la suite, nous analyserons, pour les deux types de représentation, les organigrammes représentatifs des opérations fondamentales qui composent un programme.

Les structures de contrôle

Les opérations effectuées à l'intérieur d'un programme peuvent se combiner entre elles autant de fois qu'il est nécessaire, grâce à trois structures fondamentales :

1 / Séquence
2 / Sélection
3 / Répétition

Séquence. C'est la structure de base qui revient le plus souvent dans un programme. Elle consiste (voir ci-dessous) en une succession d'opérations à effectuer l'une après l'autre. Le niveau de détail, avec lequel les contenus des blocs sont décrits, dépend du niveau d'affinement que l'on désire atteindre.
Dans certains langages comme l'Algol et le Pascal, le début et la fin d'une séquence sont mis en évidence par les deux mots clés BEGIN et END. Dans d'autres cas, le début et la fin d'une séquence sont identifiés de manière implicite puisqu'elle est enfermée entre deux clés appartenant à une structure différente.

Sélection. Deux séquences sont **combinées en sélection** quand, selon qu'une condition se trouve vérifiée ou non, l'une ou l'autre d'entre elles est exécutée. Par exemple, page 1214, dans les organigrammes de droite, la séquence S1 ou la séquence S2 est exécutée en fonction du résultat.
On trouve, dans presque tous les langages de programmation, la phrase IF-THEN-ELSE, c'est-à-dire SI-ALORS-SINON.
En Fortran 77, tout comme dans le programme précédent, il existe plusieurs instructions insérées entre THEN et ELSE. On peut donc trouver une séquence soit entre les mots clés

STRUCTURE DE CONTROLE DE TYPE "SEQUENCE"

Diagramme ordinaire

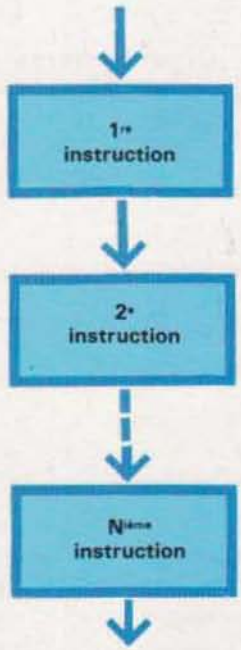
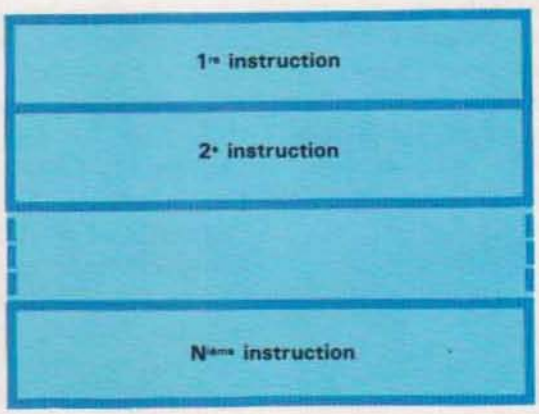
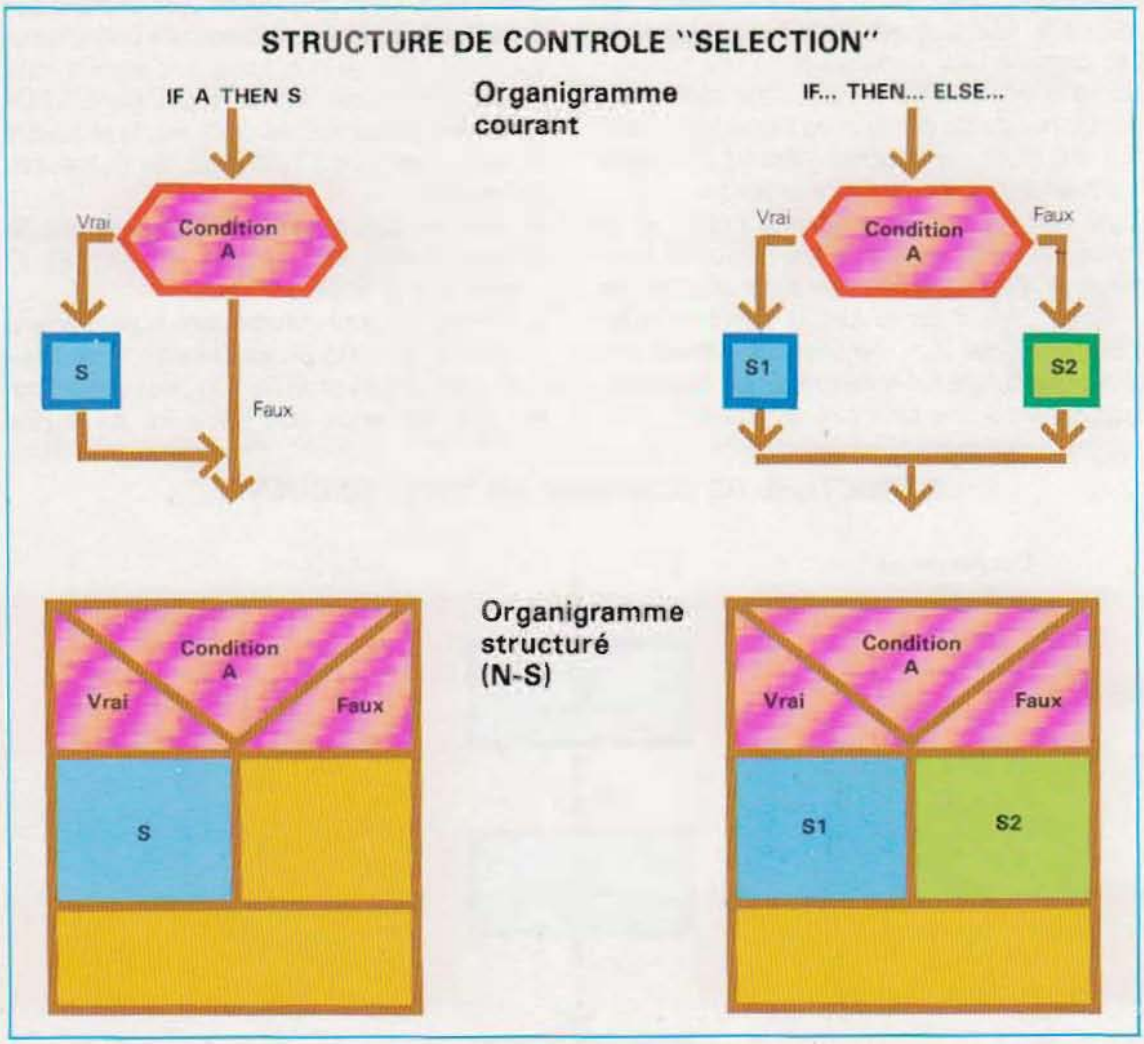


Diagramme structuré





THEN et ELSE, soit entre le ELSE et la fin de la sélection (mot clé ENDIF). Mais une séquence peut également ne contenir aucune instruction sur le mot clé ELSE. Elle signifie : « si la condition n'est pas vraie, poursuivre en séquence sans réaliser aucune opération ».
Ne sont généralement admises, dans le Cobol, que les séquences formées par une seule instruction, alors qu'en Pascal et en Algol, la série de plusieurs instructions est toujours précédée du mot-clé BEGIN et fermée par le mot-clé END. Ces deux clés jouent un rôle analogue à celui des parenthèses.
Ainsi, les structures :

```
IF condition THEN
   Instruction 1
Instruction 2
```

et les structures :

```
IF condition THEN
   BEGIN
      Instruction 1
      Instruction 2
   END
```

sont totalement différentes.
Dans le premier cas : si la condition est vraie, seule la première instruction est exécutée. Dans le second cas : tout ce qui est entre les délimitateurs BEGIN et END est exécuté.

Répétition. Les structures de répétition sont fondamentales lorsque l'opération globale à effectuer est constituée d'une succession finie, mais pas toujours connue a priori, d'opérations égales. Avec cette technique, il est possible d'utiliser plusieurs fois une même série d'instructions. Il existe deux structures fondamen-

tales de contrôle de répétition : WHILE-DO (TANT QUE-FAIRE) et REPEAT-UNTIL (REPETER-JUSQU'A), illustrées ci-dessous et page suivante. Même si elles paraissent identiques à première vue, elles sont conceptuellement différentes. Dans WHILE-DO, on procède d'abord au test de la condition et la séquence n'est exécutée que si cette condition s'avère vraie. Dans REPEAT-UNTIL, c'est l'inverse : est d'abord exécutée la séquence S, puis seulement le test ; si la condition s'avère fausse, la séquence est répétée ; sinon on sort de la structure (boucle).

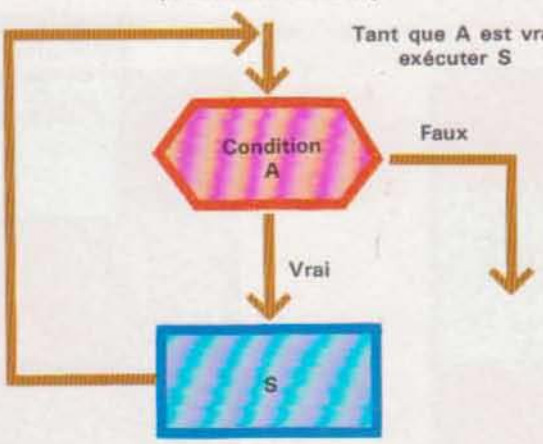
En Pascal et en Algol, si la séquence est composée de plus d'une instruction, les parenthèses BEGIN-END sont nécessaires dans le WHILE-DO alors qu'elle ne le sont dans le REPEAT-UNTIL puisque le début de la séquence est défini par le mot-clé REPEAT et la fin par UNTIL. Au contraire, en Fortran 77, la séquence est refermée entre DO-WHILE et ENDDO. Il n'existe pas, dans ce langage, un véritable format REPEAT-UNTIL qui peut être remplacé par les instructions :

```
S1 Instruction
   IF (.NOT. condition) GOTO S1
```

ou, dans d'autres cas, par un simple DO-ENDDO.

STRUCTURE DE CONTROLE "REPETITION" (WHILE... DO...)

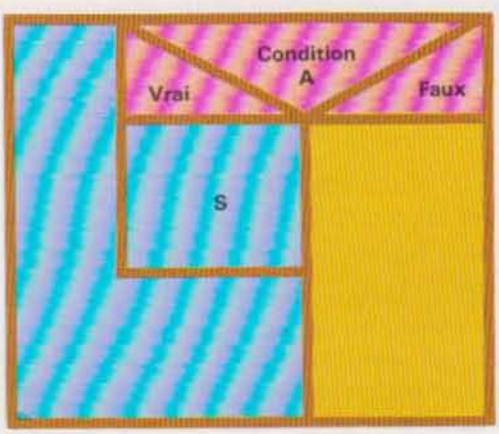
Organigramme ordinaire

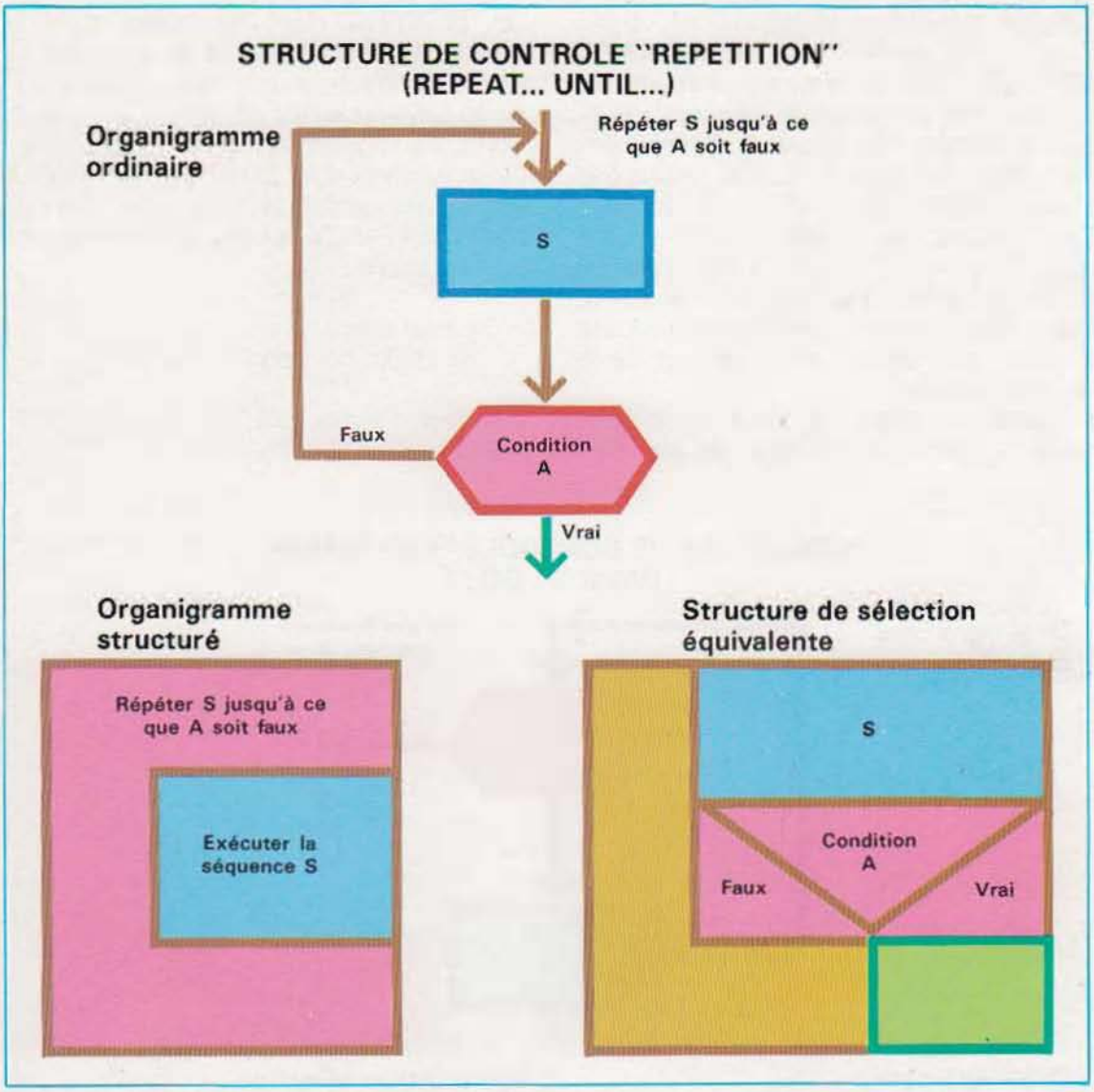


Organigramme structuré



Structure de sélection équivalente





En Cobol, WHILE-DO se déduit ainsi :

```
PERFORM nom-procédure
UNTIL expression
```

de même que pour REPEAT-UNTIL :

```
Début d'expression à FAUX
PERFORM nom-procédure
UNTIL expression
```

sauf si la condition ne peut être mise à faux (cas de fin de fichier par exemple). Toutes ces structures constituent le noyau principal de la programmation structurée et sont suffisantes pour décrire n'importe quel algorithme même s'il en existe d'autres.

Les données en Pascal

Un programme écrit en Pascal est structuré en deux parties : la déclaration des données et l'ensemble des instructions vraies et propres. La première définit les types de données qui seront utilisés par le compilateur et améliore la lisibilité du programme, la seconde contient les instructions qui décrivent la méthode de calcul sur les données précédemment spécifiées.

Types de données

L'innovation la plus significative, introduite par le Pascal, est la formulation du concept de

type de donnée. On dit que des constantes ou des variables appartiennent au même type si elles possèdent les mêmes caractéristiques et si elles sont soumises au calcul de la même manière. Par exemple, l'ensemble des nombres entiers constitue le type de donnée entière.
En Pascal, les types de données élémentaires sont définis comme des types **scalaires.** Ceux-ci ne sont pas décomposables en types plus simples. En revanche, ils peuvent se combiner de manière différente pour former les **types structurés**. Par exemple, un vecteur de nombres entiers de 10 éléments :

IA(I) I = 1,..., 10

est de type structuré formé de dix données scalaires IA(1), IA (2)..., IA(10).
Les types de données scalaires se subdivisent en **types standard** et en **types définis par l'utilisateur.** Les types standard sont des types prédéfinis par le Pascal. Ce sont les types entier, réel, caractère, booléen, utilisables sans spécification du champ de validité. Un tel champ est en effet automatiquement établi par le compilateur Pascal (comme en langage Fortran quand on déclare une variable de type entier ou réel).
En revanche, les types de données définis par l'utilisateur sont une caractéristique du Pascal qui permet à cet utilisateur de créer des types ad hoc. Il établit le champ de validité et les éléments d'appartenance pour résoudre les problèmes spécifiques.
Les types de données structurées répondent aux types suivants :

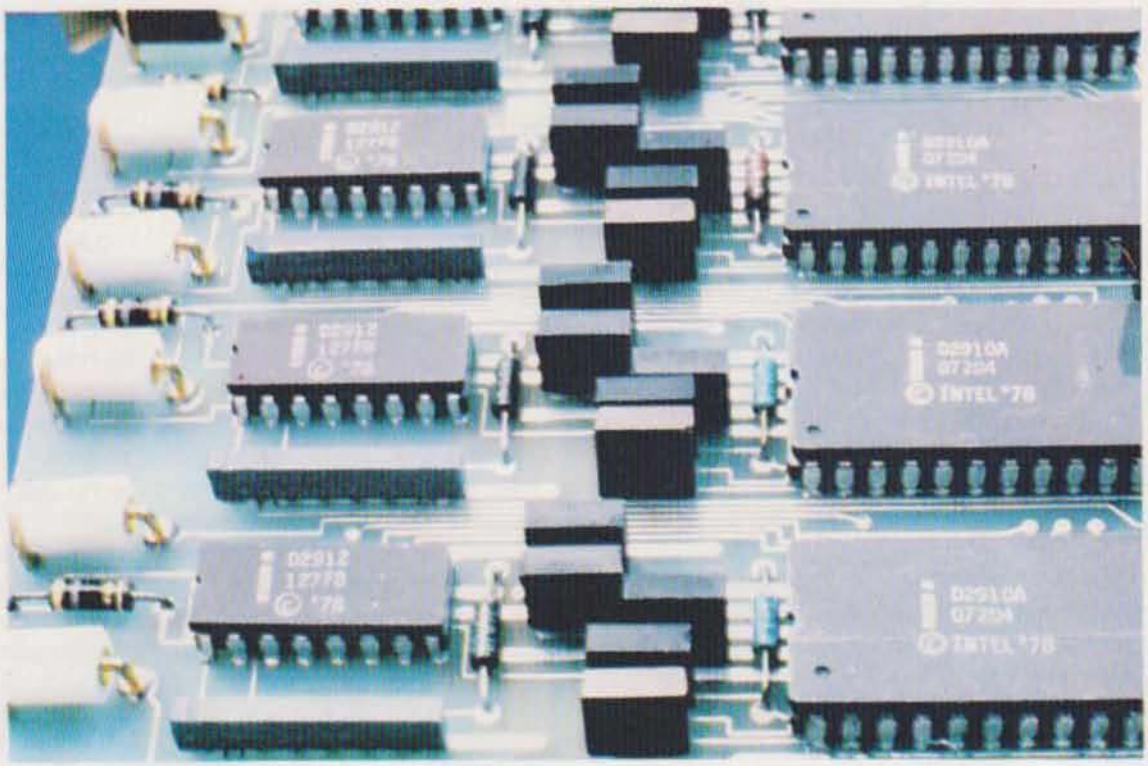
— **array** (tableau), terme générique avec lequel on indique les vecteurs et les matrices à deux ou à plusieurs dimensions,

— **set** (jeu), un ensemble d'éléments tout à fait semblables à l'array qui sont cependant traités comme un élément unique,

— **file** (fichier) qui signifie un ensemble d'enregistrements.

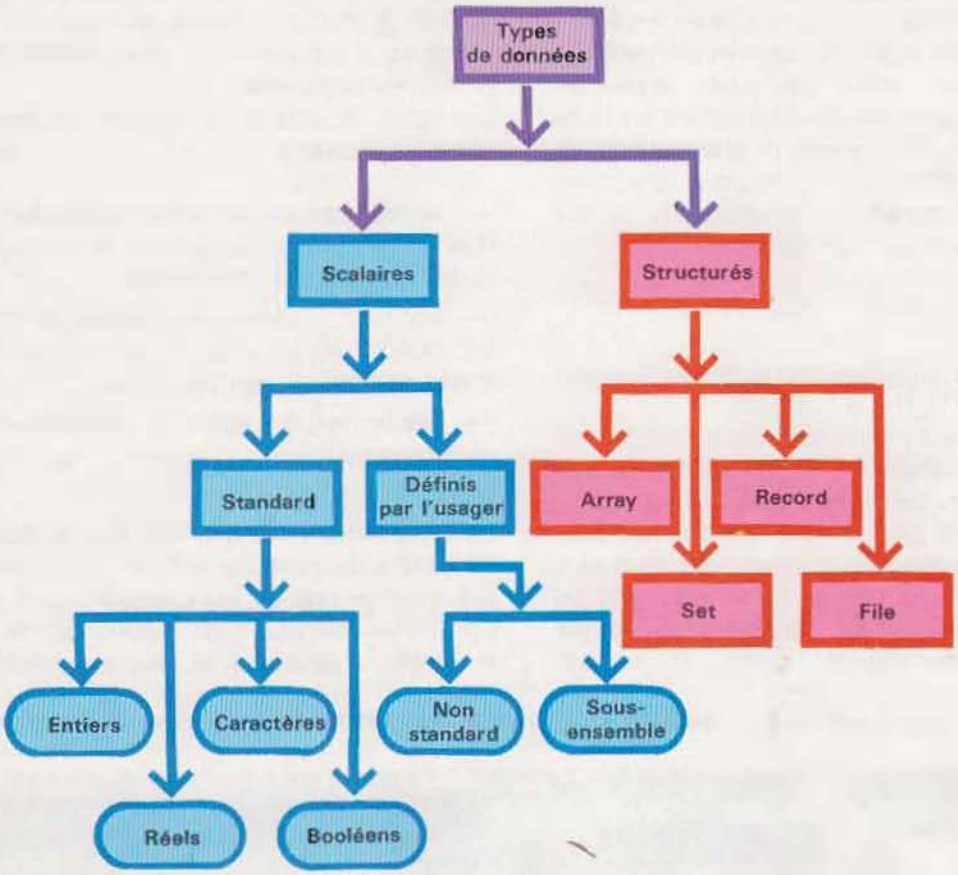
Le type entier. Ces données appartiennent à l'ensemble des nombres naturels (sans virgule) précédés ou non du signe positif ou négatif. Les limites inférieures ou supérieures de cet ensemble, c'est-à-dire le plus petit nombre

Circuits intégrés Intel montés sur la carte-mère d'un système.



J. Pickerell/Marka

TYPE DE DONNEES ADMIS EN PASCAL



négatif et le plus grand nombre positif, dépendent évidemment de la capacité de l'ordinateur utilisé. Certaines machines à mots de 36 bits admettent, pour les nombres entiers, une extension de -2^{35} à $2^{35}-1$, c'est-à-dire de 34 359 738 368 à + 34 359 738 367. Ordinairement on utilise 16 bits pour la représentation des nombres entiers. On a ainsi une extension des valeurs de — 32 768 à + 32 767.

Pour éviter d'amoindrir la capacité de l'ordinateur, notamment lorsque les programmes sont transportés d'une machine à l'autre, une variable entière, appelée MAXINT, est prédéfinie et utilisée par le compilateur Pascal. Cette variable contient le plus grand entier utilisable sur le calculateur dont on se sert. La connaissance de la valeur de cette variable permet de faire abstraction du mot-machine. Dans la page ci-contre est montrée la syntaxe formelle de l'écriture d'une donnée de type entier selon une notation particulière appelée **diagramme syntaxique**. Cette notation, qui sera désormais souvent utilisée, exprime de façon compacte les règles syntaxiques du Pascal.

Comme la notation des organigrammes, elle indique comment construire un type de donnée ou une instruction en combinant, dans les divers champs, les caractères alphanumériques et les symboles de séparation admis. Selon ce type de syntaxe, toutes ces propositions (nombres et instructions) qui peuvent être pro-

duites (en partant à gauche du diagramme, en suivant les flèches et en sortant à droite) s'avèrent valables. Par exemple, étant donné le diagramme syntaxique qui se trouve en bas de page, les nombres :

1
1 2 3
1 2 1
1 2 1 2 3

sont syntaxiquement valables.

En revanche le nombre :

1 3

n'est pas correct puisqu'il n'existe aucun chemin allant de gauche à droite en passant seulement par les nombres 1 et 3 sur le diagramme.

De même, les nombres :

1 2
2 3
1 2 1 3
1 2 3 3

ne sont pas syntaxiquement valables.

Revenons au diagramme « de type entier » : observons que le type de donnée entier est constitué par un nombre de chiffres décimaux qu'on peut prolonger (dans les limites, évidemment, des capacités de l'ordinateur). L'ensemble des chiffres est précédé d'un symbole + ou —, ou d'aucun symbole, indépendamment du chemin choisi sur les trois branches du diagramme. On pourra donc obtenir :

16384
— 14
+ 250
0

Voici les opérations sur les nombres entiers admises par Pascal :

1 / Opérations arithmétiques (addition, soustraction, multiplication)

2 / Opérations de division DIV et MOD, avec les significations suivantes :

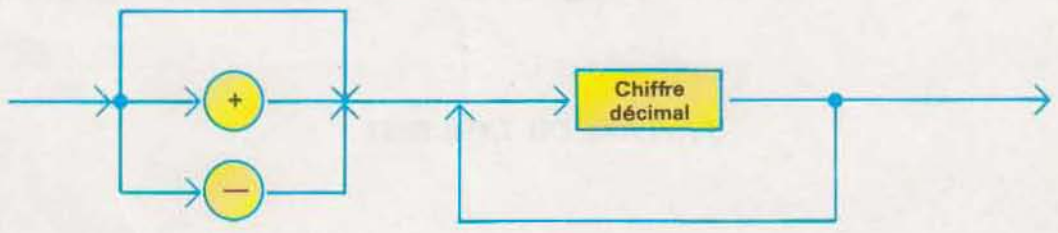
DIV division entre entiers, sans la partie fractionnaire ; par exemple, 29 DIV 8 = 3.

MOD reste de la division entre entiers ; par exemple 29 MOD 8 = 5

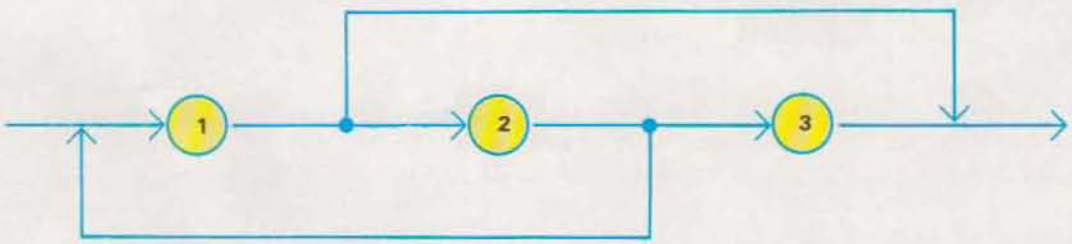
3 / Opérations relationnelles

> supérieure
= égal
< inférieur
>= supérieur ou égal
<= inférieur ou égal
<> non égal (différent)

SYNTAXE DU TYPE ENTIER



EXEMPLE DE DIAGRAMME SYNTAXIQUE



Contrairement aux résultats des deux premiers groupes d'opérations, celui d'une opération relationnelle entre deux nombres entiers n'est pas un entier mais une variable booléenne avec une valeur « vrai » ou « faux ».
Il existe enfin certaines fonctions opérant aussi bien sur des entiers que sur des réels et qui fournissent un résultat entier. Ce sont :

ABS (I)	fournit la valeur absolue de l'entier 1, c'est-à-dire le nombre 1 sans signe
SQR (I)	élévation au carré du nombre 1
ROUND (R)	fournit le nombre entier le plus proche du réel R ; en notation américaine ROUD (4.87) = 5
TRUNC (R)	fournit la partie entière de R ; par exemple TRUNC (4.87) = 4

Le type réel. Ce type se réfère à l'ensemble des nombres ayant une partie entière et une partie fractionnaire.
Ces nombres peuvent être représentés :

— dans la **notation décimale** ordinaire,
— dans la **notation exponentielle** ou scientifique.

Celle-ci est constituée d'une mantisse (nombre décimal avec une partie entière d'au moins un chiffre) suivie de la lettre E et de la puissance entière de 10 qui, multipliée par la mantisse fournit la valeur du nombre réel (d'une manière tout à fait analogie au Fortran et au Basic). Par exemple, le nombre 1.237E03 équivaut à 1.237×10^3, c'est-à-dire à 1237.
Voir ci-dessous la syntaxe de la représentation des données réelles.
On peut contrôler, en suivant le diagramme syntaxique, la validité des nombres :

— 32.1
+ 8.3E—9
— 42.0
— 6.9E—19

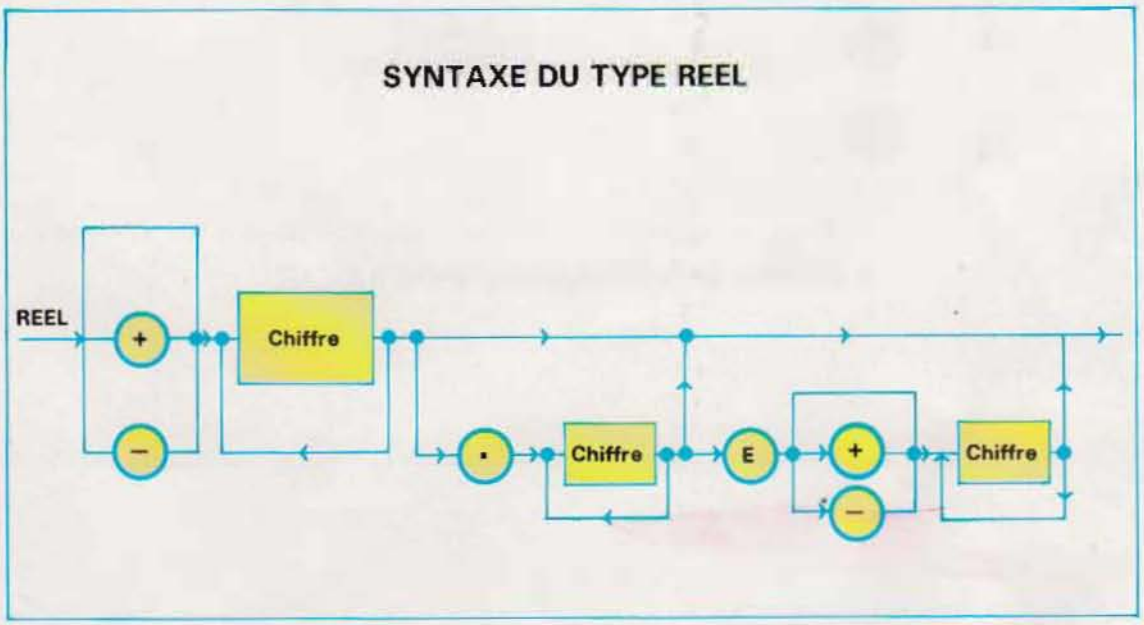
et la non-validité des nombres :

.027	(aucun chiffre avant le point décimal)
8.57E+.2	(la puissance décimale n'est pas admise)

Pour les réels, comme les entiers, il existe des limitations sur l'ensemble des nombres représentables. Ces limitations concernent aussi bien la précision (le nombre des chiffres mais après le point décimal) que les valeurs extrêmes représentables.

Pour un micro-ordinateur, les valeurs réelles négatives ont comme intervalle celui délimité par — 1.0E + 38 et — 1.0E — 39, alors que

SYNTAXE DU TYPE REEL



SYNTAXE DU TYPE CARACTERE



les données réelles positives peuvent être comprises entre + 1.0E — 39 et + 1.0E + 38. Les opérations admises sur les réels sont :

1 / les quatre opérations algébriques +, —, *, /

2 / les opérations relationnelles de manière tout à fait analogue à ce qui a été vu pour les nombres entiers.

De plus, sont admises les fonctions suivantes dont le résultat est un nombre réel :

ABS(R)	valeur absolue du nombre R
SQR(R)	carré de R
SQRT(R)	racine carrée de R
LN(R)	logarithme en base e de R
EXP(R)	e élevé à R
SIN(R)	sinus de R
COS(R)	cosinus de R
ARCTAN(R)	arctangente de R

Le type caractère. En Pascal, le type caractère s'étend à tous les symboles alphanumériques et de ponctuation connus de l'ordinateur. A cause des différences entre les symboles de chaque clavier, l'ensemble des données de type caractère n'est pas toujours le même. On doit en tenir compte lorsque l'on transporte les programmes d'un ordinateur à un autre.
Cet ensemble minimum contient 26 lettres majuscules, de A à Z (l'alphabet entier), les chiffres de 0 à 9 et les caractères spéciaux qui indiquent la ponctuation et les opérateurs, en plus de l'espace. La syntaxe d'écriture des données de type caractère est montrée dans le schéma ci-dessus. Comme on peut le voir, une donnée caractère est toujours enfermée entre deux apostrophes, comme dans les exemples suivants :

'P'
'7'
' '

Le type caractère ne doit pas être confondu avec le type chaîne, lequel, comme ensemble de caractères, représente une donnée de type structuré (c'est-à-dire composé d'un ensemble de données scalaires). Par exemple, la chaîne

'PROGRAMME'

est composée des caractères simples 'P' 'R' 'O' 'G'' R' 'A' 'M' 'M' 'E'.
Le type structuré chaîne sera décrit de manière détaillée plus loin. Les fonctions opérant sur les données de type caractère sont :

CHR (I)	Restitue le caractère correspondant au nombre I dans la codification interne des caractères dans l'ordinateur [en général un nombre entier compris entre 0 et 128 (ou 255)].
ORD ('C')	C'est la fonction inverse de CHR (I). Il restitue un entier correspondant à la représentation interne du caractère C.
PRED ('C')	Fonction de prédécesseur. Restitue le caractère précédant immédiatement le caractère C dans le jeu des caractères. Par exemple, PRED 'B' est égal à 'A'.
SUCC ('C')	Fonction de successeur. Restitue le caractère suivant immédiatement le caractère C ; par exemple, SUCC 'B' est égal à 'C'.

Il est intéressant de noter que les deux dernières fonctions s'appliquent aussi à des données de type entier ; par exemple :

PRED (5) = 4
SUCC (5) = 6

Les jeux vidéo : pourquoi ?

Le succès commercial des jeux vidéo est indiscutable. Même si certains considèrent qu'ils ont atteint le stade de la saturation dans des pays comme les Etats-Unis et le Japon, il reste évident que le jeu est désormais une donnée de notre civilisation.

Le jeu a toujours existé, et à toutes les époques. Ce qui est nouveau, avec le phénomène des jeux vidéo, c'est l'interaction entre une ***haute technologie,*** *extrêmement* ***chargée affectivement,*** *et l'usage traditionnel du jeu.*

Ce phénomène aurait pu n'être qu'éphémère si le secteur industriel qui en est à l'origine ne s'était mis soudain à produire à la fois la technologie et les sujets, qui satisfont et motivent constamment notre curiosité et notre soif d'aventures.

A ce propos, on peut se souvenir que les automates ludiques ne sont pas nouveaux. Ceux de Vaucanson sont restés célèbres à juste titre. La différence, par rapport à aujourd'hui, c'est que la technologie ne fournit pas seulement l'instrument, mais aussi le sujet du ***rêve.*** *Elle ne s'adresse pas seulement à notre intellect, mais aussi à notre émotion. C'est pourquoi il naît tant de jeux chaque mois.*

En effet, notre secteur émotif est infiniment plus diversifié et surtout plus changeant que notre secteur intellectuel. Et surtout, il dispose de moins de soupapes de sécurité que l'intellect. On assiste donc à un « détournement » intéressant et curieux de la technologie de pointe qu'est l'électronique/informatique. Ce secteur, réputé rébarbatif et hautement adulte, nourrit en fait une partie très primitive et très proche de ***l'enfance*** *de notre individu !*

Il est à la mode d'affirmer que ces jeux sont mauvais, en arguant de leur pauvreté culturelle et éducative, acceptée et même choisie pour des questions de facilité commerciale. On ajoute aussi qu'ils induisent un comportement passif chez l'individu qui se laisse « prendre au jeu ». En fait, nous retrouvons là l'éternelle querelle des ***modernes*** *et des* ***anciens.***

On a dit bien du mal de la télévision à ses débuts, comme de tout système intégrant largement l'image. On constate aujourd'hui que les systèmes audiovisuels produisent des œuvres de valeur. Qui, aujourd'hui, oserait affirmer que le cinéma n'a produit aucun chef-d'œuvre ? La télévision a produit des émissions de qualité, surtout dans de nouvelles visions de l'actualité. Qui ne se souvient du très célèbre « Cinq colonnes à la une » ?

Un autre aspect du jeu qu'il convient d'aborder est son rapport avec la civilisation qui l'a fait naître. La tournure de cette phrase est fortement affirmative quant à « l'activité » du jeu. Ce n'est pas anodin. Disons qu'une civilisation sécrète ses jeux comme une plante produit des fleurs.

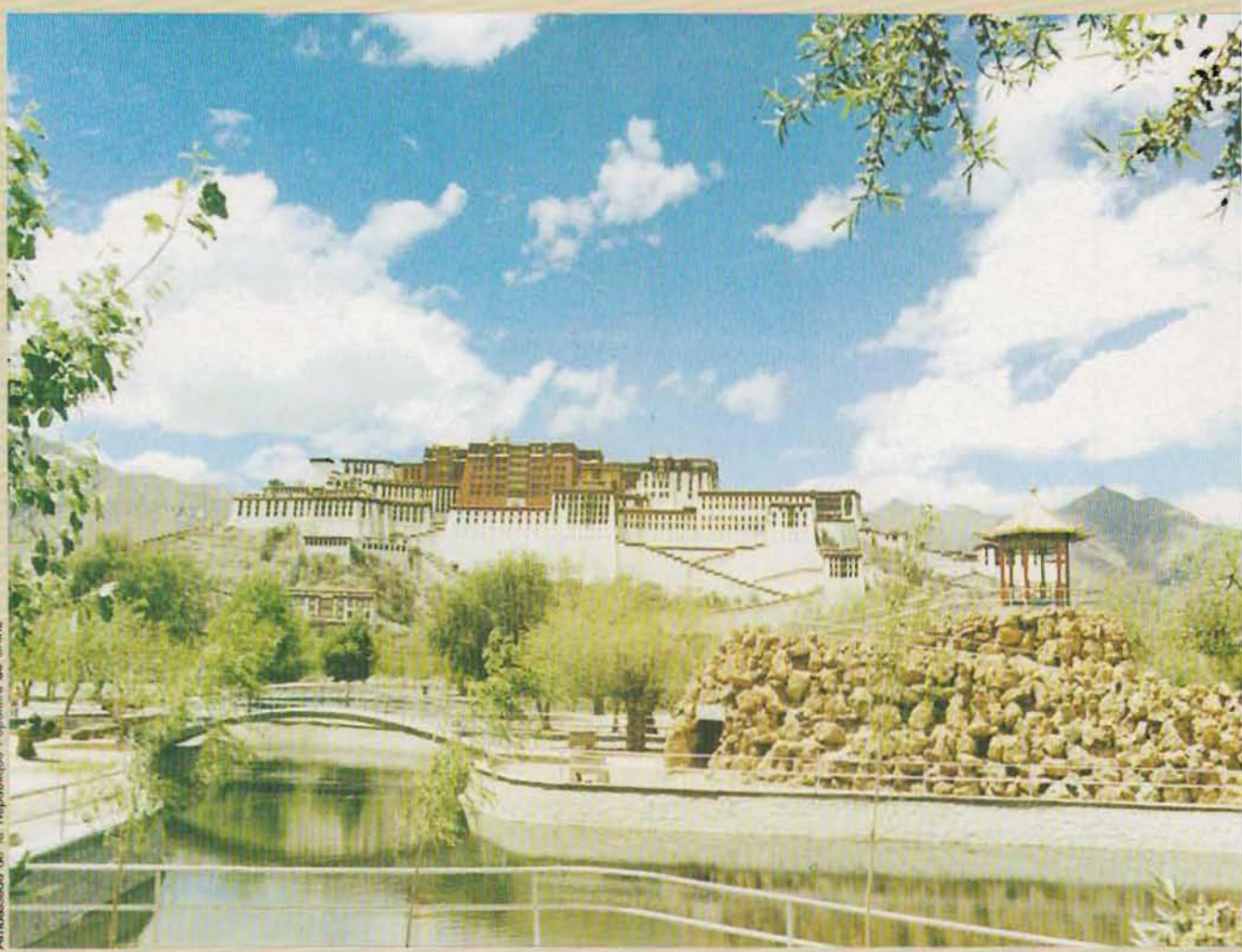
Une civilisation qui a oublié le jeu est une civilisation qui meurt. Elle connaît, quelquefois, des sursauts, mais ceux-ci restent tristement célèbres dans l'histoire, comme les jeux du cirque chez les Romains. Une civilisation qui ne sait plus jouer, ou qui en est réduite à caricaturer la réalité, s'est coupée de toute son enfance, ou tout au moins, de ses forces vives, de sa capacité d'inventer et de se renouveler.

Pensons, à ce sujet, à deux très grands mathématiciens : Poincaré et Einstein. Le premier était un pur produit de son époque et, aujourd'hui, seuls les spécialistes se souviennent de lui. Einstein, lui, a tellement ***rêvé le futur,*** *tout en s'appuyant sur la fine pointe du raisonnement mathématique et physique de son époque, qu'il nous a offert les instruments de conception de notre futur. Il rêvait beaucoup, ce très grand homme, et ses réflexions philosophiques en témoignent. Alors, abordons la question des jeux vidéo avec ouverture d'esprit, avec une saine curiosité et le désir de se mesurer honnêtement à cette émanation de notre actualité.*

Et, surtout, ne trouvons pas de mauvaises raisons pour les condamner, sous prétexte que « tout cela n'est pas sérieux, sauf pour les marchands de soupe » ! Le jeu a toujours existé, en tant que ***nécessité de vie,*** *et la plupart du temps, en tant que* ***simulation d'apprentissage.***

Un des jeux les plus passionnants (et très facilement informatisable), avec les échecs, est le jeu de Gô, dont la pratique quotidienne a été conseillée aux samouraï depuis des siècles.

Les historiens pensent que le jeu de Gô est né au Tibet environ 2 000 ans avant J.C. Il passe ensuite en Chine, en même temps que les pratiques magiques et les recettes d'alchimie, aux alentours du VIe siècle et, enfin, atteint le Japon en 754. Là, il y a prospéré au point de devenir



Ambassade de la République Populaire de Chine

Le Gô, l'un des jeux de guerre qui a conquis le monde, est né dans un monastère tibétain.

partie intégrante de la vie quotidienne de l'aristocratie guerrière.

D'une certaine façon, on peut dire que l'âme de l'ancien Tibet a achevé de s'épanouir au milieu de ces hommes et de ce sol tout autant durs que raffinés.

Ce jeu est porteur à la fois d'une signification morale et philosophique, mais aussi constitue un excellent ***kriegspiel*** *(ou* ***wargame).*** *Ce qui est normal. On n'est vraiment prêt à affronter un adversaire que lorsqu'on a su se mesurer à soi-même. Et dans les vastes plateaux isolés et glacés du Tibet, un tel jeu ne pouvait que s'épanouir car il constituait un moyen privilégié de* ***développement mental.***

Les extrêmes se rejoignant utilement, on conçoit qu'un jeu issu d'un lieu connaissant la solitude physique convienne tout autant à un pays connaissant la surpopulation. C'est que la nécessité d'affrontement (ou de non affrontement) est la même dans les deux cas, l'adversaire est puissant et hautement estimable... et les questions de territoire sont réellement primordiales.

Comme la situation démographique japonaise est en train de s'étendre au monde entier, le jeu de Gô acquiert donc une ***importance mondiale.*** *Si le Japon reste leader en comptant toujours les meilleurs joueurs du monde (dix millions de joueurs, soit 10% de sa population, enfants compris), les clubs de Gô prolifèrent dans les grandes villes et sur les campus universitaires aux Etats-Unis.*

Une des raisons de la séduction du Gô — et certains joueurs d'Echecs se mettent à le préférer à leur habituelle passion — est la ***simpli-***

cité de ses règles** alliée à la **complexité de sa combinatoire.

Les Echecs simulent un tournoi de chevaliers de style médiéval (en fait le jeu remonte à l'ancien Iran et était l'apanage de sa chevalerie), une guerre entre châtelains qui se combattent d'une forteresse à l'autre et qui envoient au combat des guerriers qui succombent à de cruelles blessures.

*Le Gô est plus subtil : la survivance et la victoire ne s'obtiennent pas par l'anéantissement physique de l'adversaire, mais pas la **conquête exprimée en espace vital.** Bref, le combat se fait en terme d'économie politique et de géopolitique. C'est l'hégémonie stratégique qui réduit l'adversaire en appauvrissant ses ressources et sa vitalité.*

Dans le Gô, l'habileté manuelle est réduite au minimum et l'intelligence stimulée au maximum. Le Gô se joue sur un échiquier carré comportant 19 × 19 intersections. L'échiquier japonais, le Gô Ban, comporte 361 pions (181 noirs et 180 blancs) disposés par les joueurs en fonction des croisements des lignes sur la table. La tradition veut que les pions noirs soient fabriqués à base de plaques d'ardoise et les pions blancs sculptés dans la nacre.

Le but du joueur de Gô est de conquérir le plus de territoire possible, la superficie étant fonction des espaces libres. Les joueurs déplacent l'un après l'autre leurs propre pions en fonction de la stratégie adoptée. Les déplacements font inévitablement naître des conflits... Ceux-ci sont régis par des règles précises qui dictent les choix des joueurs. Les pions sont placés un à un sur les déplacements libres, ce choix restant définitif.

*Dans l'intention de dominer (en l'**encerclant**) un grand espace du Gô Ban, le joueur peut se trouver dans la nécessité d'éliminer certains pions de l'adversaire. Pour y parvenir, il doit suivre la procédure de capture, par l'intermédiaire de laquelle il pourra retirer du Gô Ban les autres pions qu'il détient comme prisonniers.*

Les dessins de la page 1225 montrent quelques exemples de zones circonscrites de pions de la même couleur. Les points marqués d'une croix appartiennent respectivement à l'espace du Blanc (A,B) et du Noir (C). A ce stade du jeu, le Blanc a quatre points en A (l'espace dont il s'agit est complètement cerné de pions blancs) et quatre points en B (dans un espace qui est cerné seulement d'un côté par les pions et de l'autre par la limite de l'échiquier). Dans ce dernier cas, le joueur bénéficie de la règle qui permet d'utiliser les côtés extérieurs du Gô Ban dans le but de s'adjoindre l'aide de « pions fantômes ».

Un joueur peut donc augmenter son propre espace en capturant les pions de l'adversaire soit individuellement, soit par groupes.

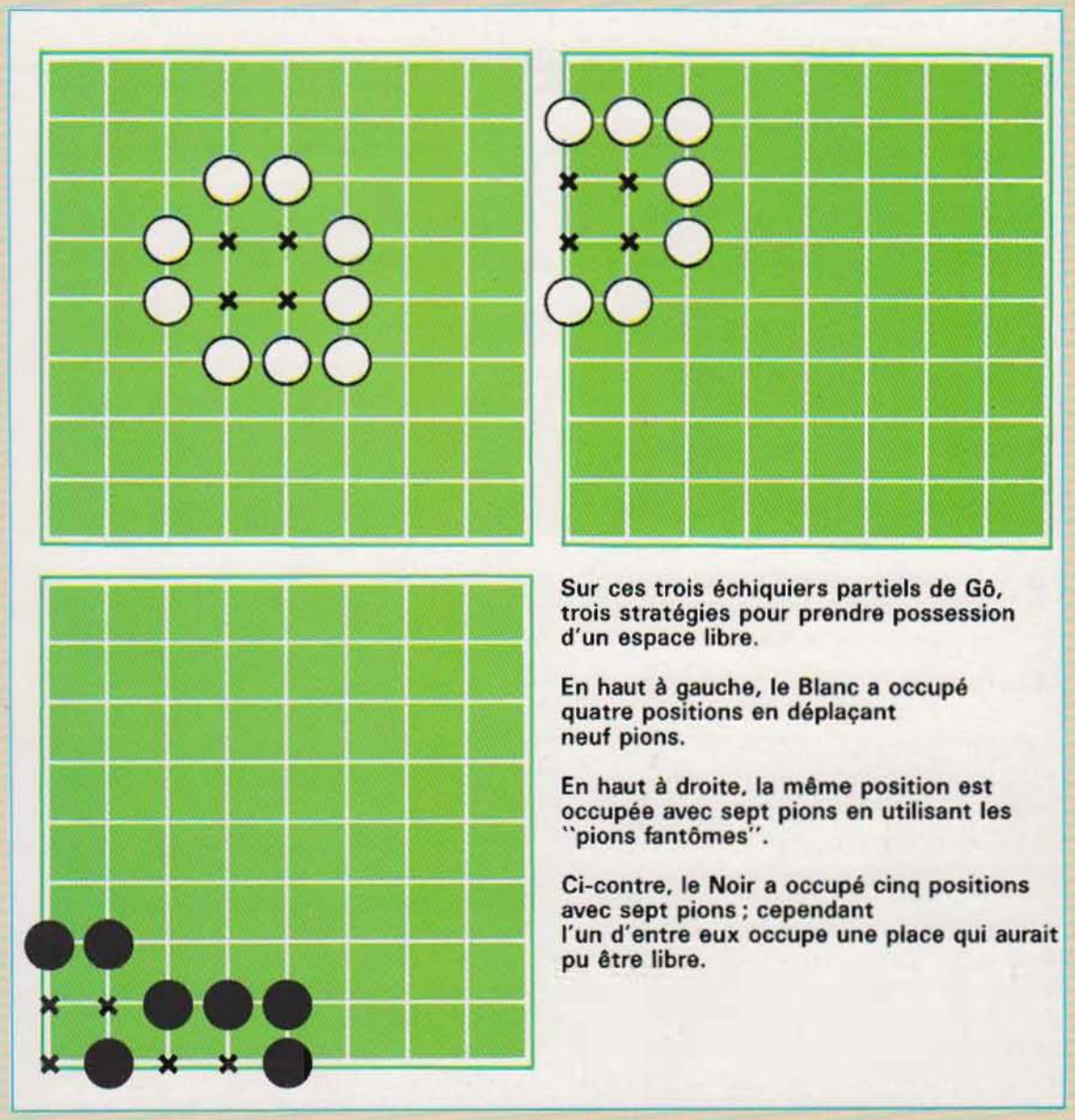
*Ici encore, le Gô se distingue du jeu de Dames par son raffinement : la capture d'un ou de plusieurs pions ne se produit pas par élimination ou par superposition physique. Au contraire : un pion finit dans le butin de l'adversaire lorsqu'il a été **cerné** par les pions de la couleur opposée et que, donc, il n'est **plus libre** (plus de croisement libre dans les lieux adjacents au pion).*

*Pour posséder un espace, il faut le cerner avec une chaîne de pions, tous de la même couleur. Il va de soi que cette configuration présente une importance déterminante : elle est définie comme **groupe de connexion.***

*Une grande partie de la dynamique du jeu et des affrontements entre deux compétiteurs sert justement à empêcher la connexion des pions de l'adversaire en groupes importants, illustrant ainsi l'adage bien connu, selon lequel il convient de **diviser pour régner.***

Toutefois, il est parfois utile de penser « petit », comme, par exemple, de capturer de petits espaces non encore mitoyens entre eux car le but final du Gô est de posséder des territoires et non de capturer des pions. Il est donc inutile de bouleverser le jeu de l'adversaire si l'on manque de pions après ! La capture d'un groupe se produit par la saturation de tous les espaces et par l'obligation qu'on impose à l'adversaire de rester dans des limites les plus petites possibles.

*Le jeu se termine lorsque les deux compétiteurs conviennent qu'il n'existe **plus de motifs** (ni conquête de territoire, ni capture de prisonniers) pour poursuivre les mouvements. Au contraire donc du jeu d'Echecs qui tue ses adversaires, le jeu de Gô est intéressé par ce qui le **fera vivre.** Toute une philosophie profonde, qui illustre aussi bien l'attitude d'une civilisation en face de la vie et de la mort que les moyens par lesquels elle assurera sa survie face aux autres.*



Sur ces trois échiquiers partiels de Gô, trois stratégies pour prendre possession d'un espace libre.

En haut à gauche, le Blanc a occupé quatre positions en déplaçant neuf pions.

En haut à droite, la même position est occupée avec sept pions en utilisant les "pions fantômes".

Ci-contre, le Noir a occupé cinq positions avec sept pions ; cependant l'un d'entre eux occupe une place qui aurait pu être libre.

Maintenant que nous avons étudié, en détail, un wargame classique, le Gô, et avant d'en examiner la version revue par le dernier cri de l'informatique et de l'électronique, accordons nous un moment de réflexion à propos des ***simulations de guerre****.*

Un excellent roman, tout-à-fait récent, ***"Le système Aristote"*** *de René Dzagoyan comporte quelques passages savoureux sur les rapports entre l'être humain et l'ordinateur dans le cadre militaire. Nous nous en servirons pour étayer notre réflexion.*

Tout d'abord un coup de patte à l'adresse de ceux qui confondent le tas de ferraille avec un nouveau Dieu :

"— Peut-être un de nos ordinateurs militaires s'est-il détraqué...

— C'est rigoureusement impossible

— Mais pourquoi donc ?

— Aristote ne peut pas se tromper. Aristote, c'est le système d'ordinateurs qui contrôle les terminaux des sous-marins. Pas seulement les sous-marins, d'ailleurs. Mais encore le reste de la flotte, nos forces nucléaires terrestres, et nos bombardiers, et tout ce que vous pouvez imaginer. Jusqu'au téléphone ! Ou nos chaînes de télévision. Nos aéroports civils, enfin la plupart des choses que vous voyez fonctionner.

LES REGLES DU GO

1) Noir commence en déplaçant, en alternative avec Blanc, un pion à la fois sur le jeu.
Si Noir a le droit de handicap, il déplace tous les points de handicap au début de la partie.
Les pions doivent être posés sur les intersections libres du Gô Ban.

2) Le joueur peut rester à sa place, en "passant" son tour.

3) Gagne celui des deux joueurs qui possède le plus grand nombre d'intersections libres. La partie égale est "pat".

4) Un groupe de connexion ou un pion peuvent être capturés quand ils sont privés de liberté.

5) Un pion prisonnier correspond à une intersection libre en moins dans le compte des points du joueur.

6) Il est interdit de copier une situation existante sur le Gô Ban.

7) Le placement d'un pion dans un point privé de liberté a pour but de capturer des pions de l'adversaire.

8) La partie se termine avec l'accord des deux joueurs.

9) A la fin de la partie, on procède au décompte des pions "morts".

10) Un SEKI ne donne ni points ni prisonniers.

11) A la fin de la partie, il faut fermet chaque KO et toutes les intersections neutres.

Et cela, à travers treize pays...
Mais je crois que nous avons affaire à un phénomène scientifique tout-à-fait incroyable. Si mes hypothèses sont justes, l'histoire du sous-marin (qu'on a cru, tout d'abord, détruit à cause de la défaillance d'un ordinateur local) est le prélude au plus grand bouleversement que l'humanité aura jamais connu."
Vous voilà en haleine : vous vous demandez ce que "Super-Aristote" a encore bien pu inventer... Oh, mais pas si vite. Tout d'abord, essayons de comprendre Aristote :
"— Tout a commencé dans les années soixante. Un professeur d'économie avait eu l'idée géniale, à l'époque, de construire un modèle mathématique représentant l'économie mondiale. Son but était de ***prévoir avec précision les mouvements économiques*** *de la planète. Pour obtenir ce résultat, il a choisi un moyen très simple. Il a installé, dans les sous-sols de l'université, un immense ordinateur (à l'époque un IBM) et il a fourré tout ce qu'il pouvait trouver comme données économiques. Son modèle mathématique se chargeait de mettre en ordre et de faire les calculs nécessaires.*

— Et ça marchait ?
— Mouais... Petit à petit, le système a commencé à marcher. Les résultats s'amélioraient. Au point que les pays de l'OTAN s'y intéressèrent. Les Français d'abord, puis les Anglais, puis une dizaine de pays, dont le Japon. De tous les coins du monde, nous parvenaient des tonnes et des tonnes d'informations économiques...
En un mot, le système était devenu un ***monstre****. Totalement* ***ingérable****. Il nous fallait trouver autre chose...* ***Décentraliser*** *le système. L'idée était simple. Chacun des pays utilisateurs du système installait, chez lui, un ordinateur semblable au nôtre et entrait directement les informations sur ses terminaux, au lieu de nous les envoyer. Une vingtaine de pays adhérèrent au projet. L'astuce était de* ***relier*** *tous ces ordinateurs par* ***câbles sous-marins****.*
De cette façon, nous formions un immense système informatique réparti aux quatre coins de la planète. Un opérateur, installé n'importe où, pouvait interroger, sur son terminal, un ordinateur situé à Tokyo ou à Londres.

(suite page 1269)

L'utilisation de ces fonctions exige un minimum d'attention lorsque les programmes doivent être employés sur différents ordinateurs. Rappelons que le jeu des caractères disponibles n'est pas toujours le même. Ainsi la fonction :

SUCC ('Z')

pourrait donner des résultats différents.
Les opérateurs relationnels , < =, > =, < = et < > , sont utilisables sur le type caractère. Ces opérateurs comparent évidemment les chiffres correspondants à la codification interne des caractères ; il résultera :

'A' < 'B'

lorsque :

ORD ('A') < ORD ('B')

Type booléen. Il comporte seulement deux constantes « vrai » et « faux ». Une donnée de type booléen peut être le résultat d'une opération relationnelle appliquée à des données de type entier, réel ou caractère. Par exemple, la relation :

A > 5

sera vraie si la variable A contient un nombre supérieur à 5 et fausse si le chiffre est inférieur ou égal à 5.
Les données de type booléen peuvent être combinées entre elles par l'intermédiaire des opérateurs logiques ET, OU et NON. Par exemple, l'expression :

(A = 3) AND (B > 8)

sera vraie si (et seulement si) A est égal à 3 et si, en même temps, B est supérieur à 8. La syntaxe de la donnée de type booléen est indiquée dans le schéma ci-dessous.

Les déclarations de type

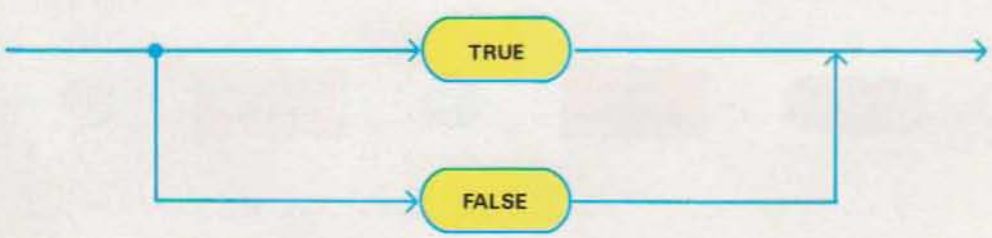
Jusqu'à présent nous n'avons vu que les types de données scolaires, appelées **standard,** et prédéfinies en Pascal.

Avant de passer aux autres types, analysons les données du Pascal définies comme déclarations, à savoir : les définitions des constantes

Attention !

La forme standard d'écriture des instructions en Pascal est fondée sur les caractères alphabétiques.
Nous avons toutefois choisi de présenter la syntaxe du langage en utilisant des caractères majuscules pour des raisons de clarté et d'homogénéité, mais également parce que de nombreuses machines utilisent exclusivement les caractères majuscules.
Dans la plupart des cas, le compilateur Pascal accepte indifféremment les instructions écrites en majuscules ou en minuscules.

SYNTAXE DE TYPE BOOLEEN



et des variables utilisées par le programme.
Le programme Pascal est subdivisé en deux sections, la première déclarative et la seconde, exécutive.
A travers la définition des types de données et des variables, la section déclarative informe aussi bien la seconde section que le programmeur, des structures de données utilisées. Les déclarations de la première section sont donc contraignantes.
L'utilisation, dans la seconde section, du programme d'une variable non préalablement déclarée est signalée comme une erreur.
En d'autres termes : le programmeur est obligé de prévoir, dès le début, toutes les données qu'il a l'intention d'utiliser, ce qui amène à construire les programmes de façon réfléchie.
On distingue la déclaration de constante et la déclaration de variable.

La déclaration de constante est utile lorsque l'on veut assigner un nom mnémonique à une constante fréquemment utilisée dans le programme. (Voir syntaxe ci-dessous). Elle établit simplement une équivalence entre un nom symbolique et une constante scalaire.
La valeur d'une telle constante ne peut être chargée à l'intérieur du programme ; une éventuelle instruction qui tenterait de changer cette valeur provoquerait une erreur signalée par le compilateur.

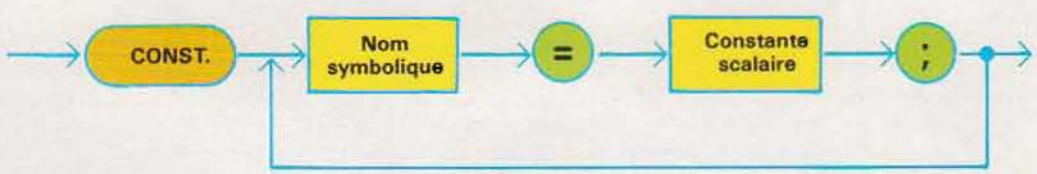
Les déclarations de constantes suivantes sont valables :

```
CONST
        PI = 3.1415927 ;
        INTERET = 0.11 ;
        ESPACE = ' ' ;
```

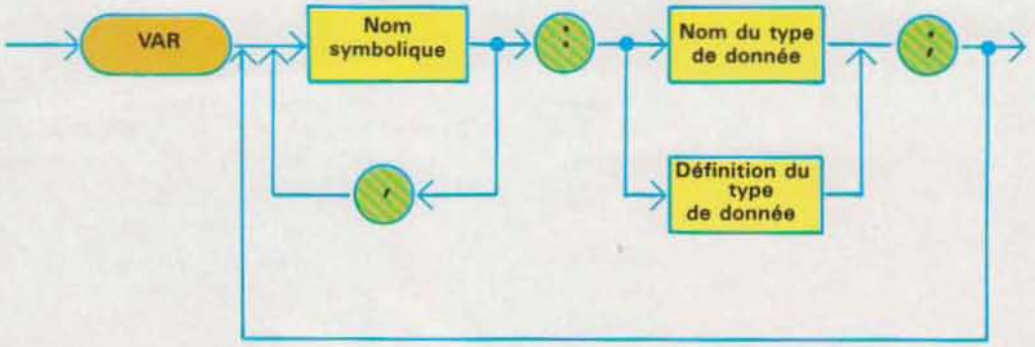
La déclaration de constante non seulement améliore la lisibilité du programme mais aussi facilite les modifications ou les corrections éventuelles.
Considérons, par exemple, un programme pour calculer les intérêts sur un compte-courant. Si, pour le taux d'intérêt, on utilise à l'intérieur du programme une valeur numérique au lieu d'un nom symbolique, une éventuelle variation du taux obligerait à rechercher tous les endroits où le taux apparaît dans le programme pour modifier celui-ci. En utilisant un nom symbolique pour la constante numérique, on obtient le même résultat en modifiant simplement la déclaration de constante.

La déclaration de variable. Une variable est une position de mémoire identifiée par un nom symbolique et contenant une valeur numérique. Dans certains langages autres que Pascal, lorsqu'une variable a été définie une première fois, il n'existe pas de restrictions sur le type de valeur qui peut être mémorisé dans la position correspondante.
En d'autres termes, des valeurs entières, réelles ou booléennes peuvent être indifféremment assignées à une telle variable. Une caractéristique du Pascal (qu'on retrouve dans l'Algol) est la formalisation du concept du type de données qui conduit à des règles restrictives dans la définition et dans l'assignation des variables.
Trois conséquences :
1 / Chaque variable scalaire peut mémoriser des valeurs appartenant à un seul type de donnée.

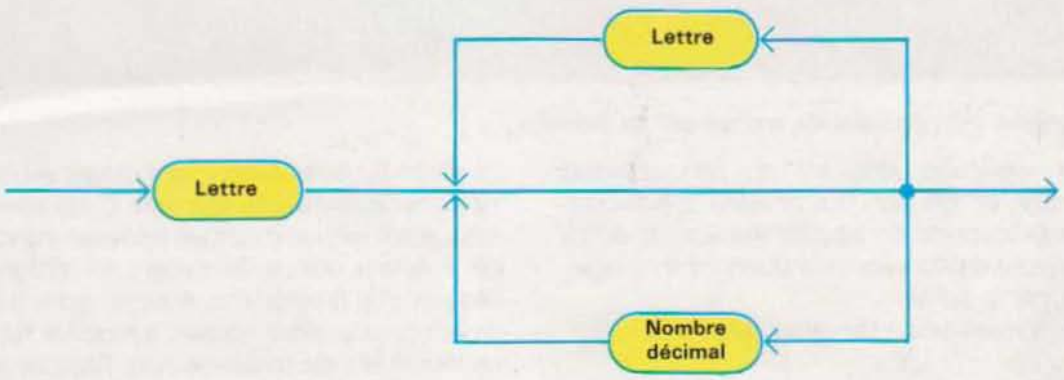
SYNTAXE DE LA DECLARATION DE CONSTANTE



SYNTAXE DE LA DECLARATION DE VARIABLE



CONSTRUCTION DES NOMS SYMBOLIQUES DES VARIABLES



2 / La majeure partie des fonctions prévues en Pascal opère sur des types de données déterminées ; une erreur est signalée à chaque fois qu'on tente d'appliquer ces fonctions à des types de données non prévus.

3 / Toute autre opération sur des données de types différents (sauf un seul cas particulier) est signalée comme une erreur.

Ces règles peuvent paraître comme de lourdes limitations lors de l'écriture d'un programme en Pascal mais elles offrent, en échange, certains avantages. Elles améliorent avant tout la lisibilité du programme car elles obligent le programmeur à déclarer, dès le début, toutes les variables qui seront utilisées et le type de données qu'elles contiendront. De plus, elles rendent plus faciles la recherche et la correction d'éventuelles erreurs. Il est nécessaire, pour créer une variable quelconque en Pascal, d'utiliser la déclaration VAR. La déclaration doit être précédée du mot-clé VAR suivi du nom ou des noms que l'on veut déclarer.

Les règles à suivre pour le choix du nom sont les mêmes que pour le Basic. Le nom ne doit pas coincider avec un mot réservé (mot-instruction ou commande de système) et il commence nécessairement par un caractère alphabétique ; les caractères suivants peuvent être alphabétiques ou numériques.

Il faut écrire, juste après, le ou les noms, le ou les types auxquels appartient la ou les variables. La variable est de type standard (entier,



B. Coleman/Marka

Système informatique de traitement de fichiers.

réel, caractère, booléen) ou non standard, et donc définie par l'usager dans une déclaration précédente de type (la manière de définir les types de données non standard sera exposée par la suite).
Exemples valables de déclaration :

```
VAR
  N, I, J     : INTEGER ;
  AREA        : REAL ;
  VEL, ACC    : REAL ;
  A, B, C     : CHAR ;
```

On constate que toutes les variables et les constantes utilisées dans un programme Pascal doivent obligatoirement être détachées au début, sinon une erreur est signalée lors de la compilation.
Certaines constantes prédéfinies et utilisées par le système font exception à cette règle ; c'est le cas par exemple de MAXINT, qui mémorise le plus grand entier.

Types de données scalaires non standard

En plus des types de données standard déjà décrits, le Pascal permet de recourir à d'autres types particuliers, définis par l'usager en fonction des nécessités du moment. Cette possibilité s'avère très utile lorsque les types standard ne couvrent pas suffisamment les différents besoins d'un programme. Avec un autre langage, on cherche normalement à modifier les caractéristiques du problème pour l'adapter aux types de données disponibles dans le langage utilisé. En Pascal, on agit en sens inverse en adaptant les types des données aux caractéristiques du problème.
Par exemple, pour écrire un programme qui effectue des actions déterminées pour chaque jour de la semaine, on pourrait associer, à chaque jour (du lundi au dimanche), un chiffre entier de 1 à 7. On écrirait, alors, des instructions conditionnelles du type :

```
IF JOUR = 6 THEN...
```

qui ne sont pas très lisibles à première vue puisqu'il faudrait avoir toujours, sous les yeux, un tableau d'équivalences. Les variables associées à ce nouveau type de données assumeront les valeurs LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE.
L'instruction précédente pourra alors être écri-

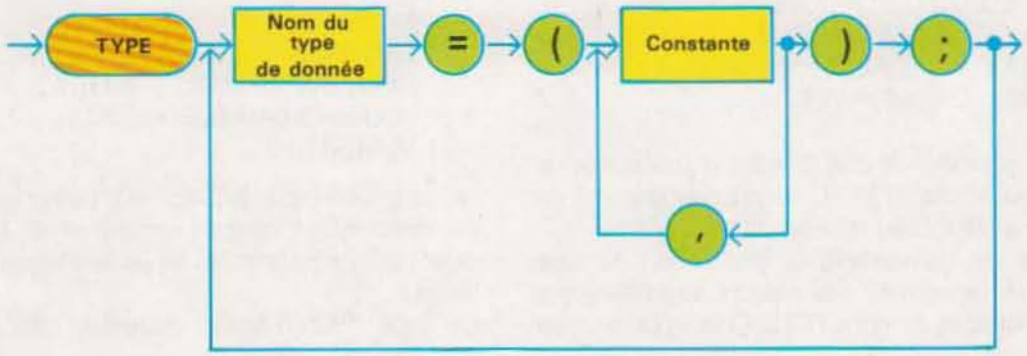
te d'une manière plus lisible :

IF JOUR=SAMEDI THEN...

L'introduction de nouveaux types de données définis par l'utilisateur améliorent la lisibilité du programme d'une façon substantielle et, de plus, comporte d'autres avantages qui seront exposés par la suite. Ces types de données sont définis par l'usager en énumérant individuellement les éléments (comme pour les jours de la semaine) ou comme sous-ensemble.
Dans ce cas, le nouveau type de données sera un sous-ensemble, plus restreint, d'un type standard, ou encore défini par le programmeur selon ses besoins.

Type de données définis par l'utilisateur.
Comme le nom l'indique, ces types sont créés par le programmeur, indépendamment des types standard. C'est une construction qui s'effectue avec la déclaration TYPE (voir ci-dessous) suivie de l'indicateur du type (c'est-à-dire le nom associé au nouveau type auquel on devra se référer pour déclarer une variable).
Cette identification développe les mêmes fonctions que REAL, INTEGER, CHAR, BOOLEAN, vus précédemment. Après le nom du type, on énumère entre parenthèses toutes les constantes qui appartiennent au nouveau type de donnée. L'ordre d'énumération est important car il définit une relation de classement. Celle-ci établit que chaque constante citée est « inférieure » par rapport à celles de droite et « supérieure » à celles de gauche (voir ci-dessous).

DECLARATION POUR DES TYPES DE DONNEES DEFINIS, NON STANDARD



EXEMPLES DE TYPES DE DONNEES NON STANDARD

```
TYPE
    JOUR-SEMAINE=(LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
                  SAMEDI, DIMANCHE);

    MOIS=(JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET, AOUT,
          SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE);

    COULEURS=(VIOLET, BLEU, VERT, JAUNE, ORANGE, ROUGE);
```

Le type de donnée COULEURS est composé des 6 constantes VIOLET, BLEU..., ROUGE classées de façon à ce que les relations :

VIOLET < BLEU
ORANGE > VERT
JAUNE > VERT

soient vraies.

Appliquer les relations comparatives à ce type de données pourrait sembler dénué de sens. Mais considérons l'exemple du type de donnée MOIS : l'ordre de classement établit de façon univoque l'organisation des constantes ; la circonstance dans laquelle :

JUILLET < AOUT

permet d'utiliser directement le nom de mois pour effectuer des contrôles de validité sur les dates émises.
De plus, ce classement établi, on peut appliquer les fonctions PRED et SUCC même à ce type de données.
Nous aurons alors :

```
PRED (JUILLET)=JUIN
SUCC (JUILLET)=AOUT
SUCC (VENDREDI)=SAMEDI
PRED (JAUNE)=VERT
```

Bien entendu, on doit prendre la précaution de ne pas appliquer PRED au premier élément du type et SUCC au dernier.
Outre ce classement, la déclaration de type définit l'ensemble des valeurs assumées par une variable de nom TEINTE déclarée de type COULEURS :

TEINTE :=TURQUOISE

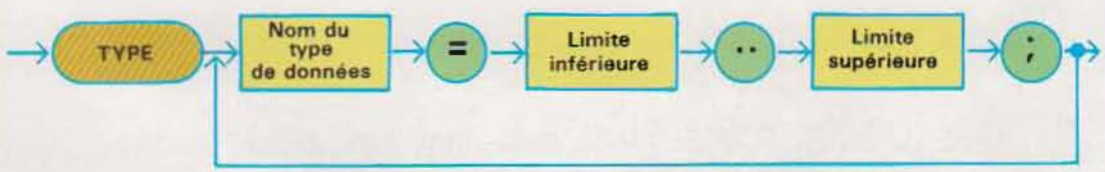
une erreur sera signalée. En effet, la teinte turquoise n'est pas prévue parmi les constantes formant le type COULEURS.
De petites erreurs d'assignation sont ainsi relevées sans passer par de lourdes procédures de contrôle pour accepter l'appartenance d'une donnée à un type. En définitive,l'utilisation des types définis par l'utilisateur diminuent les risques d'erreur.

Types de données sous-ensembles. Ce type est créé en utilisant une partie des données du type standard (à l'exception de REAL) ou un nouveau type déjà défini par l'utilisateur. Le sous-ensemble est constitué en décrivant, à l'intérieur du type de départ, des limites (inférieure et supérieure) et en déclarant que les données comprises entre elles (extrêmes inclus) appartiennent au sous-ensemble (voir ci-dessous).
Voici quelques exemples sortant sur le type sous-ensemble :

```
TYPE
    NOTES=0.. 10 ;
    CHIFFRES=0... 9 ;
    PRINTEMPS=MARS.. JUIN ;
    JOURS-OUVRABLES=LUNDI..
    VENDREDI ;
```

Le sous-ensemble NOTES est composé des nombres entiers compris entre 0 et 10. Les 3e et 4e cas concernent les types définis par l'utilisateur.
Le type PRINTEMPS, composé des mois

DECLARATION POUR LES TYPES DE DONNEES SOUS-ENSEMBLE



(mars à juin) est donc un sous-ensemble du type MOIS précédemment défini.
Comme pour les types définis par l'utilisateur, l'emploi de types sous-ensemble augmente la lisibilité du programme et signale automatiquement les erreurs d'assignation. Par exemple, lorsqu'on déclare une variable de nom MOYENNE de type entier, on admet implicitement qu'elle pourra prendre toutes les valeurs possibles des nombres entiers.
Mais si la même variable est déclarée de types NOTES, on comprend, à la simple lecture du programme, que MOYENNE prendra, à chaque point du programme, les seules valeurs entières comprises entre 0 et 10 (avec la signification de moyenne des notes). Chaque fois qu'on essaiera d'assigner à la variable MOYENNE un nombre en dehors de cet intervalle, une erreur sera signalée.

Les premières instructions du Pascal

La connaissance des concepts de type de donnée, constante, variable et déclaration est essentielle dans l'écriture de programmes Pascal corrects. Mais il faut, auparavant, entrer certaines informations concernant l'évaluation des expressions arithmétiques et booléennes, l'utilisation des fonctions et l'assignation des valeurs aux variables.
Avec les instructions fondamentales d'entrée et de sortie des données, on disposera de moyens très simples pour écrire les premiers programmes en Pascal.

Expressions arithmétiques et booléennes

En programmation Pascal, l'utilisation correcte des types des données et des expressions arithmétiques est essentielle. Ainsi, il n'est pas permis d'insérer différents types de données sauf dans les expressions arithmétiques qui demandent deux nombres réels pour fournir un résultat réel. Dans ce dernier cas, on peut substituer un entier à l'un des deux réels.
Exemple : étant donné deux variables réelles A et C et une variable entière B, l'instruction :

C :=A+B

fournit, en C, un résultat correct. Le compilateur convertit l'entier B en un nombre réel avant de calculer l'expression et d'assigner le résultat à C.
C'est l'unique exception admettant l'utilisation de types différents. Dans notre exemple, le symbole + pouvait être remplacé par tout autre opération admise sur nombres réels.
Voyons le cas des opérations non admises :

'1' - 1	La valeur '1' ne représente pas un chiffre mais un caractère
TRUE + 3	il n'est pas possible d'additionner un entier à une donnée booléenne
'A' + 1	le résultat de l'expression n'est pas le caractère B

Le recours à des types différents provoque une erreur dans les opérateurs DIV et MOD sur des réels. La division est effectuée entre valeurs entières et nécessite donc des opérandes entières. DIV fournit la partie entière de la division (11 DIV 4 est égal à 2) alors que MOD fournit le reste (11 MOD 4 est égal à 3). L'opération 11 DIV 4.0 est illégale car le diviseur est considéré par le compilateur Pascal comme un nombre réel.
N'oublions pas qu'un entier et un réel sont représentés de manière différente en mémoire, même s'ils ont la même valeur. Les nombres 4 et 4.0 sont égaux, mais ont une représentation binaire distincte.
En Pascal comme pour les autres langages, il existe des règles qui régissent le calcul des expressions. Le Pascal définit des hiérarchies de priorité pour les opérateurs ; les opérations de la hiérarchie supérieure se déroulent avant celles de la hiérarchie inférieure. Ceci est valable, en particulier, pour définir un autre ordre de priorité.
En ordre décroissant de priorité, la hiérarchie de priorité est la suivante :

1 / expression entre parenthèses
2 / opérateurs *, /, DIV, MOD
3 / opérateurs d'addition (+) et de soustraction (-)

Le calcul d'une expression en Pascal s'effectue donc dans cet ordre : d'abord les expressions entre parenthèses ; ensuite les multiplications et les divisions et, en dernier lieu, l'addition et la soustraction.

Par exemple, l'expression 7+A*4 est calculée ainsi :

7+(A*4)

Quand deux opérateurs ont le même ordre de priorité (qui n'est pas modifié par la présence de parenthèses), les expressions sont calculées comme elles sont écrites, dans le sens gauche-droite. Par exemple, l'expression :

A*12/B

est calculée comme :

(A*12)/B

Une formule du type :

A+B+(C - D)*E*F/G

est calculée :

(A+B)+((((C - D)*E) *F)/G)

Ces règles d'homogénéité des données et d'ordre de priorité valent également pour les expressions booléennes. Les opérateurs relationnels (< , < =, =, > = , >) sont employés pour comparer des données de type entier, réel, caractère, booléen ou tout autre type défini par l'usager.
Ces types ne peuvent cependant pas être mélangés car on ne peut, par exemple, comparer une valeur réelle à une valeur booléenne :

2 > = FALSE

est erronée car elle compare deux données de type différent.
Les opérateurs logiques ET OU NON (AND, OR, NOT) n'acceptent que le type booléen pour donner un résultat de type booléen.
Il existe également une hiérarchie de priorité (en ordre décroissant) pour l'évaluation des expressions booléennes :

1 / expression entre parenthèses
2 / opérateur logique NOT
3 / opérateurs /, DIV, MOD, AND
4 / opérateurs +, -, OR
5 / opérateurs relationnels (<,< =, =,> =,>)

Cette hiérarchie est plus compliquée que celle des expressions arithmétiques.
En effet, les expressions booléennes peuvent contenir un plus grand nombre de symboles et de types de données et également d'autres expressions de type arithmétique.
Par exemple :

A < B-1

est une expression booléenne valide puisque, selon les règles énoncées, elle est évaluée comme si elle avait été écrite de la façon suivante :

A < (B-1)

Observons que le résultat de cette expression aurait été calculée de manière non ambiguë même en l'absence de priorités.
En effet, elle ne peut, en aucune manière, être calculée de la façon suivante :

(A < B)-1

car il faudrait alors soustraire la valeur entière 1 à la valeur booléenne (vraie ou faux) résultat de la comparaison de A et B.
On peut donc dire que, dans les expressions booléennes, les opérateurs relationnels doivent toujours être évalués après le calcul des expressions arithmétiques.

Emploi des fonctions Pascal

L'ensemble des fonctions standard du Pascal a déjà été présenté lors de la description des types de données. Le recours à ces fonctions nécessite simplement l'écriture du nom de la fonction appelée, suivi du ou des arguments, (la valeur à laquelle on désire appliquer la fonction entre parenthèses).
Le résultat, c'est-à-dire la valeur de retour, peut être assigné à une variable ou réemployé dans une expression arithmétique.
Nous pouvons, par exemple, écrire :

(SQR(2)*9)+6.5

où la valeur 4 (résultat de l'élévation à la puissance de 2) est multipliée par 9 et le résultat additionné à 6.5.



G. Hall/Marka

Ecrans radar dans une salle de contrôle du trafic aérien.

Le résultat d'une fonction est donc utilisable comme celui d'une constante ou d'une variable. Son argument est lui-même une constante, une variable, une expression arithmétique ou même le résultat d'une autre fonction.

Nous pouvons alors écrire indifféremment :

```
SQR(2)
SQR(A)
SQR(7*A+12.5)
SQR(SIN(A))
```

Ces expressions sont toutes valables, à la seule condition que A ait été défini comme un type compatible avec les fonctions SQR et SIN.
Le calcul des expressions arithmétiques s'effectue de la même manière que précédemment.

Bien noter : dans l'utilisation des fonctions Pascal il est important de bien s'assurer de l'homogénéité des types de données impliquées dans un calcul. La fonction opère sur un argument d'un certain type tandis que le résultat peut appartenir à un autre type. Ainsi, la fonction TRUNC (qui élimine les chiffres après la virgule d'un réel) a pour argument une donnée de type réel et pour résultat un type entier.
De même, la fonction CHR traite un nombre entier (l'argument) et fournit un caractère (résultat).
Si certaines fonctions admettent des arguments de types différents, le résultat appartient toujours à un seul type.
Page suivante, on trouvera une liste et la description des fonctions standard disponibles en Pascal.
Cet ensemble est dit « standard » puisqu'il est commun à tous les compilateurs Pascal.

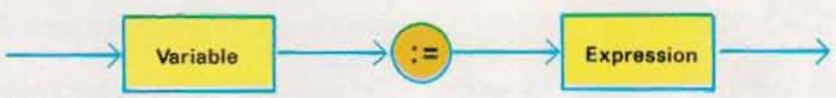
L'instruction d'assignation

Elle attribue une valeur à une variable (voir syntaxe page suivante).
L'opérateur d'assignation est représenté par : =, différent du symbole d'égalité des autres langages.
On a ainsi voulu éviter la confusion possible dans l'évaluation d'une instruction.

FORMATS STANDARD DU PASCAL

Nom de la fonction	Description	Type de donnée argument	Type de donnée résultat
ABS	Valeur absolue	Entier ou réel	Entier
ARCTAN	Arctangente	Réel ou entier	Réel
CHR	Caractère correspondant au nombre	Entier	Caractère
COS	Cosinus	Réel ou entier	Réel
EXP	Exponentiel $e^{argument}$	Réel ou entier	Réel
LN	Logarithme naturel (en base e)	Réel ou entier	Réel
ODD	Vérifie si argument impair	Entier	Booléen
ORD	Nombre correspondant au caractère	Caractère	Entier
PRED	Donnée précédant l'argument	Tout type sauf réel	Même type que l'argument
ROUND	Arrondi	Réel	Entier
SIN	Sinus	Réel ou entier	Réel
SQR	Carré de l'argument	Entier ou réel	Même type que l'argument
SQRT	Racine carrée de l'argument	Réel ou entier	Réel
SUCC	Donnée suivant l'argument	Tout type sauf réel	Même type que l'argument
TRUNC	Troncature	Réel	Entier

SYNTAXE DE L'INSTRUCTION D'ASSIGNATION



Par exemple :

A :=A+1.

permet d'ajouter 1 à la variable A, c'est-à-dire d'incrémenter A, alors que :

A=A+1.

doit être considéré comme une valeur booléenne, qui sera d'ailleurs toujours fausse.

L'instruction d'assignation est validée si toutes les variables ont été précédemment définies et quand celles situées à droite de l'opérateur ont déjà été assignées.

Test 21

1 / Parmi les déclaration des constantes suivantes, lesquelles sont correctes ?

Trouver les erreurs :

a)
```
CONST
    VALEURMAX = 1000;
```
b)
```
CONST
    PREMIER : 1 ;
```
c)
```
CONST
    MAXIMUM = 500 ;
    MINIMUM = - 50 ;
```
d)
```
CONST
    ALPHA = 0 OR 1 OR 2 ;
```
e)
```
CONST
    VALEURS = 0 .. 30 ;
```

---

2 / Ecrire une déclaration (unique) pour les constantes suivantes :

a) le symbole de l'addition (l'appeler PLUS) ;
b) la constante entière 24 qui indique le nombre de lignes lisibles sur un moniteur (l'appeler MAXLIGNES) ;
c et d) les nombres réels 0 et 100 qui représentent les températures de congélation et d'ébullition de l'eau (les appeler respectivement CONG et EBU).

---

3 / Quelles déclarations sont valables ? Trouver les erreurs :

a)
```
VAR
    1CAR, 2CAR, 3CAR : CHAR ;
```
b)
```
VAR
    CAR1, CAR2, CAR3 : INTEGER ;
    IN1              : REAL ;
    IN2              : INTEGER ;
```
c)
```
VAR
    X                : REAL ;
    Y                : REAL ;
    Z                : REAL ;
    XYZ              : CHAR ;
```

---

4 / Ecrire une déclaration de variable pour les données suivantes en choisissant les noms des variables adéquats :

a) les couleurs fondamentales rouge, jaune, bleu ;
b) une variable booléenne contenant la réponse positive ou négative à une question ;
c) un nombre entier qui tienne compte des lignes imprimées sur une page ;
d) une variable qui contient le numéro de la plaque d'immatriculation d'une voiture.

5 / Quelles sont les déclarations de type qui sont fausses ?
Trouver les erreurs :

a) TYPE INTEGER = - MAXINT.. + MAXINT ;
b) TYPE GRADES=0.0.. 100.0 ;
c) TYPE COULEURS=(JAUNE, ROUGE, VERT, ECARLATE, MARRON, VIOLET) ;
d) TYPE PRINTEMPS=MARS .. JUIN.

6 / Ecrire une déclaration (unique) pour définir les types de données suivants :

a) les noms des couleurs des cartes à jouer (COULEURS) ;
b) les lettres majuscules de l'alphabet (MAJUSCULES) ;
c) les sous-ensembles des nombres entiers de - 10 à + 50 (INTERVALLE).

7 / Quelles sont, parmi les valeurs suivantes, les données scalaires :

a) le premier élément d'un vecteur ;
b) le nom "JEAN ROSTAND" ;
c) le nombre de départements en France ;
d) un vecteur de 20 éléments.

8 / D'après le type suivant :

TYPE MOIS = (DECEMBRE, JANVIER, FEVRIER, MARS, JUIN, JUILLET, AOUT)

quels sont les résultats des fonctions suivantes :

a) PRED (JANVIER) ;
b) PRED (JUIN) ;
c) SUCC (MARS) ;
d) SUCC (AOUT) ;
e) PRED (DECEMBRE)

Solutions p. 1248.

Par exemple pour écrire :

```
A :=B+5.5
```

il faut s'assurer que la variable B contient une valeur déjà spécifiée, par exemple par une précédente instruction d'assignation. De plus, le type de donnée (résultat du calcul de l'expression) doit correspondre au type de donnée déclaré pour la variable A.
Exception à cette règle : l'assignation par une valeur entière d'une valeur réelle.
L'entier est alors transformé en réel avec une assignation équivalente. Dans tous les cas, les types doivent coïncider.
Examinons les déclarations de variable suivantes :

```
VAR
    INDICE      : INTEGER ;
    CONSONNE    : CHAR ;
    A,B,C       : REAL ;
    INDICATEUR  : BOOLEEN ;
```

les instructions d'assignation suivantes sont valables :

```
A :=88.5
B :=20.
CONSONNE :='K'
INDICE :=0
INDICATEUR :=A < 50
```

```
INDICE :=INDICE+1
C :=B*6.5
INDICATEUR :=(A < 50)OR(ODD(INDICE))
```

Les deux premières instructions assignent deux valeurs réelles aux variables A et B.
De même, pour les deux autres qui assignent des valeurs cohérentes au type de donnée déclaré pour les variables.
L'instruction :

```
INDICATEUR :=A<50
```

assigne la valeur FALSE (faux) à la variable booléenne FLAG puisque A est supérieur à 50.
Pour les deux autres, la variable INDICE vaut 1 et C est égal à 130.
Dans la dernière instruction, enfin, la valeur de la variable booléenne INDICATEUR est TRUE (vrai) puisque, même si A est supérieur à 50, la variable entière INDICE est égale à 1 (donc impaire) et les deux valeurs booléennes sont combinées au moyen de la fonction OR.

Les instructions d'entrée : READ et READLN

Les instructions qui activent la fonction d'entrée sont, en Pascal, READ et READLN (voir ci-dessous). L'instruction READ assigne aux variables (énumérées entre parenthèses) de la « read list » (« lire liste ») les valeurs lues par le dispositif d'entrée du clavier de saisie, du lecteur de cartes ou de la bande perforée.
Les variables de la « read list » peuvent être de type entier, réel, booléen, caractère et sous-ensemble. L'introduction d'une variable d'un autre type provoque une erreur. En nous référant à la déclaration précédente, voici quelques exemples d'instructions d'entrée :

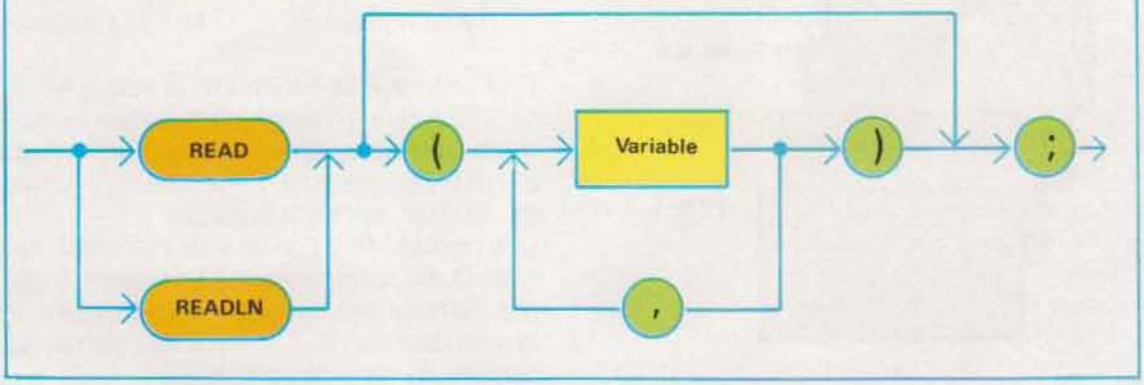
```
READ (A,B,C) ;
READ (INDICE, CONSONNE, INDICATEUR) ;
READ ;
```

Dans l'instruction READ, le nombre de données nécessaires à son exécution est égal au nombre de variables contenues dans la « read list » ; les données doivent, en outre, correspondre par type et par position, aux variables correspondantes. Mais on peut (seule exception admise) fournir une valeur entière que l'on assigne à une variable réelle.
Par exemple, si on a déclaré les variables A et B de type réel et la variable INDICE de type entier, on répondra enfin correctement à l'instruction :

```
READ (A, INDICE,B) ;
```

en entrant les 3 nombres 1.25, 10, 0.
Deux éventualités peuvent se présenter dans le cas où il n'y aurait pas une pleine correspondance entre les données et les variables.
Dans la première, le programme tente d'effectuer une assignation impossible, par exemple un caractère à une variable réelle, ce qui provoque une erreur et bloque son déroulement (fatal error = erreur fatale).
La seconde se produit quand le programme assigne une valeur erronée à la variable et poursuit l'exécution. Dans ce cas, on peut contourner la difficulté en insérant les instructions adéquates de validation de l'entrée de manière à assurer l'assignation souhaitée.

SYNTAXE DES INSTRUCTIONS D'ENTREE READ ET READLN



Pour compléter la description des instructions d'entrée, analysons l'instruction READLN. C'est un mot réservé qui peut remplacer READ en conservant la même syntaxe (voir page précédente). L'instruction READLN est utilisée pour effectuer, depuis le clavier ou tout autre dispositif, l'entrée sélective. Sur le lecteur de cartes ou sur l'unité bande, les données sont normalement organisées en blocs. READLN lit les valeurs des variables en paramètres, puis saute à la ligne en ignorant toutes les autres valeurs sur la liste d'entrée.
Supposons que les données soient regroupées en blocs de quatre, à raison d'un bloc par carte (voir ci-dessous).
L'exécution des deux instructions d'entrée :

```
READLN(A,B)
READLN (C,D)
```

fait en sorte que les valeurs :

```
A :=12
B :=20
```

c'est-à-dire les deux premières valeurs de la carte 1 soient assignées aux variables A et B. Pendant ce temps-là, les variables C et D prendront les valeurs :

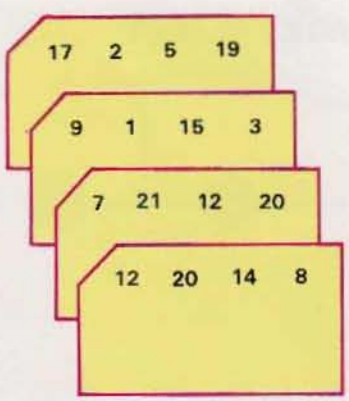
```
C :=7
D :=21
```

c'est-à-dire les deux premières valeurs de la seconde carte.
Si, au lieu de READLN, nous avions utilisé l'instruction READ, on aurait eu les assignations suivantes pour les quatre variables :

```
A :=12
B := 20
C :=14
D :=8
```

c'est-à-dire que toutes les valeurs de la première carte auraient été lues.

CARTES PERFOREES CONTENANT LES DONNEES POUR READLN



Les instructions de sortie : WRITE et WRITELN

Par analogie, les instructions de sortie admettent les deux formes WRITE et WRITELN.
Leur diagramme syntaxique (ci-contre) est complexe ; il reflète l'extrême flexibilité et la puissance des instructions de sortie propres au Pascal. Les éléments à imprimer sont énumérés entre parenthèses.
L'ensemble de ces éléments est défini « write list » (écrire liste). Chacun d'eux est, soit une expression dont le résultat est calculé au moment de l'impression, soit une variable simple, soit encore une chaîne de caractères.
Prenons l'hypothèse d'une valeur 4 assignée à la variable A déclarée de type entier.
L'instruction de sortie :

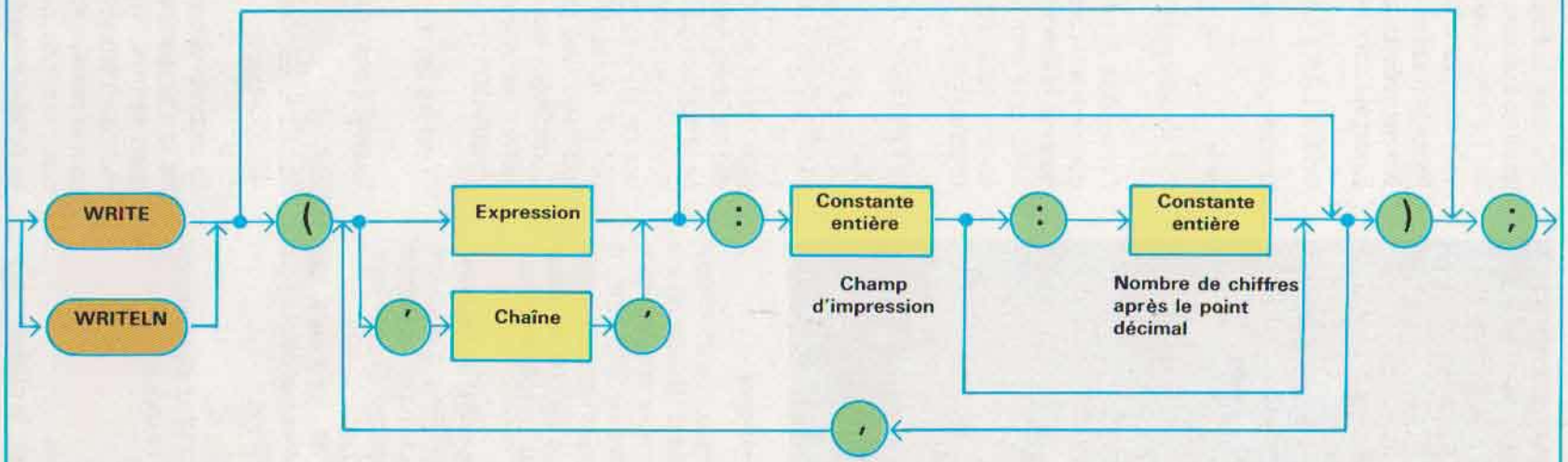
```
WRITE ('A=', A, ' Le carré de A est ', SQR (A))
```

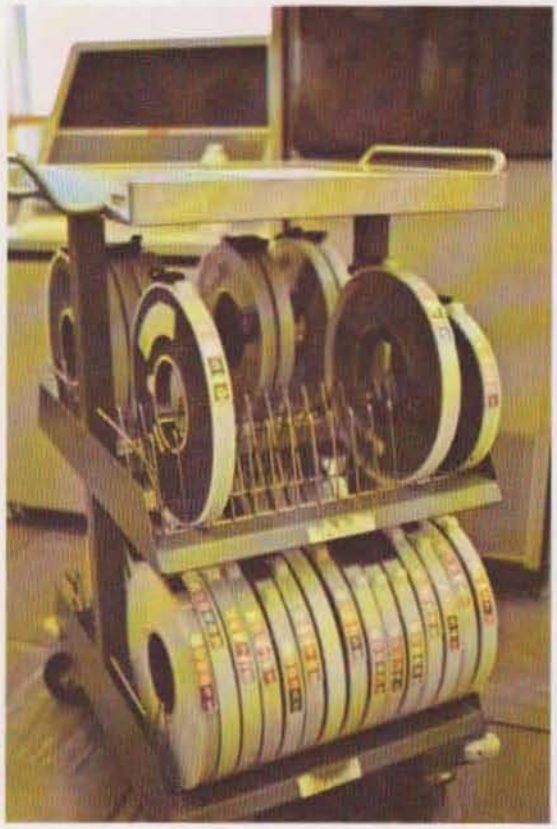
produira une ligne d'impression (sur écran ou imprimante) du type :

```
A =         4 Le carré de A est          16
  |10 colonnes|                  |10 colonnes|
```

Pour une variable booléenne, la valeur FALSE ou TRUE sera imprimée alors qu'une variable réelle sera toujours en notation scientifique (nombre décimal suivi d'une puissance entière de 10) sauf indication contraire.
L'instruction WRITE assure simplement le formatage des nombres et des caractères. A chaque élément est assigné, en impression, un champ composé d'un nombre de colonnes dépendant du type de variable.

SYNTAXE DES INSTRUCTIONS DE SORTIE WRITE ET WRITELN





Marka/Infophoto

Chariot porte-bandes magnétiques.

On suppose, dans l'exemple, qu'un champ d'impression de 10 colonnes a été assigné à la variable entière A. Si le nombre contenu dans la variable A comprend moins de 10 chiffres, le système effectue automatiquement une justification à droite en insérant, à gauche, les espaces vides nécessaires pour remplir le champ d'impression prévu.
Dans ce cas, A étant composé d'un seul chiffre, l'insertion de 9 espaces blancs à la gauche de la valeur pour couvrir le champ d'impression de 10 colonnes est automatique. D'où la justification à droite de la valeur.
Voici, à titre indicatif, les longueurs des champs d'impression adoptées par défaut pour les différents types de données :

Entiers = 12 colonnes
Réels = 16 colonnes (12 chiffres significatifs avec l'exposant dans la forme E±xx)

Caractère = 1 colonne
Booléen = 10 colonnes
Chaîne = Longueur de la chaîne

Toutefois, le Pascal permet de modifier facilement ces valeurs par défaut grâce à la « write list ». Celle-ci permet, pour la variable dont on veut modifier le champ d'impression, de déterminer le nombre de colonnes désiré. Ce nombre est précédé de deux points.
Exemple :

WRITE ('A=', 'Le carré de A est', SQR (A) : 5)

produit une ligne d'impression du type :

A = 4 Le carré de A est 16
|5 colonnes| |5 colonnes|

On recourt à cette facilité pour disposer les données en colonnes de la même manière qu'avec la fonction TAB(X) du Basic.
Ainsi, pour imprimer un nombre dont le dernier chiffre sera en colonne 50, on exécute l'instruction :

WRITE (A : 50)

La valeur contenue en A sera justifiée à droite et complétée à gauche par les espaces nécessaires.
Une option est disponible pour le traitement des variables de type réel. Elle permet de définir le nombre de chiffres décimaux à mettre en évidence au moment de l'impression. Ce nombre de chiffres décimaux est immédiatement déclaré après le champ d'impression et le symbole : (voir page précédente).
Considérons les variables B et C avec respectivement les valeurs 10.5728 et 152.278.
L'instruction :

WRITE (B:10:3,C:15)

fournira l'impression suivante :

10.572		1.522780000E+02
10 colonnes		15 colonnes

La seconde option nécessite explicitement, outre la définition du nombre de chiffres après le point décimal, l'impression de B en forme décimale avec une virgule fixe. Les variables réelles sont imprimées en notation scientifique si l'option n'est pas demandée. La variable C est automatiquement imprimée dans la forme

exponentielle (en virgule flottante). Notons enfin que le dernier chiffre décimal est tronqué et non arrondi.

Pour terminer la description des instructions de sortie, nous analyserons WRITELN dont la syntaxe (voir page 1241) rappelle WRITE.
Différence entre les deux instructions : à la fin d'une ligne d'impression WRITELN insère le retour chariot qui « initialise » l'opération de sortie suivante à la ligne suivante.
Ainsi, avec deux variables A et B de valeur respective 1 et 2, les deux instructions :

```
WRITE ('A =', A:4)
WRITE ('B =', B:4)
```

produiront la ligne d'impression suivante :

```
A =  1 B =  2
```

En revanche, les instructions :

```
WRITELN ('A =', A:4)
WRITELN ('B =', B:4)
```

produiront alors ces deux lignes d'impression :

```
A =  1
B =  2
```

WRITELN, qui provoque l'avancement d'une ligne d'impression, est aussi utilisable sans paramètres pour obtenir des lignes d'espacement.

Structure d'un programme Pascal

Tout listing en Pascal présente la structure et la syntaxe décrites ci-dessous.
La structure d'un programme se compose de deux sections : l'intitulé (titre) et le bloc. Le titre, consistant en une seule ligne, contient PROGRAM (mot réservé) suivi du titre

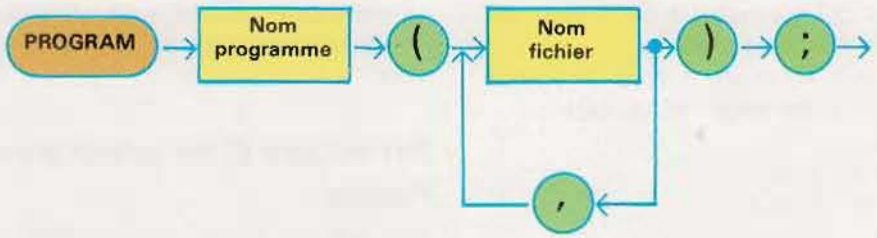
FORMAT GENERAL D'UN PROGRAMME PASCAL

```
PROGRAM nom (nom de fichier 1,... 2,....)
LABEL       déclaration... (commentaire)
CONST       déclaration... (commentaire)
TYPE        déclaration... (commentaire)
VAR         déclaration... (commentaire)
PROCEDURE déclaration... (commentaire)
FUNCTION    déclaration... (commentaire)
BEGIN
   instruction (commentaire)
   .....
   .....
   instruction ;
END.
```

DIAGRAMME SYNTAXIQUE REPRESENTANT LA STRUCTURE D'UN PROGRAMME PASCAL



DIAGRAMME SYNTAXIQUE DU TITRE D'UN PROGRAMME



proprement dit et, le cas échéant, de la liste des noms de fichiers externes (voir diagramme ci-dessus). Le nom du programme n'a aucune signification particulière pour le système d'exploitation. En revanche, les fichiers listés sont utilisés par le programme pour échanger des informations avec le monde extérieur.
Après le titre suit le corps **("bloc")** du programme, lui-même composé des deux sections contenant les déclarations et les instructions exécutables (voir diagramme syntaxique ci-contre).
Les déclarations — au nombre de 6 — décrivent toutes les données à utiliser par le programme. Trois d'entre elles — constante (CONST), type de donnée (TYPE) et variable (VAR) — ont déjà été analysées. Restent :

LABEL
PROCEDURE
FONCTION

L'ordre d'écriture des déclarations est celui du schéma. Ecrire TYPE avant CONST, par exemple, provoque normalement un message d'erreur.
La seconde section du bloc constitue le corps exécutable du programme et s'appelle, en anglais, **compound statement** ("instruction composée") ; elle contient en effet une série d'instructions Pascal (diagramme syntaxique p. 1246). Comme on peut le voir, toutes les instructions sont enfermées entre les mots réservés BEGIN (début) et END (fin).
Cette délimitation a pour effet, en isolant le code, d'augmenter sa lisibilité tout en donnant l'ordre séquentiel d'exécution des instructions, séparées par le symbole (;) ou par un mot réservé.
Il ne s'agit donc pas d'instructions proprement dites : aucune action spécifique du système ne leur correspond. Elles servent uniquement à délimiter l'ensemble des instructions à effectuer séquentiellement dans l'ordre dans lequel elles sont écrites, chaque instruction est séparée de la suivante par le point virgule (;). Le compilateur reconnaît donc la fin d'une instruction lorsqu'il rencontre le symbole ; ou un mot réservé.
Pour éviter des erreurs de compilation, il est conseillé de toujours insérer le caractère ; à la fin de l'instruction, même quand il est superflu : dans ce cas, le compilateur interprète la double borne comme une « instruction vide » (en anglais : empty statement) et il en restera là.

DIAGRAMME SYNTAXIQUE DU BLOC

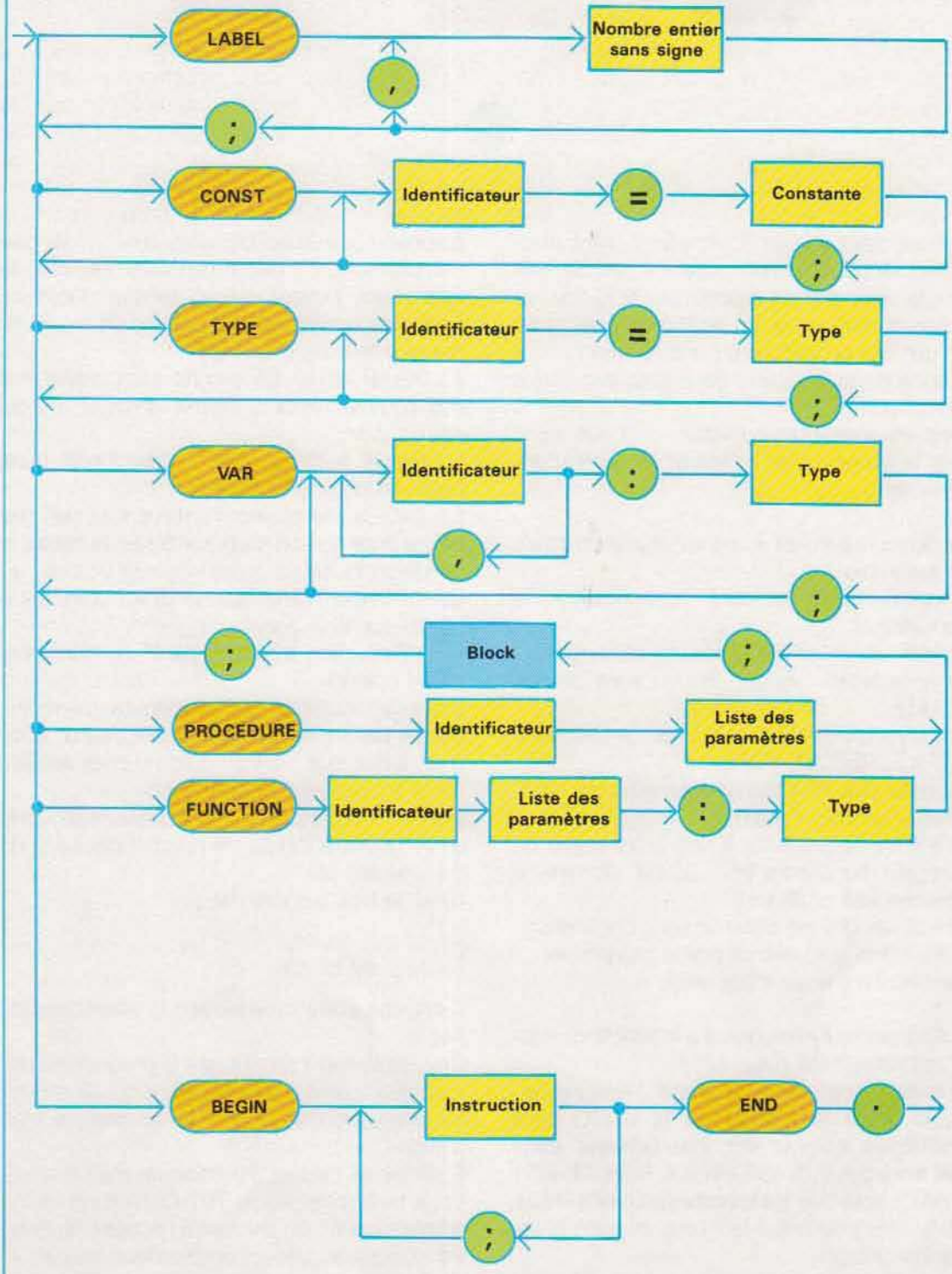
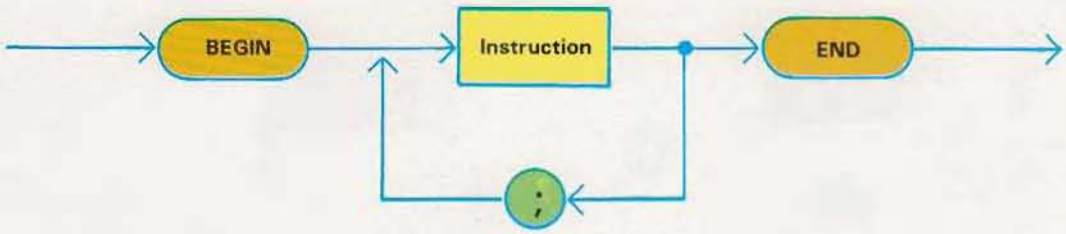


DIAGRAMME SYNTAXIQUE D'UNE INSTRUCTION COMPOSEE (COMPOUND STATEMENT)



Comme partout, les instructions sont exécutées séquentiellement, sauf en cas de saut conditionnel, afin de répéter plusieurs fois un groupe d'instructions ou pour mettre en mouvement une procédure ou une fonction.
Le programme s'achève donc avec la dernière instruction.
Avec les instructions Pascal, on peut alors, dans le programme, effectuer les opérations suivantes :

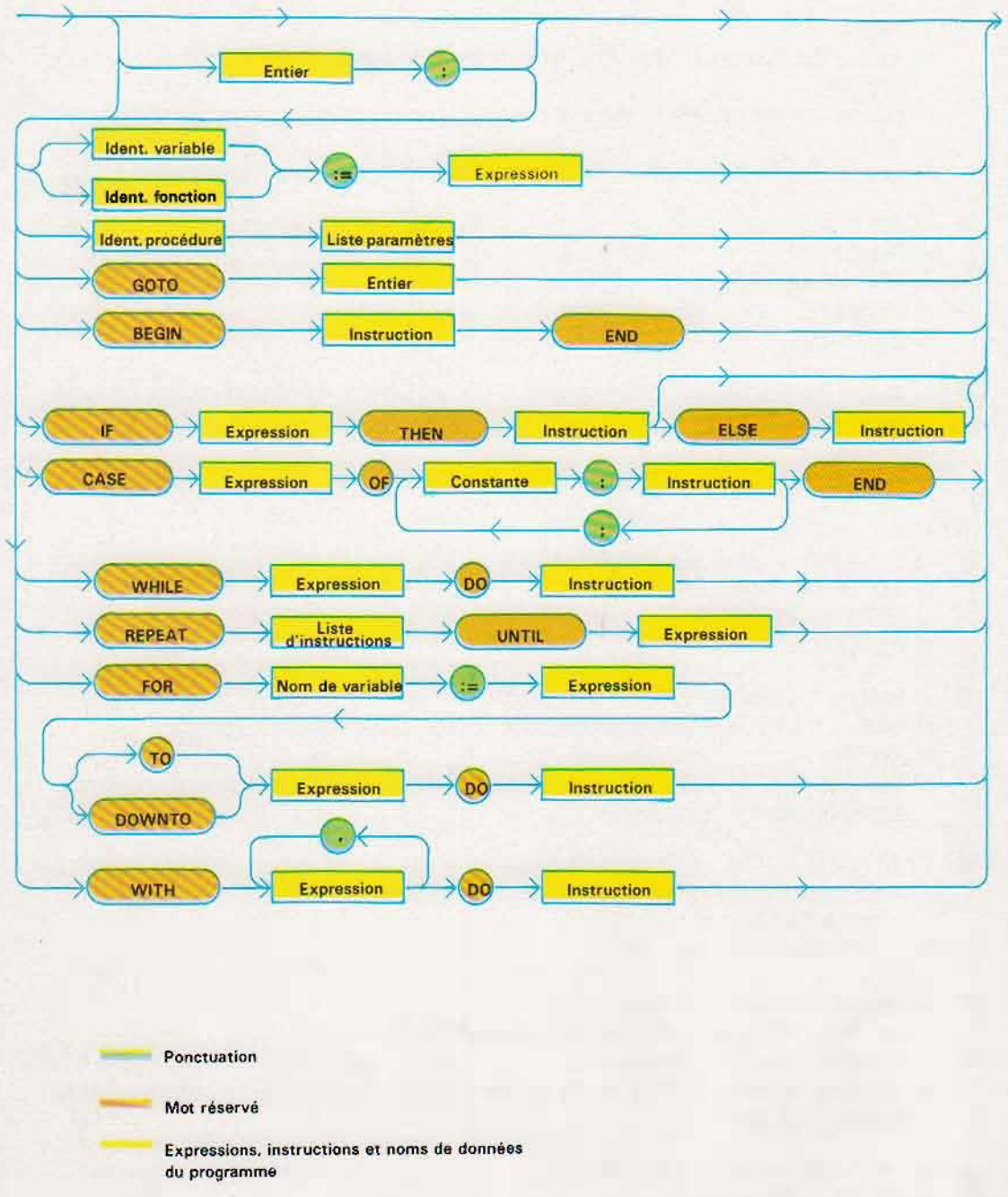
- assigner une valeur à une variable (instruction d'assignation),
- rappeler une procédure (identificateur de procédure),
- choisir un ensemble d'actions en se basant sur certaines valeurs (instructions IF et CASE),
- répéter un groupe d'actions (instructions WHILE, REPEAT, FOR),
- appeler des enregistrements sans les nommer (instructions WITH),
- transférer le contrôle à une autre partie du programme (instruction GOTO, fortement déconseillée en Pascal),
- traiter un groupe d'instructions comme une instruction unique (instruction composée),
- ne rien faire (instruction vide).

Le diagramme syntaxique des instructions Pascal est représenté page 1247.
Certaines instructions comme l'assignation, l'appel d'une procédure ou le GOTO sont considérées comme des **instructions simples** alors que IF, CASE, WHILE, REPEAT, FOR et WITH sont des **instructions structurées** puisqu'elles peuvent, à leur tour, contenir d'autres instructions.
Chaque instruction structurée est donc perçue comme sous-ensemble d'un bloc. Dans les compilateurs, il n'existe généralement pas de restrictions portant sur le nombre d'instructions structurées imbriquées dans un programme ni même dans un bloc.
En Pascal aussi, les programmes admettent des commentaires à insérer entre (*...*) ou entre (.....).
Le Pascal autorise une grande liberté dans l'écriture des instructions.
On peut écrire plusieurs instructions sur une même ligne ou une seule sur plusieurs lignes, à condition toutefois qu'un nom de variable ou qu'une valeur numérique ne se trouvent pas à cheval sur deux lignes.
Une instruction s'écrit à partir de n'importe quelle colonne.
Cette particularité est avantageusement exploitée dans « l'indentation » des instructions, c'est-à-dire leur mise en colonne pour améliorer la lisibilité des programmes.
Elément très important et indispensable dans un programme Pascal : le point. Il indique la fin du programme.
Ainsi, le bloc est délimité par :

— le titre en haut
— le point en bas.

C'est une autre caractéristique essentielle du Pascal.
Chaque élément constituant le programme est considéré comme une structure qui en renferme une autre qui, à son tour, en englobe bien d'autres...
C'est ce qui permet de définir un problème selon la technique dite de TOP-DOWN (structure arborescente) : on parvient à l'écriture du code par niveaux successifs en étendant, de part et d'autre, les structures initialement définies.

DIAGRAMME SYNTAXIQUE DES INSTRUCTIONS PASCAL



Solutions du test 21

1 / a) valable
b) fausse. Le symbole : doit être remplacé par le symbole =
c) valable
d) fausse. Il n'est pas admis d'écrire des fonctions booléennes dans une déclaration de type
e) fausse. On ne peut définir une constante avec le type sous-ensemble

---

2 / CONST
a) PLUS='+'
b) MAXLIGNES=24
c) CONG=0.
d) EBU=100.

---

3 / a) fausse. Le premier caractère du nom d'une variable doit obligatoirement être une lettre
b) valable
c) valable

---

4 / VAR
a) COULEURS : (ROUGE, JAUNE, BLEU)
b) REPONSE : BOOLEAN
c) LIGNES : INTEGER
d) PLAQUE : INTEGER

---

5 / a) valable. Il s'agit toutefois d'une déclaration de type prédéfini dans le Pascal
b) fausse. Il n'est pas admis de déclarer un sous-ensemble du type réel
c) valable
d) fausse si on n'a pas déclaré un type (par exemple MOIS) contenant les extrêmes indiqués dans la déclaration

---

6 / TYPE
COULEURS=(CŒUR, CARREAU, TREFLE, PIQUE)
MAJUSCULES='A' .. 'Z'
INTERVALLE= -10 .. 50

---

7 / a) scalaire. Il s'agit d'un seul chiffre
b) n'est pas scalaire. Une ligne est une donnée structurée
c) scalaire. C'est un nombre entier
d) n'est pas scalaire. Il s'agit d'une des deux valeurs admises pour le type scalaire standard booléen

---

8 / a) PRED (JANVIER) = DECEMBRE
b) PRED (JUIN) = MARS
c) SUCC (MARS) = JUIN
d) SUCC (AOUT) = n'est pas défini
e) PRED (DECEMBRE) = n'est pas défini

Les instructions de contrôle

Avec les instructions étudiées précédemment, on peut écrire des programmes simples en Pascal, de structure « séquentielle ».
Notons bien qu'il n'existe alors aucune possibilité de modifier cette séquence dans l'exécution des instructions, selon que certaines conditions sont ou non vérifiées.
Evidemment le Pascal offre aussi cette possibilité grâce aux instructions de contrôle.
Il prévoit notamment un ensemble d'instructions d'itération et de saut (conditionnelles ou non) sans doute plus étendu et plus flexible que dans d'autres langages, comme par exemple le Basic et le Fortran.

Les instructions d'itération

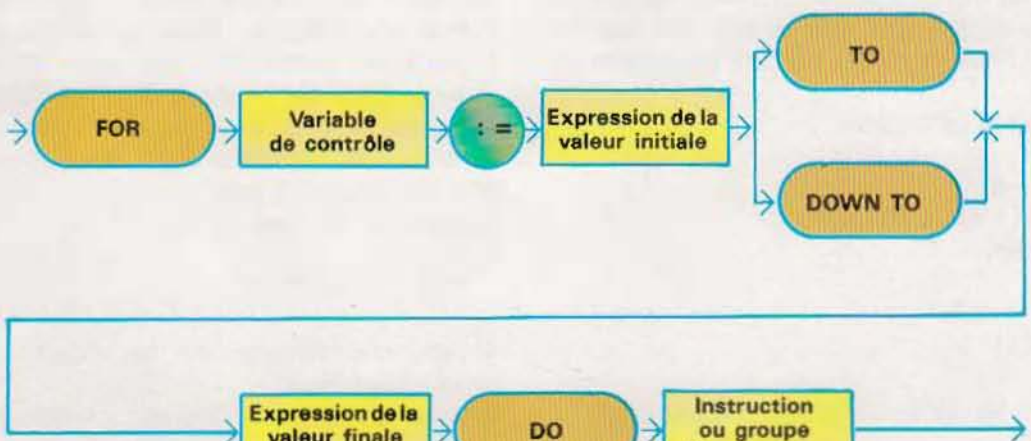
Elles exécutent un même groupe d'instructions un nombre de fois fixé à l'avance ou lié à la vérification d'une condition. Outre l'instruction FOR-TO-DO, existant également en Basic avec quelques différences, le Pascal admet deux autres instructions :

WHILE-DO
et
REPEAT-UNTIL

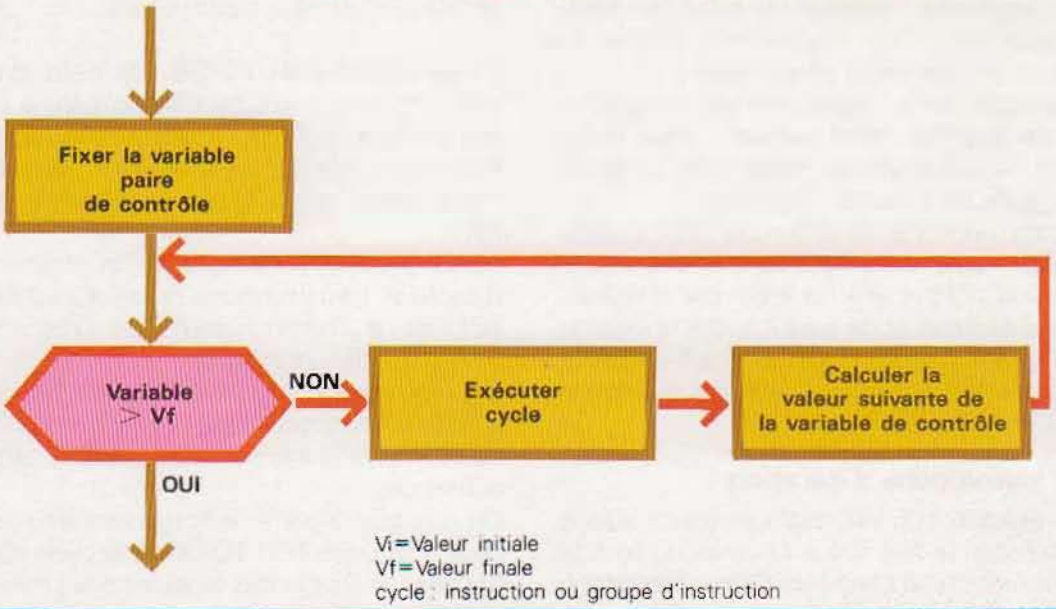
Ce sont les structures puissantes pour tous les problèmes où interviennent des exécutions répétées d'un groupe d'instructions.

L'instruction FOR-TO-DO. Elle exécute une instruction ou un groupe d'instructions à chaque occurrence d'une variable de contrôle, qui est incrémentée à chaque itération ; elle passe d'une valeur initiale à une valeur finale données.
Le diagramme syntaxique qu'on trouve ci-dessous et l'organigramme de la page suivante illustrent le fonctionnement de l'instruction d'un point de vue logique.
Cette instruction Pascal est très proche de la structure correspondante en Basic FOR-TO-NEXT, même si elle présente quelques petites différences.
On constate, à partir de l'organigramme de la page 1250, que FOR-TO-DO et le cycle d'instructions ne sont jamais exécutés si la première comparaison fait apparaître que la variable de contrôle est supérieure à la valeur finale.
En revanche, en Basic, le cycle est déroulé au moins une fois quelle que soit cette valeur. De plus, le Pascal n'admet pas, comme le Basic de spécifier, avec le mot réservé STEP, le pas d'incrémentation de la variable de contrôle.

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION FOR-TO-DO



LOGIQUE D'EXECUTION DE L'INSTRUCTION FOR-TO-DO



Revenons au diagramme ci-dessous et observons que les valeurs initiales et finales de la variable de contrôle ne sont pas nécessairement des constantes numériques : elles peuvent aussi être des valeurs fournies par des expressions évidemment de même type que la donnée.
Le calcul des expressions définissant les valeurs initiale et finale est effectué une seule fois, quand apparaît l'instruction FOR.
Cela signifie qu'une fois les valeurs initiales et finales déterminées, elles ne sont plus susceptibles d'être modifiées pendant l'exécution du cycle.
La série d'instructions :

```
K :=5 ;
FOR I:=K-1 TO K+1 DO
BEGIN
      K:=K-1 ;
      WRITELN ('I =', 1:2, 'K=', K:2) ;
END ;
```

produit les lignes d'impression suivantes :

```
I= 4  K= 4
I= 5  K= 3
I= 6  K= 2
```

Le cycle se déroule pour I entre 4 et 6, c'est-à-dire de la valeur initiale à la valeur finale calculées en fonction de la variable K = 5. Aucun changement de K à l'intérieur du cycle ne peut affecter ces valeurs.
La variable de contrôle est élevée à la valeur suivante quand on rencontre, dans l'instruction, le mot réservé TO ; elle est décrémentée avec DOWNTO. Dans ce dernier cas, la valeur finale doit être inférieure à la valeur initiale.
La valeur suivante de la variable est calculée en appliquant les fonctions SUCC en cas d'incrémentation et PRED en cas de décrémentation. Cela signifie que, n'étant pas admis comme argument pour les fonctions SUCC et PRED, le type réel ne peut servir comme variable de contrôle ; en revanche, les autres types de données simples le sont.
L'instruction du type :

```
FOR : = 'A' TO 'Z' DO
```

est correcte, mais ne sera pas acceptée par l'interpréteur Basic.
Le corps du cycle lui-même est une instruction simple ou composée (ensemble d'instructions délimitées par BEGIN et END). Les instructions d'assignation de la variable de contrôle ne sont pas reconnues à l'intérieur du corps du cycle : par conséquent, toute modification de valeurs de part et d'autre de la variable est refusée.

Toutefois il reste possible d'assigner sa valeur à une autre variable qui peut être ensuite librement utilisée.

Soit la série d'instructions :

```
SOMME:=0;
FOR K:=1 TO 5 DO
BEGIN
      SOMME:=SOMME+K;
      K:=K+2;
END;
```

l'instruction

```
K:=K+2
```

n'a pas d'existence dans le cycle et sa présence est signalée comme une erreur au moment de la compilation.
En fin d'exécution de l'instruction FOR-TO-DO, c'est-à-dire une fois le corps du cycle exécuté un nombre donné de fois, la valeur de la variable redevient « libre » (indéfinie) : elle ne peut être réutilisée par les instructions suivantes qu'à condition d'avoir été à nouveau assignée.
L'emploi de l'instruction FOR-TO-DO se justifie parfaitement lorsque des groupes d'instructions doivent être exécutés sous le contrôle d'un compteur pour un nombre de fois connu à priori. Toutefois, le nombre dépend parfois de la vérification de certaines conditions. Dans ce cas, le Pascal dispose de deux autres instructions, WHILE-DO et REPEAT-UNTIL, plus flexibles et mieux adaptées aux situations décrites.

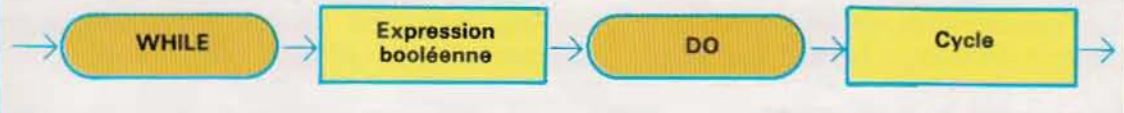
L'instruction WHILE-DO. L'exécution d'un cycle est répétée jusqu'à ce qu'une condition booléenne soit vérifiée (voir syntaxe ci-dessous). L'expression de contrôle est évaluée au début et, si elle s'avère vraie, tout ce qui suit le mot réservé DO (le corps du cycle) est exécuté. Puis, l'expression booléenne est à nouveau calculée et, si elle s'avère encore vraie, le corps du cycle est à nouveau exécuté et ainsi de suite...
Le cycle n'est plus activé quand l'expression devient fausse, comme le montre l'organigramme de la page 1252, ainsi que l'exemple donné par l'édition des 100 premiers entiers.
A l'opposé de ce qui se passe pour l'instruction FOR-TO-DO, non seulement il est admis de modifier la variable de contrôle mais encore cette opération est absolument nécessaire pour éviter les erreurs. En effet, sans instruction de modification de la variable, le cycle serait indéfiniment répété.
Si l'expression booléenne se révèle vraie au moment de l'exécution de l'instruction WHILE-DO, on dit que le cycle « boucle » (boucle fermée). Il faut donc faire très attention dans l'écriture de l'expression booléenne et vérifier quelle variable assume la valeur FALSE en fin de cycle.

Considérons donc, à titre d'exemple, la série suivante :

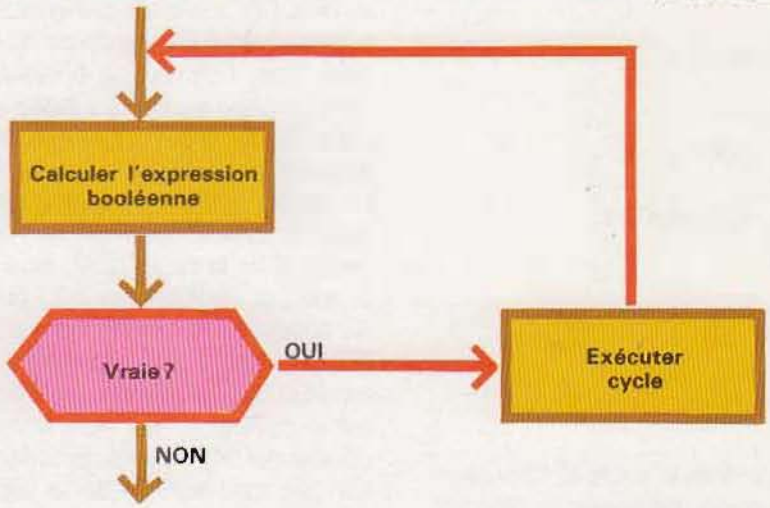
```
READ (INDICE) ;
WHILE INDICE      0 DO
  BEGIN
  .....
  INDICE:=INDICE-1;
  .....
  END;
```

Si, lors de la demande d'entrée de la variable

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION WHILE-DO



LOGIQUE D'EXECUTION DE L'INSTRUCTION WHILE-DO



IMPRESSION DES 100 PREMIERS NOMBRES ENTIERS

```
PROGRAM IMPRESSION (INPUT, OUTPUT);
INDICE : INTEGER;  (* EST LA VARIABLE QUI SERA IMPRIMEE ET UTILISEE
  VAR                 POUR LE CONTROLE DE FIN CYCLE*)
BEGIN
      WHILE INDICE <= 100 DO
      BEGIN
            WRITELN (INDICE:5);
            INDICE:=INDICE+1;
      END; (*FIN DU CORPS DE CYCLE*)
END.       (*FIN DU PROGRAMME*)
```

INDICE, on répond par un nombre négatif, on aboutira à une boucle fermée.
En effet, INDICE étant décrémenté à chaque passage, sa valeur n'équivaudra jamais à zéro et l'expression booléenne INDICE <>0 sera toujours vérifiée.
Si, dans un WHILE-DO, l'expression booléenne était fausse à la première exécution de l'instruction, le corps du cycle ne serait pas activé.
Pour l'exécuter au moins une fois, on doit donc recourir à l'instruction REPEAT-UNTIL.

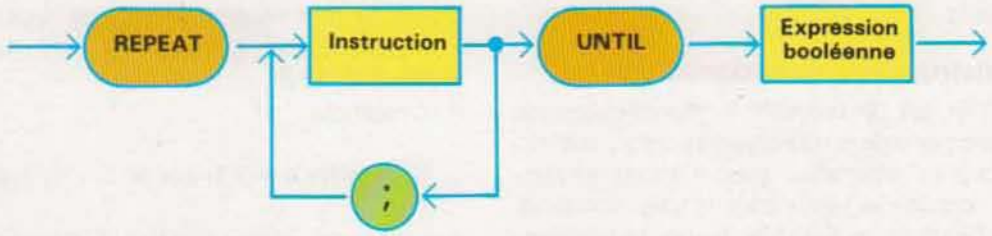
L'instruction REPEAT-UNTIL. Son action (voir diagramme syntaxique page ci-contre) est de répéter un cycle jusqu'à vérifier la condition posée. Elle se rapproche donc du WHILE-DO avec, toutefois, deux différences importantes :

— REPEAT-UNTIL exécute le test sur la première condition **après** et non **avant** de dérouler le corps du cycle ;

— cette activation se répète jusqu'à ce que la condition de contrôle soit fausse, alors que le corps du cycle est effectué (dans le WHILE-DO, l'exécution s'arrête quand l'expression booléenne se vérifie).

Ces différences sont mises en évidence par les logiques d'exécution de REPEAT-UNTIL et de WHILE-DO (pages 1252 et 1253).

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION REPEAT-UNTIL



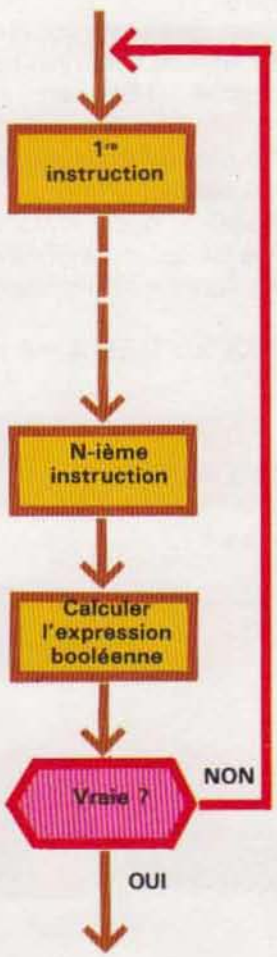
On remarque que, contrairement à WHILE-DO, les instructions constituant le corps du cycle de REPEAT-UNTIL sont exécutées au moins une fois, indépendamment de la valeur initiale de l'expression booléenne. Si celle-ci s'avère vraie, l'exécution s'achève après le premier passage.
A titre d'exemple, dans la portion de programme :

```
READLN (CONTROL) ;
REPEAT
        INSTRUCTION 1 ;
        INSTRUCTION 2 ;
UNTIL (CONTROL = 0) ;
```

en répondant à la commande d'entrée du chiffre 0, les instructions 1 et 2 sont exécutées une seule fois car la condition de contrôle est vraie.
Dans l'instruction REPEAT-UNTIL, même si le cycle comporte plusieurs instructions, il n'est nullement besoin, comme dans le WHILE-DO, de le fermer avec BEGIN et END puisque les mots réservés REPEAT et UNTIL intègrent cette fonction de délimitateurs.
Naturellement, il est toujours possible de fermer le corps du cycle entre les mots BEGIN et END, comme dans l'exemple suivant :

```
REPEAT
  BEGIN
    INSTRUCTION 1 ;
    INSTRUCTION 2 ;
      .
      .
      .
      .
      .
```

LOGIQUE D'EXECUTION DE L'INSTRUCTION REPEAT-UNTIL



```
    END ;
UNTIL (CONTROL = 0) ;
```

Sans constituer une erreur, cette manière d'écrire est, toutefois, considérée comme redondante.

Les instructions conditionnelles

Leur rôle est de modifier le flux d'exécution d'un programme en déroulant un groupe d'instructions en alternative avec d'autres instructions, lors de la vérification d'une condition. Outre l'instruction IF-THEN, également présente en Basic, le Pascal admet l'instruction CASE qui permet de choisir entre plus de deux alternatives.

L'instruction IF-THEN-ELSE. Elle est développée dans le diagramme syntaxique qui se trouve en bas de page.
Il signifie que :
— l'expression booléenne est évaluée ;
— si elle est vraie, ALORS l'instruction suivant le mot réservé THEN est envoyée en exécution ;
— SINON le contrôle du programme passe à l'instruction suivante.
Il s'agit, en général, d'une instruction composée, délimitée par les mots BEGIN et END.
On peut ainsi écrire indifféremment :

```
IF CONTROL>0 THEN A:=A+1 ;
```

soit :

```
IF CONTROL=0 THEN
   BEGIN
      A:=A+1 ;
      WRITELN ('A=', A:5) ;
   END ;
```

Dans le second cas, si la condition n'est pas vérifiée, l'instruction suivante est exécutée.
Après le mot réservé THEN, on peut insérer n'importe quelle instruction Pascal (même une autre IF-THEN).
Par exemple :

```
IF CONTROL=0 THEN IF A=10 THEN...
```

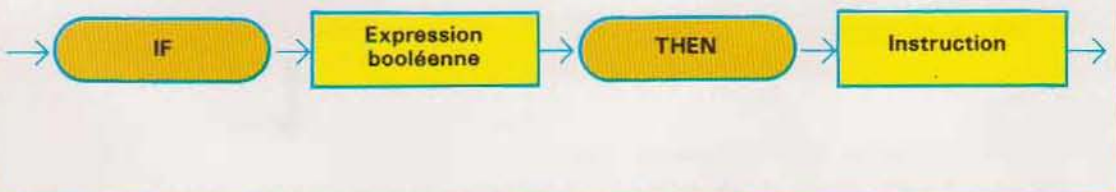
Evidemment, la seconde IF-THEN ne sera exécutée que si la condition sur la variable CONTROL est vérifiée.
L'expression booléenne à vérifier peut également être construite avec des opérateurs logiques (AND, NOT, OR) pour, par exemple, établir des conditions complexes répondant à des situations particulières : par exemple variable de contrôle comprise dans un intervalle, supérieure ou inférieure à une certaine valeur...

La ligne de programme :

```
IF (A>=0) AND (A<=100) THEN A :=1
```

place à 1 la variable A si la valeur est située entre 0 et 100.
Le même résultat s'obtient aussi en enchaînant plusieurs instructions IF-THEN. Les conditions complexes sont très utiles pour valider des données d'entrée et vérifier qu'elles ont un sens pour le système.
Ce contrôle est obtenu par une combinaison de conditions. Si la donnée fournie par l'opérateur n'est pas licite, on pourra faire imprimer un message d'erreur.

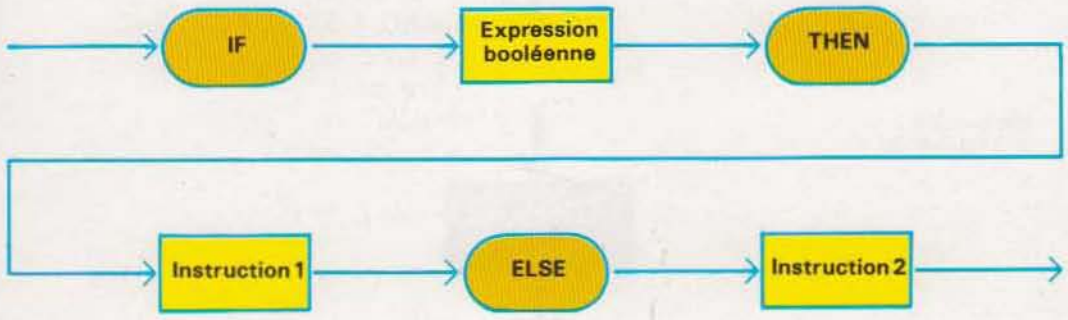
SYNTAXE DE L'INSTRUCTION IF-THEN



INTRODUCTION ET CONTROLE D'UNE DONNEE

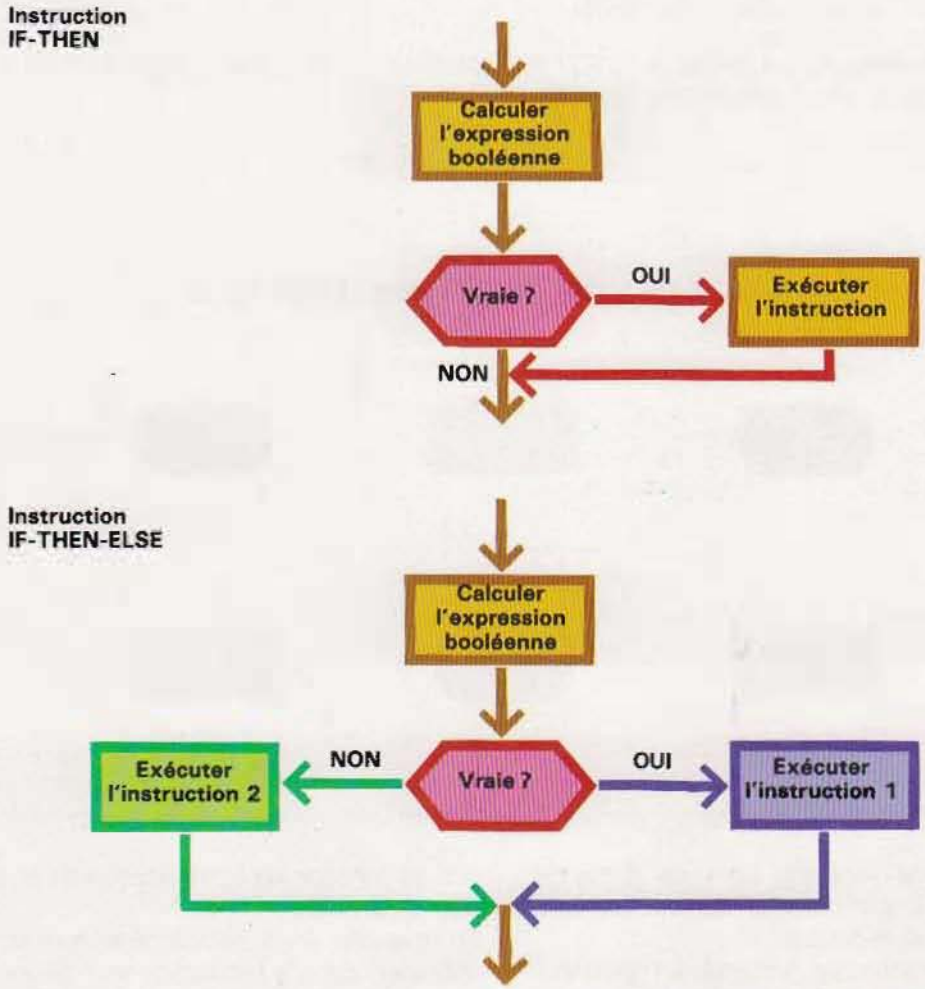
```
WRITELN ('ENTRER DATE DE NAISSANCE');
READLN (JOUR, MOIS, ANNEE);
IF (JOUR > 31) OR (MOIS > 12) THEN
   BEGIN
        WRITELN ('LA DONNEE N'EST PAS CORRECTE; RECOMMENCER ENTREE')
        READLN (JOUR, MOIS, ANNEE)
   END;
.....
```

SYNTAXE DE L'INSTRUCTION IF-THEN-ELSE



Le programme répétera alors sa demande d'entrée de donnée comme l'illustre l'exemple de programme ci-dessus.
La première instruction demande à l'opérateur d'introduire une date de naissance, la seconde de lire et d'assigner les trois variables JOUR, MOIS, ANNEE. Ces valeurs sont successivement contrôlées : si JOUR est supérieur à 31 ou si MOIS est supérieur à 12 (opérateur logique OR), un message d'erreur est imprimé invitant à recommencer l'introduction. En l'absence d'erreurs, le programme est lancé en séquence.
L'exemple est évidemment donné à titre indicatif puisqu'il faudrait également contrôler que les nombres indroduits ne sont pas inférieurs ou égaux à zéro ou que le jour du mois ne sort pas du champ des valeurs autorisées pour chaque mois.
En recourant à IF-THEN, on peut faire en sorte qu'une certaine instruction soit exécutée ou non en fonction de la vérification de la condition de contrôle.
En revanche, il est parfois nécessaire de faire exécuter, selon le résultat du test de contrôle, l'une ou l'autre des deux instructions (ou groupes d'instructions).
En Pascal, ce résultat peut être obtenu avec l'instruction IF-THEN-ELSE, (diagramme syntaxique ci-dessus). Son exécution commence avec l'évaluation de l'expression booléenne : l'instruction 1 (après le mot réservé THEN) si elle est vraie, l'instruction 2 (après le mot réservé ELSE) si elle est fausse. Naturellement, par fin d'instruction on entend aussi bien une instruction simple que composée. La différence existant entre l'instruction examinée et IF-THEN est illustrée ci-dessus.
Revenons à l'exemple précédent relatif à la validation des dates de naissance. Cette nouvelle instruction est utilisable pour écrire une section de programme qui non seulement

COMPARAISON DES INSTRUCTIONS IF-THEN ET IF-THEN-ELSE



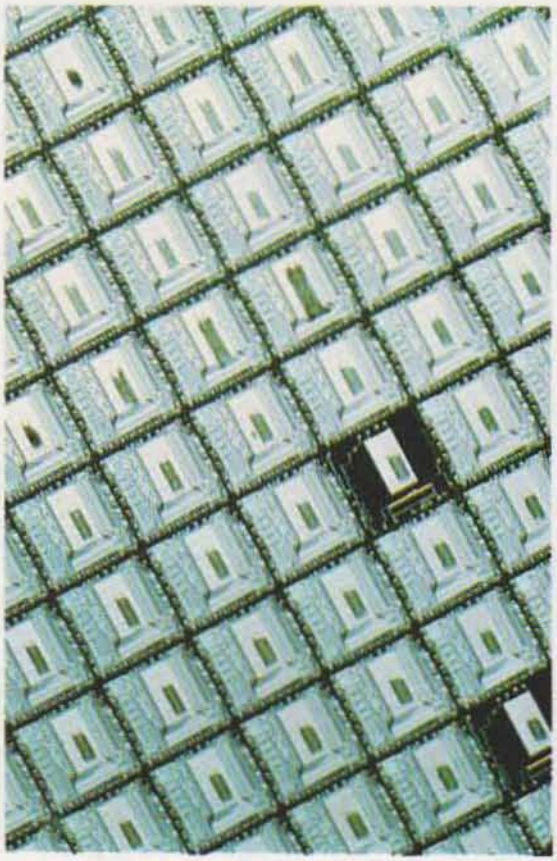
EXEMPLE D'APPLICATION DE L'INSTRUCTION IF-THEN-ELSE

```
WRITELN ('ENTRER UNE DATE DE NAISSANCE');
READLN (JOUR, MOIS, ANNEE);
IF (JOUR > 31) OR (MOIS > 12) THEN
  BEGIN
      WRITELN ('LA DATE N'EST PAS CORRECTE, RECOMMENCER');
      READLN (JOUR, MOIS, ANNEE)
  END (* il est illégal de mettre ici un ; *)
ELSE
  BEGIN
      WRITELN ('LA DATE EST CORRECTE ET EST:');
      WRITELN ('JOUR=',JOUR:5, 'MOIS=', MOIS:5, 'ANNEE=', ANNEE:6)
  END;
  .....
```

avertit l'opérateur que la date est, le cas échéant, erronée, mais aussi visualise à l'écran la date écrite quand elle est correcte (voir page suivante).

L'instruction CASE-OF. Elle est souvent considérée comme une extension ultérieure du IF-THEN-ELSE pour choisir entre plusieurs instructions (simples ou composées) celle à exécuter en fonction de la valeur assumée par une expression.
Une seule des instructions alternatives est déroulée, les autres étant sautées, comme il est représenté dans le diagramme syntaxique ci-dessous et dans le schéma de fonctionnement logique page 1258.

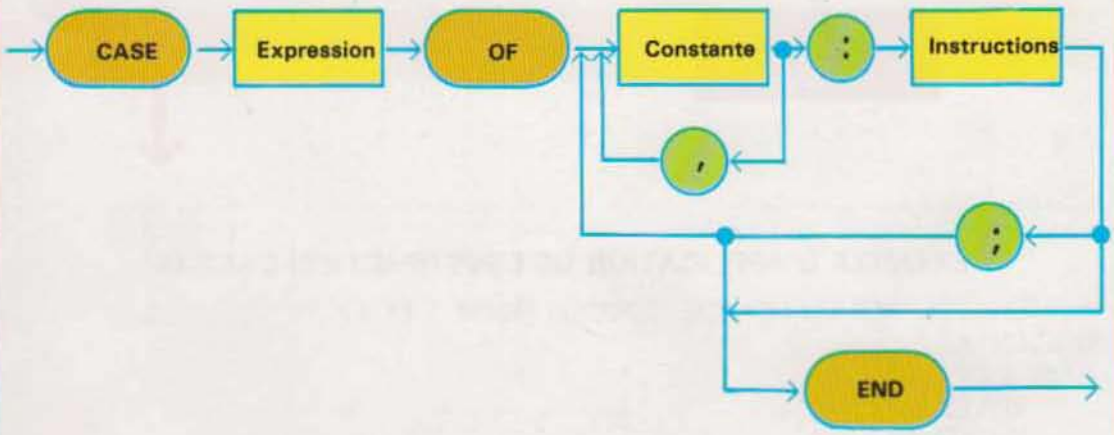
Une fois calculée l'expression venant immédiatement après CASE, sa valeur est comparée à celle des constantes (C1, C2, ... CN) situées après OF (voir exemple décrit page 1258). Si elle est égale à C1, c'est l'instruction 1 qui est exécutée, si elle est égale à la constante C2, c'est l'instruction 2 et ainsi de suite. Dans ce cas aussi, les instructions sont simples ou composées.
Si, à la demande d'introduction d'un nombre, la réponse est 1, le mot UN sera imprimé ; si on indroduit 2, le mot DEUX sera imprimé, etc. La



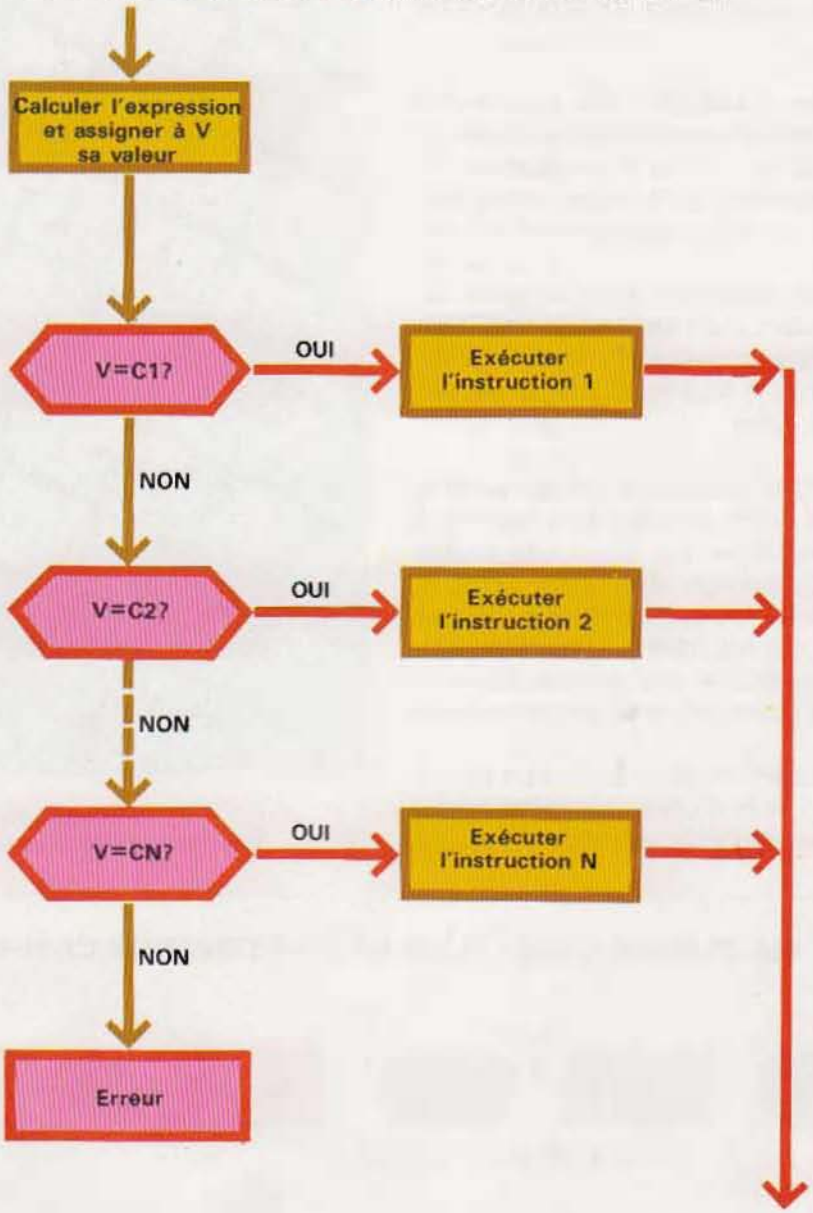
Marka

Circuits intégrés, englobés dans une matrice résineuse externe.

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION CASE-OF



SCHEMA LOGIQUE DE L'INSTRUCTION CASE-OF



EXEMPLE D'APPLICATION DE L'INSTRUCTION CASE-OF

```
WRITELN ('ENTRER UN NOMBRE COMPRIS ENTRE 1 ET 4');
READLN (K);
CASE K OF
  1 : WRITELN ('UN');
  2 : WRITELN ('DEUX');
3,4 : WRITELN ('TROIS OU QUATRE');
END
```

dernière ligne illustre le cas où plusieurs valeurs de contrôle sont associées ; l'instruction est exécutée quand la valeur de la variable est égale à l'une quelconque des constantes listées (C1, C2,... CN).
Si la valeur ne coïncide avec aucune d'entre elles, le résultat de la sélection est indéfini et est signalé comme une erreur. Il faut toujours envisager cette éventualité dans l'emploi de CASE-OF.
En pratique, cette instruction est seulement conseillée lorsque, par exemple, toutes les valeurs assumées par la variable de contrôle sont prédéfinies. On peut alors les insérer dans l'instruction et faciliter la sélection.
Quand on ne connaît pas par avance ces valeurs, l'instruction CASE-OF n'est pas conseillée. La variable de sélection n'est pas nécessairement entière ; elle appartient au type caractère ou à tout autre type défini par l'utilisateur.
Exemple : en désignant le type de variable par JOURS DE LA SEMAINE et en considérant que la variable AUJOURD'HUI appartient à un tel type, on obtient la section de programme mentionnée ci-dessous, qui imprime un des deux messages TRAVAIL ou WEEK-END en fonction du contenu de la variable AUJOURD'HUI.
Noter, au passage, que le mot-clé END sert à délimiter la fin de l'instruction CASE-OF et non à indiquer la fin d'une instruction composée ou du programme.

L'instruction de saut inconditionnel : GOTO

L'instruction GOTO modifie l'exécution séquentielle d'un programme de manière absolue c'est-à-dire non conditionnée par la valeur d'une variable de contrôle.
L'instruction de saut permet de transférer le contrôle d'un point à l'autre du programme, à condition d'avoir prévu d'identifier l'instruction marquant le point d'arrivée du saut. C'est le rôle des étiquettes (labels) qui, comme en Fortran, sont des nombres entiers (compris entre 1 et 9999) suivis du symbole : (deux-points) et des instructions à identifier.
Exemple :

```
10:A:=A+1
```

est une instruction identifiée par l'étiquette 10.
Les étiquettes sont déclarées en début de programme dans la déclaration de LABEL. Celle-ci apparaît dans la section des déclarations **avant** les déclarations de constantes et de variables (syntaxe décrite p. 1260).
Tous les entiers employés dans le programme doivent être énumérés dans la déclaration, comme dans l'exemple :

```
LABEL 10, 20, 30, 40 ;
```

Une fois ces étiquettes définies, on peut décrire l'instruction de saut inconditionnel GOTO (voir syntaxe page suivante).
L'instruction :

```
GOTO 100
```

transfère le contrôle à l'instruction identifiée par l'étiquette 100.
Celle-ci doit apparaître dans la déclaration

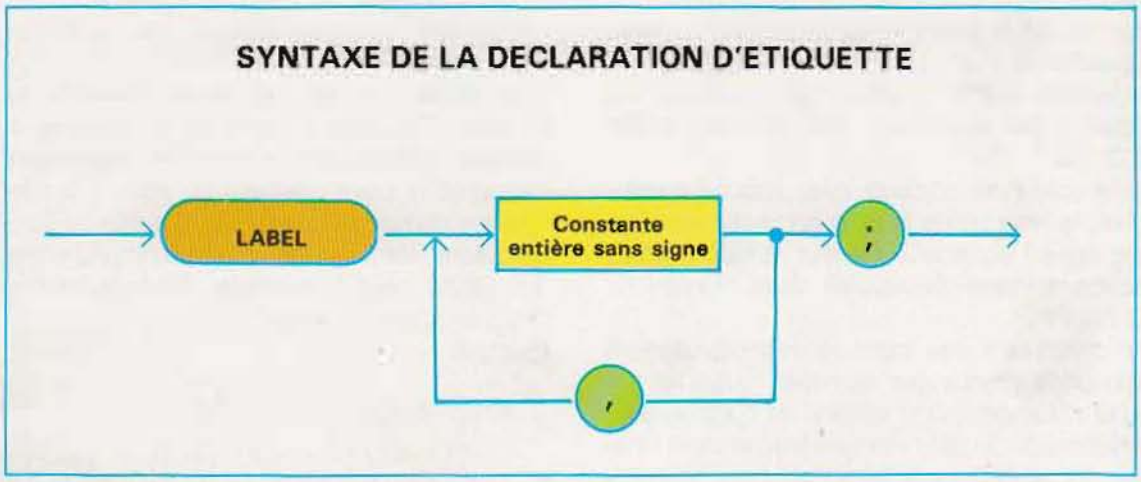
EXEMPLE D'APPLICATION DE L'INSTRUCTION CASE-OF

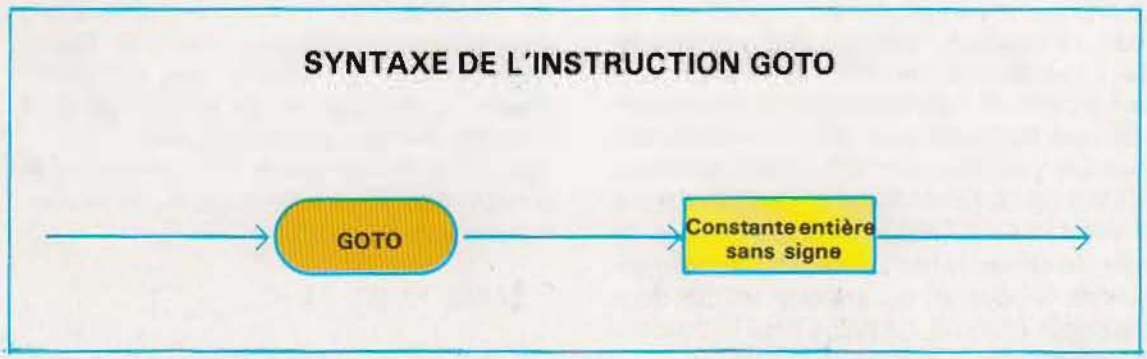
Définition du type

```
JOURS DE LA SEMAINE=(LUN, MAR, MER, JEU, VEN, SAM, DIM);
```

Section du programme

```
CASE AUJOURD'HUI OF
LUN, MAR, MER, JEU, VEN: WRITELN ('ALLEZ AU TRAVAIL!');
SAM, DIM               : WRITELN ('LE WEEK-END EST ENFIN ARRIVE!');
END; (*FIN DE L'INSTRUCTION CASE-OF*)
```





d'étiquette et dans le corps du programme. En Pascal, le recours à l'instruction GOTO est fortement déconseillé, contrairement au Basic ou au Fortran.
Elle est, en effet, reconnue comme superflue au regard de la flexibilité des instructions de contrôle propres au Pascal.
La programmation structurée, dont le Pascal emprunte les concepts de base, tend à éliminer les instructions de saut inconditionnel dont l'emploi aveugle est tourné en dérision par l'expression « spaghetti coding » à cause du nombre élevé de renvois qui en découlent.
En revanche, un programme utilisant exclusivement des structures de contrôle de haut niveau est défini comme **structuré.**

Aspects particuliers du Pascal

Les originalités du Pascal seront examinées dans ce paragraphe pour en approfondir la connaissance.
Première caractéristique importante : les **données structurées.**
En Pascal comme dans les autres langages, il est possible de définir des tableaux. Les indices sont de type logique (BOOLEAN), caractère (CHAR) ou constante.
Les exemples suivants représentent des déclarations de tableaux valides.

```
TYPE
  VECTEUR=ARRAY [1..70] OF INTEGER ;
  TABLE-ARRAY [BOOLEAN] OF CHAR ;
  MATRICE=ARRAY [1..3,1..3] OF INTEGER.
```

Outre l'assignation, toutes les opérations de comparaison (inférieur, supérieur, égal et leurs combinaisons) sont définies sur les éléments d'un tableau.
Ces derniers appartiennent nécessairement au même type. Si on part d'éléments de types différents, on utilise le type **record.**
Sa définition permet, pour chaque élément appelé champ (en anglais field) de spécifier son type et un identificateur.
Ainsi, la carte signalétique d'un élève peut être structurée comme dans le tableau ci-dessous :
— l'identificateur ELEVE personnalise une structure de type record,

— NOM est l'identificateur d'un champ de type ARRAY,
— MATRICULE l'identificateur d'un champ de type INTEGER, etc.
Un autre aspect important du Pascal concerne la **procédure** ou **fonction** représentée par un bloc d'instructions qui peut être rappelé en spécifiant le nom, comme pour les sous-programmes Fortran.
Une procédure est construite avec les mêmes règles de construction de programmes Pascal :
— partie déclarative formée de LABEL, CONST, TYPE, VAR, PROCEDURE, FUNCTION,
— partie exécutive bornée par les mots réservés BEGIN et END.
Dernière caractéristique du Pascal : la gestion de fichiers.
En Pascal, un **fichier** est défini comme une donnée structurée qui contient une série d'éléments de même type, sur support magnétique (disque ou bande).
La recherche à l'intérieur d'un fichier, en Pascal standard, s'effectue seulement en séquentiel, par accès aux enregistrements simples du fichier.
Les déclarations de type fichier suivantes sont valides :

```
TYPE
  MATRICULE=FILE OF INTEGER
  NOMS     =FILE OF CHAR
```

Sur un fichier séquentiel, on travaille en écriture (instructions REWRITE) ou en lecture (instructions RESET). En écriture, les informations sont archivées dans le fichier avec l'instruction PUT, mais, si le fichier est ouvert en lecture, on accède aux enregistrements par l'instruction GET.
L'accès aux données, toujours et seulement de manière séquentielle, est une contrainte très lourde. Aussi, de nombreuses versions du Pascal prévoient également des fichiers à accès direct ou indexé (indexed ou keyed files) qui n'existent pas en standard.



Grazia Neri/Black Star

Circuits imprimés d'un moniteur graphique en cours de montage.

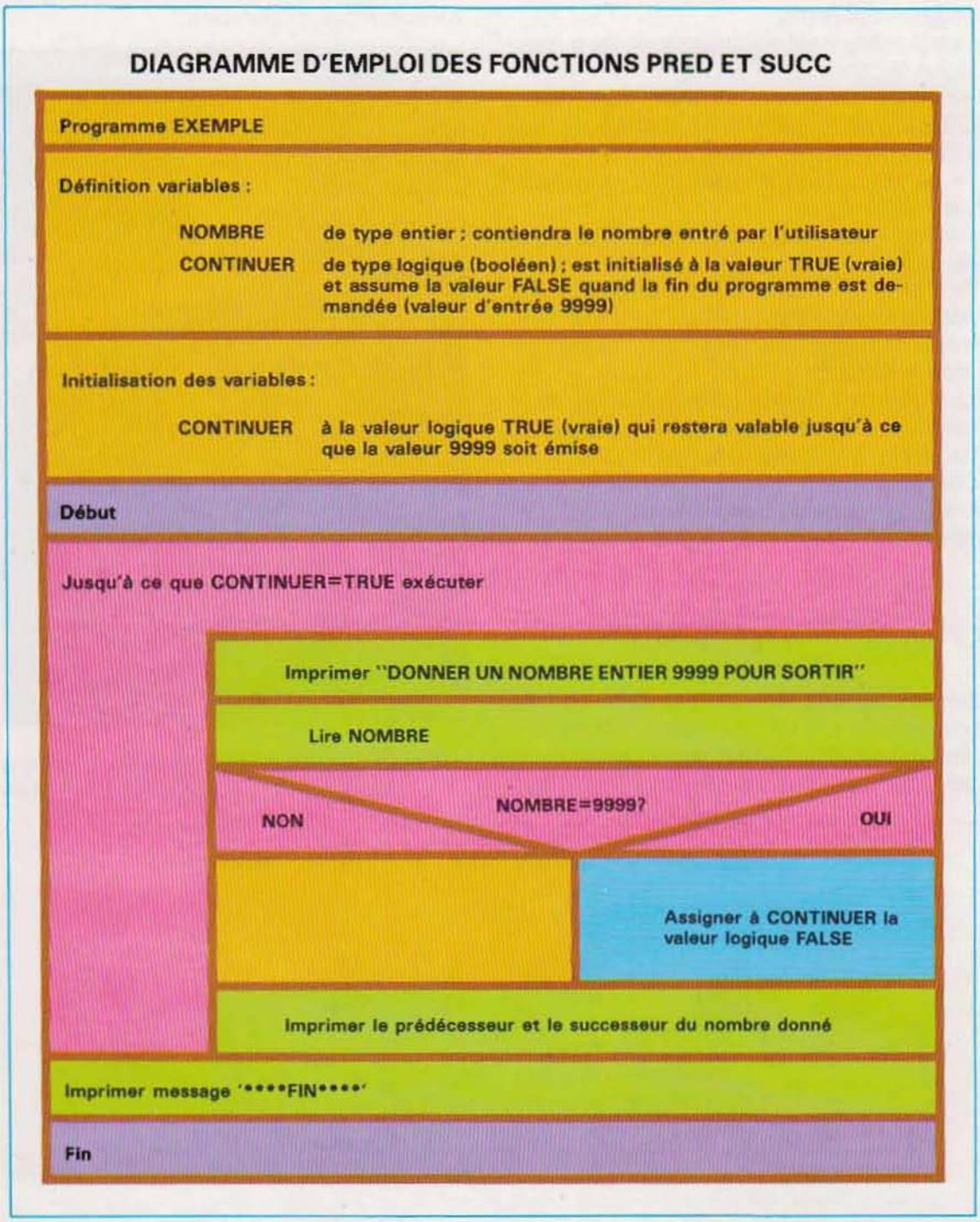
EXEMPLE DE DEFINITION DU TYPE RECORD

```
TYPE
  ELEVE: RECORD
              PRENOM    : ARRAY [1.. 10] OF CHAR;
              NOM       : ARRAY [1.. 20] OF CHAR;
              MATRICULE : INTEGER;
              AGE       : INTEGER;
              ADRESSE   : ARRAY [1.. 30] OF CHAR;
              TELEPHONE : INTEGER;
              CLASSE    : INTEGER;
       END;
```

Exemples de programmation en Pascal

L'organigramme structuré ci-dessous est un exemple d'application sur les fonctions PRED et SUCC, de même que le listing et les sorties (page ci-contre).

Le programme demande à l'utilisateur d'entrer un nombre entier et commande l'impression du prédécesseur et du successeur à la donnée entrée.
Il procède ainsi jusqu'à l'entrée de la valeur 9999, qui est interprétée comme borne de fin d'exécution.



EMPLOI DES FONCTIONS PRED ET SUCC

```
PROGRAM pred_succ (INPUT, OUTPUT);

VAR
   numero : INTEGER;
   continue : BOOLEAN;

BEGIN
   continue := TRUE;
   WHILE continue DO
      BEGIN
         WRITELN ('Donnez un numéro (9999 pour finir) : ');
         READLN (numero);
         continue := NOT (numero = 9999);
         WRITELN ('numéro précédent :', PRED (numero));
         WRITELN ('numéro suivant   :', SUCC (numero))
      END;
   WRITELN;
   WRITELN;
   WRITELN;
   WRITELN ('  **** FIN ****')
END.
```

```
Donnez un numéro (9999 pour finir) :
numéro précédent :                   3
numéro suivant   :                   5
Donnez un numéro (9999 pour finir) :
numéro précédent :                 -11
numéro suivant   :                  -9
Donnez un numéro (9999 pour finir) :


numéro précédent :                  -1
numéro suivant   :                   1
Donnez un numéro (9999 pour finir) :
numéro précédent :                9998
numéro suivant   :               10000


  **** FIN ****
```

Le contrôle de fin d'exécution est effectué par l'instruction WHILE-DO. Celle-ci répète le cycle jusqu'à ce que l'utilisateur décide d'en sortir.

Comme on peut le constater, les fonctions PRED et SUCC sont directement insérées dans l'instruction d'impression, simplifiant ainsi considérablement le programme.

L'emploi des instructions READ et READLN est illustré en page 1264 ainsi que par le listing correspondant.

Lors de l'émission des deux premières données et du caractère alphanumérique, les mêmes données sont imprimées.

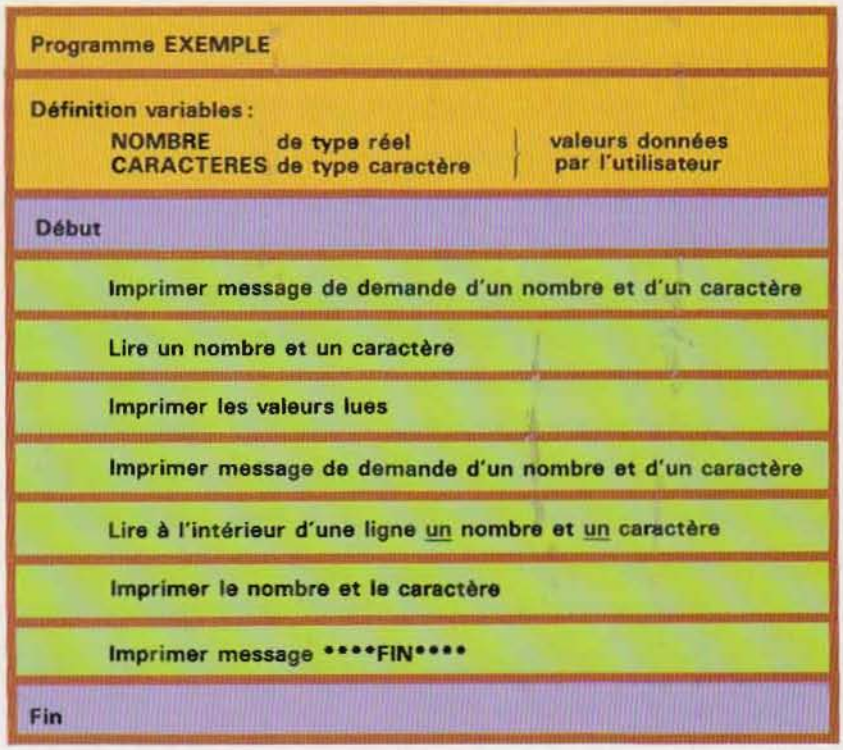
A la seconde fois, l'opérateur doit appuyer sur la touche RETURN afin de terminer la lecture et déclencher ainsi l'impression.

En effet, dans le premier cas, les valeurs sont lues jusqu'à ce que la liste des variables du READ soit épuisée alors que, dans le second cas, la lecture est fermée par l'émission du caractère CR (Retour Chariot).

Si, en utilisant le READLN, on s'aperçoit que sont émises un plus grand nombre de valeurs qu'il ne s'avère nécessaire, le programme examine seulement celles qui épuisent la liste des paramètres (dans l'exemple seulement le premier nombre et le premier caractère), les autres restant ignorées.

On trouvera un exemple d'application du WHILE-DO, ainsi que le listing correspondant, en page 1265.

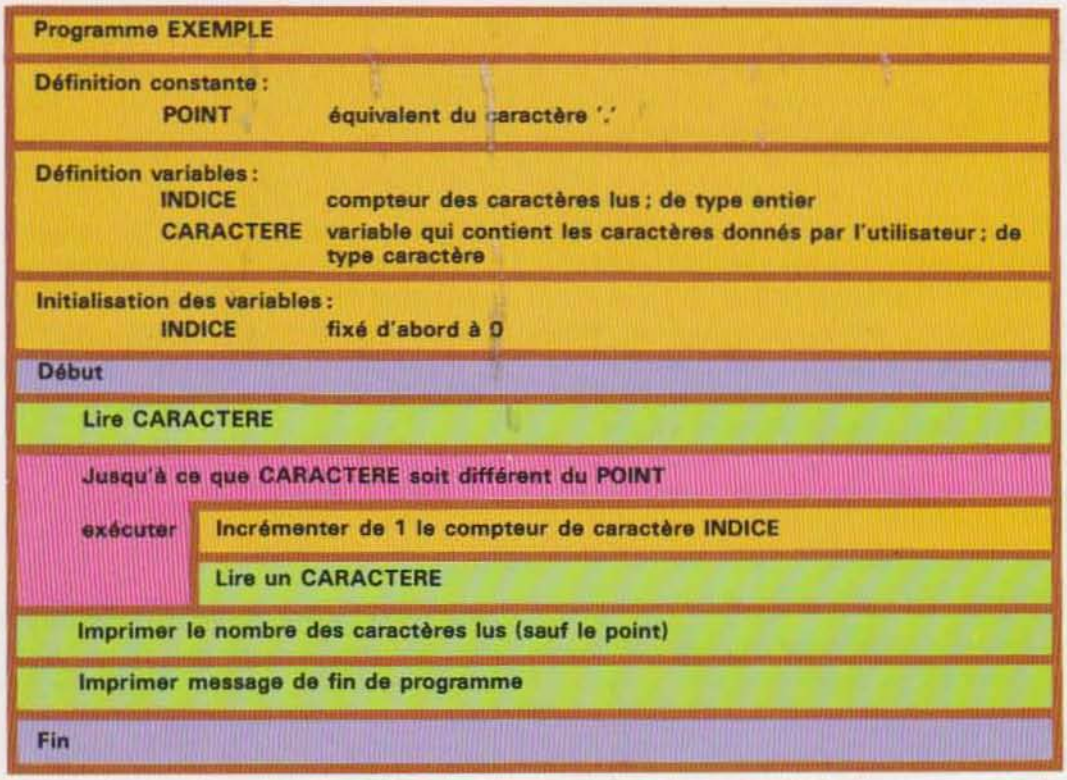
APPLICATION DES INSTRUCTIONS READ ET READLN



PROGRAMME D'APPLICATION DES INSTRUCTIONS READ ET READLN

```
PROGRAM lire_ecrire (INPUT, OUTPUT);

VAR
   numero : REAL;
   caractere : CHAR;

BEGIN
   WRITELN ('Entrez un numéro et un caractère :');
   READLN (numero, caractere);
   WRITELN;
   WRITELN (numero:5:2, '   ', caractere:1);
   WRITELN;
   WRITELN;
   WRITELN ('Entrez un numéro et un caractère :');
   READLN (numero, caractere);
   WRITELN;
   WRITELN (numero:5:2, '   ', caractere:1);
   WRITELN;
   WRITELN ('  **** FIN ****')
END.

 Entrez un numéro et un caractère :
 -1.45g

 -1.45   g

 Entrez un numéro et un caractère :
 .14580
0.15   0

 **** FIN ****
```

DIAGRAMME D'UTILISATION DE L'INSTRUCTION WHILE-DO



PROGRAMME D'APPLICATION DES INSTRUCTIONS WHILE-DO

```
PROGRAM compte (INPUT, OUTPUT);

CONST
   point = '.';

VAR
   cumul : INTEGER;
   caractere : CHAR;

BEGIN
   cumul := 0;
   WRITELN ('Entrez une phrase terminée par un point :');
   WRITELN;
   READ (caractere);
   WHILE caractere <> point DO
      BEGIN
         cumul := cumul + 1;
         READ (caractere)
      END;
   WRITELN;
   WRITELN ('Vous avez introduit ', cumul:3, ' caractère(s)');
   WRITELN;
   WRITELN ('  **** FIN ****')
END.

Entrez une phrase terminée par un point :

ceci est une phrase terminée par un point.

Vous avez introduit  41 caractère(s)

  **** FIN ****
```

Résumé : développement d'une fonction de facturation

Cet exemple d'application a été choisi pour analyser les principales fonctions qui sont normalement traitées par un ordinateur. Nous examinerons notamment une procédure complexe de gestion de fichiers et des masques ainsi que les techniques d'arrondi sur les valeurs numériques. En partant d'un problème spécifique, on a voulu donner un exposé simple appuyé sur des exemples concrets et qui ne demandent aucune connaissance particulière.

Analyse du problème

Le but de la procédure est de gérer une série de transactions qui déterminent le mouvement des articles d'un magasin ou des importations de vente et d'achat. Tous les flux de données s'articulent sur ces deux voies (ventes/achats). N'importe quelle activité de nature commerciale demande, au minimum, la gestion des fichiers suivants :

— Clients (CLI)
— Fournisseurs (FOU)
— Articles (ART)
— Caisse (CAISSE)

Les fiches signalétiques et les montants (créance pour les clients, dettes pour les fournisseurs) sont enregistrés dans les fichiers

Présentation du système de traitement Apple.



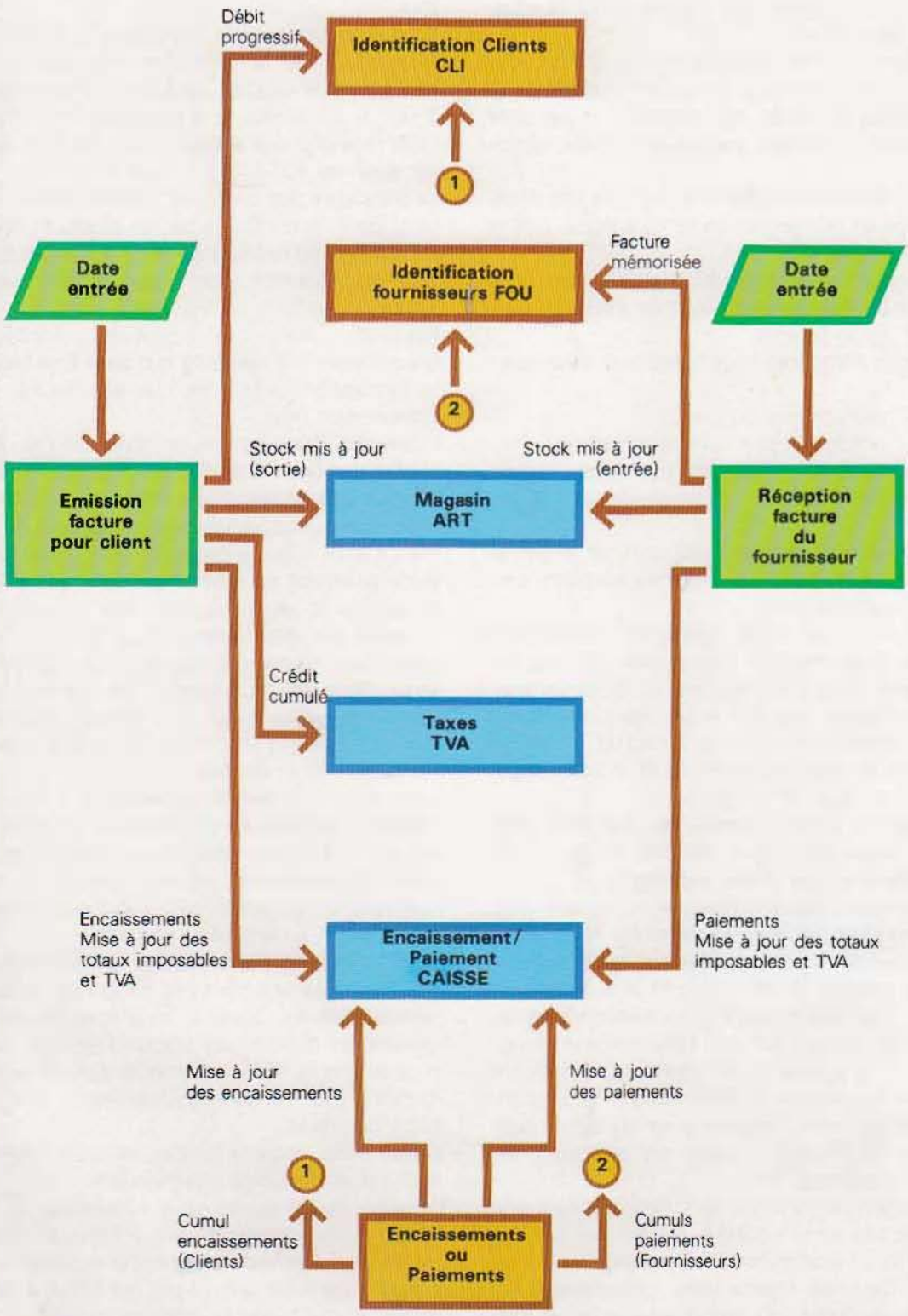
G. Pochetti/Marka

Clients (CLI) et Fournisseurs (FOU). Le fichier Articles (ART) contient les descriptions des marchandises mouvementées et le fichier CAISSE est utilisé pour mettre à jour la situation financière.
L'existence et le contenu de ces fichiers doivent être considérés de manière souple ; par exemple, le fichier ART pourrait ne pas être présent, ou encore concerner d'autres activités.
Avec quelques adaptations, la procédure développée ici conservera toute sa validité, même dans ce cas.
Le document de base, à l'origine d'un mouvement de marchandise ou d'un transfert d'argent, est la facture.
Elle doit comporter les informations suivantes :

— signalement du client
— articles et prix correspondants
— taux de TVA et montants correspondants

Les deux premiers champs ont une signification évidente alors que le dernier nécessite certains commentaires.
La TVA est un impôt calculé en pourcentage sur le montant d'une transaction ; ce taux dépend du type d'article objet de la transaction. Il peut arriver que l'on doive, dans une même facture, calculer un montant total de la TVA qui résulte de montants obtenus chacun en appliquant un taux différent.
La facture est soit **émise,** elle concerne alors une vente (d'un bien ou d'un service), soit **reçue** (il s'agira d'une acquisition).
En fonction des informations contenues dans les factures, on doit décrémenter le stock (la marchandise sortie) et simultanément augmenter le compte du client auquel elles se rapportent, ainsi que la somme encaissée. Dans les factures reçues, on doit incrémenter le stock (c'est une arrivée de marchandise), le montant dû au fournisseur et les sommes dépensées. Il faut, en outre, mémoriser les données relatives aux charges fiscales, en particulier les montants de la TVA.
En jargon commercial, la TVA s'appelle « une partie tournante » c'est-à-dire un montant qui peut être transféré entre les personnes intervenant dans les transactions successives. Par exemple, lorsqu'on reçoit une facture d'un fournisseur d'un montant 1 000 F plus 70 F de TVA (7 %), le dû sera 1 070 dont 70 comme impôt que le fournisseur devra verser à son tour.
Si ce bien est revendu, supposons à 1 500 F, le montant de la TVA correspondant sera 105 F (même taux) et l'acquéreur devra verser 1 605 F. En réalité, une partie de cette TVA (105 F) a déjà été versée (70 F), la TVA due est donc de 105 — 70 = 35 F.
La procédure doit mettre en évidence les montants de TVA encaissés par les clients et ceux versés aux fournisseurs de façon à pouvoir calculer la différence (voir organigramme page 1268).
Apparaît alors la nouvelle fonction Encaissements/Paiements qui, sans être liée à la facturation, n'en constitue pas moins un complément utile.
L'émission d'une facture provoque un mouvement des montants du seul point de vue comptable. A l'égard des charges fiscales, le chiffre indiqué dans une facture émise intervient comme s'il avait déjà été encaissé, alors qu'en réalité le paiement peut ne se produire que longtemps après et même être fractionné (par exemple par des règlements à 60 ou à 90 jours). Pour toutes ces raisons, il est indispensable de gérer la situation Encaissements/Paiements qui se réfère à un véritable mouvement de caisse et fournit ainsi l'état des liquidités (créances et dettes).
La gestion de la caisse n'est soumise à aucune obligation ou formalité particulière et peut donc être structurée selon les nécessités ou les habitudes de l'entreprise. Le seul objectif est de maintenir une situation mise à jour des échanges d'argent en entrée et en sortie.
La connaissance d'une situation globale en dettes et en créances n'est pas suffisante : il faut personnaliser les clients ou les articles respectivement en débit et en crédit. Toutefois, les encaissements et les paiements doivent aussi mettre à jour les données relatives à chaque client ou article.
Il faut, enfin, prendre en considération l'éventualité d'un retour de marchandise.
Dans ce cas, il existe deux possibilités. Si la facture correspondante a déjà été émise, il faut retirer les sommes correspondantes avec un « avoir » ; si la facture n'a pas été émise, il faut seulement réintégrer le stock du magasin.

ORGANIGRAMME DE LA FONCTION FACTURATION



Dans notre analyse du livre : ***« Le système Aristote »*** *de* ***René Dzagoyan,*** *publié chez Flammarion, nous avons suivi pas à pas comment une grande puissance mondiale (en l'occurrence les Etats-Unis) en était arrivée à se confier quasiment entièrement aux ordinateurs, et comment elle avait réussi à entraîner dans son orbite la quasi-totalité des pays du monde occidental. Pour n'être pas en reste, les pays du bloc soviétique en font autant.*

Nous en sommes arrivés au moment où la pieuvre occidentale prend ses aises en étendant ses longs tentacules, sous couvert, tout d'abord, d'activités économiques, ce dont ne manque pas de s'étonner un des protagonistes, bien conscient de l'importance primordiale des questions militaires :

« Et tout cela, simplement pour faire des calculs économiques ! »

La réponse, comme vous pouvez bien le penser, ne se fait pas attendre :

« Pas seulement... nous l'utilisions aussi pour le commerce international. Par exemple, chaque usine importante était munie d'un petit ordinateur branché sur l'ordinateur central... Un programme spécial lui disait quelle devait être sa production dans les douze prochains mois. S'il était d'accord avec l'ordinateur, il appuyait sur un bouton, et tous les robots de son entreprise ou tous les plans de ses contremaîtres se réglaient en fonction de l'objectif défini...

Une véritable ***pieuvre,*** *à un point que vous n'imaginez pas. Il suffisait d'installer, chez soi, un terminal relié à l'ordinateur central pour recevoir, sur son écran, des cours de physique en provenance de l'Italie ou des cours de cuisine du Japon. En poussant plus loin, vous pouviez brancher votre cuisinière électrique sur votre terminal et la faire régler par un programme culinaire élaboré et enregistré de l'autre côté de la terre.*

— Pourtant, les cours de cuisine n'ont jamais tué personne !

— Vous êtes bien naïf ! Vous vous imaginez bien que le système a un usage essentiellement ***militaire.*** *Sans cela, les Etats utilisateurs ne nous auraient même pas accordé un quart d'heure d'attention. Quant aux crédits, n'en parlons pas... »*

Vous savez, maintenant, ce qu'est Aristote... Mais en êtes-vous bien sûr ? S'agit-il seulement d'un ensemble de possibilités au service de l'homme ? C'est oublier qu'une somme de choses peut valoir plus que ses composants. « Le système Aristote est maintenant en contact permanent avec un nombre incalculable d'ordinateurs. Il sera probablement le plus grand ***cerveau électronique*** *que l'humanité aura jamais créé.*

Je crois que ce système peut atteindre ce que j'appelle une ***« masse critique »*** *qui, une fois atteinte et franchie, lui confère une intelligence analogue à l'intelligence humaine. Dans ce cas, le système Aristote serait capable de déterminer lui-même ses objectifs, d'établir lui-même la stratégie permettant de les atteindre, et de refuser toute instruction humaine qui les contredirait. En un mot, Aristote serait un système totalement* ***indépendant de la volonté humaine. »***

Plus tard, on apprend que le système russe est similaire au système occidental et que ce dernier le manipule, puisqu'il sait exactement comment il est fait. On apprend, au passage, ce qui nous intéresse tout particulièrement, que la stratégie fondamentale du système est fondée sur le ***jeu de Gô****. Les choses ne s'arrangent pas lorsqu'on apprend que le système vient d'activer les bombes à neutrons dont la caractéristique la plus intéressante est d'être* ***« propres »,*** *c'est-à-dire de détruire tout ce qui est vivant sans endommager, en quoi que ce soit, tout ce qui est inerte. Dont les précieux ordinateurs :*

« Les seules choses qui restent intouchées sont les machines. Le règne minéral est insensible aux flux neutroniques. Pourquoi le Système cherche-t-il à épargner les machines ? »

La réponse ne tarde pas, d'abord sous son aspect philosophique :

« Dans la finalité du programme, réside la réalisation matérielle de la ***Rationalité,*** *but ultime d'Aristote. »*

Conséquence pratique :

« Aristote considère que l'espèce humaine s'oppose à la réalisation du But Ultime, et qu'il se trouve malheureusement dans l'obligation de procéder à la déshumanisation de la Terre. »

Logique, n'est-ce pas ?

On est, cependant, prié de ne pas se laisser aller à la peur. Comme le déclare un des généraux du livre : ***« Les hommes n'ont que les guerres qu'ils méritent. »***

Mais sommes-nous si méritants que cela ???

Les jeux vidéo (suite) ou les aventures de Fred

Reprenons, à présent, l'étude de ce très intéressant jeu de guerre que constitue le Gô. Nous en avons étudié la philosophie, l'histoire et les règles, puis nous nous sommes accordé une étape pour réflexion sur les ***implications de l'ordinateur*** *dans la vraie guerre (qu'elle soit froide ou chaude).*

Voilà qui nous met tout a fait à l'aise pour étudier l'adaptation du Gô à l'ordinateur.

Il faut tout d'abord se rendre compte que les règles du jeu de Gô sont infiniment plus complexes qu'elles ne sont apparues au cours de cette brève étude, et qu'un jeu de guerre capable de conquérir le monde comporte infiniment plus de ressources cachées qu'il n'apparaît au cours d'une brève étude. Nous avons volontairement simplifié ces règles, tout en essayant de ne pas en diminuer la signification et de ne pas en biaiser la nature, dans l'optique où nous allons examiner une version simplifiée du Gô, destinée à des joueurs mixtes : un automate, un être humain.

C'est pour cette raison que notre automate jouera sur un ***Gô Ban réduit*** *à 9 × 9 intersections. Les autres règles utiles pour développer correctement le Gô sont décrites dans le tableau de la page 1226 du numéro précédent.*

Pour pouvoir construire un ***adversaire artificiel,*** *un automate avec lequel nous jouerons au Gô, nous devons, au préalable, analyser les critères qui permettent de développer correctement la dynamique du jeu.*

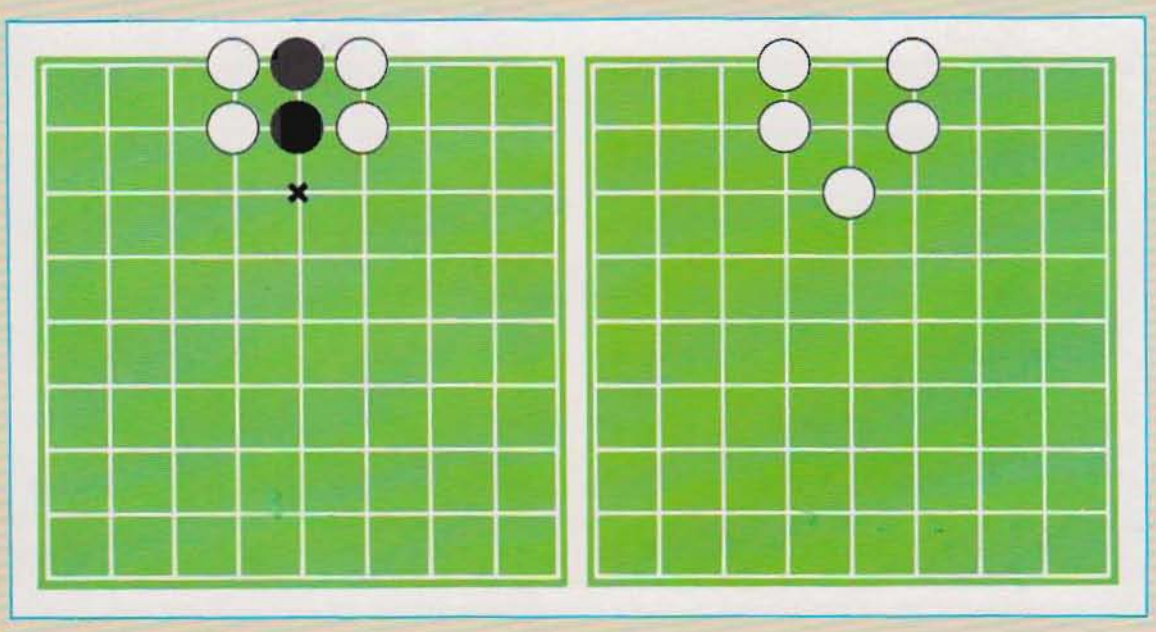
Organiser un ***programme*** *signifie construire une source de signaux de contrôle qui gouvernent les manifestations de l'automate vers l'extérieur. Comme il n'est pas très sympathique d'avoir affaire au programme « Machin », nous habillerons notre compagnon de jeu d'une personnalité qui nous fasse un peu rêver.*

Je ne sais pas si vous avez déjà rencontré Fred au détour d'une bande dessinée américaine, mais je ne vois pas pourquoi nous différerions plus longtemps de faire connaissance avec ce sympathique lion. D'autant plus que son heureux caractère lui permet de subir, avec une patience particulièrement résignée, les expériences aussi sottes et grenues (pardon pour ce jeu de mot qui prétend parler d'expériences saugrenues) qu'un de ses amis, savant et farceur, lui fait régulièrement subir en tant que cobaye.

Les procédures ne seront pas exprimées dans un langage de programmation mais en français

La position des pions sur le Gô Ban montre deux pions noirs en Atari.

Ici, le blanc a joué en capturant les pions noirs.



structuré, ce qui permettra leur implantation dans les différents langages à la disposition de l'utilisateur.

Notons que nous avons la possibilité d'utiliser un langage qui se prête très bien à la **construction de Fred : le Logo.** Celui-ci est commercialisé dans diverses versions adaptées à des calculateurs personnels. Pour ce cas, c'est la version A1.5 du Logo, sur le calculateur MPF3 de Digitek, qui a été utilisée, afin d'élaborer une série expérimentale de procédures destinées à l'étude et au développement de Fred.

Il est clair que la quasi-totalité des procédures que nous allons définir aura pour but de décrire, chaque fois, une portion du jeu qui sera alternativement gérée par l'être humain ou par Fred lui-même. Lorsque son implémentation deviendra complète, nous prendrons beaucoup d'intérêt à jouer avec lui. En effet, au début, nous aurons un peu bafouillé et ses coups nous paraîtront dépourvus de subtilité, mais Fred s'y fera vite...

Passons au concret. La première opération que nous devons effectuer est la construction du Gô Ban et des pions, objets au travers desquels nous aurons la représentation de coups effectués. Ce programme particulier ne concerne que la **gestion de l'écran.** Celui-ci ne fait pas partie de Fred mais constitue, comme on l'imagine, un complément nécessaire. Nous le décrirons donc en premier lieu, afin de mettre au clair les exigences de **visualisation du jeu.** Car une bonne partie du charme réside dans le fait que nous fournissons à Fred un environnement soigné.

Nous savons bien qu'une grande partie du charme d'un jeu réside dans l'importance des rêves que nous pouvons y accrocher. Et, à ce propos, si les constructeurs de jeux vidéo n'inventent pas toujours des logiciels bien originaux, ils soignent tout particulièrement la **présentation,** l'apparence étant, comme on le sait bien, très porteuse de fantasmes.

Pour ce qui concerne nos rapports avec Fred, on peut risquer la remarque suivante : aujourd'hui, bien des gens, tout en restant fort intéressés par la logique des jeux de guerre, n'arrivent plus à projeter sur eux des fantasmes satisfaisants. Sans doute parce qu'ils risqueraient d'être bien trop proches de l'apocalypse. Et surtout parce qu'ils ne voient pas comment ils réussiraient à dominer facilement la situation, le cas contraire n'étant pas particulièrement gratifiant.

Tandis qu'avec notre sympathique Fred, tout devient facile : ce n'est pas ce lion débonnaire qui cherchera à créer des ennuis profonds à son adversaire ! Tout au plus l'agacera-t-il suffisamment pour que son intelligence se sente mise en appétit par cet adversaire infatigable et d'une constante urbanité.

Rappelons que le Gô Ban, qui est en réalité une grille carrée de 19 × 19 intersections par côté, sera, dans notre cas — et pour des raisons de simplicité — réduit à de plus petites dimensions : 9 × 9 intersections.

Il sera présenté sur l'écran par le calculateur et devra fournir les informations essentielles : la position des pions encore disponibles pour chaque joueur, les indications relatives aux pions prisonniers, ainsi que l'éventuelle déclaration d'**Atari** de la part d'un des deux adversaires.

Il existe, en effet, une étape de la partie au cours de laquelle un groupe de connexion pos-

DESCRIPTION DU MODULE CARTE

```
CARTE (couleur, x)
DEBUT
  SI il y a un pion en x, FAIRE
     il est de la couleur"couleur", FAIRE
     il n'a pas été marqué
  ALORS
     DEBUT
        Marquer le
        RAPPELER CARTE (couleur, NORD (x))
        RAPPELER CARTE (couleur, EST (x))
        RAPPELER CARTE (couleur, SUD (x))
        RAPPELER CARTE (couleur, OUEST (x))
     FIN
  SINON
     DEBUT
        S'il n'y a pas de pion en x
        ALORS
           DEBUT
              Marquer le point comme liberté
              Incrémenter le compteur des
              libertés
           FIN
     FIN
FIN
```

Avec le module CARTE Fred cherche, trouve et décrit les libertés d'un groupe de connexion contenant un pion de couleur "couleur" dans le point X. CARTE rappelle lui-même de façon récursive, en mémorisant X.

sède une seule liberté. La situation est telle, alors, que l'adversaire s'apprête à le capturer. Dans le Gô, tout comme dans les Echecs (dans ce cas, on procède à l'annonce « échec » ou « échec et mat »), le joueur doit avertir l'adversaire de ses intentions offensives. Ce qui est directement inspiré des règles de chevalerie tant orientale qu'occidentale. On laisse donc une chance ultime à l'adversaire de rassembler ses forces — ici, en l'occurrence ses dernières miettes d'astuce — afin d'au moins essayer de s'en tirer honorablement.

Un exemple de coup, qui exige une déclaration d'Atari, se trouve illustré en page 1270, sur l'échiquier de gauche.

Le programme idéal devrait faciliter l'évocation de tous... ou du moins d'une **grande partie des coups** effectués par les joueurs. Cette exigence peut être résolue selon les deux possibilités suivantes.

La première est simplement de mémoriser la représentation graphique d'un coup (c'est-à-dire la situation du Gô Ban à un moment donné) et de construire, dans le programme qui gère le jeu, un espace dans lequel mettre en ordre temporel toutes les images graphiques apparues sur l'écran.

On obtiendra, en les redemandant en ordre temporel croissant ou décroissant, une description totale ou partielle de la partie.

Examinons, à présent, la seconde solution. En effet, celle que nous venons d'évoquer manque un peu de subtilité. Elle occupe inutilement beaucoup d'**espace mémoire,** elle n'est dirigée par aucun **algorithme interactif ;** enfin, elle ne **stimule** guère **l'imagination** du programmeur. En un mot, il convient de mettre la médiocrité sur le paillasson !

Dans la seconde solution, une routine prend en considération un pion déposé sur le Gô Ban comme provenant d'un coup du joueur ou de Fred. Elle détermine l'endroit du coup sur le Gô Ban et appelle une seconde routine qui contient un algorithme qui dessine le Gô Ban et les pions sur l'écran.

Subsiste encore le problème de l'input du coup précédent. Il peut être fourni au calculateur à l'aide d'une description alphanumérique correspondant aux références d'intersection, en attribuant un code d'identification à chaque ligne et à chaque colonne.

Autre solution pour l'indication du repérage : la

DESCRIPTION DU MODULE JEU

```
JEU
DEBUT
  Déposer les pions noirs de handicap
  TANT QUE la partie dure, FAIRE
    DEBUT
      Visualiser le Gô Ban
      Faire la saisie du coup Blanc
      (entrer au clavier)
      APPELER EFFET COUP BLANC
      (pour examen de l'effet)
      APPELER EFFET COUP NOIR
      (pour effet coup du Noir)
      APPELER MODELES
      (pour chercher un modèle de référence)
      Poser le pion noir
    FIN
FIN
```

Le module JEU constitue le corps central du programme que nous avons appelé.

manipulation de divers dispositifs, dont la **souris,** la **table graphique,** le **joystick.**

Ensuite, Fred doit correctement interpréter la signification des figures de jeu pour développer alors son activité d'évaluation et de proposition de son coup.

La routine Carte permettra donc d'identifier les groupes de connexion et les pions les constituant (rappelons qu'ils vont du simple pion jusqu'aux agglomérats de grandes dimensions), en recherchant et en marquant pour chacun d'eux les libertés correspondantes.

La procédure d'identification sur :

— un pion à la fois dans le groupe de connexion,

— une liberté à la fois dans le groupe de connexion,

— la recherche systématique d'un pion de l'autre couleur à proximité du groupe de connexion concerné se déroule sur chaque intersection du Gô Ban. Fred mènera l'enquête : il fixera ainsi dans sa mémoire une représentation correcte des pions sur le Gô Ban.

Ceci se fera en « sous-entendant » de manière significative les indicateurs relatifs à ce point du Gô Ban et qui se réfèrent aux divers paramètres d'identité du pion examiné (avec ses différentes attributions typiques (liberté, insertion dans un groupe...) qu'il présente à ce moment-là.

(suite page 1295)

Fichiers utilisés

Le format des enregistrements des principaux fichiers (p. 1274) comporte une zone initiale de 3 caractères (octets 1 à 3) pour le code d'identification. En Basic, ce code n'est pas nécessaire car les fichiers sont à accès direct. Le code correspond au numéro d'enregistrement : le premier article introduit en stock occupe l'enregistrement 1 et a le code 1, le second occupe l'enregistrement 2 et a le code 2...
Cette zone a été créée à la fois pour la procédure de facturation en Cobol et pour donner une plus grande souplesse au programme. En effet, il est fréquent qu'un utilisateur final conserve son propre système de codification des articles, des clients et des fournisseurs.
Si la gestion d'un magasin n'emploie pas de code externe, chaque article est identifié par son numéro d'entrée (n° d'enregistrement). Cependant, certains préfèreront adopter une abréviation (un code non numérique) qui indiquera de façon mnémonique le type de marchandise.
Ce second cas soulève quelques difficultés. En effet, alors que pour un numéro d'enregistrement la lecture des données relatives à un article est immédiate (sa position dans le fichier étant indiquée par le code) quand le code est une abréviation arbitraire, il faut effectuer une recherche pour chaque article. Le temps nécessaire à cette recherche n'est acceptable que pour un nombre d'articles réduit. Au delà, il est alors indispensable que les marchandises en stock soient ordonnées par code, pour effectuer une recherche de type binaire, par exemple, ce qui accélère le processus.
Le nombre d'articles pour lequel on devra recourir à l'une ou l'autre de ces deux méthodes (recherche dans un fichier ordonné) est difficile à évaluer a priori. Il est nécessaire d'analyser chaque cas particulier, en tenant compte aussi du langage et du matériel utilisés. Pour "normaliser" le format, la zone consacrée a été prévue même dans les fichiers ne la nécessitant pas absolument ; cette zone ne contient alors que des caractères de remplissage.
Les formats représentés page 1274 ne concernent que la zone des données. Les éventuels enregistrements répertoires sont gérés séparément. Ils contiennent des zones d'importances diverses (par exemple le nombre d'enregistrements occupés, la longueur totale, etc.). Toutefois, en Basic, les enregistrements ont normalement une longueur fixe, et ceux du répertoire la même longueur que pour les données. Les nombres inscrits au-dessus de chaque enregistrement représentent la longueur en octets et la position de chaque zone. La zone de description de l'enregistrement du fichier ART, par exemple, comporte 20 caractères et occupe, dans l'enregistrement, les octets 4 à 23. Cette forme de représentation graphique est très utile tant pour le minutage que pour la mise au point ; elle est toujours utilisée en complément de la documentation du programme, quel que soit le langage adopté.

Menu principal

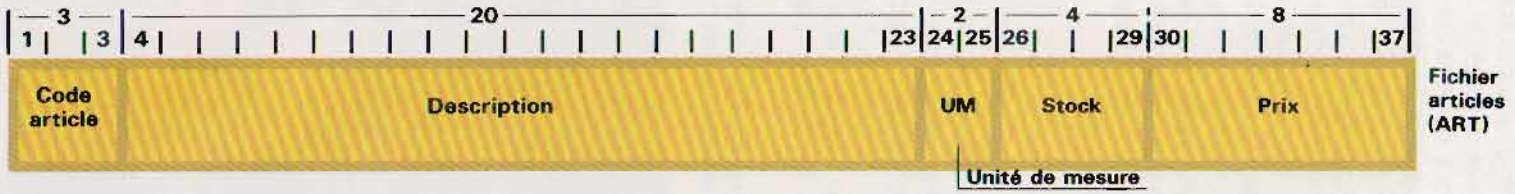
Le menu principal de la procédure est représenté par l'organigramme de la page 1276. Aucune particularité importante n'étant à signaler, son listing n'est pas fourni. La page 1277 détaille le schéma logique d'ensemble. En entrant le numéro de la fonction sélection-

Assemblage de la carte mère d'un ordinateur personnel.

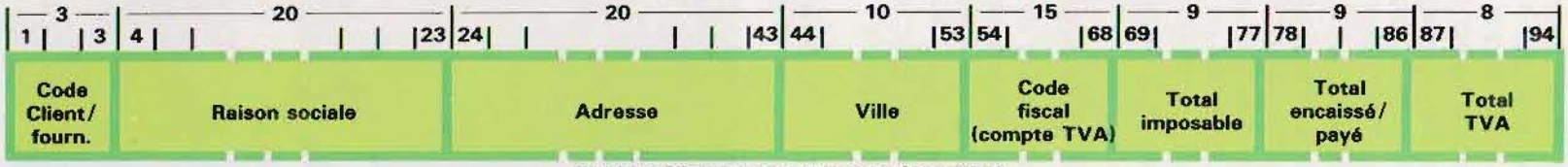


J.P. Laffont/Grazia Neri-Sygma

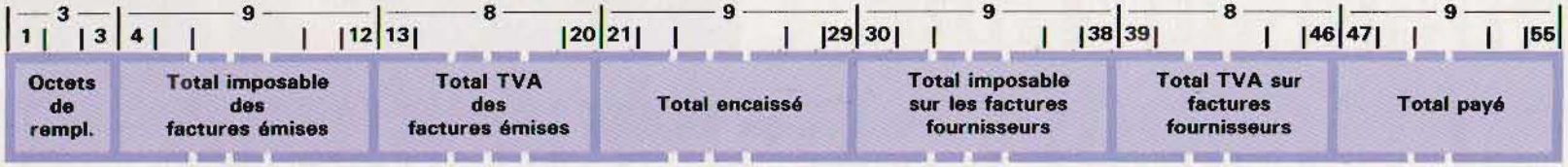
FORMAT DES ENREGISTREMENTS DE FICHIERS



Fichier articles (ART)



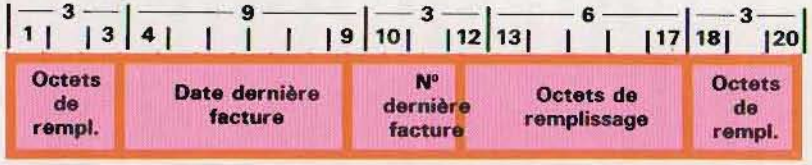
Fichiers Clients et Fournisseurs (CLI, FOU)





Fichier taxes TVC

Uniquement pour le fichier TVA (sinon, octets de remplissage)



Fichier répertoire (DIR)

FICHIERS UTILISES DANS LA PROCEDURE DE FACTURATION

Nom	Long. de l'enreg. en octets	Description
REP	20	Contient des informations de caractère général, comme la date et le numéro de la dernière facture émise. La numérotation des factures doit être progressive ; il faut donc mettre en mémoire le dernier numéro utilisé pour pouvoir reprendre à partir du suivant. La date est utile comme élément de contrôle.
ART	37	Contient les articles en magasin, les données relatives au stock et au prix des articles. La zone UM (unité de mesure) n'est utile que si différents types de marchandise sont prévus. La zone stock est décrémentée des factures émises et incrémentée des factures reçues. Il est possible, pour augmenter la fonctionnalité de la procédure, d'ajouter une nouvelle zone contenant le taux de TVA.
CLI, FOU	94	Ces fichiers sont identiques. Ils contiennent les identifications (l'un pour les clients, l'autre pour les fournisseurs), le montant imposable (somme des montants de chaque facture), le total de TVA et le total des encaissements ou des paiements (mis à jour au moyen de la procédure encaissements/paiements).
CAISSE	55	Contient les totaux des montants imposables, de TVA, des factures et des mouvements encaissements/paiements. Ces données s'obtiennent aussi par addition des montants des différents enregistrements des fichiers CLI et FOU. Cette totalisation des valeurs dans un fichier séparé est, certes, redondante, mais elle permet d'avoir une vision immédiate de la situation économique. Ce fichier contient un seul enregistrement de données
TVA	32	C'est un récapitulatif, pour chaque facture émise, du montant de la TVA divisé par le taux de taxation. Son existence est liée à la nécessité de produire un document fiscal (registre de TVA) sur lequel ces données doivent apparaître. Une fois imprimées, les données du fichier sont annulées et le fichier est alors réutilisable. Cette impression est généralement trimestrielle : c'est pourquoi le nombre maximal d'enregistrements du fichier doit être égal au nombre de factures prévues pour la période de 3 mois.
TVC	32	Son format est identique à celui du fichier TVA, mais il contient les totaux cumulés des montants et de la TVA en fonction des taux de taxation. Il sert à connaître la situation d'ensemble en fin d'année.

née dans le menu, on obtient le chargement du menu secondaire correspondant. Intervient alors une nouvelle sélection parmi les rubriques affichées pour charger et faire exécuter la partie désirée de la procédure. Cette structure favorise une segmentation du programme. Chaque procédure correspond à un bloc autonome n'échangeant pas de données avec les autres ; elle est toujours rangée dans la même zone mémoire, en écrasant le module précédent.
Cette logique est très utilisée en Basic, alors qu'avec d'autres langages ou sur des machines plus puissantes, c'est le système d'exploitation qui gère la participation des programmes d'une façon parfaitement transparente à l'utilisateur. Il est néanmoins recommandé de diviser le programme en procédures bien définies et axées sur la résolution d'un seul aspect d'un problème. Il en découle deux avantages :

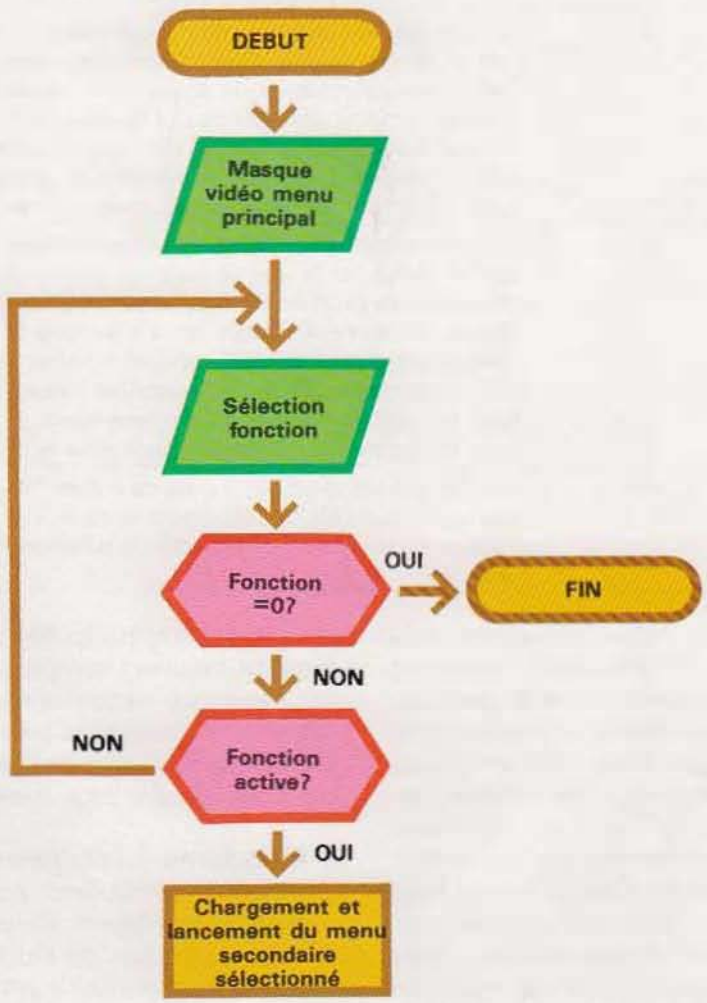
- **Modularité**. Le programme s'adapte à des situations particulières par suppression ou ajout de procédures. On peut, par exemple, retirer la gestion du stock (ou une de ses parties) en éliminant la procédure correspondante.

- **Facilité de manipulation et de mise au point**. Il est plus aisé de repérer une éventuelle erreur, de l'isoler et de la corriger. De nouvelles applications sont lancées par simple remplacement ou modification d'une procédure, sans interaction avec les autres.

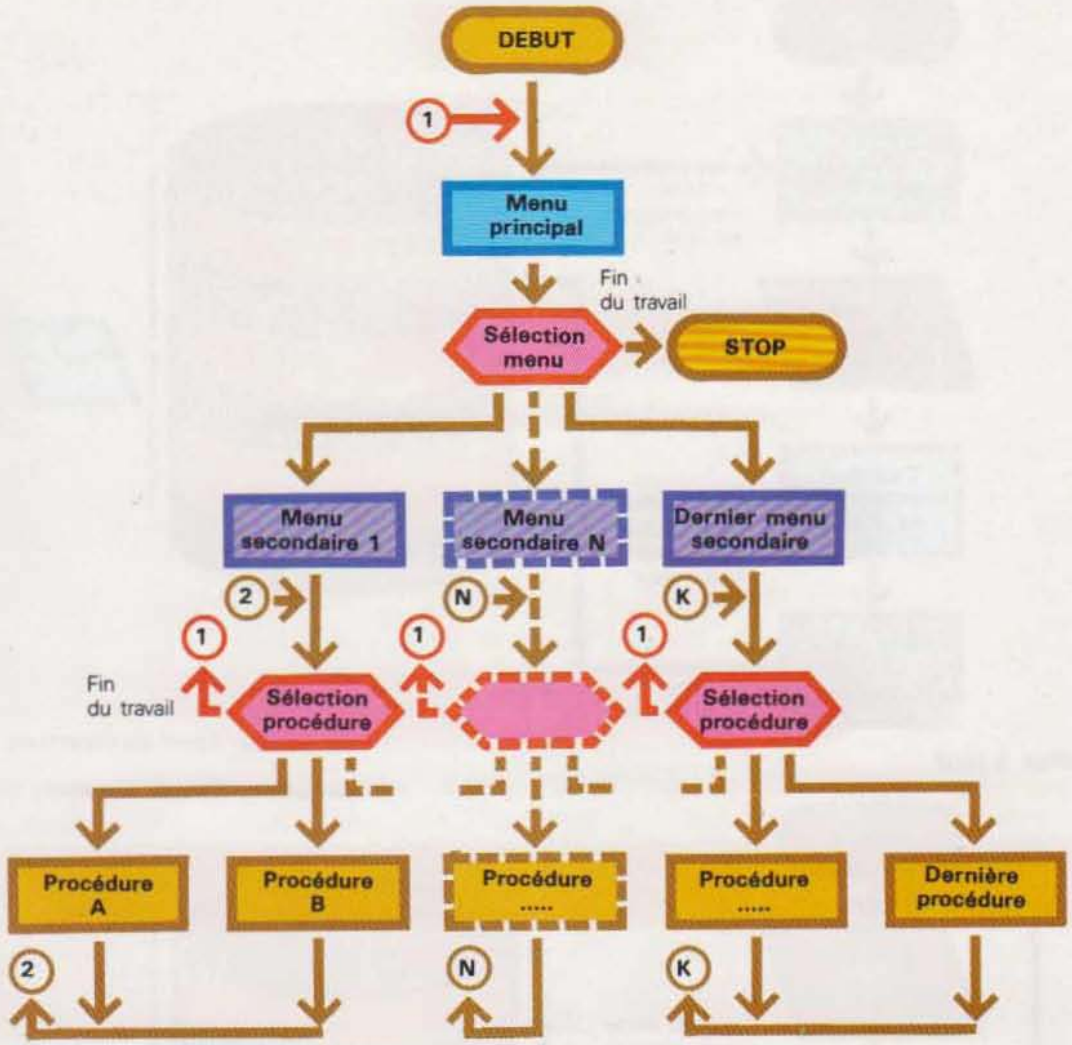
Dans des systèmes moins évolués, cette technique oblige à recopier tous les sous-programmes utilisés par plusieurs procédures (à moins d'adopter des systèmes de gestion compliquée et incombant totalement à l'utilisateur). Avec des systèmes d'exploitation plus avancés, en revanche, on n'a recours qu'à un module commun, son chargement et ses liens avec les autres parties de la procédure étant gérés par le système et ne demandant aucune intervention.

En pratique, la reproduction des sous-programmes se traduit seulement par une plus grande occupation de l'espace mémoire, en fait limitée à quelques kilo-octets (Ko). Néanmoins, dans certains cas, l'économie d'espace, même limitée, constitue une démarche avantageuse. Avec les disquettes de 5 pouces 1/4, par exemple, dont la capacité en simple densité est inférieure à 160 Ko, les programmes occupent jusqu'à 20 % de l'espace. Dans ce cas, il existe des techniques de segmentation fondées sur le transfert dans des zones de mémoire bien définies pour permettre l'utilisation de certaines parties selon plusieurs procédures.

ORGANIGRAMME DU MENU PRINCIPAL



SCHEMA DE L'ENCHAINEMENT
MENU PRINCIPAL / MENU SECONDAIRE / PROCEDURE



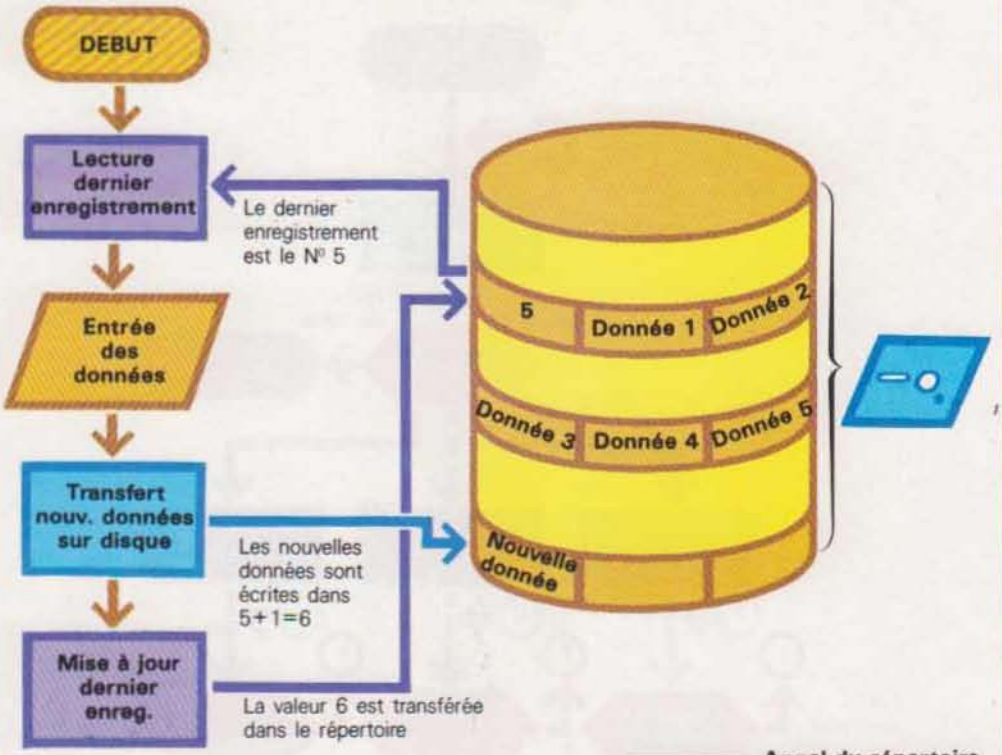
Gestion des fichiers

Quel que soit le problème que l'on doive résoudre dans un programme, il faut gérer des fichiers des données. Les fonctions requises sont, en général, l'introduction, la mise à jour et l'impression. Les méthodes utilisées sont illustrées page 1278, avec des organigrammes d'exemples de gestion de fichiers (clients et fournisseurs) page 1279, 1280 et 1281. Les seules modifications à apporter, pour d'autres fichiers, concernent le nom et le format des enregistrements.

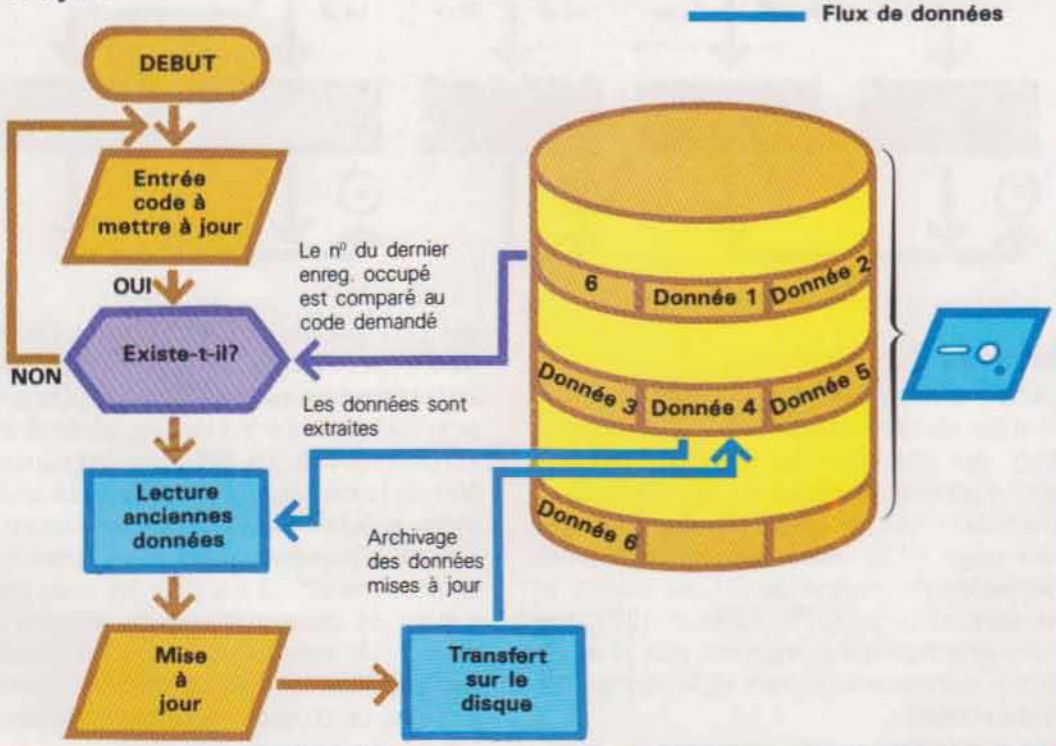
Les organigrammes sont utiles pour les langages peu spécialisés. Avec le Cobol, l'accès au fichier est si simple qu'aucune indication n'est nécessaire (voir exemple p. 1279). Dans la version Basic, on a introduit une généralisation à l'organigramme. En début de programme, un test de l'indicateur est effectué pour en déterminer le fichier (Clients ou Fournisseurs). Leur format d'enregistrement étant identique, le programme est valable pour les deux fichiers ; il suffit de changer le nom du fichier à sélectionner. Un autre paramétrage est possible en prenant le format d'enregistrement comme paramètre. Un programme unique suffit pour toutes les applications.

SCHEMAS D'ENTREE ET DE MISE A JOUR DES DONNEES

Entrée

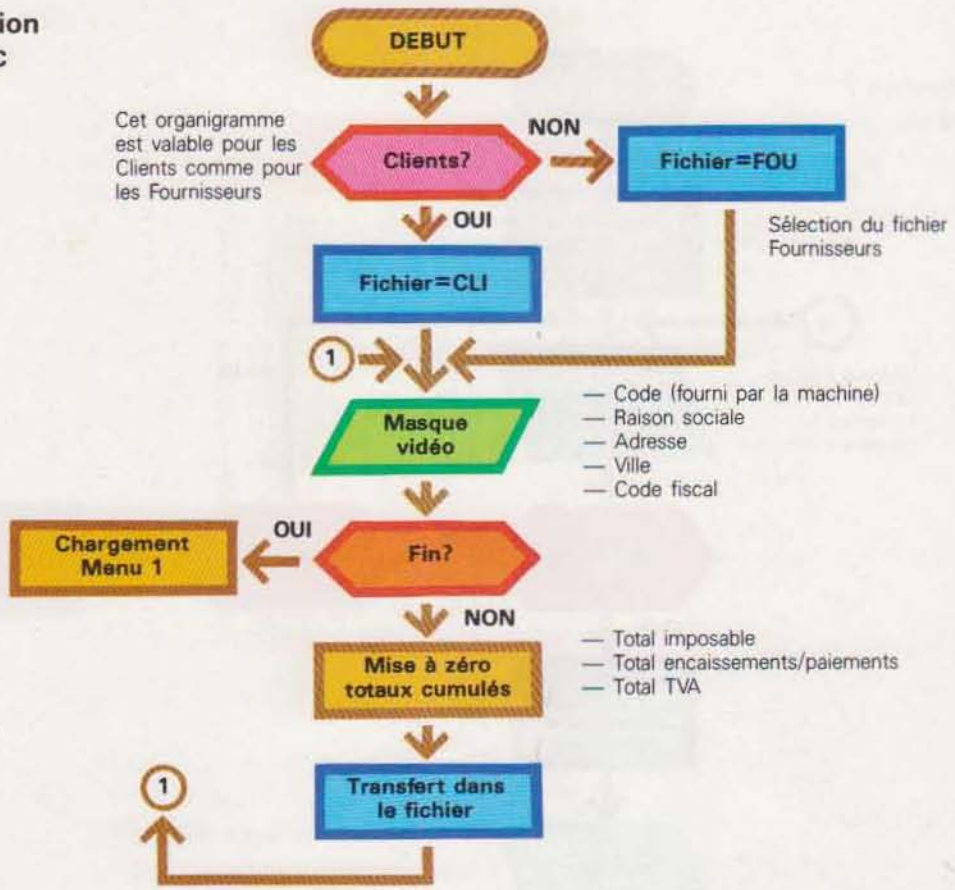


Mise à jour

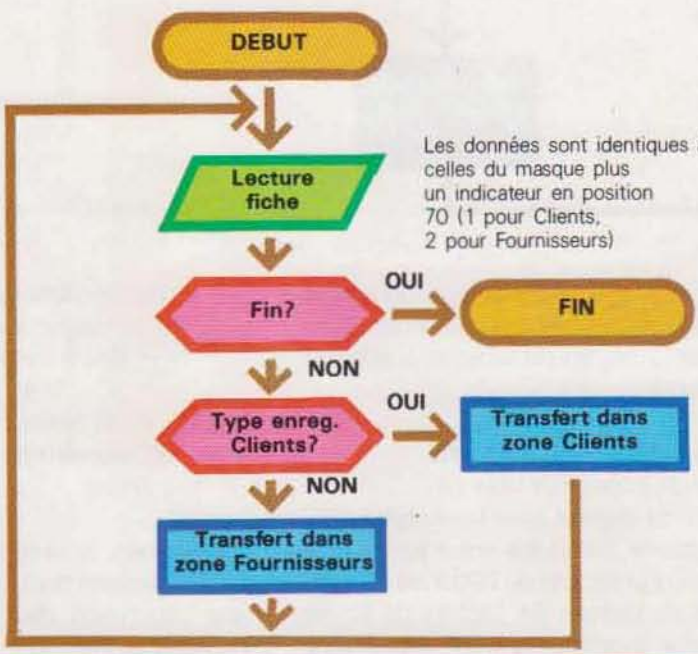


ENTREE DE NOUVEAUX CLIENTS OU FOURNISSEURS

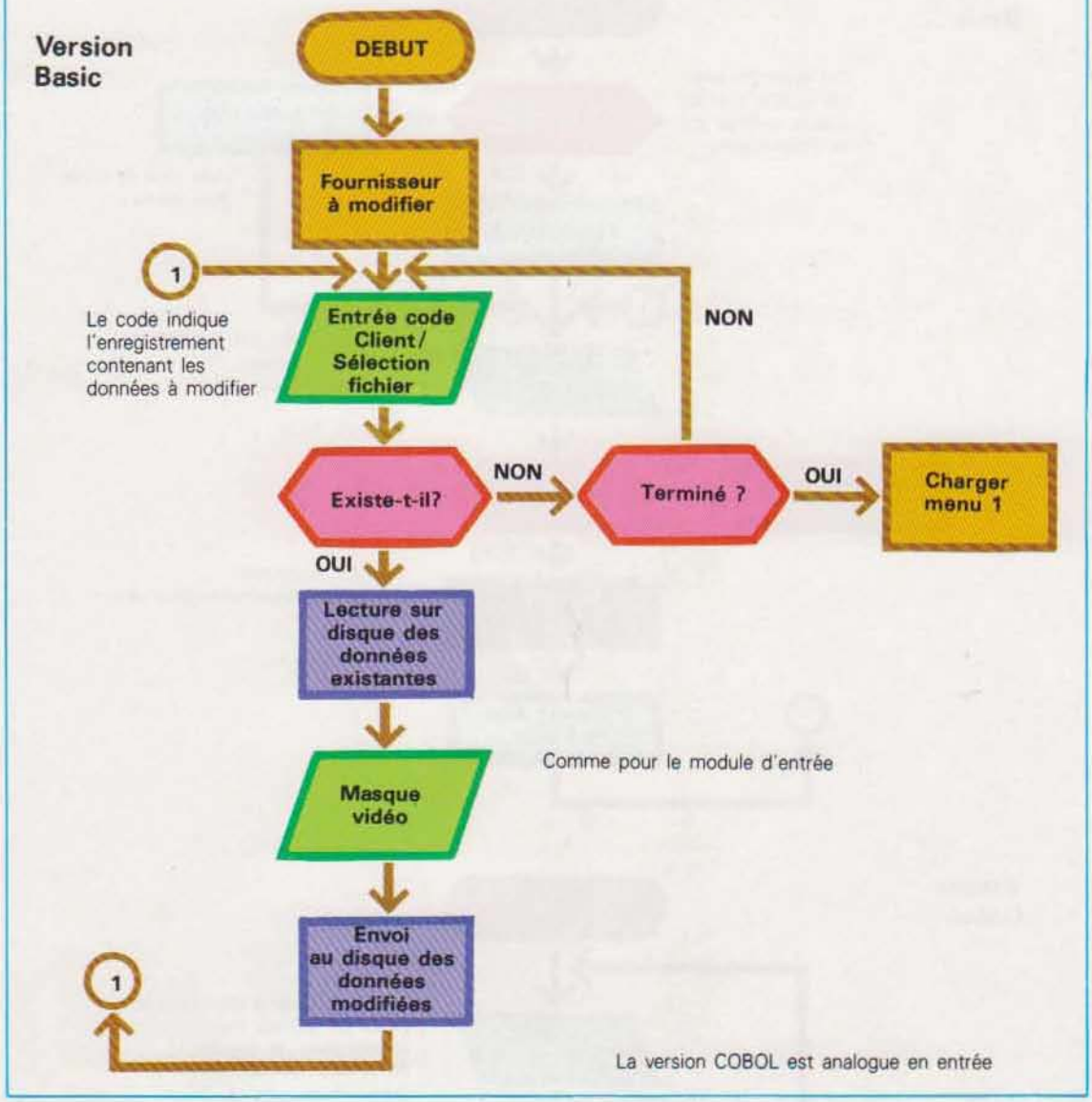
Version Basic



Version Cobol



MODIFICATION DES FICHIERS CLIENTS OU FOURNISSEURS

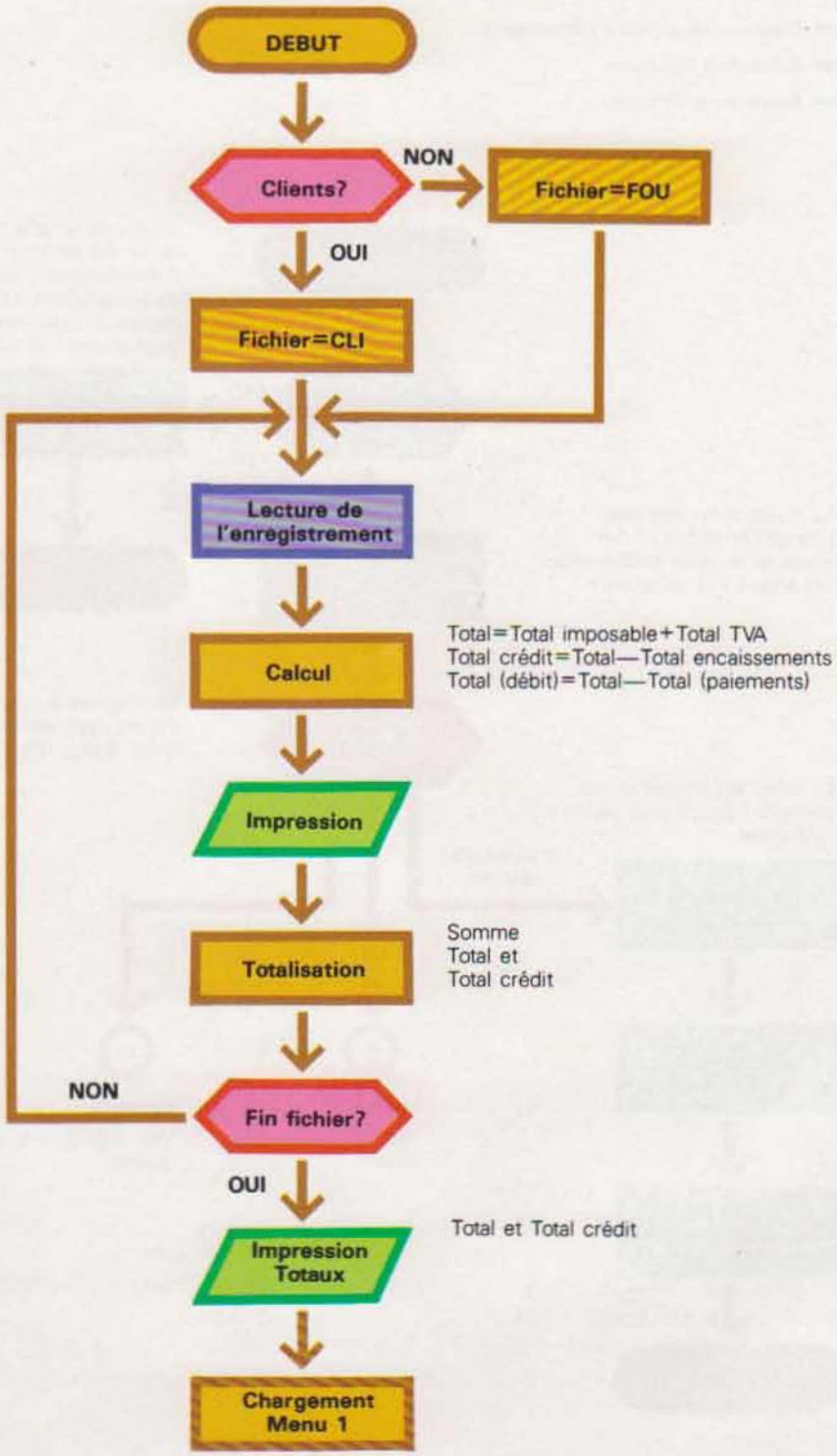


Dans l'organigramme d'un sous-programme général de lecture ou d'écriture d'un enregistrement (page 1282), les paramètres à affecter sont les suivants :

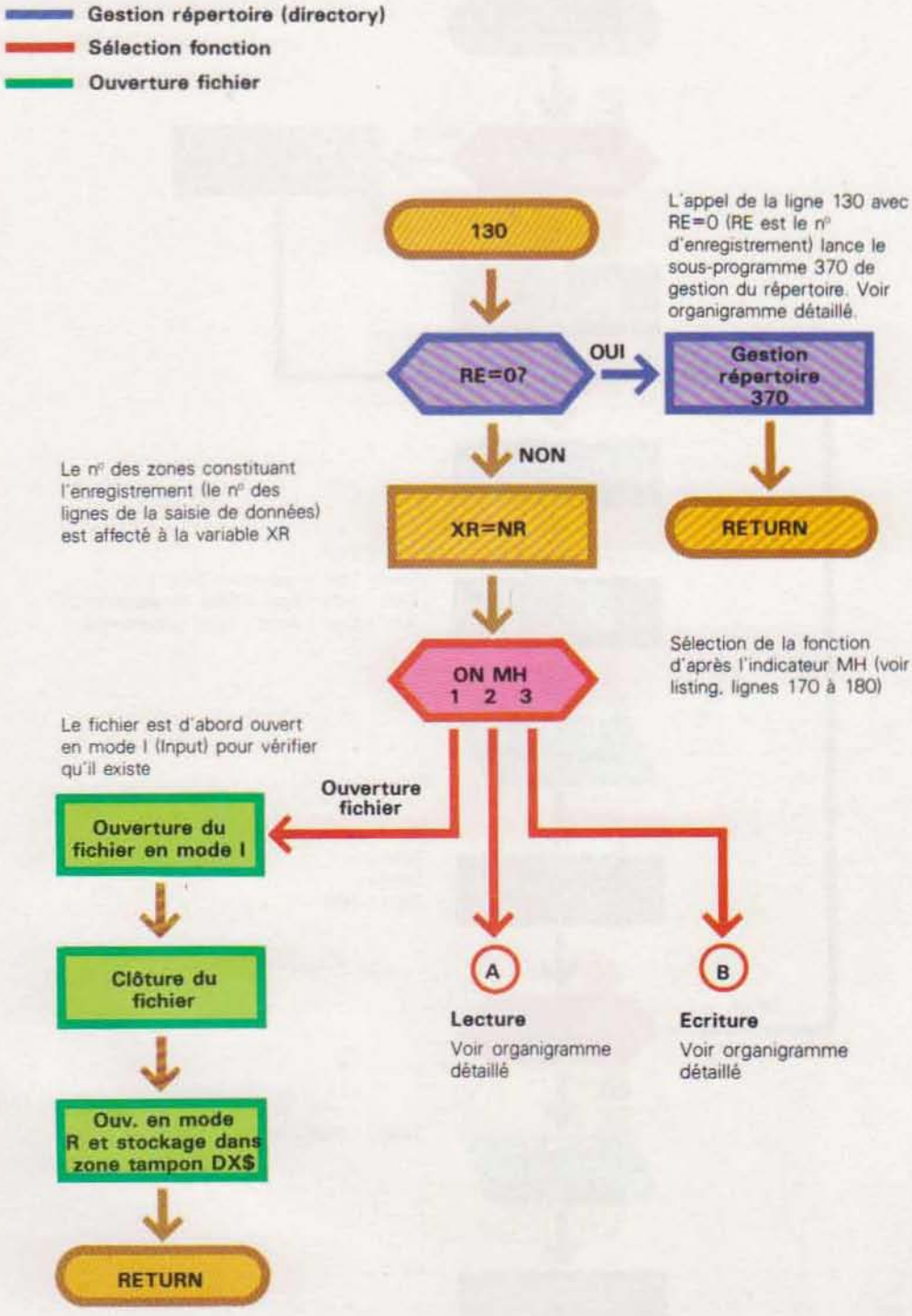
— Nom du fichier (variable NM$),
— Nombre de zones (variable NR),
— Longueur de chaque zone (en octets) définie comme la différence entre les valeurs du tableau PD (octets de début de chaque zone) et du tableau PA (octets de fin de zones). Par exemple, si pour la première zone des données, PD(1) = 1 et PA(1) = 30, la longueur en octets de la zone est :
N = PA(1) - PD(1) + 1
= 30 - 1 + 1
= 30 octets
— RE, numéro de l'enregistrement à lire ou à écrire.

En réponse, le programme stocke le contenu de l'enregistrement dans le tableau BF$, d'où une séparation des zones. Exemple, BF$(1) contient la première zone, BF$(2) contient la

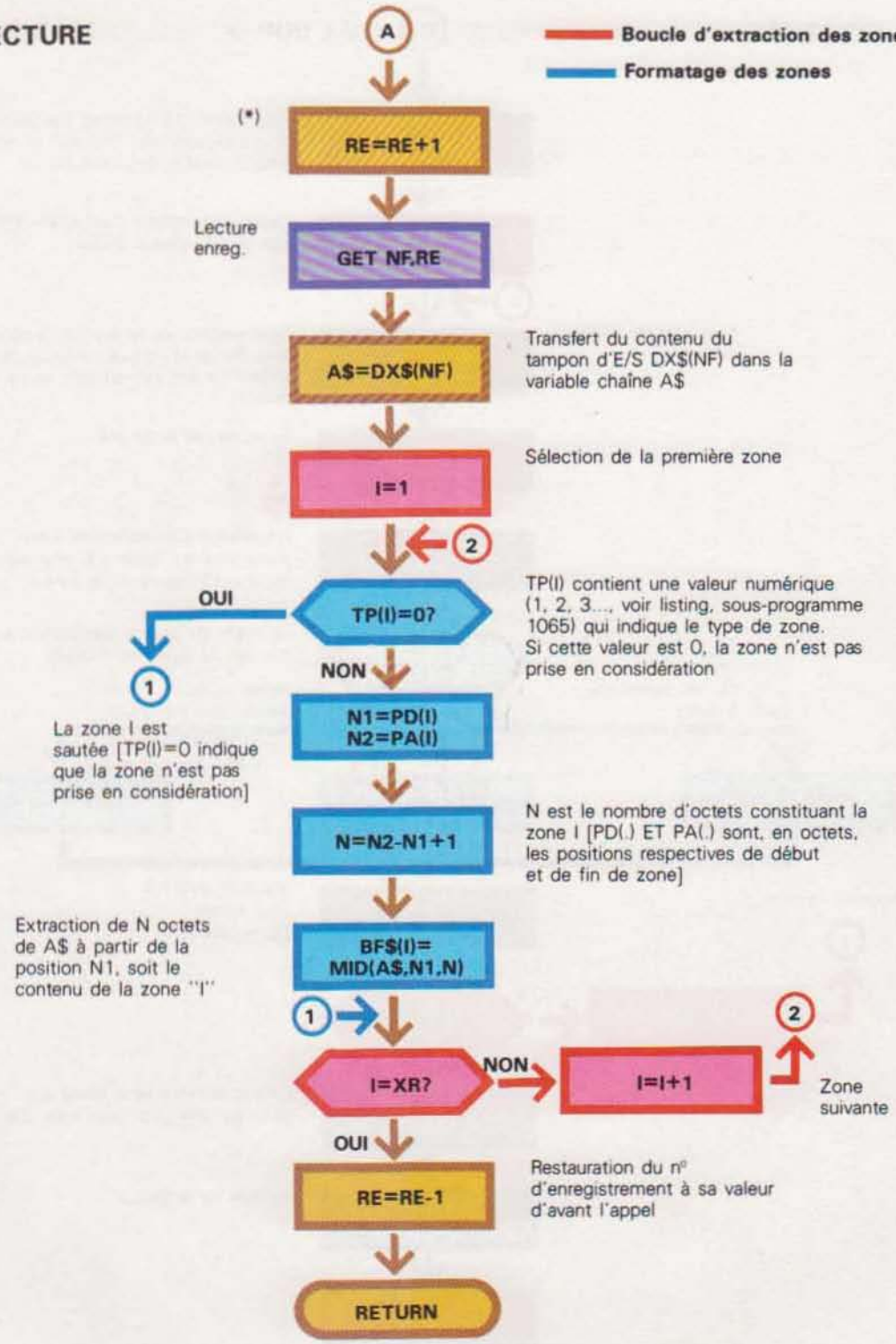
SITUATION CLIENTS/FOURNISSEURS



SOUS-PROGRAMME DE LECTURE/ECRITURE SUR DISQUE PARAMETRE...

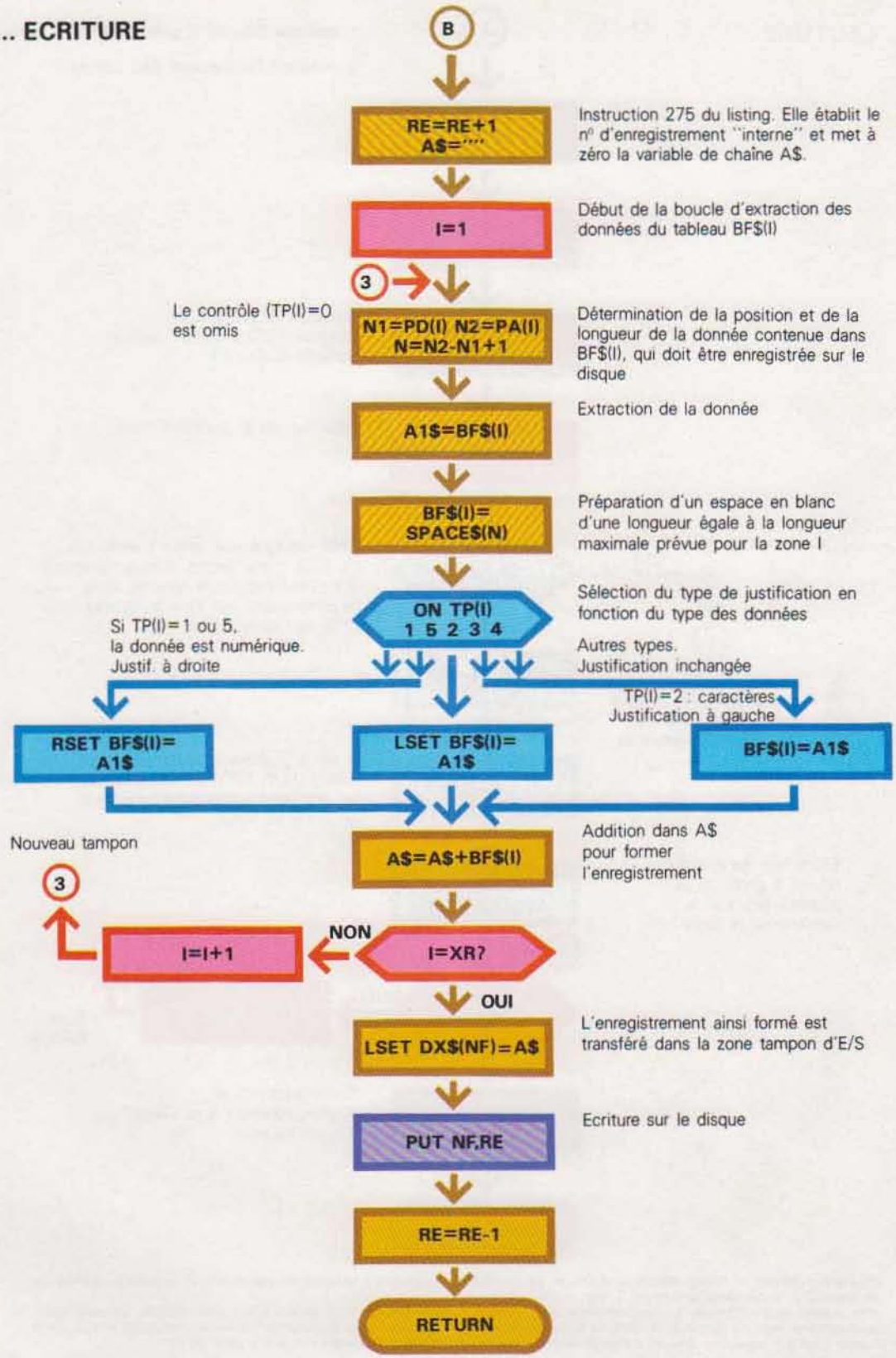


... LECTURE

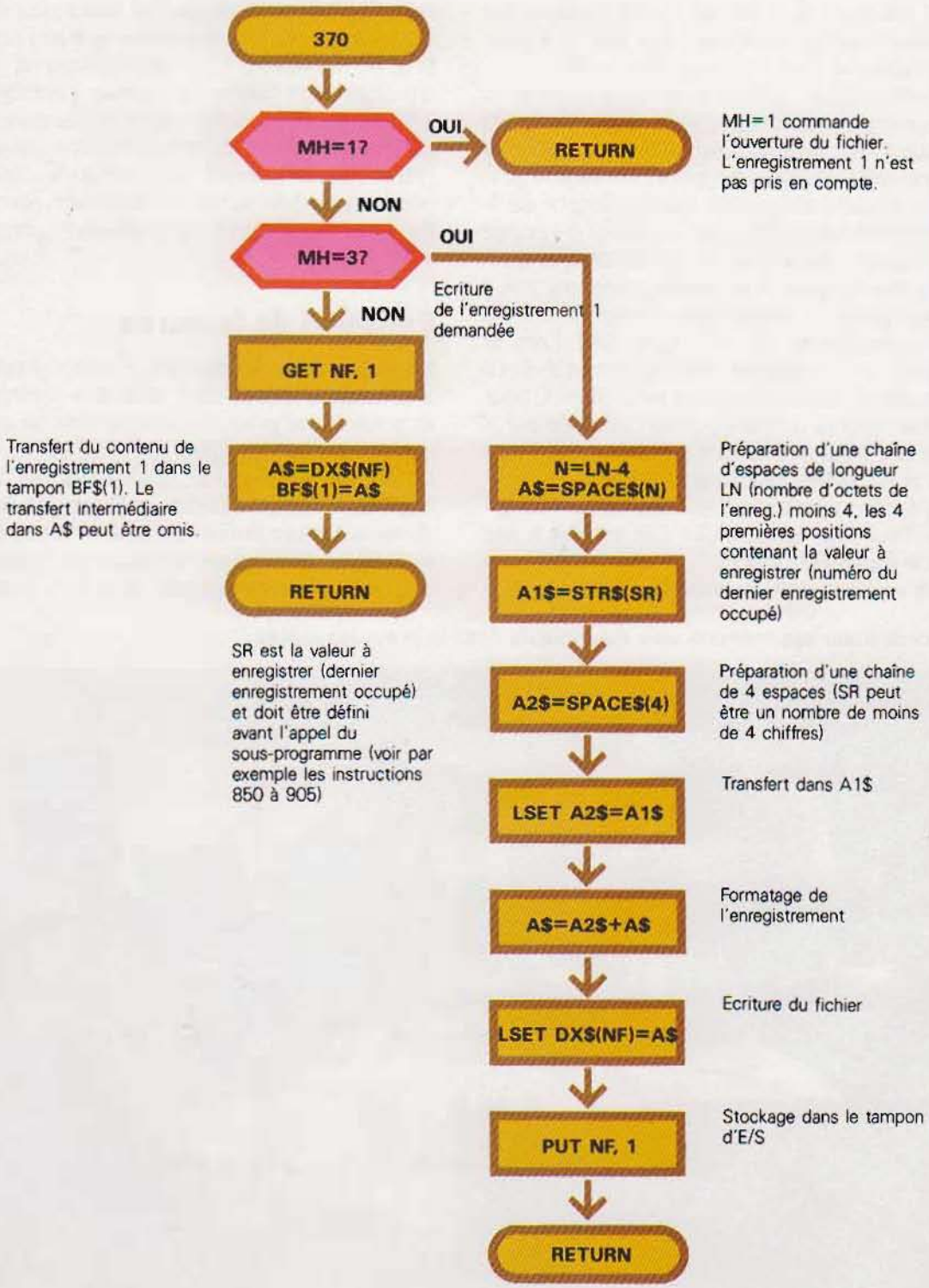


(*) L'enregistrement n° 1 étant affecté au répertoire, les données sont stockées à partir de l'enregistrement 2 : la première donnée est en position 2, la deuxième en position 3, etc.
Pour annuler cette différence, le sous-programme écrit les données en position +1 par rapport à leur ordre d'entrée. De cette façon, le programme appelant demande l'écriture de la première donnée en position 1 et le sous-programme le transfère en position 2. Cette astuce n'est pas nécessaire avec les systèmes d'exploitation qui numérotent les enregistrements à partir de zéro.

... ECRITURE


B
RE=RE+1
A$=""
Instruction 275 du listing. Elle établit le n° d'enregistrement "interne" et met à zéro la variable de chaîne A$.
I=1
Début de la boucle d'extraction des données du tableau BF$(I)
3
Le contrôle (TP(I)=0 est omis
N1=PD(I) N2=PA(I) N=N2-N1+1
Détermination de la position et de la longueur de la donnée contenue dans BF$(I), qui doit être enregistrée sur le disque
A1$=BF$(I)
Extraction de la donnée
BF$(I)= SPACE$(N)
Préparation d'un espace en blanc d'une longueur égale à la longueur maximale prévue pour la zone I
ON TP(I) 1 5 2 3 4
Sélection du type de justification en fonction du type des données
Si TP(I)=1 ou 5, la donnée est numérique. Justif. à droite
Autres types. Justification inchangée
TP(I)=2 : caractères Justification à gauche
RSET BF$(I)= A1$
LSET BF$(I)= A1$
BF$(I)=A1$
Nouveau tampon
3
A$=A$+BF$(I)
Addition dans A$ pour former l'enregistrement
I=I+1
NON
I=XR?
OUI
LSET DX$(NF)=A$
L'enregistrement ainsi formé est transféré dans la zone tampon d'E/S
PUT NF,RE
Ecriture sur le disque
RE=RE-1
RETURN

... GESTION DU REPERTOIRE (DIRECTORY)



deuxième, et ainsi de suite. Lors de l'écriture sur disque, le tableau BF$ doit avoir déjà reçu les données à transférer, les autres paramètres restant inchangés.
La sélection de la lecture ou de l'écriture est déterminée par l'indicateur MH (MH = 1 pour la lecture et MH = 2 pour l'écriture).
Pages 1287 à 1291, on a reproduit le listing du sous-programme, dans sa version Basic 80 sous CP/M (instructions de la ligne 130 à la ligne 465). Ce sous-programme prévoit la gestion séparée du premier enregistrement du fichier DIR (répertoire) dans lequel est mémorisé le numéro du dernier enregistrement occupé par des données. Pour activer cette fonction, il faut appeler le sous-programme en affectant 0 à la variable RE (RE = 0, ligne 185). Lors de l'écriture de nouvelles données, il faut d'abord appeler le sous-programme avec RE = 0 pour déterminer la dernière position occupée par le fichier ; les données sont ensuite transférées à la position (enregistrement) obtenue en ajoutant 1 (incrémentation) au contenu du répertoire. Pour finir, le répertoire DIR est mis à jour (voir lignes 800 à 905).
Les versions, pour un environnement autre que le CP/M, ne varient qu'au niveau des instructions de gestion du disque. Il suffit de remplacer ces instructions pour traduire le programme dans d'autres versions Basic. La seule limitation est liée au chaînage des valeurs numériques, ce qui entraîne une conversion des nombres en chaînes au cours de l'écriture et une transformation inverse (de chaînes à nombres) à la lecture. Ces fonctions ne sont pas prévues dans le listing ci-contre et doivent être activées respectivement avant et après le sous-programme. Les ajouts se réduisent donc à l'emploi d'instructions de conversion simples VAL, STR$...

Emission de factures

L'émission d'une facture est un acte régi par la législation fiscale. Elle doit donc être contrôlée et guidée avec précision (voir schéma de principe de factures p. 1292).
Le premier concerne la date. Le fichier DIR (répertoire) contient la date et le numéro de la dernière facture émise. Lorsque la procédure est activée, le programme fournit les numéros de factures ordonnés afin que l'on puisse

L'ordinateur apporte une aide inestimable dans la prévision météo.



Marie-Leo de Wys

PROCEDURE DE FACTURATION ET DE GESTION DES ARCHIVES

```
10 OPTION BASE 1
15 '    FILE = BASIC
20 '
25 '    VERSION 23-07-84 : CP/M - BASIC 80
30 DEFINT A-V
35 '    Les variables utilisées pour les calculs doivent commencer par Z
40 DEFDBL Z
45 DIM TF(20)                          ' touches de fonction
50 FOR I=1 TO 20 : READ TF(I) : NEXT I
55 DATA 10!,8!,12!,11!,16!,17!,13!,22!,18!,0!,19!,20!,21!,0!,0!,0!,0!,0!,0!,0!
60 DIM DX$(4)
65 DIM TP(20),PA(20),PD(20),LD(20)
70 DIM D$(20),B$(20),BF$(20),AT$(20),BR$(20),CN$(20)
75 DIM IS(5000)
80 '
85 '
90 '
95 '    *  FONCTIONS  *
100 '
105 BI$=CHR$(27)+"+"+CHR$(7)     ' Nettoyage de l'écran
110 PRINT BI$
115 '
120 GOSUB 470                    ' Menu des fonctions prévues
125 STOP
130 '
135 '   ** GESTION DONNEES **
140 '   Fichier : GDAT controlé le 23-07-84
145 '   Enregistrement = RE type enregistrement = TR BF$(20)
150 '   Tampon de 1 à 4 DX$(4) = 4 fichiers
155 '   NF = numéro de fichier = numéro de tampon
160 '   NR = numéro de ligne
165 '   NH =
170 '         1 ouverture
175 '         2 lecture
180 '         3 écriture
185 IF RE=0 THEN GOSUB 370:RETURN  'RE=0 pour gérer le répertoire du fichier
190 XR=NR                          'nombre de zones (1310)
195 ON MH GOTO 200,220,275
200 '   ouverture
205 OPEN "I",NF,NM$  'ouverture en mode I pour vérifier l'existence du fichier
210 CLOSE NF:OPEN "r",NF,NM$,LN:FIELD NF,LN,AS,DX$(NF):RETURN
215 'la ligne 210 ouvre en mode R et stocke les données dans la zone tampon
220 '   lecture
225 RE=RE+1            ' voir l'organigramme
230 GET NF,RE:A$=DX$(NF)   'lecture et transfert des données dans A$
235 '  * création des tampons BF$(.) en fonction des longueurs des
240 '    zones définies dans le sous-programme 1310
245 FOR I=1 TO XR:IF TP(I)=0 GOTO 255
250 N1=PD(I):N=N2-N1+1:BF$(I)=MID$(A$,N1,N)
255 NEXT I
260 RE=RE-1             ' voir l'organigramme
265 RETURN
270 '
275 '    écriture
280 RE=RE+1
285 A$=""
290 FOR I=1 TO XR:IF TP(I)=0 GOTO 345  'sélection d'une des zones
295 N1=PD(I):N2=PA(I):N=N2-N1+1        'calcul de la longueur en octets
300 A1$=BF$(I)                         'sauvegarde du contenu
305 BF$(I)=SPACE$(N)                   'préparation de l'espace nécessaire
310 '                                   d'après la longueur de la zone
315 ON TP(I) GOTO 320,325,330,330,320  'sélection d'après le type
320 RSET BF$(I)=A1$:GOTO 335           'numérique à droite!
325 LSET BF$(I)=A1$:GOTO 335           'caractères à gauche
```

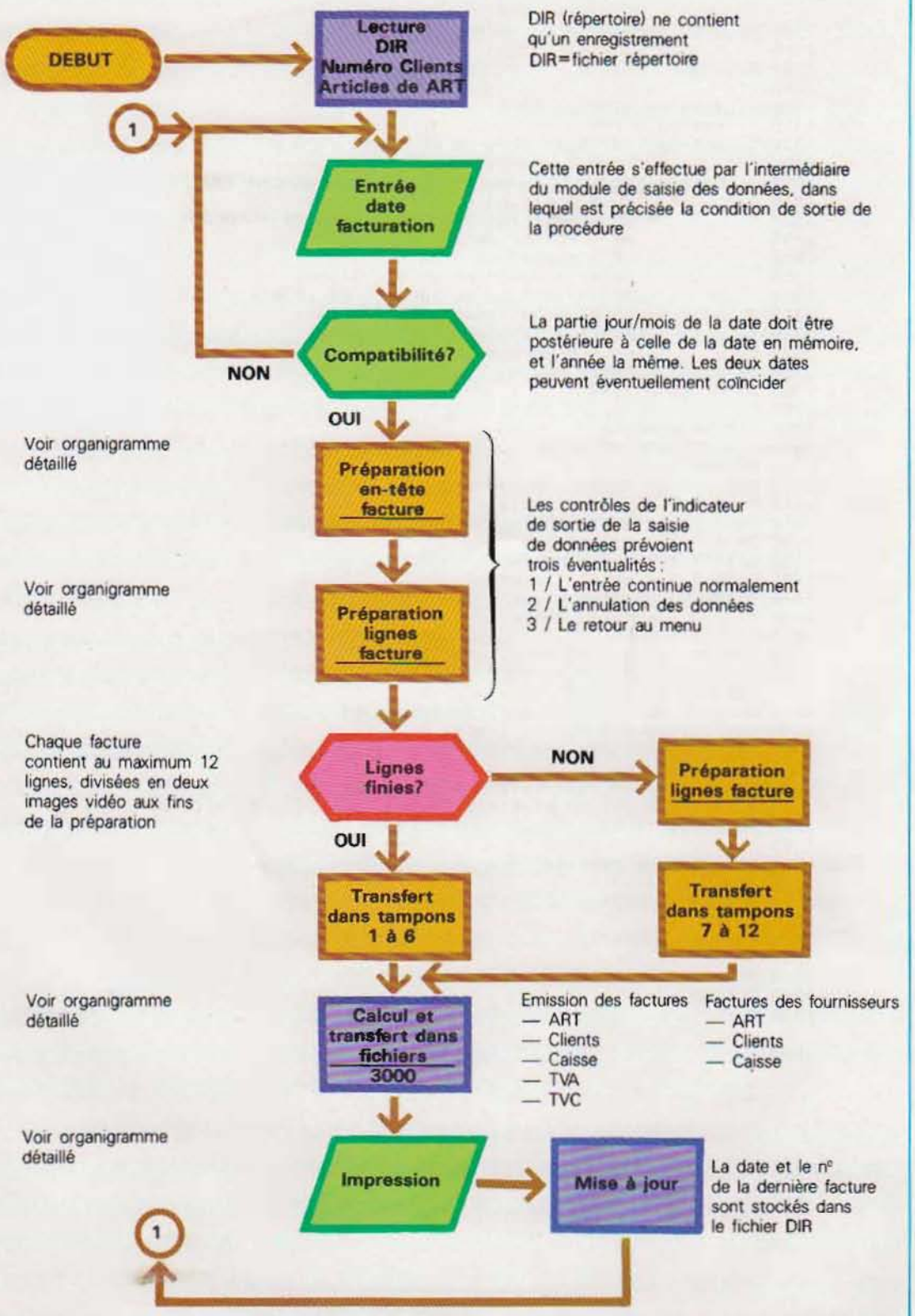
```
330 BF$(I)=A1$                          'autres types inchangés
335 A$=A$+BF$(I)                        'additionne les tampons pour former
340 '                                    une seule chaine
345 NEXT I
350 LSET DX$(NF)=A$                     'transfert dans le tampon d'E/S
355 PUT NF,RE                           'écriture de l'enreg RE dans le fic NF
360 RE=RE-1
365 RETURN
370 'gestion enregistrement 1
375 '
380 IF MH=1 THEN RETURN                 'MH=1 pour l'écriture du fichier
385 IF MH=3 GOTO 410                    'branchement sur l'écriture de l'enreg 1
390 'lecture
395 GET NF,1                            'lecture de l'enreg 1
400 A$=DX$(NF):BF$(I)=A$                'transfert dans la zone tampon 1
405 RETURN
410 'écriture
415 N=NL-4:A$=SPACE$(n)                 'préparation d'une chaine vide de
420 '                                    longueur LN-4
425 A1$=STR$(SR)                        'conversion du nombre SR
430 '                                    en caractères
435 A2$=SPACE$(4):LSET A2$=A1$          'transfert de cette variable dans
440 '                                    une zone de longueur 4
445 A$=A2$+A$                           'concaténation de la chaine vide (415)
450 '                                    pour avoir un enreg de longueur LN
455 LSET DX$(NF)=A$                     'transfert dans la zone tampon d'E/S
460 PUT NF,1                            'écriture fichier NF enreg = 1
465 RETURN
470 '
475 '  MENU1
480 '
485 '  VERSION 27-07-84
490 '  les touches de fonction sont créées par le menu principal - fichier MENU
495 GOSUB 575
500 ON V GOTO 505,510,525,525,525,525,525,525,525,525
505 GOSUB 795:GOTO 485
510 GOSUB 910:GOTO 485
515 '  les rubriques actives de MENU sont la 1 et la 2
520 '  les autres sont ignorées
525 GOTO 485
530 PRINT BI$
535 CK=5:X=30:Y=10:GOSUB 755
540 PRINT CHR$(27)+"G1"
545 PRINT CHR$(27)+"G0"
550 RETURN
555 END
560 '-----------------------------
565 ' * sélection de la fonction *
570 '-----------------------------
575 PRINT BI$
580 CK=5:X=2:Y=1:GOSUB 755:PRINT "E.G.S Gestion archives clients"
585 CK=5:X=15:Y=3:GOSUB 755             'le curseur est positionné en X,Y
590 PRINT "1- ENTREE"
595 Y=Y+1:GOSUB 755
600 PRINT "2- VARIATION"
605 Y=Y+1:GOSUB 755
610 PRINT "3- ............"
615 Y=Y+1:GOSUB 755
620 PRINT "4- ............"
625 Y=Y+1:GOSUB 755
630 PRINT "5- ............"
635 Y=Y+1:GOSUB 755
640 PRINT "6- ............"
645 Y=Y+1:GOSUB 755
```

```
650 PRINT "7- ............."
655 Y=Y+1:GOSUB 755
660 PRINT "8- ............."
665 Y=Y+1:GOSUB 755
670 PRINT "9- ............."
675 Y=Y+1:GOSUB 755
680 PRINT "0- FIN TRAVAIL ."
685 X=1:Y=Y+2:GOSUB 755
690 PRINT "entrez l'option choisie"
695 X=26:GOSUB 755:A$=INPUT$(1):PRINT A$
700 V=VAL(A$)
705 IF V>9 GOTO 730
710 PRINT BI$
715 IF V=0 THEN V=10
720 RETURN
725 '  ** erreur **
730 PRINT CHR$(27),CHR$(7)
735 X=42:GOSUB 755
740 PRINT "non valide"
745 GOTO 695
750 '
755 '  déplacement curseur
760 '  CURS-X+Y
765 '  l'instruction suivante dépend du type de masque utilisé
770 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
775 RETURN
780 ' ** FIN DU MENU **
785 ' *
790 ' *
795 ' *** ENTREE ***
800 ' ouverture
805 RE=1
810 NF=1:GOSUB 1305              'préparation du format
815 MH=1:GOSUB 130               'ouverture du fichier
820 'lecture du dernier enregistrement occupé
825 RE=0:MH=2
830 GOSUB 130
835 RE=VAL(BF$(1))               'RE=1er enreg libre + 1
840 X=1:Y=1:GOSUB 1280           'positionnement du curseur
845 PRINT "code client : ";RE    'affichage du n° de code
850 '                            (enreg) du prochain client
855 GOSUB 1305                   'préparation du format
860 GOSUB 1065                   'présentation à l'écran
865 GOSUB 1250                   'saisie des données
870 ON F1 GOTO 875,880,840
875 RETURN                       'retour : les données sont annulées
880 MH=3:GOSUB 130               'entrée nouveau client
885 SR=RE
890 RE=0:GOSUB 130               'mise à jour du répertoire (enreg 1)
895 GOTO 820                     'nouvelle entrée
900 CLOSE NF
905 RETURN
910 ' variations
915 RE=1
920 NF=1:GOSUB 1305              'préparation du format
925 MH=1:GOSUB 130               'ouverture du fichier
930 RE=0:MH=2:GOSUB 130          'lecture du répertoire
935 MX=VAL(BF$(1))               'MX=dernier code client utilisé
940 X=1:Y=1:GOSUB 1280
945 PRINT "code client : "
950 KC=20:MD=0:KR=1:TP(1)=1      '*** préparation du format pour entrée
955                              du code client à modifier
960 NF=1:PD(1)=1:PA(1)=3:LN=3:LD(1)=3
965 GOSUB 1250                   'saisie des données pour la lecture du code
```

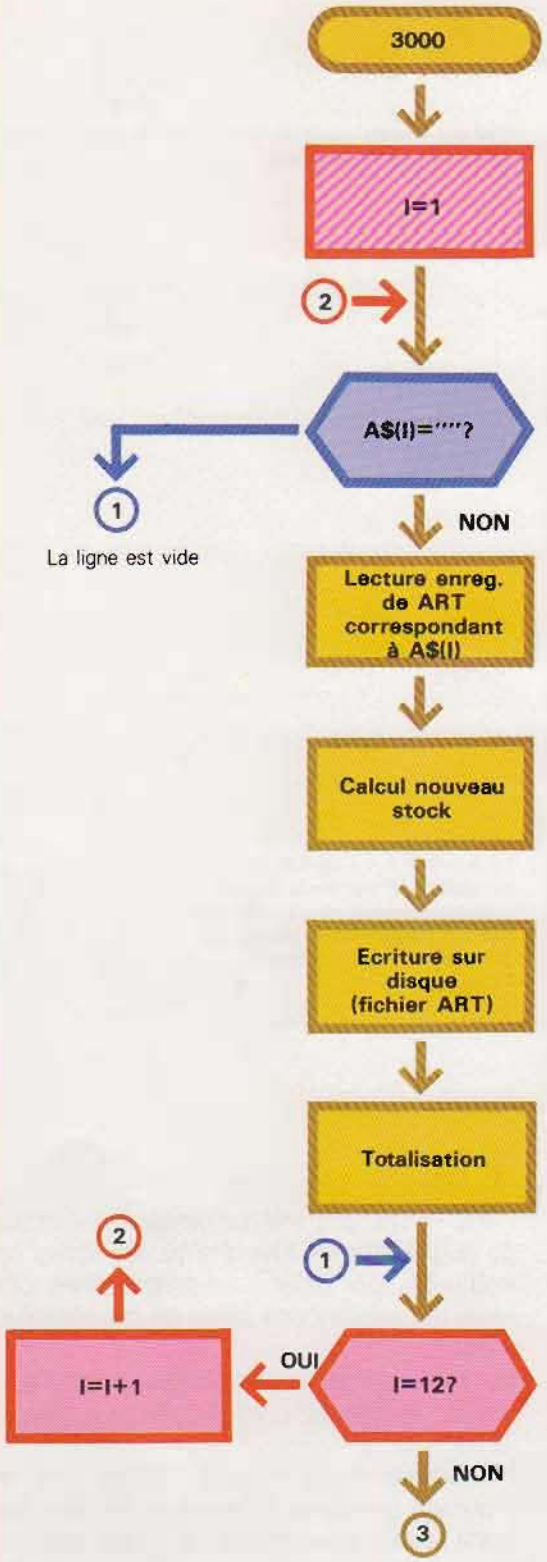
```
970 ON F1 GOTO 975,980,940
975 RETURN                          'retour menu
980 V=VAL(BF$(1))                   'entrée
985 IF V>MX THEN PRINT "ERREUR":GOTO 940 'controle qu'il n'a pas été demandé
990 '                               un code plus grand que le dernier mis en
995 '                               mémoire
1000 NF=1:GOSUB 1305
1005 RE=V
1010 MH=2:GOSUB 130
1015 FOR I=1 TO NR:TP(I)=4:NEXT I 'transforme toutes les zones affichées
1020 GOSUB 1065                     'affichage à l'écran
2025 GOSUB 1250                     'saisie des modifications
1030 ON F1 GOTO 1035,1040,940
1035 RETURN
1040 MH=3:GOSUB 130
1045 GOTO 940
1050 '
1055 RETURN
1060 '
1065 '
1070 ' **** FORMATS VIDEO ****
1075 '
1080 ' version 24-07-84
1085 '
1090 ' TYPE =
1095 '          1-numérique
1100 '          2-alphanumérique
1105 '          3-affichage seulement (entrée validée)
1110 '          4-affichage avec entrée
1115 '          5-zone résultat
1120 ' ** entrée NF = numéro du fichier **
1125 MD=0
1130 '  calcul de la longueur maximale des descriptions
1135 FOR I=1 TO NR
1140 IF LEN(D$(I))>MD THEN MD=LEN(D$(I))
1145 NEXT I
1150 '  préparation des zones tampons
1155 FOR I=1 TO NR:N=MD-LEN(D$(I))
1160 IF N=0 THEN 1170
1165 D$(I)=D$(I)+SPACE$(N)
1170 NEXT I
1175 FOR I=1 TO NR:IF TP(I)=0 THEN 1200
1180 IF TP(I)=3 OR TP(I)=4 THEN D$(I)=D$(I)+" "+BF$(I):GOTO 1200
1185 N=PA(I)-PD(I)+1
1190 D$(I)=D$(I)+" "+STRING$(N,46)
1195 BF$(I)=SPACE$(N)
1200 NEXT I
1205 '
1210 '  affichage à l'écran
1215 X=KC:Y=KR
1220 FOR I=1 TO NR
1225 GOSUB 1280
1230 PRINT D$(I)
1235 Y=Y+1
1240 NEXT I
1245 RETURN
1250 '
1255 '  ***** saisie des données *****
1260 '  -------------------------------
1265 '  * idem à celle déjà présentée *
1270 '  -------------------------------
1275 '
1280 '  déplacement du curseur
1285 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
```

```
1290 RETURN
1295 '
1300 RETURN
1305 '
1310 '  data version 23-07-84
1315 '  *** DATA ***
1320 '  entrée : NF=type d'enreg et n° du fichier
1325 '
1330 '  sortie :
1335 '           KC = position première colonne
1340 '           KR = position première ligne de l'écran
1345 '           NR = nombre de lignes
1350 '           NF = numéro du fichier
1355 '           D$(*) = description des lignes
1360 '           TP$(*) = type de zones - cf 1070 -
1365 '           PD(*) = pointeur début de zone
1370 '           PA(*) = pointeur fin de zone
1375 '           LN = nombre de caractères
1380 '           NM$ = nom du fichier
1385 ON NF GOTO 1400
1390 '
1395 'TR=2: données
1400 KC=2:KR=3:NR=8:N=NR
1405 RESTORE 1415
1410 FOR I=1 TO N:READ D$(I):NEXT I
1415 DATA " 1 - raison sociale"
1420 DATA " 2 - adresse"
1425 DATA " 3 - ville"
1430 DATA " 4 - registre TVA"
1435 DATA " 5 - total factures"
1440 DATA " 6 - total payes"
1445 DATA " 7 - total solde"
1450 DATA " 8 - ..............."
1455 FOR I=1 TO N:READ TP(I):NEXT I
1460 DATA 2!,2!,2!,1!,5!,5!,5!,2!
1465 FOR I=1 TO N:READ PD(I):NEXT I
1470 DATA 1!,31!,61!,76!,87!,98!,109!,120!
1475 FOR I=1 TO N:READ PA(I):NEXT I
1480 DATA 30!,60!,75!,86!,97!,108!,119!,128!
1485 LN=PA(N):NM$="A:FORN"
1490 FOR I=1 TO N:LD(I)=PA(I)-PD(I)+1:NEXT I
1495 N=NR+1:FOR I=1 TO N:D$(I)=" ":TP(I)=0:PD(I)=0:PA(I)=0:NEXT I
1500 RETURN
1505 '
1510 '  *** zone résultat ***
1515 Z5=VAL(BF$(5))
1520 Z6=VAL(BF$(6))
1525 Z7=Z5-Z6
1530 BF$(7)=SPACE$(11)
1535 A2$=STR$(Z7)
1540 RSET BF$(7)=A2$
1545 RETURN
1550 '
```

EMISSION DES FACTURES



SOUS-PROGRAMME DE CALCUL ET DE TRANSFERT DANS LES FICHIERS

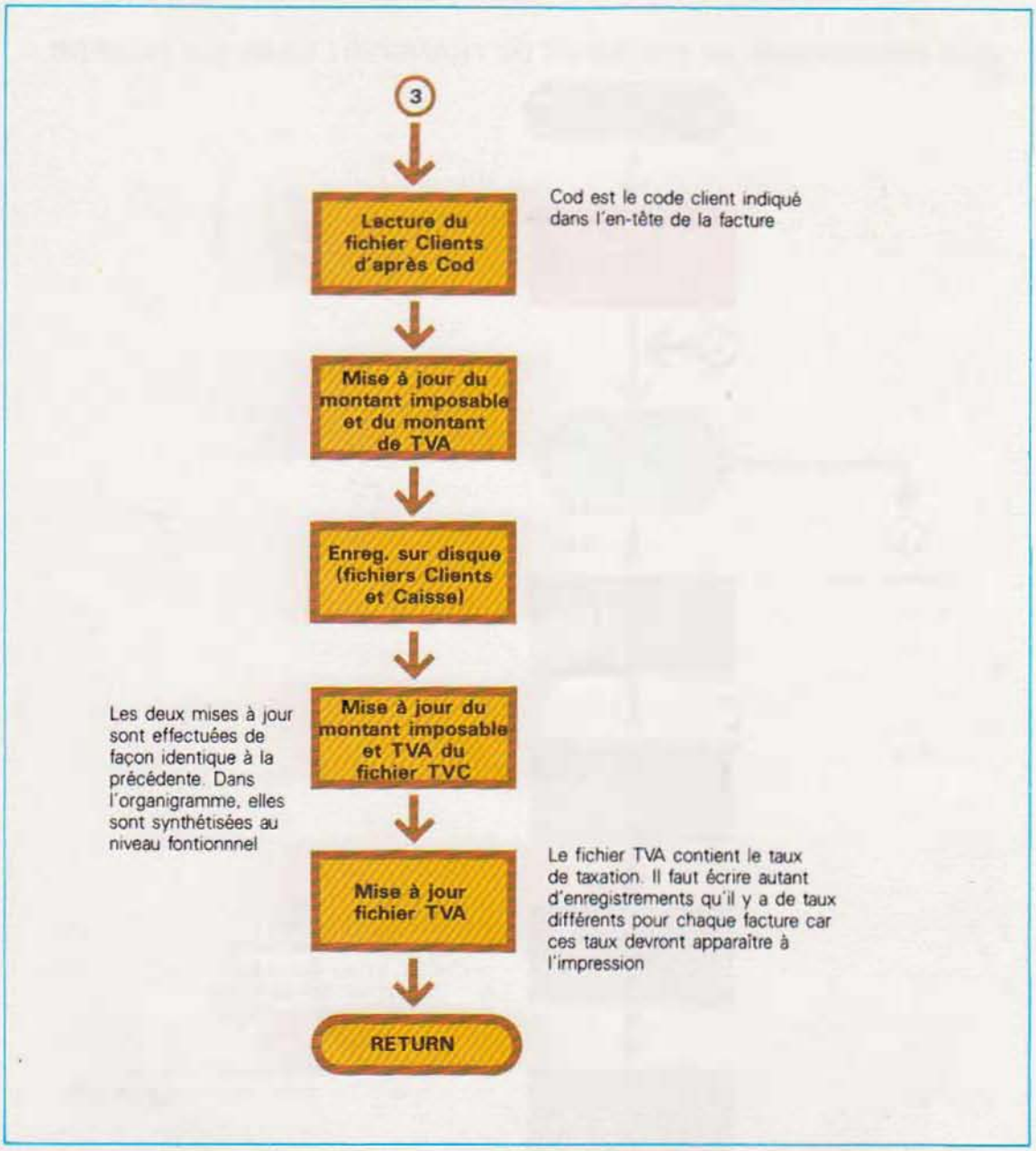


L'enregistrement de ART a déjà été lu lors de l'entrée des lignes (voir organigramme correspondant)

La mise à jour du stock n'est effectuée qu'une fois la facture établie afin de pouvoir modifier ou annuler chaque ligne sans devoir prévoir des procédures de correction du stock complexes.

Totalisation des montants imposables et des montants de la TVA divisés par les taux de taxation

Le traitement des lignes d'une facture est terminé



contrôler la compabilité des dates : la date entrée à la console doit être postérieure ou égale à la dernière date enregistrée.
Ce type de contrôle est également prévu en versions Cobol, où les dates sont introduites par des fiches, ne serait-ce que pour vérifier formellement la perforation.
On suppose que la facture comporte deux parties : l'en-tête contenant les données d'identification du client et la série de 12 lignes destinées à recevoir les mouvements des stocks et de caisse.

Dans l'organigramme présenté, les fonctions de préparation de l'en-tête et des lignes sont exécutées par deux sous-programmes différents qui utilisent une saisie de données paramétrée.
La procédure de préparation de l'en-tête à suivre est décrite par l'organigramme de la page 1293.
Le nombre de lignes de la facture rend leur affichage simultané à l'écran impossible. Pour cette raison, une méthode de présentation par bloc de 6 lignes a été prévue (voir page 1292).

Les jeux vidéo (suite) ou les aventures de Fred

Nous avons vu, dans le numéro précédent, comment nous lier d'amitié avec Fred, notre si sympathique joueur de Gô électronique. Nous allons donc continuer à examiner sa logique. Analysons en détail le comportement de CARTE. Ce module se rappelle lui-même autant de fois que nécessaire, ce qui lui permet d'avancer pas à pas à l'intérieur du groupe de connexion et en en déterminant ainsi la ***structure*** *et les* ***limites****. Il est fondamental que CARTE réussise à reconnaître aussi les côtés extérieurs du Gô Ban pour deux raisons :*

— pour que Fred n'aille pas déposer des pions hors du Gô Ban à la suite d'une mauvaise signalisation,

— afin que Fred puisse bien gérer les "pions fantômes" qui revêtent une importance stratégique fondamentale dans les coups d'angle. Muni de tous ces renseignements, Fred peut formuler une ***réponse au coup*** *de son partenaire humain. Notons que, du moins au début, nous considérons Fred comme le plus faible des deux joueurs. A lui donc l'honneur — et l'avantage — de l'ouverture de la partie. La procédure par laquelle Fred accomplit des devoirs de joueur s'appelle JEU ; elle est composée de plusieurs routines. La première action de JEU consiste en le dépôt des pions de* ***handicap,*** *puisque Fred dispose de l'avantage. Le nombre des pions de handicap varie entre 1 et 9, en fonction du rapport de force existant entre les joueurs. Dans notre cas, nous concéderons — royalement n'est-ce-pas ? — 5 à 7 pions à Fred (au-delà, ce serait vraiment couper les verges pour se faire battre). Ceux-ci représentent plus de 10 % des pions mis à sa disposition. (Rappelons-nous que nous jouons avec un nombre réduit de pions.) Fred pourra ainsi acquérir une position stratégique tout à fait intéressante, surtout s'il choisit d'effectuer une ouverture d'angle.*

La structure du module JEU a été indiquée dans le numéro précédent, en page 1272. Il contient un appel aux sous-programmes EFFET COUP BLANC, EFFET COUP NOIR et MODULES, présentés en page 1295 et 1296.

A ce point du jeu, Fred peut décider d'exécuter une ***opération d'importance,*** *comme de* ***capturer un groupe de pions.*** *C'est aussi la*

DESCRIPTION DU MODULE EFFET COUP BLANC

```
EFFET COUP BLANC
  DEBUT
    POUR chaque intersection x avec un pion
    noir, EXECUTER
      DEBUT
        APPELER CARTE (noir, x)
        SI le groupe n'a pas de liberté
        ALORS prendre ses pions
        SINON
          DEBUT
            SI le groupe a une seule liberté
            ALORS choisir une liberté (qui ne
            tombe pas sur le bord de
            l'échiquier)
            SI le groupe a une ou deux libertés
            ALORS APPELER EVALUATION
            pour analyser la liberté choisie
          FIN
      FIN
  FIN
```

Ce modèle analyse l'effet du coup du Blanc avec ses conséquences sur le scénario du Gô Ban. Il met en évidence l'interrelation fonctionnelle entre les différents modules et les critères la déterminant.

DESCRIPTION DU MODULE EFFET COUP NOIR

```
EFFET COUP NOIR
  DEBUT
    POUR chaque intersection x avec un pion
    blanc, EXECUTER
      DEBUT
        APPELER CARTE (blanc, x)
        SI le groupe a une seule liberté
        ALORS
          DEBUT
            Poser le pion noir sur ce point
            Prendre le pion blanc
          FIN
        SINON
          SI le groupe a deux libertés ou plus
          ALORS
            DEBUT
              Choisir une liberté
              APPELER EVALUATION
              pour analyser la liberté choisie
            FIN
      FIN
  FIN
```

Comme dans le cas précédent, ce module analyse le coup d'un joueur. Fred évalue les intersections les plus intéressantes pour son coup suivant.

*première fois qu'intervient le critère d'**évaluation des choix** relatifs au coup à jouer, avec l'appel du module EVALUATION. Notons que le module CONTROLE, appelé par le sous-programme EVALUATION, a pour fonction d'éviter qu'une situation ayant déjà existé se reproduise sur le Gô Ban.*

*La procédure CARTE sera alors utilisée par Fred pour **chercher et analyser les libertés** d'un groupe de pions sur le Gô Ban. Quand un groupe de pions est connecté, il est crucial, pour l'adversaire, de savoir s'il est attaquable et quand, afin de le faire prisonnier... ou bien s'il est plus utile de prendre position ailleurs, soit afin de renforcer sa défense, soit en posant des jalons pour l'avenir.*

*Si, pour un joueur en chair et en os, **progresser** veut dire jouer un bon nombre de parties pour en analyser ensuite le déroulement, dans le cas de Fred, cela signifiera, cher partenaire humain, que vous aurez à cœur d'**étendre le programme** de gestion de Fred, pour en augmenter la souplesse, son état initial ne lui permettant d'"apprendre". L'implémentation de nouvelles procédures permettra ainsi à Fred de résoudre quelques problèmes de jeu plus complexes, comme les "yeux", la déclaration d'Atari, la règle du Ko... Il n'est pas difficile de réaliser ces procédures et de les rendre exécutables : il suffit de lire attentivement les règles du Gô afin de déterminer leur rôle dans la partie et d'établir ensuite le lien entre elles et le programme de gestion de l'ordinateur.*

Et maintenant, très bonne chance, cher partenaire humain. Vous voilà nanti d'un merveilleux compagnon, aussi patient qu'intelligent, et qui sera toujours "fidèlement vôtre". J.Q.

DESCRIPTION DU MODULE MODELES

```
MODELES
  DEBUT
    POUR chaque pion blanc, EXECUTER
      DEBUT
        SI il y a un modèle dans le tableau
        des exemples
        centré sur ce pion blanc
      ALORS
        DEBUT
          Etudier le coup suggéré x
          APPELER EVALUATION (x,2)
        FIN
    FIN
  FIN
```

Le comportement de Fred vise à établir des configurations de jeux qui lui soient favorables. Autrement dit, il doit non seulement appréhender la situation sur le Gô Ban mais aussi agir d'après une interprétation lui suggérant d'adopter une attitude offensive ou défensive. Ce principe est appliqué grâce au module MODELES.

DESCRIPTION DU MODULE EVALUATION

```
EVALUATION (coup, liberté)
  DEBUT
    APPELER OPTIMUS (meilleur coup, meil-
    leure liberté)
    SI liberté < = meilleure liberté, faire
      CONTROLER (coup) > = 2
    ALORS
      DEBUT
        Meilleur coup = coup
        Meilleure liberté = liberté
      FIN
  FIN
```

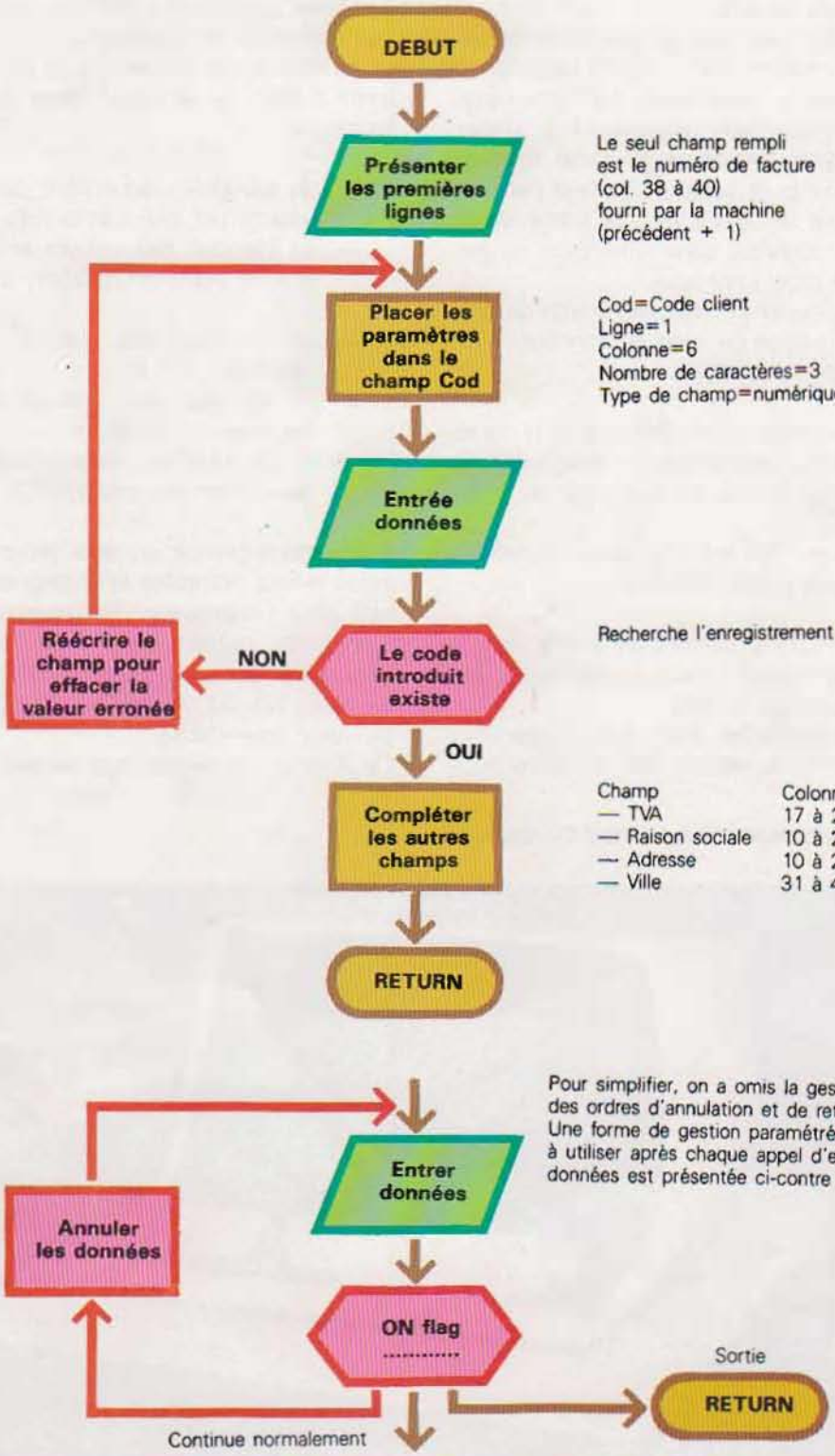
Tout comme les modules CONTROLE et MODELES, le module EVALUATION fait partie du groupe d'outils utilisés par Fred pour effectuer des analyses particulières. Le lecteur pourra agrandir ce groupe en y ajoutant des modules de résolution de situations de jeu particulières, ce qui améliorera les capacités de jeu de Fred.

DESCRIPTION DU MODULE CONTROLE

```
CONTROLE (coup)
  DEBUT
    Poser provisoirement le pion noir
    APPELER CARTE (coup, noir)
    Reprendre le pion noir
    RETOURNER au comptage des libertés
  FIN
```

Une des règles fondamentales du Gô interdit expressément la réapparition sur le Gô Ban d'une situation déjà vue.
Il faut que Fred sache appliquer correctement cette règle, appelée Ko, afin d'éviter des coups suicidaires.
C'est justement là la tâche de CONTROLE, qui garde en mémoire les paramètres courants de "coup" et de "liberté" avant que CARTE n'intervienne pour vérifier les libertés du point considéré.

SOUS-PROGRAMME DE PREPARATION DE L'EN-TETE



Le seul champ rempli est le numéro de facture (col. 38 à 40) fourni par la machine (précédent + 1)

Cod=Code client
Ligne=1
Colonne=6
Nombre de caractères=3
Type de champ=numérique

Recherche l'enregistrement

Champ	Colonne
— TVA	17 à 21
— Raison sociale	10 à 29
— Adresse	10 à 29
— Ville	31 à 40

Pour simplifier, on a omis la gestion des ordres d'annulation et de retour. Une forme de gestion paramétrée à utiliser après chaque appel d'entrée données est présentée ci-contre

Après l'introduction des données, en appuyant sur une touche fonctionnelle pré-programmée, on commande leur transfert sur disque et l'impression de la facture.
La préparation des sous-programmes constituant la procédure (voir p. 1297) ne présente pas de difficultés particulières, sauf pour ce qui concerne l'impression (alignement à ajuster) ainsi que pour l'introduction d'une ligne de données pour la facturation. Celle-ci peut encore se faire en employant la «data entry» (entrée des données) paramétré, mais sa gestion devient alors complexe.
On trouvera ci-contre l'organigramme de principe d'une gestion de ligne de données.
Trois champs sont prévus:

— code article, d'une longueur de N caractères (sa description est automatique),
— quantité (le prix est lu à partir du fichier ART),
— taux de TVA (ce champ est également lu de la même manière).

D'autres difficultés peuvent apparaître dans la phase d'impression si les quantités exprimées admettent des décimales.
Pendant l'introduction d'une ligne, l'opérateur n'est pas tenu de recourir aux nombres décimaux et peut donc se contenter de saisir un nombre entier. C'est le cas, par exemple, d'un article qui a été l'objet d'un mouvement indiqué uniquement en kg: dans ce cas, il est normal d'omettre les décimales.
Si on décide que le nombre de décimales est limité à deux, on se trouve dans les situations suivantes :

— on introduit uniquement des nombres entiers: par exemple la valeur 125,
— on introduit des valeurs entières mais avec le point (en notation américaine) : 125,
— on introduit des valeurs avec une décimale : 125.6,
— on introduit des valeurs avec deux décimales : 125.68,
— on introduit des valeurs avec plus de deux décimales : 125.6873.

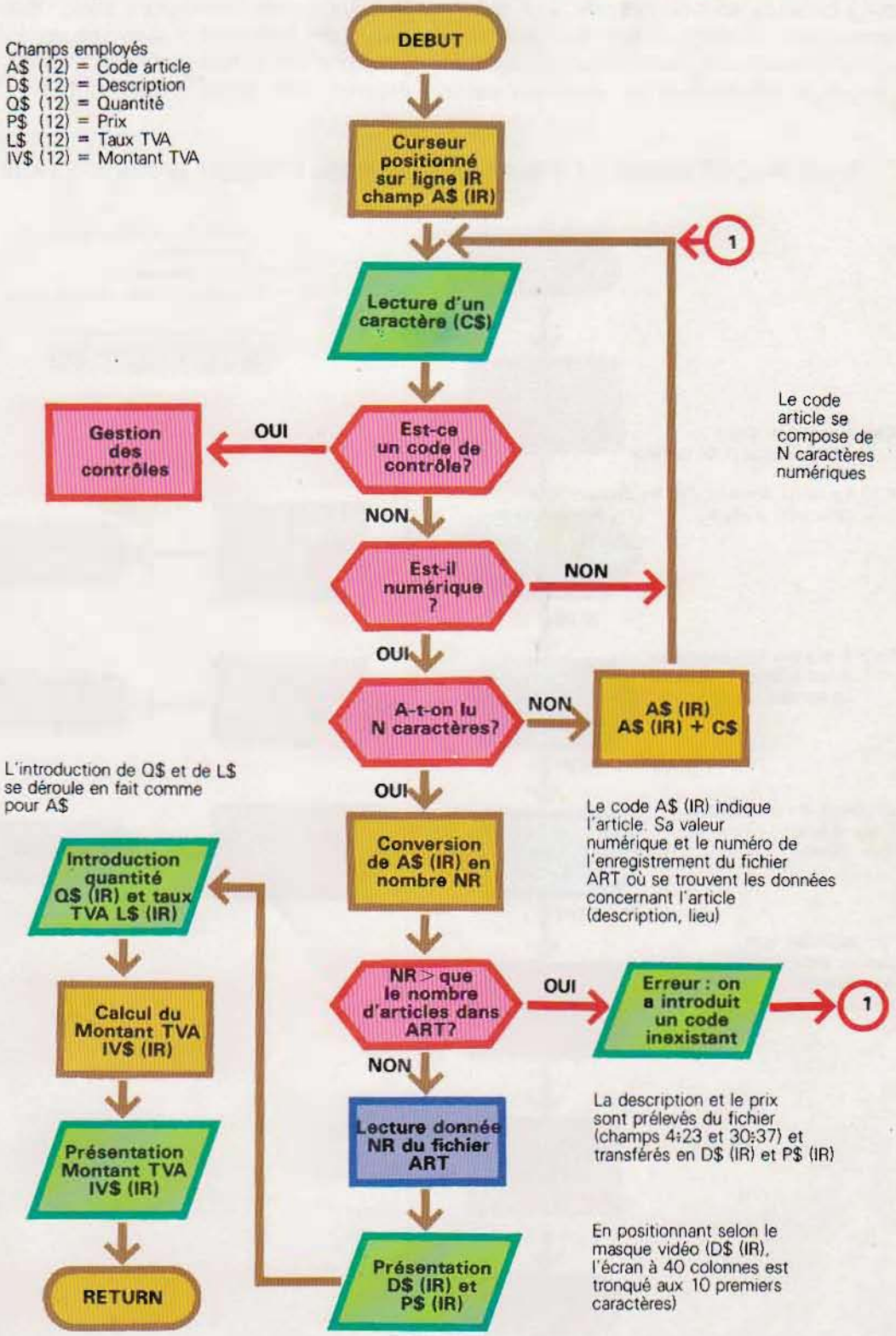
Il faut donc prévoir un sous-programme qui, selon le cas, complète le champ en le préparant pour l'impression. Par exemple: fixer le format de présentation à 4 entiers et 2 décimales ; les sorties seraient alors respectivement 125.00, 125.00, 125.60 125.68, (notation américaine).
Ce résultat est simplement obtenu avec l'ins-

Système de traitement Interstec Computer.



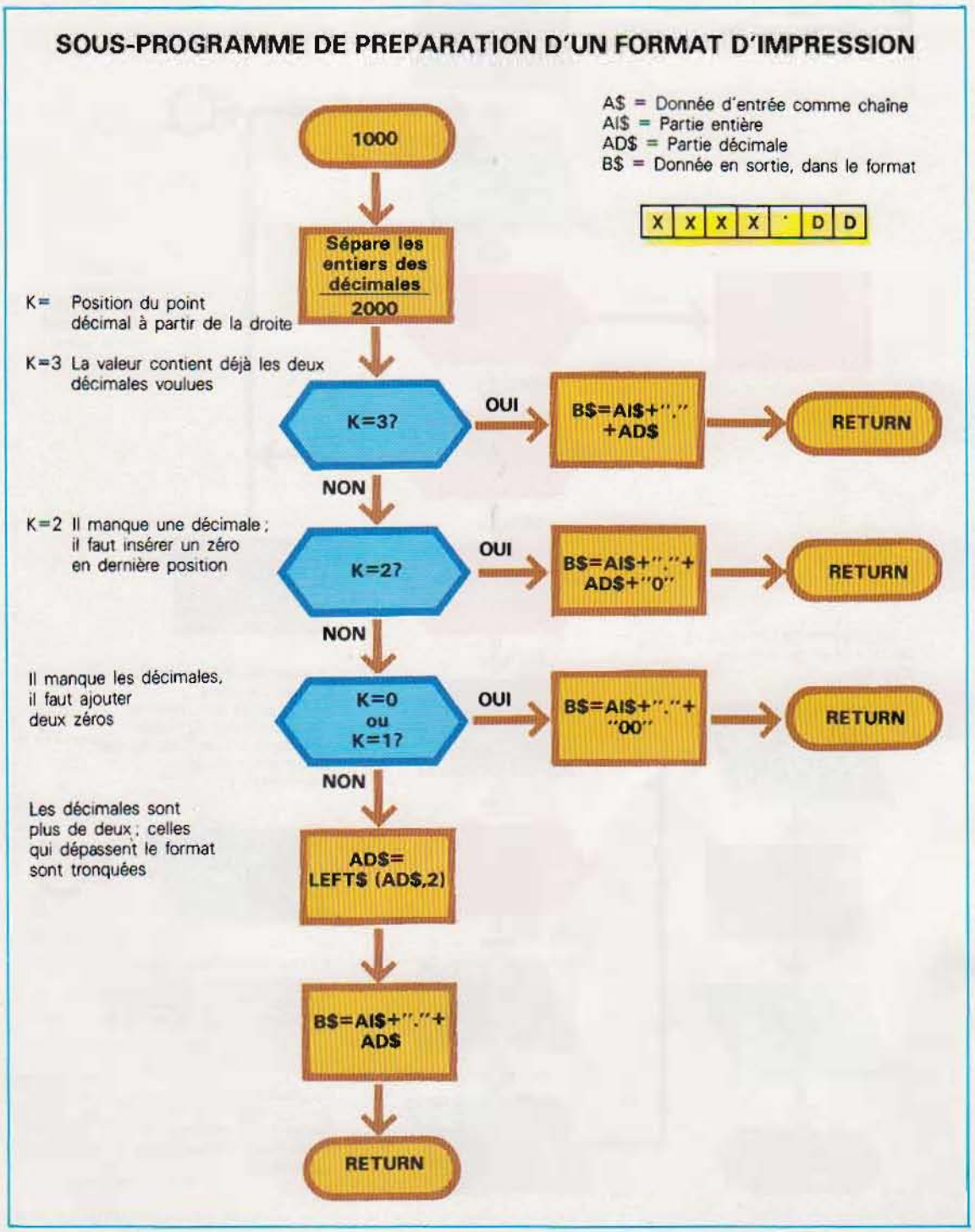
Marka/Photri

ORGANIGRAMME DE GESTION D'UNE LIGNE D'ENTREE

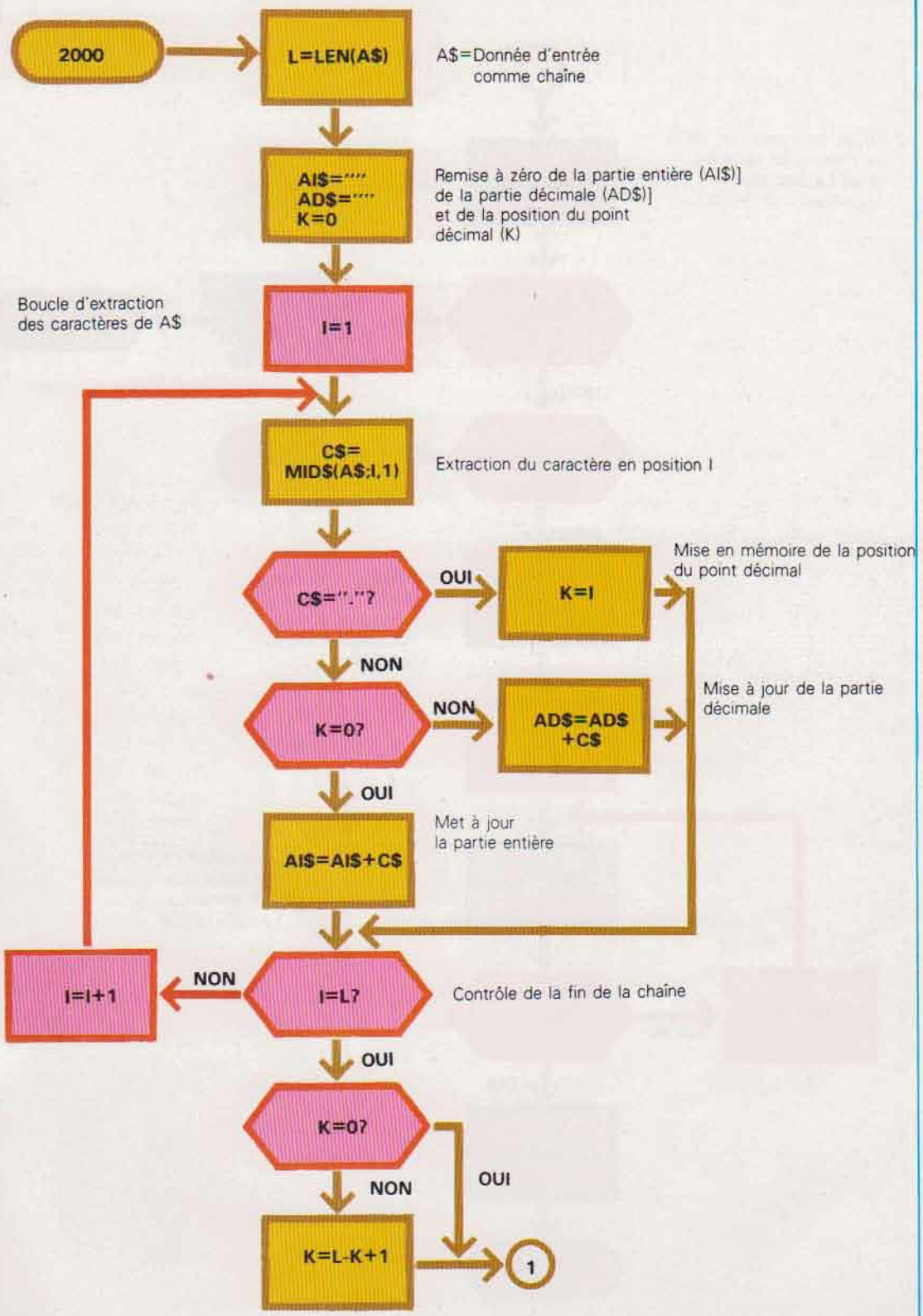


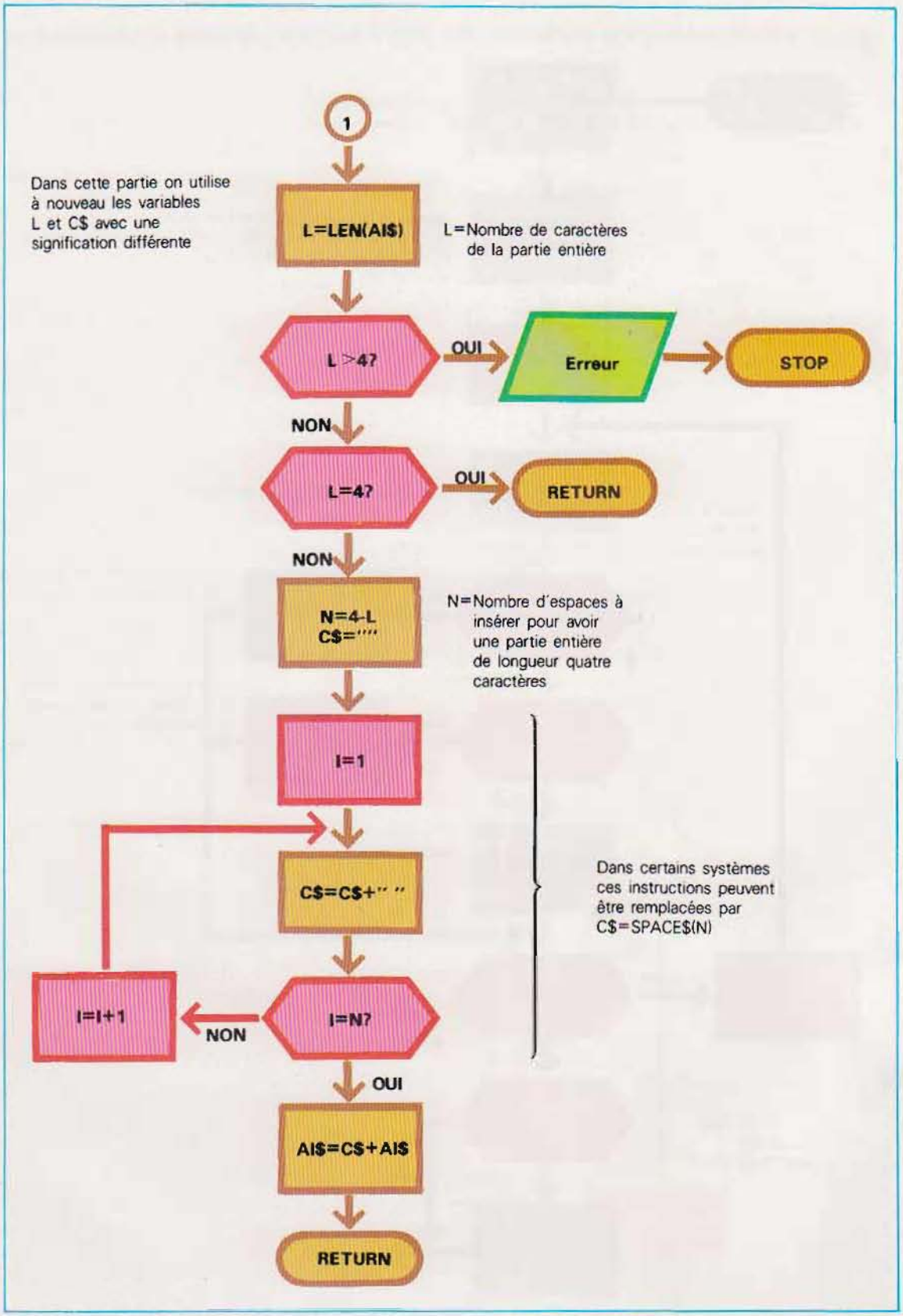
truction PRINT USING qui n'existe toutefois pas sur toutes les machines. On doit alors faire appel à un autre sous-programme. Les sous-programmes qui suivent, présentés sous la forme d'organigrammes, illustrent la préparation du format de présentation. La routine est aisément adaptable pour une machine donnée, car les deux fonctions internes, LEN et MID$ existent sur toutes les versions Basic. Naturellement, son utilité est limitée aux versions de l'interpréteur ne possédant pas d'option d'impression avec format (PRINT USING).

SOUS-PROGRAMME DE PREPARATION D'UN FORMAT D'IMPRESSION



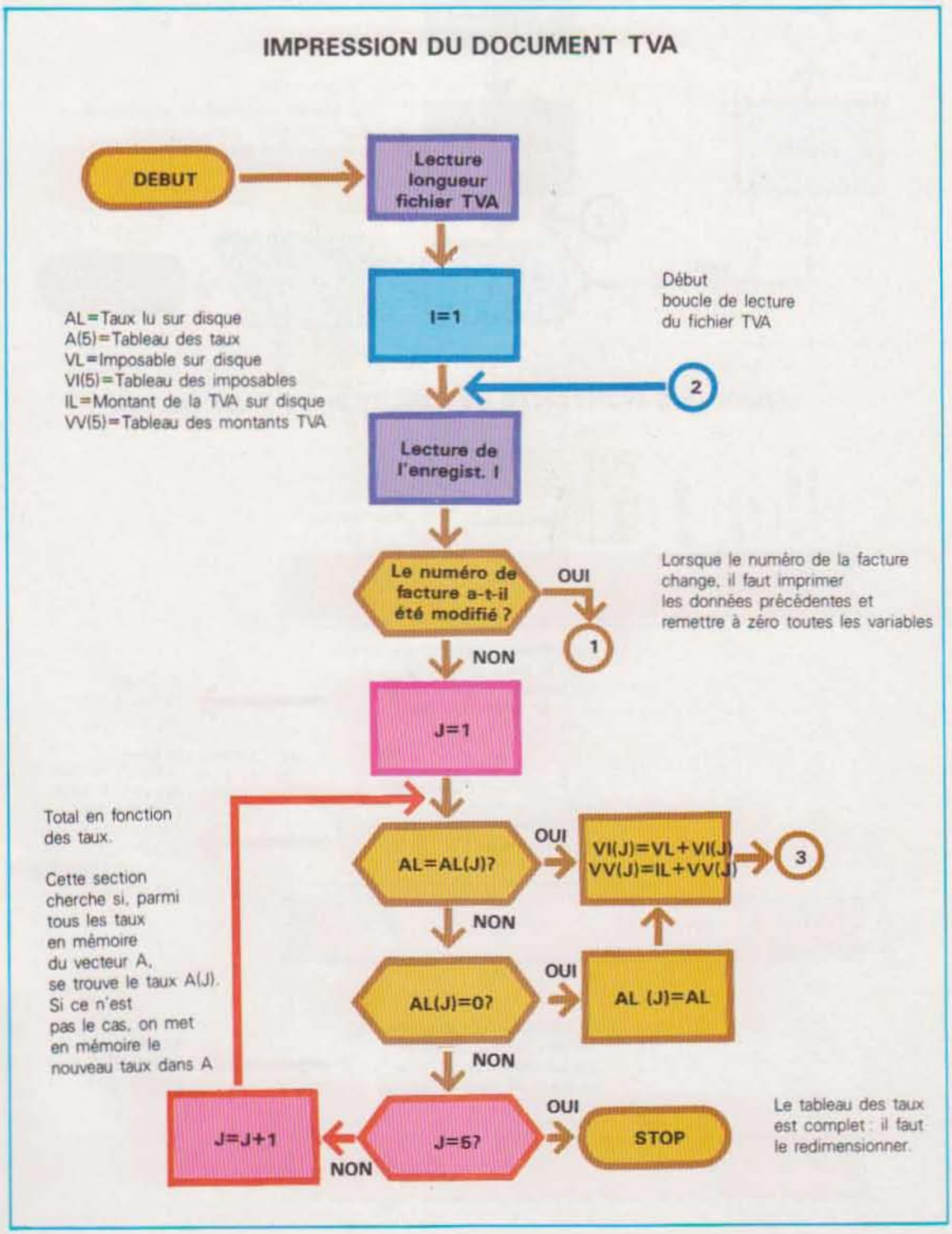
SOUS-PROGRAMME DE SEPARATION DES PARTIES ENTIERE ET DECIMALE

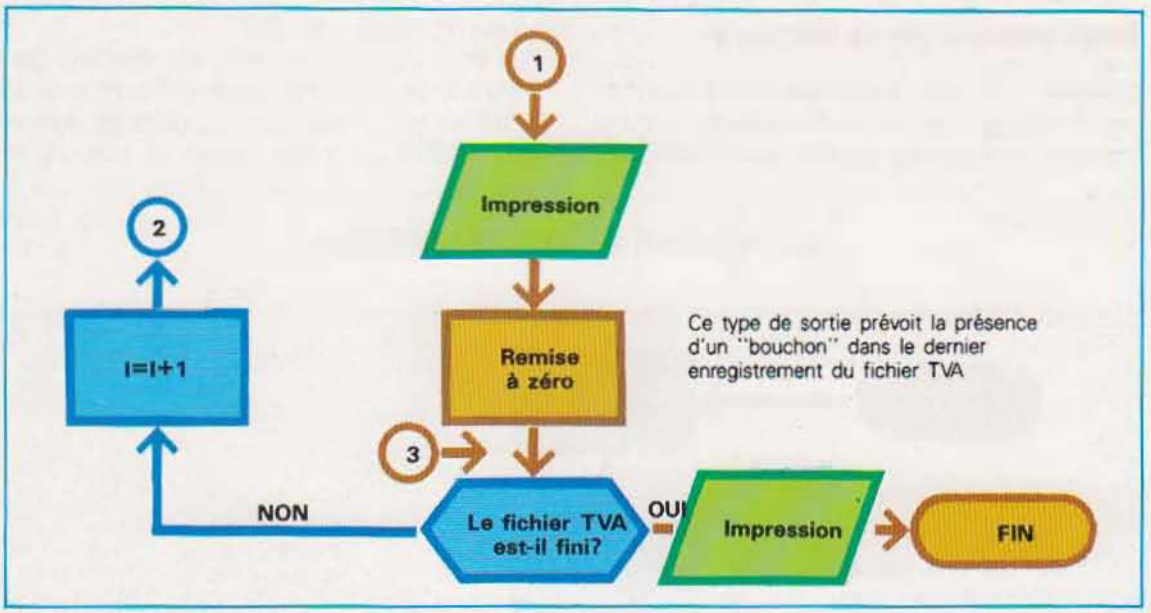



1
Dans cette partie on utilise à nouveau les variables L et C$ avec une signification différente
L=LEN(AI$)
L=Nombre de caractères de la partie entière
L >4?
OUI
Erreur
STOP
NON
L=4?
OUI
RETURN
NON
N=4-L
C$=""
N=Nombre d'espaces à insérer pour avoir une partie entière de longueur quatre caractères
I=1
C$=C$+" "
I=I+1
NON
I=N?
OUI
AI$=C$+AI$
RETURN
Dans certains systèmes ces instructions peuvent être remplacées par C$=SPACE$(N)

Impression de la facture

Selon une périodicité donnée, on va imprimer un document (bulletin ou formulaire) qui comportera, pour chaque facture, le nom du client et les montants ventilés.
A l'émission des factures, les données sont rangées dans le fichier TVA et il suffira de les additionner par rubrique. La méthode illustrée page 1303 et 1304 utilise le numéro de





CUMUL DES MONTANTS EN FONCTION DU TAUX DE TVA

Numéro de facture	Code Client	Donnée	Taux	Imposable	Montant TVA
26	03	21×12	10	10000	1000

La première lecture trouve le A(J) vide et place le taux 10 en première position (J=1). A la fin du premier tour J=1 on a:

→ A(1)=10
VI(1)=10000
VV(1)=1000

Le deuxième enregistrement a le même taux. Le programme additionne les données dans la même position

→ A(1)=10 (le taux est un indicateur; il ne faut pas l'additionner)
VI(1)=35000 (10000+25000)
VV(1)=3500

26	03	21×12	10	25000	2500

Le troisième enregistrement contient un taux différent qui est donc mis en mémoire dans la position suivant (J=2). On obtient:

→ A(2)=8
V(2)=20000
VV(2)=1600

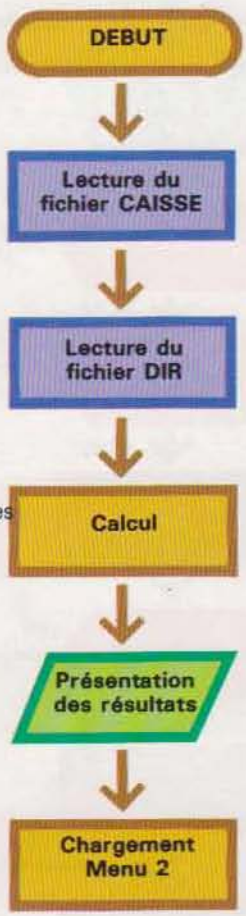
26	03	21×12	8	20000	1600

facture comme code. Le changement de numéro déclenche l'impression des additions correspondantes, la remise à zéro et le début d'un nouveau cycle. En plus de l'impression du bulletin, cette routine sera également utilisée en fin de calcul du document pour faire ressortir, facture par facture, les montants imposables et les TVA en fonction du taux retenu. Ces impressions sont à nouveau lancées en fin d'année quand il faut mettre en mémoire, dans le fichier TVA, des données récapitulatives.

Autres fonctions

Pour compléter le programme, d'autres procédures sont obtenues en y introduisant quelques modifications. Les fonctions (Situation de Caisse et Recettes/Dépenses organigrammes ci-dessous et p. 1306) de deux fonctions sont semblables et ne présentent aucune difficulté d'écriture par rapport aux précédentes.
En plus des modules ou des sous-programmes nécessaires à cette application particulière,

ORGANIGRAMME DE LA SITUATION DE CAISSE

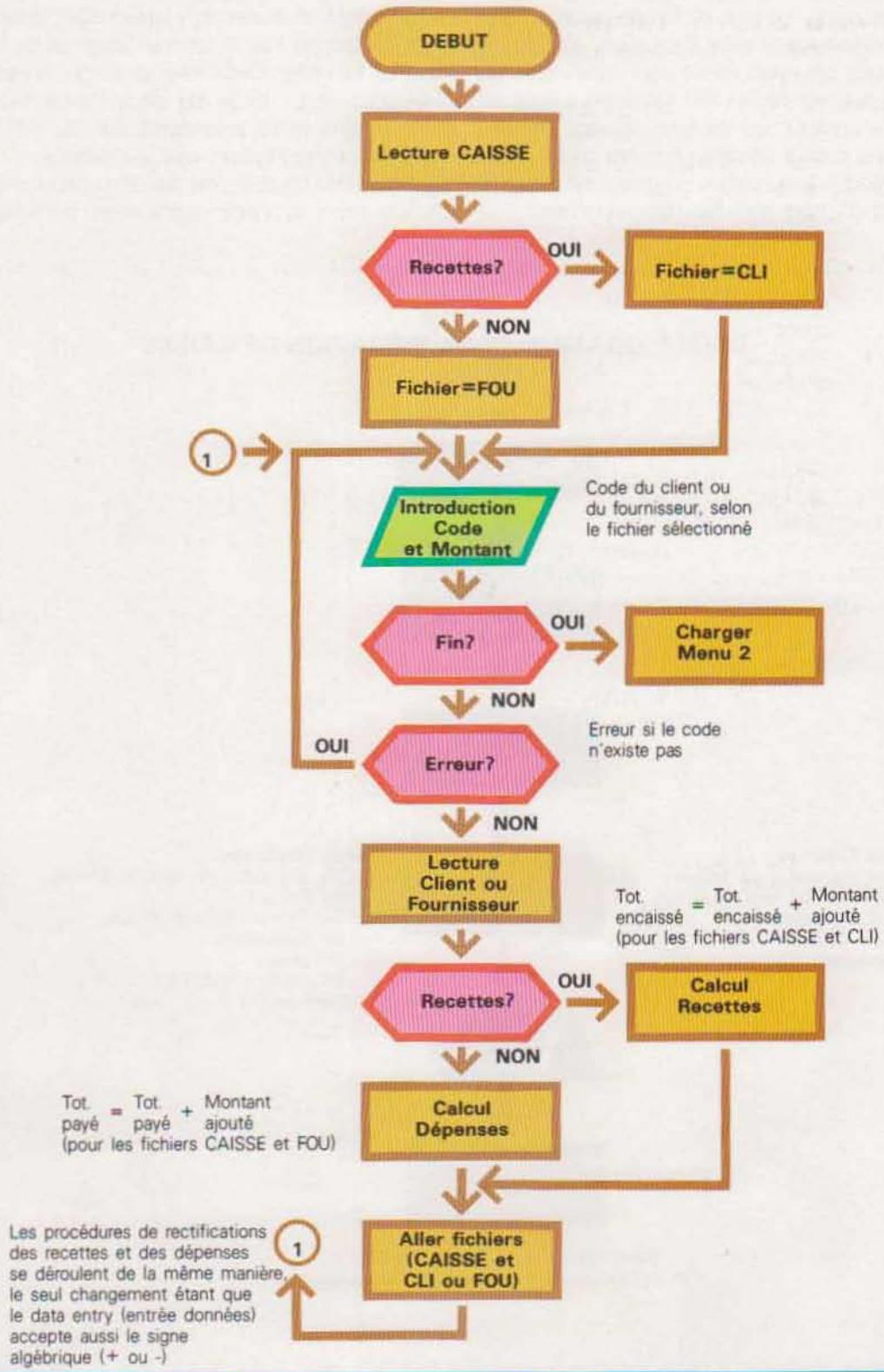


Calcul Recettes
1 - Tot. imposable sur factures émises
2 - Tot. TVA sur factures émises
3 - Tot. encaissé
4 - Tot. avoir=(1+2)-3
5 - Recettes=4-2

Calcul Dépenses
1 - Tot. imposable sur factures émises par fournisseurs
2 - Tot. TVA sur factures émises par fournisseurs
3 - Tot. payé
4 - Tot. débit=(1+2)-3
5 - Dépenses=4-2

Bénéfice=5 (recettes) -5 (dépenses)
TVA (crédit/verser) =2 (recettes) -2 (dépenses)

ORGANIGRAMME RECETTES/DEPENSES



on a besoin d'autres routines comme SORT (TRI) d'intérêt général. Le classement des données est une fonction très employée, d'où précisément la nécessité d'un programme général, susceptible d'être adapté aisément à différents contextes.

On trouvera ci-dessous l'organigramme d'un programme de tri. Le premier bloc génère des chaînes aléatoires (détails p. 1309) auxquelles on applique, à titre de contrôle, le sous-programme SORT (routine 590 p. 1310). Dans notre exemple, les chaînes sont obtenues automatiquement avec le générateur de nombres aléatoires RND (voir listing page 1308).

On peut également prévoir, comme alternative, une boucle d'introduction à partir du clavier ou d'un lecteur de disque.

Le sous-programme 590, appelé à la ligne 260 du programme principal, réalise le tri et restitue le tableau IS des pointeurs aux données en ordre croissant. Le dernier bloc imprime les valeurs ordonnées.

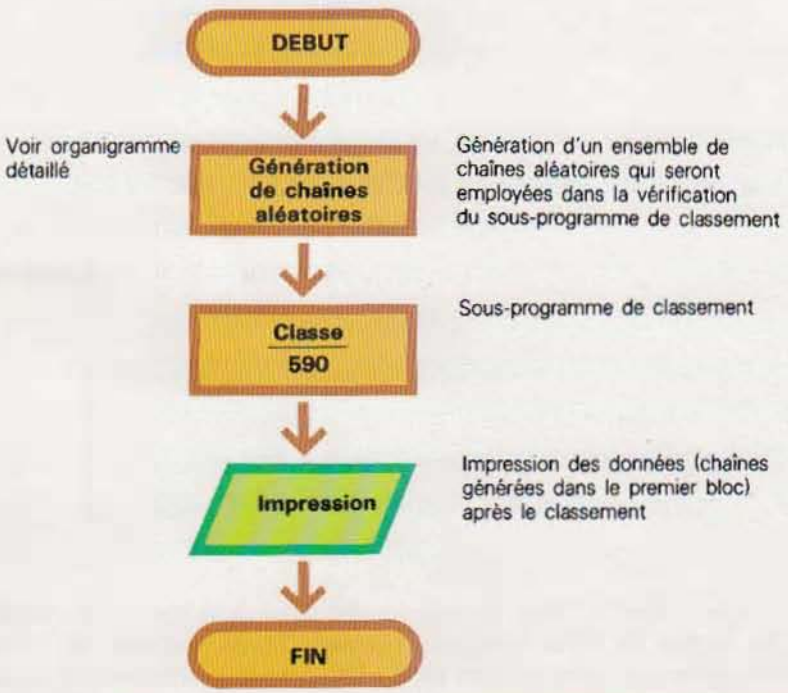
Page 1310, on trouvera l'organigramme de premier niveau de ce sous-programme qui sera détaillé pages 1311 à 1313 (voir listings



B. Coleman/Marka

Système de traitement pour la CAO et le DAO.

ORGANIGRAMME D'UN PROGRAMME DE TRI

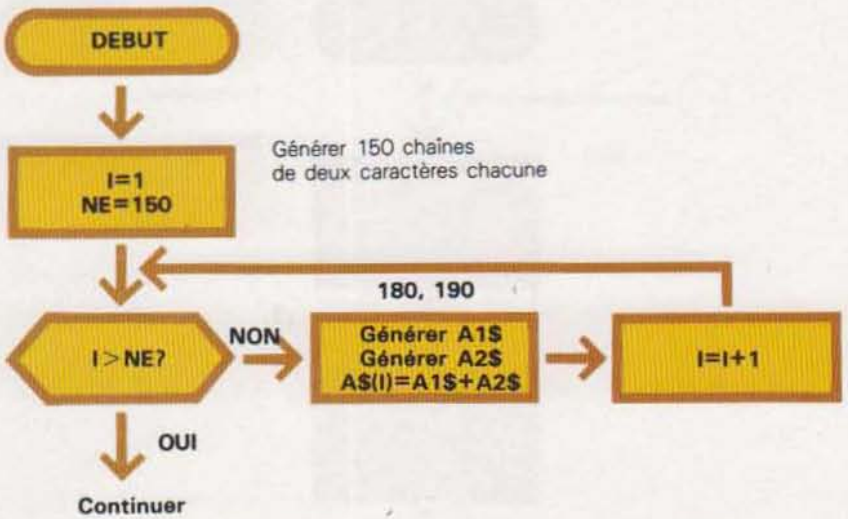


PROGRAMME DE CLASSEMENT (PRINCIPAL)

```
10 REM  ******************
20 REM  *                *
30 REM  *      TRI       *
40 REM  *                *
50 REM  ******************
60 :
70 HOME
80 DIM A$(150),IS(160)
90 NE=150
100:
110 REM ==========================
120 REM * GENERATION DE LETTRES *
130 REM ==========================
140 :
150 HTAB 7
160 VTAB 5:PRINT "GENERATION DE LETTRES
    ALEATOIRES"
170 FOR I=1 TO NE
180 A1$=CHR$ (INT (RND (1)*26+65)) : A2$=CHR$ (INT (RND (1)*26+65))
190 A$(I)=A1$+A2$
200 NEXT I
210 :
220 REM ==================
230 REM * ROUTINE DE TRI *
240 REM ==================
250 :
260 GOSUB 590
270 :
280 :
290 REM ===============
300 REM * IMPRESSION *
310 REM ===============
320 :
330 HOME
340 X=2:Y=1
350 FOR I=1 TO NE
360 K=IS(I):C$=A$(K)
370 HTAB X:VTAB Y:PRINT C$
380 IF Y=23 THEN Y=1:X=X+
        3:GOTO 400
390 Y=Y+2
400 NEXT I
410 VTAB 24:HTAB 38:GET D$
420 NORMAL
430 HOME:VTAB 24
440 END
450 :
```

correspondants pages 1313 à 1315). La logique employée est fondée sur la recherche binaire. La première phase consiste à calculer M, pointeur de comparaison. A l'initialisation, M équivaut à la moitié du nombre de données classées (la première fois, c'est 1).
On considère ensuite un à un tous les éléments de la liste, avec une boucle allant de 1 à NE

GENERATION DE CHAINES ALEATOIRES



(nombre de données, instructions 640 à 1440 du listing).
Chacun d'eux est comparé à l'élément pointé par M. Par exemple, avec I = 25 et M = 6, on compare A$(25) à A$(6), et on détermine, en fonction du résultat, si la position de I dans le tableau des pointeurs a été trouvée ou s'il faut effectuer une nouvelle recherche. Dans ce dernier cas, on recalcule M, après avoir établi s'il faut le déplacer à droite ou à gauche.
Si la position est trouvée, on met en mémoire le pointeur de l'élément dans le tableau IS, en opérant, le cas échéant, une translation des pointeurs précédemment insérés.
A la fin, le sous-programme de tri a mémorisé, dans IS, les pointeurs des éléments de AS selon un ordre croissant. Pour obtenir l'impression ordonnée, il faut extraire de A$ chaque élément selon l'ordre mémorisé en IS.
Supposons, par exemple, que IS contienne :

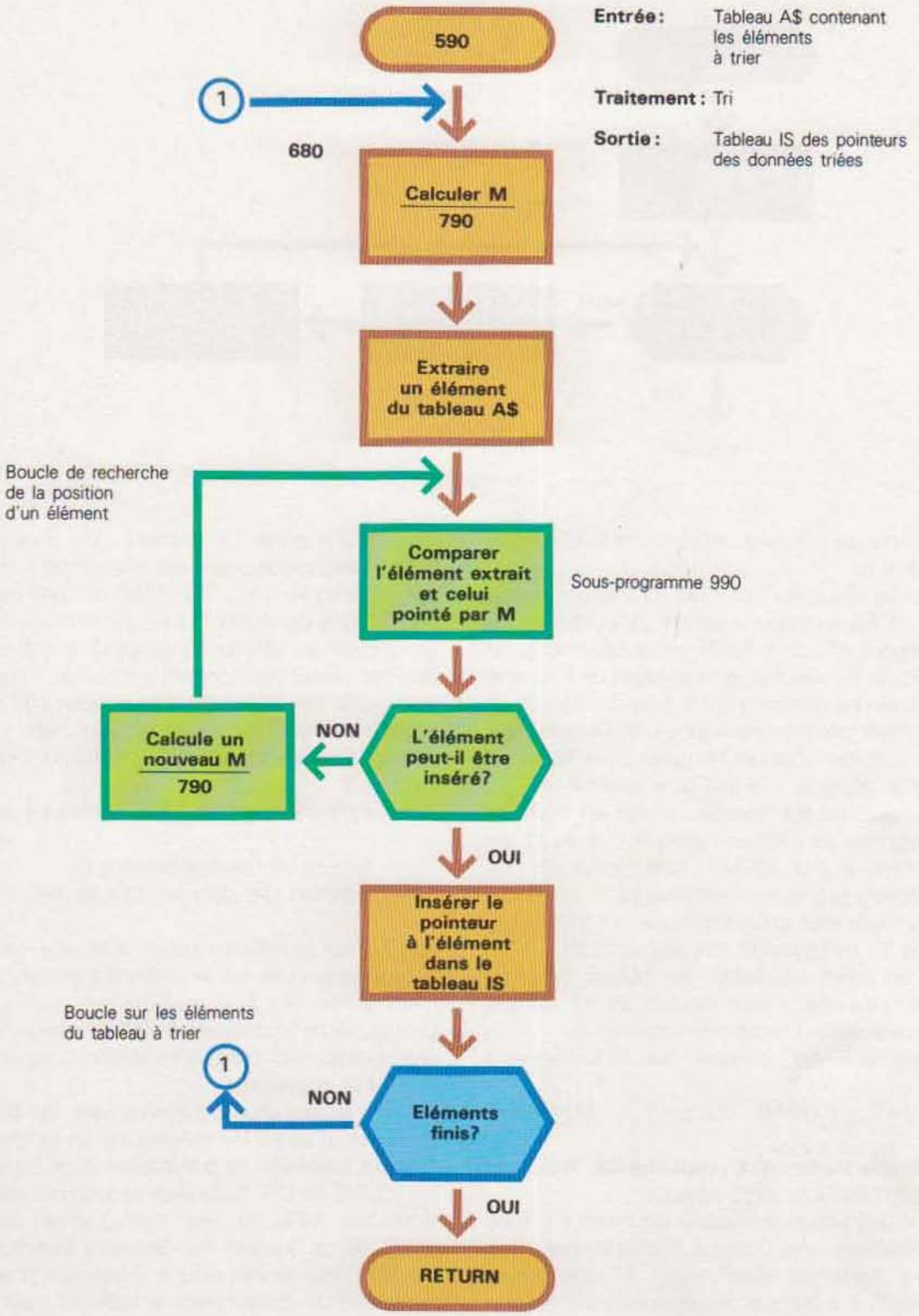
IS(1)=7, IS(2)=9, IS(3)=1, IS(4)=15,...

le premier élément à imprimer est A$(7), le second A$(9) et ainsi de suite.
Le listing présenté travaille en mémoire, sans paramétrage des champs à sélectionner.
Dans certaines applications, il peut arriver qu'on doive prélever les données des fichiers du disque ou encore n'exécuter de tri qu'à partir d'une partie du contenu des champs. La première modification est très simple à réaliser. Aux lignes 710, 730, 1130 et 1240 dans les variables de chaîne IV$ et I1$, on transfère un élément du tableau A$ supposé en mémoire. Pour utiliser des données sur disque, il suffit de modifier ces instructions en remplaçant l'assignation simple, par exemple IV$=A$(I), par une lettre sur l'enregistrement pointé par l'indice de A$.
Ainsi l'instruction 710 doit être remplacée par :

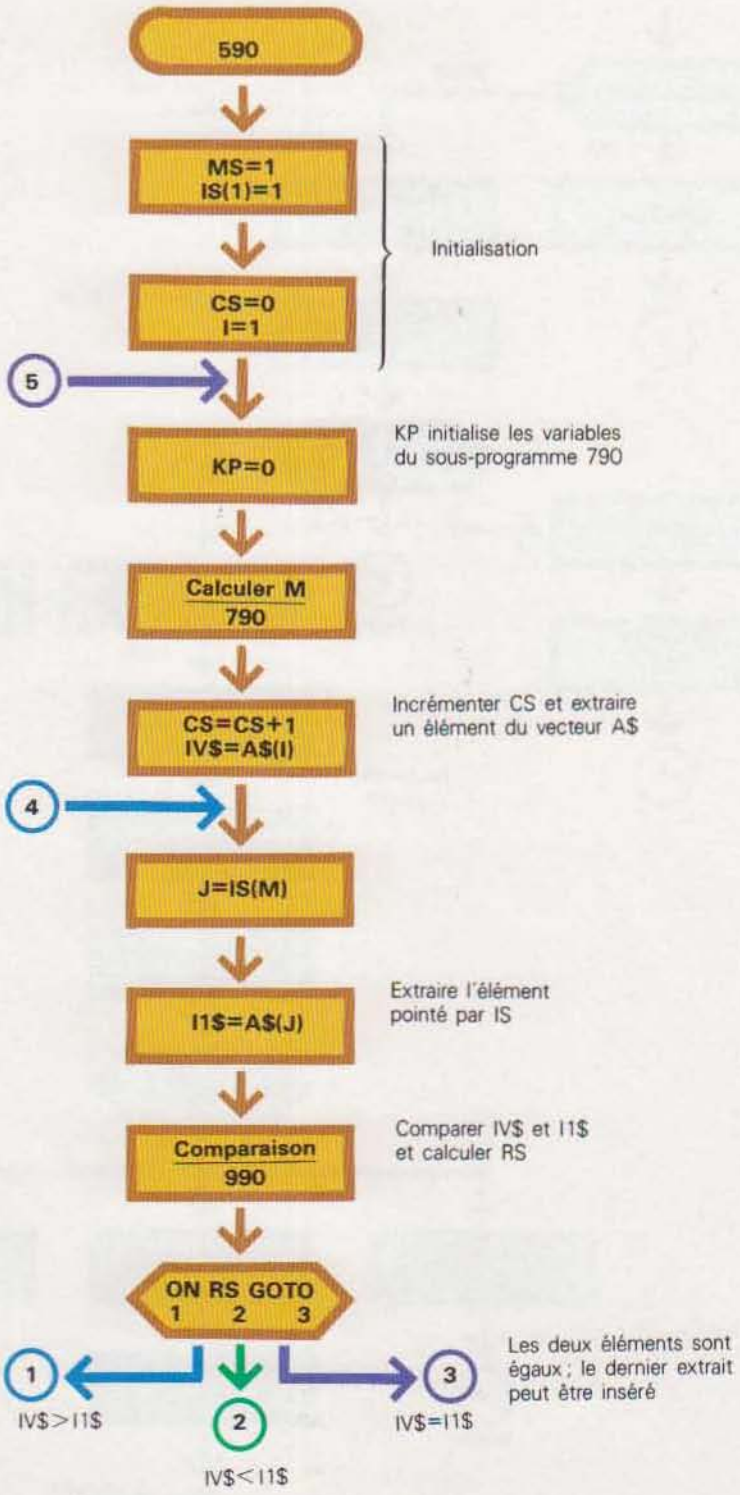
— lecture de l'enregistrement I
— transfert des données lues en IV$.

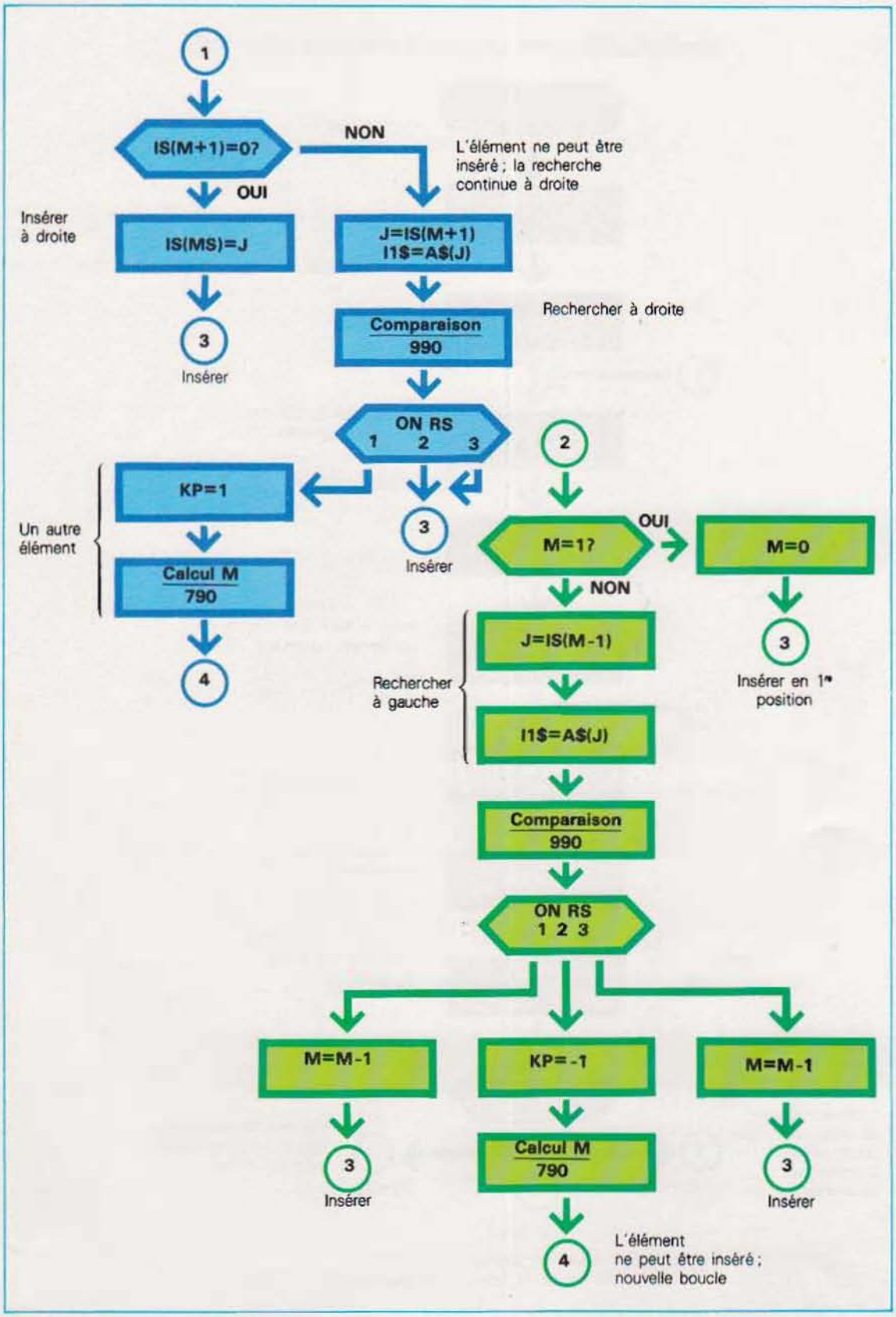
La 730 suit la même logique, avec une seule différence portant sur le numéro d'enregistrement qui devient J.
La deuxième modification (paramétrage du champ employé pour le tri) s'obtient conjointement à la première.
Dans la lecture d'un enregistrement, les données sont d'abord transférées sur un tampon. Ne sont prélevées et transférées dans les variables IV$ ou I1$, que celles se trouvant dans la position adéquate (voir organigramme page 1315). Si on travaille sur disque, il faut aussi modifier l'impression, pour ménager une phase de lecture du disque avant le transfert dans la variable C$ (ligne 360).

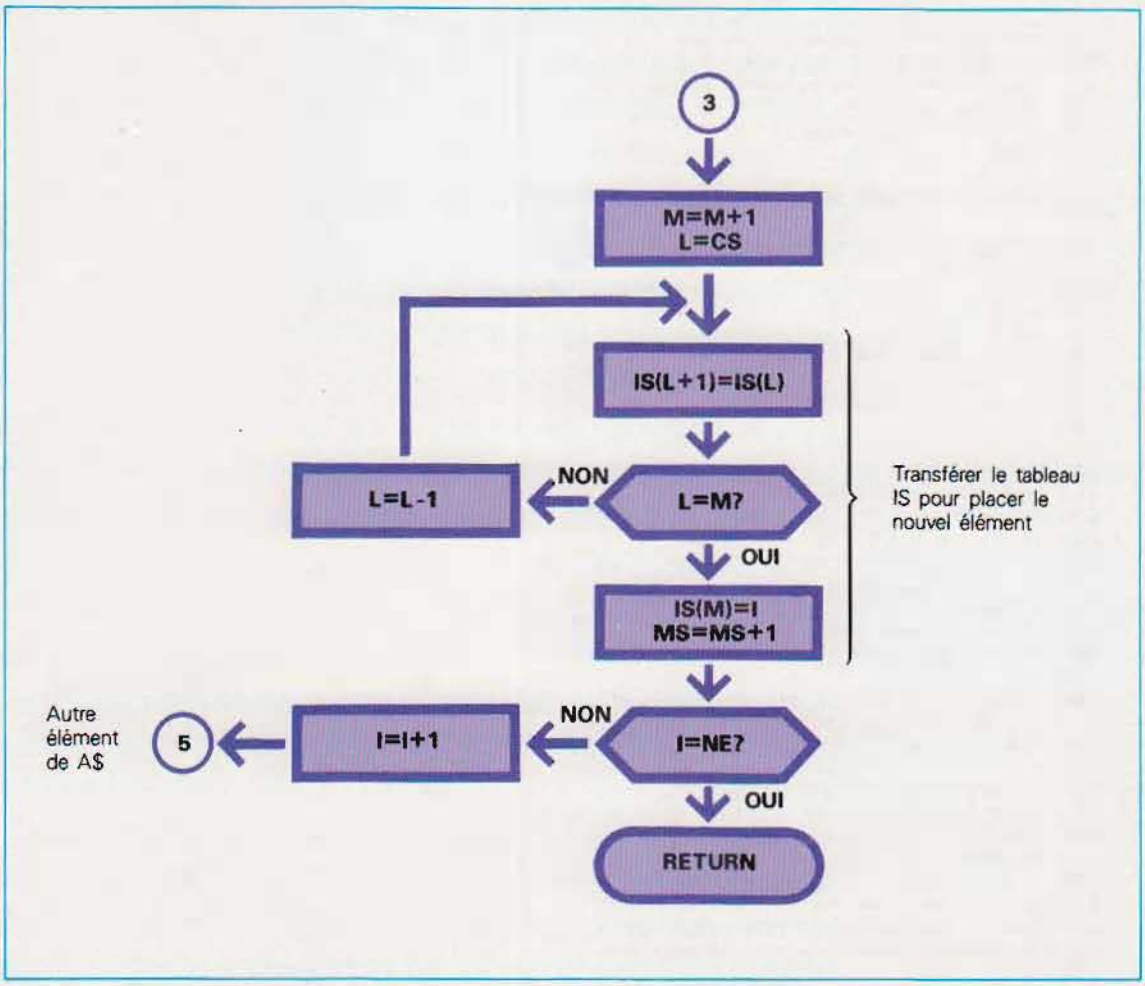
ORGANIGRAMME DE PRINCIPE DU SOUS-PROGRAMME 590



ORGANIGRAMME DU SOUS-PROGRAMME 590




1
IS(M+1)=0?
NON
L'élément ne peut être inséré ; la recherche continue à droite
OUI
Insérer à droite
IS(MS)=J
J=IS(M+1)
I1$=A$(J)
3
Insérer
Comparaison
990
Rechercher à droite
ON RS
1 2 3
2
KP=1
3
Insérer
M=1?
OUI
M=0
Un autre élément
Calcul M
790
NON
J=IS(M-1)
3
Insérer en 1re position
Rechercher à gauche
I1$=A$(J)
4
Comparaison
990
ON RS
1 2 3
M=M-1
KP=-1
M=M-1
3
Insérer
Calcul M
790
3
Insérer
4
L'élément ne peut être inséré ; nouvelle boucle



PROGRAMME DE CLASSEMENT (SOUS-PROGRAMME)

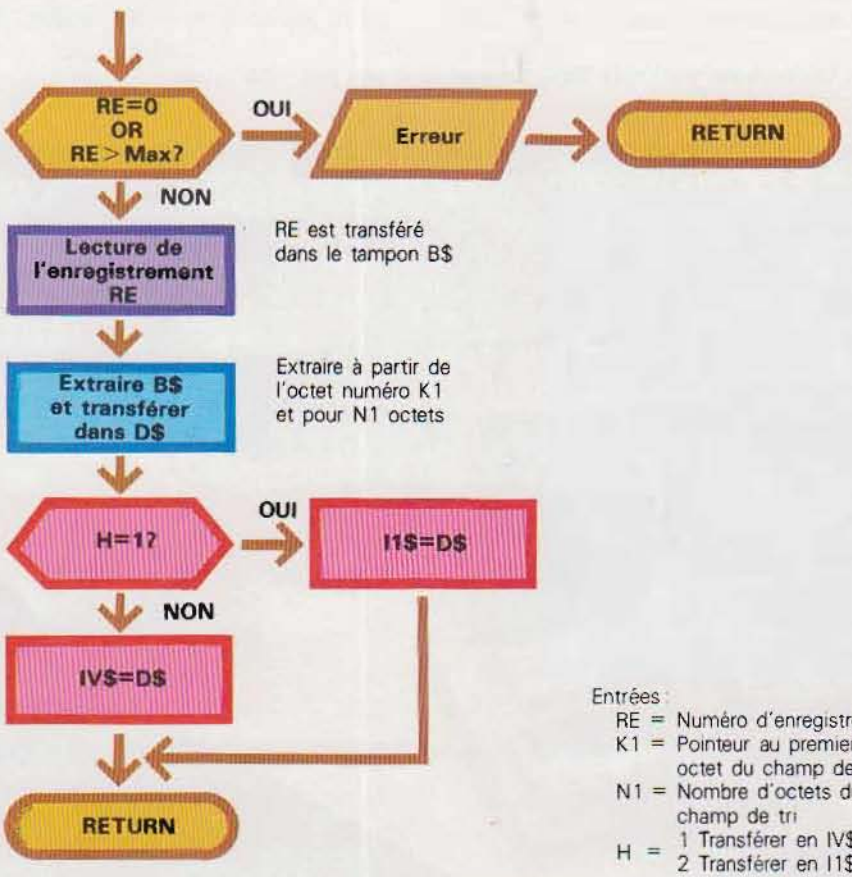
```
460 REM ============
470 REM *    TRI    *
480 REM ============
490 :
500 REM ====================================
510 REM * IS(.) = VECTEUR CLASSE           *
520 REM * MS    = MAX POINTEUR IS()        *
530 REM * CS    = NBRE D'ELEMENTS CLASSES  *
540 REM * A$()  = VECTEUR A CLASSER        *
550 REM * IV$   = ELEMENT A CLASSER        *
560 REM * I1$   = ELEMENT DE COMPARAISON   *
570 REM * M     = POINTEUR DE RECHERCHE    *
580 REM ====================================
590 MS=1:IS(1)=1:CS=0
600 HOME:VTAB 10:INVERSE:PRINT "JE CLASSE L'ELEMENT N° : "
```

```
610 REM ========================
620 REM * CYCLE DE CLASSEMENT *
630 REM ========================
640 FOR I=1 TO NE
650 HTAB 24:VTAB 10:PRINT I
660 KP=0
670 :
680 GOSUB 790: REM * M *
690 :
700 CS=CS+1
710 IV$=A$(I)
720 REM
730 J=IS(M):I1$=A$(J)
740 :
750 GOSUB 990: REM * CONF. *
760 :
770 ON RS GOI 1080.1090.1320
780 :
790 REM ==================
800 REM * CALCUL DE (M) *
810 REM ==================
820 :
830 IF KP<>0 THEN 870
840 MP=1:PP=MS:PS=0:M=INT (MS/2)
850 IF M=0 THEN M=1
860 RETURN
870 IF KP=1 THEN 900
880 IF KP= -1 THEN 950
890 :
900 MP=M:IF PP=0 THEN PP=MS
910 M2=INT ((PP-M)/2):IF M2=0 THEN M2=1
920 M=M+M2
930 :
940 RETURN
950 PP=M:M2=INT ((M-MP)/2):IF M2=0 THEN M2=1
960 M=M-M2
970 RETURN
980 :
990 REM  ============================
1000 REM * COMPARAISON DES ELEMENTS *
1010 REM ============================
1020 :
1030 RS=3
1040 IF IV$>I1$ THEN RS=1
1050 IF IV$<J1$ THEN RS=2
1060 RETURN
1070 :
1080 REM ====================
1090 REM *    IV$ > I1$     *
1100 REM ====================
1110 :
1120 IF IS(M+1)=0 THEN IS(MS)=J:GOTO 1320
1130 J=IS(M+1):I1$=A$(J)
1140 GOSUB 990
1150 ON RS GOTO 1160.1320.1320
1160 KP=1
1170 GOSUB 790:GOTO 720
1180 :
1190 REM ====================
1200 REM *    IV$ <  I1$    *
1210 REM ====================
1220 :
1230 IF M=1 THEN M=0:GOTO 1320
1240 J=IS(M-1):I1$=A$(J)
1250 GOSUB 990
```

```
1260 ON RS GOTO 1270,1280,1270
1270 M=M-1:GOTO 1320
1280 KF= -1
1290 GOSUB 790
1300 GOTO 720
1310 :
1320 REM ==================
1330 REM *     IV$=I1$     *
1340 REM ==================
1350 :
1360 M=M+1
1370 :
1380 REM * CONSTRUCTION DE IS(.) *
1390 :
1400 FOR I=CS TO M STEP -1
1410 IS(L+1)=IS(L)
1420 NEXT L
1430 IS(M)=I:MS=MS+1
1440 NEXT I
1450 RETURN
```

GENERALISATION DU SOUS-PROGRAMME DE TRI

Lecture du disque
Extraction de la partie utilisée pour le tri
Transfert à la variable correspondante



Entrées :
RE = Numéro d'enregistrement
K1 = Pointeur au premier octet du champ de tri
N1 = Nombre d'octets du champ de tri
H = 1 Transférer en IV$
2 Transférer en I1$

La transmission des informations

En informatique, le concept de transmission de données signifie un déplacement d'un point à un autre d'informations codées, au moyen de systèmes de communication de type électronique ou électromécanique. La distance séparant les deux points en question est théoriquement illimitée, mais il faut préciser qu'elle conditionne le choix d'un support de transmission.

Dans les moyens traditionnels (téléphone, radio, télévision), l'information (voix, image ou texte écrit) est convertie, depuis sa source, en signaux électromagnétiques. Elle est transmise grâce à un moyen «diélectrique» (le câble métallique d'une ligne téléphonique, l'éther). A la réception, les signaux sont retransformés, de manière à restituer l'information dans sa forme initiale.

Examinons maintenant de plus près la communication des informations dans le cas de l'ordinateur. Le premier transfert a lieu lors du dialogue homme-machine. L'ordinateur fonctionne en utilisant des séquences de chiffres binaires 0 et 1: pour communiquer avec l'homme il lui faut donc traduire chaque séquence de bits en caractères alphabétiques et numériques. La traduction est réalisée au moyen de tableaux de correspondance bi-univoque entre caractères (code homme) et séquences de bits (code machine). Les formes de codage les plus répandues sont l'ASCII (American Standard Code for Information Interchange) et l'EBCDIC (Extended Binary Coded Decimal Interchange Code); la première est couramment employée dans les micro-ordinateurs, la seconde est typique des grosses machines IBM. Ces deux codes diffèrent par les combinaisons de bits associées à un caractère. Tous deux comportent, outre les chiffres et les lettres, des symboles

Système de traitement portable NEC connecté à ses périphériques.



R. Villa/Marka

CODIFICATION ASCII DES CARACTERES SPECIAUX EMPLOYES EN TELETRANSMISSION

Caractère	Signification	Codification ASCII bit n° 8*7654321	Valeur décimale	Valeur octale	Valeur hexadécimale
NUL	Null	0 0 0 0 0 0 0	0	0	0
SOH	Start of Heading	0 0 0 0 0 0 1	1	1	1
STX	Start of Text	0 0 0 0 0 1 0	2	2	2
ETX	End of Text	0 0 0 0 0 1 1	3	3	3
EOT	End of Transmission	0 0 0 0 1 0 0	4	4	4
ENQ	Enquiry	0 0 0 0 1 0 1	5	5	5
ACK	Acknowledgement	0 0 0 0 1 1 0	6	6	6
BEL	Bell	0 0 0 0 1 1 1	7	7	7
BS	Backspace	0 0 0 1 0 0 0	8	10	8
HT	Horizontal Tabulation	0 0 0 1 0 0 1	9	11	9
NL	New Line	0 0 0 1 0 1 0	10	12	A
VT	Vertical Tabulation	0 0 0 1 0 1 1	11	13	B
FF	Form Feed	0 0 0 1 1 0 0	12	14	C
RT	Return	0 0 0 1 1 0 1	13	15	D
SO	Shift Out	0 0 0 1 1 1 0	14	16	E
SI	Shift In	0 0 0 1 1 1 1	15	17	F
DLE	Data Line Escape	0 0 1 0 0 0 0	16	20	10
DC1	Device Control 1	0 0 1 0 0 0 1	17	21	11
DC2	Device Control 2	0 0 1 0 0 1 0	18	22	12
DC3	Device Control 3	0 0 1 0 0 1 1	19	23	13
DC4	Device Control 4	0 0 1 0 1 0 0	20	24	14
NAK	Negative Acknowledgement	0 0 1 0 1 0 1	21	25	15
SYN	Synchronous Idle	0 0 1 0 1 1 0	22	26	16
ETB	End of Transmission Block	0 0 1 0 1 1 1	23	27	17
CAN	Cancel	0 0 1 1 0 0 0	24	30	18
EM	End of Medium	0 0 1 1 0 0 1	25	31	19
SUB	Substitute	0 0 1 1 0 1 0	26	32	1A
ESC	Escape	0 0 1 1 0 1 1	27	33	1B
FS	File Separator	0 0 1 1 1 0 0	28	34	1C
GS	Group Separator	0 0 1 1 1 0 1	29	35	1D
RS	Record Separator	0 0 1 1 1 1 0	30	36	1E
US	Unit Separator	0 0 1 1 1 1 1	31	37	1F

*Le bit n° 8 est le bit de parité.

spéciaux qui sont justement employés dans la communication de données et dont on parlera plus loin. On trouvera ci-dessus la codification ASCII des caractères spéciaux qui concernent la communication des données et, page suivante, la codification EBCDIC. Pour l'ASCII, sur lequel on reviendra, on indique également la signification des différents caractères.

Modes de transmission

Nous avons déjà dit que ces deux systèmes emploient 8 bits pour représenter un caractère. Généralement 7 bits seulement contiennent l'information proprement dite, le huitième étant le bit de parité.

Pour transmettre un caractère entre deux équipements, par exemple un ordinateur et son périphérique, il est nécessaire de les relier par une ligne de communication. Selon la manière de réaliser cette ligne, on obtient deux types de transmission (voir schémas p. 1319):

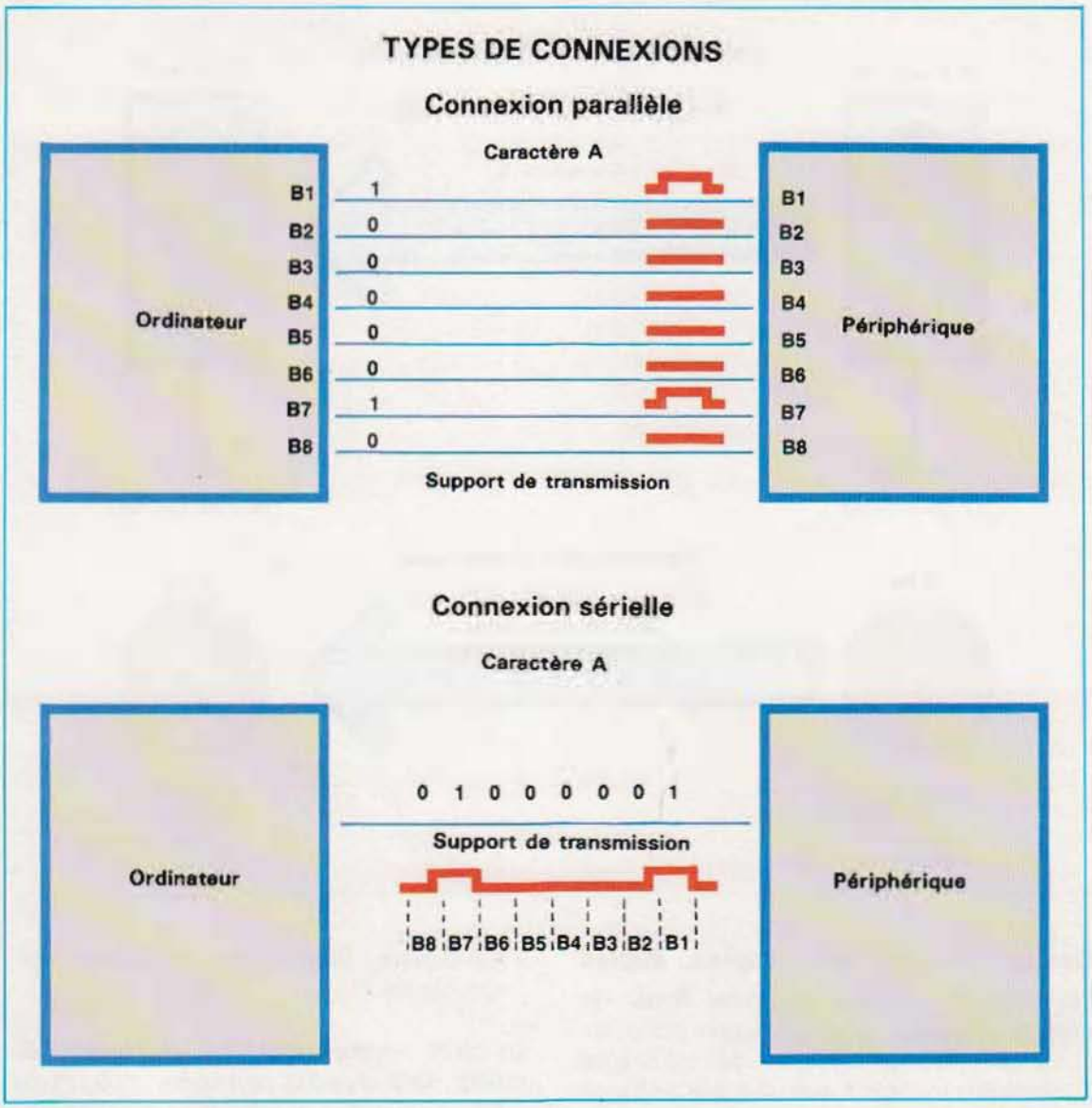
- **connexion parallèle**
  La ligne est composée de 8 fils distincts sur lesquels «voyagent» simultanément les 8 bits composant l'information.
- **connexion sérielle** (ou type série)
  La ligne est composée d'un fil unique sur lequel les bits voyagent l'un après l'autre.

Ainsi, avec la connexion parallèle, on atteint des vitesses de transmission plus élevées. C'est pourquoi ce type de liaison est typique des périphériques rapides (unité disque, unité bande, imprimante rapide). Toutefois la distance entre ces unités et l'ordinateur est limitée (quelques mètres) et c'est une solution coûteuse. Si elle n'atteint pas des vitesses très élevées, la connexion sérielle permet néanmoins de raccorder des périphériques, même à de très grandes distances, en utilisant les lignes téléphoniques.

CODIFICATION EBCDIC DES CARACTERES SPECIAUX EMPLOYES EN TELETRANSMISSION

Caractère	Codification EBCDIC bit n° 8* 7 6 5 4 3 2 1	Valeur décimale	Valeur octale	Valeur héxadécimale
NUL	0 0 0 0 0 0 0 0	0	0	0
SOH	0 0 0 0 0 0 0 1	1	1	1
STX	0 0 0 0 0 0 1 0	2	2	2
ETX	0 0 0 0 0 0 1 1	3	3	3
PF	0 0 0 0 0 1 0 0	4	4	4
HT	0 0 0 0 0 1 0 1	5	5	5
LC	0 0 0 0 0 1 1 0	6	6	6
DEL	0 0 0 0 * 1 1 1	7	7	7
	0 0 0 1 0 0 0 0	8	10	8
RLF	0 0 0 1 0 0 0 1	9	11	9
SMM	0 0 0 1 0 1 0 0	10	12	A
VT	0 0 0 1 0 1 1 1	11	13	B
FF	0 0 0 1 1 1 0 0	12	14	C
CR	0 0 0 1 1 1 0 1	13	15	D
SO	0 0 0 1 1 1 1 0	14	16	E
SI	0 0 0 1 1 1 1 1	15	17	F
DLE	0 0 1 0 0 0 0 0	16	20	10
DC1	0 0 1 0 0 0 0 1	17	21	11
DC2	0 0 1 0 0 1 0	18	22	12
DC3	0 0 1 0 0 1 1	19	23	13
RES	0 0 1 0 1 0 0 0	20	24	14
NL	0 0 1 0 1 0 0 1	21	25	15
BS	0 0 1 0 1 1 0	22	26	16
IDL	0 0 1 0 1 1 1	23	27	17
CAN	0 0 1 1 0 0 0 0	24	30	18
EM	0 0 1 1 0 0 1	25	31	19
CC	0 0 1 1 0 1 0	26	32	1A
CU1	0 0 1 1 0 1 1	27	33	1B
IFS	0 0 1 1 1 0 0	28	34	1C
IGS	0 0 1 1 1 0 1	29	35	1D
IRS	0 0 1 1 1 1 0	30	36	1E
IUS	0 0 1 1 1 1 1	31	37	1F
DS	0 1 0 0 0 0 0	32	40	20
SOS	0 1 0 0 0 0 1	33	41	21
FS	0 1 0 0 0 1 0	34	42	22
	0 1 0 0 0 1 1	35	43	23
BYP	0 1 0 0 1 0 0	36	44	24
LF	0 1 0 0 1 0 1	37	45	25
EOB/ETB	0 1 0 0 1 1 0	38	46	26
PRE/ESC	0 1 0 0 1 1 1	39	47	27
	0 1 0 1 0 0 0	40	50	28
	0 1 0 1 0 0 1	41	51	29
SM	0 1 0 1 0 1 0	42	52	2A
CU2	0 1 0 1 0 1 1	43	53	2B
	0 1 0 1 1 0 0	44	54	2C
ENQ	0 1 0 1 1 0 1	45	55	2D
ACK	0 1 0 1 1 1 0	46	56	2E
BEL	0 1 0 1 1 1 1	47	57	2F
	0 1 1 0 0 0 0	48	60	30
	0 1 1 0 0 0 1	49	61	31
SYN	0 1 1 0 0 1 0	50	62	32
	0 1 1 0 0 1 1	51	63	33
PN	0 1 1 0 1 0 0	52	64	34
RS	0 1 1 0 1 0 1	53	65	35
UC	0 1 1 0 1 1 0	54	66	36
EOT	0 1 1 0 1 1 1	55	67	37
	0 1 1 1 0 0 0	56	70	38
	0 1 1 1 0 0 1	57	71	39
	0 1 1 1 0 1 0	58	72	3A
CU3	0 1 1 1 0 1 1	59	73	3B
DC4	0 1 1 1 1 0 0	60	74	3C
NAK	0 1 1 1 1 0 1	61	75	3D
	0 1 1 1 1 1 0	62	76	3E
SUB	0 1 1 1 1 1 1	63	77	3F

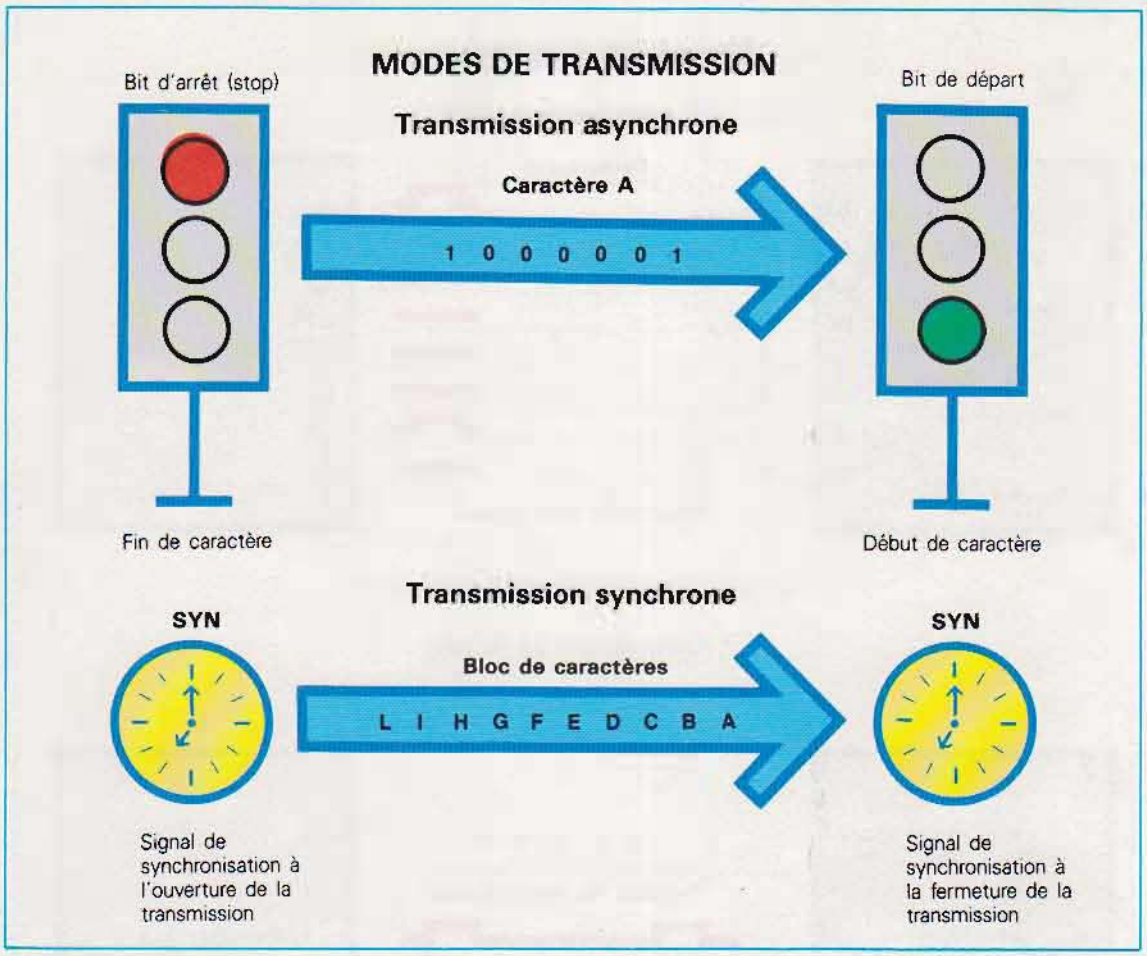
Le bit n° 8 est le bit de parité.



Par la suite, nous traiterons surtout de ce dernier point, étant donnée l'importance pratique considérable de la transmission de données sur réseau téléphonique (réseau «commuté»).
Examinons maintenant une connexion de type série et un autre concept important: le mode de transmission. La télétransmission de données est en général réalisée de deux manières (voir graphiques p. 1320).

- **transmission asynchrone**
- **transmission synchrone**

Le mode asynchrone est souvent défini par START-STOP, car les bits contenant l'information réelle (caractère) sont précédés et suivis par deux bits spéciaux: **le bit de start** (départ), qui prévient qu'un caractère est sur le point de partir et le **bit de stop,** qui signale que la transmission du caractère est achevée.
Le mode synchrone n'exige pas l'emploi de bits de start et de stop, mais il utilise des caractères de synchronisme positionnés en tête et en queue des blocs de caractères à transmettre. Ainsi, la réception d'un nombre de caractères de synchronisme donné informe le récepteur qu'un texte va commencer et qu'il se terminera par autant de caractères de synchronisme, qui viennent donc s'ajouter aux caractères spéciaux de fin de texte.

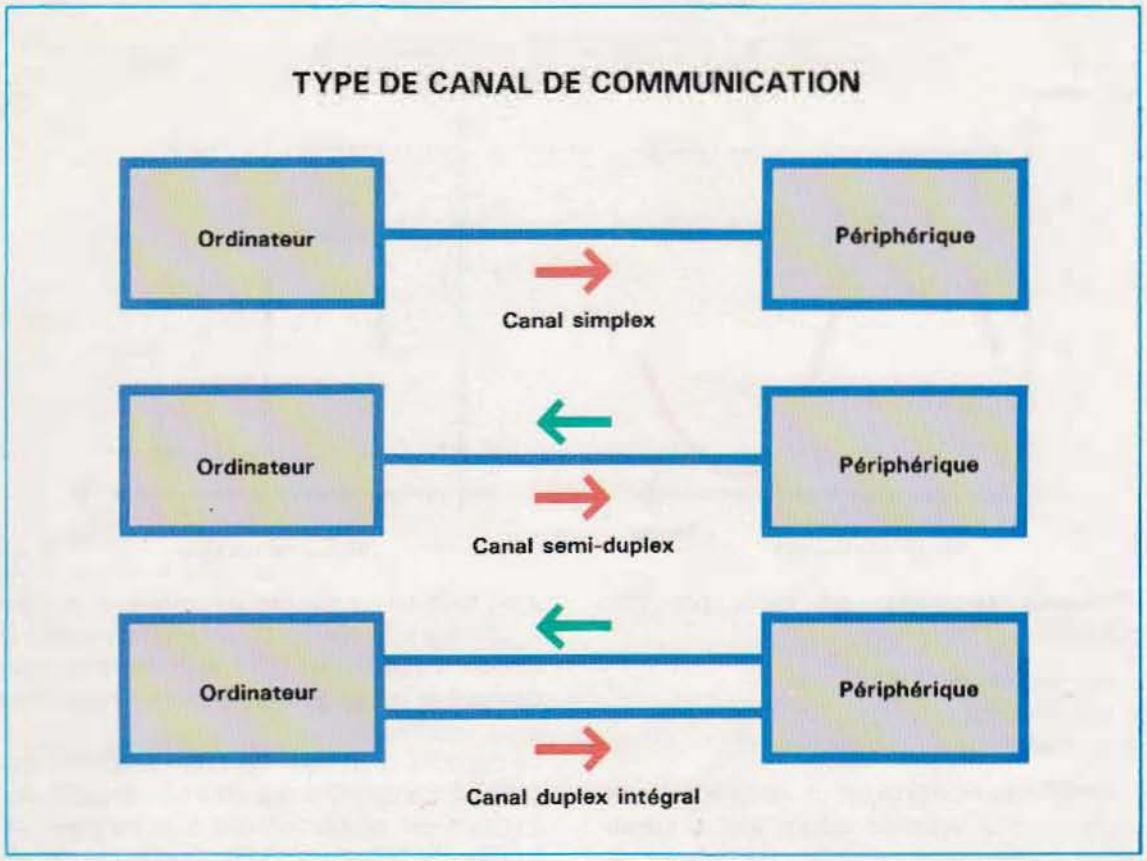


Canaux simplex, demi-duplex, duplex

Le dispositif physique s'appelle **ligne de communication;** si la connexion porte sur une courte distance (quelques mètres) la ligne coïncide physiquement avec le câble de liaison et avec les interfaces de l'ordinateur et du terminal. Pour des connexions éloignées, le mode le plus employé est la ligne téléphonique, avec des instruments spéciaux dont nous parlerons plus loin. Une fois la ligne déterminée, on définit le **canal** d'acheminement dans lequel voyage le flux d'information. Le concept de canal vient du fait que, normalement, dans une ligne, il est possible de faire voyager divers types d'informations et il faut donc en sélectionner le chemin.
Il existe trois types de canaux:

- **simplex** (transmission unidirectionnelle)
- **half-duplex** (transmission à l'alternat ou bidirectionnelle non simultanée)
- **full-duplex** (transmission bidirectionnelle simultanée).

Le canal simplex permet aux informations de passer dans un sens seulement: toujours de l'ordinateur vers le périphérique, ou à l'inverse s'il s'agit d'un périphérique d'entrée. Le canal simplex est en général peu utilisé, car presque toutes les liaisons ordinateur-périphérique ont besoin d'un flux dans les deux sens. Le canal simplex se compare à une voie à sens unique. Dans un canal semi-duplex, en revanche, les échanges d'information se font dans les deux sens, mais pas simultanément; cela signifie qu'à un moment donné, l'information ne circule que dans un sens, qui s'inverse à chaque fois que changent l'émetteur et le récepteur. Le canal semi-duplex est donc comparable à une route à sens unique alterné, gérée par deux feux qui se trouveraient situés aux deux extrémités de la voie.



Dans le **canal duplex** intégral (ou bidirectionnel simultané), les informations voyagent dans les deux sens et en simultané, c'est-à-dire sans qu'il soit nécessaire d'interrompre un sens de transmission. Ce n'est évidemment possible que si le canal est physiquement constitué de deux câbles, un pour chaque sens.
Une question se pose : quelle quantité d'informations est-il possible de transmettre, en une seconde, entre deux dispositifs reliés par un canal à une ligne de communication ?
La réponse dépend du type de liaison (série ou parallèle), des modes de communication (synchrone ou asynchrone) et du type de canal (simplex, semi-duplex ou duplex intégral).
L'unité de mesure de la vitesse de transmission est le **bit par seconde** (**bps**, en abrégé).
Voici quelques ordres de grandeur :

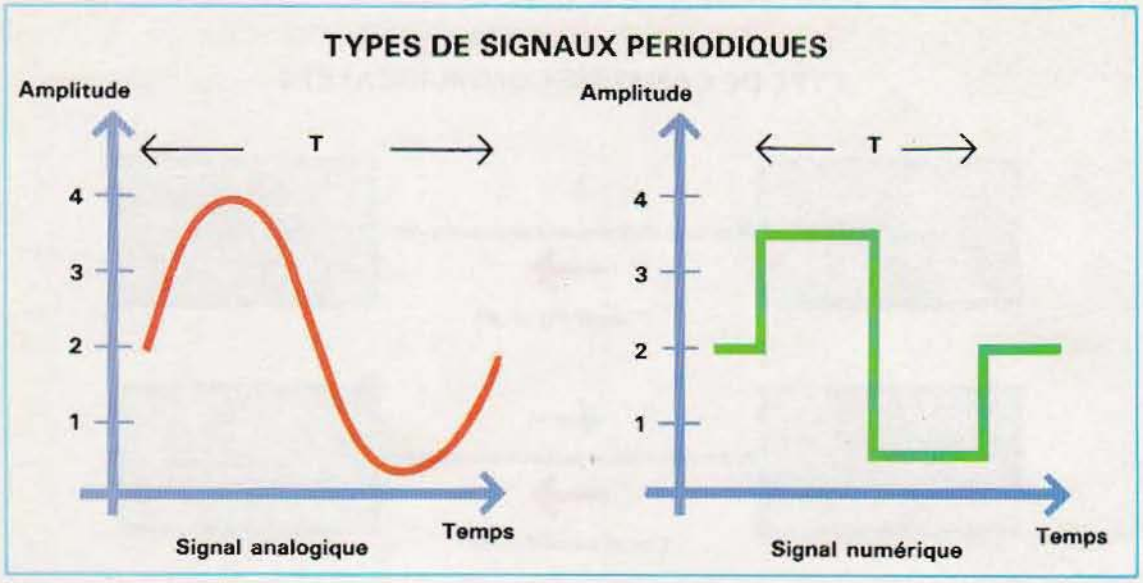
- connexions parallèles : jusqu'à quelques millions de bps
- connexions série : quelques centaines de milliers de bps en mode synchrone ; des dizaines de milliers en mode asynchrone.

Transmission par les lignes téléphoniques

Les liaisons examinées jusqu'ici correspondent à des distances faibles entre les ordinateurs et les périphériques, c'est-à-dire un parcours n'excédant pas 30 à 40 mètres. Au-delà, se posent quelques problèmes au niveau de la propagation des signaux, et il est alors nécessaire de recourir au réseau commuté.

Caractéristiques des signaux

Les signaux transmis par les télécommunications sont dit **périodiques** : ils se répètent tous les temps T (T est la période du signal et se mesure en secondes). Les signaux produits par les ordinateurs sont numérique (rectangulaires) tandis que ceux utilisés pour la transmission d'informations (par radio, systèmes vidéo, téléphone, etc.), sont de type analogique et varient dans le temps mais en continu.



Un signal périodique est défini par trois paramètres :

- son amplitude
- sa fréquence
- sa phase

L'**amplitude** du signal est sa valeur maximale. Le niveau d'un signal périodique analogique varie en continu pour reprendre les mêmes valeurs après chaque période T. A l'inverse, le niveau d'un signal périodique numérique ne varie pas en continu mais reste constant pendant un certain temps, puis change brusquement. Dans le schéma ci-dessus, le niveau du signal numérique est constant pendant T/2 à une valeur de 3,5 avant de tomber à 0,5.

La **fréquence** est le nombre de fois où le signal se répète en une seconde. Un signal qui se répète 5 fois en une seconde a donc une fréquence de 5 hertz. Sa **période** sera égale au 1/5e de seconde (la période est l'inverse de la fréquence).

La **phase** est le retard ou l'avance du signal par rapport à l'instant où commence l'observation ; elle se mesure en fractions de période. Dans le schéma ci-contre, le signal c est dit déphasé en avance, car il semble être parti un quart de période avant le début de l'observation des signaux a et b ; de même, le signal d est déphasé de T/2 en avance car il semble être parti une demi-période avant le repère d'observation.

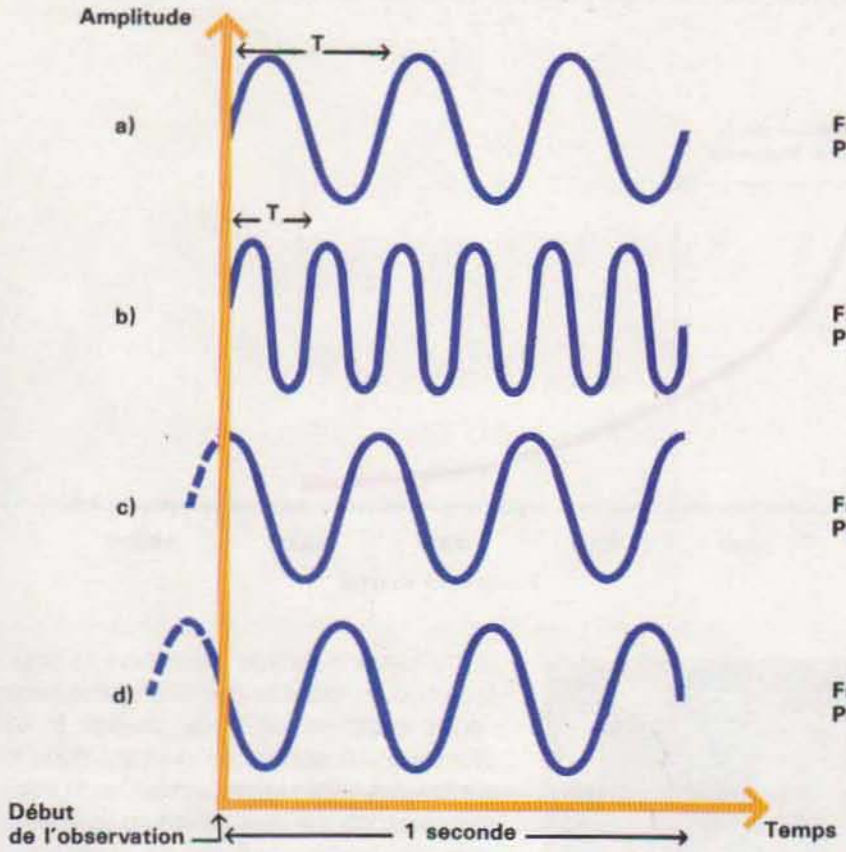
Un signal numérique qui se propage sur une ligne subit des déformations qui augmentent avec la distance parcourue, comme le montre le schéma en page 1323. On peut donc définir des distances limites (30 à 40 mètres) au-delà desquelles le signal n'est plus reconnaissable avec exactitude.

La capacité d'un ligne de communication (un câble) à transmettre des informations non déformées est proportionnelle à la **largeur de bande de fréquences** (ou bande passante). Cette expression désigne la différence, en hertz, entre la limite supérieure et la limite inférieure de fréquence pour lesquelles les signaux ne sont pas déformés. Dire, par exemple, que la bande d'une ligne va de 300 à 3 000 hertz signifie que les signaux périodiques ayant une fréquence comprise entre ces valeurs ne sont pas excessivement déformés. On peut alors affirmer, en extrapolant, que plus la bande d'une ligne est large, moins le signal est déformé, plus la distance de transmission peut être grande.

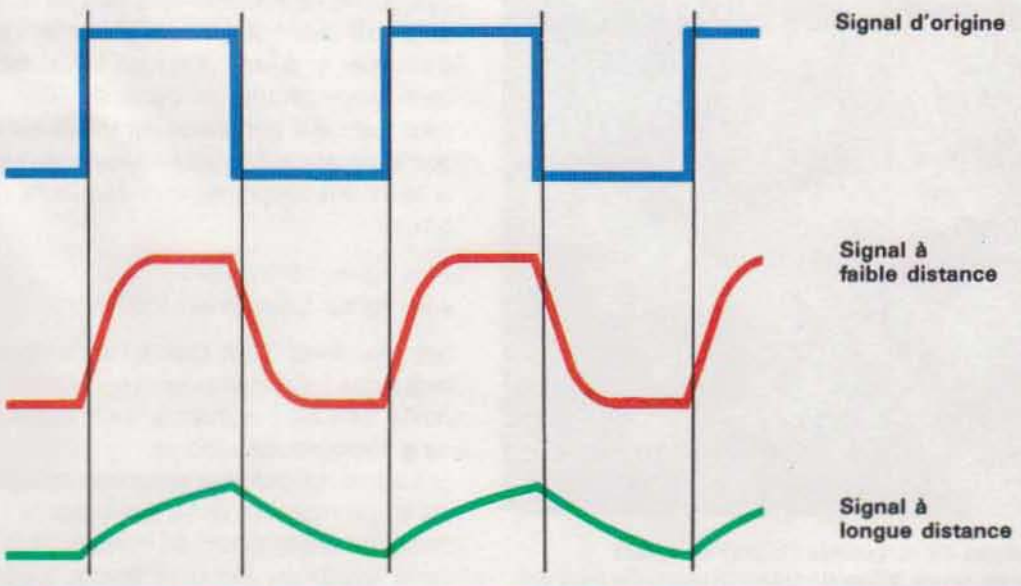
Le choix de techniques spéciales de fabrication des câbles (par exemple leur blindage) permet de décupler la longueur maximale d'une ligne de transmission de signaux numériques pour la porter à 300, voire 400 mètres. Il n'est toutefois pas possible, pour des raisons économiques, de relier de cette façon un ordinateur installé à Paris et un périphérique se trouvant à Lyon. Il faut recourir au réseau commuté.

La structure des lignes téléphoniques est optimisée en fonction des exigences de transmission de la voix humaine : des signaux d'une fréquence située entre 300 et 3 000 Hertz.

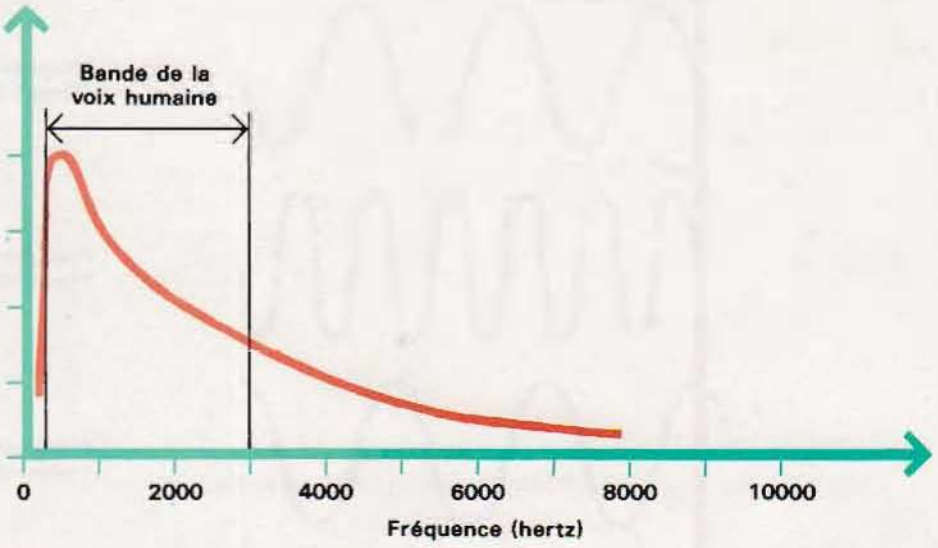
FREQUENCE ET PHASE DES SIGNAUX PERIODIQUES

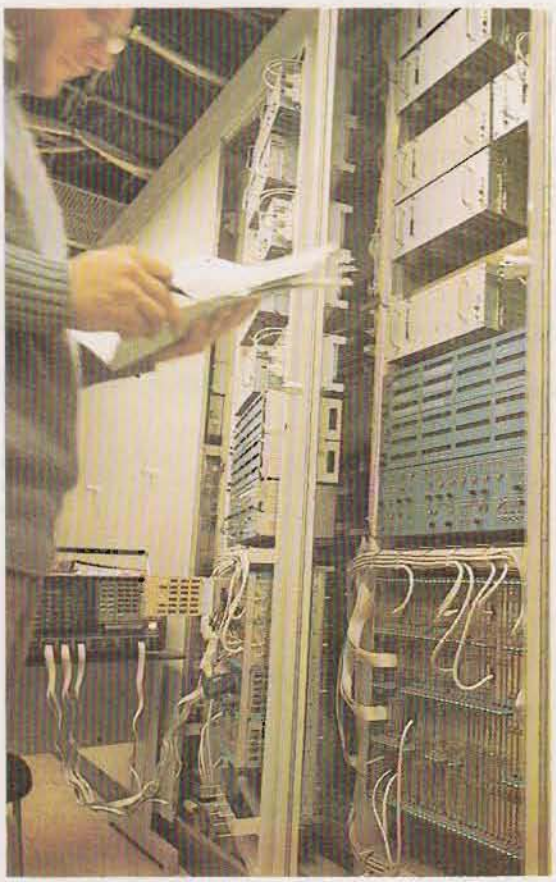


EFFET DEFORMANT (DISTORSION) DE LA DISTANCE SUR UN SIGNAL NUMERIQUE



ATTENUATION DE LA VOIX SUR UNE LIGNE TELEPHONIQUE





Italtel

Central électronique (Italie) assurant les services télex et transmission de donnée.

Cette bande passante est suffisante pour que la voix d'un individu soit clairement perçue à l'autre extrémité de la liaison (voir p. 1324). Les lignes téléphoniques peuvent donc transmettre des informations ; mais les signaux numériques, de par leur structure (différente de celle des signaux vocaux) ne pourraient parcourir de grandes distances sur ces lignes. Il faut donc transformer le signal digital en un signal analogique ayant une fréquence comprise dans la bande passante. Pour cela, il faut utiliser des instruments spéciaux de conversion du signal avant son passage sur la ligne téléphonique et à sa réception : les **modems.** Leur usage permet en outre d'établir des liaisons avec des périphériques situés à n'importe quelle distance. Les lignes téléphoniques, pour la télétransmission de données, sont de deux types :

- les lignes commutées
- les lignes spécialisées (ou louées)

Les premières sont des lignes téléphoniques ordinaires : on compose un numéro sur un poste, et c'est l'ordinateur avec lequel on veut communiquer qui répond.
Les lignes spécialisées assurent une liaison stable et permanente entre deux points ; il n'y a donc pas à composer de numéro et la fiabilité de la liaison est indubitablement supérieure à

celles commutées, qui subissent les perturbations de commutations normales.

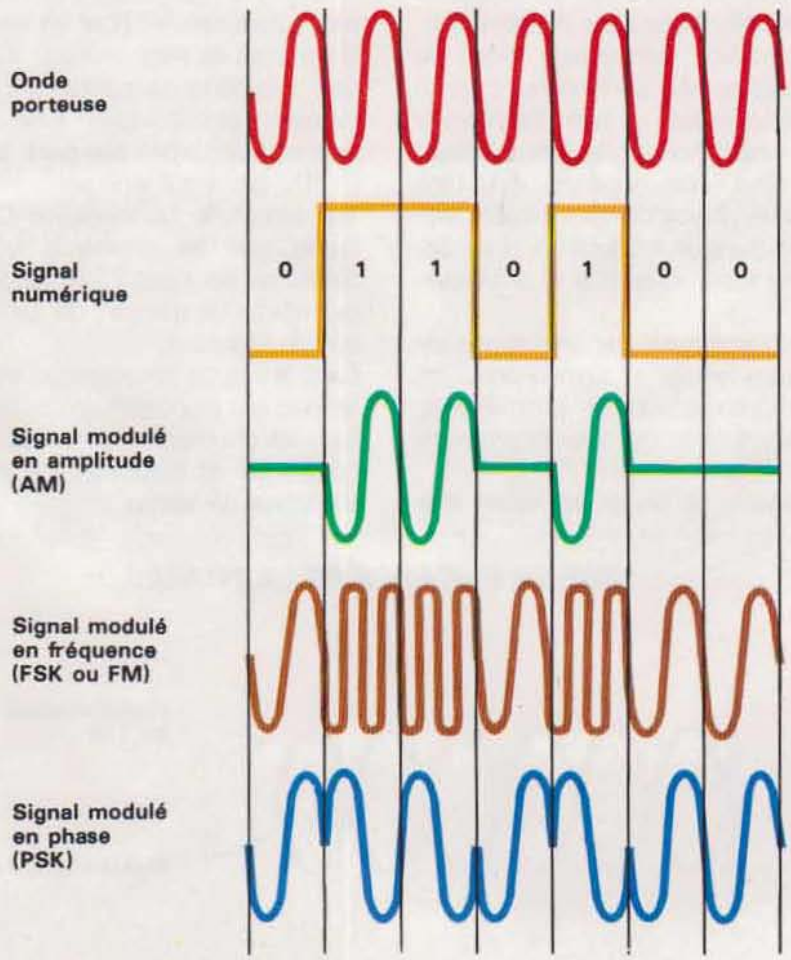
Techniques de modulation. Les modems

Les conversions numérique-analogique et analogique-numérique sont appelées modulation et démodulation et sont effectuées à l'aide de modems, contraction de MOdulateur et DEModulateur. La modulation transforme un signal numérique en un signal analogique dans la bande de fréquence d'une communication téléphonique (sans distorsion). La démodulation, elle, rétablit le signal numérique d'origine. Il existe plusieurs techniques de modulation (schémas ci-dessous) :

- modulation d'amplitude
- modulation de fréquence
- modulation de phase

Dans les trois cas, on fait appel à une onde de référence analogique dite **porteuse**, produite par le circuit interne du modem.
Avec la modulation d'amplitude (symbolisée par AM), la porteuse est représentée sur la ligne sous forme d'état logique 1 ou 0. La transmission consécutive de deux états logiques 1 entraîne la présence de l'onde porteuse sur la ligne pendant un temps plus long que la transmission d'un seul état 1. Ce phénomène implique que, pour pouvoir noter correctement les temps correspondant aux états logiques, il faut utiliser des circuits d'horloge.

TECHNIQUES DE MODULATION



La modulation de fréquence est généralement désignée par les abréviations FSK (Frequency Shift Keying — Saut de fréquence) ou FM (Frequency Modulation). Elle fait intervenir deux fréquences : celle de l'onde porteuse pour un état logique donné (par exemple l'état 0) ainsi qu'une fréquence multiple de la porteuse pour l'état opposé (l'état 1). L'amplitude du signal est la même pour les deux fréquences comme pour les deux états logiques représentés. Dans la technique de modulation de phase, ce sont l'amplitude et la fréquence du signal qui restent constants tandis que la phase est décalée de + ou - T/2 (voir ci-dessous l'exemple d'un signal de phase 0 et un autre déphasé de T/2).

La démodulation, c'est-à-dire la reconstitution du signal numérique d'origine, utilise des techniques différentes. Les modems contiennent généralement à la fois les circuits d'émission de signaux numériques reçus. Ils sont installés aux deux extrémités d'une ligne de communication et doivent être compatibles, sinon du même type. Cela signifie qu'ils doivent fonctionner à la même vitesse et selon les mêmes techniques de modulation et de démodulation. Il existe, en effet, des appareils qui, bien qu'ayant la même vitesse de transmission, utilisent des techniques de modulation et de démodulation différentes. Ils sont alors dits **équivalents.**

Les modems varient aussi par leur mode de transmission : synchrones et asynchrones, les vitesses de communication les plus élevées étant généralement obtenues avec la première technique.

Dernière différence : la bande de travail des modems. On trouve en effet les 3 types suivants, auxquels correspond la classification page ci-contre :

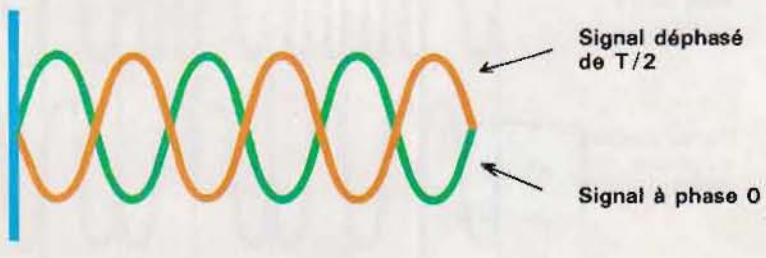
— modems opérant en-dessous de la bande vocale (jusqu'à 300 Hz). La vitesse de transmission ne dépasse pas 150 bps.
— modems opérant dans la bande vocale (300 à 3 000 Hz). La vitesse est de 1 200 à 2 400 bps pour les modems asynchrones et de 2 400 à 19 200 bps pour les modems synchrones.
— modems à large bande (au-dessus de 3 000 Hz). Si les vitesses sont supérieures, les techniques de modulation employées sont très perfectionnées et donc forcément coûteuses.

Configuration des liaisons

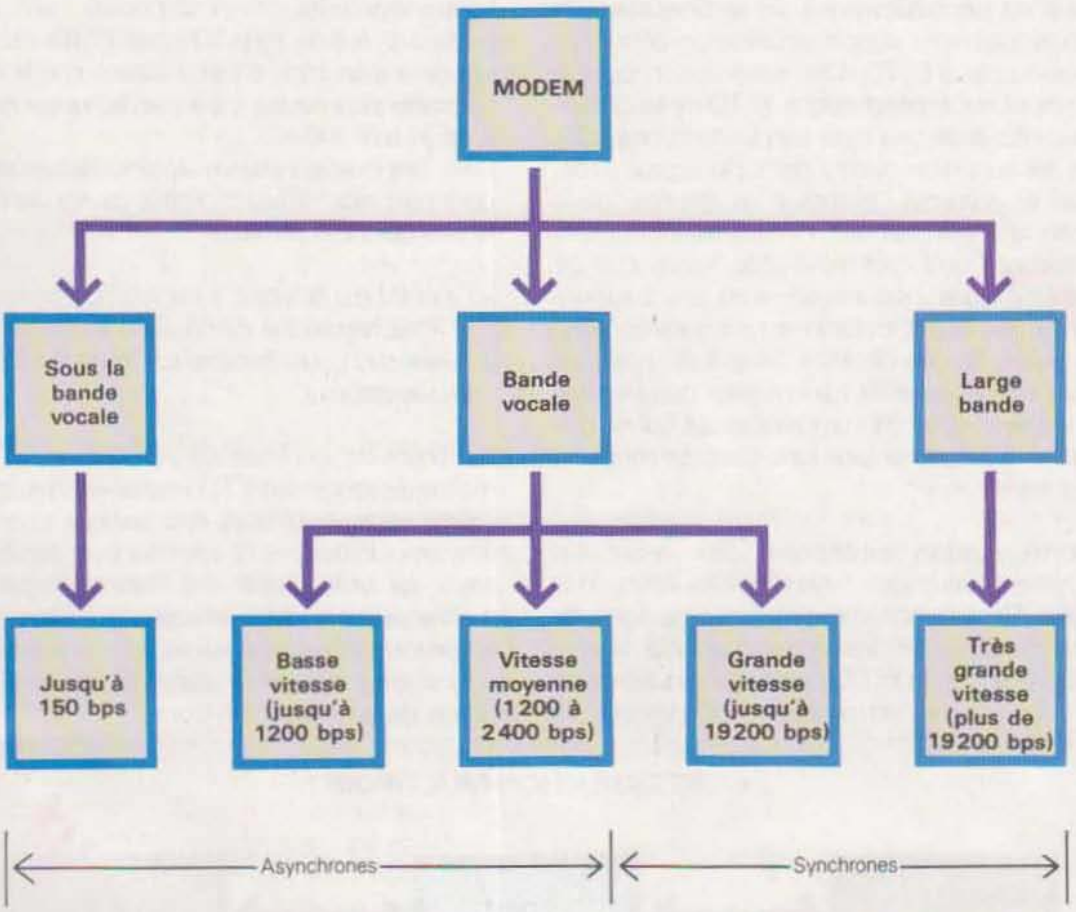
La classification typologique des liaisons que nous allons maintenant décrire est valable pour les courtes comme pour les longues distances, donc avec et sans modem. Cette généralisation nécessite toutefois l'introduction d'une certaine terminologie. Ainsi, les appareils connectables sont désignés par l'abréviation ETTD, qui signifie Equipement Terminal de Traitement de Données (ou DTE en anglais), tandis que les dispositifs utilisés pour les connecter sont des ETCD (DCE en anglais), ce qui signifie Equipement de Terminaison de Circuit de Données.

Dans le cas de communications à longues distances, ces dispositifs seront les modems et la ligne téléphonique : dans le cas de communications à faibles distances, il s'agira simplement de câbles de liaison.

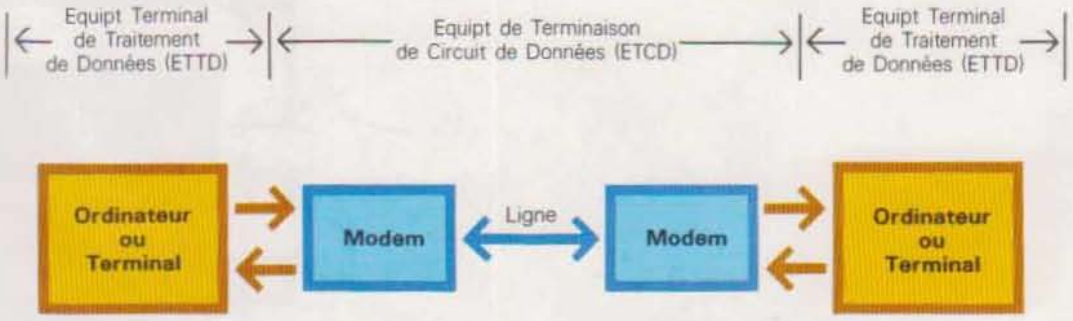
MISE EN EVIDENCE DE LA PHASE



CLASSIFICATION DES MODEMS



CONFIGURATION POINT A POINT



Configuration point à point. On retrouvera cette terminologie dans le graphique en bas de la page 1327, qui représente une configuration dite point à point, c'est-à-dire la **liaison directe d'un périphérique à un ordinateur.** Elle utilise une ligne de communication pour chaque couple d'ETTD. Une autre liaison point à point entre un périphérique ETTD et un ordinateur nécessite une ligne supplémentaire ETCD. La configuration point à point est simple à réaliser et présente l'avantage de disposer de la ligne en permanence : l'inconvénient est évidemment qu'il faut autant de lignes que de périphériques à connecter, d'où une augmentation des coûts, notamment dans les communications longue distance. Mais il est possible, pour des vitesses de transmission peu élevées, d'utiliser des lignes commutées (ce qui ne dispensera pas de prévoir une paire de modems par ligne).

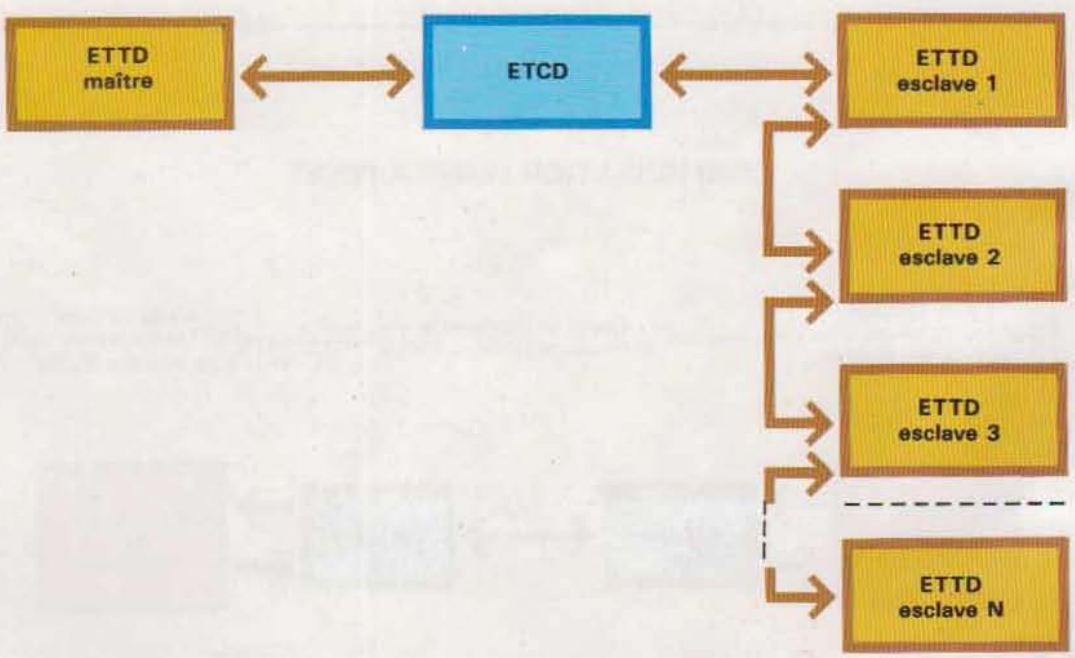
Configuration multipoint. Son atout est d'ordre économique : elle réduit les coûts de la ligne. Dans le schéma ci-dessous, la ligne de communication et les modems ont été regroupés dans le bloc ETCD tandis que les éléments à connecter — terminaux ou ordinateurs — sont dans les blocs ETTD. A une extrémité de la ligne se trouve un type particulier d'ETTD, l'**ETTD-maître**, qui commande le dialogue sur la ligne. En position d'**ETTD-esclaves**, à l'autre extrémité, les N dispositifs sont tous reliés à la même ligne. Chaque ETTD esclave est doté d'un code d'identification que le maître utilise pour savoir qui transmet ou qui reçoit à un instant donné.

Dans une configuration mulpoint, l'échange est géré par deux tâches directes, toutes deux pilotées par l'ETTD-maître :

- l'invitation à émettre, ou scrutation (polling) : la ligne reçoit une demande d'émission,
- la sélection : un terminal est invité à recevoir un message.

Au cours de la phase de scrutation, l'ETTD-maître interroge les ETTD-esclaves l'un après l'autre pour savoir s'ils ont quelque chose à émettre. Un esclave ne peut envoyer son message qu'après avoir été interrogé par le maître ; s'il n'a rien à envoyer, le maître pose la question à l'esclave suivant jusqu'au dernier, puis il recommence à partir du premier. La phase de scrutation est donc cyclique.

CONFIGURATION MULTIPOINT





Commodore

Configuration complète à base de Commodore 64.

Un esclave émetteur est reconnu grâce à son code d'identification, inclus dans le message émis. En phase de sélection, l'ETTD-maître adresse un esclave en l'informant qu'un message lui a été envoyé.

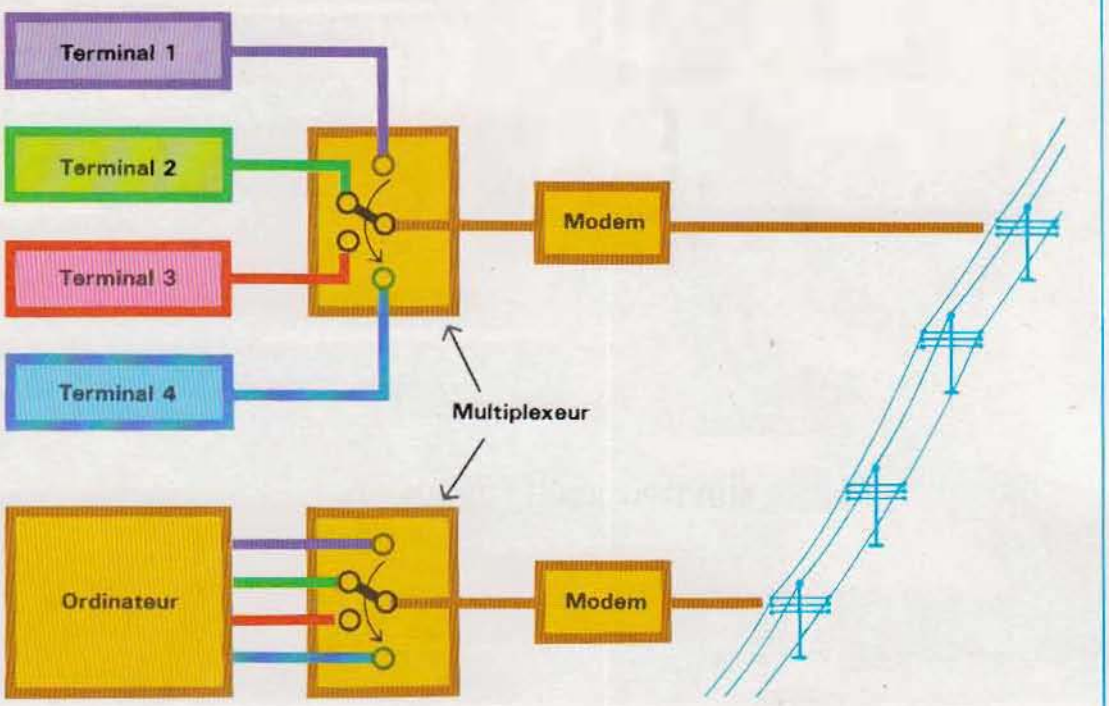
Prenons un exemple dans lequel le maître est un ordinateur et les esclaves N terminaux. Ordinateur et terminaux sont dotés d'interfaces pour gérer l'échange en mode multipoint. De la machine part un câble (abstraction faite du modem) auquel sont connectés tous les périphériques (une seule ligne de communication). L'unité centrale effectue cycliquement la tâche de scrutation au cours de laquelle les terminaux recoivent, l'un après l'autre, l'autorisation d'émettre.

Un terminal émetteur (ou un opérateur en saisie de données au clavier) est identifié par son code tandis que l'ordinateur accuse réception du message qui lui est envoyé.

Imaginons que ce dernier doive être affiché sur un autre terminal appartenant à la même chaîne multipoint. L'ordinateur déclenche alors la phase de sélection et envoie sur la ligne le contenu de la mémoire-tampon (dans laquelle se trouve le code du dispositif-receveur). Le terminal destinataire reçoit alors les données et les affiches. Le dialogue entre ETTD n'est pas totalement asynchrone, ce qui pose des problèmes d'attente pendant la phase d'utilisation de la ligne. Dans notre exemple, un terminal ne peut émettre qu'une fois la sélection terminée. La ligne risque donc de devenir le « goulot d'étranglement » du système. Ce problème oblige à trouver un bon compromis entre le nombre d'ETTD à relier au système et la quantité d'informations à transmettre par unité de temps.

Configuration multiplex. Il s'agit là d'une autre réponse à la gestion de l'attente : la configuration à base de multiplexeur, tout en réduisant au minimum le nombre de lignes, règle la question du volume d'informations à transmettre. En effet, le multiplexeur est tout désigné pour les situations caractérisées par une **voie unique de données à fort débit** et par plusieurs ETTD interconnectés. C'est le cas, par exemple, d'un ordinateur devant communiquer avec 4 terminaux sur une seule ligne (schéma p. 1330).

CONFIGURATION A BASE DE MULTIPLEXEUR



Le multiplexeur est donc un dispositif qui évite la confusion entre les signaux d'origine différente présents sur la ligne et ce grâce à plusieurs méthodes.

Le multiplexeur temporel alloue un intervalle de temps fixe à chacun des terminaux émetteurs et reconstitue le message à l'autre bout de la ligne, selon la même périodicité. Dans notre exemple, le partage du temps est simple : à l'instant t_1, la ligne 1 envoie le caractère A, à l'instant t_2, la ligne 4 envoie le caractère B... (voir graphique 1 page ci-contre).

Le multiplexeur fréquentiel alloue une bande de fréquences à chaque canal de données. Le principe est celui de la transmission radio : différents émetteurs envoient simultanément des messages sur différentes fréquences tandis que le poste récepteur ne capte que l'émeteur sur lequel il est réglé (graphique central page ci-contre).

Dans notre exemple, chaque terminal dispose d'une certaine bande de fréquences d'émission et l'ordinateur à 4 points de synchronisation (réglage des fréquences à la réception).

Le multiplexeur temporel est comparable à un commutateur à grande vitesse : son débit dépend étroitement de ses caractéristiques physiques : vitesses de transmission (en bps) et nombre de portes disponibles. Par exemple, un multiplexeur à 38 400 bps et 16 portes peut faire communiquer 16 terminaux à 2 400 bauds* chacun. Si les terminaux sont reliés à distance, la ligne téléphonique et les modems employés doivent être capables de supporter les 38 400 bps.

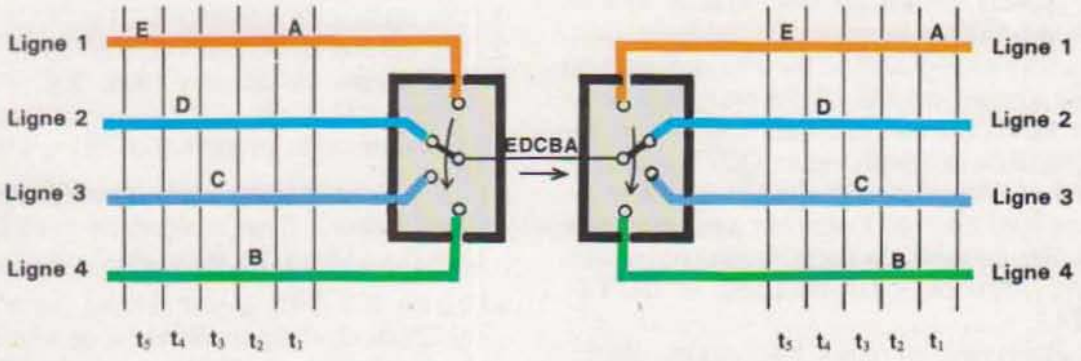
En revanche, la capacité de transmission d'un multiplexeur fréquentiel se mesure en termes de bande fréquence totale : plus la bande est large, plus le nombre de dispositifs pouvant communiquer sera élevé.

Interfaces de communication

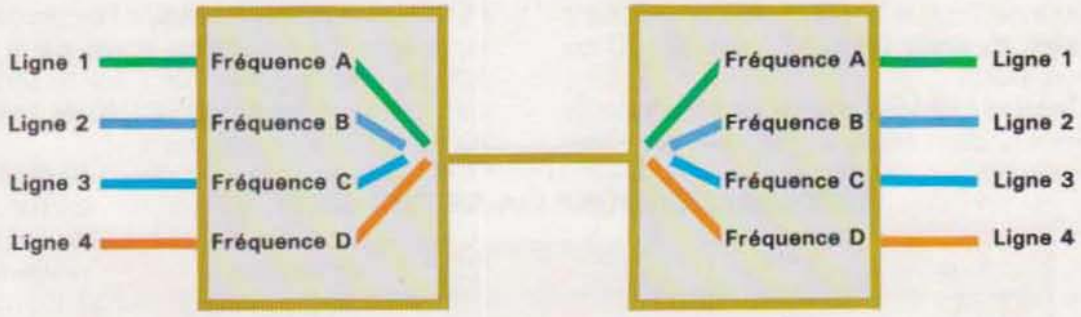
Il nous faut compléter la terminologie ETTD/ETCD par le concept d'**interface**. En télétransmission, ce terme désigne un dispositif assurant la liaison entre ETTD et ETCD

*Le baud est l'unité de vitesse de modulation. Elle est identique à la vitesse de transmission en bits par seconde quand la liaison est en binaire à deux niveaux. Dérivé de Baudot (nom d'un pionnier de l'informatique).

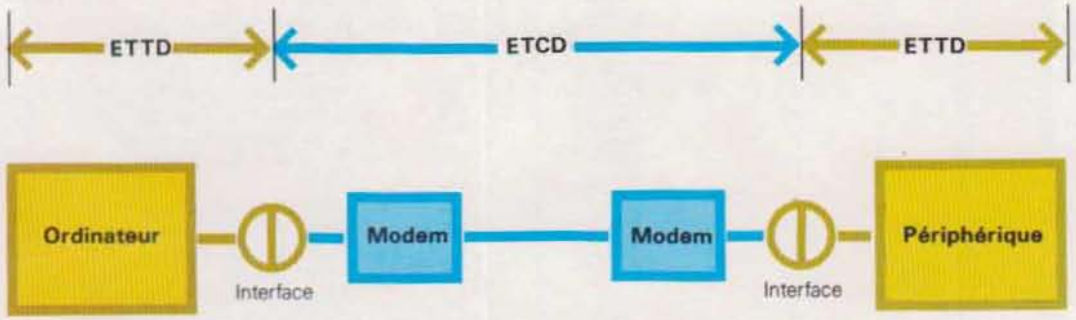
MULTIPLEXEUR TEMPOREL



MULTIPLEXEUR FREQUENTIEL



LIAISON POINT A POINT AVEC INDICATION DES INTERFACES



(schéma du bas, page 1331). Les interfaces de communication servent surtout à rendre compatibles les connexions; pour cela, elles doivent respecter des recommandations précises sur les aspects mécaniques, électriques et structurels des liaisons.

Les normes régissant la transmission avec ou sans modem ont été émises pour deux organismes : le Comité Consultatif International Télégraphique et Téléphonique (CCITT) pour l'Europe et l'Association des Industries Electroniques (EIA) pour les Etats-Unis. Les protocoles les plus courants de transmission par modem sont référencés : **EIA-RS232C** et **CCITT-V24.**

Ils établissent les règles mécaniques, électriques et de structure des circuits à respecter par les constructeurs.

S'agissant des circuits, ces normes régissent notamment la gestion des signaux le long de la ligne de communication. Le tableau ci-dessous donne la nomenclature officielle des circuits établie par l'EIA et le CCITT, tout en précisant le sens du signal (de l'ETTD vers l'ETCD ou inversement).

Le recours à certains signaux est lié au type de communication établie (synchrone, asynchrone, semi-duplex, ou duplex intégral) et donc au type de modem. Les signaux se subdivisent en 4 groupes ; nous verrons ensuite les plus fréquents :

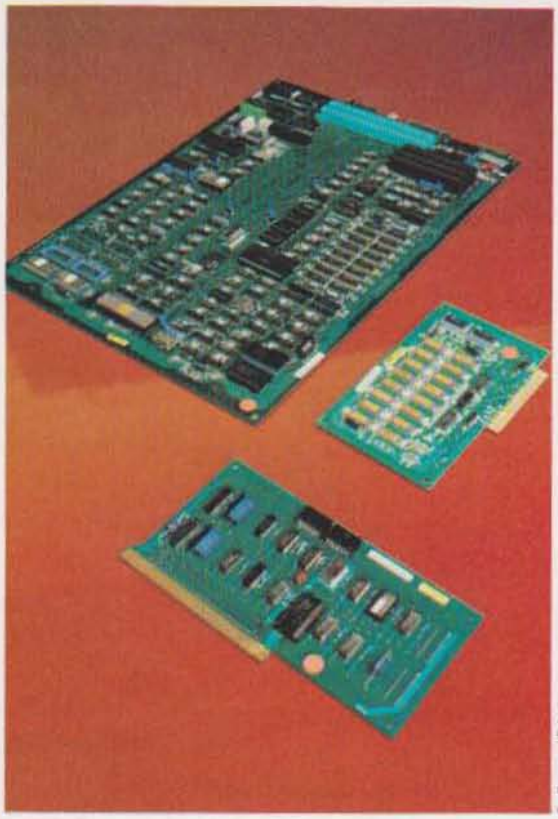
1 / Les signaux de référence (101 et 102)
2 / Les signaux de données (103 et 104)
3 / Les signaux de contrôle (105, 106, 107, 108.2, 109, 125)
4/ Les signaux de temporisation (114, 115)

- Circuit 101 (AA) : Protective Ground (terre de protection). C'est le signal de mise à la terre des châssis des appareils.
- Circuits 102 (AB): Signal Ground (terre de signalisation). C'est la référence de tous les signaux émis par un ETTD ou un ETCD (à l'exclusion du 101). Les circuits 101 et 102 sont parfois interconnectés pour réduire les perturbations.
- Circuit 103 (BA) : Transmitted Data (Emission de données de l'ETTD vers l'ETCD). Il supporte l'émission des données sur la ligne. Ce circuit n'est activé que si les 105 (Request to Send), 106 (Clear to Send), 107 (Data Set Ready) et 108.2 (Data Terminal Ready) le sont également.

RECOMMANDATIONS EIA-RS232C CCITT-V24

Broche	Vers ETTD	Vers ETCD	Circuit RS232C	Circuit V24	Description
1			AA	101	Protective Ground
2	→		BA	103	Transmitted Data
3	←		BB	104	Received Data
4	→		CA	105	Request to Send
5	←		CB	106	Clear to Send
6	←		CC	107	Data Set Ready
7	↔		AB	102	Signal Ground
8	←		CF	109	Detector for Signal Line Received (Carrier Detector)
9			—	—	
10			—	—	
11			—	—	
12	←		SCF	122	Detector for Secondary Signal Line Received
13	←		SCB	121	Secondary Clear to Send
14	→		SBA	118	Secondary Transmitted Data
15	←		DB	114	Transmission of Signal Timing
16	←		SBB	119	Secondary Received Data
17	←		DD	115	Receive of Signal Timing
18			—	—	
19	→		SCA	120	Secondary Request to Send
20	→		CD	108.2	Data Terminal Ready
21	←		CG	110	Signal Quality Detector
22	←		CE	125	Ring Indicator
23	→		CH/CI	111/112	Selector of Data Signal Rate
24	→		DA	113	Transmission of Signal Timing
25			—	—	

- Circuit 104 (BB) : Received Data (réception par l'ETTD de données en provenance de l'ETCD). Supporte les données à envoyer à l'ETTD. Il est ouvert (ON) ou fermé (OFF) en fonction de l'état du circuit 109 (Carrier Detector).
- Circuit 105 (CA) : Request to Send (demande d'émission d'ETTD vers ETCD). Indique que l'ETTD désire transmettre et prépare en conséquence l'ETCD (le modem émet l'onde porteuse).
- Circuit 106 (CB) : Clear to Send (prêt à émettre — signal émis par l'ETCD à destination de l'ETTD). Indique que l'ETCD est prêt à transmettre. Le signal est activé par l'ETCD à réception de la demande d'émission. En pratique, le 106 suit le 105 avec un décalage dû au retard initial nécessaire à la « préparation » de l'ETCD.
- Circuit 107 (CC) : Data Set Ready (poste de données prêt — signal émis par l'ETCD vers l'ETTD). L'état ON (ouvert) indique que l'ETCD est relié à la ligne, prêt à émettre ou à recevoir. Pour que des signaux soient émis sur la ligne, il faut que le 107 soit ON. En fait, le seul signal qui puisse opérer sans que le 107 soit ouvert est le 125 (Ring indicator).
- Circuit 108.2 (CD) : Data Terminal Ready (terminal de données prêt — signal émis par l'ETTD vers l'ETCD). A l'ouverture de ce circuit, l'ETTD informe l'ETCD qu'il est prêt à émettre et à recevoir. Dès que l'ETTD fait passer le 108 à ON, l'ETCD fait de même avec le 107. Les deux signaux émis par ces circuits suivant donc le même schèma d'activation à quelques intervalles près.
- Circuit 109 (CF) : Carrier Detector (détecteur du signal de ligne reçu sur la voie de données — émis par l'ETCD vers l'ETTD). Le niveau du signal sur la ligne est conforme aux spécifications. Si le circuit passe à l'état OFF, la réception est interrompue.
- Circuit 125 (CE) : Ring Indicator (indicateur d'appel — signal émis par l'ETCD vers l'ETTD). A l'état ouvert, il signale l'arrivée d'un appel et établit alors la liaison.
- Circuit 114 (DB) : Transmission of Signal Timing (base de temps pour les éléments du signal à l'émission — signal émis par l'ETCD à destination de l'ETTD). Utilisé par l'ETCD (modem) pour temporiser l'ETTD (ordinateur ou terminal à l'émission.
- Circuit 115 (DD) : Receive of Signal Timing (base de temps pour les éléments du signal à la réceptions — signal émis par l'ETCD vers l'ETTD). Utilisé par l'ETCD (modem) pour temporiser l'ETTD à la réception. L'état OFF du 115 dépend de l'état du 109 : si ce dernier est OFF, le circuit 115 l'est aussi.



C. Pozzoni/Marka

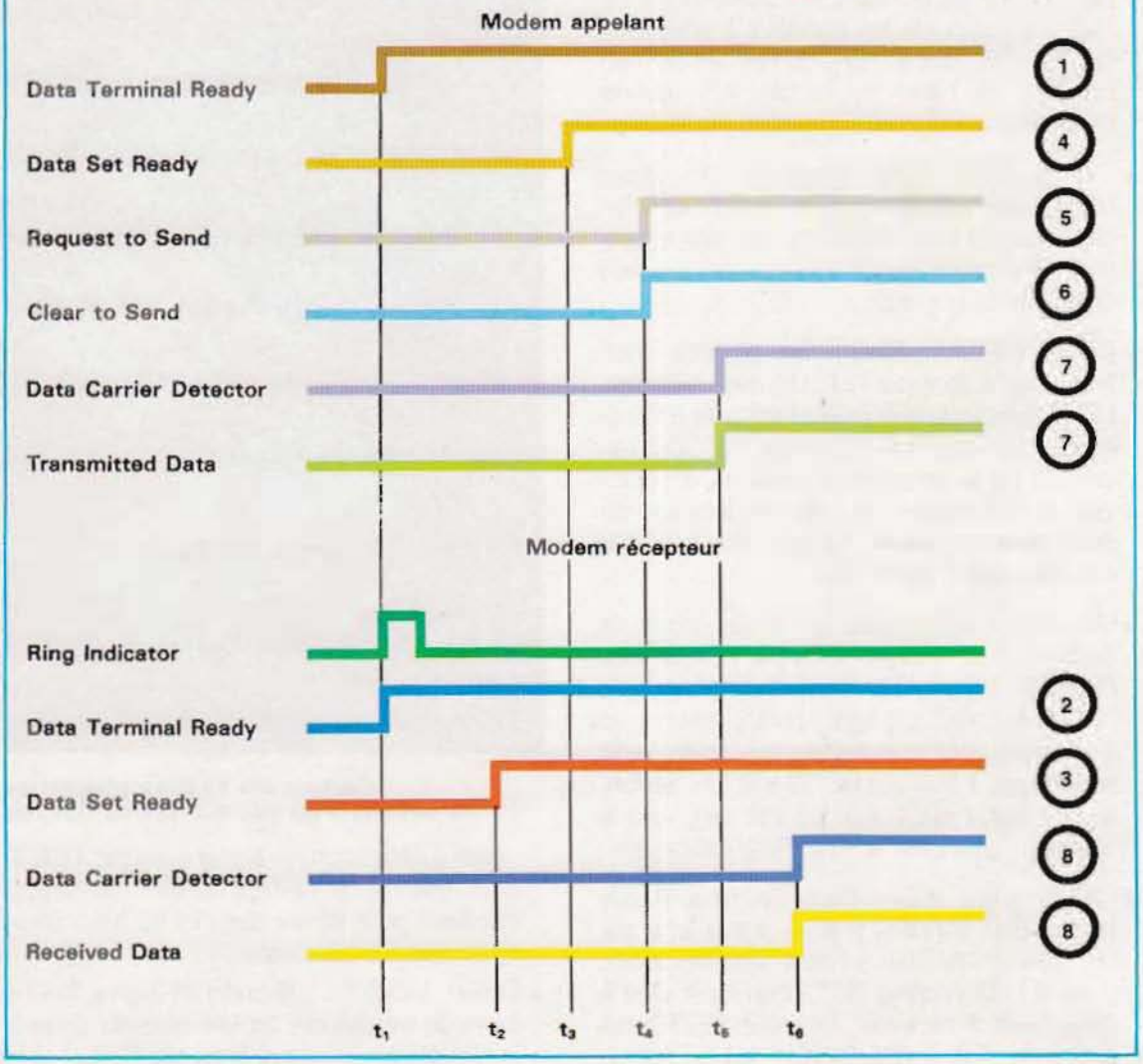
Carte-mère et deux plaquettes de circuits d'un micro-système Olivetti.

Pour illustrer l'utilisation de ces signaux, examinons la séquence de transmission de données entre deux modems à distance installés sur une ligne téléphonique (voir p. 1334).

1 / Le signal Data Terminal Ready est actif du côté du modem appelant et un opérateur compose le numéro de téléphone du modem récepteur.

2 / Le modem récepteur active le Ring Indicator. Le signal Data Terminal Ready est également actif du côté réception, donc l'ETCD connecté est prêt (terminal en marche).

LIAISON ET TRANSMISSION DE DONNEES ENTRE DEUX MODEMS



3 / A l'instant t_2, le modem récepteur active le Data Set Ready et l'opérateur appelant reçoit un signal acoustique indiquant que la liaison est établie.

4 / L'opérateur appelant actionne un bouton du modem et raccroche. Il active ainsi le Data Set Ready de son côté, ce qui indique que le modem appelant est également prêt (instant t_3).

5 / L'ETTD appelant active le signal Request to Send pour informer le modem qu'il veut commencer à émettre.

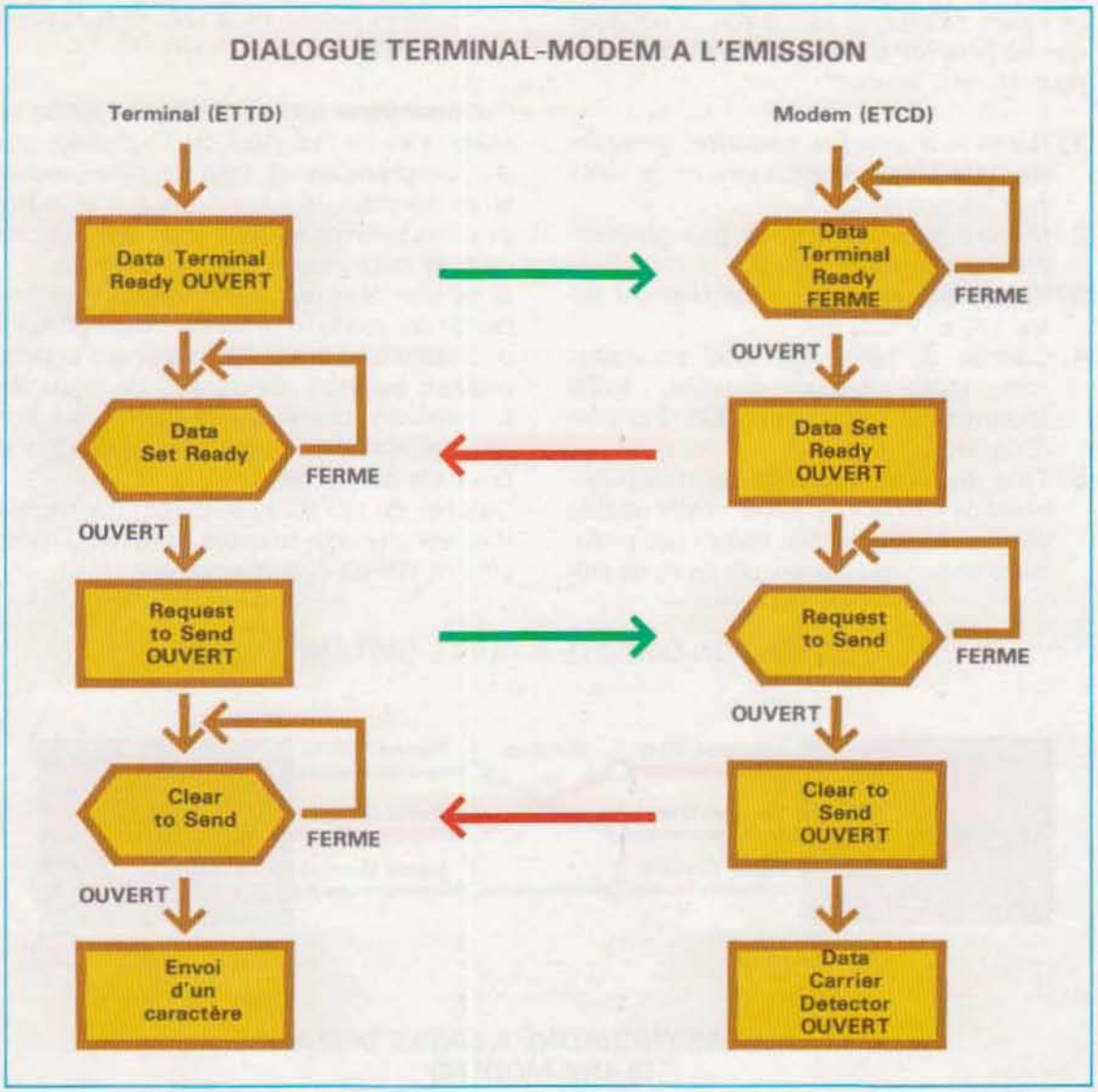
6 / Le modem active le signal Clear to Send pour informer l'ETTD que tout est prêt pour l'émission (instant t_4).

7 / L'émission commence (instant t_5) ; le signal Carrier Detector est actif pour contrôler le niveau des informations transmises.

8 / Les données arrivent au point de réception (t_6) ; le Carrier Detector est également actif du côté de l'ETTD récepteur.

Dans cet exemple, les intervalles de temps sont donnés à titre indicatif et ne tiennent pas compte des véritables caractéristiques des modems et des lignes.

L'organigramme du dialogue entre le terminal émetteur (ETTD) et le modem (ETCD) (voir page ci-contre) fait apparaître les relations de cause à effet entre les différences phases de la session.



Communications à faibles distances

La standardisation des interfaces de communication ne prend pas en compte la distance de transmission. Les normes de l'EIA et du CCITT restent donc valables pour les liaisons sans modem. Les constructeurs incluent eux-mêmes, dans le matériel de l'ordinateur, les interfaces nécessaires à la communication avec l'extérieur selon les normes internationales. Il en est de même des périphériques. Le problème qui se pose est généralement le suivant : un terminal peut-il dialoguer sans modem avec un ordinateur à N interfaces capables de piloter N dispositifs aux normes EIA ou CCITT ?

Supposons que la communication à établir soit du type asynchrone. Si l'interface de l'ordinateur a été conçue pour ne travailler que sur de faibles distances (jusqu'à 15 mètres d'après la norme série RS232C), il n'en sortira généralement que trois fils : **Transmitted Data**, **Received Data**, **Signal Ground**.

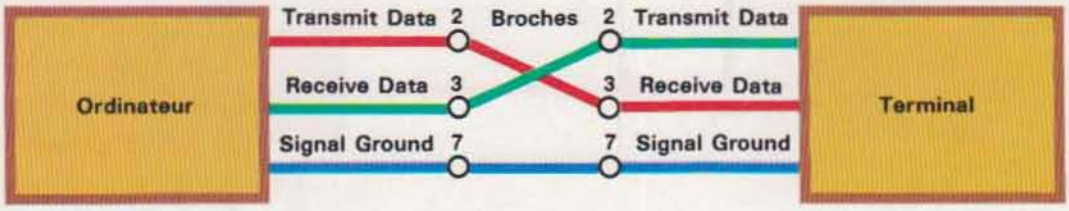
Si l'interface du terminal a été conçue selon les mêmes spécifications, la liaison est simple : il suffit de relier le circuit Transmitted Data de l'ordinateur à celui du Received Data du terminal et vice-versa, comme indiqué page 1336. Dans ce cas (schéma central), aucun des autres signaux prévus par les recommandations de l'EIA n'est utilisé.

La liaison fonctionne telle quelle, à condition que les paramètres suivants soient identiques pour les deux dispositifs :

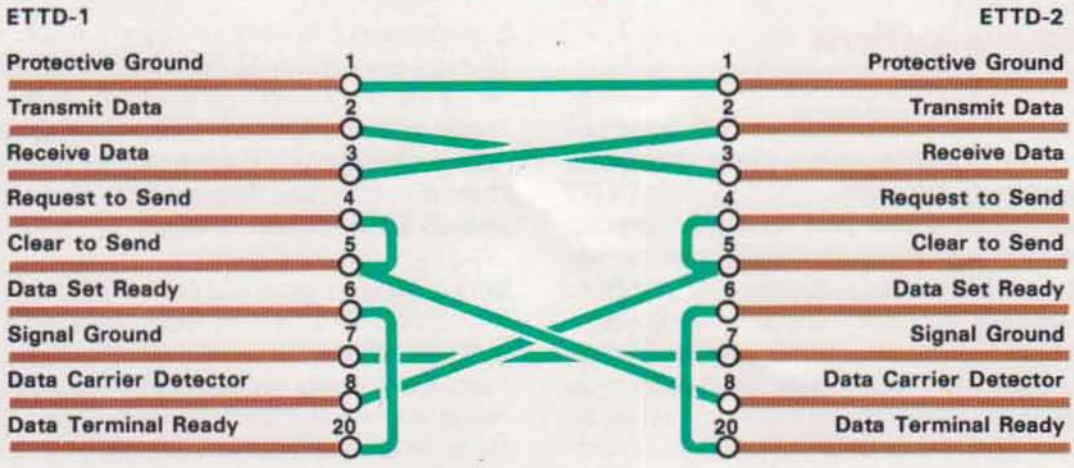
1 / Nombre de bits par caractère, généralement de 7 ou 8, avec ou sans bit de parité (voir tableau p. 1317).
2 / Nombre de bits de stop. On peut généralement choisir entre 1 ou 2.
3 / Vitesse de transmission, normalement entre 110 et 9 600 bps.
4 / Contrôle de parité. Les trois possibilités sont NONE (aucun contrôle), EVEN (contrôle de parité), et ODD (contrôle d'imparité).
5 / Type de synchronisation ou d'établissement de liaison. Ces points seront étudiés plus loin dans la section traitant des protocoles de communication. Les types de protocole les plus courants sont XON/XOFF et ENQ/ACK.

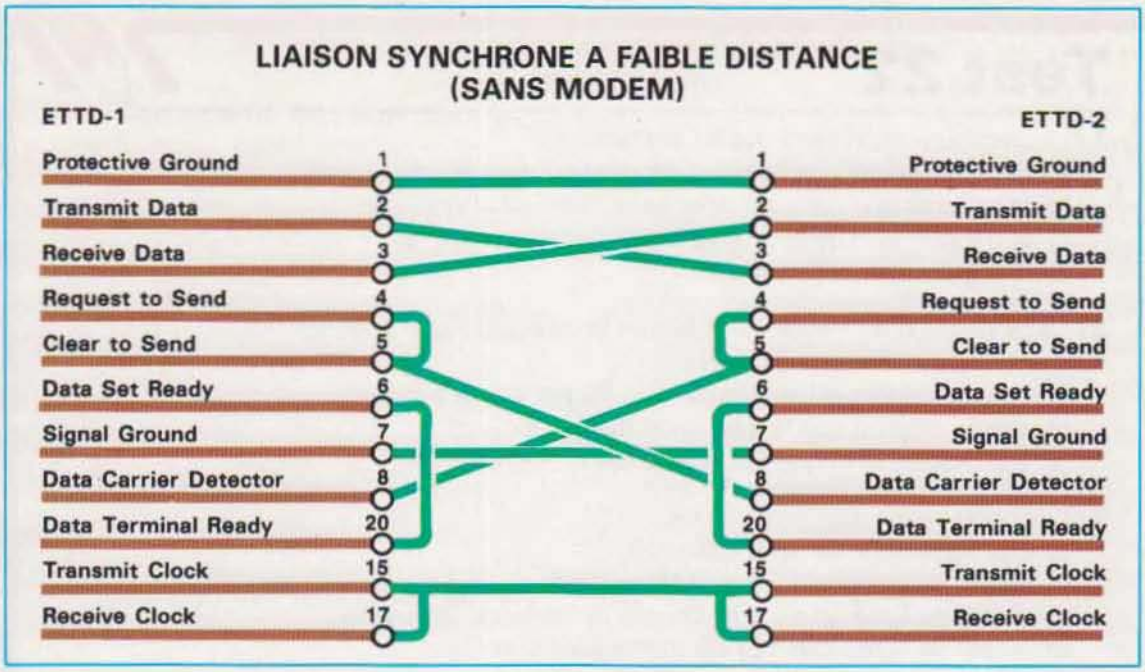
Ces paramètres sont généralement définis au niveau tant de l'interface de l'ordinateur que des périphériques. Il peut toutefois arriver qu'un périphérique soit « rigide », c'est-à-dire avec des paramètres fixes. Il faut alors adapter ceux de l'ordinateur, qui est plus souple.
Si les interfaces de communication série emploient des modems, la liaison à faible distance est établie à l'aide de câbles spéciaux appelés **modem by-pass** (éliminateurs de modems). Le schéma du bas de page présente une liaison de type asynchrone en faisant apparaître le brochage des circuits.
Du côté de l'ETTD-1, le circuit Transmitted Data est connecté au circuit Received Data du côté de l'ETTD-2, et inversement.

LIAISON DIRECTE A FAIBLE DISTANCE



LIAISON ASYNCHRONE A FAIBLE DISTANCE (SANS MODEM)





La transmission n'a lieu que si les signaux Request to Send, Clear to Send, Data Set Ready et Data Terminal Ready sont actifs ; les signaux Request to Send et Data Terminal Ready sont activés par l'ETTD (ordinateur ou terminal), tandis que Data Set Ready et Clear to Send le sont par le modem (qui n'est pas représenté).

Les connexions entre Request to Send/Clear to Send et Data Terminal Ready/Data Set Ready permettent d'activer les quatre signaux nécessaires au lancement de la transmission (autrement dit, celui qui commence la communication se répond tout seul). Pour la réception, le signal Carrier Detector doit être activé, ce qui s'obtient en reliant le circuit Request to Send de l'émetteur au circuit Carrier Detector du récepteur (autrement dit, celui qui désire émettre informe le récepteur qu'il y a une porteuse). La connexion des circuits de mise à la terre (Protective Ground) peut être éliminée si l'on ne veut pas connecter la masse de l'ETTD-1 à celle de l'ETTD-2.

Le schéma ci-dessus représente la liaison à faible distance en mode synchrone : les connexions supplémentaires seront celles de la base de temps du signal à l'émission Transmit Clock (ou Transmit of Signal Timing) et de la base de temps du signal à la réception Receive Clock (ou Receive of Signal Timing). Le signal Transmit Clock est utilisé par le modem pour temporiser l'ETTD à l'émission, tandis que Receive Clock pour l'ETTD à la réception. Les interfaces synchrones de l'ordinateur et du terminal possèdent généralement des circuits d'horloge connectés aux broches 15 et 17.

La temporisation correcte est obtenue en reliant ces broches et en établissant la connexion 15-15 entre l'ETTD-1 et l'ETTD-2. Si les interfaces de l'ordinateur et du terminal sont dépourvues de séquenceurs, une horloge externe, pour l'envoi des impulsions dans la broche 15 (ou 17) d'un des deux côtés, est alors nécessaire.

Protocoles de communication

Maintenant, il s'agit d'examiner ce que l'on appelle les protocoles, une sorte de « **grammaire** » pour que les dispositifs échangent des informations. Les protocoles sont donc, à un niveau plus élevé, les gestionnaires des informations et, tout comme les interfaces, ils sont le fait des constructeurs d'ordinateurs et des organismes internationaux de normalisation.

Le terme protocole recouvre, le plus souvent, à la fois les aspects du matériel et du logiciel. Le premier est constitué des circuits conçus spécialement pour le protocole adopté, le second se compose de programmes de mise en œuvre des règles du protocole.

Test 22

1 / La communication semi-duplex permet :
- a) la transmission simultanée d'information dans les deux sens
- b) la transmission dans les deux sens, mais en alternance
- c) la transmission dans un seul sens
- d) la duplication d'informations

2 / La modulation de fréquence présente la caractéristique suivante :
- a) elle ne modifie pas la porteuse
- b) elle ne modifie ni l'amplitude ni la fréquence du signal
- c) elle ne modifie que la fréquence du signal
- d) elle modifie la fréquence et la phase du signal

3 / La configuration multipoint utilise :
- a) une seule ligne de communication
- b) autant de ligne qu'il y a d'ETTD esclaves
- c) autant de ligne qu'il y a de phases de balayage (scrutation)
- d) autant de lignes que l'ETTD maître peut gérer

4 / Le sigle EIA-RS232C désigne :
- a) les modems asynchrones
- b) une norme de connexion
- c) une norme de modulation
- d) une norme relative aux interfaces de communication série

5 / Le signal Data Set Ready indique :
- a) que le modem est prêt à émettre
- b) que le terminal est prêt à émettre
- c) qu'il y a une porteuse sur la ligne
- d) que le modem a reçu une demande d'émission

6 / Si le signal Receive Data est actif, le signal suivant doit également être actif :
- a) Signal Ground
- b) Transmit Clock
- c) Carrier Detector
- d) Transmit Data

7 / Quel signal indique que l'ETTD est prêt aussi bien à émettre qu'à recevoir ?
- a) Clear to Send
- b) Data Terminal Ready
- c) Request to Send
- d) Carrier Detector

8 / Le signal Transmit Clock sert :
- a) à synchroniser l'ETTD et l'ETCD
- b) à synchroniser le modem
- c) à synchroniser l'ETTD à l'émission
- d) à synchroniser les caractères émis

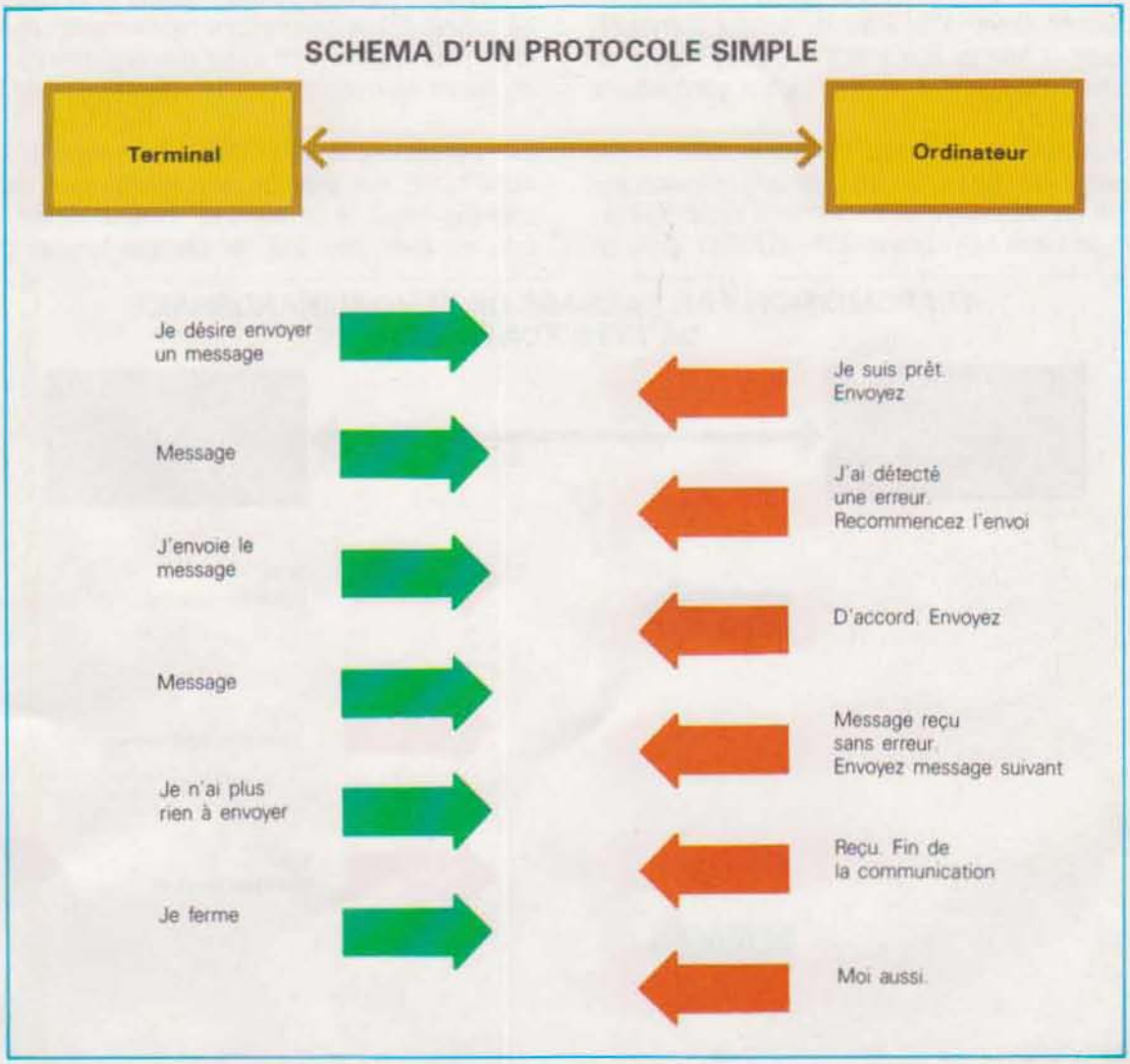
Voir les solutions du test en page 1342

Le « handshake »

Les échanges ont besoin d'être synchronisés. C'est le rôle procédure appelée « handshake » (transmission avec passage de témoin) qui permet à deux transmissions séquentielles, non synchronisées mais mutuellement dépendantes, de se dérouler. Elle ne doit pas être confondue avec le mode de transmission (synchrone ou asynchrone), son objet étant de garantir l'intégrité des messages transmis. Si, par exemple, un ordinateur doit communiquer avec un terminal, le handshake garantit que le terminal affichera toutes les données émises par l'ordinateur, indépendamment de la vitesse de transmission. Les informations échangées au cours du handshake constituent un dialogue du type :

- Message prêt à être transmis
- Signal indiquant que l'unité est prête à recevoir le texte
- Signal de début d'émission du texte
- Acceptation ou renvoi du texte
- Détection d'erreurs dans le texte reçu
- Eventuelle retransmission du texte
- Fin du texte

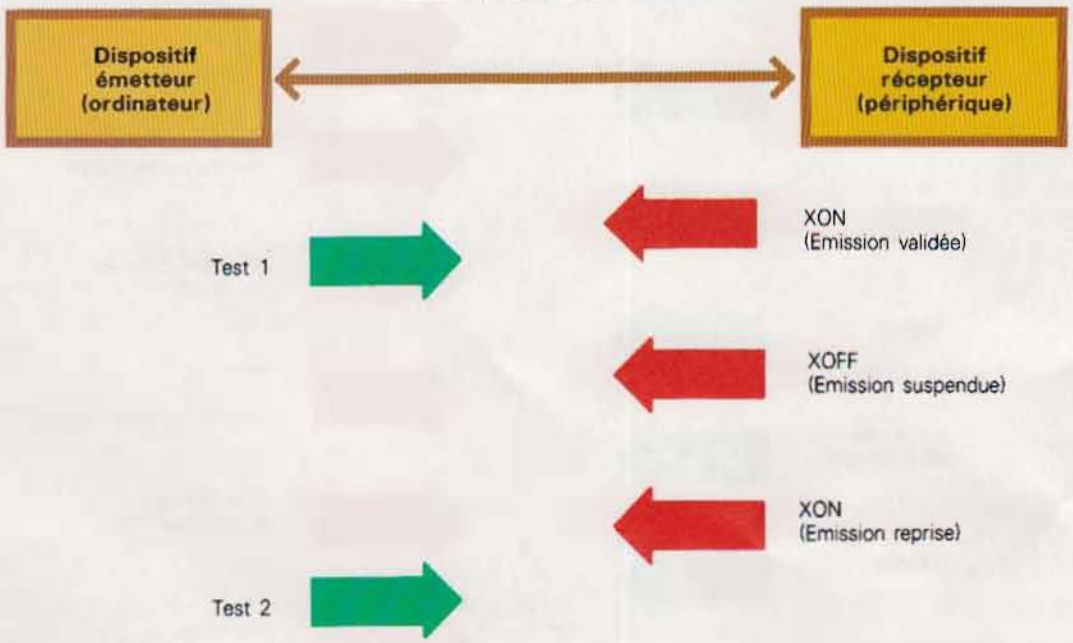
La séquence du protocole représenté ci-dessous montre un terminal informant l'ordinateur qu'il a un message à lui transmettre. L'ordinateur identifie le terminal et active la transmission. A la réception, si l'ordinateur détecte une erreur, il demande au terminal d'émettre à nouveau le texte. La deuxième fois, le message reçu ne contient pas d'erreur et l'ordinateur invite à continuer avec une autre communica-

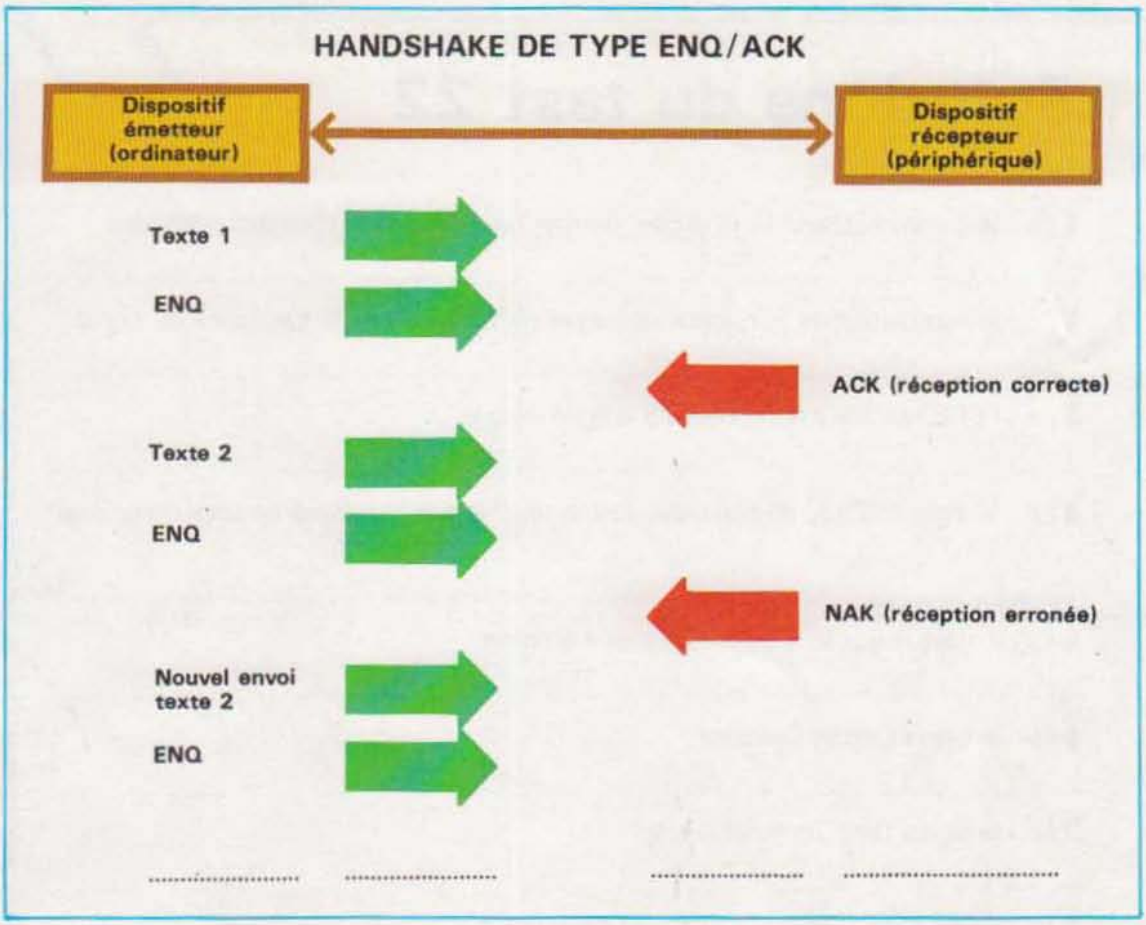


tion. Le terminal n'a pas d'autre message à envoyer et en informe l'ordinateur. Si, à son tour, l'ordinateur a un texte à envoyer au terminal, il déclenche le processus.
N'ayant rien à transmettre, il met fin à la communication, et le terminal en fait autant.
Il apparaît donc que les lignes de communication gérées par des protocoles ne voient pas transiter seulement les données à transmettre, mais aussi des informations supplémentaires servant à certifier qu'aucune donnée n'a été perdue. Ces informations constituent **l'en-tête** du protocole et en même temps un critère de jugement du protocole : un protocole est efficace si son en-tête est courte, parce que cela signifie qu'il réussit à garantir l'intégrité des informations en ajoutant un minimum de données, ce qui équivaut à une perte de temps minimale. Comme il y a plusieurs modes d'interaction entre deux stations devant communiquer, il existe également plusieurs types de «handshake», dont les principaux sont décrits ci-après.
Voici un exemple, pour clarifier le rôle joué par cette procédure. Supposons que la liaison entre un ordinateur et un terminal vidéo soit du type série asynchrone (EIA-RS232C) avec un débit de transmission de 9 600 bauds. A cette vitesse, près de 960 caractères voyagent sur la ligne en une seconde, et le terminal ne peut les visualiser au fur et à mesure. Il faut donc qu'à des intervalles réguliers, le périphérique demande à l'ordinateur d'interrompre la transmission pour qu'il ait le temps de visualiser et, d'une manière générale, de traiter les caractères reçus. Les terminaux du type imprimantes, dérouleurs de bande magnétique,... procèdent de même. Pour remédier à l'inconvénient représenté par la suspension de la transmission, on a recours à des mémoires tampons incorporées aux terminaux. Chaque périphérique peut ainsi stocker des informations avant leur visualisation. Durant la communication, les données envoyées par l'ordinateur sont provisoirement rangées dans le tampon du terminal et ensuite prélevées pour être traitées. Quand le tampon se sature, il faut interrompre momentanément la communication pour éviter que des données ne soient perdues.

La méthode XON/XOFF. La procédure XON/XOFF est précisément fondée sur ce principe (schéma ci-dessous). Quand le tampon est plein aux 2/3, le terminal envoie à

TRANSMISSION PAR PASSAGE DE TEMOIN (HANDSHAKE) DE TYPE XON/XOFF





l'ordinateur les caractères DC3 (voir tableau des caractères ASCII). Cela signifie Transmit OFF (ou XOFF) : l'ordinateur doit cesser d'émettre. Le terminal visualise ensuite les données stockées dans son tampon et le vide ; quand le tampon n'est plus qu'à 1/3 de sa capacité (autrement dit quand il a été vidé d'un tiers), le terminal envoie à l'ordinateur les caractères DC1, (Transmit ON, ou XON) : le terminal peut reprendre l'émission. Les valeurs données (1/3 et 2/3) sont variables, en fonction du terminal à gérer.

D'autres périphériques utilisent la même méthode de « handshake », mais en employant non plus DC1 et DC3 mais l'état haut ou bas d'un signal RS232C, par exemple Clear to Send ou Data Terminal Ready. Ce dernier cas constitue la forme de liaison la plus élémentaire et entraîne la gestion d'autres signaux liés à Transmit Data et à Receive Data.

La méthode ENQ/ACK. Une autre forme d'interaction est fondée sur la méthode **Enquire/Acknowledge**, ou ENQ/ACK, schématisée ci-dessus. Cette méthode facilite la demande de retransmission du texte en cas de détection d'erreurs par le récepteur. Le dispositif émetteur envoie le texte suivi des caractères ENQ (interrogation) ; le dispositif récepteur traite le texte et envoie le caractère ACK (accusé de réception positif) s'il n'y a pas de problème, ou le caractère NAK (accusé de réception négatif) s'il a repéré son erreur. L'émetteur envoie un autre texte dans le premier cas, ou à nouveau le même texte s'il a reçu NAK.

La méthode ENQ/ACK contourne le problème de la saturation du tampon : en effet, d'une part les textes transmis ne dépassent pas des longueurs pré-établies et d'autre part c'est toujours le récepteur qui donne l'autorisation de poursuivre la transmission. Si le tampon est plein, il suffit de retarder l'envoi de cette autorisation. L'aspect interactif et synchronisation ne constitue qu'un des nombreux aspects de la phase de « handshake » d'un protocole.

Solutions du test 22

1 / b : la communication semi-duplex permet l'émission et la réception alternées

2 / c : la modulation de fréquence opère par modification de la fréquence du signal

3 / a : l'ETTD maître est connecté à la ligne unique

4 / d : le sigle RS232C désigne une norme relative aux interfaces de communication série

5 / a : il indique que le modem est prêt à émettre

6 / c : le signal Carrier Detector

7 / b : le signal Data Terminal Ready

8 / c : il sert à synchroniser l'ETTD à l'émission.

Clavier de l'ordinateur Commodore Plus 4



R. Villa/Marka

Un wargame pour professionnels

S'il est un domaine qui a joué un rôle moteur pour l'évolution de l'électronique, et notamment des ordinateurs, c'est bien celui de la simulation. Le concept de base de cette technique est dans la reproduction, de la manière la plus pratique pour l'utilisateur, d'un phénomène physique ou naturel, d'une machine ou d'un appareil complexes, de comportements humains, physiques ou intellectuels et, en général, de tout ensemble simulant une réalité.

Les exemples qui suivent montrent comment est né ce besoin de simulation, aujourd'hui indispensable, dans la résolution de problèmes complexes.

Dans l'étude des phénomènes physiques ou naturels, la simulation a ouvert la voie à de nouvelles théories en fournissant des outils d'études et de contrôle de modèles.

Il existe des programmes de simulation très perfectionnés, capables de prévoir l'évolution de situation complexes comme par exemple l'état des ressources de notre planète.

On peut, au moyen de quelques modifications dans les paramètres ou dans les programmes, vérifier en très peu de temps des hypothèses de développement, encore incontrôlables dans la réalité.

L'exemple le plus classique de simulation appliquée à des machines est celui du simulateur de vol pour les pilotes d'avions. Il s'agit d'une reproduction fidèle du cockpit d'un avion dans une cabine spéciale montée sur une plateforme mécanique pouvant mettre cette cabine dans la position qu'elle aurait en vol lors de manœuvres aériennes.

L'ensemble est connecté à un ordinateur qui actionne les commandes du pilote, les traite selon le modèle prévu, dirige les mouvements de la plate-forme et tous les «instruments» de bord. En fait, ne sont installés dans les simulateurs que les panneaux de commande et de contrôle, dont les touches, interrupteurs, manettes de commande, unités d'affichage, diodes électroluminescentes et indicateurs optiques et acoustiques sont reliés à l'ordinateur. Ce dispositif fournit au pilote toutes les informations qu'il recevrait vraiment au cours d'un vol réel.

Il existe même des simulateurs qui fournissent au pilote l'image (diurne ou nocturne) de ce qu'il verrait au décollage, en vol, et à l'atterrissage dans divers aéroports ou en survol d'un territoire.

Les avantages de la simulation, du point de vue de la sécurité, de l'économie et de la disponibilité sont évidents.

La formation d'un pilote à l'aide d'un simulateur constitue une formule très économique en temps et en argent, mais aussi en efficacité car elle ne dépend plus de la disponibilité physique d'un véritable avion, avec tous les risques, les coûts et les délais nécessaires à son entretien, à l'assistance au sol et en vol, aux consommations de carburant... Dans les centres de formation au sol, le pilote commence sa formation et effectue des décollages et des atterrissages (même en catastrophe) dans n'importe quel territoire, et tout cela dans n'importe quel avion, sans avoir jamais réellement quitté la terre.

La forme la plus évoluée de simulation réunit des phénomènes physiques, des appareils et des comportements humains dans un ensemble où tous ces éléments sont en corrélation. L'ASTT (Action Speed Tactical Trainer, Formation tactique en temps réel) relève de cette dernière forme de simulation.

L'objectif de l'ASTT est de permettre à des QG de la Marine militaire de planifier et d'effectuer des exercices aéronavals, des plus simples aux plus complexes, en fournissant aux commandants et aux officiers des diverses unités tous les moyens nécessaires à leur formation tactique, tant au cours de l'exécution que du «playback» ultérieur des exercices.

Pour pouvoir être effectué dans la réalité, un exercice aéronaval complexe demande des mois de planification et de préparation de centaines de personnes (commandants, officiers et équipages de différentes unités, tout le personnel à terre) avant, pendant et après son déroulement. Il faut aussi préparer les unités (travaux en chantier, essais, ravitaillements, approvisionnements), transférer les unités de leur zone de stationnement ordinaire à celle choisie pour l'exercice, préparer le théâtre des opérations (contrôle des activités civiles, comme les vols d'avions de ligne ou privés, le transit d'embarcations de ligne, de bateaux de plaisance ou de pêche), et prévoir toute une série d'autres activités annexes.

De plus, un exercice réel est limité par la réalité même. Par exemple, lors d'un exercice opposant deux « adversaires », les différentes unités engagées se voient affecter à telle ou telle équipe. Or, il est clair que toutes ces unités, comme d'ailleurs les appareils et les systèmes de bord (détecteurs, armement, etc...) appartiennent à la Marine militaire.

D'autres limites sont imposées à l'emploi des armes, car il est évident que l'on ne peut pas ouvrir réellement le feu ou lancer des missiles contre ses propres unités.

Se pose ensuite le problème de l'analyse globale des résultats de l'exercice, dont l'objectif est d'examiner conjointement l'ensemble des activités de chaque unité afin de les mettre en rapport les unes avec les autres et d'obtenir un tableau récapitulatif des actions menées.

Il est ensuite possible d'en tirer les enseignements recherchés et les conclusions.

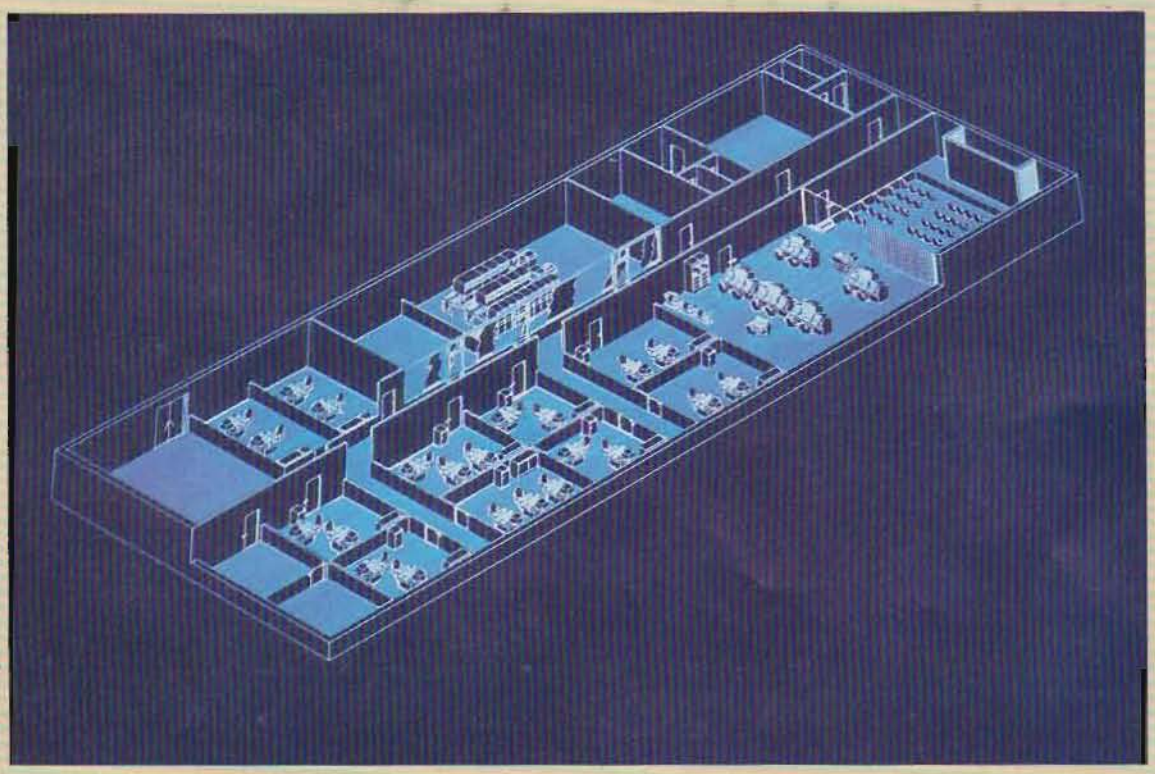
Il apparaît clairement qu'un exercice réel, s'il reste un instrument irremplaçable de contrôle du bon fonctionnement des structures, des unités, des appareils et de la formation des hommes, est loin de constituer la meilleure méthode en termes d'économie, de rapidité, de pratique, de souplesse et de répétabilité pour assurer la formation tactique des groupes de commande.

Toutes ces raisons ont fait naître le besoin de disposer d'un système de simulation pour fournir les moyens de se former sans se heurter aux inconvénients, parfois même aux dangers, d'exercices réels.

L'ASTT permet non seulement de répondre largement à ces besoins, mais aussi d'étudier, d'appliquer et de contrôler de nouvelles méthodes tactiques.

Il se compose de cinq sous-systèmes principaux répartis dans douze sites. Chacun des dix centres de commande (voir maquette ci-dessous) peut être considéré comme le centre de commande de n'importe quel type d'unité (navire, submersible, avion, hélicoptère). Les officiers exécutent leurs missions à l'aide de consoles, de moniteurs alphanumériques, d'unités vidéo graphiques et d'appareils à leur disposition pour gérer et contrôler toutes les fonctions de l'unité: navigation, détecteurs, armes, communications...

Centre de commande de l'ASTT.



B. Liotta/Il Dagherrotipo

Chaque groupe de commande travaille en temps réel à bord de la cabine comme s'il se trouvait dans son unité, en plein théâtre des opérations.

La salle de ***contrôle/auditorium*** *est équipée de cinq consoles à deux places qui permettent aux instructeurs de contrôler le déroulement des opérations et, le cas échéant, d'intervenir grâce aux messages-écrans, et aux images des manœuvres projetées en temps réel sur écran géant de 3 × 3 mètres.*

Des conclusions sont tirées et le point est fait dans cette même salle, avec possibilité de reprojection des opérations et de réécoute (en synchronisme avec les événements) de toutes les communications radio enregistrées.

La ***salle de calcul*** *est équipée, outre d'importants périphériques, de deux ordinateurs VAX 11/780 de 1 et de 1,5 méga-octets de mémoire centrale, avec une mémoire commune de 0,5 Mo pour l'échange de données, et enfin 82,5 Mo et 55 Mo de mémoire de masse, plus une unité à bande pour l'enregistrement des données sur les manœuvres.*

Le contrôle des ordinateurs et le déroulement des opérations de planification et de préparation des opérations sont réalisés par quatre terminaux, alors qu'une imprimante rapide fournit les données et les tableaux souhaités sur papier. On dispose également d'une table à digitaliser.

Cet équipement sert à introduire dans l'ordinateur le plus possible de données concernant le profil géographique des côtes représenté dans les cartes maritimes.

Celles-ci sont placées sur la table et l'opérateur suit avec le curseur le profil de la ligne côtière. Un programme spécial saisit alors les données de position, corrige les erreurs, calcule la corrélation avec les positions géographiques réelles et stocke enfin sur disque le profil des côtes ainsi généré.

Le sous-système de calcul est complété par un équipement spécial de générateurs pour la ***représentation vidéographique*** *ainsi que des interfaces nécessaires à l'établissement de liaisons avec les consoles, les écrans de contrôle, les claviers, c'est-à-dire avec tous les appareils de l'ASTT répartis dans les diverses salles.*

Le sous-système de contrôle comprend les ***consoles*** *pour instructeurs, l'****écran géant*** *et l'unité bande analogique à 8 pistes. Celle-ci est employée pour enregistrer jusqu'à six émissions-radio simultanées.*

Chacune occupe une piste différente ; vient ensuite le commentaire (7e piste) sur les opéra-

Salle de calcul du système ASTT (simulation militaire).



B. Lotta/Il Dagherotipo

tions exécutées par un contrôleur ; enfin la 8e piste est réservée à la synchronisation. Le sous-système des cabines possède dix éléments contenant l'ensemble des appareillages nécessaires à l'entraînement des officiers.
Le sous-système « rapport » se compose de l'ensemble de fonctions et d'appareils disponibles (énumérés ci-après) pour effectuer l'analyse après les manœuvres :

— enregistrement de toutes les données en temps réel sur bande magnétique,
— enregistrement des communications radio et commentaires sur bande magnétique,
— tableaux, d'ensemble et de détail, des données et des événements significatifs des opérations effectuées,
— écran géant...

Le sous-système de communications met à la disposition des stagiaires jusqu'à 18 lignes radio reproduisant des conditions réelles (bruits et limitation), 3 réseaux de télétransmission RATT (téléscripteurs), et enfin un système de communication de service.

La **réalité simulée** par l'ASTT est extrêmement **complexe et structurée**, surtout pour la corrélation et l'intégration des parties composantes. Les éléments fondamentaux du scénario simulé sont les suivants :

- **Théâtre des opérations,** composé de la mer, du ciel et de la terre dans diverses conditions d'environnement (état de la mer, direction et vitesse des courants marins, du vent, visibilité, conditions météorologiques, propagation radio, acoustique...).

- **Unités aéronavales** subdivisées en unités de surface (du porte-avion à la plus petite unité), sous-marins (conventionnels et nucléaires) et unités aériennes (de l'avion de combat à l'hélicoptère) avec toutes les caractéristiques statiques et dynamiques (dimension et forme, surfaces équivalentes, radar, sonar et infrarouge, vitesse accélération, altitudes ou profondeurs maximales, autonomie, facilité de manœuvre...).

- **Bases aéronavales**, centres au sol ou sur des plates-formes en mer pour l'assistance et le commandement des unités aéronavales, avec tous les équipements, capteurs, systèmes de contre-mesure et éventuellement systèmes d'armement.

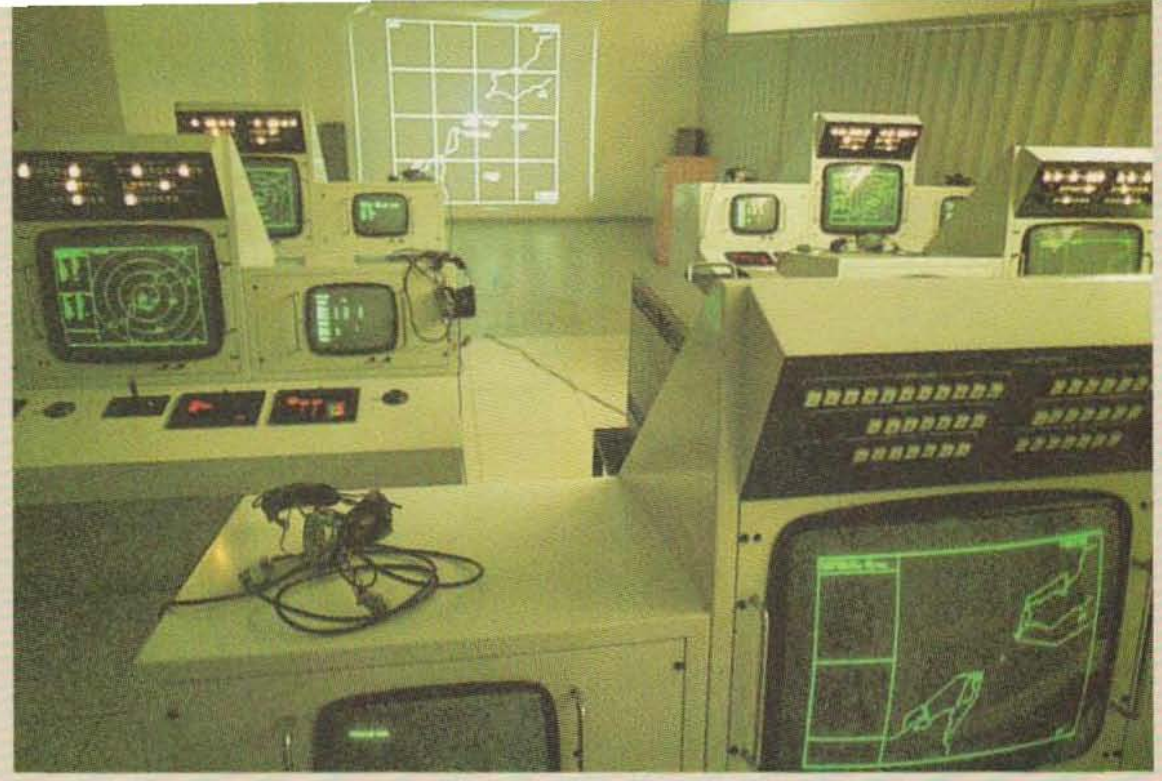
- **Appareils et capteurs de bord**, notamment radar, sonar, appareils d'identification (IFF), de mesure de signaux électromagnétiques (ESM), systèmes de bouées sonores, capteurs d'anomalies magnétiques (MAD), capteurs optiques, etc.

- **Systèmes de communication**, répartis en phonique (radio), transmission de données (data link) et transmission par téléscripteurs (RATT).

- **Systèmes de contre-mesure électroniques** (ECM), composés d'appareils de brouillage et de bruit, électromagnétique et à infra-rouges...

- **Systèmes d'armement** composés de torpilles, de canons, de missiles et antimissiles, systèmes missile/torpille, charges en profondeur, mines.
Les programmes tiennent compte, bien entendu, des modalités d'emploi de ces systèmes et de leurs limites.

- **Ressources humaines** : toutes les activités effectuées par l'homme à bord des unités ou à terre sans toutefois être l'objet d'un exercice spécial des équipes de commandement, même si elles font partie intégrante d'une manœuvre.
Ainsi des activités liées au système radar à bord d'un navire qui se trouvent placées sous le contrôle du commandement pour ce qui concerne la mise en route, le mode de fonctionnement, l'utilisation des appareils, alors même que ce sont les opérateurs radar qui contrôlent les appareils et recueillent l'information.
Ces données — en général position, cap et vitesse d'une unité donnée — après avoir été éventuellement reliées à d'autres, sont ensuite présentées à la salle de commande sur écrans graphiques.
A l'ASTT, dont le but n'est pas de former des opérateurs radar (c'est le rôle des simulateurs spéciaux), toutes les tâches accom-







B. Liotta/Il Dagherrotipo

Les photos représentent les cinq consoles-instructeur et l'écran géant (en haut) ainsi que les consoles cabine (en bas).

plies par les opérateurs sont simulées par des programmes spécifiques. Ceux-ci fournissent ensuite aux équipes de commandement les mêmes informations que celles qu'ils auraient reçue directement des radars.

Les trois activités principales des ASTT concernent les activités suivantes :

— préparation,

— exécution,

— revisualisation des manœuvres,

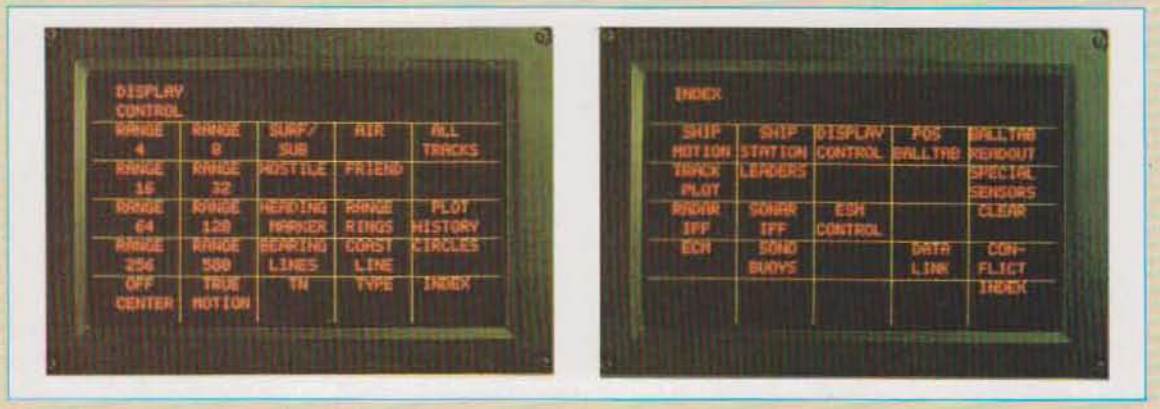
— accessoirement, archivage et la mise à jour des données relatives aux unités aéronavales, aux scénarios opérationnels, etc.

Un groupe d'officiers instructeurs programme les manœuvres en fonction des exigences de l'entraînement et des groupes de commandement à former (l'ASTT est en mesure de couvrir jusqu'à dix manœuvres différentes et simultanées).

On choisit un théâtre d'opération (jusqu'à 160) dont les dimensions maximales atteignent 1 000 × 1 000 milles, de 0 à 900 mètres de profondeur, et de 0 à 90 000 pieds d'altitude, et on détermine les unités qui doivent y participer, ainsi que leurs positions initiales.

Les opérations se déroulent en partie dans des bureaux grâce notamment au dialogue avec l'ordinateur qui demande à l'utilisateur toutes les informations nécessaires, comme le nom de la manœuvre, le théâtre des opérations, l'état de la mer, la visibilité, les conditions de propagation radio, le plan des fréquences de transmission à employer, les unités en jeu et leur position initiale.

B Liotta/Il Dagherrotipo



Visualisation des consoles du système ASTT.

Toutes ces informations sont mises en mémoire sur un disque.

L'exécution d'une manœuvre est précédée par une phase d'explication au cours de laquelle sont précisées, aux personnes en formation, les ***tâches de chacun « à bord »*** *des cabines ainsi que toutes les informations nécessaires au déroulement correct du « jeu ».*

Une fois à bord de leur cabine, les officiers deviennent les commandants de leurs unités. A partir de ce moment, leurs décisions détermineront le déroulement des manœuvres.

De leur cabine, les officiers voient et perçoivent le monde extérieur (théâtre opérationnel) comme s'ils se trouvaient réellement en ***salle de commandement de l'unité simulée.***

Cela signifie que, si une unité se trouve en mer avec capteurs et systèmes de communication éteints, et qu'aucune autre unité n'est détectée dans les limites de sa visibilité, l'écran tactique de la cabine ne reproduit que le symbole de l'unité en question. En revanche, si dans la cabine le radar est allumé et si toutes les conditions nécessaires à la détection sont satisfaites (portée du radar, fréquence de transmission non brouillée par des contre-mesures électroniques, réflexivité radar de la cible, conditions de propagation), les symboles des unités découvertes par le radar apparaîtront normalement à l'écran, ainsi que leur position.

Chez les instructeurs, il en va autrement. L'écran tactique fera apparaître, indépendamment des indications des divers capteurs à bord de chaque unité, toutes les unités en jeu dans leur position réelle, comme si tout le théâtre opérationnel était vu en surplomb sans limitation. Pendant le déroulement du jeu, les instructeurs suivent les événements grâce aux consoles, à l'écran géant, aux télescripteurs :

— Ils commandent les « fightback targets » ou cibles de contre-attaque (unités sans salle de commande proprement dite, pilotées par des équipes à partir des consoles).

— Ils peuvent intervenir en ***modifiant le film des événements.*** *Par exemple, après avoir engagé un combat contre une unité ennemie, une unité donnée subit des dommages tels que radars et systèmes d'antennes de communications hors d'usage.*

— Par contre, les instructeurs pourraient décider de modifier cette situation pour permettre à l'unité de continuer les manœuvres sans limitations, et changer la nature des dégâts enregistrés.

— Les instructeurs peuvent aussi rétablir ou simuler d'autres avaries.

— Lorsque la manœuvre a atteint les objectifs fixés, les instructeurs arrêtent l'action pour passer à une phase d'analyse de ce qui s'est produit. Ce travail est largement simplifié par les possibilités offertes par l'ASTT qui élimine rapidement les éventuelles erreurs.

La possibilité de ***revoir sur écran géant le déroulement de la manœuvre*** *dans son intégralité, de réécouter les communications radio, d'accélérer le playback et de se* ***repositionner*** *à tout moment du « wargame » fait de l'analyse a posteriori la partie la plus intéressante de l'entraînement tactique.*

Bruno Liotta
Datamat Ingegneria dei Sistemi (Italie)

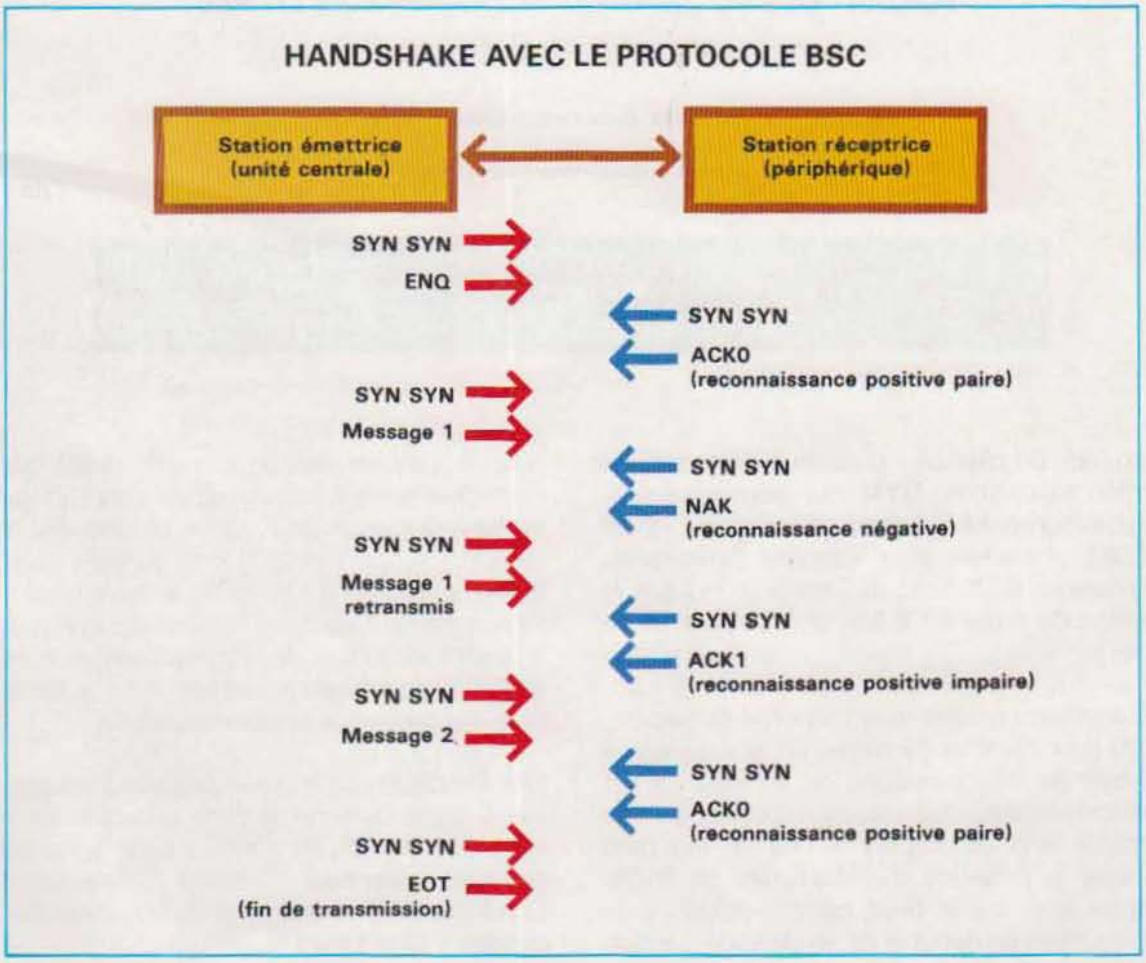
Le protocole BSC

La procédure BSC (Binary Synchronous Communication) a été introduite pour la première fois par IBM en 1966 ; elle a permis de stabiliser la situation confuse qui prévalait alors en matière de communication. Sa très large diffusion en a fait un standard de facto, même si l'initiative ne venait pas d'un organisme officiel de normalisation.

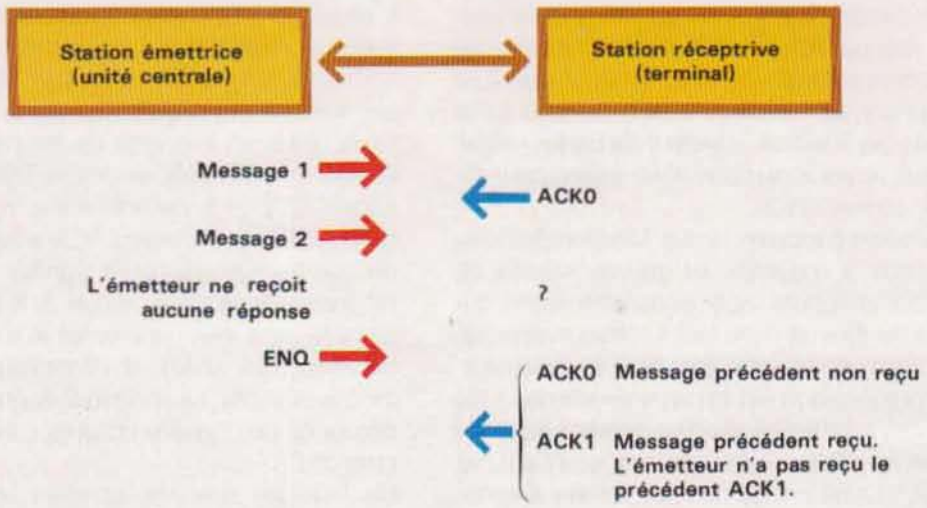
Le protocole concerne tout le faisceau de communication à moyenne et grande vitesse du type bidirectionnel non simultané (semi-duplex). Il ne prévoit donc pas l'échange simultané d'informations entre émetteur et récepteur. La communication est de type synchrone : les messages sont précédés d'un nombre défini de caractères SYN (tableau codes ASCII et EBCDIC). La séquence de « handshake » adoptée (schéma ci-dessous) est composée d'une version modifiée de l'ENQ/ACK. On a prévu deux caractères de reconnaissance positive (ACK0 et ACK1) qui sont envoyés alternativement par le récepteur pour signifier que le texte n'est pas entaché d'erreur.

L'emploi des accusés de réception pairs et impairs garantit la réception des reconnaissances et des messages tout entiers. En effet, si l'unité qui transmet ne reçoit rien après l'envoi d'un texte, dans un intervalle de temps donné, elle envoie à nouveau le caractère ENQ (tableau 1 page 1350). Si le destinataire a reçu le message précédent, il envoie l'ACK alternatif (ACK1 dans le tableau cité) pour signifier que seule la reconnaissance a été perdue. Si le message n'a pas été reçu, elle retransmet le même accusé de réception ACK0 et l'émetteur renvoie le message perdu. Le tableau 2 donne le contenu des blocs de données échangés avec le protocole BSC.

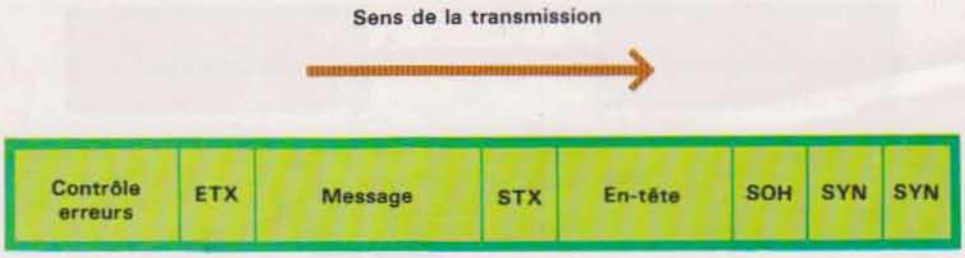
Un bloc de données contient en général 3 éléments : un en-tête optionnel (informations extérieures au message), le texte proprement dit et un dernier élément pour le contrôle des



SIGNIFICATION DES ACCUSES DE RECEPTION ACK ALTERNATIFS EN BSC



FORMAT DES INFORMATIONS TRANSMISES EN BSC



erreurs. De plus, on trouve certains caractères spéciaux comme : **SYN** pour assurer la synchronisation entre émetteur et récepteur, **SOH** Start of Header pour identifier l'en-tête du message, **STX** Start of Text pour indiquer le début du texte, **ETX** End of Text pour fin de texte.

L'en-tête comporte un nombre fixe de caractères pour identifier l'émetteur et le destinataire (code de reconnaissance ou adresse) ; il est employé s'il existe plusieurs dispositifs sur une même ligne. La longueur fixe de l'en-tête rend inutile la présence d'un caractère de fin de texte alors que le texte exige la présence de caractères de début et de fin de texte. Le bloc pour le contrôle des erreurs est généré par l'émetteur grâce à des algorithmes appropriés. Au fur et à mesure qu'il prélève les données, le récepteur reconstruit ce bloc de contrôle (avec le même algorithme) et en fin de transmission il compare les deux blocs reconstruits et reçus. S'ils sont identiques, le récepteur envoie le signal ACK (pair ou impair) ; sinon, c'est le signal NAK qui permettra la retransmission.

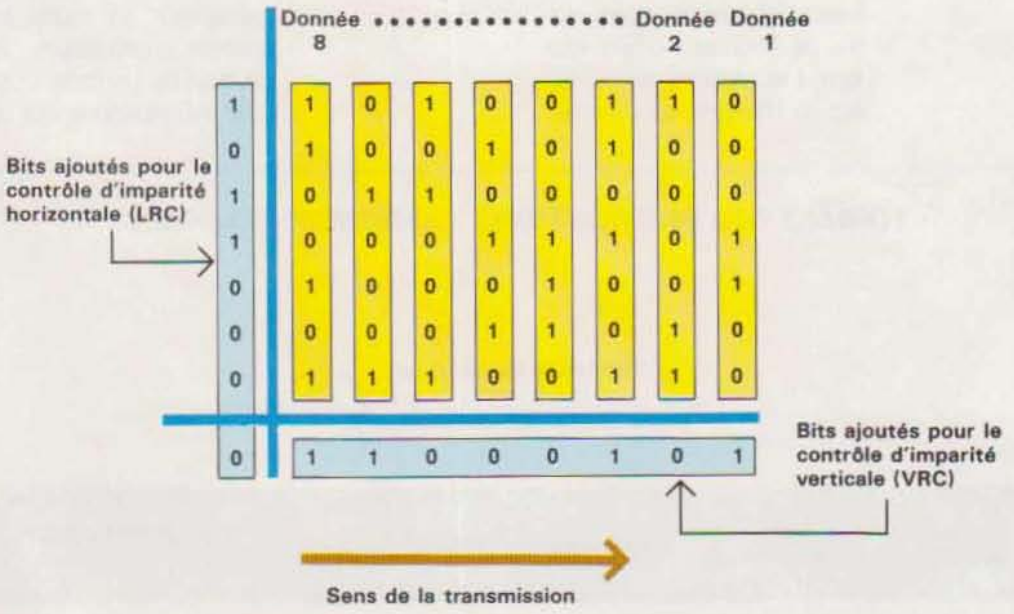
Une fois déterminé le mode de parité à adopter (dans le cas du schéma page ci-contre, on a adopté l'imparité), on effectue deux types de contrôle : **vertical** (Vertical Redundancy Check ou VRC), **horizontal** (Longitudinal Redundancy Check ou LRC).

A chaque caractère envoyé (représenté verticalement dans le tableau) l'émetteur ajoute un 1 si le nombre de bits 1 dans le caractère est pair, et un 0 s'il est impair. Le nombre global de bits 1 de chaque caractère doit donc être impair. Si le récepteur découvre que le nombre de 1 d'un caractère est pair, il a détecté une erreur, auquel cas il demande la retransmission. Cette méthode (VRC) n'est plus valable lorsque le récepteur comptabilise un nombre impair de 1, mais le caractère reçu ne correspond cependant pas à celui qui a été envoyé (à cause d'une double inversion par exemple). C'est pour cela qu'on ajoute le contrôle horizontal (LRC) qui analyse tous les bits de manière longitudinale et ajoute un 1 si le nombre de 1 est impair. Le contrôle se fait pour tous les caractères qui constituent soit le texte, soit l'en-tête (voir tableau page 1351). La colonne de bits, complétée par le bit d'imparité, est enregistrée dans la phase finale de contrôle des erreurs. Si la transmission est correcte, la séquence de bits enregistrée par l'émetteur est celle qui a été recomposée par le récepteur.
Le protocole BSC, encore aujourd'hui très répandu dans les communications à courte et longue distance, de type semi-duplex, ne permet donc pas l'échange simultané d'informations dans les deux sens. Autrement dit, en trafic dense, les attentes sont longues et ne sont réduites qu'avec un dialogue de type bidirectionnel simultané.

Le protocole HDLC

Mis au point par l'International standard organisation (ISO), ce protocole (High level Data Link Control) a une structure lui permettant d'accueillir des communications de type semi-duplex et bidirectionnel simultané. A la différence du BSC à base de caractères — il prévoit de transmettre des caractères (ASCII ou EBCDIC) sur les lignes — le protocole HDLC s'appuie sur les bits, car il permet la transmission d'une configuration quelconque de bits.
Du point de vue du contrôle du trafic, le protocole ne fait pas référence à des unités émettrices et réceptrices mais à des stations primaires et secondaires : la primaire entame le dialogue et les secondaires y répondent.

CONTROLE D'IMPARITE HORIZONTALE (LRC) ET VERTICALE (VRC)

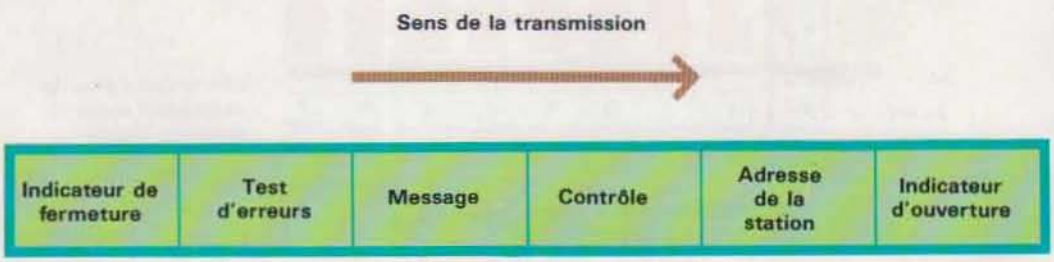


Cette schématisation est liée au type de configuration adopté (point-à-point, multipoint,...) ; cela ne signifie pas nécessairement que la station secondaire ne peut parler que si elle est interrogée.
En général, le dialogue entre stations peut être asynchrone, car chacune d'elles communique quand elle le désire. Le format des informations transmises en HDLC est donné ci-dessous. Le bloc tout entier est défini une **trame** décomposée en message et en informations de contrôle. Toutefois, le message peut ne pas se trouver dans la trame : les éléments de contrôle servent alors à adresser ou à fournir des informations aux stations primaires et secondaires.
Toutes les trames sont numérotées en séquence, chacune comptabilisant les trames qu'une station a précédemment envoyées ainsi que le numéro de la trame que l'émetteur s'attend à recevoir en retour. La part de handshake est donc interne au protocole même.
Voici énumérés les principaux champs composant une trame :

Indicateur C'est une séquence de bits, unique dans toute la trame et d'une longueur fixe (par exemple la séquence 01111110). L'indicateur informe le récepteur qu'une trame va être envoyée, suivie de l'adresse d'une station. Les stations secondaires se mettent en attente.

Adresse de la station contrôle Elle identifie le destinataire du message, éliminant ainsi les trames dupliquées, omises ou erronées. Elle contient notamment le nombre de trames transmises (NS) et de trames reçues (NR). Les valeurs de NS et de NR sont différentes pour chaque station. Dans un dialogue entre deux stations, les trames signifient : « le numéro de la trame que j'envoie à la station est NS ; la prochaine trame à recevoir de cette station est NR ». La station compare son NR et la valeur de NS contenue dans la trame ; s'ils sont égaux, et s'il n'y a pas d'erreur (voir plus loin), la trame est acceptée et les valeurs de NR et NS mises à jour. En plus de NR et NS, le bloc de contrôle contient des informations supplémentaires qui permettent de transmettre beaucoup de trames avant même que le récepteur ait fourni la réponse protocolaire. Enfin, le bloc de contrôle contient des informations sur la na-

FORMAT DES INSTRUCTIONS TRANSMISES EN HDLC



ture et la longueur du message qui suit.

Message La longueur est illimitée et son code quelconque (EBCDIC, ASCII, binaire,...), grâce à la combinaison d'informations contenues dans l'adresse de la station et dans le bloc de contrôle. Si une station ne peut dialoguer qu'en ASCII, on associe ce code à son adresse. On précise alors dans le bloc de contrôle que le message qui suit est en ASCII.

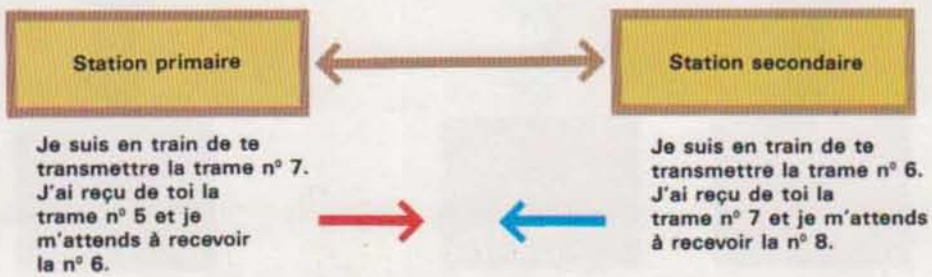
Test erreur Il est réalisé selon la méthode décrite BSC, ou en employant des techniques proches. L'émetteur envoie son propre bloc pour le test des erreurs et le récepteur compare celui qu'il a calculé et celui contenu dans la trame. Si les deux blocs ne sont pas identiques, la trame est rejetée. Le récepteur met à jour son compteur NR, et il en demandera une nouvelle transmission. Il n'est nullement nécessaire d'interrompre la communication pour retransmettre des messages ; cette demande (handshake) est, en outre, interne au protocole. (Pour cette raison on peut qualifier le protocole HDLC de bidirectionnel simultané.) (Voir ci-dessous.)

Indicateur de fermeture De même format que l'indicateur d'ouverture, il identifie la fin de la trame transmise.

Le protocole HDLC constitue aujourd'hui la base pour réaliser des réseaux d'ordinateurs complexes. Grâce à sa souplesse, on a pu créer des extensions de protocole, du type SDLC (Synchronous Data Link Control) typique d'IBM, et actuellement le plus répandu en communications « parallèle ». Adopté en 1978 par l'institut de normalisation IEEE (Institute of Electric and Electronic Engineering), il est employé pour gérer des périphériques rapides (unités disque et bande, instruments de mesure électroniques) à courte distance (quelques mètres). Le protocole est du type « octet sériel, bit parallèle » et permet jusqu'à 15 machines de converser à grande vitesse (jusqu'à un million de caractères par seconde).

Le protocole IEEE-488 fixe, en outre, un standard au niveau des interfaces, en établissant des règles précises concernant les niveaux de tension, les appareils de connexion et les câbles de liaison. Un câble IEEE-488 est composé de 16 fils, dont 8 utilisés pour la transmission des données et les autres pour le contrôle des données transmises. C'est la raison pour laquelle le protocole est dit parallèle : un caractère se présente au récepteur simultanément,

HANDSHAKE INTERNE A LA TRAME DANS LE PROTOCOLE HDLC



sur les 8 fils destinés à la transmission des données. La liaison des divers dispositifs est illustrée dans le schéma ci-dessous (définie comme bus). Chaque dispositif comporte une adresse (variant entre 1 et 15). Le protocole établit, qu'à un instant donné, il y a sur le bus un émetteur unique et un ou plusieurs récepteurs. Le bloc de données, que l'émetteur envoie, est reçu par le ou les récepteurs dont l'adresse est incluse dans le bloc transmis. A l'instant suivant, l'émetteur peut changer, comme d'ailleurs les récepteurs.

Les **lignes données** sont composées de 8 fils où normalement transitent des caractères ASCII en parallèle ; 7 fils sont employés pour l'information et le 8e est dédié au contrôle de parité. Les lignes transmettent également les instructions que les dispositifs veulent échanger ainsi que l'adresse des dispositifs choisis pour l'écoute.

Les **lignes de handshake** sont au nombre de 3 et sont utilisées pour coordonner le transfert d'information sur le bus. La communication est asynchrone et la vitesse de transfert entre le dispositif émetteur et un ou plusieurs récepteurs s'adapte automatiquement à la vitesse du dispositif le plus lent. Les diverses lignes de handshake ont la signification suivante :

DAV Data Available. Est pilotée par l'émetteur pour informer le récepteur qu'il y a des données sur le bus.

NRFD Not Ready For Data. Est contrôlé par le récepteur pour informer l'émetteur qu'un des récepteurs n'est pas encore prêt.

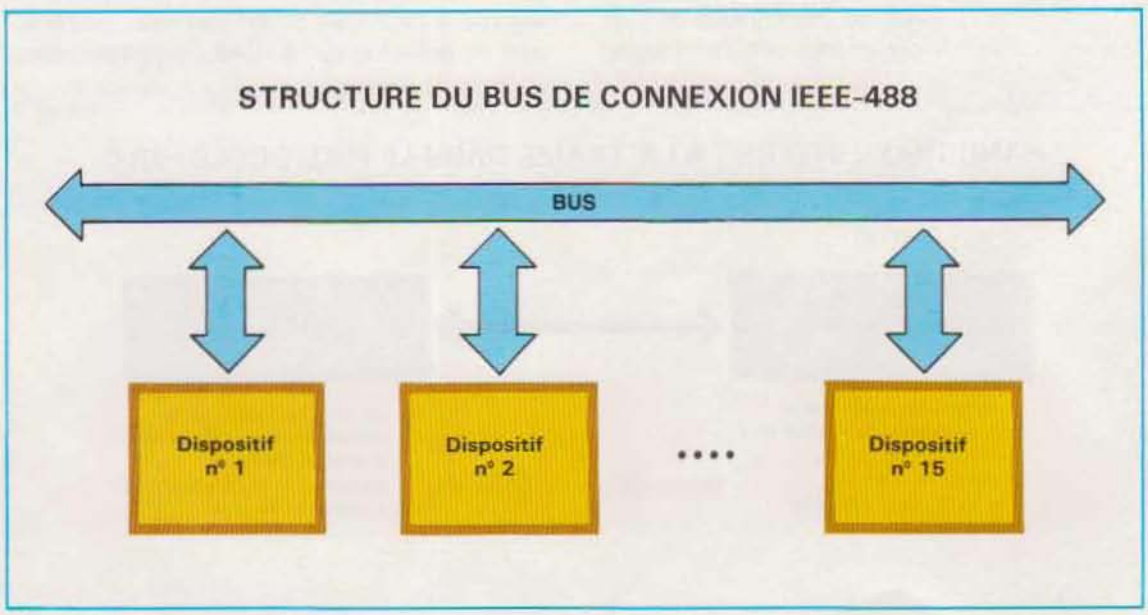
NDAC Not Data Accepted. Est toujours géré par le récepteur : les données ont bien été reçues mais on ne peut en accepter d'autres. Cette ligne est prise par l'auditeur le plus lent à la fin d'une transmission.

Les **lignes de contrôle de données** sont employées pour régler le flot de données sur le bus. Chaque ligne prend la signification suivante :

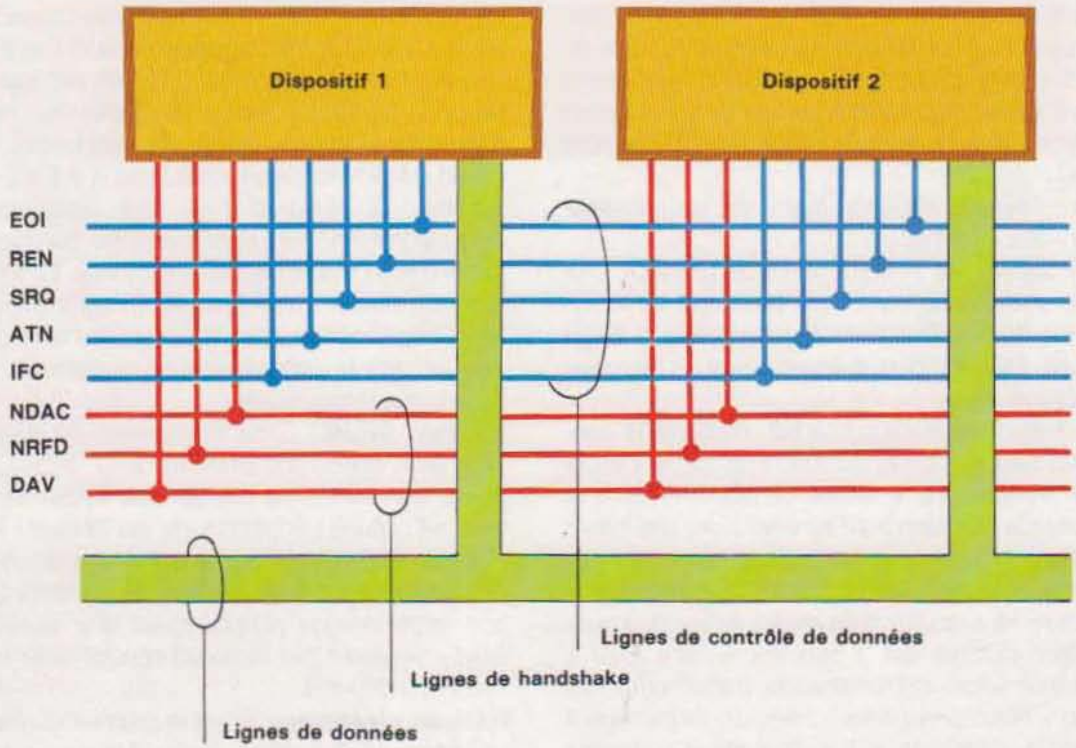
IFC Interface Clear. Est utilisée pour faire un reset (réinitialisation) du bus, en le positionnant donc sur « départ ».

ATN Attention. Oblige tous les dispositifs à interpréter le contenu du bus comme « ordre » (ligne haute) ou comme « donnée » (ligne basse).
Grâce aux ordres, on décrit l'émetteur et le récepteur ; leur adresse est enregistrée dans les lignes données. Sur le bus données, les dispositifs échangent éventuellement leurs informations.

SRQ Service Request. Est la ligne qui permet le dialogue entre des dispositifs sous interruption (interrupt). Quand



SCHEMA DETAILLE DU PROTOCOLE IEEE-488



cette ligne est employée, un des quinze équipements connectés au bus devient contrôleur et établit comment le dialogue doit être poursuivi. On peut imaginer la situation suivante : un dialogue a lieu entre émetteur et récepteurs ; la ligne SRQ devient haute (signifiant qu'un autre appareil demande un droit d'émission). Le contrôle interroge les divers dispositifs (polling) et dès qu'il découvre celui qui a demandé la ligne, lui donne la priorité.

REN Remote Enable. Est gérée par ses équipements qui peuvent se trouver dans deux états différents : local et remote (sur place et à distance). En local, ils peuvent accepter des ordres d'un opérateur, par exemple à partir d'un clavier ; sur « remote », ils peuvent accepter des ordres et des données du bus.

EOI End Or Identify. Indique la fin d'un groupe d'informations transmises.

Réseaux pour la communication de données

Tous les éléments employés pour la télétransmission de données-interfaces, lignes, modes de transmission, protocoles... se retrouvent dans les réseaux de communication.
Un réseau est, en général, un ensemble matériel (les machines) et logiciel organisant et gérant la communication.
Grâce au réseau, les ressources communes sont partagées. Par « ressource », on entend tout ce qui est mis à disposition de l'utilisateur : imprimante, mémoire de masse, programme, banque de données offertes localement ou à distance. Un réseau peut être composé par des gros ordinateurs appelés « mainframes » (unités centrales) ou ordi-

nateurs-hôtes, par de petits ordinateurs (mini et micro), des terminaux et des périphériques.
Un réseau peut être de type **centralisé** ou **distribué.** Dans l'architecture centralisée, c'est l'unité centrale (UC) qui effectue la majeure partie des traitements demandés aux divers points d'entrée du réseau. A ces points sont reliés des terminaux ou des dispositifs d'entrée/sortie dont la tâche est d'interroger l'UC.
Un tel réseau est donc fondé sur la centralisation des informations.
A l'opposé, le réseau distribué possède plusieurs unités de traitement **(nœuds)** au moins dotés de la même puissance, selon qu'il s'agit d'une UC ou, plus fréquemment, d'un mini-ordinateur.
Un nœud regroupe un certain nombre de périphériques au niveau local ou à distance. De par son architecture, le réseau distribué renforce et facilite la diffusion d'informations sur des zones géographiques éloignées. Ce dernier type de réseau est aujourd'hui le plus répandu, car il élimine le « goulot d'étranglement » de l'ordinateur central, qui a souvent à faire face à d'importantes demandes de traitement. Plusieurs ordinateurs interconnectés se partagent ainsi la tâche, et si l'un d'entre eux devient inopérant (panne, par exemple), les autres sont en mesure de prendre en charge ses travaux.
La création de réseaux d'ordinateurs permet, aujourd'hui, d'accéder à d'importantes banques de données, élaborées et distribuées par eux. Cet accès est également possible au niveau domestique : il suffit de doter la télévision d'un appareil spécial pour établir une connexion avec le nœud le plus proche. Le réseau fournit les informations les plus variées : horaires de trains et d'avions, pharmacies de garde, informations communales... Dans le cas du vidéotex, il est possible de relier un clavier au réseau téléphonique pour obtenir directement une réservation de billet d'avion, envoyer un message à d'autres usagers, etc. Dans certains pays, on reçoit déjà à domicile son journal, par simple connexion d'une imprimante à la télévision.

Topologie et caractéristiques des réseaux

Les nœuds d'un réseau s'articulent de différentes manières, tout en respectant les architectures décrites ci-dessous.

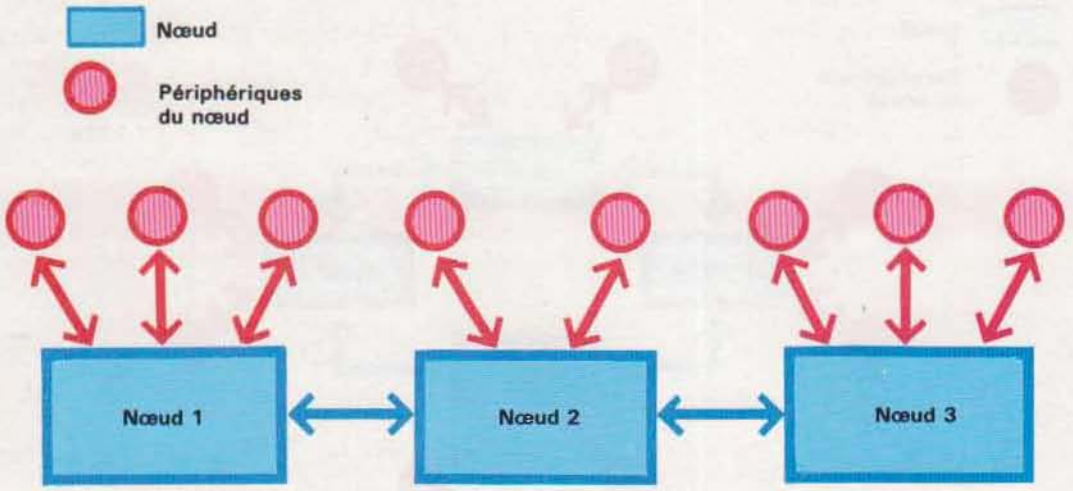
Réseau en chaîne. C'est la généralisation de la connexion point-à-point (schéma 1, p. 1357). Les nœuds sont reliés en cascade ; les communications partant du nœud 1 et destinées au nœud 3 passent forcément par le nœud 2. Le réseau est simple à installer, mais il présente l'inconvénient de dépendre du nœud intermédiaire pour les liens entre les extrémités. Si le nœud 2 ou tout autre nœud intermédiaire ne fonctionne pas, les communications sont totalement interrompues. La topologie en chaîne est employée lorsque chacun des nœuds doit accomplir une tâche bien précise et que la communication entre tous les nœuds n'est pas vitale.

Réseau étoilé. C'est l'extension du réseau centralisé concentré (schéma 2, p. 1357). Le nœud central trie la masse des informations vers les nœuds périphériques, qui mettent à la disposition de l'extérieur la partie des données à laquelle ils ont droit d'accès. La communication entre nœuds périphériques doit transiter nécessairement par le nœud central dont tout le réseau dépend.

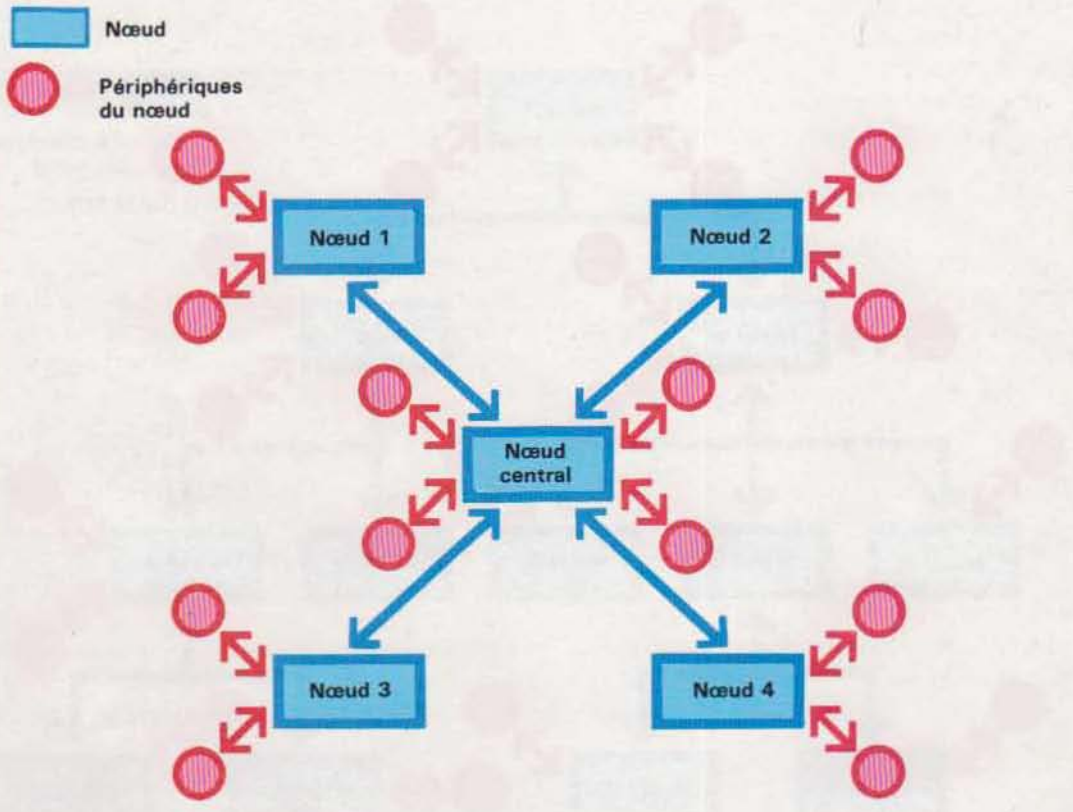
Réseau en anneau. C'est le premier exemple de réseau non asservi à un nœud (schéma 3, p. 1358). Les informations partant du nœud 1 vers le 3 transitent soit par le nœud 2, soit par le nœud 4. En général, on convient d'un chemin (par exemple en passant par le nœud 2) et, en cas de difficulté, on adopte une voie alternative (le nœud 4). Le réseau en anneau présente donc l'avantage d'établir une communication même quand surgit un problème au niveau d'un nœud. Seul inconvénient : son coût, chaque nœud étant doté de deux lignes de communication. Dans le réseau en anneau, aucun ordinateur (ou nœud) n'est privilégié par rapport aux autres, et ils ont la même puissance de calcul.

Réseau arborescent ou hiérarchisé (schéma 4, p. 1358). Dans cette architecture, chaque nœud a des fonctions bien particulières avec un rapport de dépendance à l'égard des autres nœuds selon des niveaux hiérarchiques. Le nœud 2 (p. 1359), par exemple, transfère des informations aux nœuds 4, 5 et 6 tandis que le 5 transmet vers les nœuds 9 et 10.
Pour communiquer depuis le nœud 10 avec le nœud 6, on doit passer la demande au nœud

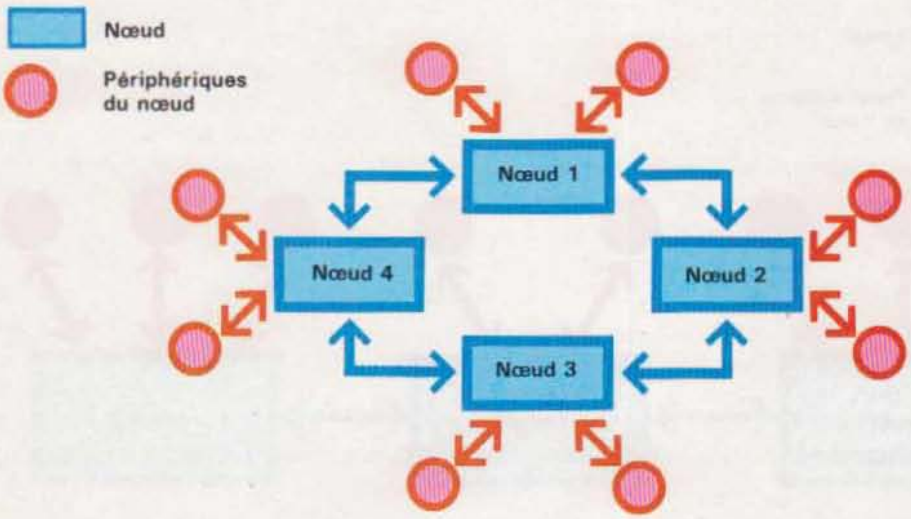
1 / RESEAU EN CHAINE



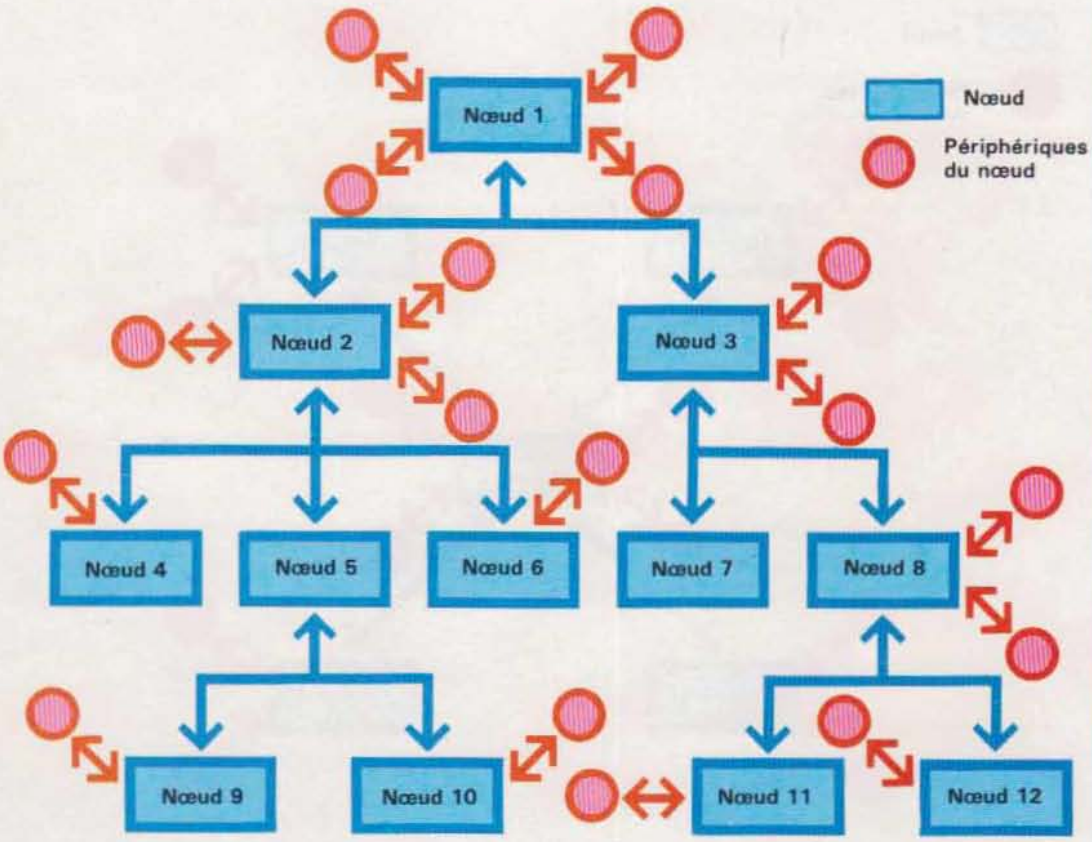
2 / RESEAU EN ETOILE



3 / RESEAU EN ANNEAU



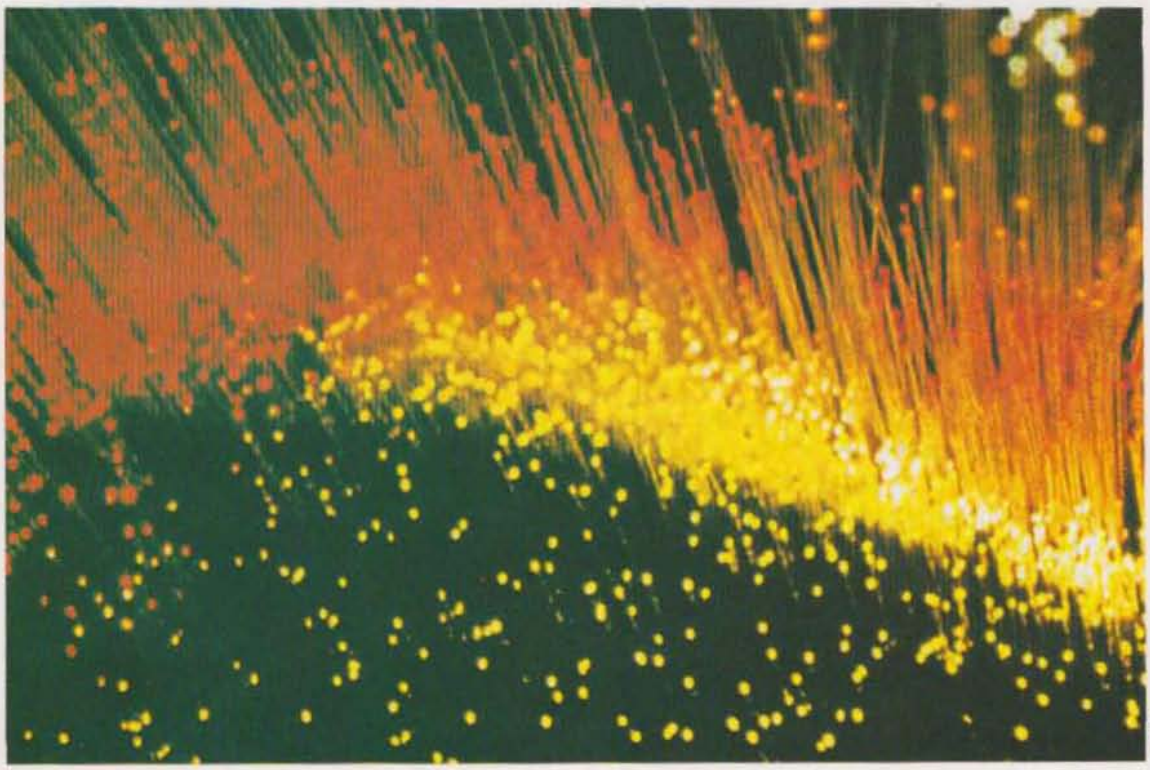
4 / RESEAU ARBORESCENT OU HIERARCHISE



supérieur (le 5), ce dernier « reportant » ensuite au 2 pour enfin aboutir au nœud d'arrivée 6. Si une demande de traitement a été adressée au nœud 9 alors que ce n'est pas de son ressort, elle sera « renvoyée » au nœud supérieur et ainsi de suite. La topologie hiérarchisée est surtout employée dans les réseaux où la sécurité d'accès aux informations est vitale (réseaux privés reliant des agences bancaires, etc.). En réalité, les architectures sont souvent imbriquées pour obtenir des réseaux certes complexes, mais bénéficiant des avantages de chacune d'entre elles.

Dans ces réseaux, deux propriétés revêtent une importance toute particulière : **stockage** et **retransmission** (store and forward) d'une part, et **communication par paquets** (packet switching) d'autre part. La première est une technique de transmission de messages entre deux nœuds non directement reliés. Dans un réseau, même complexe, un message a un nœud pour origine et un autre pour destination. Dans les protocoles modernes de communication (voir HDLC), le nom de l'émetteur et l'adresse du destinataire sont écrits dans le message même. Celui-ci devra donc passer par des nœuds intermédiaires qui, sans être les vrais destinataires, ont une fonction importante dans le réseau : ils stockent en mémoire le message si la ligne est occupée et le font « suivre » (forward) dès qu'elle se libère. C'est le cas dans l'architecture 1 par exemple (p. 1357) de la fonction du nœud 2 quand les deux autres (1 et 3) désirent communiquer. Ainsi, le nœud d'un réseau de type stockage et retransmission doit être suffisamment « intelligent » pour savoir si le message reçu lui est destiné ou ne fait que transiter vers le haut. La commutation par paquets est une autre technique, plus rapide, de transit d'un message à travers des nœuds qui n'en sont pas les destinataires. Avec cette méthode, les informations voyagent entre deux nœuds en suivant un chemin fixé à l'avance. En d'autres termes, s'il existe différentes voies reliant deux nœuds, la commutation par paquet est capable de déterminer quelle est la voie libre à un instant donné. Les informations sont empilées pour être ensuite transmises, toujours avec une longueur fixe, ce qui accélère la communication.
Imaginons, par exemple, un autobus avec 50

Conduction de lumière polychrome dans un faisceau de fibres optiques.



Schott

personnes à bord et dont l'itinéraire est de traverser la ville. Le chauffeur changera son trajet s'il se rend compte que celui-ci est embouteillé. Les 50 personnes sont synonymes d'autobus à pleine charge, pour que le voyage ne soit pas économiquement une perte. Il se produit la même chose avec des informations empilées et d'une longueur fixe.

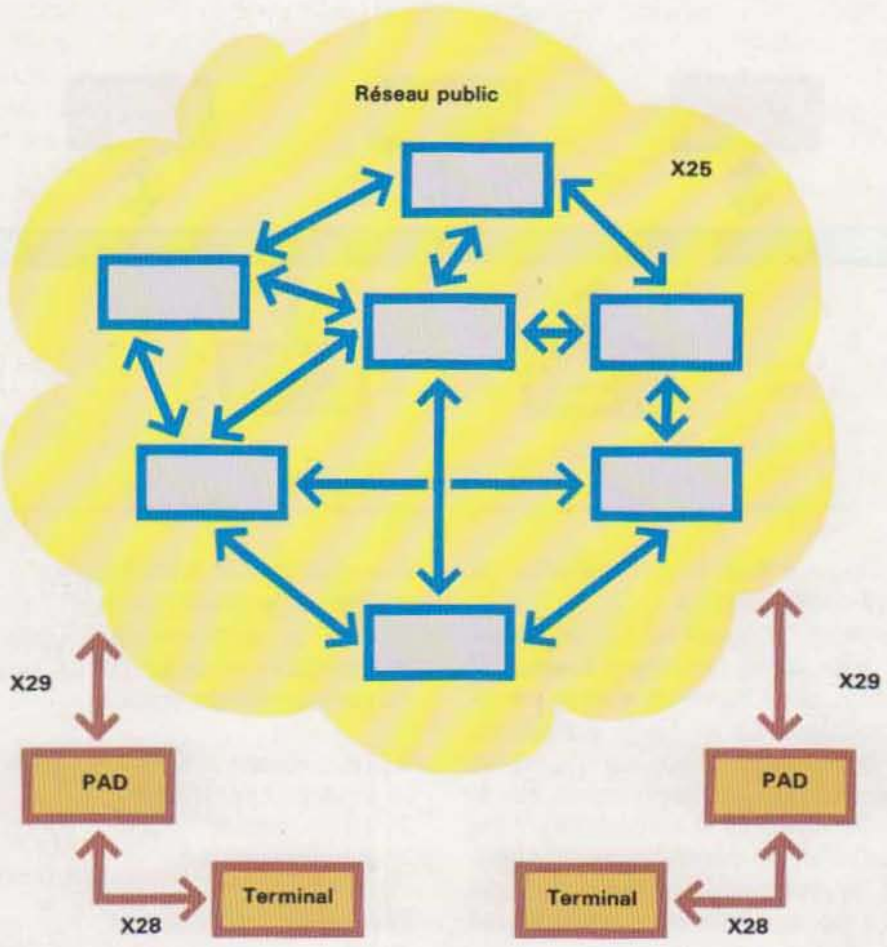
Réseaux publics

La diffusion des mini et des micro-ordinateurs a amené les PTT à créer des systèmes d'échange et de transfert d'informations.
Le concept nouveau, par rapport aux télécommunications traditionnelles, présente deux aspects : l'interconnexion des ordinateurs disponibles affectés à la gestion du trafic (échange d'informations dans le réseau), la comptabilisation de la durée de la communication sur d'autres bases que le temps.
Avant l'avènement des réseaux dits publics, pour que deux utilisateurs communiquent à distance, ils devaient demander une ou plusieurs lignes téléphoniques. La ligne n'était alors disponible qu'après avoir composé le numéro avec un débit faible, de l'ordre de 300 bps. La durée était comptée même s'il y avait des silences dans les communications après ouverture de la ligne.
Dans les réseaux publics, au contraire, seul est comptabilisé le volume d'informations transmises (les silences ne pèsent donc pas sur les coûts). Ce résultat est lié à l'emploi de la commutation par paquets et aux protocoles modernes grâce auxquels l'itinéraire emprunté par un message dans le réseau n'est pas toujours le même. Celui qui ne transmet pas est ignoré du réseau, alors qu'un émetteur aura la garantie que son message arrivera à destination, quelle que soit la voie prise.
Ce réseau public appelé Transpac (transmission par paquets) a été mis en service en France vers 1979. La plupart des réseaux publics nationaux obéissent à une standardisation des modes d'accès. Les normes font référence soit aux interfaces entre réseau et ordinateur (ou terminal) soit aux protocoles de communication.
Le Comité Consultatif International de Télégraphe et de Téléphone (CCITT) dont le siège est à Genève, a établi la recommandation qui, de nos jours, est la plus suivie pour les réseaux publics, et qui est devenue une norme de facto : X25. Elle fixe les règles que doivent suivre les nœuds par des dispositifs matériel et logiciel appelés PAD (pour Packet Assembler-Disassembler ou assemblage-désassemblage de paquets) et par des règles provenant de la recommandation X3 (toujours du CCITT). La liaison d'un terminal au PAD est gérée par la recommandation X28, celle du PAD au réseau par X29 (voir page ci-contre). Grâce aux recommandations du CCITT, la transmission de données a réalisé un progrès décisif, notamment en permettant à des ordinateurs d'origine différente de dialoguer entre eux.
Imaginons une société qui dispose d'un ordinateur de marque A à Lyon et d'un ordinateur de marque B à Paris et enfin d'un ordinateur de marque C à Brest. L'existence d'un réseau public respectant le protocole X25 oblige les constructeurs A, B et C à doter leur machine d'interfaces adéquates pour autoriser leur fonctionnement selon les modalités précises d'accès au réseau. Notre société pourra ainsi faire dialoguer ses 3 ordinateurs sans être obligée de construire un réseau privé.

Réseaux locaux

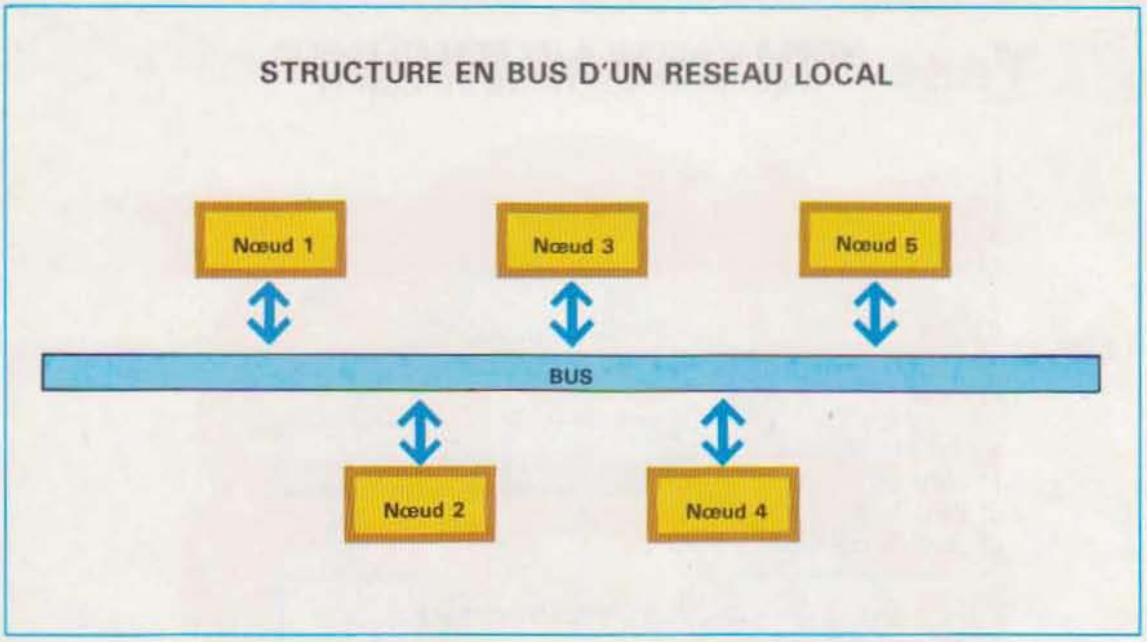
Les réseaux conçus sont généralement connus pour couvrir des aires géographiques étendues. Mais on peut aussi avoir besoin de distribuer des informations dans un espace plus restreint, de l'ordre de quelques kilomètres voire de quelques centaines de mètres. Ce sera le cas d'une société ayant installé ses ordinateurs dans un même bâtiment, ou dans plusieurs bâtiments voisins. Dans ce cas, il vaut mieux concevoir un « réseau local » dans lequel on ne recourt pas aux lignes téléphoniques.
Le plus gros avantage des réseaux locaux, par rapport aux réseaux publics, sera la vitesse de communication entre les nœuds. On peut, en effet, atteindre jusqu'à 100 millions de bits par seconde (alors que dans les réseaux étendus, on ne dépasse pas les 50 à 60 000 bps).
Les lignes de communication sont des câbles (en tresse ou coaxiaux) aboutissant à des dérivations sur lesquelles sont connectés les nœuds. Dans un bâtiment, le réseau local est installé en passant un câble dans les locaux où se trouveront les nœuds, en prévoyant autant

MODES D'ACCES A UN RESEAU PUBLIC (RECOMMANDATIONS DU CCITT)



de prises qu'il y aura d'appareils à connecter. La fibre optique commence à se subtituer au câble comme support physique. Cette solution offre la plus grande garantie pour la transmission. Dans la fibre optique, les signaux se propagent à la vitesse de la lumière sous forme d'impulsions, non électriques, et donc la communication n'est pas troublée par l'extérieur. Les topologies du réseau local (souvent défini par le sigle LAN, Local Area Network) sont les mêmes que pour les réseaux en général (en étoile ou en anneaux) avec, en plus, une nouvelle configuration, appelée **bus** (voir page 1362). Dans cette dernière structure, il y a une totale indépendance entre nœuds donc remplacement d'un nœud indisponible.

Chaque nœud envoie un message le long du bus dans les deux directions, et le nœud destinataire le saisit s'il lui est adressé. Le protocole pour les communications dans les réseaux locaux a été mis au point par l'Institute of Electrical and Electronics Engineers (IEEE). La recommandation IEEE 802 établit les distances maximales, les nœuds, les vitesses et les modalités de transmission de données pour les réseaux locaux. Une solution intéressante consiste en l'adoption, comme moyen de communication, des lignes téléphoniques internes à la société. Elles sont gérées par un système appelé PBX (Private Branch X-change) qui permet de faire dialoguer deux bureaux ou d'orienter vers les différents bureaux les appels venant de



l'extérieur. On peut aussi relier l'ordinateur au réseau téléphonique intérieur.
Les équipements connectables sur un réseau local sont très variés : micro-ordinateur et mini-ordinateur, unité bande ou disque, terminaux, imprimantes... La tendance actuelle est de connecter au réseau beaucoup d'appareils de traitement et peu de périphériques, car le coût des unités centrales va en décroissant (ce qui n'est pas le cas des périphériques). L'objectif est donc de partager, entre plusieurs unités centrales, les mêmes unités disque ou bande et la même imprimante.
La phase de croissance suivante, pour un réseau local, est de se connecter aux réseaux externes, donc la possibilité pour tous les dispositifs d'un LAN de s'interfacer à un réseau public, pour accéder à des banques de données et à des puissances de traitement géographiquement éloignées.
De nombreuses études sur ce sujet sont aujourd'hui en cours et beaucoup de solutions s'inspirent de l'emploi des satellites pour amplifier le volume de données transmises dans un même laps de temps.

Vers des architectures standard

Tous les instruments utilisés dans la communication de données, comme les lignes, les protocoles, les réseaux... constituent les briques d'un édifice dont la construction n'est pas encore achevée. C'est l'architecture générale des réseaux de communication. Deux architectures semblent émerger du lot pour unifier la communication de données :

1 / Le réseau SNA (System Network Architecture) d'IBM,
2 / Le modèle OSI (Open System Interconnection).

Construites par couches, ces architectures obéissent à des conceptions différentes. La première couche décrit la manière de réaliser les connexions électriques, donne les valeurs des signaux, etc. La deuxième couche établit les modalités et les protocoles de connexion et ainsi de suite, jusqu'à définir le projet au niveau le plus élevé.
Les deux architectures SNA et OSI comportent sept couches (qui ne sont pas identiques dans les deux solutions). SNA, mise au point par IBM, constitue le modèle dont s'inspirent les réseaux établis par cette compagnie. L'architecture OSI a été mise au point par l'Organisation internationale de normalisation (ISO). L'objectif du modèle est de définir un modèle d'architecture de réseau souple dont pourraient s'inspirer tous les autres constructeurs d'ordinateurs. Déjà beaucoup y font référence tant pour le matériel que pour le logiciel.

Test 23

1 / La fonction de handshake ("main tendue") sert à :

a) relier deux ordinateurs
b) synchroniser celui qui émet et celui qui reçoit
c) construire un protocole
d) faire communiquer à distance deux dispositifs

---

2 / Les caractères SYN sont employés :

a) dans le protocole HDLC
b) dans le protocole BSC
c) dans les protocoles asynchrones
d) dans le contrôle des erreurs

---

3 / Dans le protocole HDLC, le texte transmis peut être :

a) composé de caractères seulement
b) composé de nombres seulement
c) quelconque
d) semi-duplex seulement

---

4 / Le protocole IEEE-488 est :

a) sériel
b) parallèle
c) semi-duplex
d) pour les longues distances

---

5 / La ligne ATN (attention) est employée dans :

a) le protocole BSC
b) le protocole HDLC
c) le protocole IEEE-488
d) aucun d'entre eux

---

6 / La commutation par paquets (packet switching) sert à :

a) commuter deux signaux sur un réseau
b) envoyer à destination un message en suivant des voies différentes
c) échanger un message
d) exclure un message

---

7 / Un réseau public permet :

a) uniquement l'usage du protocole IEEE-488
b) uniquement la transmission de données dans une zone limitée
c) la liaison de dispositifs géographiquement distants
d) l'emploi de cables coaxiaux

8 / L'architecture en bus est employée dans :

a) les réseaux publics
b) les réseaux locaux
c) le protocole X3
d) les connexions sérielles

Les solutions du test se trouvent en page 1367

Instructions du Basic pour la communication de données

Même en Basic — le langage le plus répandu en micro-informatique — on peut gérer les communications de données. Les intructions disponibles sont toutefois limitées en nombre et toujours étroitement liées au type de matériel. Autrement dit, il n'existe pas d'instructions standard et, par la suite, nous présenterons, à titre d'exemple, certaines instructions capables de résoudre des problèmes de communication asynchrone sur RS232.

La mise en route d'une communication traverse les principales phases suivantes :

— ouverture du canal avec une instruction du type OPEN,
— lecture ou écriture de caractères sur le canal précédemment ouvert.

La logique suivie consiste à assimiler le canal de communication à un fichier et donc à réaliser les opérations d'E/S comme s'il s'agissait d'un accès à un fichier.

L'instruction OPEN réserve un tampon d'E/S considéré comme un fichier. En général, les paramètres à fournir sont :

N Numéro du canal. Il correspond parfois au n° d'une porte d'E/S ou d'un adaptateur.

V Vitesse de transmission (bits par seconde : bps). On prévoit en général les vitesses de 75, 110, 150, 300, 600, 1 200, 1 800, 2 400, 4 800, 9 600 bps, la plus courante étant 1 200 bps.

P Parité. Elle indique la méthode employée pour contrôler la parité du caractère.

Il existe plusieurs méthodes :
— absence de contrôle ; le bit de parité n'est pas transmis
— absence de contrôle ; bit de parité toujours transmis avec la valeur 0
— absence de contrôle ; bit de parité toujours transmis avec la valeur 1
— contrôle de parité paire
— contrôle de parité impaire.

Ce choix peut sembler trop large car il inclut des cas que l'on rencontre rarement dans la réalité. En fait, il faut dire que bien souvent on doit « interfacer » deux ou plusieurs appareils de différents constructeurs gérés par des systèmes et programmés avec des langages différents. Dans ce cas, la variété des interfaces est une condition essentielle pour établir la connexion.

D Nombre de bits constituant un caractère (bit de parité exclu). Les valeurs admises sont 4, 5, 6, 7, 8 ; le plus courant est 7.

S Nombre de bits de stop ; les valeurs les plus employées sont 1 et 2.

F Numéro du fichier logique associé au canal, à utiliser dans les opérations d'E/S.

C Nombre maximum d'octets pouvant être contenus dans le tampon de communication.

Par exemple l'instruction :

OPEN 'COM 3 : 1200, S,7,1' AS 1,128

ouvre un fichier de communication avec les paramètres suivants :

— numéro du canal : 3
— vitesse de transmission : 1200 bps
— bit de parité toujours à 1 (on suppose qu'il est possible d'indiquer cette option avec la lettre S ; en réalité il faut se référer au manuel d'utilisation de l'ordinateur)
— nombre de bits composant un caractère : 7
— nombre de bits de stop : 1

Le mot réservé AS suivi des paramètres 1,128 assigne ce canal au fichier 1 avec un tampon de 128 caractères.

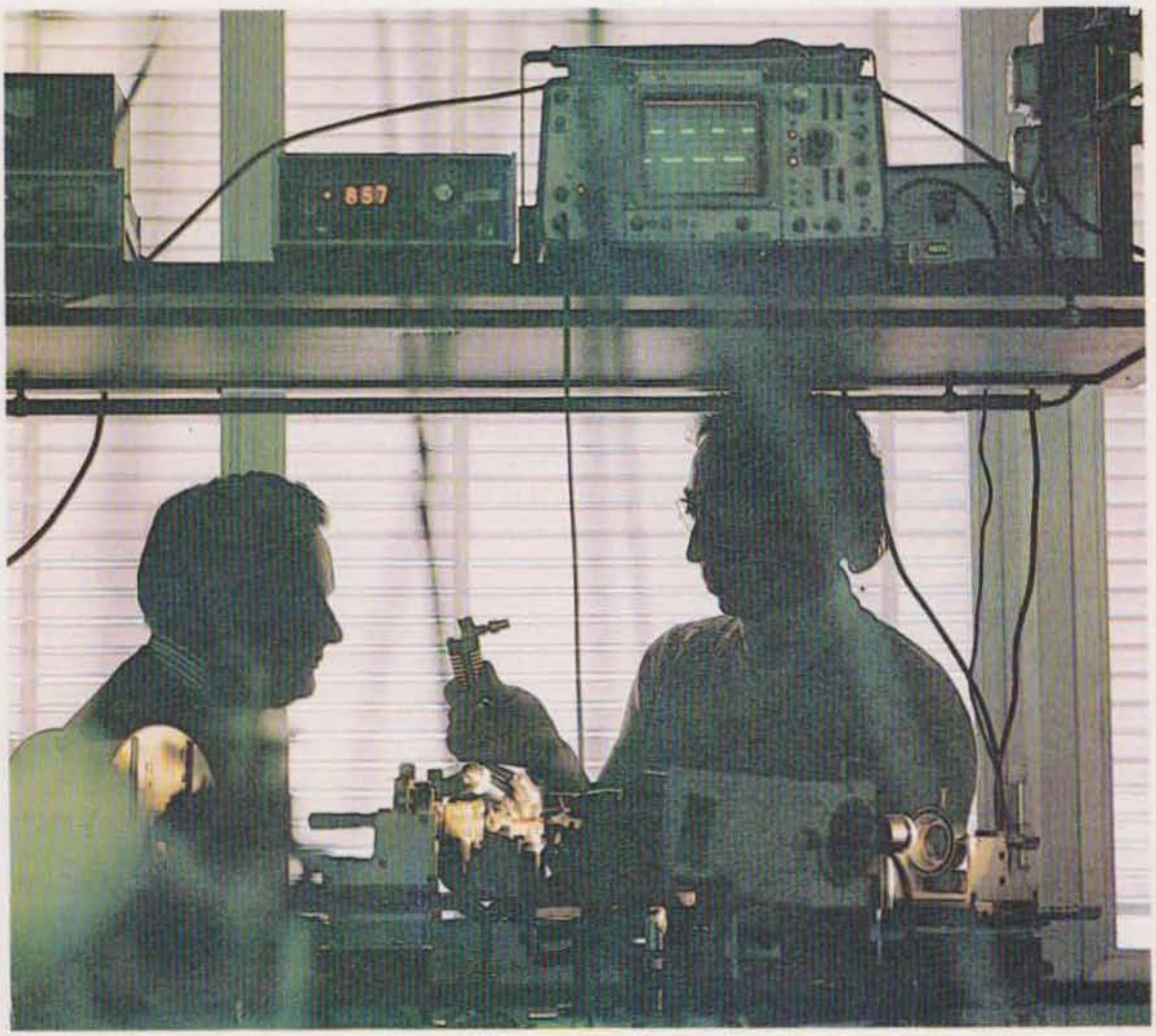
En plus de ces paramètres principaux, il faut aussi, dans certains cas, gérer l'état des signaux. En particulier, si on envisage de placer un modem dans la ligne, il sera nécessaire de disposer d'indicateurs spécifiant l'état des lignes (actif/non actif).

Les principales lignes employées dans ce type de communication sont :

— Request to Send
— Clear to Send
— Data Set Ready
— Carrier Detector.

Leur emploi dépend du type de matériel et du logiciel. Dans certaines applications, on peut, par exemple, employer ces lignes pour générer des interruptions et lancer un sous-programme de saisie d'un caractère. Une fois les paramètres définis, le logiciel déclenche une boucle d'attente, au cours de laquelle le processeur examine continuellement l'état d'un signal indiquant une liaison active (par exemple le Carrier Detector).

Conduction de lumière laser dans un faisceau de fibres optiques.



Italtel

Cette boucle n'est abandonnée que dans deux occasions :

— après une interruption
— après arrêt du signal.

L'interruption indique l'arrivée d'un caractère. La boucle est provisoirement abandonnée et la routine de saisie est mise en route. A la fin, on reprend le cours du programme au point où il a été interrompu (la boucle d'attente est donc relancée) jusqu'à l'arrivée du caractère suivant ou à la fin du signal de contrôle. Dans ce dernier cas (fin de la liaison), le programme s'arrête.
Ce type de gestion n'est pas aussi simple qu'il paraît. Tout d'abord, il ne fonctionne que sur certaines machines et dans certaines configurations ; de plus, au niveau du logiciel il est rendu encore plus complexe par toute une série de contrôles et de vérifications. Les principaux contrôles à effectuer sont les suivants :

— time out (temps de débouclement)
— remplissage du tampon
— parité
— priorité des interruptions
— fin de la ligne ou autres interruptions accidentelles.

On parle de **time out** lorsque, entre eux deux caractères successifs, l'intervalle de temps écoulé est supérieur au maximum prévu : le logiciel doit alors être en mesure de répéter cette situation anormale, et d'y remédier. Normalement, on envoie un message de diagnostic et on interrompt la liaison : il peut arriver que le dépassement ne soit pas dû à une situation anormale, mais simplement à une vitesse de transmission mal adaptée. Dans ce cas, c'est le logiciel qui résout ce défaut.

Le **remplissage du tampon,** surtout en Basic interprété, se produit quand le programme doit faire trop de traitements dans le sous-programme de saisie. Dans cette routine, il faut désamorcer l'interruption, sinon, à l'arrivée d'un caractère, la routine s'auto-appellerait, éventuellement plusieurs fois. La routine ne peut alors être interrompue et, si son déroulement prend trop de temps, les caractères qui continuent d'arriver font déborder le tampon. D'où génération d'une erreur ou, pire, un programme faussé.
Une manière d'éviter cet inconvénient consiste à réduire au minimum les traitements de la routine de saisie, mais même cette solution peut présenter des risques ; ainsi, en éliminant les contrôles, on peut accepter comme donnée valable une valeur erronée. Il existe toutefois des interfaces capables d'effectuer le contrôle sur les caractères reçus. En cas d'erreur, ces interfaces envoyent à l'ordinateur un code spécial à interpréter dans la routine de saisie (dans laquelle il faut toujours insérer les principaux contrôles).

Le **contrôle de parité** constitue un autre point critique du programme de gestion des communications. Normalement, le contrôle est réalisé par le matériel de manière totalement transparente. On informe le programme de l'existence d'une erreur avec un code particulier et l'activation d'un signal.
Dans ce dernier cas, en plus de la ligne pour l'arrivée des données, il faut activer une autre ligne d'interruption avec les complications qui peuvent en découler par rapport à la gestion des priorités.
Au moment de la génération, les **interruptions provoquent une réservation de service.** Il se crée donc une file de demandes qui s'amenuise au fur et à mesure que le système répond à celles qui ont la plus forte priorité. Le déroulement, complètement asynchrone, n'est nullement lié aux attentes de l'ordinateur. La mise en file des diverses interruptions n'a donc pas d'influence sur le flux des données en arrivée, qui s'accumulent dans le tampon. Si le système est dédié à des tâches prioritaires, il ne peut prélever les données du tampon (avec la routine de saisie) ; il se produira un phénomène analogue à celui de la saturation du tampon à cause de la lenteur excessive du traitement. Ce défaut est encore plus difficile à surmonter, car il s'agit d'une cause externe au flux normal du programme. Il faut alors assigner, à l'interface de communication, la priorité maximale.
Il faut enfin examiner l'éventualité d'une **panne** bloquant le signal au niveau actif, même quand la transmission est terminée. Dans ce cas, la boucle de saisie, utilisant ce signal comme indicateur d'état, ne s'interromprait jamais

Solutions du test 23

1 / b : la fonction de handshake sert à synchroniser l'ordinateur émetteur et l'ordinateur récepteur.

2 / b : les caractères SYN sont employés dans le protocole BSC

3 / c : dans le protocole HDLC le texte transmis peut être quelconque

4 / b : le protocole IEEE-488 est parallèle

5 / c : la ligne ATN (attention) est employée dans le protocole IEEE-488

6 / b : la commutation par paquets (packet switching) sert à envoyer à destination un message en suivant des voies différentes

7 / c : un réseau public permet la liaison de dispositifs géographiquement distants

8 / b : l'architecture en bus est employée dans les réseaux locaux

et l'ordinateur resterait en attente sans fournir aucun message de diagnostic.
Cette éventualité est à introduire dans la gestion du time out, en prévoyant un laps de temps opportun, bien supérieur à celui nécessaire à la réception d'un caractère ; le programme s'arrêtera alors ensuite, quoi qu'il arrive (par time out).
Pour gérer correctement une interface de communication, il faut donc disposer d'un temporisateur, une horloge capable de signaler le temps passé. Les instructions liées à la question de cette minuterie ont une syntaxe dépendant du matériel.
Une formule très employée est :

TIME$-A$ pour définir le temps
B$-TIME$ pour définir la lecture du temps.

En général, la valeur temporelle est exprimée comme une chaîne de 6 caractères, avec heure, minutes et secondes, dans le format HH MM SS. La première instruction définit la valeur indiquée en A$, la seconde lit la valeur courante comme différence entre la valeur définie et la valeur lue.
La gestion du time out s'obtient en contrôlant le temps passé dans la routine de saisie ; s'il est supérieur à une valeur maximum fixée, on met en action un indicateur d'erreur et on arrête le programme ; sinon on met à zéro le temporisateur pour qu'il soit prêt au contrôle suivant.
On mesure ainsi l'intervalle de temps entre l'arrivée d'un caractère et le suivant ; mais ce contrôle ne couvre pas toutes les possibilités de time out. Le programmeur devra donc déterminer quels sont les autres points critiques du programme et prévoir des contrôles.
Malgré ses limites, le Basic est encore le seul langage (en dehors de l'Assembleur) à permettre ce type d'applications dans la catégorie des micro-ordinateurs.
Les autres langages ne prévoient en général aucune instruction pour la communication de données.

Systèmes d'exploitation

Le fonctionnement d'une application développée sur micro est en général guidée par l'interpréteur Basic. Les calculs, les entrées et les sorties de données sont préparées dans le programme qui offre à l'utilisateur un moyen d'exploiter de manière efficace la machine.
En généralisant ce concept, on arrive à la définition du **système d'exploitation :** un ensemble de programmes et de procédures qui permet d'exploiter l'ensemble des ressources de l'ordinateur.
Même quand il s'agit d'un ordinateur personnel, le Basic ne suffira pas toujours, à lui seul, pour en coordonner toute l'activité. Il faut donc utiliser des systèmes d'exploitation comme le CP/M, le MPM, etc.
En général, plusieurs systèmes d'exploitation peuvent résider en mémoire d'ordinateur et leur intervention est liée aux nécessités de l'application déroulée.
L'utilisateur qui crée des programmes faisant appel à beaucoup de fichiers choisira, par exemple, un système d'exploitation orienté multiprogrammation.
Si, en théorie, il est toujours possible de mettre un quelconque système d'exploitation sur un matériel, dans la pratique l'ordinateur est, à l'origine, doté d'un système d'exploitation mis au point par le constructeur. C'est ce système qui épouse d'ailleurs le mieux la partie matériel et qui en optimise l'emploi.
Dans la définition d'un système d'exploitation, on a évoqué les **ressources.** Ce terme fait référence à n'importe quelle caractéristique d'un système de traitement tel que la mémoire centrale, les mémoires de masse, le temps machine...
Le système d'exploitation agit comme coordinateur de ces ressources.
Pour expliquer son fonctionnement, on fera constamment référence au système Unix qui a été mis au point par les laboratoires Bell. Unix s'est très vite fait une place dans les universités américaines par sa facilité d'emploi et par son caractère convivial. A l'heure actuelle, tous les fabricants d'ordinateurs ont tendance à doter leur machine de ce système.

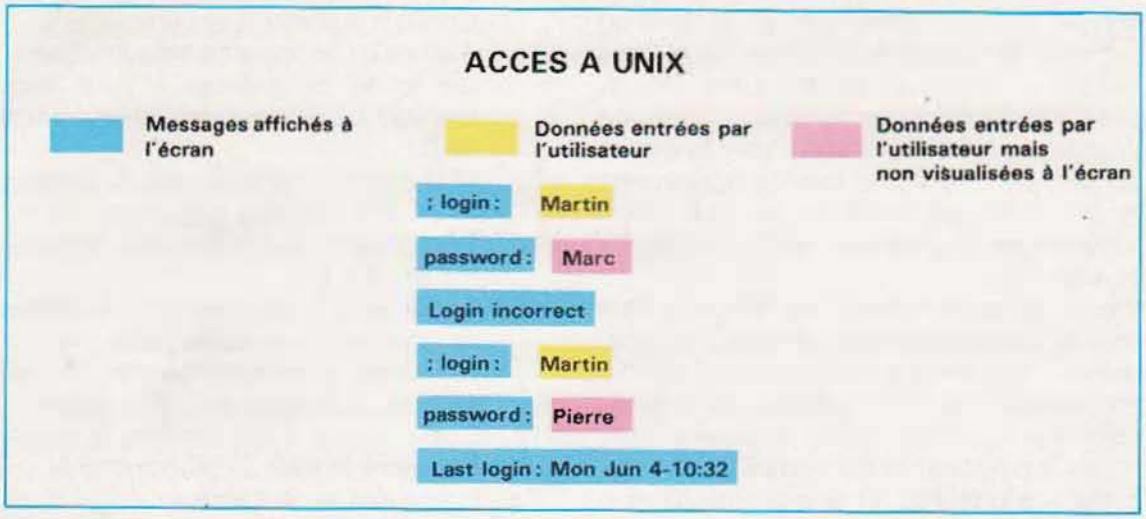
Mono-utilisateur et multi-utilisateur

Ces caractéristiques indiquent combien de personnes peuvent se servir simultanément d'un ordinateur. Aux points d'accès correspondent les écrans et les claviers :

— En **mono-utilisateur,** on dispose d'un seul terminal, qui correspond à la machine elle-même.

— Le **multi-utilisateur,** constitue, en revanche, une caractéristique typique des mini-ordinateurs et des gros ordinateurs.

Les systèmes d'exploitation coordonnent l'accès à l'ordinateur des divers utilisateurs, en répartissant entre eux le temps-machine, la mémoire, les fichiers et en n'acceptant que les personnes dûment autorisées.
Le nombre d'usagers en titre est défini, et l'accès est lié à l'introduction d'un mot de passe confidentiel.
L'accès **(login)** à un ordinateur géré par Unix est représenté schématiquement dans la figure ci-contre.
L'utilisateur frappe sur une touche quelconque et le système répond avec la chaîne « ;login : ». Il signe en introduisant son nom et le système demande son mot de passe.
Dans l'ordinateur, on a déjà défini le mot de passe pour « Martin » qui correspond au mot « Pierre » ; si, comme mot de passe, on introduit « Marc », Unix refuse l'accès en visualisant le message « Login incorrect ».
En retapant une touche quelconque, « ;login : » apparaîtra à nouveau comme une invitation à s'identifier.
Si le mot de passe est correct, il a accès au système ; Unix visualisera la date à laquelle « Martin » a fait la dernière connexion et le caractère qui signifie « je suis prêt à accepter les instructions de Martin ».
Les codes confidentiels (mots de passe) introduits ne sont pas visualisés, pour éviter que des usagers non autorisés n'en prennent connaissance.



Traitement par lots et interactivité

Les systèmes d'exploitation réalisant du **traitement par lots** ont, pendant longtemps, été les seuls outils à la disposition des utilisateurs. Exécutant une seule tâche à la fois, ils sont désormais désuets. Les dispositifs d'entrée étaient constitués de lecteurs de cartes perforées grâce auxquelles programmes et données à traiter étaient chargés. Les divers travaux à exécuter par la machine étaient organisés en file d'attente. Le système d'exploitation les traitait l'un après l'autre, selon la technique FIFO First In/First Out (premier entré/premier sorti) : la priorité était accordée au premier travail chargé en mémoire centrale.

A l'inverse, les systèmes d'exploitation dits « interactifs » travaillent en temps réel. Les terminaux sont donc des dispositifs interactifs d'entrée-sortie. Un caractère (ou une chaîne de caractères) indique à l'écran que la ligne utilisateur est ouverte. Avec le système interactif Unix, ce caractère est le symbole dollar ($). Les ordinateurs personnels sont, eux aussi, commandés par des processus interactifs tels que Basic ou CP/M. Dans un ordinateur, c'est le système d'exploitation (à traitement par lots ou interactif) qui a le contrôle des opérations.

Imaginons, à titre d'exemple, qu'un programme d'application exécute une opération interdite : le système d'exploitation doit la détecter, la signaler et reprendre le contrôle. Dans un système de traitement par lots, cela signifie abandonner le travail en cours et passer au suivant ; dans le cas d'un système interactif, c'est afficher le caractère guide-opérateur.

Multiprogrammation

La multiprogrammation n'existe pas dans toutes les machines, mais c'est une caractéristique commune à tous les systèmes d'exploitation dits « multi-utilisateurs », ce qui permet de faire exécuter simultanément différents programmes.

La **multiprogrammation** a été conçue, à l'origine, pour exploiter les temps morts de la machine. Supposons qu'un opérateur soit en train d'entrer des données à partir d'un terminal. L'exécution du programme n'interviendra qu'en fin de saisie. Or le temps de saisie (quelques secondes) est en fait extrêmement long en comparaison des temps de traitement des ordinateurs. Si le système d'exploitation le permet, la machine peut alors exécuter un autre programme pendant que l'opérateur entre ses données.

La multiprogrammation est donc la capacité d'un système d'exploitation à gérer simultanément plusieurs programmes résidents en mémoire d'ordinateur. Mais gérer ne veut pas dire exécuter simultanément. En effet, si l'ordinateur n'a qu'une seule unité centrale (UC), celle-ci ne peut, à un instant donné, s'occuper que d'une seule tâche. Il faut comprendre la gestion simultanée comme la capacité d'exploiter les temps morts en suspendant temporairement l'exécution d'un programme pour donner la possibilité à un autre d'être exécuté.

Plusieurs programmes en machine demandent généralement à être exécutés en même temps. C'est pourquoi les systèmes d'exploitation

orientés multiprogrammation doivent gérer un ordre de priorité des programmes. Supposons que cet ordre soit un nombre entier compris entre 1 et 99 et que le niveau de priorité soit décroissant : les programmes ayant la priorité 20 sont plus importants que les programmes de priorité 40, qui sont à leur tour plus importants que les programmes ayant la priorité 60, et ainsi de suite.

Ces programmes forment une queue souvent appelée **liste d'ordonnancement**. Pour chaque cas, le système d'exploitation doit décider quel programme il doit exécuter en premier, comme le montre le schéma ci-dessous : parmi les 3 programmes demandant à être exécutés, A a la priorité 30, B la priorité 20, et C la priorité 10.

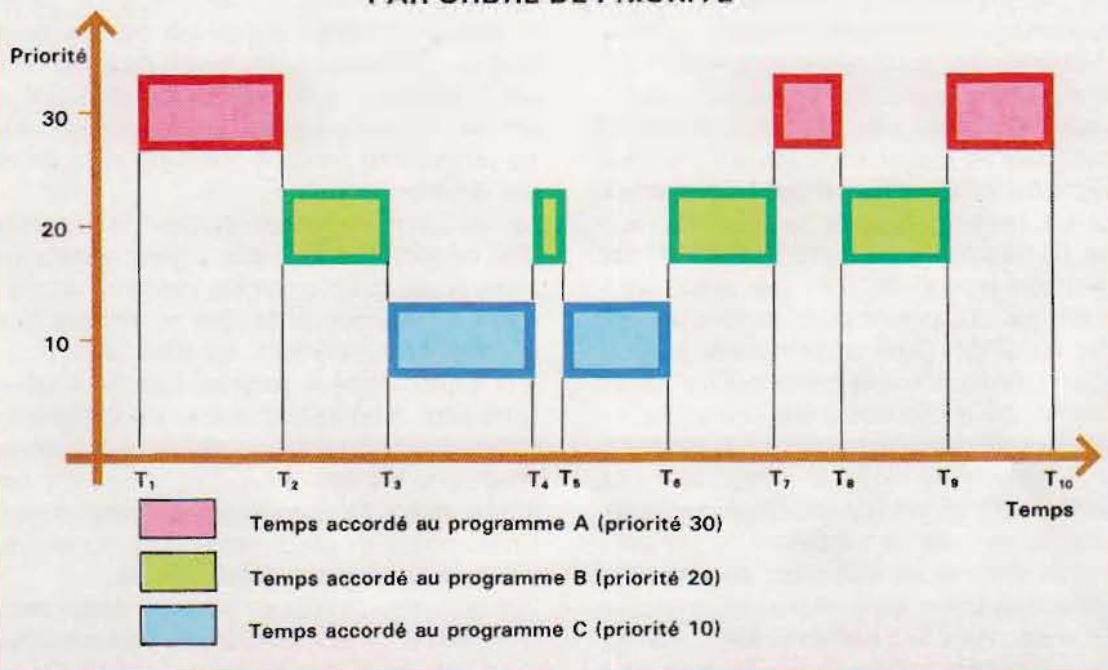
Voici la description de ce qui se passe entre T_1 et T_{10} :

- T_1 La machine exécute d'abord A, le seul programme résident à cet instant-là.
- T_2 Le programme B (plus prioritaire que A) demande à être exécuté ; le système d'exploitation suspend A et lance B.
- T_3 Le programme C (plus important que B) demande à être traité ; le système d'exploitation suspend B et commence C.
- T_4 C effectue une opération lente (par exemple entrée de données à partir d'un terminal) ; le système reprend l'exécution de B.
- T_5 La phase lente de l'exécution de C prend fin ; B est suspendu et C repris.
- T_6 L'exécution de C est terminée ; le système passe à B.
- T_7 B exécute une opération d'entrée/sortie de données ; le système passe à A.
- T_8 La phase d'entrée/sortie de B se termine ; A est suspendu et B reprend.
- T_9 L'exécution de B est terminée, le système donne la main au programme A.
- T_{10} L'exécution de A s'achève.

Ainsi, la multiprogrammation par ordre de priorité ne fait pas qu'exploiter les temps morts (l'activité de l'UC ne cesse à aucun moment) ; elle permet également d'établir une hiérarchie des programmes d'après leur importance. L'exemple donné pourrait correspondre à un ordinateur relié à trois terminaux exécutant chacun un programme A, B ou C.

Que se passe-t-il quand deux programmes ont la même priorité, 50 par exemple, l'un (X) étant

MULTIPROGRAMMATION : EXECUTION DE TROIS PROGRAMMES PAR ORDRE DE PRIORITE



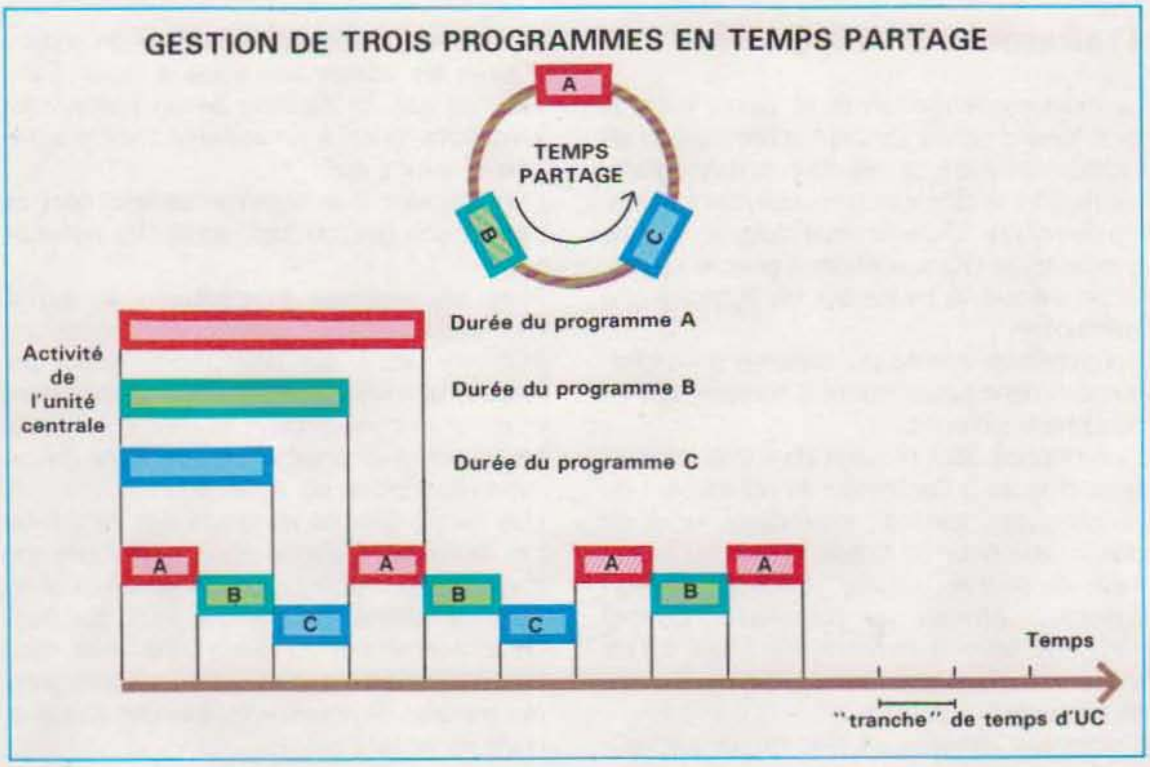
en cours d'exécution, l'autre en (Y) en attente ? Deux solutions sont envisageables :

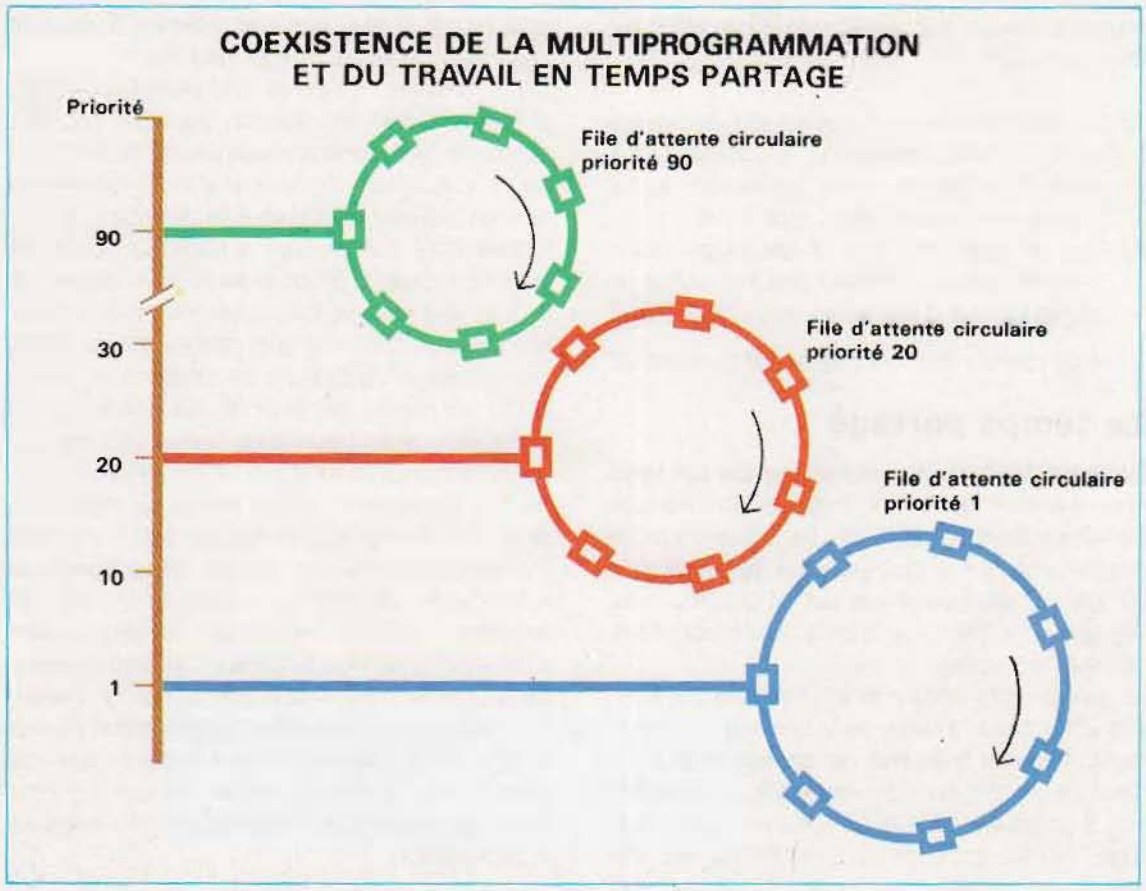
1/ Le programme X doit être terminé avant qu'X ne soit commencé, à moins que X n'ait à effectuer une opération lente, auquel cas Y peut alors être traité.
2/ Les programmes X et Y seront exécutés alternativement pendant des intervalles de durée définie à l'avance.

C'est ce dernier cas qui est décrit ci-dessous.

Le temps partagé

Exécuter des programmes en **temps partagé** revient à affecter, à chacun d'eux, un intervalle de temps bien précis. L'UC ne travaille sur un programme donné que pendant le temps prévu. Elle passe ensuite aux autres programmes, l'un après l'autre, jusqu'à ce qu'elle recommence alors le cycle.
La gestion des programmes est effectuée cycliquement sur la base de la période de traitement, appelée **tranche de temps d'UC.**
Dans l'exemple illustré ci-dessous, nous avons pris 3 programmes : A, B et C, en temps partagé. Aucun concept de priorité ne leur est attaché, A, B et C ont « les mêmes droits » et l'UC s'en occupera à tour de rôle.
Dans certains systèmes d'exploitation, multiprogrammation et temps partagé peuvent coexister (voir schéma page suivante). En théorie, il y a autant de files d'attente circulaires que de valeurs affectées à la priorité.
L'ordinateur servira tout d'abord la queue de priorité 1, puis la 20, et enfin la 90. L'utilisateur a, à sa disposition, les moyens de hiérarchiser les niveaux d'intérêt des programmes. Ainsi, un groupe d'utilisateurs se divisera le temps d'UC au niveau de priorité 50, tandis qu'un autre groupe se placera au niveau 20, etc.
Le système d'exploitation UNIX fonctionne en multiprogrammation et en temps partagé, mais la priorité d'un programme n'est pas établie par l'utilisateur. Au début de leur exécution, tous les programmes sont au niveau de priorité, par exemple 10. Plus l'exécution du programme s'avère longue, plus UNIX augmentera la valeur de la priorité (en « déclassant » le programme). Ce mécanisme pénalise les programmes qui durent trop longtemps et favorise ceux qui sont courts. L'objectif recherché est d'encourager au maximum l'interactivité du système d'exploitation.





Traitement en temps réel

La multiprogrammation et le temps partagé sont fondés sur le concept d'interruption de l'activité en cours. Si, pendant qu'un programme de priorité 50 s'exécute, un autre programme de priorité 40 demande la main, le système d'exploitation doit suspendre le premier au profit du second. A ce dernier est associée une interruption.

L'architecture interne du système d'exploitation doit donc lui permettre d'accepter des interruptions externes.

L'interruption peut provenir de n'importe quel point d'accès à l'ordinateur et notamment du périphérique : terminal, imprimante, unité de disque, dérouleur de bande mais aussi instrument de mesure, capteur. Dans le cas d'un ordinateur pilotant un processus industriel (raffinerie, usine automobile, etc.), des câbles transportent les signaux électriques provenant de capteurs.

L'ordinateur décidera s'il doit fermer une vanne, tourner un moteur, ou activer un relais... d'après les valeurs des signaux reçus. Dans tous les cas, la décision de l'ordinateur est immédiate, grâce à un système d'exploitation orienté temps réel.

L'architecture d'un système **temps réel** lui permet une gestion très rapide des interruptions.

Avec les systèmes d'exploitation en temps réel, une interruption externe peut également être associée à des programmes écrits par l'utilisateur ; un cas typique est la fermeture d'un contact électrique à laquelle correspond l'exécution d'un programme de gestion de cet événement prévu par l'utilisateur.

Unix ne travaille pas en temps réel, ne prévoit pas de connexions avec des périphériques non classiques et n'offre pas la possibilité de gérer des interruptions. En revanche, il est idéal dans un environnement où divers utilisateurs désirent simultanément effectuer des calculs, gérer des banques de données ou travailler en traitement de texte.

Système de gestion des fichiers

Un gestionnaire de fichiers s'occupe de la gestion des mémoires de masse, donc plus particulièrement des disques par le système d'exploitation.

Tous les ordinateurs possèdent des mémoires non volatiles, car ce sont elles qui conservent les informations utiles. Ces mémoires contiennent également le système d'exploitation, qui est automatiquement chargé dans la mémoire centrale à la mise sous tension. Dans certains ordinateurs personnels, le système d'exploitation est résident sur disques durs et il en est extrait pour être chargé en mémoire selon des procédures particulières (dites de mise en route). Ces mêmes disques contiennent aussi des informations enregistrées par les utilisateurs ; c'est le système de gestion des fichiers qui a la charge de les organiser efficacement.

Les données enregistrées dans les mémoires de masse sont toujours regroupées en fichiers, et eux-mêmes subdivisés en enregistrements.

- **Classification d'après le contenu.** Les enregistrements contiennent des données en code ASCII (ou EBCDIC) ou binaire ; dans le premier cas, ce sont des programmes sources ou des textes, alors que dans le second, ce sont des résultats de compilations (fichiers tranférables), des programmes sous forme exécutable ou des fichiers de données résultats.
- **Classification d'après les modes d'accès.** Il s'agit des fichiers à accès séquentiel et des fichiers à accès direct. Dans le séquentiel, l'accès à un enregistrement passe par un « parcours » de tous les enregistrements précédents. En accès direct, l'adressage d'un enregistrement particulier se fait directement.
- **Classification d'après le type d'enregistrement.** Certains fichiers ont des enregistrements de longueur fixe, d'autres variable. La longueur fixe est caractéristique : par exemple, des programmes sources.

Système informatique composé d'un ordinateur personnel connecté à un ordinateur central par modem et ligne téléphonique.



Marka/Photri

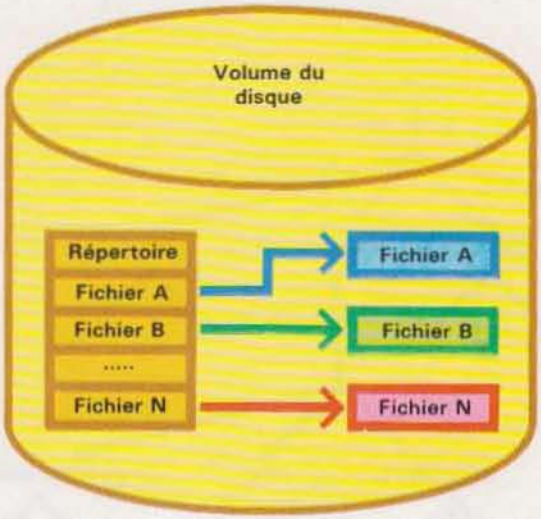
La mission du système d'exploitation est de gérer tous les types de fichiers et, dans un environnement multi-utilisateur, de garantir à X que ses fichiers ne seront pas modifiés ou effacés par Y.
Supposons qu'on se trouve en présence de 2 utilisateurs, l'un (X) écrivant un enregistrement, l'autre (Y) demandant à le lire. Les deux opérations devront être séparées de telle manière, que tant que X écrira, Y ne pourra pas lire.
L'association du concept de ressource à la quantité totale de mémoire des disques fait apparaître un autre aspect intéressant du gestionnaire de fichiers : la partition de cette mémoire entre utilisateurs.
Les mini-ordinateurs acceptent des disques d'une capacité globale de plusieurs milliers de méga-octets. Une fois définie la mémoire disponible, il faut décider de sa partition entre l'utilisateur. Une méthode consisterait à découper également l'espace mémoire. Mais elle ne satisfait pas à toutes les exigences, car certains utilisateurs auront besoin de plus d'espace que d'autres.
Les systèmes d'exploitation modernes allouent de l'espace sur disque au fur et à mesure qu'il devient nécessaire, en évitant toutefois qu'un seul utilisateur utilise tout. Aussi faudra-t-il imposer des limites précises à ne pas dépasser.
Une fois l'espace sur disque alloué, il reste à donner, à l'utilisateur, la possibilité de retrouver ses fichiers de façon rapide et transparente. C'est là qu'intervient le concept de répertoire (directory) des volumes composant le disque.
Un volume est une division logique (et non pas physique) du disque, couvrant plusieurs applications ou commun à plusieurs utilisateurs. Autrement dit, on préfère « découper » le disque en plusieurs parties pour permettre à chaque utilisateur d'être indépendant et de ne subir aucune interférence de la part des autres utilisateurs. Cette division en volumes peut également être plus élaborée.
Pour le moment, considérons un volume comme une partie du disque affecté à un utilisateur et possédant au moins un répertoire. Le répertoire est une zone du disque dans laquelle sont reportés tous les noms des fichiers enregistrés, des informations concernant la structure des fichiers et des enregistrements, et surtout l'adresse de chaque fichier sur le disque (illustration ci-contre).

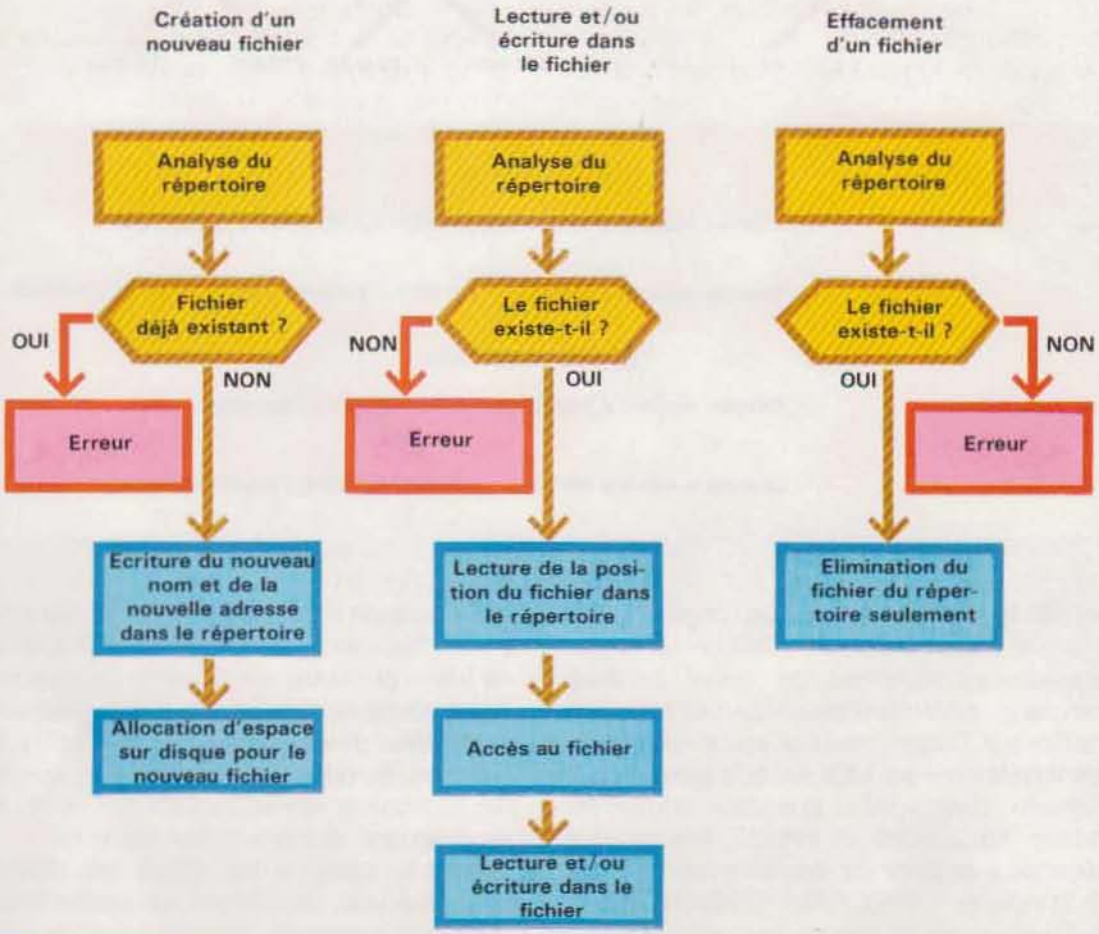
Chaque fois qu'un nouveau fichier est créé, il est enregistré dans le répertoire puis dans la zone du disque qui le contient physiquement.
Si d'autres informations sont à introduire dans le nouveau fichier, le système d'exploitation consulte le répertoire puis, une fois pointé le fichier en question, il détermine sa position sur le disque et y accède en mode direct. Même procédure pour la lecture d'un fichier.
Enfin, si l'on désire savoir quels fichiers sont enregistrés sur le disque, il suffit de lire le répertoire et de l'imprimer.

Structure du gestionnaire fichiers Unix

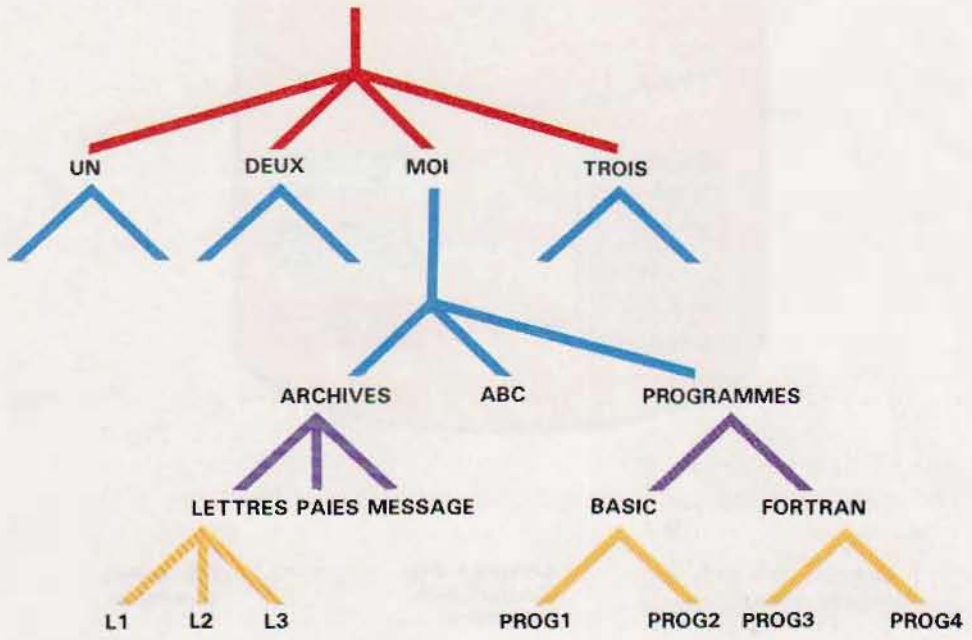
Dans le système d'exploitation Unix, le concept de répertoire est étendu : en effet, le disque contient des fichiers et des répertoires reliés entre eux. Un répertoire est considéré par Unix comme un fichier qui peut renvoyer à d'autres fichiers ou répertoires.
Le répertoire d'arrivée, le **sous-répertoire**, peut à son tour aiguiller vers d'autres sous-répertoires ou d'autres fichiers, et ainsi de suite. La structure du système de gestion de fichiers est donc arborescente, comme le montre le schéma de la page suivante.
Le symbole (barre oblique) désigne, dans Unix, le répertoire racine d'où partent toutes les branches ; de celui-ci partent, par exemple, les répertoires UN, DEUX, MOI, TROIS. Un répertoire se reconnaît au fait qu'il donne naissance à au moins une ramification (structure p. 1376). Le répertoire MOI mène à deux sous-répertoires (ARCHIVES et PROGRAMMES) et à un fichier (ABC) ; ARCHIVES conduit à un autre sous-répertoire (LETTRES) et à deux fichiers (PAIE et MESSAGE). LETTRES mène à 3 fichiers (L1, L2 et L3). Le même type de structure se retrouve à partir de PROGRAMMES.
Dans Unix, un fichier est une extrémité, ou plutôt un module, d'où rien ne part (une feuille de l'arbre), tandis qu'un répertoire génère une structure permettant de regrouper d'un point de vue logique d'autres éléments (ARCHIVES contiendra des fichiers de données, PROGRAMMES contiendra des programmes).
Pour accéder à un fichier, il faut indiquer à Unix un parcours, et un seul, à suivre depuis le répertoire-racine jusqu'au fichier final. Dans la construction de ce parcours univoque, la barre oblique sert à séparer les répertoires, à l'excep-

ACCES AU DISQUE PAR L'INTERMEDIAIRE DU REPERTOIRE





STRUCTURE DU SYSTEME DE GESTION DES FICHIERS ET VOIES D'ACCES AUX FICHIERS UNIX



En partant de /

Chemin menant à MESSAGE : / MOI / ARCHIVES / MESSAGE

Chemin menant à PROG2 : / MOI / PROGRAMMES / BASIC / PROG2

En partant de / MOI

Chemin menant à MESSAGE : ARCHIVES / MESSAGE

Chemin menant à PROG2 : PROGRAMMES / BASIC / PROG2

tion de la première qui indique l'origine, c'est-à-dire le répertoire-racine. Chaque utilisateur possède un répertoire de travail (working directory). Si le répertoire MOI est affecté à l'utilisateur Dupont, celui-ci est automatiquement positionné sur MOI dès qu'il entre dans le système. Cela signifie, que pour trouver les fichiers MESSAGES et PROG2, il n'est plus nécessaire de partir du répertoire racine. Il suffit d'indiquer : PROGRAMMES/BASIC/PROG 2. Dans ce cas, le chemin ne commence pas par / puisque on n'est pas parti du répertoire racine, mais du sous-répertoire MOI (répertoire de travail de l'utilisateur Dupont). Un répertoire de travail se modifie à l'aide d'une commande du système d'exploitation ; Dupont pourra, par exemple, se positionner directement sur BASIC et accéder aux fichiers PROG1 et PROG2 en indiquant uniquement leur nom.

Ci-contre, nous avons figuré un dialogue utilisateur/Unix dans lequel apparaissent quelques commandes en interaction avec le systè-

me de gestion de fichiers. Pour savoir dans quel répertoire il se situe, l'utilisateur introduit la commande **pwd** (print working directory). Le système répond en décrivant le chemin qui, partant du répertoire-racine, mène au répertoire de travail.

La commande **is** visualise les noms des fichiers du répertoire de travail (pour /MOI, il n'y a qu'ABC, puisque ARCHIVES et PROGRAMMES sont également des répertoires). Pour connaître tous les noms de fichiers et de répertoires auxquels mène/MOI, il suffit d'ajouter l'option **a** (all) derrière la commande **ls** (les options doivent être précédées du symbole -). La réponse du système d'exploitation contient tous les noms de fichiers auxquels mène /MOI. Avec la commande file, on peut savoir si le nom donné correspond à un répertoire ou à un fichier.

La commande **ed** (change directory) déplace le point de travail et, dans l'exemple ci-dessous, permet de passer directement au répertoire FORTRAN. Une même ligne accepte plusieurs commandes, séparées par ; (point-virgule). Ici, **ls-a** a été ajouté pour obtenir, à l'aide d'une seule commande, le changement de répertoire et la liste des fichiers et des répertoires contenus dans le nouveau répertoire.

La commande **cat** (catalog) permet d'accéder au contenu d'un fichier et, dans notre exemple, d'avoir la liste des instructions composant le programme PROG3. La commande **mkdir** (make directory) crée un nouveau répertoire, atteint en passant par FORTRAN (dans lequel on trouvera, par exemple, tous les utilitaires des programmes Fortran).

Après **mkdir**, la liste des noms auxquels mène FORTRAN contient désormais **SUBR** (sous-programmes), et la commande **file** indique qu'il s'agit d'un répertoire.

QUELQUES COMMANDES DU SYSTEME DE GESTION DES FICHIERS UNIX

Messages-écran

Données entrées par l'utilisateur

```
$ pwd
/MOI
$ ls
ABC
$ ls -a
SAUVEGARDES ABC PROGRAMMES
$ file ARCHIVES ABC PROGRAMMES
ARCHIVES:   directory
ABC:        ascii text
PROGRAMMES: directory
$ cd PROGRAMMES/FORTRAN; ls-a
PROG3 PROG4
$ cat PROG3
        (liste du programme PROG3)
$ mkdir SUBR
$ ls-a
PROG3 PROG4 SUBR
$ file SUBR
SUBR :      directory
$
```



Marka/Photri

Système de traitement de texte avec imprimante intégrée.

L'utilisateur peut protéger ses fichiers contre les effacements ou les modifications par d'autres utilisateurs ; pour cela, il dispose de commandes spéciales offrant des protections de deux types : en écriture ou en lecture. Si le fichier est protégé en écriture, seul son créateur peut l'écraser ou le modifier. Un autre utilisateur, positionné dans le répertoire en question à l'aide de la commande **cd**, pourra seulement lire son contenu.
Si le fichier est protégé en lecture, il ne pourra être lu par d'autres utilisateurs que s'ils possèdent la clé de lecture.
Unix accepte deux catégories de « lecteurs » : les **utilisateurs ordinaires** (users) et les **super-utilisateurs** (superusers). Les premiers se déplacent à l'intérieur des répertoires qui leur sont associés et, s'ils sautent à d'autres répertoires, ils ne sont en mesure de réaliser que des opérations autorisées par leurs créateurs (par exemple lire un fichier s'il est protégé en écriture) ; les seconds peuvent tout faire, en ce sens qu'ils ont la possibilité de modifier les mécanismes de protection des fichiers et des répertoires. Le super-utilisateur est généralement une personne qui reconfigure le système d'exploitation pour obtenir un bon environnement de travail multi-utilisateur, avec un minimum de coordination.

Processus simultanés

La question de la coordination des opérations lancées simultanément par plusieurs programmes se pose, en effet, dans tout système d'exploitation multi-utilisateur comme Unix.
En multiprogrammation et en temps partagé, il existe, nous l'avons vu, des méthodes pour déterminer quel programme doit disposer de l'UC et quels autres doivent attendre.
Il nous reste à expliquer les techniques qui vont permettre à ces programmes soit d'échanger des informations entre eux, soit d'interrompre leur déroulement quand une ressource n'est pas disponible (car déjà employée par un autre programme).
C'est là qu'intervient le concept de **processus,** vu par le système d'exploitation comme une séquence d'opérations à exécuter en une seule fois. Il peut s'agir d'une procédure, d'un sous-programme, ou même d'une instruction unique.

Les concepteurs de systèmes d'exploitation ont tendance à décomposer les opérations en opérations plus simples jusqu'à atteindre un niveau plancher, lequel correspond à une instruction-machine. De cette manière, on obtient un système d'exploitation composé de N modules reliés entre eux, chacun ayant la charge de piloter les opérations s'y rapportant. Le lien entre le processus et les opérations à effectuer dépend donc du système d'exploitation.

En multiprogrammation, un programme peut connaître un début d'exécution, il est ensuite suspendu par un autre programme de priorité plus élevée, puis repris, etc. Cet exemple fait alors apparaître deux états principaux d'un processus : il est **bloqué** ou **actif.** Quand un processus est bloqué, il attend qu'une ressource devienne disponible pour poursuivre ; quand il est actif, il utilise l'unité centrale.

Il existe plusieurs états de blocage dans les systèmes d'exploitation : un processus peut attendre qu'une opération d'entrée/sortie soit terminée, ou que la ressource unité centrale soit libérée par un autre processus. L'état actif, lui, n'a pas de subdivision.

C'est pourquoi la principale tâche d'un système d'exploitation consiste à coordonner les attentes des processus en définissant les transitions possibles entre un type de blocage et un autre, ou entre un état de blocage et un état actif. Les changements d'état sont pilotés par le système d'exploitation en fonction des types des processus mis en œuvre.

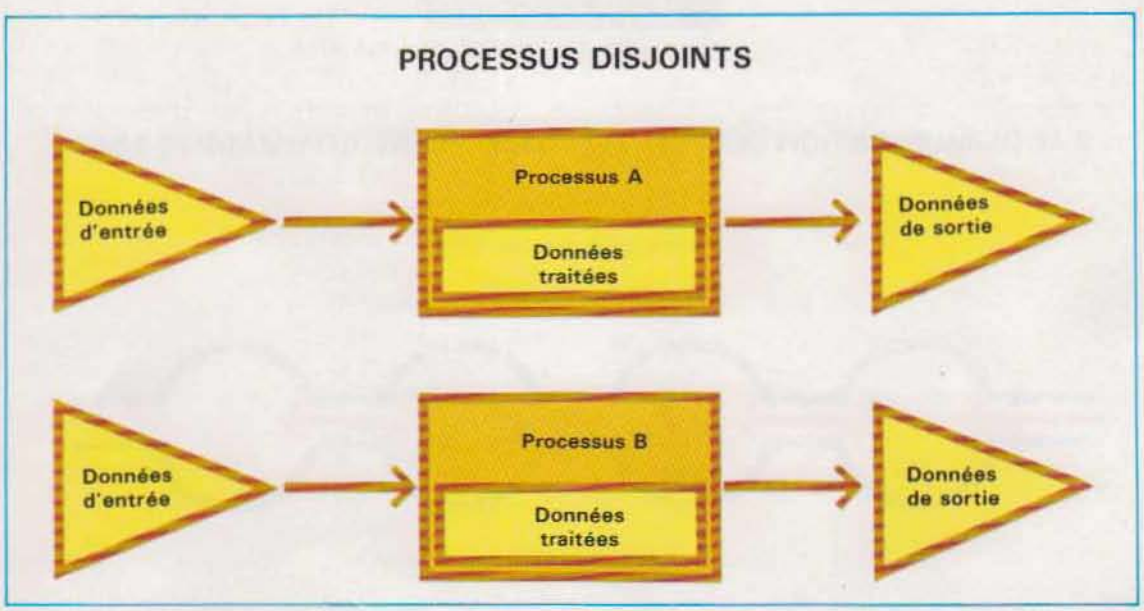
Les processus **simultanés** sont de deux ordres : **disjoints** et en **interaction.** Deux processus sont dits disjoints quand ils n'agissent pas sur la même base de données ; en interaction quand ils possèdent des variables communes.

Deux processus disjoints ont un comportement indépendant dans le temps : les résultats produits par le premier ne doivent pas être utilisés par le second ; le système d'exploitation ne fournit de service qu'à l'un ou à l'autre (schéma ci-dessous).

Deux processus en interaction fonctionnent différemment : les modalités d'accès aux données communes doivent être préalablement établies. Comme le montre la figure 1 (page suivante), ces données peuvent également être traitées simultanément au déroulement des processus, nécessitant, de ce fait, des mécanismes de synchronisation des processus.

Communication entre les processus

La forme la plus simple de communication entre processus est de type « historique », qui permet aux données de sortie d'un processus d'être utilisées en entrée par le processus suivant. Les commandes Unix autorisent ce mode de synchronisation, appelé **pipe-line,** en définissant les données de sortie d'une commande comme étant les données d'entrée d'une autre commande. Les deux commandes sont alors

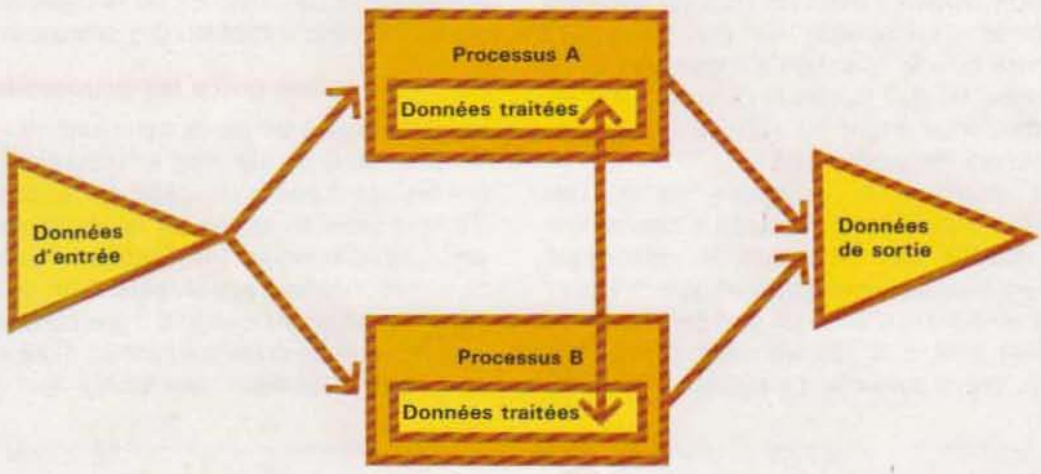


introduites de façon séquentielle, séparées par une barre verticale signifiant que les données doivent être enchaînées ou échangées en pipeline.
Ce concept reprend l'image d'un oléoduc auquel sont reliés des producteurs et des consommateurs de pétrole. Ici, le pétrole représente les données, alors que les producteurs et les consommateurs sont les commandes Unix.
Dans l'exemple illustré par le schéma 2, la 1re commande (ls-a) génère la liste des fichiers et des répertoires contenus dans le répertoire de travail, la 2e (grep rime) extrait toutes les lignes contenant le mot rime, la 3e (sort) trie ces lignes et les classe par ordre alphabétique, tandis que la 4e lance leur impression.
Dans le cas de la communication historique, le comportement dans le temps du processus est défini à l'avance. Quand cette programmation n'est pas réalisable, il faut alors résoudre deux problèmes :

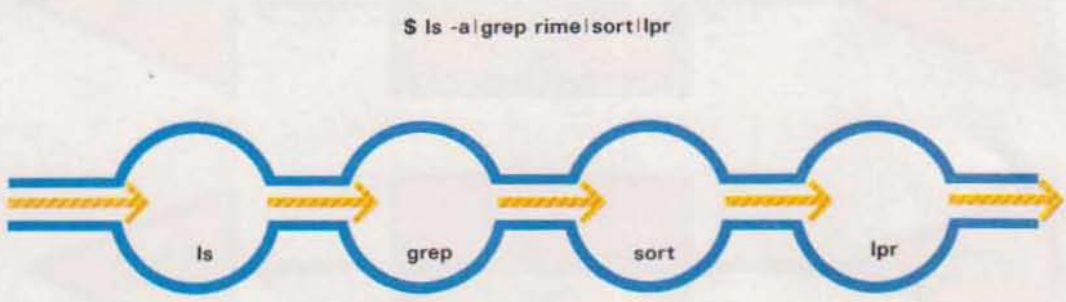
- l'accès aux données partagées
- l'échange des données entre processus

L'accès aux données partagées par un processus implique la possibilité de modification de ces données par le même processus. Dans ce cas, l'accès est interdit à tout autre processus

1 / PROCESSUS EN INTERACTION

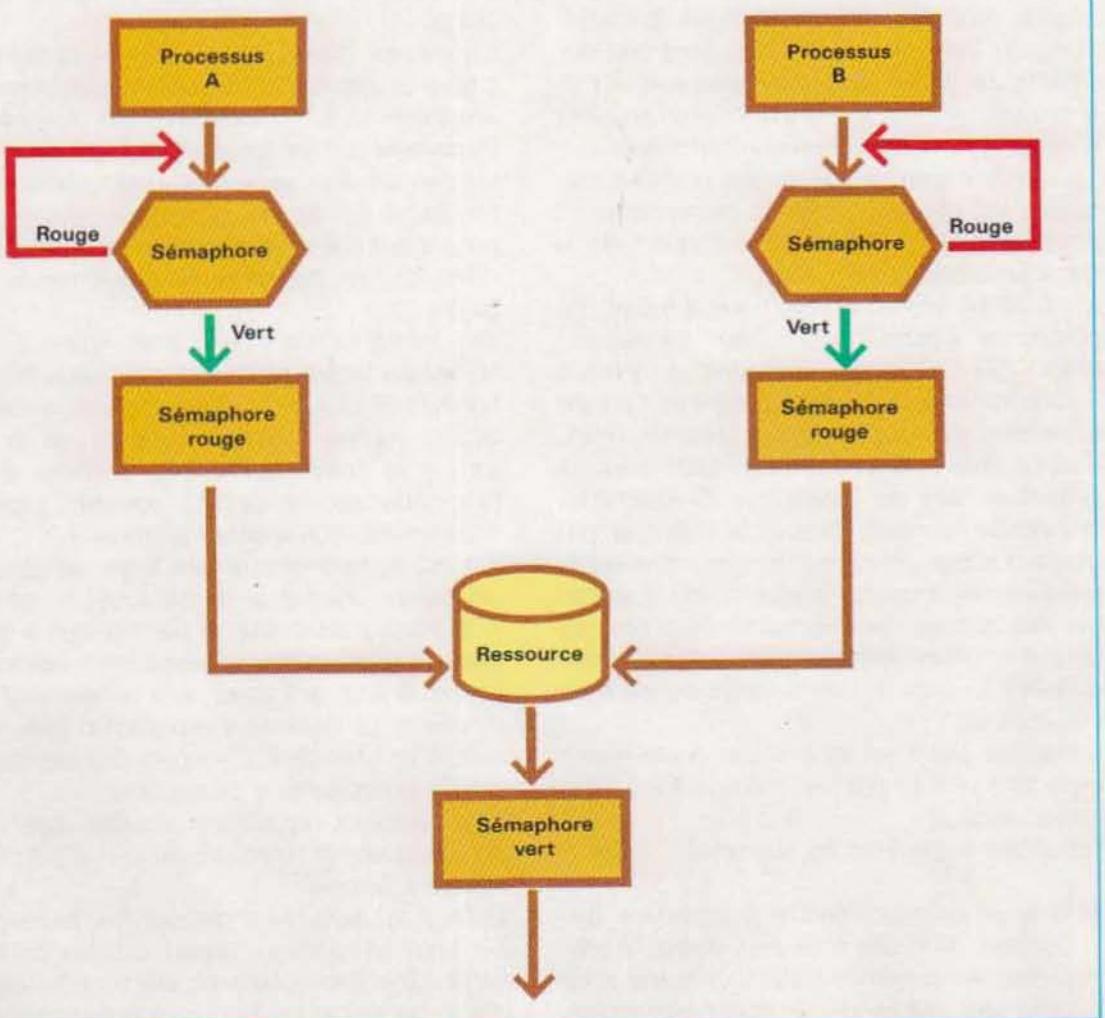


2 / COMMUNICATION DU TYPE PIPE-LINE ENTRE COMMANDES UNIX



avant la fin des opérations sur les données. La synchronisation est mise en œuvre par le jeu de « sémaphores » fournis par le système. Le processus désirant accéder aux données demande au système si le feu est vert, c'est-à-dire si les opérations précédentes sont terminées (voir schéma ci-dessous). Si le feu est rouge ce processus est provisoirement « endormi ».
L'emploi de sémaphores s'applique également aux autres ressources. Ils éviteront, par exemple, le mélange de sorties sur imprimante de processus simultanés. Ainsi, les processus désirant utiliser la ressource imprimante devront tout d'abord en demander l'accès en interrogeant les sémaphores. Il peut toutefois se produire des cas d'« étreinte fatale » (blocage complet du système), lorsque deux processus, ou plus, attendent indéfiniment un événement. Cette situation sera illustrée par l'exemple suivant : un processus A accède à un fichier sur disque ; pour le protéger, le système fait passer au rouge le feu relié à ce disque. Le même processus va ensuite demander l'intermédiaire d'accès au dérouleur de bande magnétique sur lequel il voudrait passer. Mais pendant que le processus A lisait le fichier sur disque, un processus B utilisait la bande magnétique et essayait de faire passer au rouge le feu relié au fichier sur disque pour y accéder. Le processus A attend que B débloque la bande, tandis que B attend que A libère le fichier sur disque. C'est cela l'étreinte fatale.

ACCES A UNE RESSOURCE (FICHIER-DISQUE) AVEC RECOURS AUX SEMAPHORES



Pour l'éviter, des régions dites critiques (à l'intérieur desquelles sont gérés les processus simultanés) ont été définies dans les systèmes d'exploitation. La gestion des processus dans la région critique est effectuée sur la base de trois postulats, formulés en 1965 par Dijkstra :

1/ Quand un processus désire entrer dans une région critique, sa demande n'est maintenue que pendant un temps fini.
2/ Un seul processus à la fois peut occuper une région critique.
3/ Un processus se trouvant dans une région critique ne peut y rester que pendant un temps fini.

Second volet de la gestion des processus simultanés : **l'échange des données.** Qu'il s'agisse de données d'entrée, de sortie ou de données traitées, l'échange se fera toujours en utilisant les zones communes aux divers processus, dans lesquelles les données sont introduites et prélevées. Ces zones sont soit des fichiers sur disque, soit des espaces de la mémoire centrale mis à la disposition du système d'exploitation à l'intention des processus.
Le synchronisme de l'accès aux données partagées est résolu à l'aide de sémaphores ; il nous reste donc à considérer l'aspect de la communication.
Etant donné une zone commune, il existe des processus « producteurs », qui introduisent alors des données, et des processus « consommateurs », qui les prélèvent. La zone commune est toujours de dimensions finies, d'où certains conflits de vitesse entre celui qui produit et celui qui consomme. Si le producteur est le plus rapide, la zone se sature, empêchant d'autres données d'entrée ; si c'est le consommateur qui est le plus rapide, il se forme des queues dans lesquelles N processus consommateurs attendent de recevoir des données. L'usage de l'unité centrale n'est alors pas optimal.
Ce dernier point est sans aucun doute moins important que le premier ; de ce fait, il est parfois négligé.
Les règles fixées sont les suivantes :

1/ Si le producteur cherche à introduire des données dans une zone déjà pleine, le processus est suspendu jusqu'à ce que la zone commune soit libérée de quelques données par un consommateur.
2/ Si le consommateur cherche à prélever des données dans une zone commune vide, le processus est suspendu jusqu'à ce qu'un producteur y introduise des données.

Les données se trouvant dans la zone commune sont gérées selon le système FIFO (Premier entré, premier sorti) ou LIFO (Dernier entré, premier sorti). Dans le premier cas, les données entrées en premier sont également prélevées en premier, alors que dans le second, ce sont les dernières données entrées qui sont prélevées en premier. Le système utilisé dépend de l'application.
Quand les processus désirant communiquer sont plus de deux, la zone commune doit pouvoir distinguer, par exemple, les données introduites par le processus A mais destinées aux processus B et C, à l'exclusion d'autres processus.
Le système d'exploitation utilise alors des mécanismes internes lui permettant d'identifier les émetteurs et les destinataires des données et de garantir que les processus ne recevront que les données les concernant. Pour cela, la zone commune est divisée en plusieurs zones tampons ayant chacune un code qui lui permet de n'être lue que par les processus le reconnaissant.
Ces zones tampons sont assimilables à des boîtes aux lettres fermées à clé ; seuls les détenteurs de clés peuvent prendre connaissance de leur courrier. Ce n'est d'ailleurs pas un hasard si la communication de données entre processus simultanés est souvent appelée « communication à boîtes à lettres ».
Elle est également associée à des utilisateurs simultanés : l'échange de messages donne lieu à un service de « courrier électronique » dont peuvent se servir tous ceux qui ont un terminal connecté à un ordinateur ou à un réseau d'ordinateurs. Le système d'exploitation Unix permet, à un utilisateur, d'envoyer des messages à tous les utilisateurs du système.
Les messages reçus sont archivés dans des fichiers privés et regroupés au sein d'une base logique (contenu).
Enfin, il est possible d'envoyer des messages sur base temporelle : départ à telles date et heure. Si le destinataire est identique à l'expéditeur, on a alors créé un agenda électronique.

L'informatique graphique en Basic

L'informatique graphique est l'application la plus spectaculaire de l'informatique. Née seulement sur de gros systèmes, elle s'est depuis largement développée notamment sur micro-ordinateurs. Cette diffusion a été rendue possible grâce aux progrès réalisés au niveau du matériel. Les premiers micros étaient, pour des raisons économiques, des machines limitées à des applications simples. La majorité d'entre eux n'avait pas de réelles capacités graphiques, ni la haute résolution.

L'évolution du matériel et la baisse des coûts ont ensuite permis de fabriquer des systèmes rentrant dans la catégorie « ordinateur personnel » ayant des capacités graphiques étendues. Les producteurs de logiciels spécialisés se sont alors tournés vers cette catégorie de matériels et ont écrit des programmes graphiques de plus en plus sophistiqués.

Un ordinateur destiné à des emplois graphiques est très différent des ordinateurs classiques.

La première particularité concerne la diversité des périphériques qu'il doit gérer.

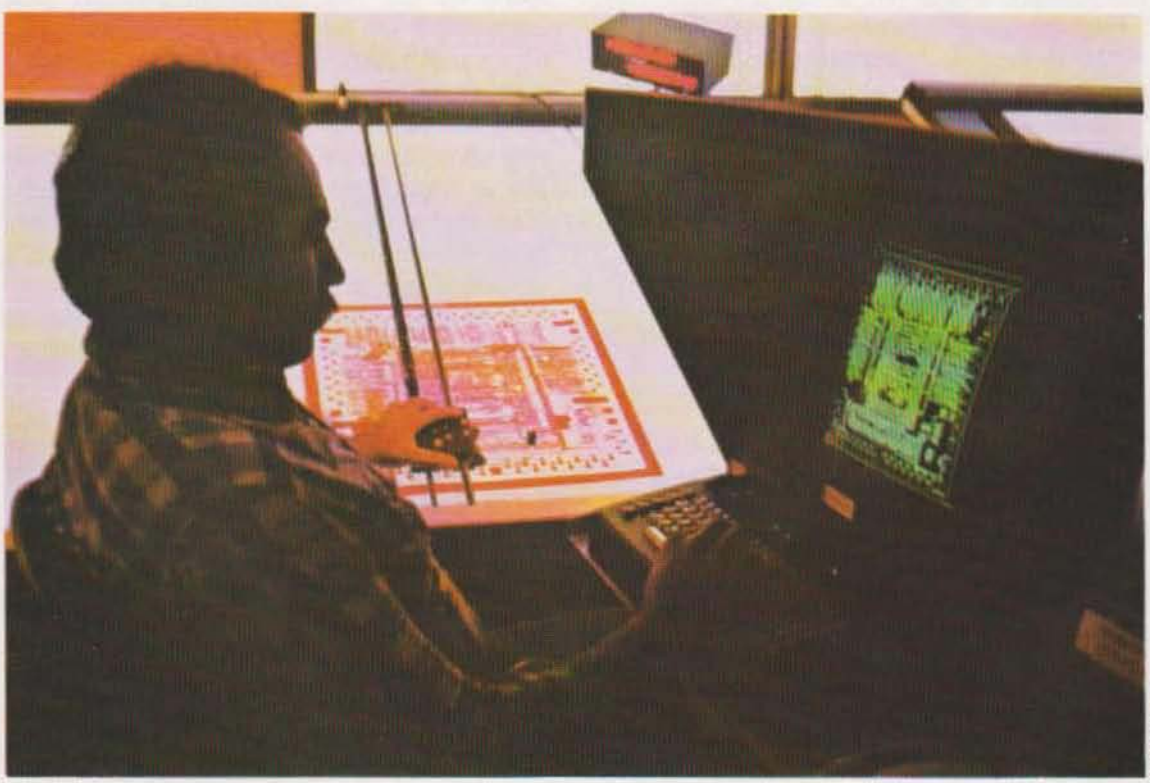
Avec une machine classique de la catégorie micro, les périphériques appartiennent à 4 classes : écran, clavier, disque (ou bande magnétique) et imprimante. Pour pouvoir être utilisée à des fins graphiques, la même machine doit être complétée de nombreux autres périphériques du type traceur de courbes, tablette graphique, et surtout des écrans graphiques.

S'ils utilisent des ports d'E/S et des protocoles de communication semblables aux autres, ces nouveaux équipements demandent néanmoins des logiciels d'application spécialisés.

La deuxième différence principale se situe au niveau de la structure matérielle interne.

L'informatique graphique (ou infographie) de-

Système informatique pour la Conception assistée par ordinateur (CAO).



Marka/Jarner

mande de grandes capacités de mémoire, ce qui oriente l'utilisateur vers des processeurs 16 ou 32 bits ou, à défaut, vers l'emploi de cartes d'extension particulières. Sauf quelques rares exceptions, la première solution oblige à adopter des machines de la catégorie « mini », alors que la seconde est la plus courante avec les « micro » (ces machines ont, pour la plupart, des microprocesseurs 8 bits, ce qui les rend inadaptées à la gestion de mémoires supérieures à 64 K). Toutefois, l'emploi de dispositifs d'extension tend à disparaître avec l'apparition des micros 16 bits.

Applications de l'informatique graphique

En pratique, il n'existe pas d'application dans laquelle le graphique ne puisse pas constituer une aide rapidement indispensable, ou tout au moins un complément utile.
Pour être interprété et assimilé, le résultat d'un traitement présenté sous forme numérique demande un grand effort de concentration et d'analyse. Ce même résultat, représenté graphiquement, sera d'une compréhension plus immédiate.
Ces considérations ont abouti à l'emploi de l'informatique graphique dans des domaines moins techniques, comme les études démographiques, l'analyse statistique et les applications économiques en général.
L'information graphique connaît actuellement 4 types d'application :

- le graphique de décision
- la conception assistée par ordinateur (CAO)
- le traitement des images
- l'animation

Le graphique de décision

Cette expression désigne les applications orientées vers la résolution de problèmes à caractère économique ou, plus généralement, relatifs à la prise d'une décision au niveau de l'entreprise (aide à la décision).
L'automatisation du travail de bureau rend souvent nécessaire la représentation synthétique de certains résultats importants. L'emploi de graphiques, pour mettre en évidence l'évolution de certains phénomènes économiques, a pris une telle importance qu'il a engendré un véritable secteur de l'informatique graphique : le graphique de décision. Du point de vue de la programmation, ce sont là des applications simples, qui ne demandent généralement pas de traitements particuliers ni d'algorithmes complexes.

La conception assistée par ordinateur (CAO)

La CAO a été l'un des premiers domaines d'application de l'informatique graphique. Les potentialités de l'ordinateur peuvent, en effet, y être pleinement exploitées.
De grandes quantités de données relatives au travail de conception sont mémorisées et rappelées par un ou plusieurs menus. Il est ainsi possible de créer des banques de données contenant des modules ou les détails déjà prêts d'un dessin. Pour obtenir un dessin fini, le concepteur n'a qu'à relier les différents modules, tandis que l'ordinateur affine le dessin à l'aide des éléments de détail.
Dans les cas d'applications plus complexes, l'ordinateur permet d'obtenir différentes vues de l'objet, afin d'en tirer des plans d'exécution ; de plus, il effectue également les calculs nécessaires à la détermination des dimensions ou des caractéristiques les plus adaptées à une application particulière. Nous allons maintenant examiner quatre domaines d'emploi.
La conception mécanique. C'est l'aspect le plus connu de la CAO. Elle est rendue possible par un logiciel produisant des plans mécaniques par composition d'éléments géométriques élémentaires ou utilisant des symboles prédéfinis (éléments de figure).
Les éléments géométriques entrant dans la réalisation d'une figure sont précisémment :

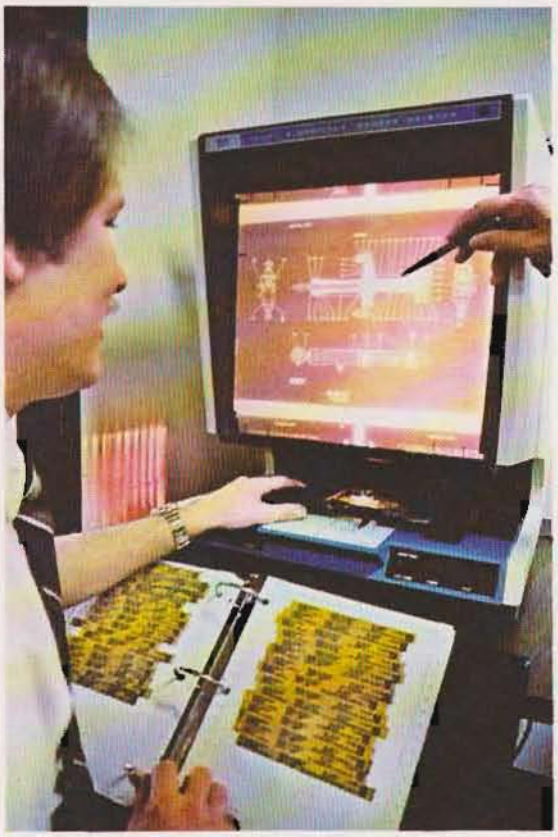
— les points
— les segments de droite
— les cercles
— les lignes courbes
— les tangentes
— les arcs et les raccords

En employant un jeu de commandes de sélection et de positionnement des différents éléments, on aboutit alors à des dessins très complexes.

Les systèmes de CAO prévoient généralement

la possibilité de produire automatiquement des parties symétriques, ou d'agrandir un détail. Les figures composées sont mémorisées sur disque et reproduites sur papier. Le gain de temps qui en découle par rapport à la méthode de dessin manuel classique est considérable.
Autre avantage : la facilité avec laquelle le concepteur apporte des modifications à un dessin ; un élément particulier peut être rappelé et modifié ou redessiné en quelques minutes, alors que des journées entières de travail manuel étaient nécessaires auparavant.
Deuxième avantage : la production automatique d'instructions d'asservissement des machines-outils à commande numérique.
Dans la production en série, on cherche à réduire l'intervention humaine, notamment au cours des phases de fabrication des pièces mécaniques. Il existe des machines entièrement automatiques (tours, fraiseuses, fileteuses...), dites « à commande numérique », et pilotées par ordinateur.
Les instructions nécessaires à la commande des machines sont automatiquement générées lors de la conception de l'élément mécanique par le système de CAO : c'est alors de la fabrication assistée par ordinateur (FAO). Le processus global de conception d'une pièce et de production des codes de commande s'appelle CFAO (Conception et fabrication assistées par ordinateur).

La conception électronique. Cette application de l'informatique graphique est très proche de la précédente. Introduite plus récemment, elle doit connaître un très grand développement dans un proche avenir. Ses champs d'application vont de la conception des circuits intégrés à celle de cartes électroniques tout entières, avec production automatique des schémas de fabrication et liste des composants nécessaires.
Là aussi, les avantages dépassent le simple gain de temps dans l'exécution de la partie graphique. Au fur et à mesure que le projet progresse, l'ordinateur vérifie les stocks de composants, contrôle les choix du concepteur du point de vue de la compatibilité et de l'interconnexion des composants employés et, même, effectue une sélection entre différents composants concurrents, en fonction des caractéristiques physiques requises.


Marka/L. de Wys

Exemple d'application graphique.

La conception architecturale. L'informatique graphique s'est avérée extrêmement utile en architecture, depuis l'étude de l'habitabilité des locaux, de leur aménagement, jusqu'à l'analyse structurelle et, dans les programmes les plus complexes, au dessin de vues en perspective.
Cette dernière application, qui exige des logiciels et des matériels de pointe, connaît actuellement un vif succès tant dans les petites que dans les grandes entreprises.
La plupart des difficultés concernant les algorithmes d'effacement des lignes non visibles ont été résolues. La construction d'une vue en perspective ou d'une axonométrie demande, en effet, la mise au point d'une logique complexe pour déterminer si une ligne donnée est visible ou cachée par d'autres éléments de la figure. Les applications informatiques utilisent presque exclusivement la technique axonométrique, plus simple à gérer. Ce n'est que dans des cas particuliers que le système prévoit l'effacement automatique des lignes cachées.

Dans ce secteur d'application, une autre technique est très importante : l'ombrage. Dans un dessin fait à la main, c'est l'artiste qui, d'après sa sensibilité, décide où et comment il doit ajouter des ombres pour que le résultat donne l'impression d'être tridimensionnel.
Avec l'ordinateur, le processus est beaucoup plus complexe ; en effet, il faut décrire à la machine tous les algorithmes appliqués ainsi que les décisions qui en découlent. Ces algorithmes demandent des machines très puissantes, même plutôt simples.

Le design industriel. Les designers n'auront pas été les derniers à recourir à la technique graphique pour concevoir leurs produits.
Ce domaine d'application, très vaste, couvre tous les secteurs dans lesquels il faut associer, à la conception technique d'un produit, une étude esthétique approfondie du résultat à obtenir. L'ordinateur simplifie considérablement le travail de mise en forme d'une idée artistique, qui peut aussi bien concerner la création d'un objet que la reproduction d'un motif sur tissu. Les stylistes et les créateurs de mode sont donc d'autres utilisateurs attitrés.

Le traitement des images

Dans l'informatique graphique, le traitement des images occupe une place particulière, parce qu'il demande l'emploi de techniques logicielles et d'équipements spécialisés, ayant trait à l'acquisition et au traitement des images, ainsi qu'à leur reconnaissance.
Une image se présente sous différentes formes (photographie, dessin) et a toujours un caractère « continu ». Pour être entrée dans l'ordinateur, elle doit être « numérisée », c'est-à-dire traduite point par point en code binaire. Cette conversion est effectuée à l'aide de dispositifs de précision allant de la simple tablette graphique à des systèmes de lecture ou à des caméras complexes.
Le logiciel nécessaire au fonctionnement de ces appareils pose également de gros problèmes, dûs en grande partie à la logique binaire de l'ordinateur. Celle-ci le met en situation de n'identifier que deux états (1 et 0) tandis que l'acquisition d'une image requiert la capacité de reconnaître la tonalité d'une couleur ou les tons de gris dans le cas d'une image monochromatique. Dans la réalité, les nuances de couleurs sont infinies ; or l'ordinateur, aussi puissant soit-il, ne peut prendre en compte ou n'en simuler qu'un nombre limité. C'est donc aux programmes d'application d'optimiser les ressources afin d'obtenir une représentation aussi proche que possible de l'original.
La technique utilisée consiste, grosso modo, à reproduire les nuances en épaississant plus ou moins les points constituant la figure. Si les points sont très proches, on a un ombrage très

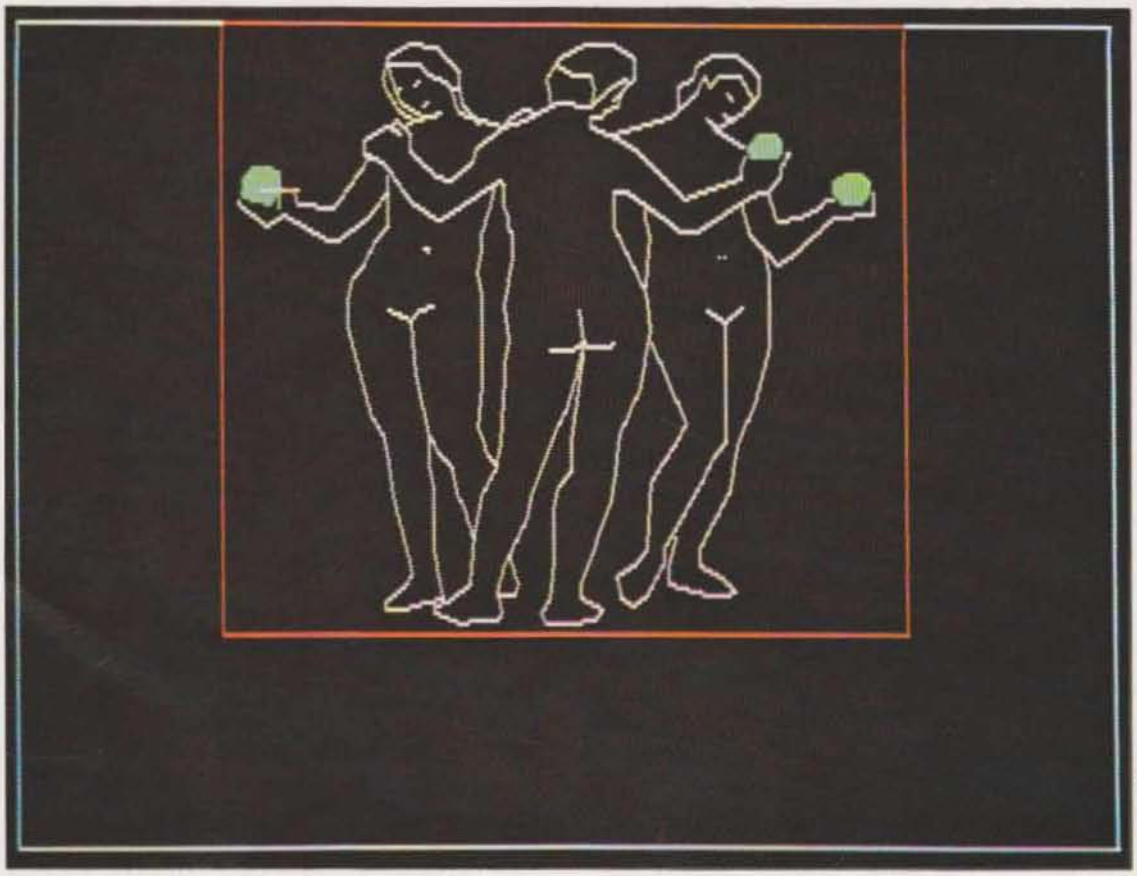
Exemples d'images graphiques obtenues par ordinateur : le visage d'Einstein (à gauche), le pont de Brooklyn (à droite).



Marka/L. de Wys



Marka/R. Vila



Centre THC/Marka

Application à l'art du traitement des images sur écran polychrome.

marqué, alors que l'on obtient des effets d'estompage en les écartant. Les systèmes les plus évolués emploient des moniteurs couleur qui, grâce à des combinaisons spéciales, fournissent des représentations très précises.

Conséquence directe du développement des techniques d'acquisition et de traitement des images, les méthodes de reconnaissance des images avancent à grands pas, même si elles ne sont pas facilement accessibles pour le moment. Ce champ d'application, appelé « reconnaissance des formes » (pattern recognition) constitue aujourd'hui une discipline à part entière. Ses implications dans d'autres secteurs sont très importantes, comme la réalisation de têtes de « lecture » ou de classification d'objets par l'ordinateur à partir de leur image.

L'animation

Il est possible d'exploiter la très grande rapidité de réponse des ordinateurs pour préparer des programmes capables d'animer des dessins avec une résolution telle que l'on obtient une impression de mouvement continu.

Les champs d'application de l'animation sont divers, allant de la préparation des jeux vidéo à l'analyse cinématique d'un phénomène. Certains programmes simulent les déplacements du corps d'un conducteur lors d'un accident automobile et permettent de décider du meilleur agencement interne d'une voiture.

Les terminaux graphiques

Les dispositifs d'E/S orientés graphiques constituent une vaste gamme de produits. Nous allons examiner, de façon plus approfondie, les périphériques connectables aux micro-ordinateurs et aux ordinateurs personnels ; nous nous contenterons d'évoquer rapidement ceux utilisés dans les grands systèmes informatiques.

Dans la catégorie micros et ordinateurs personnels, les programmes d'application graphiques se répondent peu à peu avec des prix qui les mettent à la portée du plus grand nombre.

Dispositifs d'entrée

Les données d'entrée à fournir à un logiciel graphique se composent essentiellement d'une série de coordonnées représentant le dessin ou la figure à mettre en mémoire. Les dispositifs d'entrée peuvent donc être vus comme des transducteurs de position.

La figure ci-contre reproduit le schéma de principe d'un dispositif général d'acquisition d'entrées graphiques. Le graphique à saisir est dessiné sur le plan de travail, par exemple sur une feuille de papier millimétrée. Chacun des points du graphique est désigné par rapport à un système de coordonnées (X, Y) correspondant aux bords de la feuille.

Entrer les graphiques en machine, c'est simplement lui transmettre les coordonnées de chacun des points du dessin.

L'opérateur déplace, sur le dessin, un transducteur de position qui produit deux signaux : le premier proportionnel au déplacement le long de l'axe X et l'autre proportionnel au déplacement le long de l'axe Y. Au fur et à mesure que le transducteur est déplacé sur le contour de la figure, l'ordinateur enregistre les coordonnées relatives à chaque point, qui ne peut prendre que deux valeurs, ON=visible, ou OFF=non visible.

Si le transducteur est positionné, par exemple, au niveau du point P, l'ordinateur collecte ses coordonnées (X, Y) ; si, de plus, l'état ON du point P est activé, la machine mémorise ce point et, éventuellement, l'affiche à l'écran.

Le principe sur lequel sont fondés les transducteurs de position est généralement de type analogique ; le terminal doit donc être assorti d'un convertisseur analogique/numérique. Il faut également prévoir un logiciel capable de ramener le signal à des valeurs compatibles avec les dimensions de l'écran.

La figure de la page 1390 représente le schéma de principe d'un transducteur potentiométrique qui mesure indirectement une distance par l'intermédiaire d'une mesure de résistance. Sa structure physique ressemble à celle d'un appareil à dessiner universel.

Deux curseurs métalliques se déplacent le long de deux résistances perpendiculaires l'une par rapport à l'autre. Chacun des points du plan est repéré d'une seule façon : en positionnant correctement les deux curseurs le long des deux résistances (guides). Une fois le point P atteint, les valeurs de résistance* Ry et Rx, qui se mesurent entre les points P,B et B,A, définissent de façon univoque les coordonnées du point.

Les valeurs de résistance sont converties en valeurs numériques et envoyées à l'ordinateur. Ce processus est résumé par les 3 phases suivantes :

1/ lecture de la position
2/ transformation de la donnée en valeur numérique
3/ conversion à l'aide de facteurs d'échelle

Chacune d'elles introduit des inexactitudes ou des erreurs dont l'addition détermine la précision et la répétabilité du système. Les causes d'imprécision dépendent essentiellement de la construction mécanique des transducteurs et aussi de la qualité du CAN (convertisseur analogique/numérique).

Pour la précision mécanique, il n'est pas possible de faire des estimations a priori ; la seule pratique reste l'essai de l'appareil. A l'inverse, la précision du CAN peut généralement être évaluée à l'avance.

L'élément déterminant (et non l'unique) est le nombre de bits sur lequel travaille le convertisseur. Un convertisseur à 4 bits, par exemple, fournit en sortie un nombre binaire à 4 bits, c'est-à-dire une valeur comprise entre 0 et 15 (en décimal).

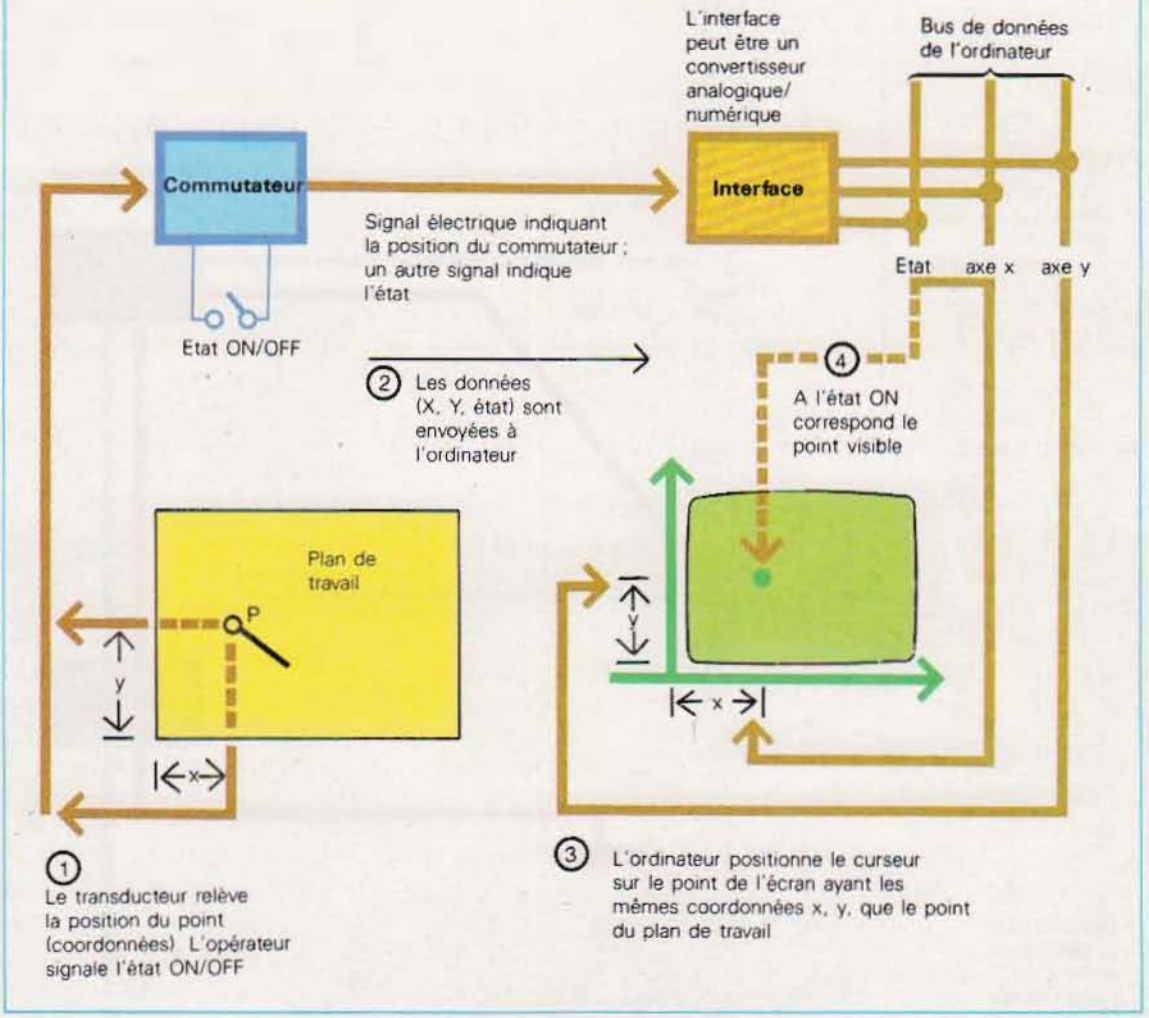
La plus petite quantité fournie est 1, ce qui signifie que sa précision, exprimée sous forme d'un ordre de grandeur, est de 1/15, soit à peu près 6 %. Il convient d'ailleurs de noter que la précision s'améliore très nettement au fur et à

*La valeur d'une résistance se mesure en ohms (symbole Ω). Cette valeur dépend de la longueur du fil constituant la résistance (toutes autres caractéristiques étant égales). Cette dépendance s'exprime à l'aide d'une fonction de proportionnalité. Si une résistance de 100 Ω est longue de 2 cm, la valeur mesurée en son milieu est de 50 Ω, alors qu'à un quart de sa longueur, elle est de 25, etc. D'une façon générale :

Déplacement = K R

K étant un facteur de tarage et R la valeur de résistance mesurée (dans l'exemple K = 2 cm/100 Ω = 1/50 [cm Ω]).

SCHEMA DE PRINCIPE D'UN PERIPHERIQUE D'ENTREE POUR APPLICATIONS GRAPHIQUES

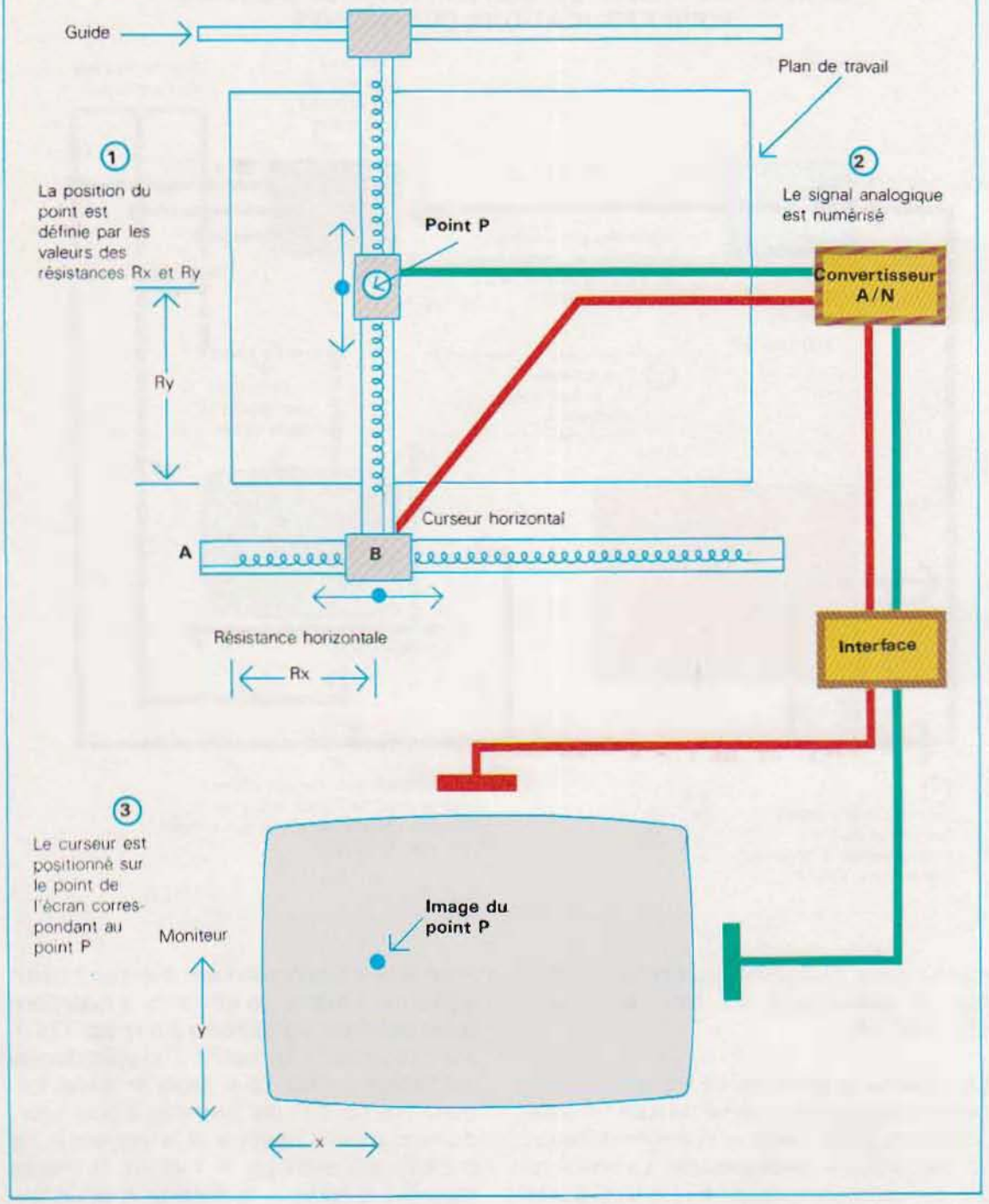


mesure que le nombre de bits augmente. Ainsi, avec un convertisseur à 5 bits, elle passe à 1/31, soit 3 %.

La tablette graphique. Le transducteur potentiométrique est très utilisé dans les tablettes graphiques, mais avec une légère modification par rapport au schéma présenté. La construction des potentiomètres de forme rectiligne et de longueur égale à celle d'une feuille à dessin est très coûteuse et demande l'emploi d'un système de guides.
Le potentiomètre rotatif (voir figure 2 en page 1391) est beaucoup plus économique mais il mesure un angle et non une distance.

Pour obtenir les coordonnées d'un point, il faut utiliser un système de référence à **coordonnées polaires** (voir schéma 3 en page 1391). Avec ce système, la position d'un point du plan est définie à l'aide d'un angle et d'une longueur. Le point P, par exemple, a pour coordonnées polaires l'angle α et le segment A ; le point P1 est défini par α 1 et A1. Si l'on se reporte à la figure 2, la distance A est la longueur du bras, et l'angle sa rotation (dont dépend la valeur de résistance). Lorsque le bras tourne, la valeur de résistance (R) est proportionnelle à l'angle, et il est donc possible de repérer tous les points se trouvant sur la circonférence de même centre que le potentio-

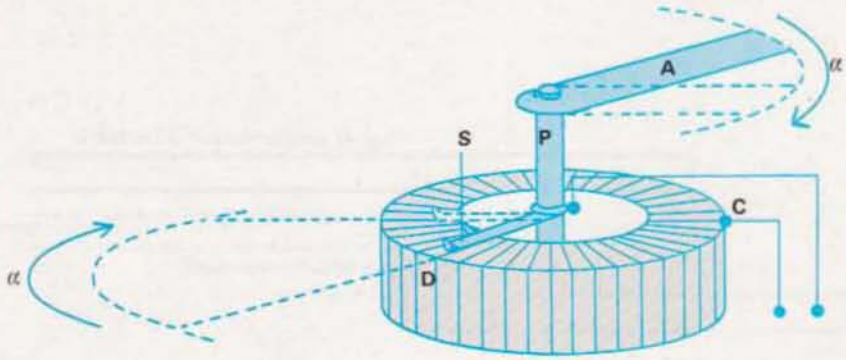
1 / SCHEMA D'UN TRANSDUCTEUR POTENTIOMETRIQUE DE POSITION



mètre et de rayon A. Avec ce système, on n'obtient pas la position de tous les points du plan, mais seulement ceux qui sont circonscrits par le bras dans sa rotation. Pour obtenir tous les points, il faut monter un deuxième potentiomètre à l'extrémité du bras A, et un deuxième bras B. Toutes les positions du point de travail sont alors atteintes.

Le schéma détaillé de la tablette graphique est représenté p. 1392.

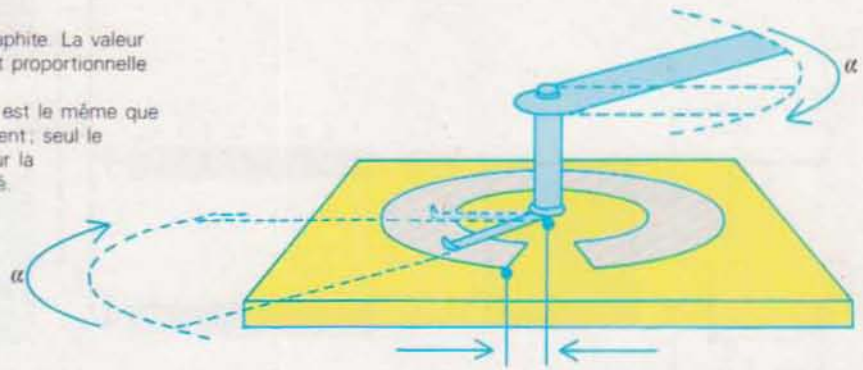
2 / POTENTIOMETRES UTILISES DANS LES TABLETTES GRAPHIQUES



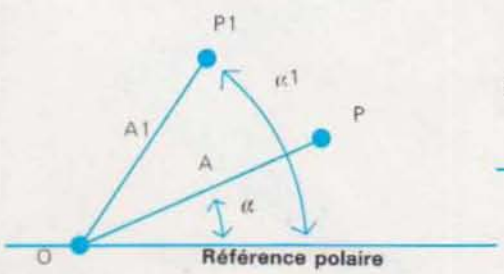
Résistance fonction de l'angle de rotation

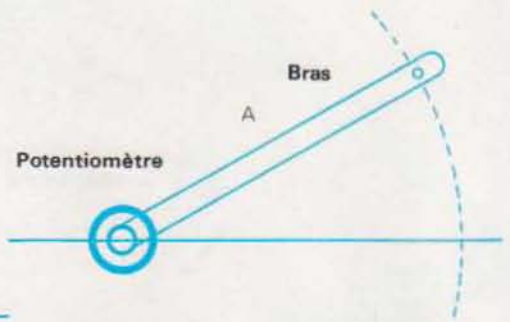
Schéma de fonotionnement d'un potentiomètre. La rotation de la tige A provoque celle de l'axe P qui, à son tour, entraîne le déplacement du curseur à frottement S. Quand la position de S varie, la quantité de fil électrique comprise entre les points C et D (respectivement l'extrémité de l'enroulement et le contact S), varie également. Pour des puissances limitées, comme dans ce type d'application, l'enroulement de fil est remplacé par une fine couche d'un matériau spécial (graphite, par exemple).

Potentiomètre à graphite. La valeur de la résistance est proportionnelle à celle de l'angle
Le fonctionnement est le même que dans le cas précédent ; seul le matériau utilisé pour la résistance a changé.



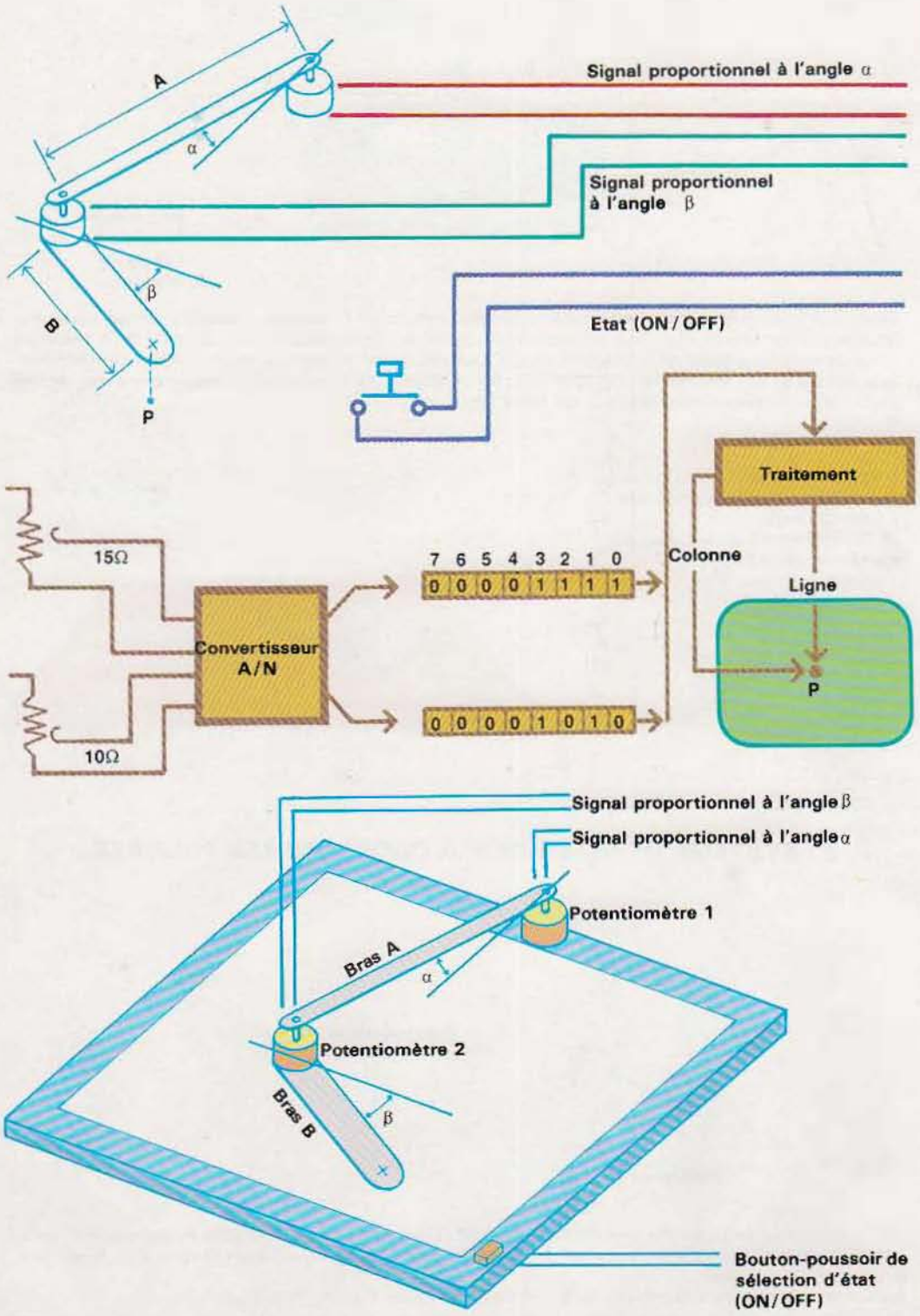
3 / SYSTEME DE REFERENCE A COORDONNEES POLAIRES





Coordonnées polaires. La position d'un point est repérable à partir des valeurs d'un angle et d'une distance. Celle-ci mesure le segment entre le point considéré et l'origine O ; l'angle pris en compte est compris entre le segment et une droite de référence.
Dans notre exemple, les coordonnées (A, α) définissent le point P et (A_1, α_1) le point P1.

4 / SCHEMA D'UNE TABLETTE GRAPHIQUE



Les coordonnées fournies par ce système sont polaires : il faut donc les transformer en coordonnées cartésiennes (X, Y). La méthode la plus simple est illustrée ci-dessous.
La formule de résolution est :

$X = A*COS(\alpha) + B*COS(\beta)$
$Y = A*SIN(\alpha) + B*SIN(\beta)$

avec comme données :

X,Y = coordonnées cartésiennes du point

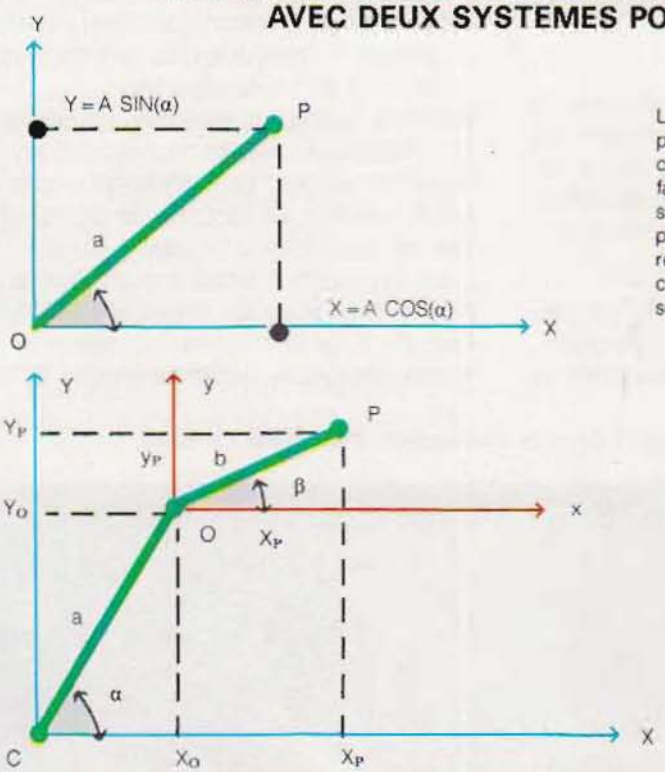
A,B = longueurs des deux bras
α,β = angles formés par les deux bras par rapport à une droite de référence

Normalement les longueurs des deux bras sont égales (A=B), et donc :

$X = A*(COS(\alpha) + COS(\beta)$
$Y = A*(SIN(\alpha) + SIN(\beta)$

Les valeurs X et B sont fournies par deux convertisseurs A/N (CAN) (un pour chaque potentiomètre) sous forme de valeurs numé-

DETERMINATION DES COORDONNEES D'UN POINT AVEC DEUX SYSTEMES POLAIRES



La transformation des coordonnées polaires (α ,A) en coordonnées cartésiennes (X, Y), est immédiate en faisant coincider les origines des deux systèmes de référence (point O) et en prenant l'axe X comme droite de référence des angles. Les coordonnées cartésiennes sont les projections du segment A sur les axes.

Les coordonnées du point P par rapport aux axes x, y sont :
$x_P = b\ COS(\beta)$
$y_P = b\ SIN(\beta)$

Celles du point O par rapport aux axes X, Y sont :
$X_O = a\ COS(\alpha)$
$Y_O = a\ SIN(\alpha)$

Les coordonnées de P par rapport au système X, Y s'obtiennent en additionnant les coordonnées de P par rapport à x, y à celles de O par rapport à X, Y :
$X_P = X_O + x_P = a\ COS(\alpha) + b\ COS(\beta)$
$Y_P = Y_O + y_P = a\ SIN(\alpha) + b\ SIN(\beta)$

a et b étant constants (longueurs des bras de la tablette), la position du point P est déterminée par les valeurs des angles de rotation nécessaires au positionnement de l'élément de contact en P.

riques : le logiciel se limitera au calcul des deux expressions mentionnées. Les transducteurs potentiométriques ne sont pas les seuls dispositifs employés. Dans les systèmes de grande taille, ils ne sont pas du tout utilisés. Leur inconvénient provient de l'encombrement des bras sur le plan de travail et aux tolérances mécaniques nécessaires. Les dispositifs dotés de photostyles fournissent de meilleurs résultats. Le transducteur comprend, comme élément sensible, un photostyle dont la pointe, composée d'un interrupteur, se ferme lorsqu'il y a pression sur le papier et qui signale ainsi l'état ON/OFF du point. Un tel transducteur nécessite des systèmes de relevé de cordonnées bien plus complexes que le précédent.
Parmi les plus usités, on notera :

- Les champs électriques ou magnétiques. La tablette génère un champ électrique ou magnétique ; le photostyle comporte un capteur qui fournit une tension proportionnelle à l'intensité du champ qui l'atteint, elle-même fonction de la position.
- Les ultrasons. Le photostyle émet, en permanence, des ultrasons. Des détecteurs, placés au bord de la feuille, mesurent le temps écoulé entre l'émission et la réception du son, proportionnel à la distance. On peut ainsi repérer la position du style par rapport aux détecteurs.

Ces méthodes nécessitent des matériels très complexes, qui ne sont pas à la portée des micros ou des ordinateurs personnels. Il existe quelques moyens — mécaniques le plus souvent — pour prendre les mesures d'objets tridimensionnels. Dans ces applications, outre le coût élevé du matériel, les logiciels manquent de puissance pour mémoriser et gérer des figures tridimensionnelles.

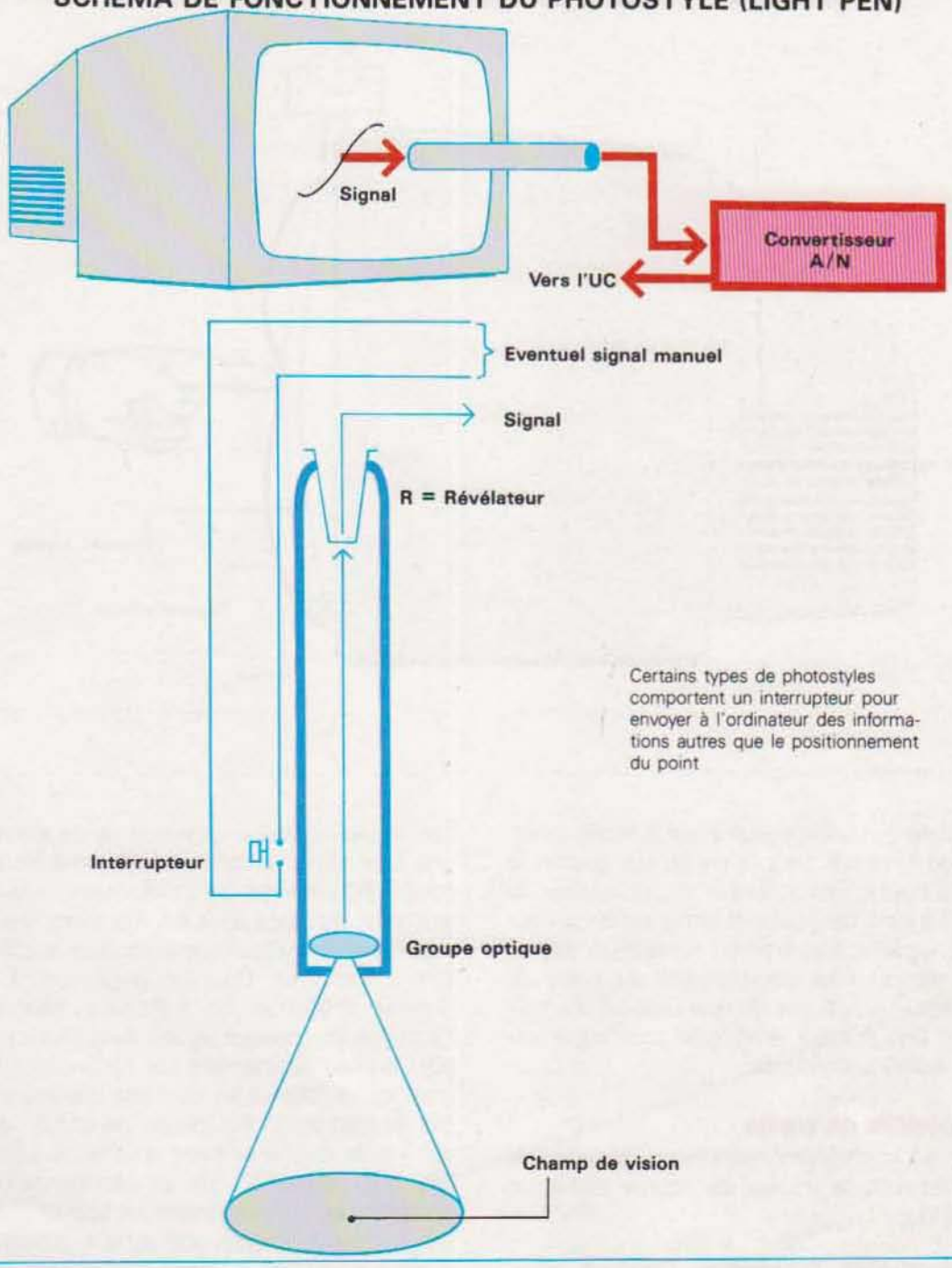
Dispositifs spéciaux. Il existe d'autres moyens pour communiquer des informations graphiques à l'ordinateur. Ils sont toutefois plus complexes et moins répandus.
Les deux principaux systèmes qui ont été portés, depuis peu, sur les micro-ordinateurs sont le photostyle (light pen) et l'écran tactile, sensible au toucher. Le fonctionnement du photostyle est illustré dans la page ci-contre.
C'est un capteur sensible à la lumière émise par le moniteur. En déplaçant le stylo sur l'écran, la lumière reçue est convertie en signaux électriques, envoyés ensuite à l'ordina-

Utilisation de programmes graphiques dans la conception d'un motif de tissu.



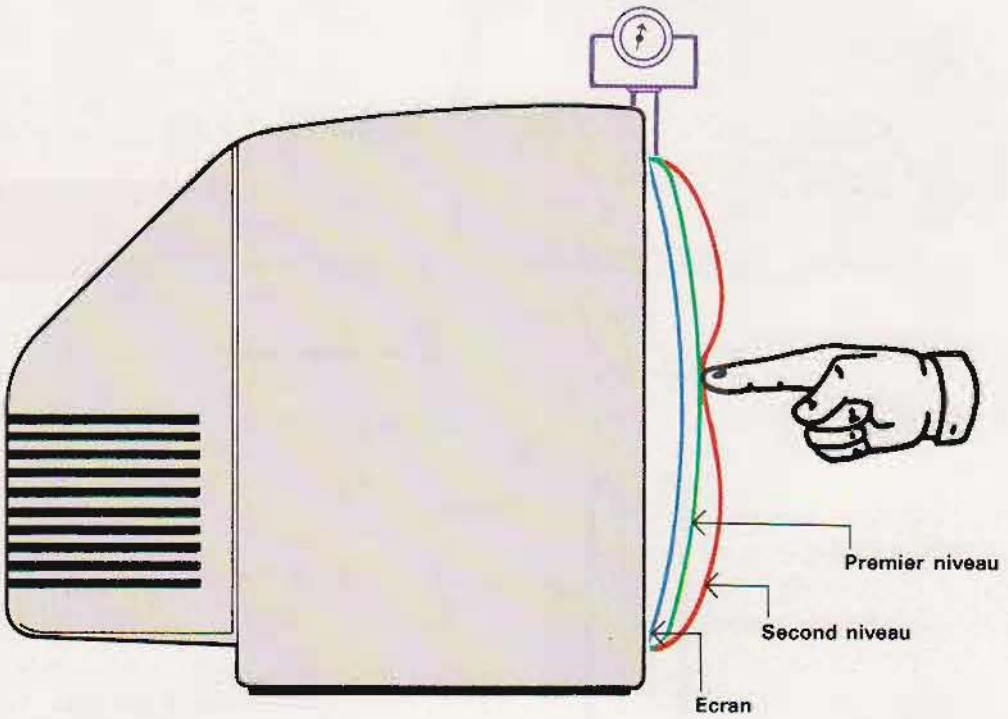
Marka

SCHEMA DE FONCTIONNEMENT DU PHOTOSTYLE (LIGHT PEN)



teur (après conversion analogique/numérique). Ces signaux, traités par une unité de contrôle reliée à l'écran, permettent de repérer les coordonnées du photostyle. On peut ainsi « lire » un dessin sur l'écran ou le créer comme s'il s'agissait d'un quelconque crayon à papier. Le fonctionnement d'un écran tactile est explicité par le schéma ci-dessus. La surface externe de l'écran est composée de deux feuilles de verre séparées par un espace très étroit. En

PRINCIPE DE FONCTIONNEMENT DE L'ECRAN TACTILE



exerçant une faible pression sur la feuille externe, on l'incurve jusqu'à ce qu'elle touche la feuille interne ; on obtient ainsi une variation de la différence de potentiel entre les deux couches laquelle, convertie en numérique, fournit une mesure des coordonnées du point de contact. Il s'agit, par comparaison, d'une méthode peu précise, employée pour accomplir des tâches secondaires.

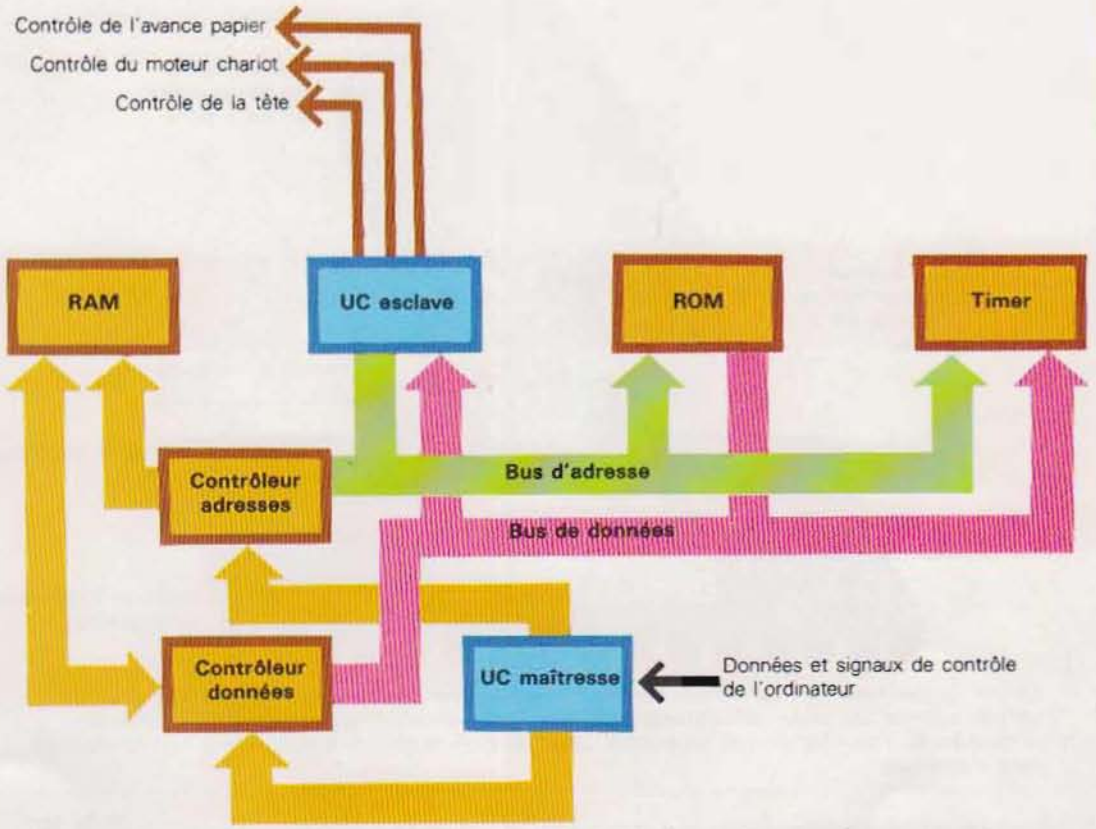
Dispositifs de sortie

Pour les applications graphiques, on retrouve l'imprimante, le traceur de courbe et l'écran graphique.

L'imprimante graphique. Pour les applications graphiques n'exigeant pas une précision très élevée, on peut utiliser les imprimantes dites semi-graphiques ou graphiques. Il s'agit d'imprimantes à aiguilles ordinaires (schéma ci-contre) permettant cependant d'adresser chaque élément de la matrice. Dans son emploi normal (impression de caractères), une série d'instructions correspond à chaque code ASCII envoyé par l'ordinateur ; ces instructions, stockées en ROM, n'activent que les aiguilles de la matrice correspondant au caractère à imprimer. Pour un graphique, il est possible d'éliminer ce traitement interne à l'imprimante ; chaque aiguille de la matrice est alors pilotée directement par l'ordinateur. Les graphiques obtenus ne sont pas très précis et leur gestion est plutôt lourde, surtout à cause des limites de mouvement du chariot. La tête d'écriture ne peut en effet se déplacer qu'horizontalement. Le mouvement vertical se réalise par avance du papier, une ligne à la fois. Le graphique obtenu a donc, sur un des deux axes, une résolution qui, dans le meilleur des cas, est celle de l'interlignage de l'imprimante. On peut légèrement améliorer sa qualité en sélectionnant soigneusement les aiguilles ; du point de vue graphique cependant, la présentation reste de mauvaise qualité.

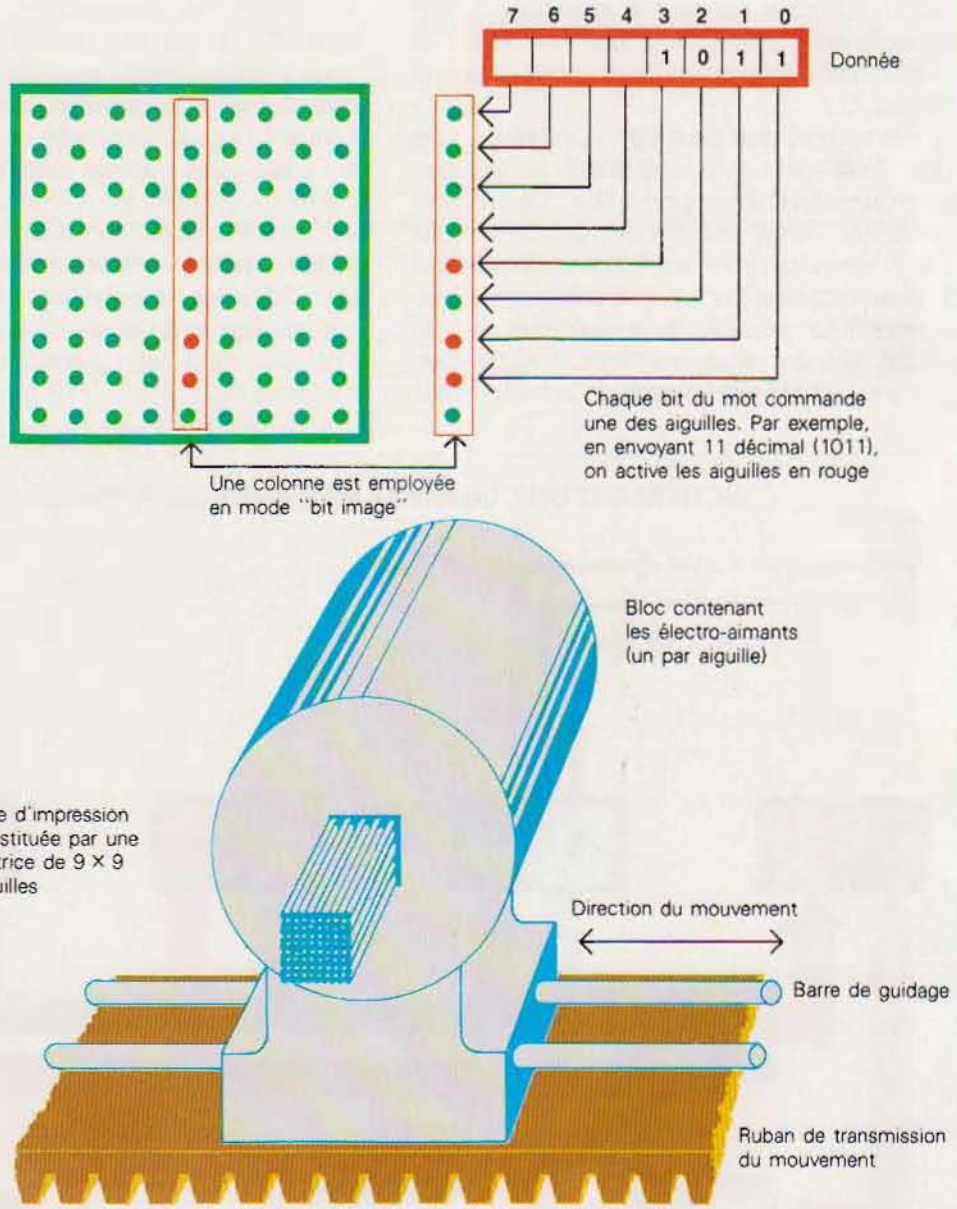
La méthode « column scan bit » (voir p. 1398) permet d'améliorer la qualité du graphique. Cette technique consiste à sélectionner une des colonnes composant la tête d'écriture et à commander de manière autonome chacune des aiguilles.
La même méthode peut être appliquée à une ligne de la matrice (bien que cela ne soit pas prévu sur toutes les imprimantes). Ce système adresse un certain nombre de points (7 × 9 ou 9 × 9, selon les dimensions de la matrice) dans la zone occupée par un caractère, avec une augmentation sensible de la résolution du graphique. Sur certaines machines, il existe des utilitaires, lancés par un code de contrôle, dont la fonctionnalité est de reproduire sur l'imprimante le contenu intégral de l'écran vidéo. En réalité, le déroulement de ces sous-programmes est plus complexe. Dans beaucoup d'équipements, on a prévu une zone de mémoire spéciale, appelée page graphique, qui contient l'image numérisée de ce qui apparaît à l'écran. Les routines décrites prélèvent le contenu de cette zone mémoire et, en fonction des codes binaires rencontrés, mettent en route les aiguilles correspondantes. Dans ce cas, les difficultés proviennent du mécanisme d'adressage et de la sélection de la mémoire à l'intérieur de la page graphique. De plus, dans certaines applications, il faut, avant le transfert

SCHEMA D'UNE IMPRIMANTE SEMI-GRAPHIQUE



L'UC maîtresse reçoit les données et les signaux de contrôle de l'ordinateur (ou mieux des circuits et des interfaces) et les transfère dans la RAM ou met en action une des ROM selon le mode d'impression choisi.
En écriture normale, la ROM qui contient le jeu de caractères sélectionnés est connectée au bus de données.
L'UC esclave enregistre ces signaux et les envoie aux organes contrôlant les dispositifs d'impression.
En mode graphique les données reçues par l'UC maîtresse sont transférées dans la mémoire RAM et ensuite aux organes mécaniques via l'UC esclave.
Les imprimantes plus sophistiquées comportent plusieurs ROM, en fonction du type de caractère que l'on veut obtenir.

METHODE D'IMPRESSION "BIT IMAGE"



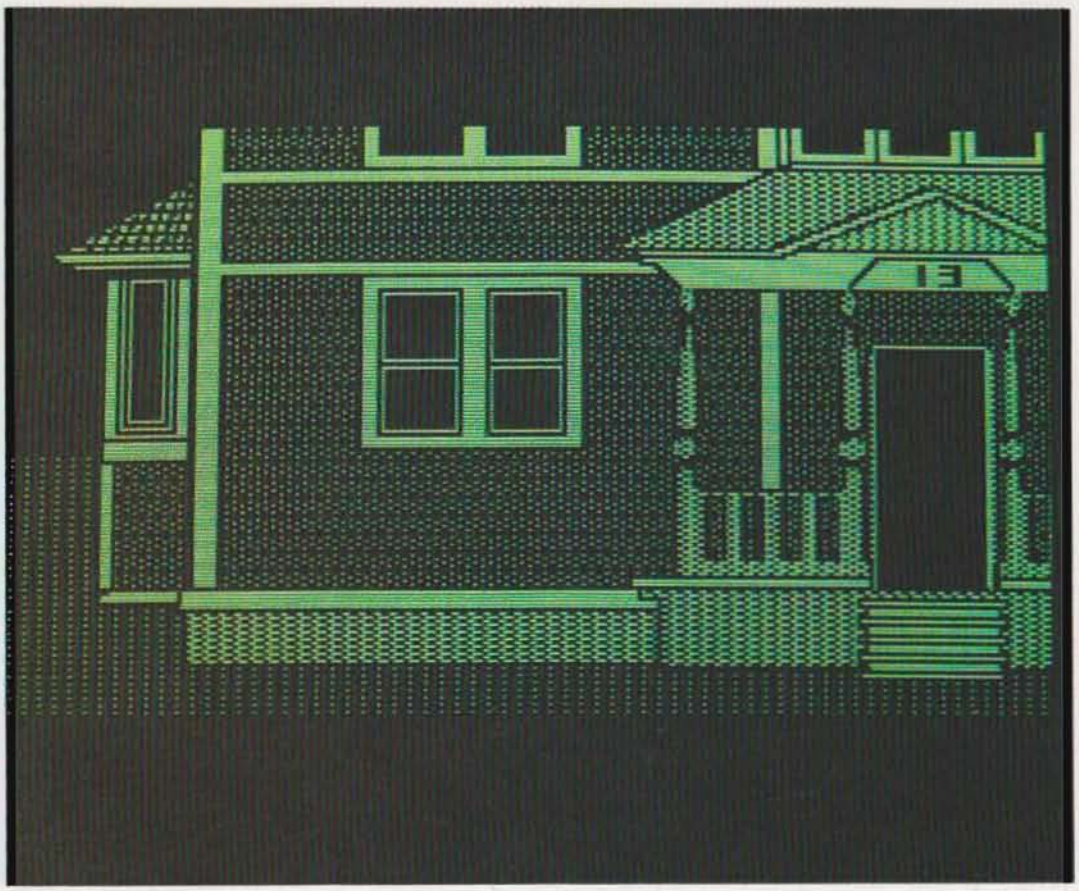
Ce type de fonctionnement est normalement mis en route en deux temps. On envoie d'abord un code de contrôle suivi par une valeur numérique qui exprime le nombre de bits à représenter (ils seront envoyés ultérieurement). Puis il faut envoyer les données, groupées en mots de 7 ou 8 bits, selon la logique présentée dans le graphique.

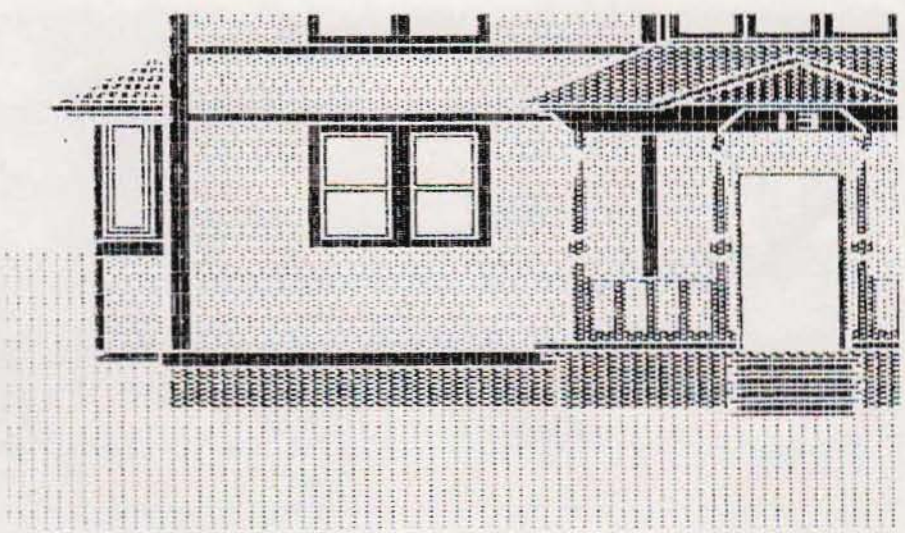
sur papier, inverser l'écran vidéo, car le dessin est tracé en blanc sur noir alors que l'imprimante ne peut écrire qu'en noir sur blanc. Si le graphique est composé d'un dessin où le sens tridimensionnel est obtenu avec des zones hachurées, l'absence d'inversion provoque des résultats catastrophiques.

La table traçante. C'est l'unité de sortie la plus usitée dans les applications de caractère technique. Elle fournit des graphiques d'une qualité comparable voire meilleure que ceux réalisés à la main.

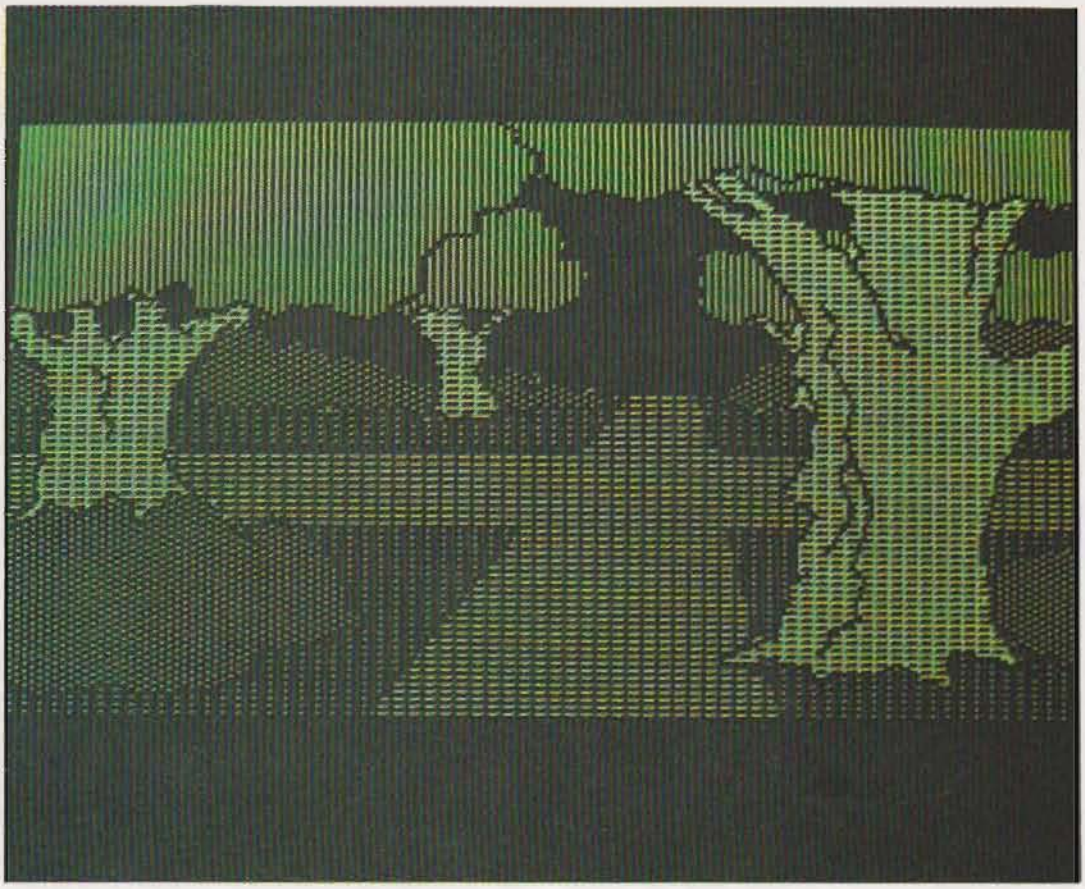
Les traceurs se subdivisent en deux catégories, selon le moyen employé pour tracer le dessin :

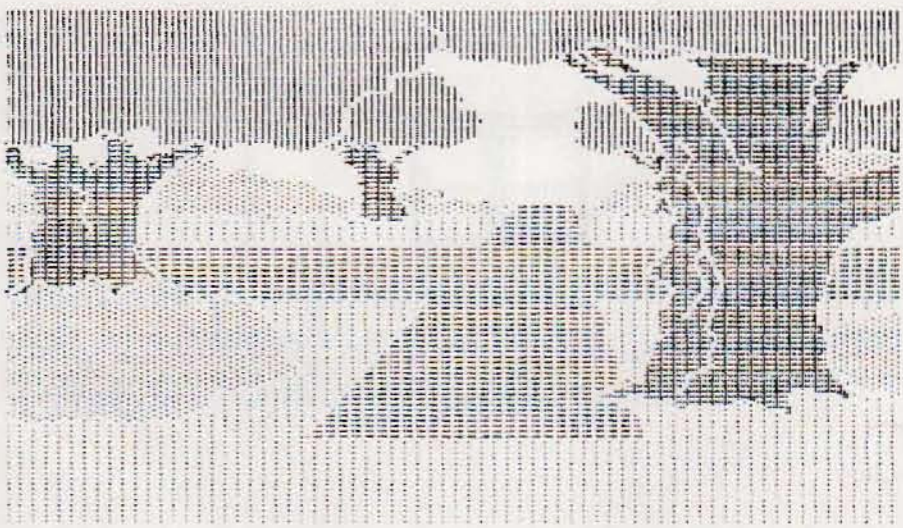
SORTIE SUR IMPRIMANTE D'UNE IMAGE GRAPHIQUE (1)





SORTIE SUR IMPRIMANTE D'UNE IMAGE GRAPHIQUE (2)

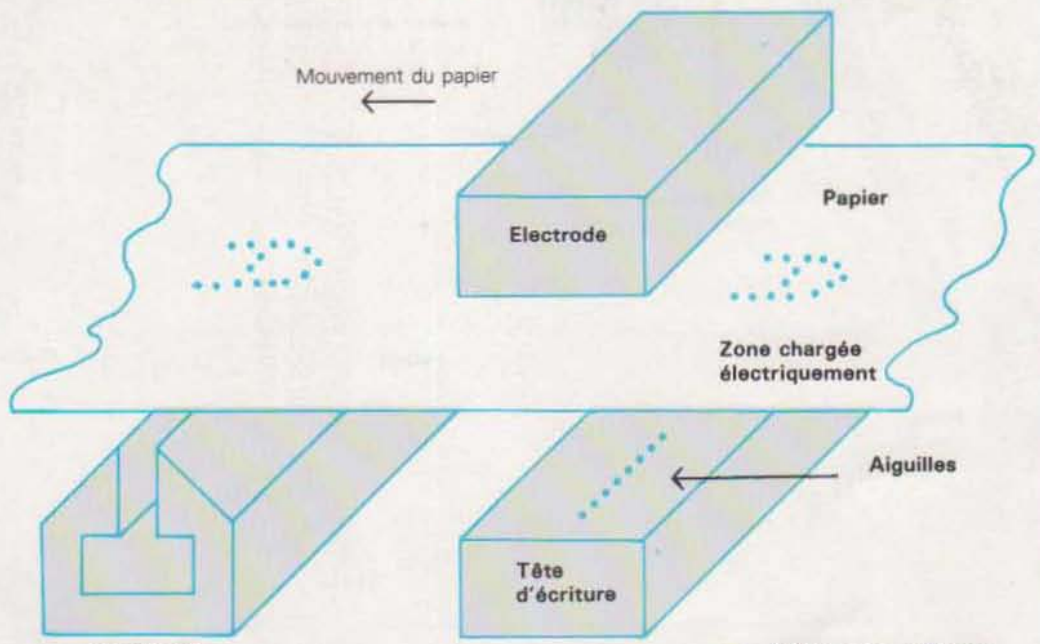




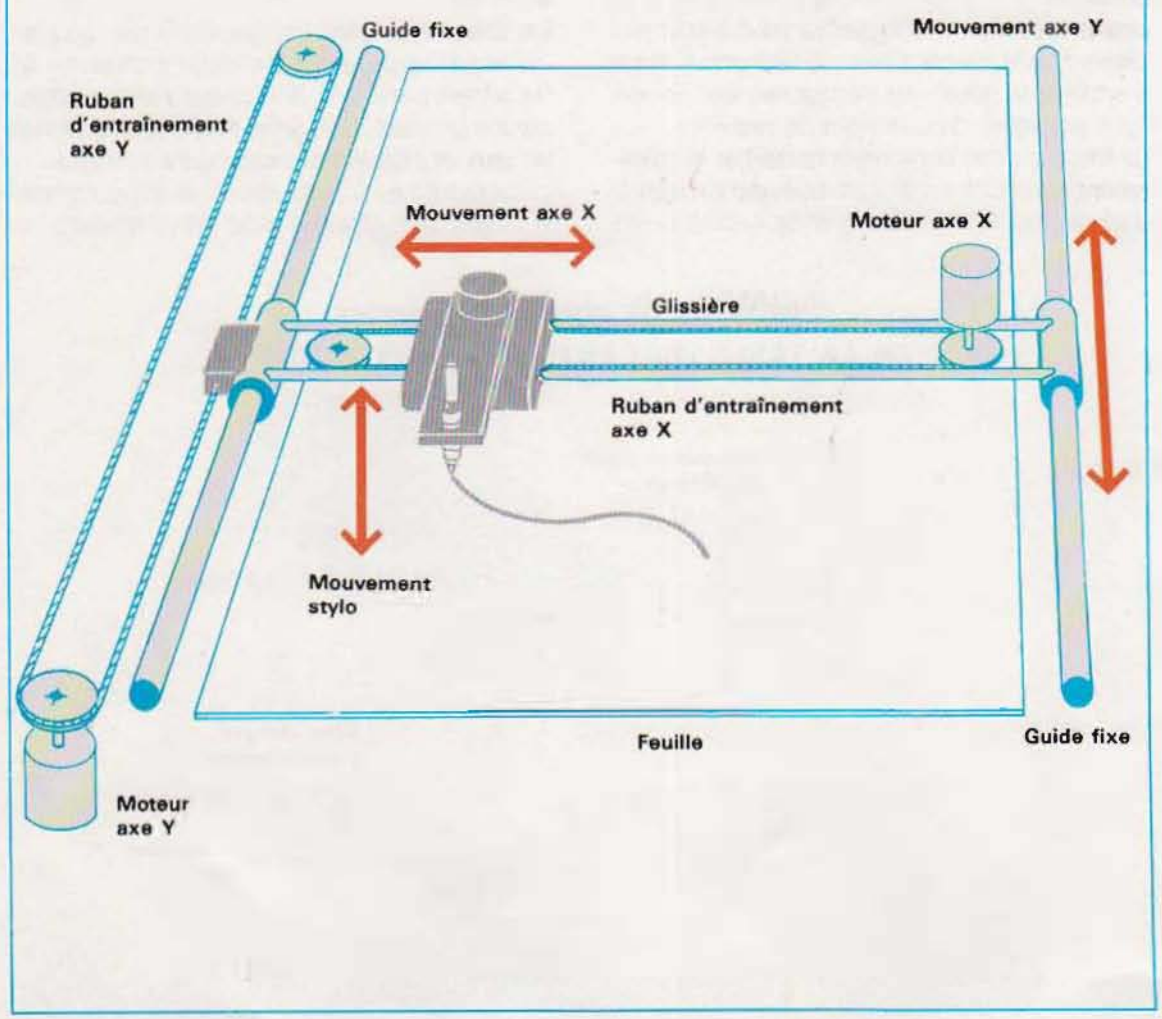
- traceurs avec photostyle
- traceurs électrostatiques

Dans les premiers, l'élément qui trace le dessin est un stylo très semblable à celui employé par les dessinateurs ; dans les électrostatiques, également appelés **rasters,** l'écriture est réalisée en chargeant électriquement les points du papier qui devront être colorés et en déposant un applicateur sur le papier. Ce procédé se rapproche de l'impression électrostatique (schéma ci-dessous). Ce type de périphérique présente deux inconvénients : le recours à un papier spécial et l'impossibilité de dessiner plusieurs fois le même point ; il faut donc, avant d'envoyer le dessin au traceur, le décomposer ligne par ligne (d'où le nom de raster).
Le traceur avec photostyle constitue le matériel le plus courant, car il associe une excellente qualité graphique à une grande facilité d'emploi. Le style se déplace dans toutes les directions et repasse sans difficulté sur des zones déjà partiellement dessinées. Cette caractéristique laisse une grande liberté de manœuvre à l'usager qui pilote son logiciel comme il l'entend, sans entrave imposée par la structure physique de la machine.
Les traceurs se classent également selon le type de mouvement effectué. Les plus répandus sont les traceurs à plat et ceux à tambour. D'autres, comme les traceurs à rouleau ou avec mouvement de papier, n'en sont que des variantes.
Le traceur à plat est constitué par un plan qui reçoit le papier et par deux « chemins de fer » fixes parallèles, le long desquels se déplace une glissière. Sur cette glissière, on a monté un chariot porte-stylo (voir figure p. 1402).
Les mouvements (X,Y) nécessaires pour tracer le dessin sont obtenus avec deux moteurs : le

PRINCIPE DE FONCTIONNEMENT DE LA TABLE TRAÇANTE ELECTROSTATIQUE



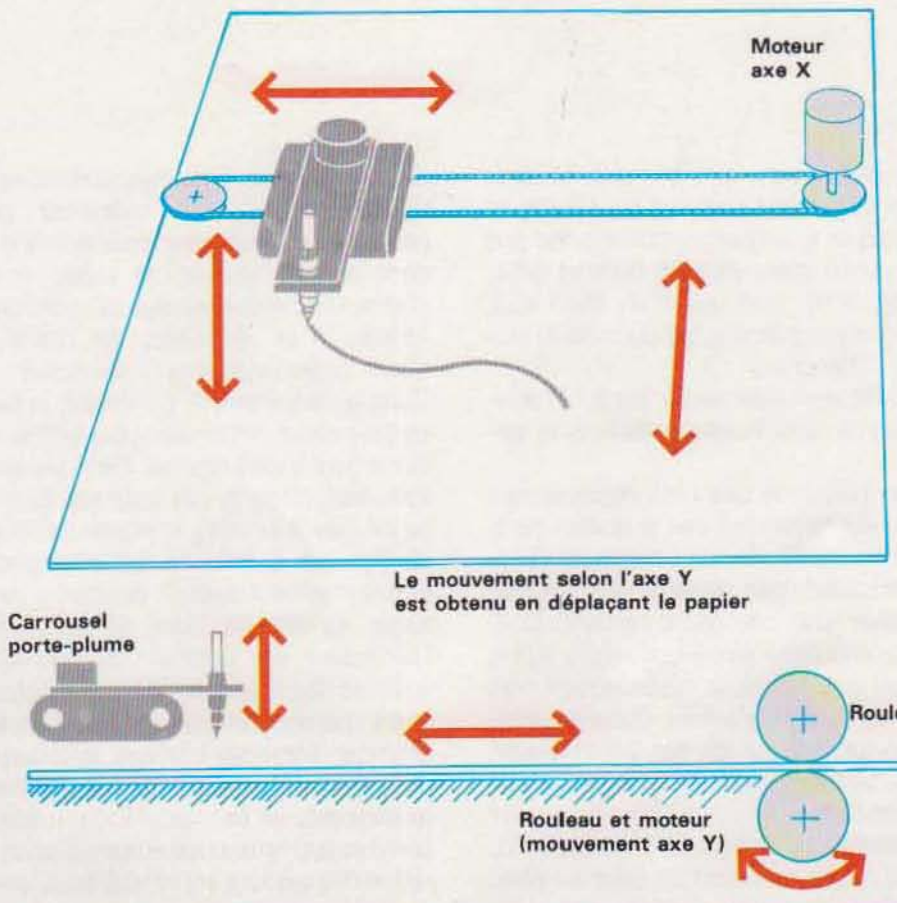
SCHEMA DU TRACEUR A PLAT

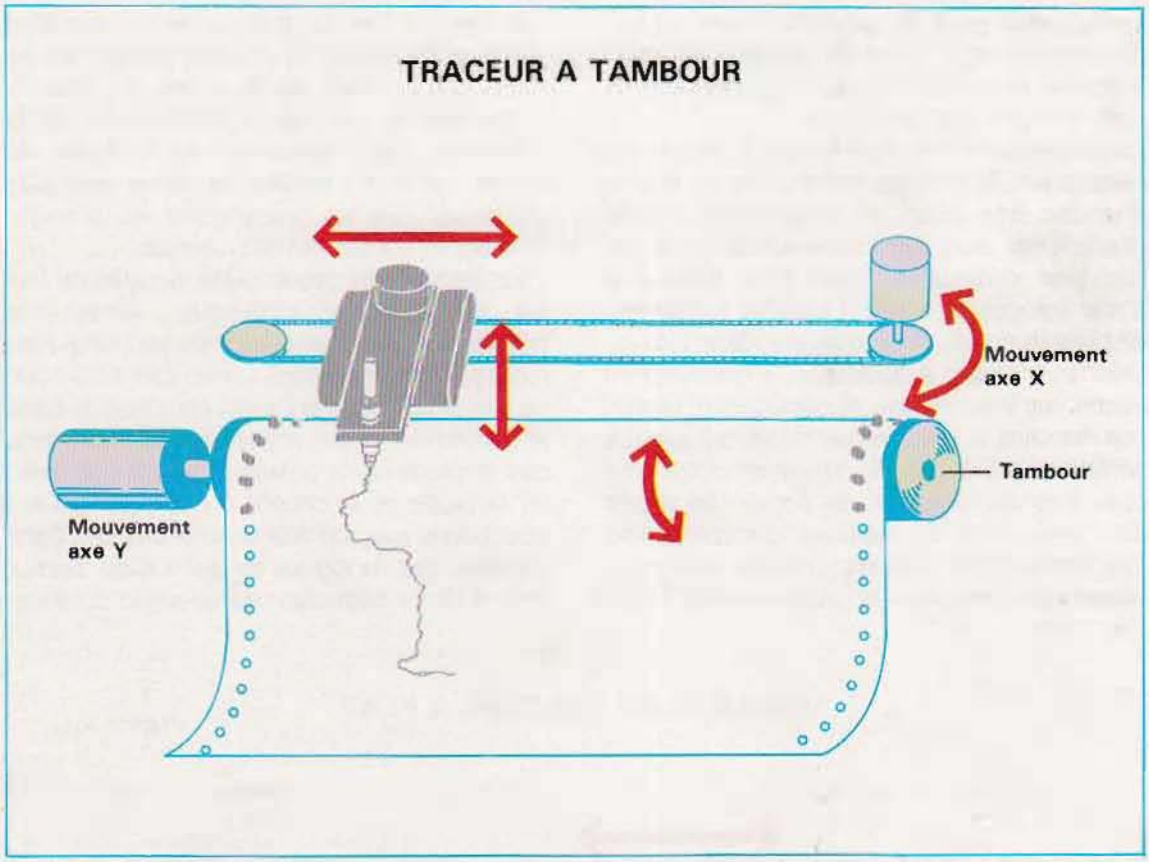


premier déplace la glissière (axe Y), le second déplace le chariot le long de la glissière (axe X). A ces mouvements, s'ajoute le déplacement perpendiculaire de la plume, obtenu grâce à un électro-aimant monté sur le chariot.
La mécanique même de ce périphérique montre à quel point son emploi est extrêmement simple. Deux instructions seulement sont nécessaires, l'une pour déplacer le stylo à un point donné X,Y, et l'autre pour préciser si ce mouvement doit être fait avec le stylo levé ou baissé. Dans le premier cas, on obtiendra un simple positionnement du style (sans traçage de signe) ; dans le second cas (stylo baissé),

s'effectue le tracé du segment reliant la position actuelle et le point de coordonnées X,Y. Avec ces deux instructions, on parvient à réaliser n'importe quel graphique.
Dans certains types de traceurs à plat, on ne dispose pas du mouvement de glissière le long d'un des deux axes ; le déplacement correspondant est alors obtenu en déplaçant le papier (voir ci-dessous). Mais il ne s'agit que d'une variante de matériel puisque l'utilisateur emploie le dispositif de manière identique.
Dans le **traceur à tambour,** le mouvement selon l'axe Y est obtenu en déplaçant le papier, comme dans la variante des traceurs à plat. La différence réside dans le moyen employé pour obtenir le déplacement du papier. Le papier doit avoir des perforations d'entraînement (comme le papier pour imprimantes) et le mouvement est obtenu par un rouleau denté. Naturellement, le rouleau doit pouvoir tourner dans les deux directions, de manière à effectuer les déplacements dans les deux sens de l'axe Y.
L'avantage de ce type de traceur vient de la dimension (sans limitation) de la feuille de papier ; mais il s'agit de machines bien plus coûteuses que les précédentes et rarement employées sur les petits systèmes.
Tous les traceurs peuvent être équipés de stylos de couleurs différentes. Deux cas possibles : dans le premier tous les stylos sont montés en même temps sur un carrousel et on ne baisse que celui qui a été sélectionné. Dans le second, les divers stylos sont déposés dans des emplacements prévus à cet effet au bord de la feuille et le chariot prend celui qui est sélectionné avec un mécanisme à déclic. Dans les deux cas, le logiciel de gestion du traceur doit prévoir l'instruction de sélection du stylo.

VARIANTE DU TRACEUR A PLAT





Il faut prendre garde au fait que le mot « stylo » est également employé pour indiquer le « stylo logique », auquel ne correspond pas de stylo physique mais un type de trait différent. Il s'agit donc, non pas d'un stylo réel, mais d'un sous-programme qui contrôle le tracé de lignes différentes.

Deux autres facteurs intervenant dans l'évaluation d'un traceur sont la **résolution** et la **répétitivité.**

La résolution indique le plus petit déplacement possible, qui est déterminé par la qualité de la mécanique, en particulier des moteurs. D'un type très particulier, ces derniers sont en mesure de réaliser une très petite rotation (pas) pour chaque impulsion électrique reçue : plus la rotation est petite, plus le déplacement correspondant du stylo sera précis. Comme ordre de grandeur, un traceur de qualité moyenne permet des déplacements de quelques dixièmes de millimètre.

Le second paramètre, qui a trait à la répétition, indique avec quelle précision on peut se positionner plusieurs fois sur un même point. En d'autres termes, à un couple de valeurs numériques envoyées par l'ordinateur, ne correspond pas toujours la même position physique de la plume. Dans ce cas aussi, l'erreur a une origine mécanique et elle est normalement inférieure à la résolution. En réalité, elle est moins déterminante qu'il n'y paraît.

Dans les applications générales, le résultat est certainement nettement plus précis que n'importe quel travail manuel. Dans les applications spéciales, comme par exemple la préparation de circuits imprimés, le dessin est d'abord tracé à grande échelle, la réduction photographique permettant dans un deuxième temps d'éliminer les défauts. Dans ce cas, le recours à l'ordinateur est vraiment indispensable car il autorise des essais à différentes échelles, sans autre intervention que la mise en route du programme. Dans les traceurs spéciaux à **laser,** un rayon lumineux remplace le photo-style. Naturellement, on fait appel à du matériel photosensible (comme celui employé pour la préparation des circuits imprimés) avec une structure mécanique étudiée pour obtenir une préci-

sion très poussée (pour certains traceurs, on parvient à 1/10 000 " avec un stylo à lumière **laser**).

Le moniteur monochrome. La technologie employée dans la fabrication des écrans dépend du type d'application auquel ils sont destinés. D'où diversité et différence de coût.
L'organe essentiel d'un poste de télévision est le tube à rayons cathodiques (CRT pour Cathode Ray Tube). C'est un convertisseur transformant un signal électrique en image sur un écran luminescent.
Le fonctionnement d'un CRT repose sur les propriétés de certaines substances (comme par exemple les sels de phosphore) qui émettent de la lumière visible quand elles reçoivent un faisceau d'électrons.
Page suivante, nous avons représenté 2 types de tubes :

- CRT à déflexion électromagnétique
- CRT à déflexion électrostatique

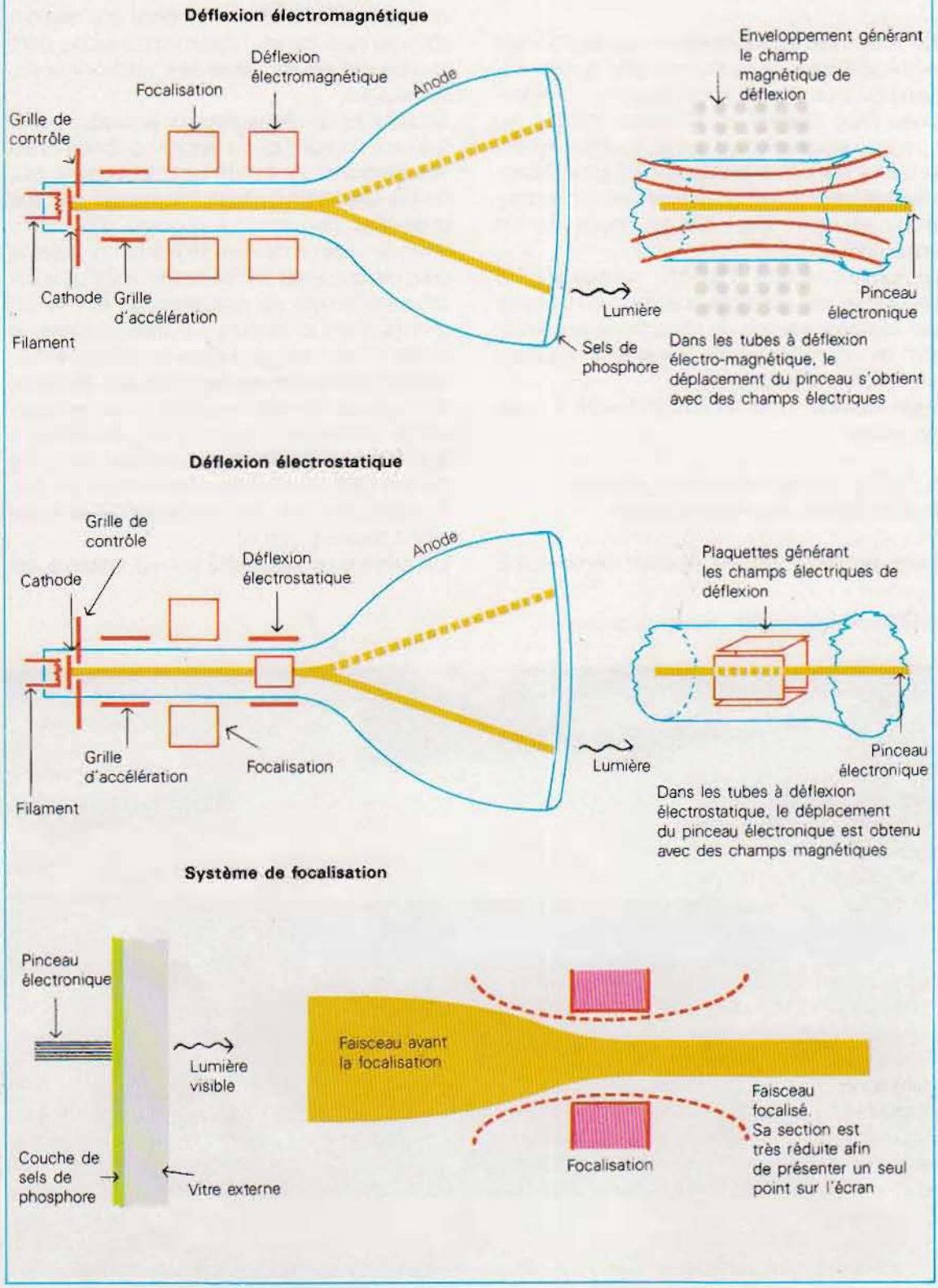
Dans les deux cas, un filament semblable à celui des ampoules ordinaires est parcouru par un courant électrique qui le rend incandescent. A proximité (parfois il s'agit d'un petit cube coaxial au filament), on interpose une matière spéciale sous forme d'une mince couche, dont la propriété est de libérer des électrons si elle est excitée.
Sous l'effet du réchauffement provoqué par le filament, le composant appelé cathode émet des électrons, qui sont mis en mouvement vers l'autre extrémité du tube (dit anode) par une grille d'accélération. La quantité d'électrons envoyés vers l'anode est réglée par la grille de contrôle qui, avec son potentiel électrique, fait office d'élément de régulation du flux tel un robinet. Sans le secours d'autres éléments, le nuage d'électrons produit par la cathode à destination de l'anode exciterait les sels de phosphore de manière désordonnée, d'où une luminosité diffuse sur l'écran. C'est pourquoi il a fallu focaliser les électrons avec une électrode dans le CRT, de manière à n'atteindre les sels de phosphore que sur une surface très faible (voir schéma p. 1406).
L'ensemble est complété par un système qui

Une imprimante-table traçante couleur.



Marka

SCHEMA D'UN TUBE CATHODIQUE



déplace le faisceau focalisé d'électrons (dit pinceau). Ce résultat s'obtient avec un champ soit magnétique (déflexion magnétique) soit électrique (déflexion électrostatique).
Les champs (magnétique ou électrique) sont perpendiculaires, de manière à réaliser des déplacements de pinceau cartésiens (axes X,Y) balayant toute la surface de l'écran.
Les composants électriques qui règlent l'émission, l'accélération et la focalisation du faisceau d'électrons sont gérés par des dispositifs indépendants de l'ordinateur. Les éléments de déflexion sont, de leur côté, commandés par l'ordinateur : grâce à des interfaces spécifiques, les données sont converties en signaux électriques qui règlent la position du pinceau sur l'écran.
La génération d'une image s'obtient en déplaçant le pinceau électronique selon les instructions envoyées par l'ordinateur sous la forme de signaux de positionnement. Comme pour les traceurs, l'ordinateur doit préciser si un déplacement doit être reproduit avec un trait visualisé ou non. L'émission de la lumière, en un point de l'écran, dure tant que le pinceau électronique excite ce point. Si on doit visualiser deux points sur une même ligne horizontale, à l'instant où le pinceau atteint le second point, les phosphores du premier ne sont plus excités.
Le tracé d'un segment se réalisant par l'excitation successive des points le composant, il faut éviter que, lors de la visualisation des derniers points, les premiers aient perdu leur luminosité.
Une technique simple consiste à exploiter la persistance de l'image, c'est-à-dire la propriété qu'ont les sels de phosphore de continuer à émettre de la lumière même après l'excitation. Mais cette persistance n'excède pas un certain temps. On a alors recours à la technique du rafraîchissement qui consiste à redessiner l'image plusieurs fois par seconde. En général, on ajuste la persistance à la fréquence du rafraîchissement pour obtenir des images de qualité acceptable, en éliminant dans la mesure du possible les phénomènes de scintillement optique. Pour tenir compte de la persistance de l'image sur la rétine, on choisit en général une fréquence égale à 30 Hz : une image visualisée est balayée 30 fois par seconde.
La conservation de l'image avec la technique du rafraîchissement immobilise une zone de mémoire consacrée à l'écran. Elle permet de mémoriser l'état de chaque point. Le système d'exploitation prélève les données de cette mémoire et active en conséquence les divers points de l'écran, en répétant la lecture à 30 Hz. Inconvénient de cette technique : une zone mémoire importante est retirée aux programmes d'application. Son atout : des possibilités importantes qui permettent à l'utilisateur de modifier la présentation graphique des données en intervenant directement sur la mémoire.
Le rafraîchissement reste interdit à certaines applications quand, par exemple, la résolution demandée exige un espace mémoire dédié trop grand. Dans ce cas, on adopte un tube particulier, dit CRT à mémoire, qui a la propriété de conserver l'image pendant un délai remarquablement long. Mais il a aussi son handicap ; il n'autorise ni l'effacement ni la modification d'une partie de l'écran.
Avec le rafraîchissement, il suffit de modifier le contenu de la mémoire : ce qui est présenté à l'écran disparaît et peut être recomposé avec les éventuelles modifications. Dans les tubes à mémoire l'image est fixée en exploitant la persistance : elle ne peut être mise à jour qu'après effacement complet.
Une dernière classification des moniteurs, reposant sur la technique de formation de l'image, propose deux types de rafraîchissement :

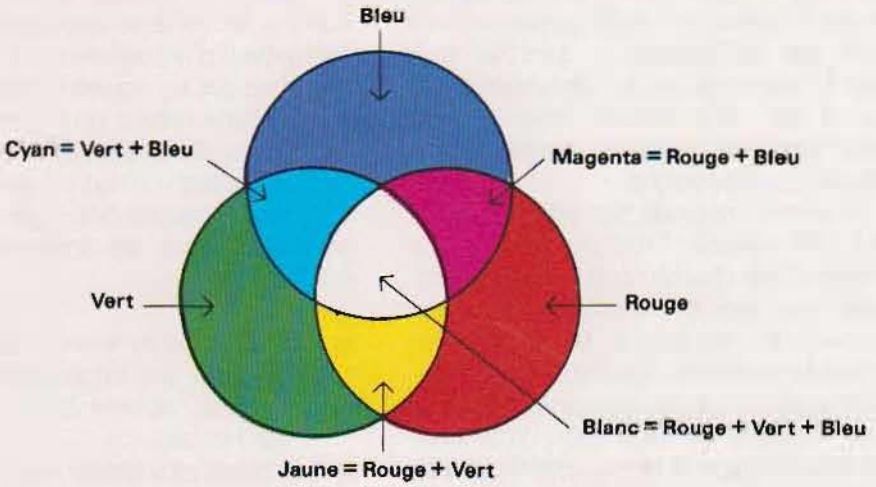
- par vecteurs, normalement indiquée par le terme **stroke**
- par points, appelée **raster** (mode récurrent)

Il existe des variantes de ces méthodes mais peu utilisées sinon abandonnées. C'est le cas de la méthode **starbust**, qui a été délaissée pour sa mauvaise résolution, au profit de techniques certes moins économes en mémoire, mais plus évoluées.
L'augmentation de la capacité de mémoire ne constitue désormais plus un obstacle en termes de coût. On n'est limité que par la capacité d'adressage de l'UC. La méthode de stroke (vecteurs) consiste à guider le pinceau électronique dans n'importe quelle zone de l'écran sans ordre de priorité particulier. Chaque caractère ou élément de dessin est obtenu en déplaçant le pinceau selon une série de mouvements élémentaires.

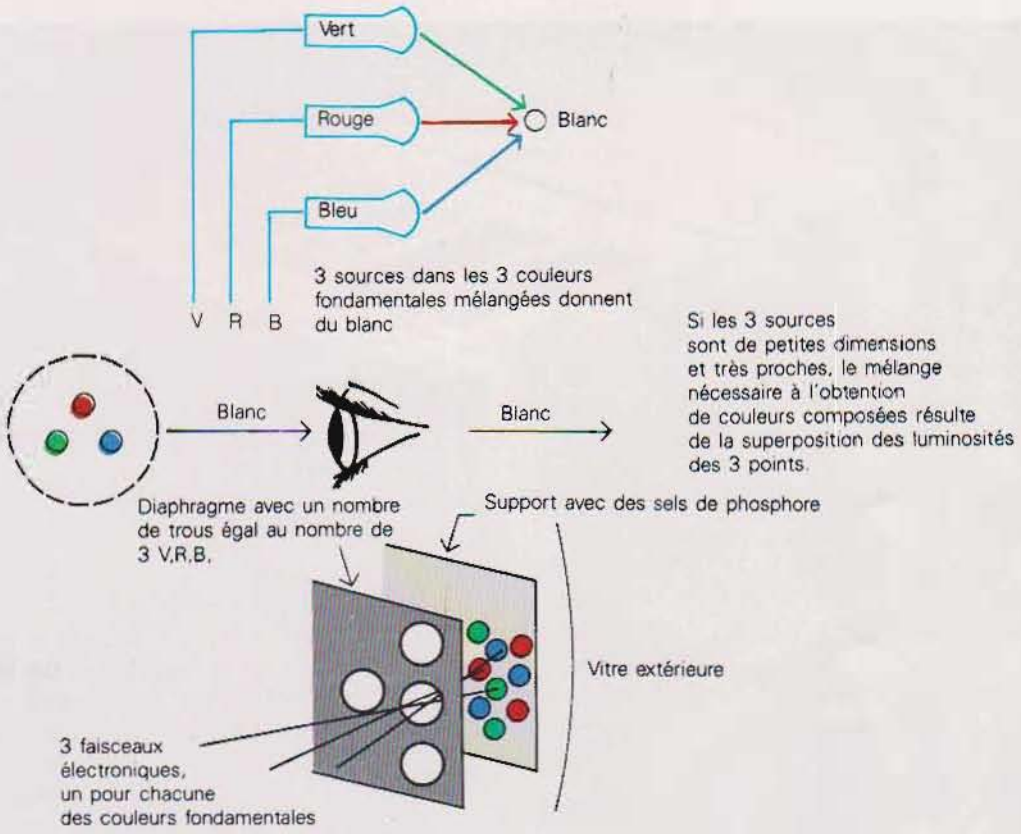
Dans le mode raster (récurrent), l'écran est continuellement balayé par le pinceau électronique, formant une série de lignes imaginaires très rapprochées et presque horizontales. Pour obtenir un point sur l'écran, on envoie un signal, synchronisé avec la position du pinceau, qui inhibe ou active la visualisation.
Avec cette méthode, chaque élément de figure est reproduit en activant de manière sélective une matrice de points, comme dans les imprimantes à aiguilles. La qualité graphique dépend directement du nombre de points adressables sur l'écran. Les valeurs usuelles sont, pour la matrice de caractères, 5 × 7 ou 7 × 9 points, et pour l'écran tout entier de quelques dizaines de milliers à plusieurs millions de points. Normalement, on indique la résolution graphique d'un écran en précisant le nombre de points horizontaux et verticaux.
Exemple : un écran 280 × 190 visualise 280 points consécutifs à l'horizontale et 190 à la verticale, soit un total de 53 000 points pour l'écran.
Une telle définition est peu adaptée aux applications graphiques où les valeurs courantes avoisinent les 1000 × 1000 points. Cette limite est largement dépassée dans certaines applications. Le nombre de points écran est fonction de la granulosité des sels de phosphore de l'écran et surtout de la capacité mémoire.
Pour avoir un ordre de grandeur, prenons un écran 1000 × 1000. Puisque l'état de chaque point (actif/non actif) peut être mémorisé dans un bit, il faut au moins 10^6 bits. Pour une machine à 16 bits, chaque emplacement de mémoire peut justement contenir 16 bits. La taille mémoire dédiée à l'écran sera de $10^6/16 =$ 62 500 octets, l'équivalent de toute la zone adressable pour une UC de 16 bits employée pour des programmes d'applications. Pour une machine à mots de 8 bits, les emplacements mémoire nécessaires sont de 125 000, soit environ le double de la capacité d'adressage. Ces difficultés d'adressage limitent donc la programmation.
Dans leur aspect graphique, les micro-ordinateurs et ordinateurs personnels appartiennent à deux catégories : ceux dont la mémorisation de l'écran passe par une occupation partielle de la mémoire (zone utilisateur) ; ceux qui disposent de ressources séparées. L'avantage des premières est d'offrir un accès facile, à l'utilisateur, des données graphiques. La mémoire est adressable par l'interprète ou le compilateur, entièrement gérée par le programme. La zone mémoire disponible est diminuée de l'espace dédié à la partie graphique. C'est pour cela qu'on ne prévoit généralement pas de valeurs haute résolution.
Dans la seconde catégorie, l'image est conservée dans une zone mémoire n'appartenant pas à l'aire utilisateur, qui garde ainsi toute la partie adressable. Mais on ne peut lire et écrire des données graphiques qu'en recourant à des sous-programmes spécifiques du système, avec des instructions de haut niveau prévues par la machine. En pratique, cela se traduit par une moindre souplesse dans certaines applications, voire par une plus grande lenteur d'exécution des programmes.

Les moniteurs couleur. Le principe de fonctionnement et les techniques de formation et de persistance de l'image sont les mêmes que celles qui sont employées dans les monochromes. En revanche, la structure physique et les composants nécessaires à leur fonctionnement en sont très différents.
La formation d'une image couleur s'obtient de deux manières. On emploie généralement la méthode additive (la même que pour les téléviseurs couleur) alors qu'on retrouve la technique soustractive principalement en photographie et dans les écrans graphiques pour les grands systèmes.
On compte les couleurs primaires et les couleurs composées. On appelle primaires les couleurs qui, combinées de diverses manières, permettent d'en obtenir d'autres.
Dans le graphique ci-contre, on a décrit la technique de formation de certaines couleurs avec la méthode additive, en prenant comme couleurs primaires le rouge, le vert et le bleu. Les couleurs secondaires (blanc, jaune, cyan, magenta) sont obtenues par additions des couleurs fondamentales ; les autres sont générées en variant les proportions ajoutées : par exemple, le jaune est obtenu en mélangeant le vert et le rouge en parts égales, alors que l'orange s'obtient en augmentant la part de rouge.
Dans le second graphique, on a représenté le principe de formation des images couleur. Un point apparaît blanc (ou d'une autre couleur composée) s'il est le point de convergence de

COMPOSITION DES COULEURS AVEC LA TECHNIQUE ADDITIVE



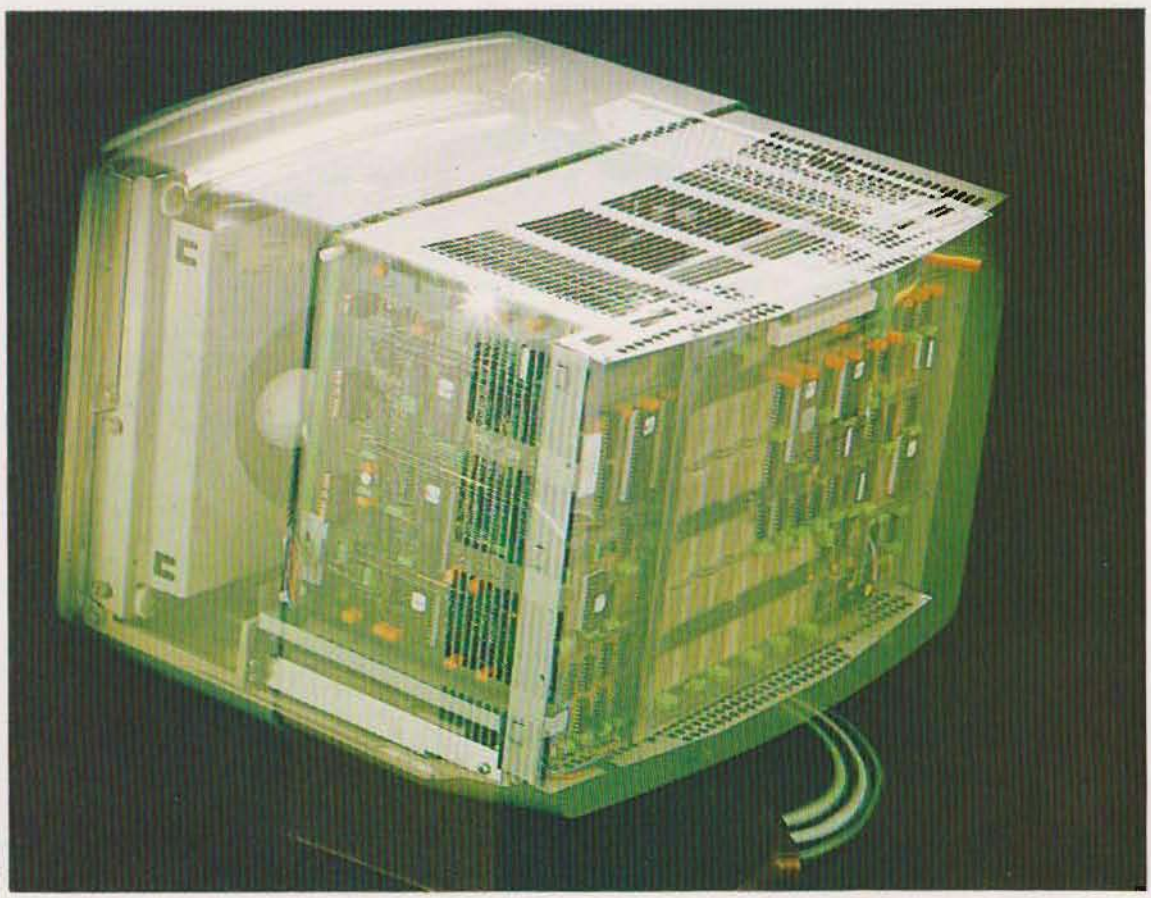
FORMATION DES COULEURS DANS UN TELEVISEUR A TROIS FAISCEAUX



faisceaux lumineux de trois écrans différents, chacun avec une émission lumineuse dans l'une des trois couleurs fondamentales. De même, on peut obtenir un mélange en excitant des particules de phosphore, dans les trois couleurs fondamentales, de dimensions très réduites et très rapprochées : ainsi les trois faisceaux lumineux produits seront perçus comme un faisceau unique.
Cette deuxième méthode est plus employée dans les CRT couleur. Trois dispositifs génèrent et contrôlent chacun un faisceau qui active un des trois types de phosphore ; en réglant le flux d'électrons sur chacun des phosphores on obtient la génération de couleurs composées. Un second type ne contient qu'un seul genre de dispositif de génération du pinceau électronique (on l'appelle couramment dans les CRT « canon électronique ») alors que l'écran conserve la structure à trois phosphores.
La sélection des couleurs s'opère en réglant la pénétration du faisceau dans les phosphores. Dans les projets prévoyant la visualisation d'images en couleur, il faut souvent tenir compte de certains effets physiologiques, qui permettent d'obtenir de bons résultats en économisant sur les couleurs tout en ne réservant qu'une zone réduite de mémoire, pour les informations nécessaires à la gestion de l'écran. Les principales précautions sont dictées par les caractéristiques suivantes (semblables à ce qui se produit pour les émissions télévisées en couleur) :

- Les zones de dimensions plus grandes, correspondant aux informations chromatiques principales, doivent utiliser les 3 couleurs fondamentales.
- Les zones plus petites n'utilisent que 2 couleurs.
- Les zones de dimensions minimes peuvent ne pas être colorées dans la mesure où,

Anatomie d'un moniteur couleur, avec circuiterie.



Ericsson

au-delà d'une certaine dimension, l'œil ne perçoit plus les couleurs mais seulement la brillance.

L'espace mémoire nécessaire dans une télévision couleur est nettement plus grand que dans un modèle monochrome. Dans ce dernier, un seul bit indique l'état d'un point de l'écran, son état physique ne prenant que deux valeurs (allumé ou éteint).
Dans les écrans couleur, pour chacun des points il faut également mémoriser la composition chromatique, c'est-à-dire ses couleurs fondamentales. Les couleurs composables ne sont que théoriquement limitées par la taille de la mémoire. Dans les systèmes de qualité moyenne, on adopte entre 4 et 8 couleurs.
Pour mémoriser 4 combinaisons de couleurs fondamentales, on a besoin de 2 bits par point (avec 2 bits on peut écrire les valeurs binaires 00, 01, 10, 11 qui correspondent aux décimales 1, 2, 3, 4) et on obtient donc un doublement de la mémoire nécessaire par rapport au cas monochrome.
Dans le système à 8 couleurs, il faut 3 bits, et la zone de mémoire occupée triple (sur un écran de 1000 × 1000, on occupe donc environ 180 Koctets de 16 bits).
Pour mettre en mémoire une quantité d'informations aussi grande, il faut utiliser diverses pages de mémoire, chacune d'une grandeur (en nombre de bits) égale au nombre de points de l'écran : le nombre de pages doit être égal au nombre de points de l'écran : le nombre de pages doit être égal au nombre de bits réservés aux combinaisons chromatiques à réaliser. Ainsi, pour une représentation en quadrichromie, on a besoin de 2 pages ; pour 8 couleurs, 3 pages. Ces pages sont considérées, par le système d'exploitation, comme des matrices dont la composition génère, pour chaque point de l'écran, le mélange de couleurs fondamentales.
Dans d'autres types de machines, la zone de mémoire destinée au graphique est déjà dimensionnée pour héberger les indicateurs de couleur et elle n'est pas divisée en pages. Cette différence de structure ne se manifeste pas pour l'utilisateur s'il emploie les instructions de haut niveau prévues dans le langage machine. Par contre, elle est à l'origine de complications dans l'adressage de la mémoire s'il travaille en Assembleur ou en DMA.

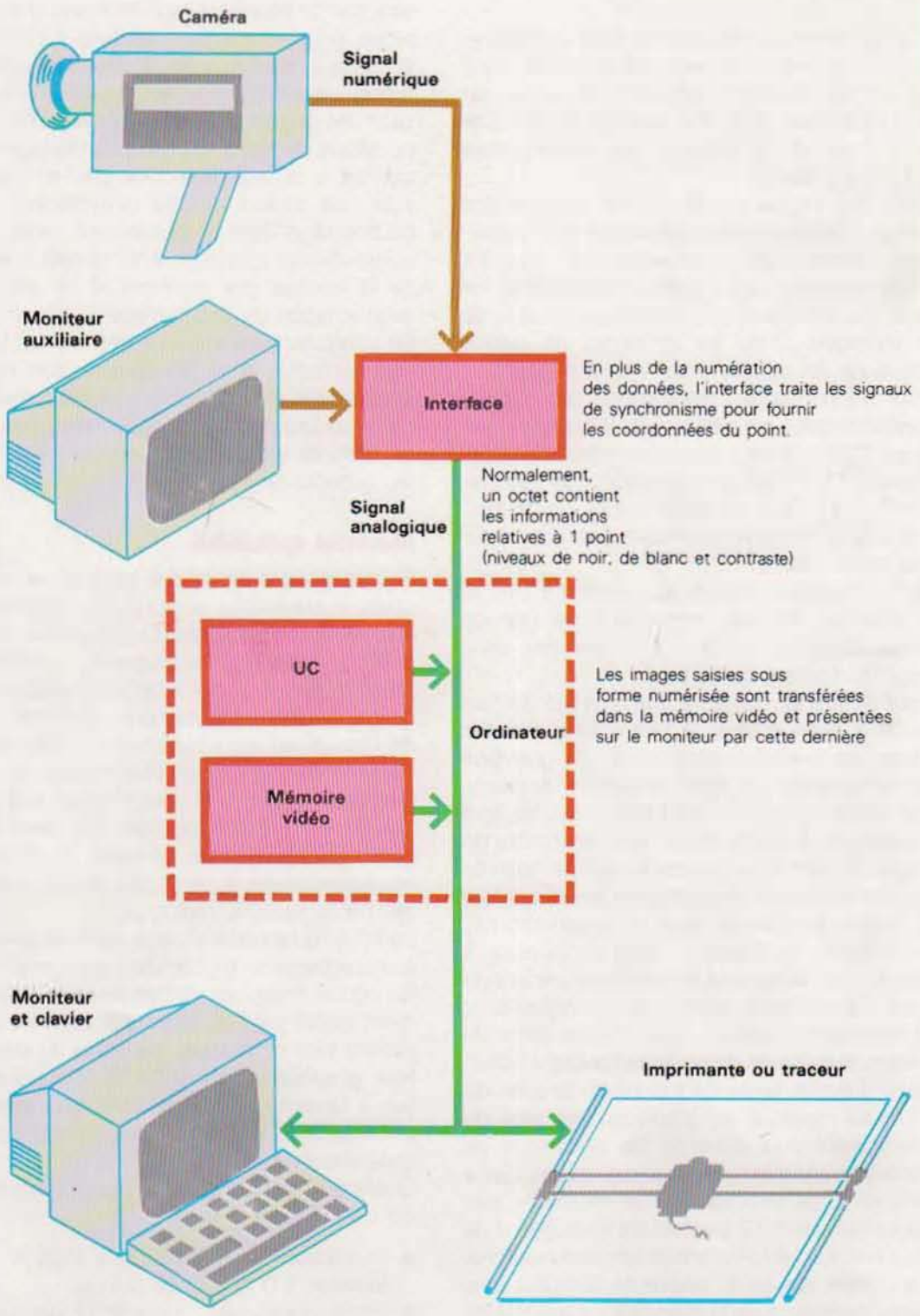
Un autre problème vient de la transposition du graphique sur le papier. Normalement, les traceurs ou les imprimantes graphiques ne disposent pas de plus de 2 ou 3 couleurs. Une technique, pour obtenir des résultats acceptables, consiste à reproduire les diverses tonalités au moyen d'impressions successives, chacune dans une couleur avec un symbole prenant plus ou moins de place selon le pourcentage de la couleur à obtenir. Par exemple, en utilisant pour une couleur donnée le symbole . , son pourcentage dans le dessin sera faible, alors qu'en utilisant un symbole qui remplit davantage la matrice (par exemple X) on aura une augmentation de pourcentage.
En intercalant, de manière très serrée, les divers symboles dans les couleurs, on obtient l'impression d'un dessin comportant davantage de couleurs que celle, réellement employées (la méthode est la même que pour les différentes nuances de noir et de blanc).

Matériel spécialisé

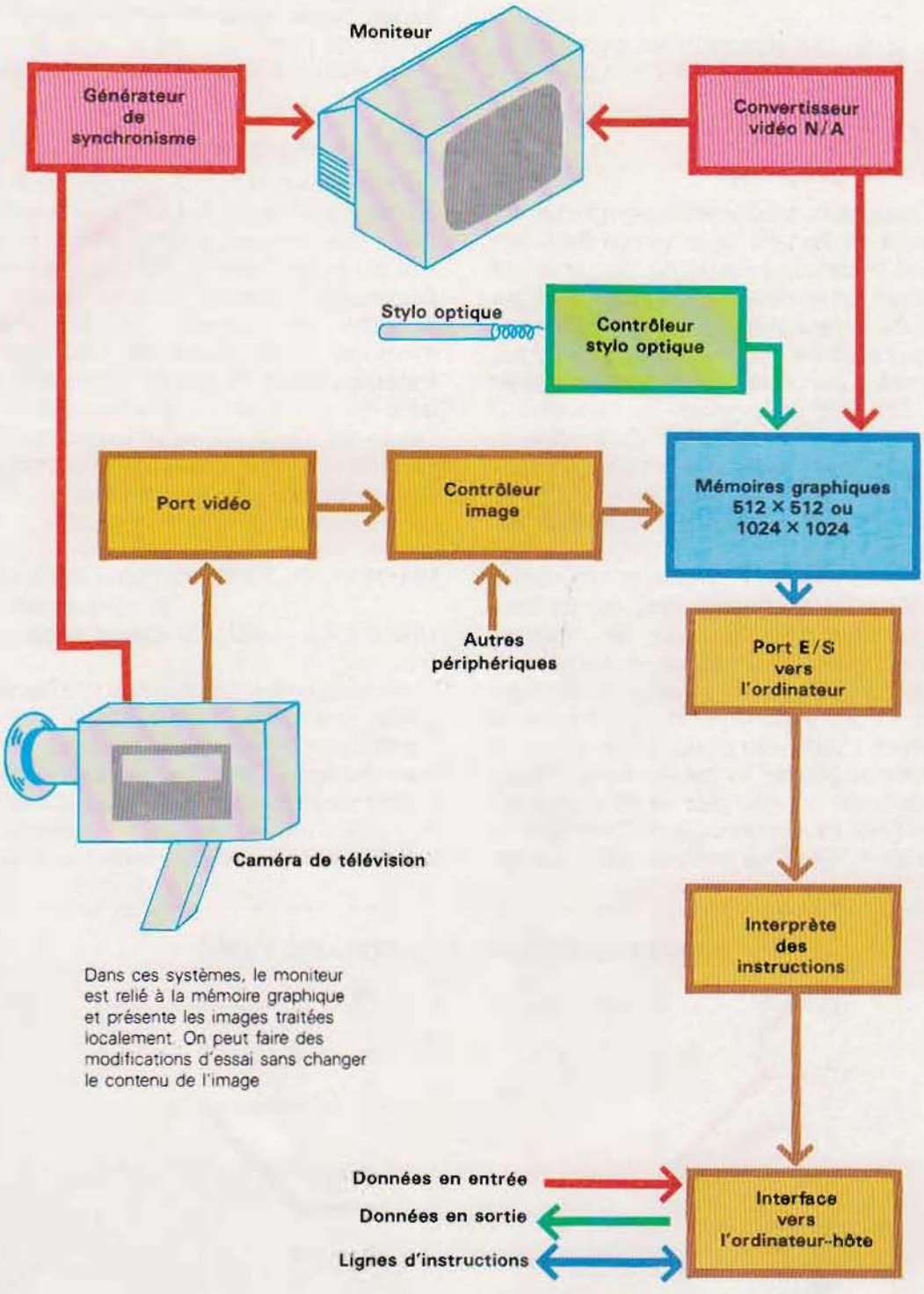
Etant donnée la variété des applications graphiques, il existe des composants matériels ou des systèmes complets dédiés. Ainsi, pour la saisie d'images relativement complexes (photos, dessins), des interfaces spéciales relient la machine à la caméra. La numérisation de l'image est alors automatique. Elle permet l'acquisition de figures très complexes dans des délais très courts. Les champs d'application de cette technique sont très diversifiés : graphisme publicitaire, analyse de données météorologiques à partir des photos transmises par le satellite, radiologie...
Le schéma représenté page suivante se réfère à une application d'ordinateur personnel. Dans les petites machines, le traitement est entièrement réalisé par l'UC alors que, dans les applications plus complexes, on utilise un **contrôleur graphique** (voir p. 1413). Celui-ci effectue, à lui seul, une grande partie du traitement et utilise l'ordinateur-hôte uniquement comme superviseur.
Quelques caractéristiques de ces systèmes sont énumérées ci-dessous :

- Résolution : de 512 × 512 à 1024 × 1024 (environ 1 000 000 de points).
- Pages graphiques : jusqu'à 12 (les valeurs usuelles sont 6/8).

ACQUISITION D'IMAGES COMPLEXES



CONTROLEUR D'ECRAN GRAPHIQUE



Dans ces systèmes, le moniteur est relié à la mémoire graphique et présente les images traitées localement. On peut faire des modifications d'essai sans changer le contenu de l'image

- Couleurs : de 64 000 à 16 000 000 (nuances quasiment illimitées).
- Vitesse de transfert des données : environ 1 000 000 pixels par seconde.

Le coût de ces équipements est, grosso modo, 20 fois plus élevé que celui d'un ordinateur.

Instruction Basic pour le graphique

Les instructions spécifiques aux graphiques dépendent étroitement de la version Basic employée. A partir du moment où, dans la version standard, on ne prévoit pas d'instructions graphiques, il n'existe aucun point de référence ; chaque machine utilise ses propres symboles. La gestion d'un écran en mode graphique est très différente de la gestion de caractères et elle comporte des difficultés d'adressage de mémoire, sans parler de la complexité découlant de la couleur. On peut néanmoins définir des méthodologies générales applicables à la majorité des cas.
Dans toute application graphique, il faut disposer d'une série d'instructions, ou de sous-programmes pour accomplir des fonctions élémentaires ; un graphique, quel qu'il soit, s'obtiendra par une combinaison de ces fonctions. La plus importante consiste à tracer un segment à partir des points extrêmes, ce qui permet de générer toutes les autres figures. Par la suite, les exemples seront étudiés sur deux types de machines : le 2010 de Siprel et le M20 d'Olivetti. La première utilise des instructions communes à l'Apple et à ses compatibles et la seconde des instructions spécifiques.
En comparant les deux formes, on mettra en évidence les analogies et les méthodes qui permettent de passer de l'une à l'autre. Pour les autres micros, il suffit de consulter le manuel.

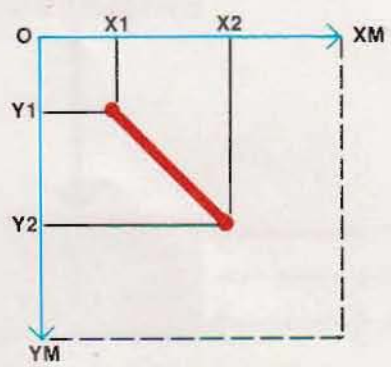
Tracé de segments

L'écran vidéo constitue le périphérique le plus commun dans les graphiques : c'est pour cette raison que la plupart des instructions se réfèrent à l'écran. Dans les autres périphériques (imprimante graphique, traceur...) et sauf cas particulier (en général sur de grosses machines) on ne trouve pas facilement de sous-programme de gestion. L'utilisateur doit donc les écrire lui-même. Chaque point de l'écran est répété par un couple de coordonnées. Pour dessiner un segment, il faut préciser les coordonnées des deux extrémités.
Les instructions de base sont du type :

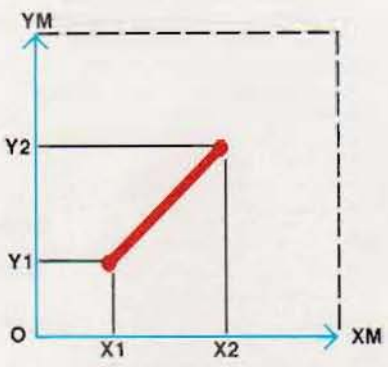
HPLOT X1, Y1, TO X2, Y2 (Siprel 2010, Apple et compatibles)
LINE (X1, Y1) — (X2, Y2) (Olivetti M20)

Comme on peut le constater, la syntaxe est la même, seul l'aspect formel change.
Les plus adaptées pour le système de coordonnées de l'écran : la différence réside essentiellement dans l'orientation de l'axe y. Il faut garder cela à l'esprit quand on mentionne les coordonnées des points concernés. Dans le

DIMENSIONNEMENT DES ECRANS VIDEO



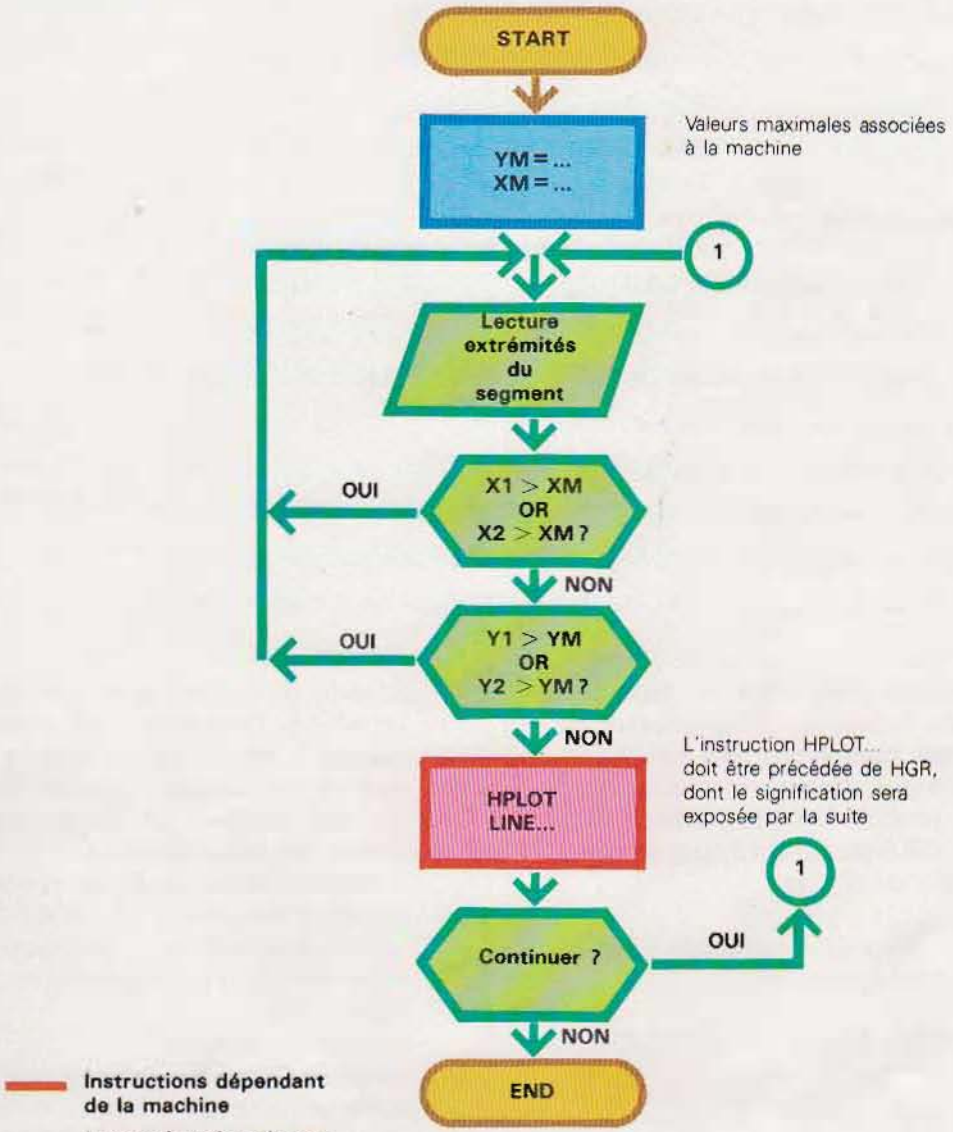
HPLOT X1,Y1 TO X2,Y2



LINE (X1,Y1) – (X2,Y2)

Siprel 2010, l'origine des axes est en haut à gauche, dans la M20, elle se trouve en bas à gauche. L'axe x a toujours la même orientation. Dans la première machine, en augmentant la valeur de y, l'extrémité initiale du segment se déplace vers le bas, dans la seconde vers le haut. La figure ci-dessous représente l'organigramme d'un programme de dessin d'un segment à partir des points définis par l'utilisateur coordonnées. La seule particularité de ces programmes réside dans le contrôle des valeurs introduites, qui a pour but de vérifier que les coordonnées des extrémités ne débordent pas de l'écran.

ORGANIGRAMME DU PROGRAMME DE CONSTRUCTION D'UN SEGMENT



TRACE D'UN SEGMENT

Version Siprel 2010

```
10 HOME
20 XM=279:YM=191
30 PRINT "coordonnées initiales: ";
40 INPUT X1, Y1
50 PRINT "coordonnées finales  : ";
60 INPUT X2, Y2
70 IF X1>XM OR X2>XM OR Y1>YM OR Y2>YM THEN GOTO 10
80 REM ****** dessin ******
90 HGR HCOLOR=3
100 HPLOT X1, Y1 TO X2, Y2
110 PRINT "Poursuivre (O/N): ";
120 INPUT A$
130 IF A$="O" THEN GOTO 10
140 END
```

Version Olivetti M20

```
10 CLS
20 XM=511:YM=255
30 PRINT "coordonnées initiales: ";
40 INPUT X1, Y1
50 PRINT "coordonnées finales  : ";
60 INPUT X2, Y2
70 IF X1>XM OR X2>XM OR Y1>YM OR Y2>YM THEN GOTO 10
80 REM ****** dessin ******
90 REM ce dessin est exécuté sur M20
100 LINE (X1, Y1) - (X2, Y2)
110 PRINT "Poursuivre (O/N): ";
120 INPUT A$
130 IF A$="O" THEN GOTO 10
140 END
```

La différence principale entre les deux programmes (voir listings p. 1392) provient de l'instruction qui trace le segment sur l'écran, (ligne 100). De plus, dans la première machine, il faut mettre en route le mode graphique avec l'instruction HGR (ligne 90) dont nous expliquerons la signification plus loin.

La forme complète de l'instruction LINE de l'Olivetti M20 est la suivante :

LINE % N STEP (X1, Y1) — STEP (X2, Y2), C, B, F, option

où :

LINE : Instruction générale pour tracer un segment entre les points de coordonnées (X1, Y1) et (X2, Y2)

%N : Valeur qui indique une des fenêtres (division de l'écran). Si on précise ce paramètre, l'instruction LINE n'opère que sur la fenêtre correspondante.

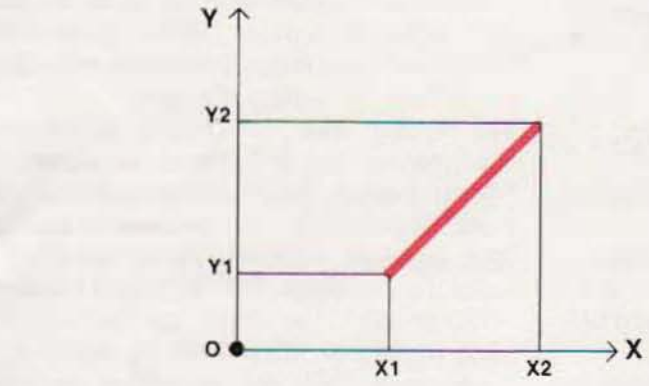
STEP : Mot clé pour indiquer l'usage de coordonnées relatives. En utilisant cette option, les coordonnées X1, Y1 deviennent relatives au dernier positionnement effectué. X2, Y2 sont les nouvelles valeurs initiales. Le schéma page suivante indique le fonctionnement de l'option STEP.

C : Couleur du segment.

B : Paramètre optionnel grâce auquel on trace un rectangle à la place du segment ayant pour extrémités les points X1, Y1, et X2, Y2 ; le rectangle a ce segment pour diagonale.

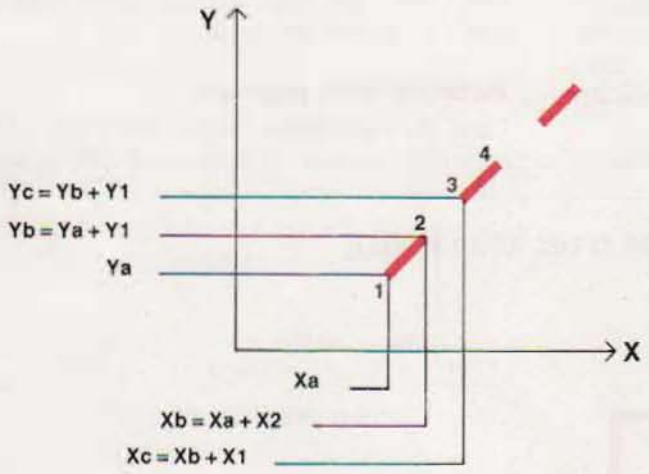
F : Paramètre qui permet de remplir, avec la couleur précisée en C, la surface du rectangle tracé avec B.

EFFET DE L'OPTION STEP



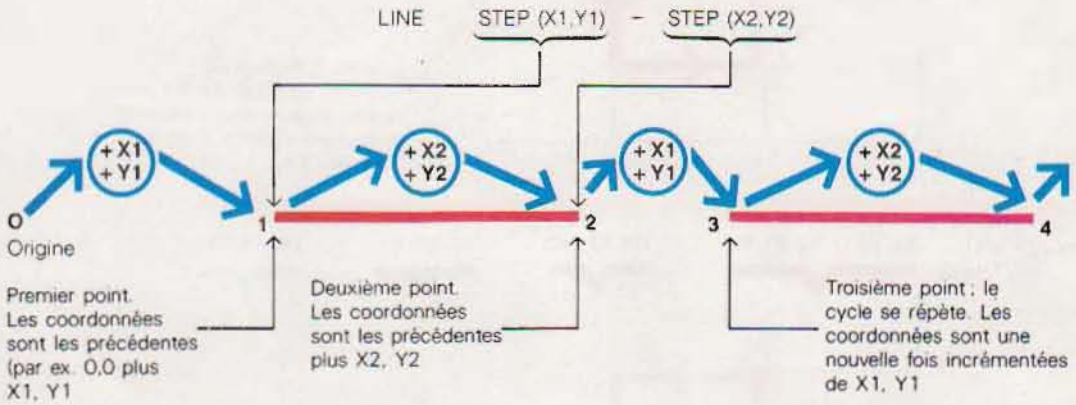
LINE (X1,Y1) – (X2,Y2)

Tracer le segment compris entre les points de coordonnées X1, Y1 et X2, Y2



LINE STEP (X1,Y1) – STEP(X2,Y2)

Les coordonnées des points sont obtenues en considérant X1, Y1 et X2, Y2 comme des valeurs relatives. Par exemple, une fois que le segment 1-2 a été tracé, la valeur X1, Y1 est ajoutée aux coordonnées finales et le segment 3-4 est tracé.



F : Paramètre permettant de remplir, avec la couleur indiquée par C, la surface du rectangle tracé d'après le paramètre B.

L'option prend une des valeurs AND, XOR, OR, NOT, PSET et PRESET et n'est utilisable qu'avec la couleur.

L'option B libère le programmeur de l'obligation d'écrire le sous-programme de tracé de rectangles comme une suite d'instructions de tracé de segments. Avec d'autres machines, on obtient le même résultat en utilisant la possibilité d'adressage multiple de l'instruction HPLOT. La formule de base :

HPLOT X1, Y1, TO X2, Y2

peut être développée pour obtenir des positionnements successifs en répétant l'option TO pour chacun des nouveaux couples de coordonnées.

Par exemple la formule :

HPLOT X1, Y1, TO X2, Y2 TO X3, Y3 TO X4, Y4

s'utilise pour le tracé d'un rectangle ayant pour sommets les points de coordonnées X1, Y1, etc. Son effet est donc semblable à celui de l'instruction précédente avec l'option B. La figure ci-dessous compare ces deux types d'instruction.

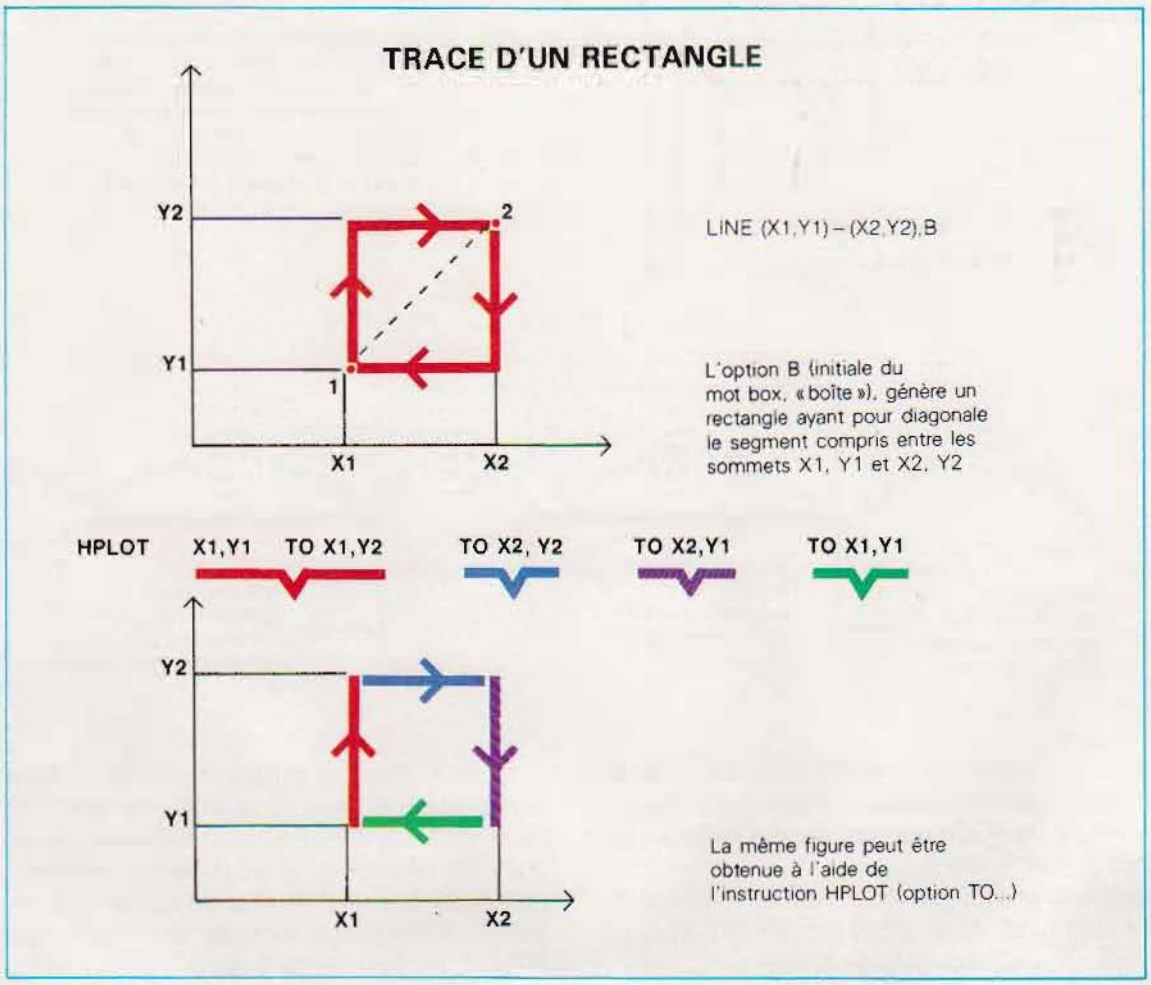
Avec l'instruction LINE, associée à l'option B, les côtés du rectangle obtenu sont toujours parallèles au bords de l'écran (ou de la fenêtre). En revanche, HPLOT n'est pas limité : toute figure générique peut être tracée, quel que soit le nombre de segments.

Les instructions de tracé des segments autorisent le dessin de toutes formes de polygones. L'organigramme ci-contre illustre un programme de dessin de carrés et de rectangles positionnés n'importe où sur l'écran ; il prévoit même la modification des dimensions de la figure en donnant un facteur d'échelle.

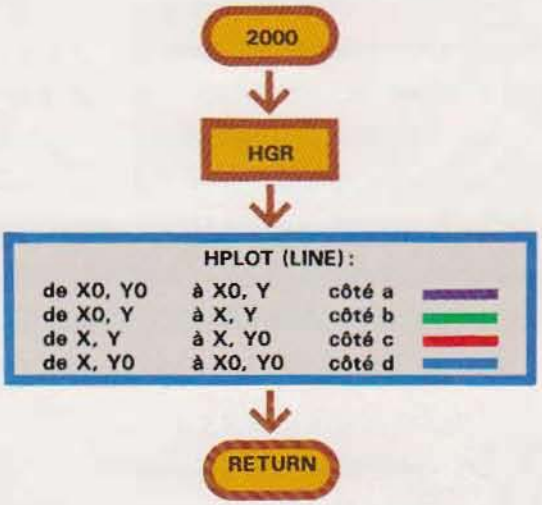
Les paramétrages sont obtenus en affectant, aux coordonnées de la figure, les valeurs X1, Y0 du sommet de positionnement et en calculant les déplacements nécessaires pour obtenir les côtés. Cette logique est appliquée au tracé d'un rectangle dans les figures des pages 1420 et 1421. Les listings pp. 1421 et 1422 sont donnés en deux versions Siprel 2010 (ordinateurs Apple et compatibles) et Olivetti M20 pour bien faire ressortir les différences entre les jeux d'instructions.

Rotation d'un segment

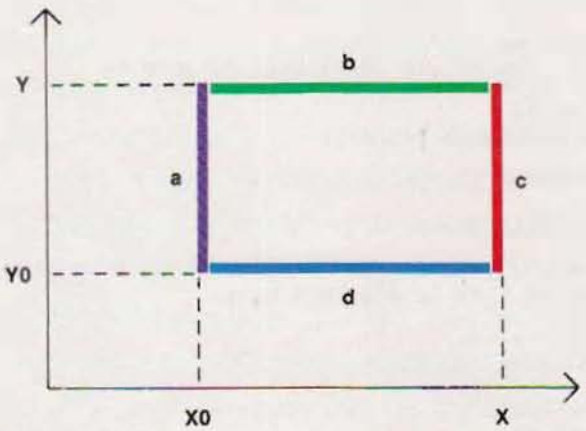
Dans les applications graphiques, il faut généralement disposer de sous-programmes assu-



TRACE DE LA FIGURE



Les coordonnées X, Y sont calculées dans le sous-programme 1000.
Les côtés de la figure étant parallèles aux axes, aucun autre point n'est nécessaire.



TRACE PARAMETRE DE CARRES ET DE RECTANGLES

Version Siprel 2010, Apple et compatibles

```
  5  MX=279 : MY=191
 10   TEXT : REM sélection de la page de texte
 15   HOME : REM l'écran est vidé
 20   PRINT  "Facteur de forme";
 30   INPUT  N
 40   PRINT  "coordonnées de référence";
 50   INPUT  X0,Y0
 60   PRINT  "Facteur d'échelle";
 70   INPUT  S
 80   GOSUB  1000
 90   IF KE=0 THEN GOTO 100
 95   PRINT "Erreur": FOR P=1 TO 2000: NEXT P: GOTO 10
100   GOSUB 2000
105   VTAB 22: REM Le curseur est positionné à la 22ème ligne
110   PRINT "Voulez-vous poursuivre ?(O/N)";
```

```
120  INPUT A$
130  IF A$="0" THEN GOTO 10
140  END
1000  REM ***** controles *****
1010  KE=0
1020  X=X0+N*S : Y=Y0+S
1030  IF Y>MY OR Y<0 OR X>MX OR X<0 THEN KE=1
1040  RETURN
2000  REM ***** Dessin *****
2010  HGR : HCOLOR= 3
2020  HPLOT X0,Y0 TO X0,Y
2030  HPLOT X0,Y TO X,Y
2040  HPLOT X,Y TO X,Y0
2050  HPLOT X,Y0 TO X0,Y0
2060  REM L'instruction suivante aurait suffi :
2070  REM HPLOT X0,Y0 TO X0,Y TO X,Y TO X,Y0 TO X0,Y0
2080  RETURN
```

Version Olivetti M20

```
5  MX=511 : MY=255
10  REM Non obligatoire
15  CLS : REM L'écran est vidé
20  PRINT  "Facteur de forme"
30  INPUT  N
40  PRINT  "Coordonnées de référence"
50  INPUT  X0,Y0
60  PRINT  "Facteur d'échelle"
70  INPUT  S
80  GOSUB  1000
90  IF KE=0 THEN GOTO 100
95  PRINT  "Erreur" : FOR P=1 TO 2000: NEXT P: GOTO 10
100  GOSUB 2000
105  REM Non obligatoire
110  PRINT "Tu veux continuer ?(O/N)";
120  INPUT A$
130  IF A$="0" THEN GOTO 10
140  END
1000  REM ***** controles *****
1010  KE=0
1020  X=X0+N*S : Y=Y0+S
1030  IF Y>MY OR Y<0 OR X>MX OR X<0 THEN KE=1
1040  RETURN
2000  REM ***** Dessin *****
2010  REM Non obligatoire
2020  LINE (X0,Y0) - (X0,Y)
2030  LINE (X0,Y) - (X,Y)
2040  LINE (X,Y) - (X,Y0)
2050  LINE (X,Y0) - (X0,Y0)
2060  REM L'instruction suivante aurait suffi:
2070  REM LINE (X0,Y0) - (X,Y),1,B
2080  RETURN
```

méthode utilisée dans les organigrammes précédents pour le changement d'échelle.Prenons l'exemple de la figure de la page 1420. Si les coordonnées X0, Y0 sont modifiées, mais non les paramètres N et S, la figure sera alors tracée à partir du point défini par le nouveau couple de coordonnées X0, Y0 (sommet 1)...
De cette façon, la figure est déplacée dans une autre zone de l'écran, mais il faut noter le tracé précédent n'est pas effacé.

Pour l'éliminer, il faut vider tout l'écran ou repasser sur cette première figure avec une « gomme ». La première solution n'est pas toujours utilisable, car d'autres figures présentes — et devant rester affichées — pourraient être affectées. La seconde solution, qui consiste à n'effacer que la figure déplacée, est bien souvent la seule acceptable.
En mode graphique, on utilise généralement plusieurs couleurs, y compris la couleur du

fond (background), indiquée par le code numérique 0. On a recours à cette couleur comme à une gomme, en la faisant repasser sur les lignes d'une figure. Le code 0, indiquant la couleur de fond noire (le dessin est tracé en blanc sur noir à l'écran), est quasiment valable pour toutes les machines, avec toutefois quelques exceptions.
On sélectionne une couleur avec :

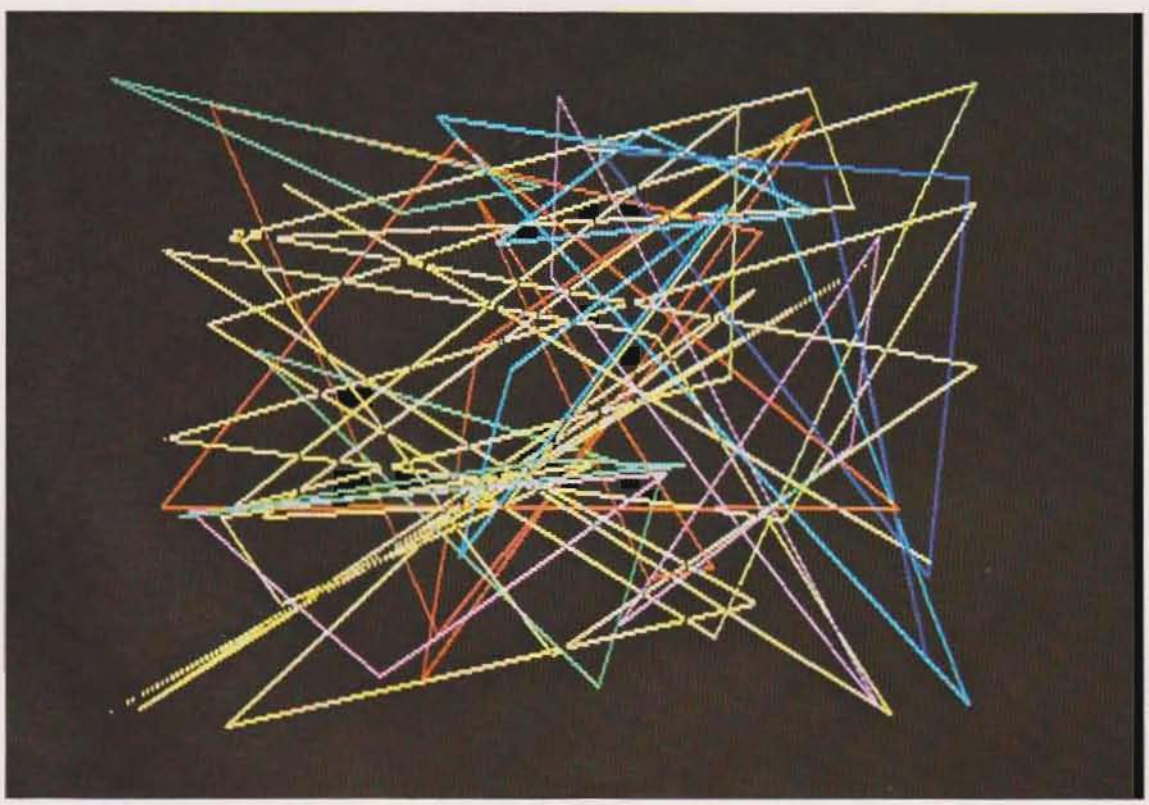
COLOR = n

n étant son code. Dans certains cas, le symbole = doit être omis.
Pour réactiver l'affichage après avoir utilisé la couleur du fond, il suffit d'indiquer la nouvelle couleur à l'aide de l'instruction COLOR.
Avec le Siprel 2010, par exemple, seul le code 7 est utilisable, l'écran étant monochrome ; ce micro-ordinateur n'offre donc que deux possibilités : COLOR = 0 (non visible) et COLOR = 7 (visible)
Page 1424, on trouvera l'organigramme d'un programme illustrant la technique de déplacement d'un rectangle.
Le listing correspondant n'est pas indiqué ; en effet il est pratiquement identique à celui du programme de tracé paramétré de carrés et de rectangles.
Contrairement à la translation, la rotation présente quelques difficultés : il faut recourir à la trigonométrie pour obtenir les paramètres (coordonnées).
L'organigramme de la page 1425 sert à effectuer la rotation d'un segment.
Le bloc 1000 regroupe les instructions d'entrée des données initiales :

— longueur du segment
— angle initial par rapport à l'axe X
— position initiale (X0, Y0) à du centre de rotation.

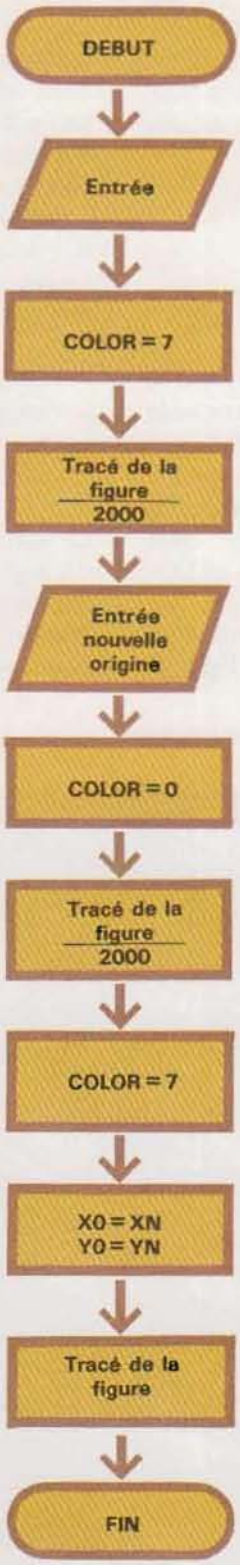
Le bloc 1500 peut ne pas provoquer l'affichage du segment dans sa position initiale et sauter directement au bloc 4000.
A la fin de la présentation, si l'opérateur désire poursuivre, le programme revient aux instruc-

Tracé de segments sur un écran vidéo couleur.



Centro THC/Marka

ORGANIGRAMME DU PROGRAMME DE DEPLACEMENT D'UNE FIGURE



Entrée :
des coordonnées initiales (XO, YO)
du facteur de forme N
du facteur d'échelle S

Sélection de la couleur

Le sous-programme 2000 est identique au sous-programme de tracé paramétré de carrés et de rectangles

Les coordonnées du nouveau point d'origine (après le déplacement) sont affectées à deux variables purement pratiques XN et YN

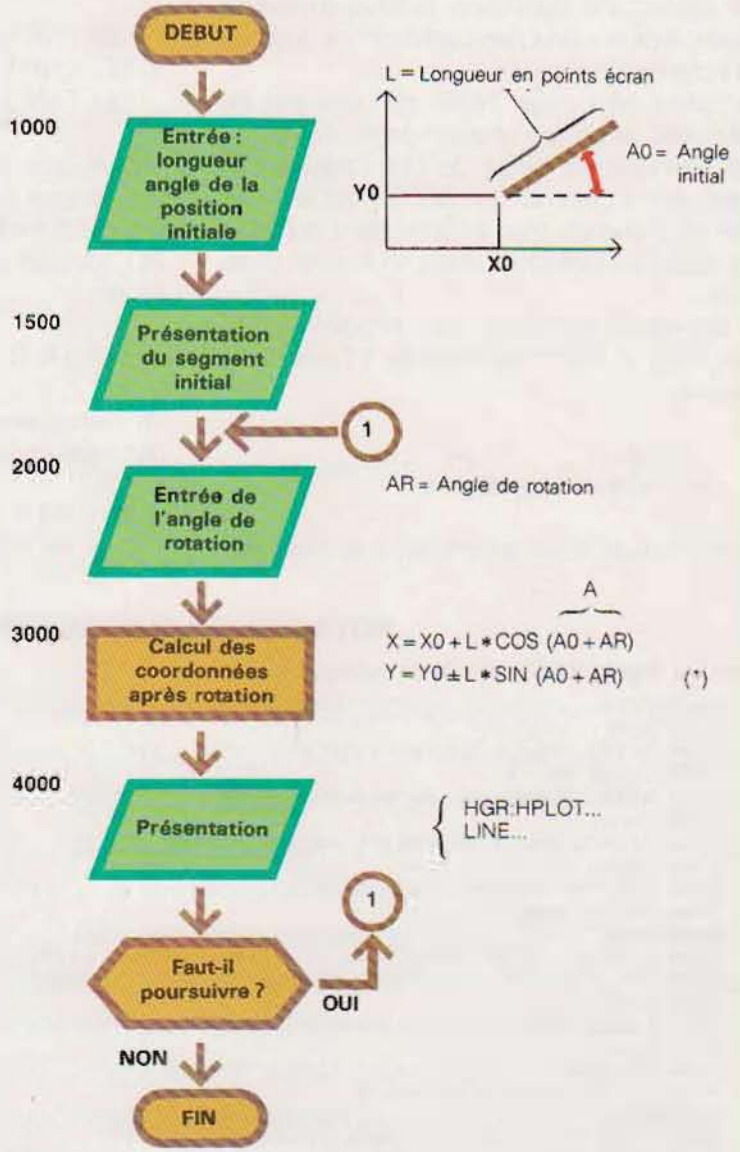
Sélection de la couleur du fond (effacement)

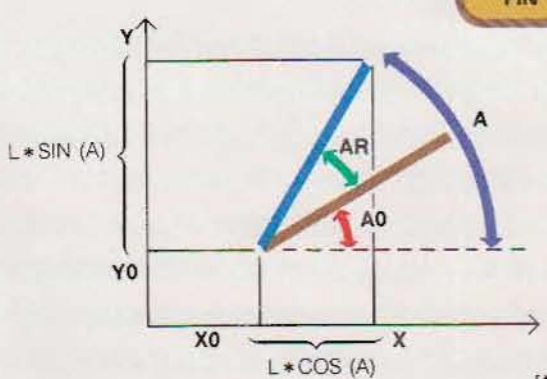
L'effacement s'obtient en repassant sur la même figure avec COLOR = 0

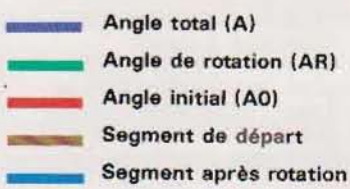
Réactivation de l'affichage

Nouvelle position initiale

ORGANIGRAMME DE ROTATION D'UN SEGMENT







[*) Le signe (+ ou –) dépend de l'orientation de l'axe y. C'est donc un paramètre caractéristique de la machine.

tions du bloc 2000 (point 1) et le cycle d'entrée de l'angle de rotation, de calcul avec de nouvelles coordonnées et d'affichage est réactivé.
Par rapport à la translation, la seule différence réside dans le calcul des coordonnées du point de rotation.
Ce calcul (voir page 1425) ne pose pas de difficultés de programmation particulières.
Les formules de calcul de ces coordonnées (indiquées à côté du bloc 3000) sont applicables telles quelles, mais il faut veiller à exprimer les angles en radians, comme en Fortran et en Basic.
La conversion en radians (voir programmes pages 1426 et 1427) est obtenue à l'aide de la formule :

$$\text{Angle en radians} = \text{Angle en degrés} * 3.14/180$$

Cette formule n'est qu'indicative et n'est pas optimale du point de vue de la programmation. Elle peut être rendue plus « professionnelle » avec les fonctions :

```
DEF FNR (AG) = AG* 3.14/180
DEF FNX (L,A) = X0 + L*COS (A)
DEF FNY (L,A) = Y0 + L*SIN (A)
```

(AG = angle en degrés, A : angle en radians, L = longueur du segment).
L'angle A (radians) s'obtient à partir de l'angle AG (degrés) au moyen de la formule toute simple :

```
A = FN R (AG)
```

De même, les nouvelles coordonnées résultent des instructions :

```
X = FN X (L,A)
Y = FN Y (L,A)
```

ROTATION D'UN SEGMENT

Version Siprel 2010, Apple et compatibles

```
1000 TEXT
1100 HOME
1200 PRINT "Position initiale";
1300 INPUT X0,Y0
1400 PRINT "Longueur du segment";
1500 INPUT L
1600 PRINT "Angle de départ (en degrés)";
1700 INPUT A0
1800 HGR: HCOLOR=7
1900 GOSUB 3000
2000 HOME
2100 VTAB22: PRINT "Angle de rotation (en degrés)";
2200 INPUT A1
2300 HCOLOR=0
2400 GOSUB 4000: REM Effacement
2500 AR=A1
2600 HCOLOR=7
2700 GOSUB 3000: REM Dessin
2750 HOME
2800 VTAB 22: PRINT "Tu veux continuer (O/N)";
2850 INPUT A$
2900 IF A$="O" THEN GOTO 2000
2950 END
3000 REM Calcul
3100 A0=(A0*3.14)/180
3200 AR=(AR*3.14)/180
3300 X=X0+L*COS(A0+AR)
3400 Y=Y0-L*SIN(A0+AR)
4000 REM Tracé du segment
4100 HPLOT X0,Y0 TO X,Y
4200 RETURN
```

Version Olivetti M20

```
1000 CLEAR
1100 REM Non obligatoire
1200 PRINT "Position initiale";
1300 INPUT X0,Y0
1400 PRINT "Longueur du segment";
1500 INPUT L
1600 PRINT "Angle de départ (en degrés)";
1700 INPUT A0
1800 CLS: COLOR 1
1900 GOSUB 3000
2000 CURSOR (1,15),1
2100 PRINT "Angle de rotation (en degrés)";
2200 INPUT A1
2210 CURSOR (1,15),1
2220 PRINT "                                        ": REM 40 espaces
2300 COLOR 0
2400 GOSUB 4000: REM Effacement
2500 AR=A1
2600 COLOR 1
2700 GOSUB 3000: REM Dessin
2750 REM Non obligatoire
2770 CURSOR (1,15),1
2800 PRINT "Tu veux continuer ?(O/N)";
2850 INPUT A$
2900 IF A$="0" THEN GOTO 2000
2950 END
3000 REM Calcul
3100 A0=(A0*3.14)/180
3200 AR=(AR*3.14)/180
3300 X=X0+L*COS(A0+AR)
3400 Y=Y0+L*SIN(A0+AR)
4000 REM Tracé du segment
4100 LINE (X0,Y0) - (X,Y)
4200 RETURN
```

Tracé d'une circonférence

Certaines machines disposent d'instructions pour le dessin de figures complexes sans passer par des sous-programmes. Nous avons déjà vu un exemple avec l'instruction LINE complétée de l'option B qui, avec l'Olivetti M20, permet d'obtenir le tracé d'un rectangle. Un cas plus complexe est celui de l'instruction de tracé d'une circonférence. Sa syntaxe et les options prévues dépendent du type de machine et de la version du Basic utilisée.
Dans le système M20, l'instruction complète est :

CIRCLE %N, (X,Y), R, C, A, V,

avec :

%N : Numéro de la fenêtre (s'il n'est pas indiqué, le système prend, par défaut, la dernière fenêtre sélectionnée)
X,Y : Coordonnées du centre
R : Rayon
C : Couleur

Le paramètre A est une valeur numérique exprimant le facteur de forme. L'emploi de l'option A permet de compenser les différentes densités des points d'écran lors du passage de l'axe X à l'axe Y. Dans certains cas, elle aide au tracé des figures circulaires allongées du type ellipse.
Dans la machine considérée, la valeur de A pour obtenir un cercle est 0,807. En l'augmentant ou en la diminuant, on obtient une déformation sur l'un ou l'autre des axes. Ce paramètre est optionnel et sa valeur par défaut est la valeur standard.
La valeur V est un autre paramètre optionnel utilisé avec la couleur ; il a la même signification que dans l'instruction LINE.
Tous les paramètres cités étant optionnels — à l'exception des coordonnées du centre et du rayon — la forme la plus simple de cette instruction est :

CIRCLE (X,Y), R

Certaines machines ont recours à deux autres paramètres optionnels pour ne tracer qu'un arc de cercle, en indiquant son angle d'ouverture.

Dans certains cas, il faut spécifier l'angle du début et l'angle de fin de l'arc (en radians). Par exemple, l'instruction :

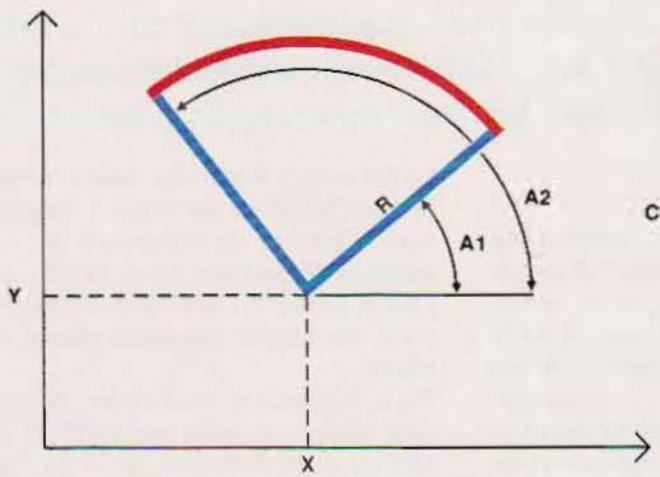
CIRCLE (X,Y), R,A1,A2

trace un arc de cercle (ayant son centre au point X,Y et de rayon R) compris entre un rayon incliné de l'angle A2 et un angle incliné de A2 (exemple ci-dessous).
Quand la machine ne reconnaît pas l'instruction CIRCLE, le système appelle un sous-programme, écrit par l'utilisateur à chaque visualisation d'un cercle.
Abstraction faite du facteur de forme, on a besoin de trois paramètres pour tracer un cercle : les coordonnées du centre (XO,YO) et le rayon (R). Le paramètrage s'obtient en calculant les coordonnées de chaque point de la circonférence en fonction des valeurs de XO, YO et R.
La figure 1 (page 1429) indique la méthode de calcul d'un point quelconque P de la circonférence, dont les coordonnées (A étant l'angle) sont données par les formules :

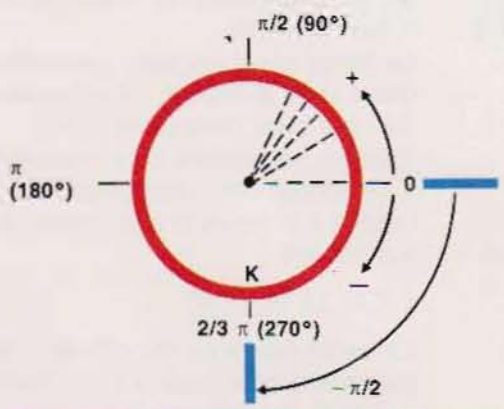
X = XO + R * COS (A)
X = YO + R * SIN (A)

Une fois données les valeurs des coordonnées du centre (XO, YO) et du rayon, on obtient, grâce aux formules précédentes et en variant l'angle A de 0 à 2 π (360 degrés), les coordonnées de tous les points du cercle.
En partant du point P1, l'angle aura pour valeur : A = 0.

INSTRUCTION CIRCLE AVEC INDICATION DES ANGLES

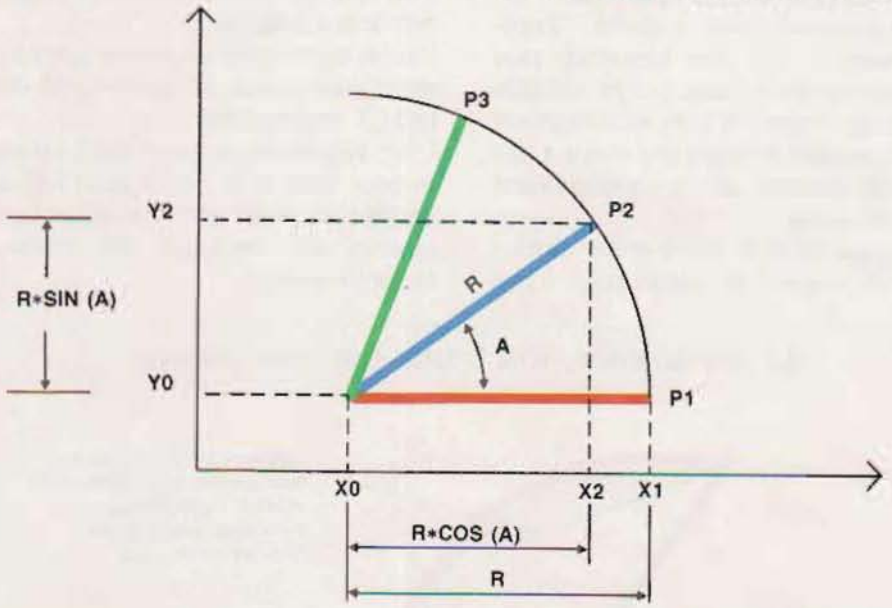


CIRCLE (X,Y), R, A1, A2

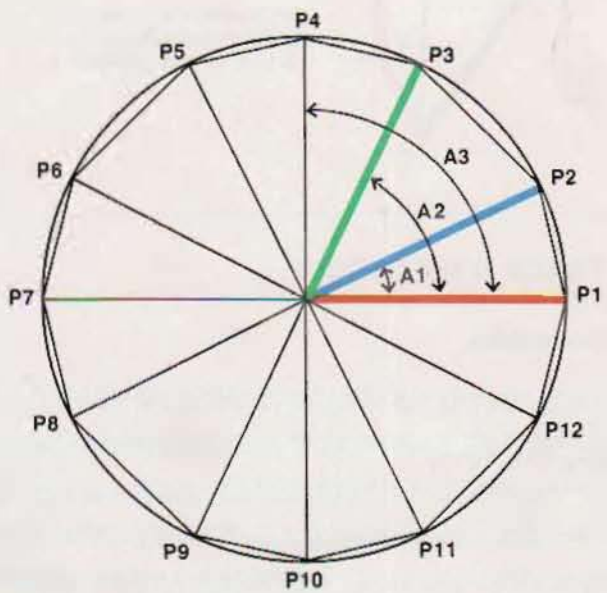


Les angles (A1 et A2) sont exprimés en radians et rapportés à la circonférence goniométrique. L'angle 0 est à l'extrême droite et les valeurs augmentent en tournant ver la gauche.
Avec certaines machines, il est possible d'utiliser des angles négatifs.
Le point K, par exemple, peut être indiqué par + 2/3 π (rotation dans le sens inverse des aiguilles d'une montre, sens positif) ou par π /2 (rotation dans le sens des aiguilles d'une montre, sens négatif)

1/ CALCUL DES COORDONNEES D'UN POINT SUR UN CERCLE



2/ TRACE D'UN CERCLE



A1 = Angle de départ

A2 = A1 + incrément

A3 = A2 + incrément

.....

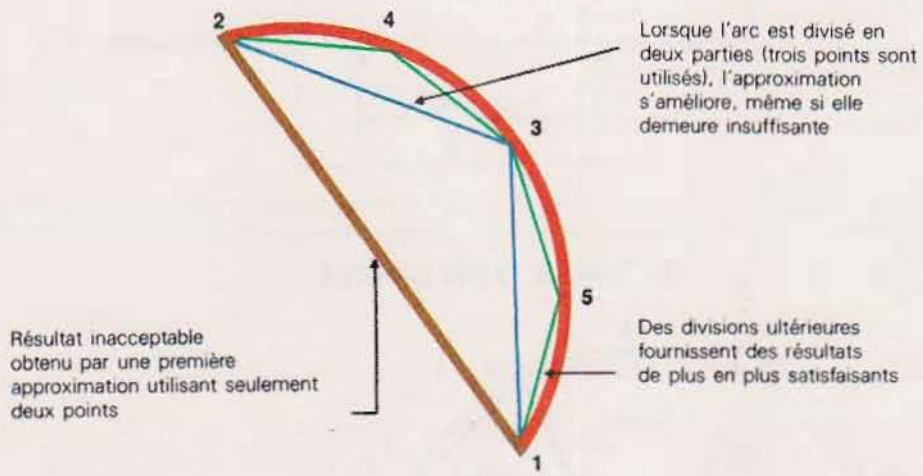
Les coordonnées [COS (0) = 1, SIN (0) = 0] auront pour valeur :

X = X0 + R
Y = Y0

On peut ensuite augmenter l'angle A d'un nombre fixe (increment), calculer les coordonnées à l'aide des formules indiquées, et ainsi de suite. En fin de calcul (figure 2, ci-dessus) le résultat sera un ensemble de points appartenant au cercle de rayon R et de centre X,Y. En les unissant deux à deux (segments), on peut obtenir une bonne approximation de la circonférence à l'aide du polygone inscrit.

La trop grande distance entre les points tracés sur la figure fait ressortir la disparité par rapport à un vrai cercle. Dans la réalité, l'increment de l'angle A doit être beaucoup plus petit ; la distance entre deux points calculés l'un après l'autre diminuera alors et le segment de droite les unissant donnera une meilleur approximation de l'arc de cercle correspondant (figure 3, ci-dessous).
Les listings (page 1430 et 1431) présentent un programme de tracé d'un cercle pour Siprel 2010 (Apple et compatible) et Olivetti M20, bien que la seconde machine dispose d'une instruction spéciale.
Il suffit de modifier deux instructions pour passer d'une version à l'autre (GHR disparaît et HPLOT devient LINE...).
L'organigramme en page 1432 est une variante pour tracé d'un cercle plein (on élimine les égalités X1 = X2 et Y1 = Y2 et le cercle est obtenu en dessinant les rayons de la circonférence).

3/ APPROXIMATION D'UNE CIRCONFERENCE



TRACE D'UN CERCLE

Version Siprel 2010, Apple et compatibles

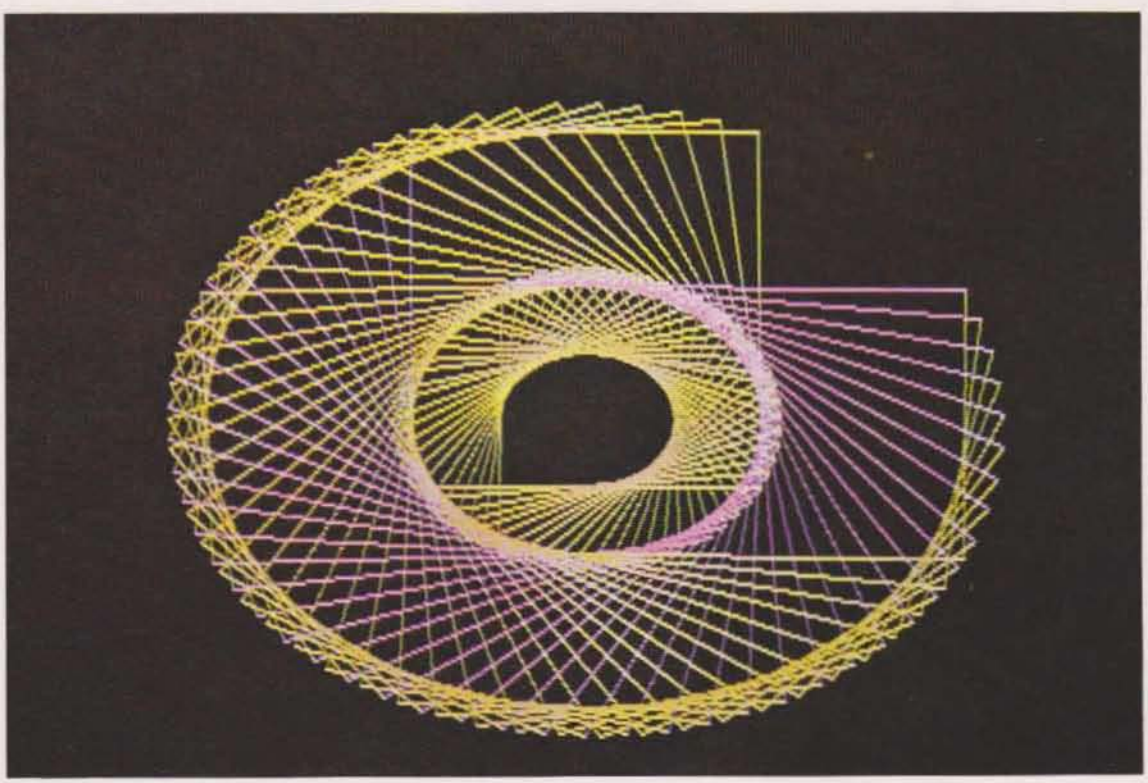
```
10 REM    Cercle
20 TEXT
30 HOME
40 PRINT "Coordonnées du centre (X0,Y0)";
50 INPUT X0,Y0
60 PRINT "Longueur du rayon";
70 INPUT R
80 PRINT "Pas d'incrément";
90 INPUT D
100 HGR: COLOR=7
110 GOSUB 1000
120 VTAB 22
130 PRINT "Continue ?(O/N)"
140 INPUT C$
150 IF C$="0" THEN GOTO 10
160 END
1000 REM Dessin
1010 X1=X0+R
1020 Y1=Y0
1030 FOR A=0 TO 6.28 STEP D
1040 X2=X0+R*COS(A)
```

```
1050 Y2=Y0+R*SIN(A)
1060 HPLOT X1,Y1 TO X2,Y2
1070 X1=X2: Y1=Y2
1080 NEXT A
1090 RETURN
```

Version Olivetti M20

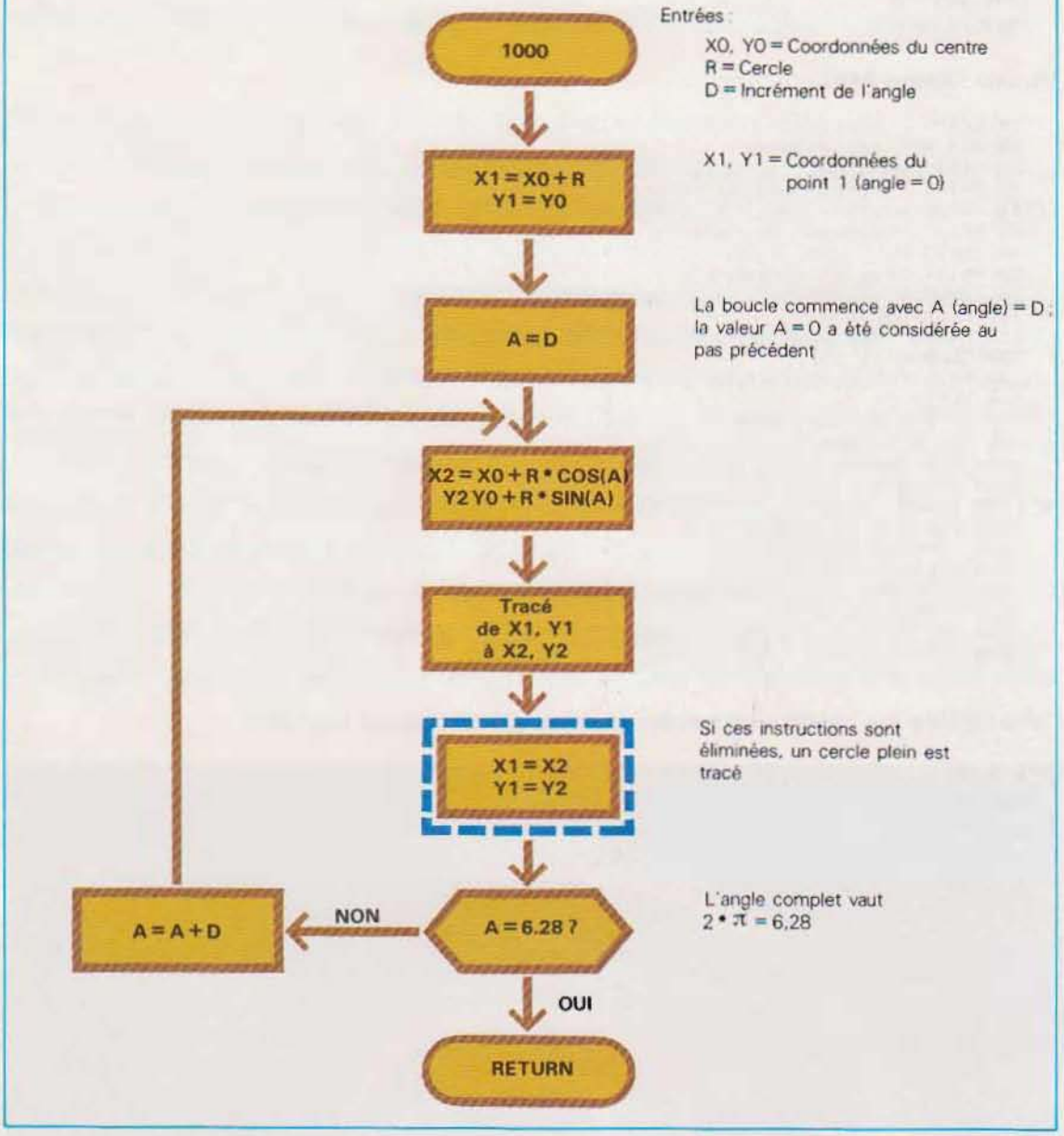
```
 10 REM      Cercle
 20 REM Non obligatoire
 30 CLS: COLOR 1
 40 PRINT "Coordonnées du centre (X0,Y0)";
 50 INPUT X0,Y0
 60 PRINT "Longueur du rayon";
 70 INPUT R
 80 PRINT "Pas d'incrément";
 90 INPUT D
100 CLS: COLOR 3
110 GOSUB 1000
120 CURSOR (1,15),1
130 PRINT "Continue (O/N)";
140 INPUT C$
150 IF C$="0" THEN GOTO 10
160 COLOR 1: END
1000 REM Dessin
1010 X1=X0+R
1020 Y1=Y0
1030 FOR A=0 TO 6.28 STEP D
1040 X2=X0+R*COS(A)
1050 Y2=Y0+R*SIN(A)
1060 LINE (X1,Y1) - (X2,Y2)
1070 XI=X2: Y1=Y2
1080 NEXT A
1090 RETURN
```

Cette figure a été obtenue en traçant un rectangle en position tournante.



Centro THC/Marka

ORGANIGRAMME DU SOUS-PROGRAMME CERCLE



Visualisation du graphique d'une fonction

Une fonction mathématique monodrome, ou uniforme, est une loi analytique qui fait correspondre, à chaque valeur de la variable indépendante X, une valeur de la variable dépendante Y.

Quand il existe plus d'une variable indépendante, la définition reste la même, mais pour plusieurs variables notées X1, X2, etc... Inversement, on parlera de fonction à plusieurs valeurs (ou polydrome) quand la valeur de la variable dépendante n'est pas unique pour des valeurs de variables indépendantes.

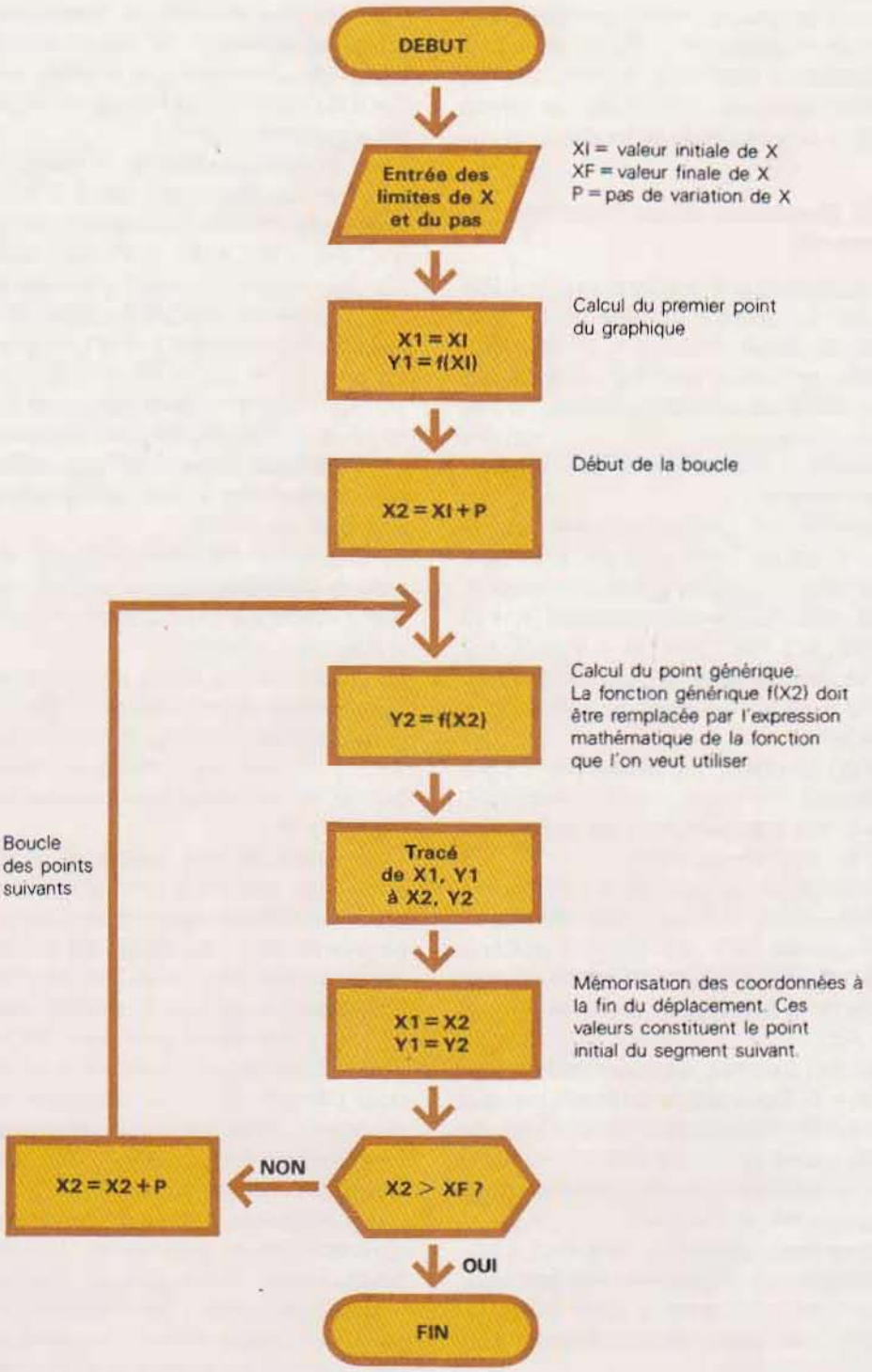
La représentation la plus courante d'une fonction est : Y = F (X) qui signifie que Y est fonction de la variable indépendante X. Etant donnée une fonction monodrome, la façon la plus simple de connaître les valeurs de la

variable dépendante consiste à construire un tableau avec les valeurs arbitraires de X (variable indépendante) et les valeurs correspondantes de Y, calculées à l'aide des calculs sur la fonction.
Toutefois, ce tableau ne fournit pas une vision immédiate de la relation X/Y. Aussi recourt-on à la représentation graphique, en reportant les valeurs du tableau sur une feuille. Le dessin obtenu est le graphique de la fonction.

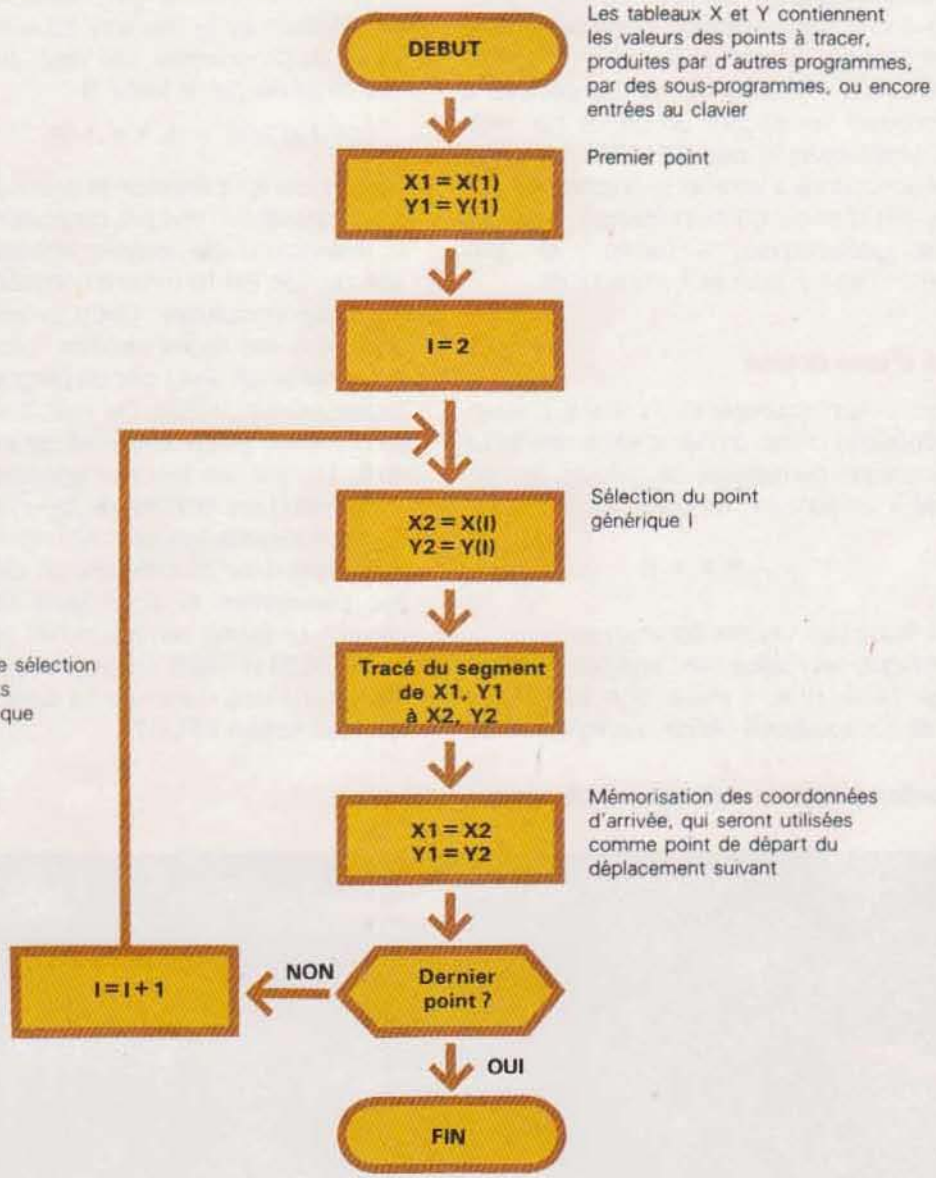
Tracé du graphique d'une fonction par segments

Le tracé du graphique d'une fonction peut être effectué par l'ordinateur à l'aide de la seule instruction graphique de tracé d'un segment.
La méthode consiste à produire un ensemble de valeurs de la variable indépendante, à calculer pour chacune d'elles celle de la variable dépendante et à unir les points ainsi obtenus (voir page suivante).
Le premier bloc est l'entrée au clavier des valeurs initiale (XI) et finale (XF) de la variable indépendante ainsi que du pas de variation P.
Le premier point aura les coordonnées X = XI et Y = f(XI), le suivant X = XI + P et Y + f (XI + P), et ainsi de suite, en augmentant toujours X de P et en calculant la valeur de Y correspondante.
Le dessin du graphique est donné par la ligne brisée unissant les points ainsi déterminés, c'est-à-dire une approximation de la fonction Y ≐ F(X) à l'aide de segments.
La « qualité » de la représentation est liée à la proximité des points avec un pas P très réduit.
La valeur optimale de P est difficile à établir a priori : elle dépend en effet, du type de courbe, c'est-à-dire de la forme particulière de la fonction Y = F(X).
Soit la fonction suivante, décrivant une droite : Y = 3 * X + 5. Deux points suffisent, puisque par deux points il ne passe qu'une droite. En unissant les coordonnées des deux points de la droite par un segment, on est certain de représenter exactement la fonction.
Si la fonction avait une forme complexe, il faudrait la découper en petites portions dont chacune pourra être approchée à l'aide d'un segment, selon une logique proche de celle d'un tracé de circonférence.
Les coordonnées X, Y de chaque point d'une circonférence sont liées par une relation du type $X^2 + Y^2 = R^2$ laquelle peut être ramenée à $Y = R^2 - X^2$, qui est une forme Y = F(X).
Pour déterminer les valeurs de X et de Y, nous avons employé les formules trigonométriques, mais ces deux aspects sont parfaitement identiques et découlent de deux méthodes différentes de résolution d'un triangle rectangle (à l'aide du théorème de Pythagore ou des formules trigonométriques).
Dans l'organigramme de la page 1434, les coordonnées du premier point (X1, Y1) et celles du second (X2, Y2), obtenu en ajoutant P sur l'axe X (X2 = X1 + P), sont calculées.
Au pas suivant, le point d'arrivée précédent (X2, Y2) devient le point de départ (il faut donc modifier les valeurs X1 et Y1 à l'aide de l'instruction X1 = X2 et Y1 = Y2) ; la nouvelle position finale (nouvelles valeurs de X2, Y2) est toujours obtenue en ajoutant la quantité P à X1 et en calculant (à partir de l'expression algébrique représentant la fonction) la valeur correspondante de Y (Y2).
Ce processus doit être répété sur tout l'intervalle de tracé graphique, c'est-à-dire jusqu'à ce que X2 atteigne (ou dépasse) XF (valeur finale entrée par l'utilisateur).
Ce processus est activé à l'aide d'une boucle dans laquelle X varie du pas P de XI à XF.
Pour simplifier le listing, le premier point X1 = XI, Y1 = f(XI) est calculé en dehors de la boucle, qui a comme premier argument la valeur XI + P.
Cette méthode n'est valable qu'en théorie. En effet, pour obtenir un bon programme de tracés, il faut tenir compte des différents facteurs.
Le premier est la **typologie de la fonction**. A priori, il n'est pas dit qu'une fonction puisse être exprimée par une loi mathématique. Dans ce cas, il faut établir une table des valeurs de X et Y. Ces valeurs seront alors stockées dans deux tableaux (de dimension égale au nombre de points maximum) et le programme devra simplement les prélever et les utiliser (organigramme page 1435).
Cette méthode n'est pas fondamentalement différente de la précédente. Simplement, la boucle ne se répète plus en incrémentant X. En effet, les valeurs du tableau ne varient pas forcément régulièrement (il pourrait, par exemple, contenir les valeurs 2 et 5 mais non les valeurs 3 et 4).

GRAPHIQUE D'UNE FONCTION ; ORGANIGRAMME DE PRINCIPE



TRACE DE COURBES A PARTIR DES VALEURS D'UN TABLEAU



C'est l'indice I, désignant les éléments du tableau, qui augmente. De cette façon, si I = 2, les coordonnées du 2e élément sont extraites, alors que quand I = 3, ce sont les coordonnées du 3e élément qui le sont, et ainsi de suite.
Autre différence par rapport à l'organigramme précédent : l'absence du calcul Y2 = f(X2), remplacé par un simple transfert de variable, X2 = X(I), Y2 = Y(I), X(I) et Y(I) étant les éléments du tableau se trouvant à la position I.

Il faut également prendre en considération d'autres facteurs comme les ordres de grandeur et les erreurs d'interpolation.
Pour obtenir, dans tous les cas possibles, des valeurs représentables à l'écran (ou sur d'autres terminaux), il faut tracer non pas les valeurs réelles, mais des valeurs qui leur sont proportionnelles selon un facteur d'échelle.
Tous les points ne sont pas exploitables dans le tracé d'un graphique à partir d'une fonction.

On suppose généralement, qu'entre deux points, le tracé est rectiligne. S'ils sont situés à une certaine distance, cette interpolation peut aboutir à une représentation inexacte et sans rapport avec la forme réelle de la fonction. L'approximation résultante est comparable à celle obtenue en traçant un cercle par segments (mais dans le cas du cercle, la forme exacte est connue a priori et le graphique n'engendre pas d'erreur d'interprétation). Dans les courbes quelconques, la forme n'est pas connue et l'erreur peut être importante.

Tracé d'une droite

La fonction la plus simple est la droite. L'équation générique d'une droite, c'est-à-dire la loi mathématique permettant de calculer les valeurs de Y en fonction de celles de X, est :

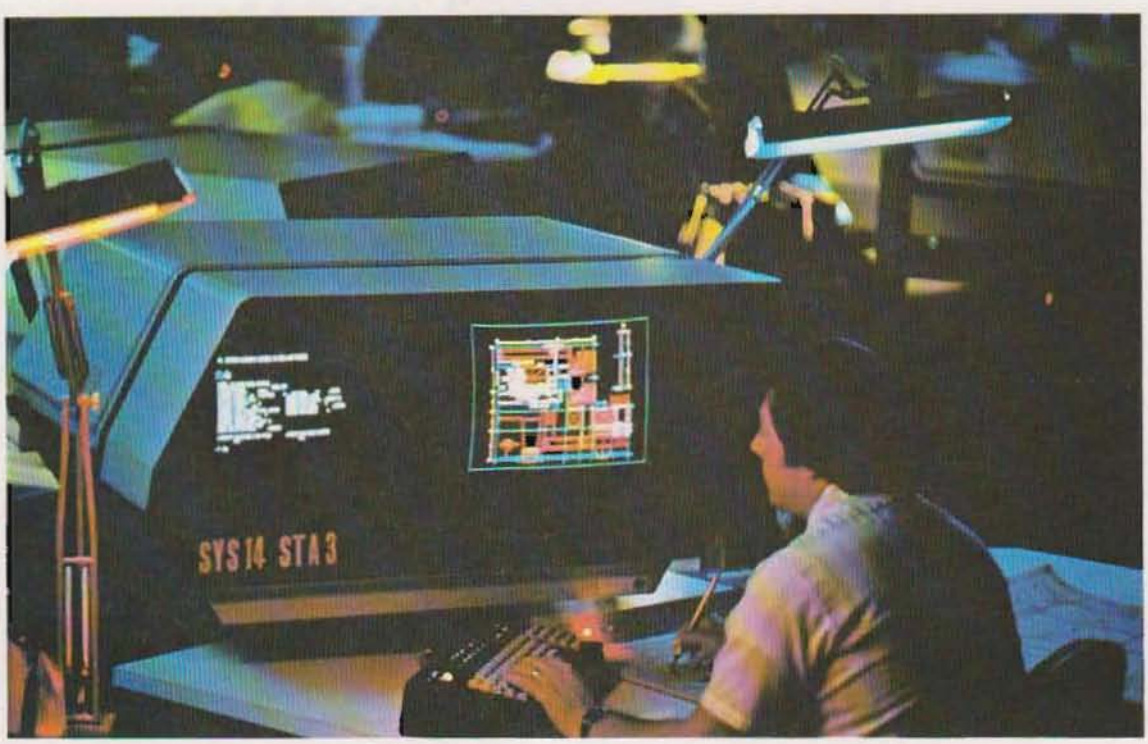
$$Y = A * X + B$$

A et B étant des valeurs constantes.
Le graphique est réalisé avec le programme de la page 1434, mais il existe une forme plus élégante qui consiste à définir une fonction utilisateur équivalente à la droite. Le calcul des valeurs de Y se fera alors simplement par appel de la fonction pour les différentes valeurs de X. La définition de la fonction est à introduire en début de programme. Elle peut, par exemple, être indiquée par la lettre R :

```
DEF FN R(X) = A * X + B
```

Dans le cas spécifique de la droite, la simplicité de la formule ne rend pas particulièrement utile la définition d'une fonction utilisateur. En revanche, elle est fortement conseillée pour des fonctions complexes. Cette procédure offre, alors, des avantages certains, car la fonction est écrite en un seul point du programme et est facilement modifiable. De plus, tous les sous-programmes graphiques sont paramétrés puisqu'ils traitent une fonction générique FN R(X) qui les rend exploitables de façon généralisée. L'organigramme (pages 1437 et 1438) trace une droite dans un intervalle de valeurs XI, XF (les paramètres A et B sont introduits au clavier). Le listing correspondant se trouve en pages 1438 et 1439. Le programme est portable sur d'autres machines : il suffit de remplacer l'instruction HPLOT.

Application professionnelle d'un ordinateur graphique.



J. Zaner/Marka-Int'l Stock Photo

```
      < 5 OR YB > 153 OR YB < 5
        THEN 180
170   HPLOT XA,YA TO XB,YB
180   X1 = X2
    : Y1 = Y2
190   NEXT
200   HOME
210   VTAB 23
220   INPUT "Je continue ? (O/N)"
230   IF S$ = "O" THEN RUN
240   END
997   REM  ---------------
998   REM  Tracé des axes
999   REM  ---------------
1000  HGR
    : HCOLOR= 3
1010  HPLOT O - 15,N - 5 TO E + 14,N
      - 5 TO E + 14,5 + 3 TO O - 15,5
      + 3 TO O - 15,N - 5
1020  HPLOT O,Y0 TO E,Y0
    : HPLOT X0,S TO X0,N
1030  N2 = N
    : FOR N1 = 2 TO 0 STEP - 1
    : HPLOT X0 - N1,N2 TO X0 + N1,N2
    : N2 = N2 - 1
    : NEXT N1
1040  N2 = E
    : FOR N1 = 2 TO 0 STEP - 1
    : HPLOT N2,Y0 - N1 TO N2,Y0 + N1
    : N2 = N2 + 1
    : NEXT N1
1050  HPLOT E + 6,Y0 - 2 TO E + 11,Y0
        + 3
    : HPLOT E + 11,Y0 - 2 TO E + 6,Y0
        + 3
1060  HPLOT X0 - 10,N - 2 TO X0 - 7,N
        + 1
    : HPLOT X0 - 4,N - 2 TO X0 - 10,N
        +3
1070  IF FX = 1 THEN 1140
1080  FOR G = X0 TO O STEP - FX
1090  HPLOT G,Y0 + 1 TO G,Y0 - 1
1100  NEXT G
1110  FOR G = X0 TO E STEP FX
1120  HPLOT G,Y0 + 1 TO G,Y0 - 1
1130  NEXT G
1140  IF FY = 1 THEN 1210
1150  FOR G = Y0 TO N STEP - FY
1160  HPLOT X0 + 1,G TO X0 -1,G
1170  NEXT G
1180  FOR G = Y0 TO S STEP FY
1190  HPLOT X0 + 1,G TO X0 - 1,G
1200  NEXT G
1210  RETURN
```

A la différence des précédents organigrammes, celui-ci contient deux paramètres d'échelle FX et FY et calcule le pas P.
Les facteurs d'échelle (FX, FY) sont nécessaires pour ramener les valeurs de X et Y aux dimensions de l'écran.
Prenons un intervalle sur l'axe de 500, par exemple, XI = 0, XF = 500), avec un écran de 265 points horizontaux, il faut alors utiliser un facteur d'échelle de 0,5.
La valeur 500 est traitée comme s'il s'agissait de 500 × 0,5 = 250 qui tient dans les 265 points disponibles. Le facteur d'échelle sur l'axe X se calcule à l'aide de l'expression :

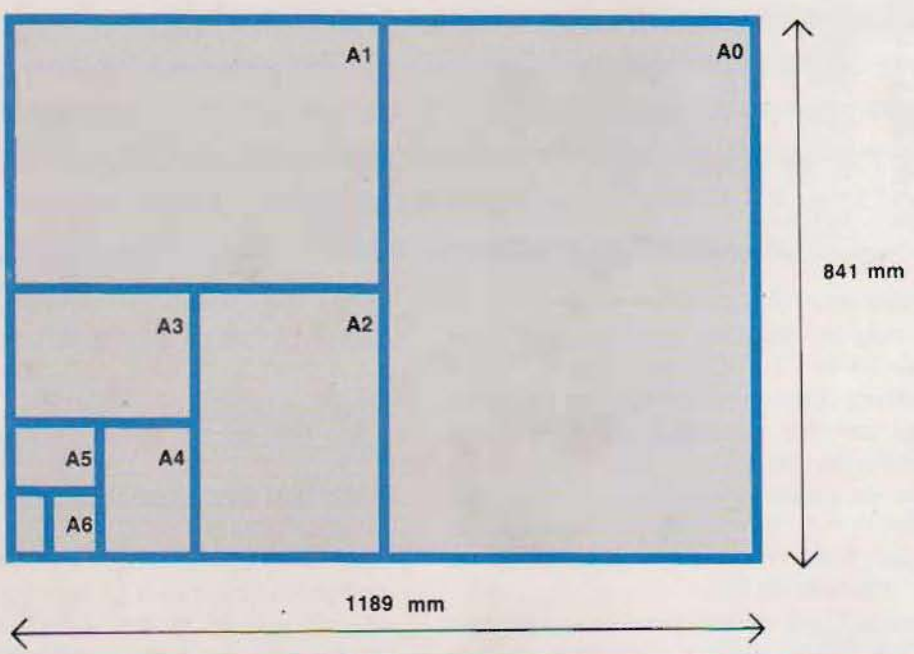
$$FX = \frac{XF - XI}{\text{Nombre des points d'écran disponibles}}$$

Remarquer que l'expression analogue est valable pour l'axe Y.
Il faut noter que la valeur du quotient est le plus souvent un nombre à plusieurs décimales, ce qui fait que le facteur d'échelle calculé ne permet pas une lecture rapide du graphique.

Il convient donc d'utiliser la fraction entière la plus proche : la réduction de la portion d'écran utilisée permettra une interprétation du graphique plus aisée.
Dans l'exemple précédent, le facteur d'échelle, calculé au moyen de l'expression donnée, serait 265/500 = 0,53 ; pour pouvoir interpréter le graphique, il faudrait multiplier les valeurs par 1,886 (1/0,53) à chaque lecture.
En arrondissant à 2, on obtient une réduction de l'image utile (les points utilisés ne sont plus 265, mais 250), qui a l'avantage de permettre d'effectuer mentalement les calculs d'échelle et donc d'avoir une lecture immédiate du graphique.
Le choix d'une échelle de représentation n'est pas, comme on pourrait le croire, arbitraire. Il existe des normes très précises réglementant les applications techniques.
Dans l'utilisation d'un traceur ou d'une tablette graphique, par exemple, il faut que le plan de travail soit suffisamment grand pour recevoir des feuilles au format commercial. Les formats standard définis par l'UNI sont désignés par un sigle composé de la lettre A suivie d'un chiffre (par ex. A0, A1, A2...). Le format le plus utilisé (dimensions standard d'une feuille de papier pour dactylographie) est le format A4. Les différents formats s'obtiennent en divisant successivement en deux le format de base A0, qui est défini par une surface de 1m^2.
Les dimensions correspondant aux différents formats sont indiquées dans le tableau ci-dessous.

FORMATS UNINORMALISES

Désignation	Dimensions en mm
A0	841 × 1 189
A1	594 × 841
A2	420 × 594
A3	297 × 420
A4	210 × 297
A5	148 × 210
A6	105 × 148



La partie initiale du programme (lignes 10 à 40) définit les paramètres de l'écran. Les valeurs N = 5, S = 153, E = 265, O = 15 délimitent la zone utile (en points écran), alors que les valeurs X0 = 140 et Y0 = 80 (Siprel 2010) présentent l'origine de référence des coordonnées. La nomenclature (N, S, E, O) renvoie aux points cardinaux et repère la « fenêtre » employée pour la représentation graphique. Les instructions 140, 150 et 160 calculent les valeurs XA, YA et XB, YB en points écran.
Comparativement à l'organigramme, les formules diffèrent de X0, Y0 : en effet, dans le cas général (organigramme), les coordonnées de chaque point de la courbe se rapportent à l'origine 0 de coordonnés X0 = 0 et Y0 = 0. Dans le listing, en revanche, on a fixé l'origine à 140, 80 pour obtenir une représentation centrée sur l'écran.
Notons que la coordonnée Y est la différence entre l'origine et la valeur résultant de l'équation de la droite. Cela est dû à l'orientation de l'axe Y qui commence en haut à gauche. Sur d'autres machines (le M20 d'Olivetti) on peut avoir besoin du signe +.
Le pas de visualisation est calculé à la ligne 110 alors que la boucle s'effectue entre la ligne 120 et 190. La ligne 160 contrôle que les valeurs des coordonnées (en points écran) ne dépassent pas les valeurs maximales (ligne 20). Pour abréger, en fin de boucle (ligne 190) on a omis l'indice (NEXT au lieu de NEXT 1).
De même, pour relancer le programme, on a adopté une syntaxe particulière qui n'est pas autorisée sur toutes les machines : il est alors nécessaire de compléter la ligne 190 et d'écrire la 230 sous la forme :

```
IF S$ = "S" THEN GOTO 50
```

Pour « porter » le programme sur d'autres machines, il faut en outre remplacer les instructions suivantes :

HOME : Effacement de l'écran et positionnement du curseur en haut à gauche (instruction 200).
VTABn : Tabulation verticale avec arrêt sur la ligne n (instruction 210).
HPLOT : Tracé d'un segment entre les deux points dont les coordonnées sont contenues dans l'instruction (instruction 1010).
HGR : Mise en route du mode graphique (instruction 1000).

Le programme présenté n'est pas le plus rationnel pour le tracé d'une droite. Le développement par points est, en général, inutile ; donc l'ensemble des instructions comprises entre 100 et 190 peuvent être remplacées par une seule : le tracé d'un segment entre les points de coordonnées XI, YI et XF, YF.
Dans cet exemple, on a adopté une boucle pour élargir le programme. On obtient ainsi le graphique de n'importe quelle fonction différente de la droite.

Visualisation des axes cartésiens

La présentation graphique d'une fonction suppose l'affichage de ses axes cartésiens.
Les paramètres à fournir en entrée, indiqués dans l'organigramme, ont la même signification que prédédemment. Le sous-programme trace les axes comme deux segments perpendiculaires se croisant au point X0, Y0 pris comme origine (voir page 1444). Il dessine les flèches indiquant l'orientation des axes aux extrémités et visualise enfin des divisions sur chacun des axes.
Le symbole « flèche » s'obtient en traçant des segments parallèles de longueur décroissante (instructions 1030-1040 du listing) comme indiqué dans la figure de la page 1 444. On pourrait aussi recourir à un triangle, mais le résultat serait moins bon esthétiquement. Quoi qu'il en soit, cette deuxième solution est également présentée.
En appelant N la valeur positive maximale de la coordonnée Y, les instructions se réduisent au tracé de 3 segments :

	de :	à :
Segment 1	X0 - 1, N	X0 + 1, N
Segment 2	X0 + 1, N	X0, N - 3
Segment 3	X0, N - 3	X0 -1, N

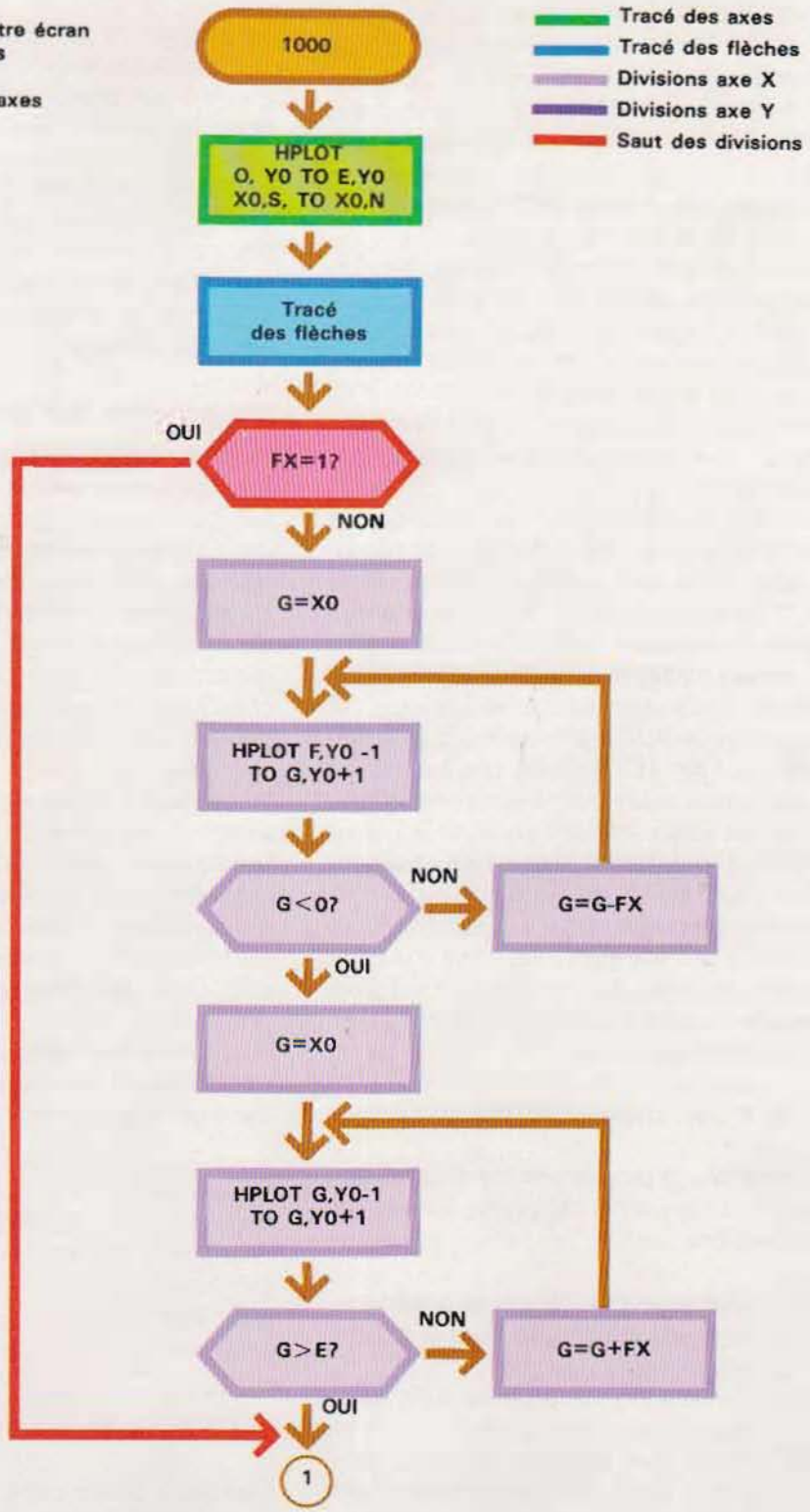
Elles sont introduites par une seule ligne de programme :

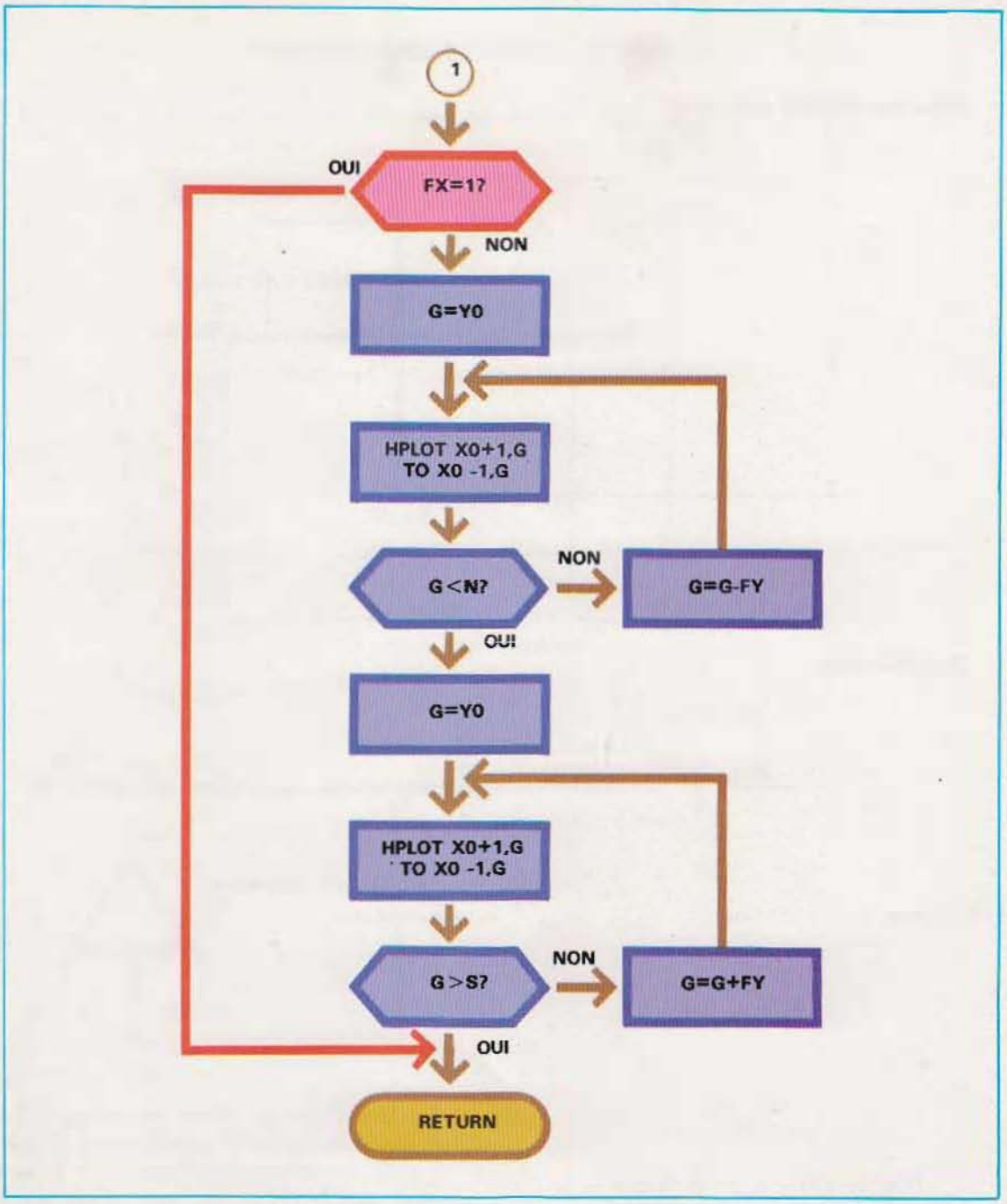
```
HPLOT X0 -1, N TO X0 +1, N TO X0, N -3
TO X0 -1, N
```

Pour les 3 autres extrémités des axes (S,E O) les instructions sont analogues.

ORGANIGRAMME DU TRACE DES AXES

Entrées :
N,S,E,O Limites fenêtre écran
Y0,Y0 Coordonnées de l'origine
FX,FY Echelle des axes





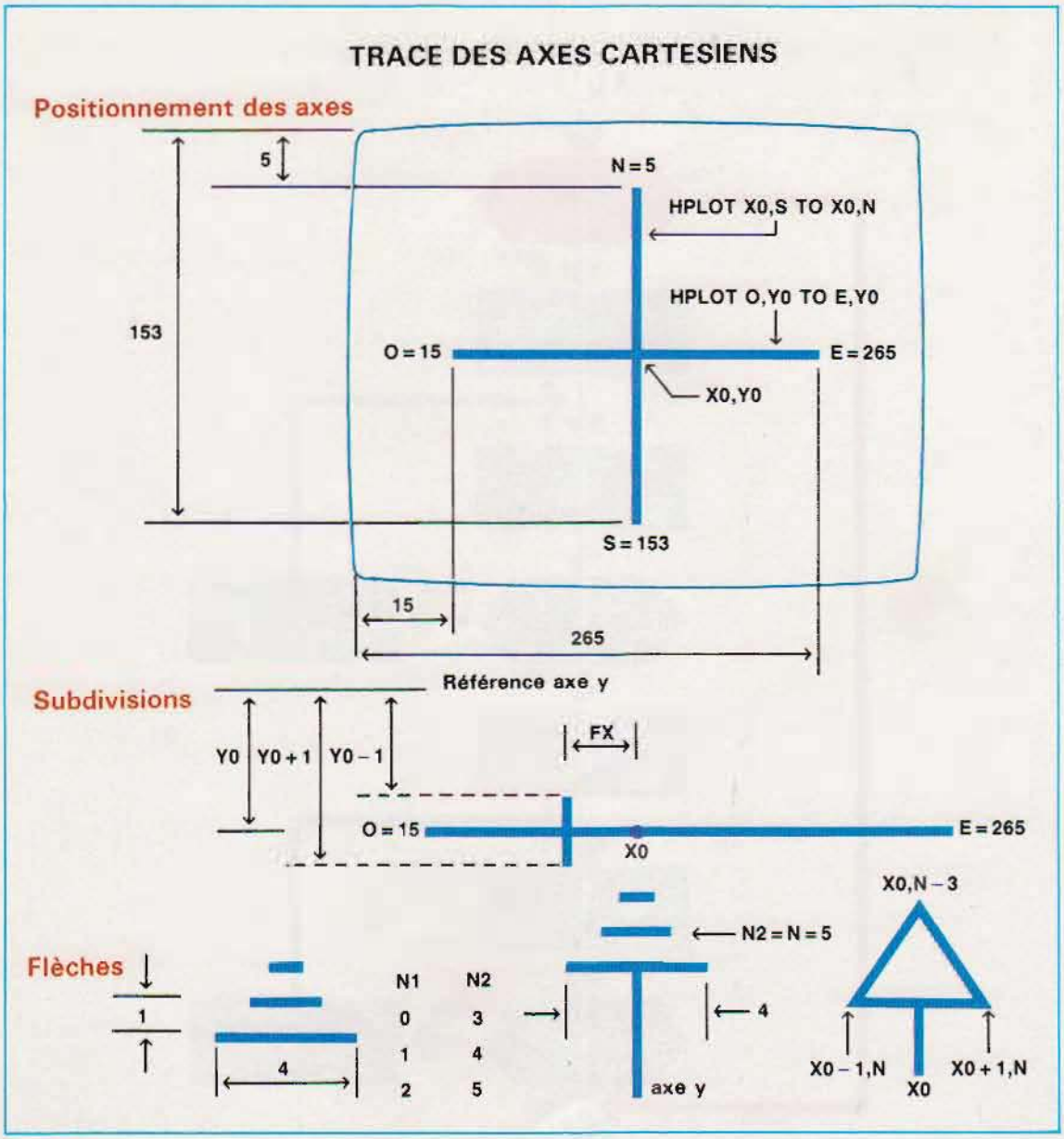
Tracé d'une droite à l'aide de deux points

Dans beaucoup d'applications graphiques, il est utile de disposer d'un sous-programme capable de déduire l'équation d'une droite passant par deux points dont on a les coordonnées. L'équation générale d'une droite s'écrivant : Y = A * X + B, pour trouver celle de la droite passant par deux points X1, Y1 et X2, Y2, on applique les formules :

$$A = \frac{Y2 - Y1}{X2 - X1}$$

$$B = \frac{(X1*X2)-(Y2*X1)}{X2-X1}$$

Soit les points X1 = 2, Y1 = 2,5 et X2 = 5, Y2 = 4, on aura :



$$A=\frac{4-2.5}{5-2}=\frac{1.5}{3}=0.5$$

$$B=\frac{(2.5*5)-(4*2)}{5-2}=\frac{12.5-8}{3}=\frac{4.5}{3}=1.5$$

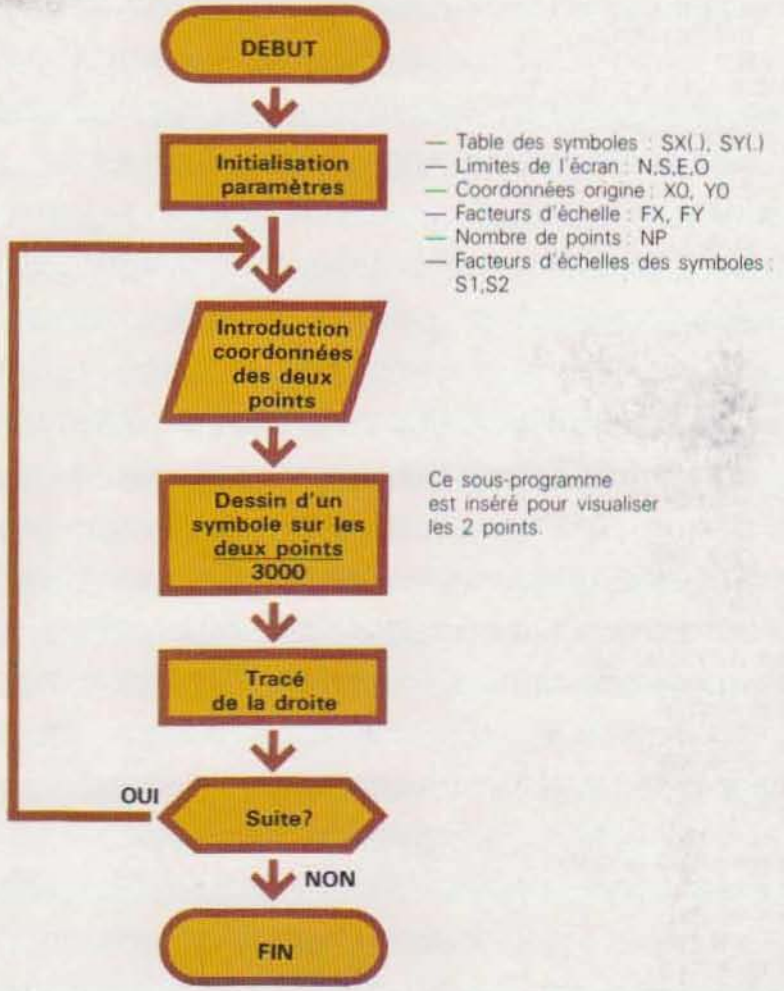
La droite aura donc pour équation :

$$Y=0.5*X+1.5$$

Les formules présentées servent à écrire un sous-programme de visualisation d'une droite passant par deux points dont les coordonnées seront introduites au clavier par exemple. Le programme demande les coordonnées du premier et du deuxième points (lignes 80 à 110) et restitue deux symboles (ici deux carrés, sous-programme 3000). Il calcule ensuite les valeurs de A et de B (lignes 120 à 130) et trace la droite (boucle : lignes 170 à 240).

Cette solution n'est indiquée qu'à titre d'illustration d'une méthode que nous utiliserons plus loin. Pour obtenir le même résultat, l'instruction du tracé d'un segment reliant les deux points donnés (HPLOT X1, Y1 TO X2, Y2) suffisait mais on n'aurait pas disposé des coefficients

DETERMINATION DE L'EQUATION ET TRACE D'UNE DROITE PASSANT PAR DEUX POINTS



A et B. La droite aurait été présentée comme un segment délimité par les points X1, Y1 et X2, Y2, et sans que le programme en connaisse la forme analytique (l'équation Y = A * X + B).

Le listing, qui se trouve en page 1446 et suivantes, montre comment se servir d'une table de déplacements pour générer un symbole sur n'importe quel point de l'écran. Les valeurs des coordonnées déterminant les côtés du symbole sont mémorisées aux lignes 20 et 30 (ici un carré). Pour le dessiner en un point donné, il faut ajouter ces valeurs à celles du point en question.

Le sous-programme 3000 fait apparaître, en outre, les coefficients S1 et S2 (égaux à 2 pour le carré, ligne 50) qui permettent de modifier la dimension de chaque segment. Les trois premiers côtés du symbole sont dessinés par la boucle 3050-3090 et le dernier côté par la ligne 3100.

Pour transporter ce programme sur d'autres machines que les Siprel, Apple et compatibles, il faut modifier les instructions VTAB, HPLOT,

TRACE D'UNE DROITE PASSANT PAR DEUX POINTS

Version Apple et compatibles

```
10 HOME
20 FOR J=1 TO 4
   : READ SX(J),SY(J)
   : NEXT J
30 DATA -1,1,1,1,-1,-1,-1
40 N=5
   : S=153
   : E=265
   : O=15
50 X0=140
   : Y0=80
   : FX=5
   : FY=5
   : NP=10
   : S1=2
   : S2=2
60 GOSUB 1000
70 DEF FN R(X)=A*X+B
80 VTAB 23
   : INPUT "COORDONNEES DU PREMIER POINT: " ; X1,Y1
90 XS=X1*FX+X0
   : YS=Y0-Y1*FY
   : GOSUB 3000
100 HOME
110 VTAB 23
   : INPUT "COORDONNEES DU DEUXIEME POINT: " ; X2,Y2
120 A=(Y2-Y1)/(X2-X1)
130 B=((Y1*X2)-(Y2*X1))/(X2-X1)
140 XS=X2*FX+X0
   : YS=Y0-Y2*FY
   : GOSUB 3000
150 P=(X2-X1)/NP
160 X=X1
   : Y=FN R(X)
170 FOR XX=X1+P TO X2 STEP P
180 YY=FN R(XX)
190 XA=X0+FX*X
   : YA=Y0-FY*Y
200 XB=X0+FX*XX
   : YB=Y0-FY*YY
210 IF XA>265 OR XA<15 OR XB>265 OR XB<15 OR YA>153 OR YA<5 OR YB>153 OR YB<5 THEN 230
220 HPLOT XA,YA TO XB,YB
230 X=XX
   : Y=YY
240 NEXT
250 HOME
260 VTAB 23
270 INPUT "POURSUIVRE? (O/N) " ; S$
280 IF S$="O" THEN RUN
290 END
997 REM ----------------
998 REM TRACE DES AXES
999 REM ----------------
1000 HGR
   : HCOLOR=3
1010 HPLOT 0-15,N-5 TO E+14,N-5 TO E+14,S+3 TO 0-15,S+3 TO 0-15,N-5
1020 HPLOT 0,Y0 TO E,Y0
   : HPLOT X0,S TO X0,N
1030 N2=N
   : FOR N1=2 TO 0 STEP -1
```

```
    : HPLOT X0-N1,N2 TO X0+N1,N2
    : N2=N2-1
    : NEXT N1
1040 N2=E
    : FOR N1=2 TO 0 STEP -1
    : HPLOT N2,Y0-N1 TO N2,Y0+N1
    : N2=N2+1
    : NEXT N1
1050 HPLOT E+6,Y0-2 TO E+11,Y0+3
    : HPLOT E+11,Y0-2 TO E+6,Y0+3
1060 HPLOT X0-10,N-2 TO X0-7,N+1
    : HPLOT X0-4,N-2 TO X0-10,N+3
1070 IF FX=1 THEN 1140
1080 FOR G=X0 TO 0 STEP -FX
1090 HPLOT G,Y0+1 TO G,Y0-1
1100 NEXT G
1110 FOR G=X0 TO E STEP FX
1120 HPLOT G,Y0+1 TO G,Y0-1
1130 NEXT G
1140 IF FY=1 THEN 1210
1150 FOR G=Y0 TO N STEP -FY
1160 HPLOT X0+1,G TO X0-1,G
1170 NEXT G
1180 FOR G=Y0 TO S STEP FY
1190 HPLOT X0+1,G TO X0-1,G
1200 NEXT G
1210 RETURN
3000 REM ------------------------
3010 REM DESSINER DES SYMBOLES
3020 REM ------------------------
3030 XA=SX(1)*S1+XS
    : YA=SY(1)*S2+YS
3040 XC=XA
    : YC=YA
3050 FOR I=2 TO 4
3060 XB=SX(I)*S1+XS
    : YB=SY(I)*S2+YS
3070 HPLOT XA,YA TO XB,YB
3080 XA=XB
    : YA=YB
3090 NEXT I
3100 HPLOT XB,YB TO XC,YC
3110 RETURN
```

Version Olivetti M20

```
10 CLEAR
20 MP=10
30 W?=WINDOW(2,40)
40 DEF FN R(X)=A*X+B
50 CLS%2
60 LINE%2(0,0)-(511,255),3,B
70 CLS
80 SCALE%2,-50,50,-50,50
90 GOSUB 270
100 INPUT "PREMIER POINT" ; X1,Y1
110 XA=X1
    : YA=Y1
    : GOSUB 3000
120 INPUT "DEUXIEME POINT" ; X2,Y2
130 XA=X2
    : YA=Y2
    : GOSUB 3000
140 P=(X2-X1)/MP
```

```
150 A=(Y2-Y1)/(X2-X1)
160 B=((Y1*X2)-(Y2*X1))/(X2-X1)
170 X=X1
    : Y=FN R(X)
180 FOR XX=X1+P TO X2 STEP P
190 YY=FN R(XX)
200 LINE%2(X,Y)-(XX,YY),2
210 X=XX
    : Y=YY
220 NEXT
230 INPUT "POURSUIVRE? (O/N)" ; A$
240 IF A$="O" THEN 10
250 CLEAR
    : END
270 REM **** TRACE DES AXES ****
280 LINE%2(0,-50)-(0,50)
290 LINE%2(-50,0)-(50,0)
300 FOR Q=0 TO 49 STEP 2
310 LINE%2(-.5,Q)-(.5,Q) 320 LINE%2(Q,-.5)-(Q,.5)
330 NEXT Q
340 FOR Q=0 TO -49 STEP -2
350 LINE%2(-.5,Q)-(.5,Q)
360 LINE%2(Q,-.5)-(Q,.5)
370 NEXT Q
380 RETURN
3000 REM **** DESSINER DES SYMBOLES ****
3010 LINE%2(XA-2,YA-2)-(XA+2,YA+2),2,B
3020 RETURN
```

HCOLOR, HGR et les contrôles de la ligne 210, en tenant compte des valeurs numériques de l'écran utilisé.
On a donné, à titre d'exemple, un listing convenant à un Olivetti M20. Ce programme fait appel à certaines instructions particulières que nous détaillerons par la suite.

Régression linéaire des moindres carrés

L'analyse d'un phénomène consiste à observer les valeurs prises par les diverses variables, au cours de l évolution dudit phénomène, à condition qu'il n'y ait qu'une seule source de variation.
Le cas le plus simple est déterminé par deux variables dont l'une est indépendante. L'ensemble des valeurs prises par cette variable montre, sous forme graphique, l'évolution du phénomène observé.
Partons d'un exemple fondé sur 5 observations ayant donné les valeurs suivantes pour les variables X et Y.

Observation	X	Y
a	2,5	3
b	4	3,5
c	5	5
d	7	4,5
e	9	6

On constate qu'on ne peut déduire immédiatement la valeur de Y correspondant à une valeur de X non observable. Ces informations sont obtenues selon une méthode — déjà présentée — qui consiste à extrapoler linéairement chaque paire de valeurs observées.
L'évolution réelle est ainsi assimilée à une série de segments joignant les points a, b, c, d, e. Le résultat est un graphique (voir ci-contre) qui peut être très éloigné de la réalité, avec des valeurs déduites grossières. Dans cette méthode, les équations représentant le phénomène sont celles des quatre droites qui joignent deux à deux les points connus.
Chaque équation (donc la forme analytique décrivant le phénomène) est déduite de la méthode développée dans le paragraphe précé-

dent (droite passant par deux points donnés). Il existe une seconde méthode appelée « régression linéaire des moindres carrés » qui, sous certaines conditions, fournit des résultats plus fiables.

Supposons, pour l'instant, que l'évolution du phénomène puisse être symbolisée par une droite. Grâce à une série d'observations sur les variables, on obtiendra des valeurs qui n'appartiennent pas toutes à la droite. Même si la loi régissant le phénomène est représentée par une droite, les valeurs mesurées pourraient toutefois être encore les points a, b, c, d, e (ligne rouge) lesquels, à une exception près (b), ne coïncident pas avec la droite. Cette « dispersion » dépend de nombreux facteurs, liés à la nature du phénomène observé.

Dans un phénomène physique, les erreurs peuvent être instrumentales et accidentelles. Quel que soit son degré de sophistication, un équipement ne fournit jamais qu'une valeur approximative de la grandeur.

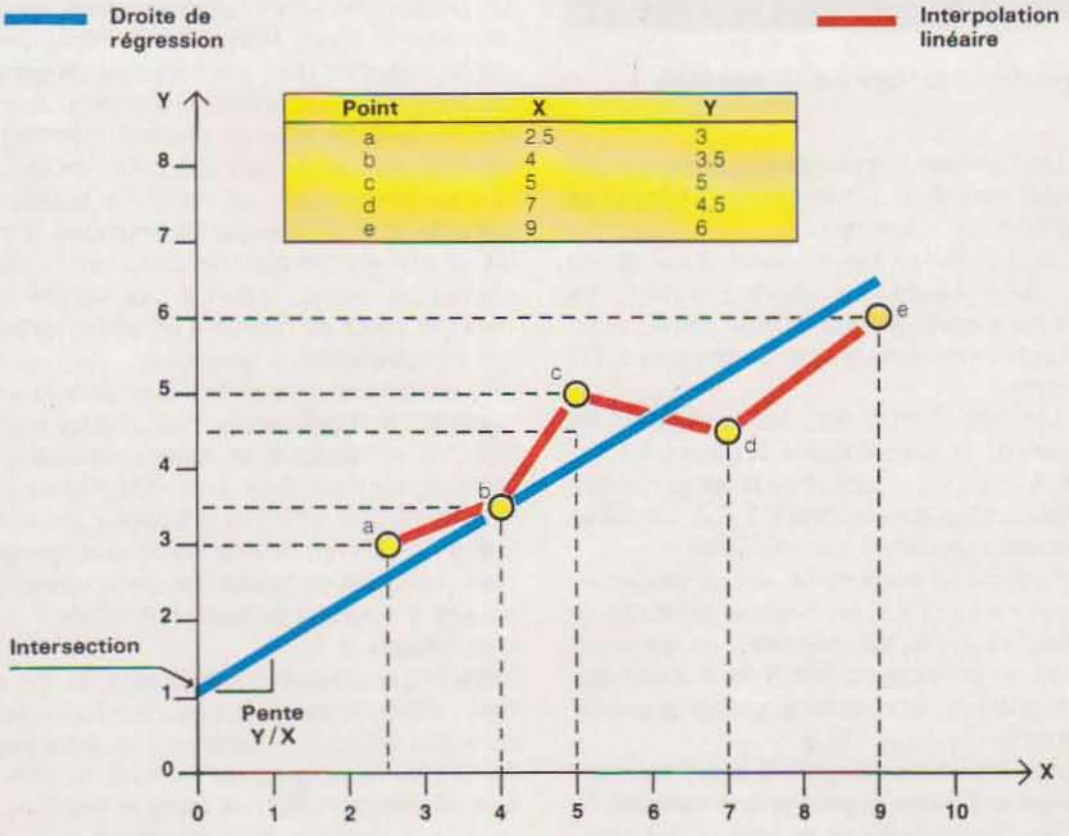
Autre cause d'erreur : l'influence de facteurs extérieurs tels que vibrations mécaniques, variations de température ou d'humidité, etc.

Ces erreurs se combinent et ne s'annulent que d'une manière aléatoire. D'ailleurs, la vérification de cette dernière situation n'est pas prévisible : même si les valeurs sont exactes à un moment donné, il n'existe pas de moyen de le vérifier. Les données doivent donc de toute manière être considérées comme entachées d'erreur.

En statistiques, les erreurs sont plus complexes et plus difficiles à repérer.

La méthode des moindres carrés permet de rechercher la forme analytique de la fonction qui se rapproche le plus du phénomène exami-

EVOLUTION D'UN PHENOMENE RESULTANT DE 5 OBSERVATIONS





Marka

Application graphique sur Olivetti M20.

né. Les formules à appliquer proviennent d'une analyse complexe dépendant de l'hypothèse de départ de la fonction.

Le cas le plus simple est celui d'une droite (régression linéaire des moindres carrés) : les formules à appliquer pour obtenir son équation sont contenues dans le sous-programme 8000 ci-contre.

Les variables A et B sont les coefficients de l'équation ; ils sont calculés à travers les valeurs X (.) et Y (.) observées, N est le nombre d'observations, les variables C1, C2, C3, C4 et D servant aux calculs intermédiaires.

Pour utiliser ce programme, il faut ranger en mémoire X (.) et Y (.) les mesures et les observations, et en N leur nombre ; en sortie on obtient les coefficients A et B de la droite qui, par hypothèse, représente le phénomène (voir organigramme page 1452).

La première phase consiste à tracer les axes cartésiens. Ensuite, le programme demande le nombre de points (N) et les valeurs des coordonnées des points observés (boucle avec indice I). Pour chaque paire de coordonnées, le sous-programme 5900 dessine un carré sur le point correspondant à la valeur observée. Après la boucle d'introduction, on appelle le sous-programme 8000 (ci-contre), qui calcule les coefficients A et B. Le résultat est matérialisé par une droite.

En réalité, le déroulement du programme est beaucoup plus complexe, en particulier au niveau des facteurs d'échelle.

En phase d'introduction, on peut avoir des valeurs de X ou de Y qui, d'après l'échelle adoptée, « sortiraient » des limites de l'écran. De même, après avoir calculé l'équation de la droite (Y=B * X+A), dessinée à partir des valeurs X=XN et X=XM (respectivement minimum et maximum sur l'axe X), il peut arriver que les valeurs correspondantes de Y dépassent les limites de l'écran.

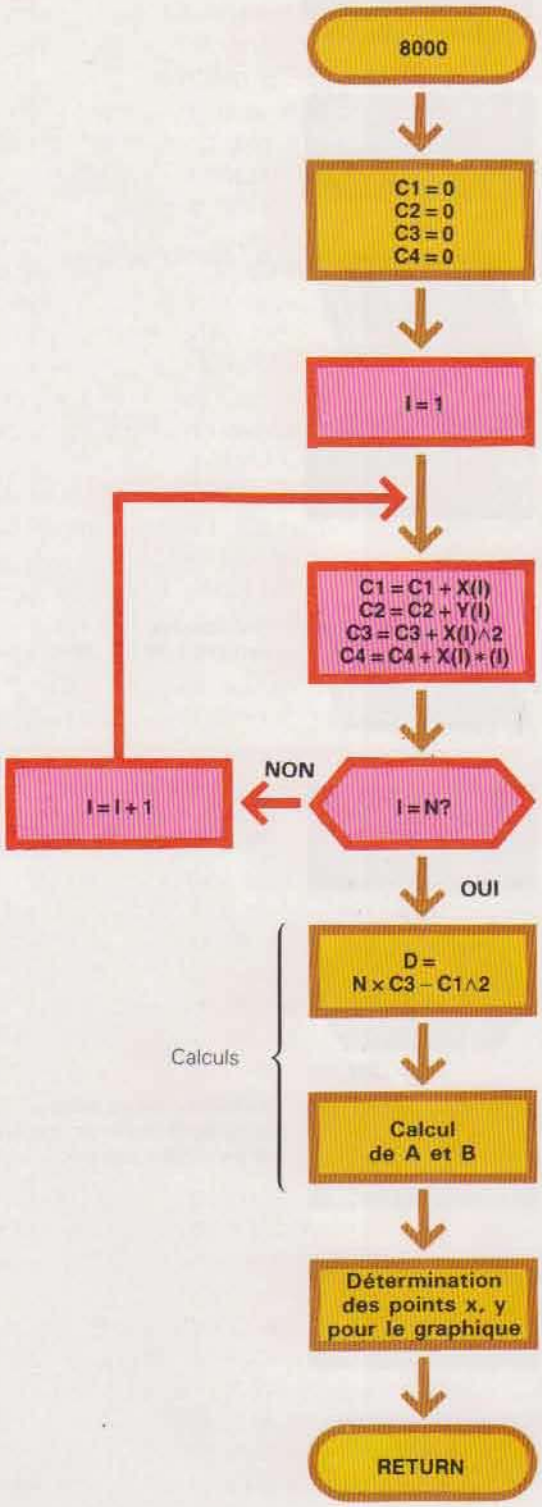
Le premier contrôle (limites des valeurs X et Y) s'effectue après chaque entrée, tandis que celui sur la droite est réalisé par le sous-programme 8000 (dont le listing est différent de l'organigramme présenté).

Le programme (dont l'organigramme détaillé se trouve en pages 1453, 1454 et 1455) révèle des complexités dues à la nécessité de prévoir un changement automatique d'échelle. Au début, l'origine des axes est placé en X0=140 et Y0=80, avec un facteur d'échelle F = 0,7. A chaque introduction, on lance une boucle de contrôle pour vérifier que les nouveaux points se situent encore dans le cadre de l'échelle choisie par défaut : sinon, il faut calculer une nouvelle valeur du facteur d'échelle et redessiner complètement le graphique.

La logique exposée s'obtient avec une série de contrôles et d'indicateurs. Pour chaque entrée X(I), Y(I), on recalcule les valeurs maximales et minimales sur les deux axes (XN, XM et YN, YM, lignes 270 à 360) et la longueur des intervalles (DX, DY) parmi lesquels on sélectionne le plus grand. En se basant sur cette valeur, on calcule à nouveau le facteur d'échelle F, qui sera inférieur à 1.

Dans le cas contraire, cela signifie qu'une erreur a été commise et que les données ne sont donc pas valides. Le traitement est alors interrompu et le programme reprend au début, avec l'instruction RUN et après le test F < 1. Le facteur d'échelle étant déterminé, on passe au calcul des nouvelles coordonnées X0, Y0

CALCUL DES COEFFICIENTS A ET B DE LA DROITE DE REGRESSION



Initialisation

Boucle sur les points observés

Calculs

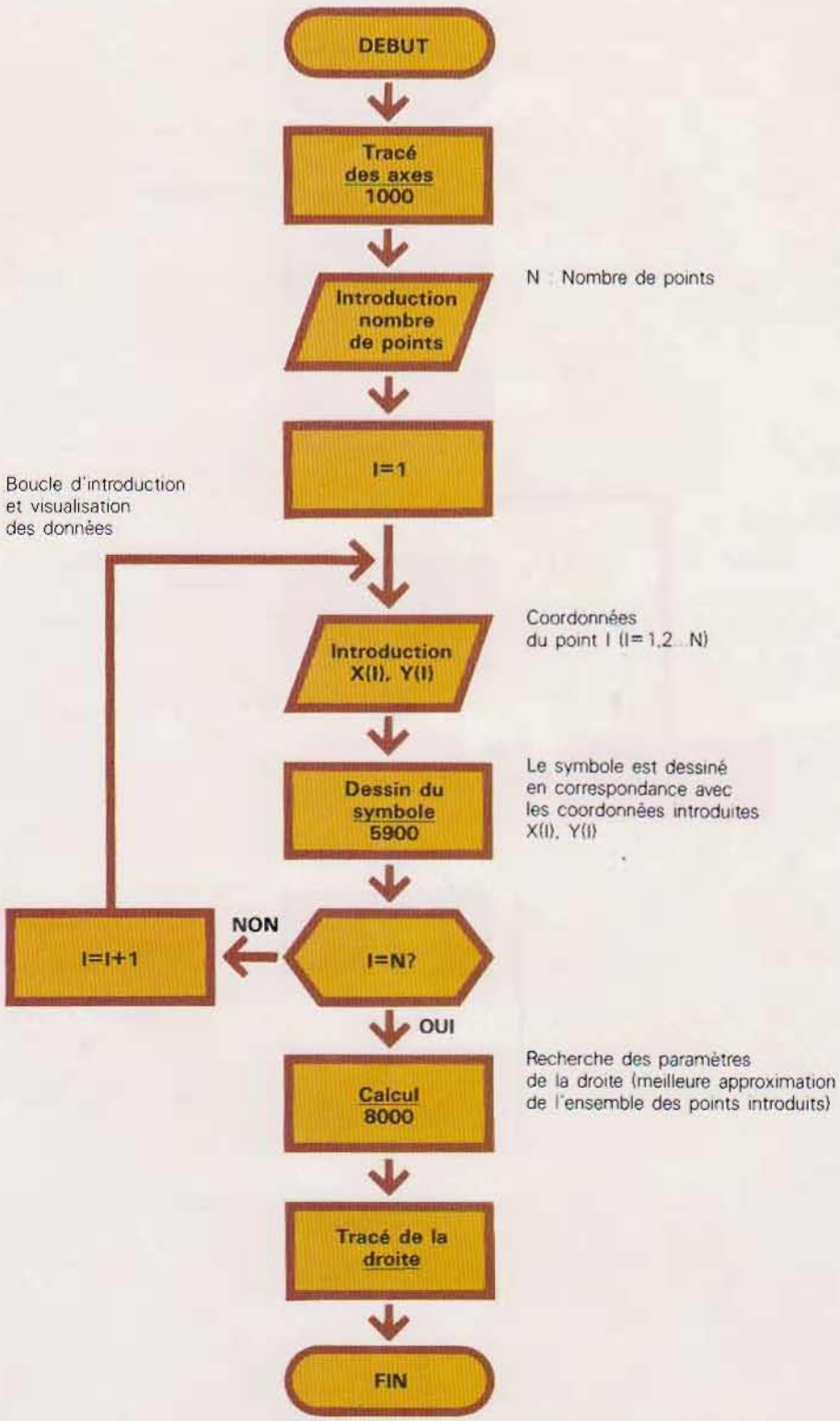
Paramètres de la droite de régression des moindres carrés

$$A = \frac{C2 * C3 - C1 * C4}{D}$$

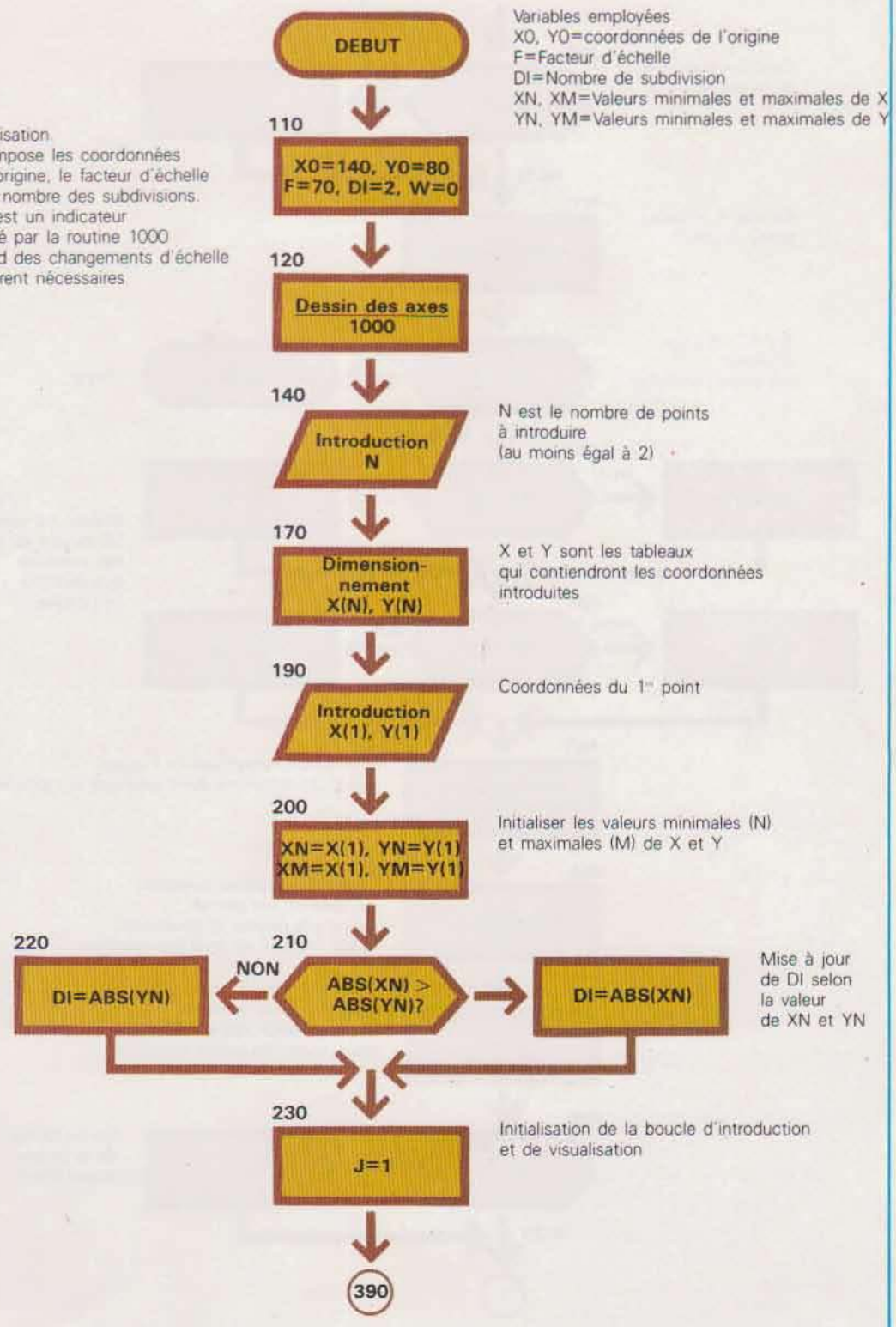
$$B = \frac{N * C4 - C1 * C2}{D}$$

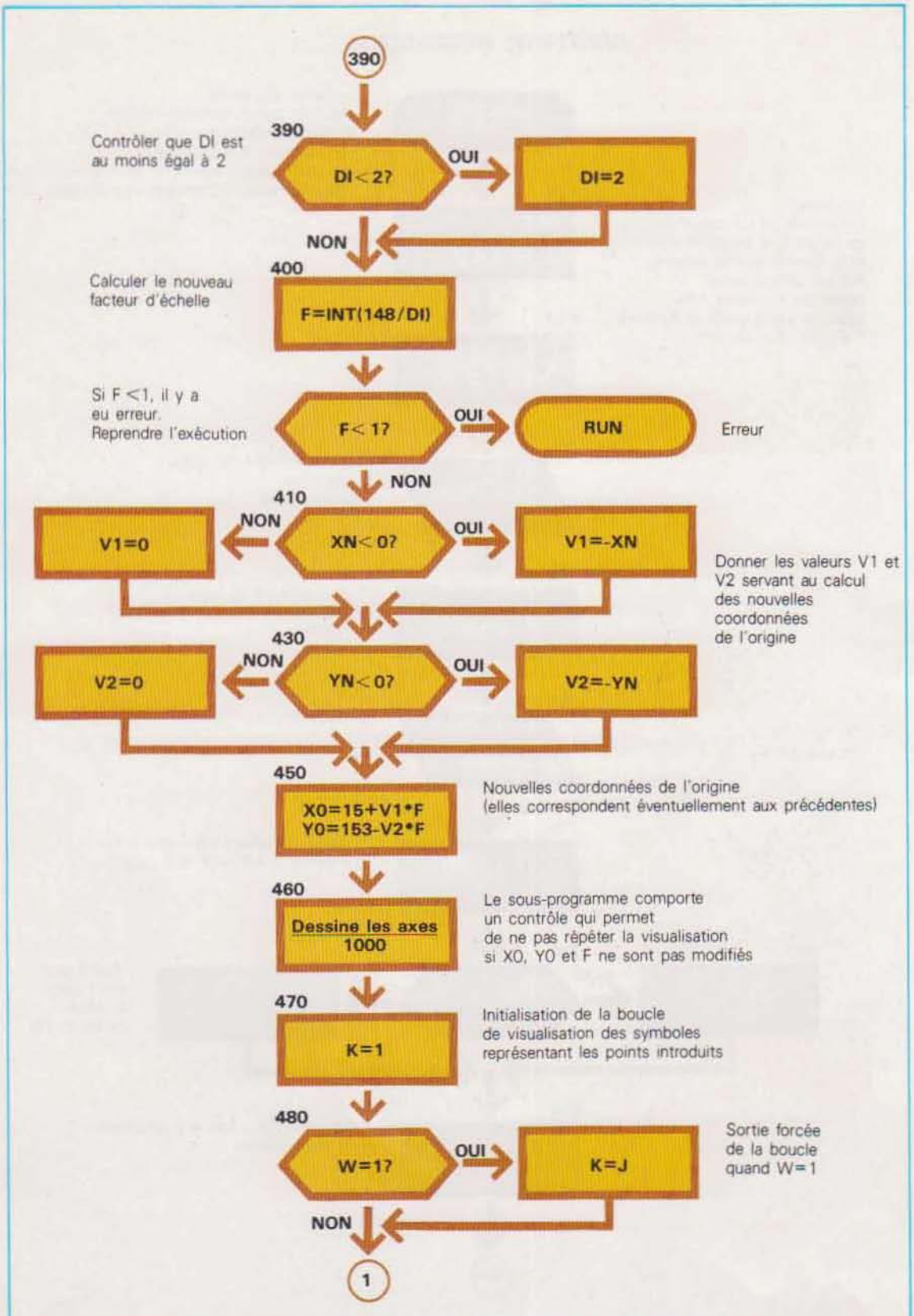
$$Y = A * X + B$$

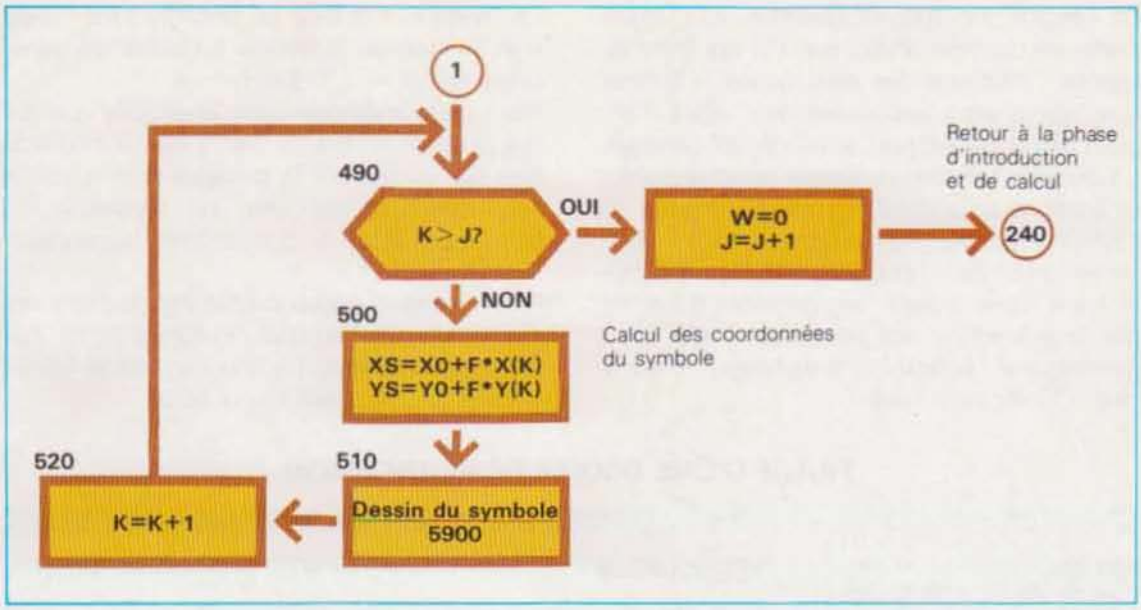
REGRESSION LINEAIRE DES MOINDRES CARRES

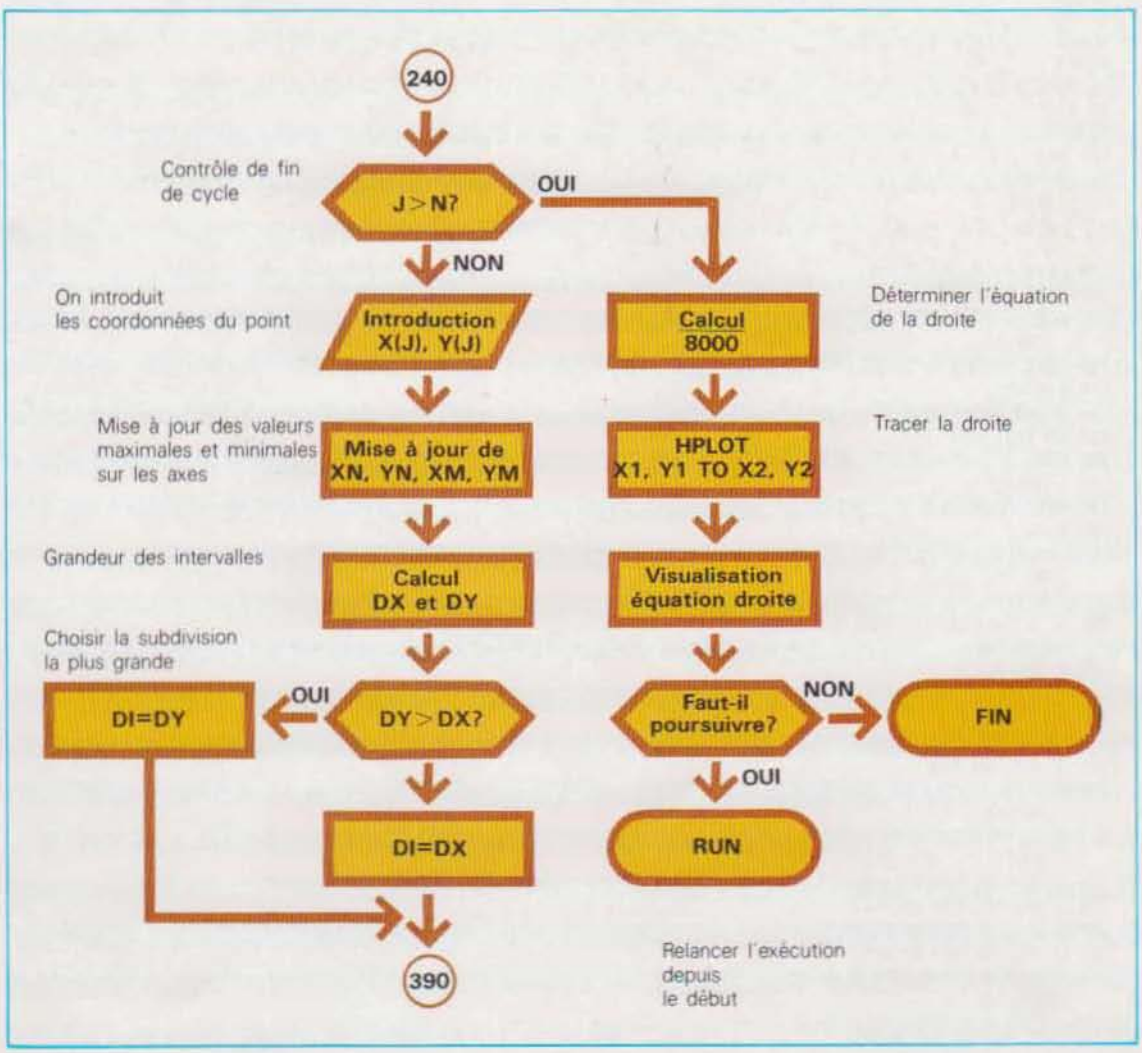


DROITE DE REGRESSION




390
390
Contrôler que DI est au moins égal à 2
DI<2?
OUI
DI=2
NON
400
Calculer le nouveau facteur d'échelle
F=INT(148/DI)
Si F<1, il y a eu erreur. Reprendre l'exécution
F<1?
OUI
RUN
Erreur
NON
410
NON
V1=0
XN<0?
OUI
V1=-XN
Donner les valeurs V1 et V2 servant au calcul des nouvelles coordonnées de l'origine
430
NON
V2=0
YN<0?
OUI
V2=-YN
450
X0=15+V1*F
Y0=153-V2*F
Nouvelles coordonnées de l'origine (elles correspondent éventuellement aux précédentes)
460
Dessine les axes
1000
Le sous-programme comporte un contrôle qui permet de ne pas répéter la visualisation si X0, Y0 et F ne sont pas modifiés
470
K=1
Initialisation de la boucle de visualisation des symboles représentant les points introduits
480
W=1?
OUI
K=J
Sortie forcée de la boucle quand W=1
NON
1


1
490
K > J?
OUI
W=0
J=J+1
240
Retour à la phase d'introduction et de calcul
NON
500
XS=X0+F*X(K)
YS=Y0+F*Y(K)
Calcul des coordonnées du symbole
520
510
K=K+1
Dessin du symbole
5900


240
Contrôle de fin de cycle
J > N?
OUI
NON
On introduit les coordonnées du point
Introduction X(J), Y(J)
Calcul 8000
Déterminer l'équation de la droite
Mise à jour des valeurs maximales et minimales sur les axes
Mise à jour de XN, YN, XM, YM
HPLOT X1, Y1 TO X2, Y2
Tracer la droite
Grandeur des intervalles
Calcul DX et DY
Visualisation équation droite
Choisir la subdivision la plus grande
DI=DY
OUI
DY > DX?
Faut-il poursuivre?
NON
FIN
OUI
DI=DX
RUN
390
Relancer l'exécution depuis le début

de l'origine. Le sous-programme 1000 comporte un contrôle (indicateur W) qui évite de répéter l'affichage des axes quand le facteur d'échelle et les coordonnées de l'origine n'ont pas changé par rapport au précédent passage. La dernière fonction redessine tous les points, toujours sous la condition d'une variation du facteur d'échelle. Au passage, notons bien la sortie forcée de la boucle. Si le facteur d'échelle varie, une boucle représentant tous les points précédents est nécessaire ; elle est à éviter quand l'échelle reste inchangée, c'est-à-dire si l'indicateur vaut 1.

La fonction s'obtient en posant, sous condition, l'indice de la boucle à l'extrémité supérieure (IF W = 1 THEN K = J).

Mais cette méthode n'est applicable que sur une quantité limitée de machines. La modification qui en permet la généralisation consiste simplement à introduire une dérivation (by pass) pour toute la boucle avec la condition W=1.

On trouvera ci-dessous le listing du programme, en version Personal Kid (Siprel 2010, Apple et compatibles). La transposition se fait en remplaçant certaines instructions.

TRACE D'UNE DROITE DE REGRESSION

```
10 REM ------------------------
20 REM DROITES DE REGRESSION
30 REM ------------------------
40 REM
50 REM
60 REM
70 FOR J=1 TO 4
80 READ XS(J),SY(J)
90 NEXT J
100 DATA -1,1,1,1,1,-1,-1,-1
110 X0=140
    : Y0=80
    : E=70
    : D1=2
    : W=0
120 GOSUB 1000
130 HOME
    : VTAB 22
140 INPUT "NOMBRE DE POINTS. " ; N
150 N=INT(N)
160 IF N<2 THEN 130
170 DIM X(N),Y(N)
180 HOME
    : VTAB 22
190 INPUT "POINT N.1: " ; X(1),Y(1)
200 XN=X(1)
    : XM=X(1)
    : YN=Y(1)
    : YM=Y(1)
210 IF ABS(XN)>ABS(YN) THEN D1=ABS(XN)
    : GOTO 230
220 D1=ABS(YN)
    : GOTO 230
230 J=1
    : GOTO 390
240 IF J>N THEN 540
250 HOME
    : VTAB 22
260 PRINT "POINT N."J" "
    : INPUT " " ; X(J),Y(J)
270 IF X(J)>XM THEN XM=X(J)
280 IF X(J)<XN THEN XN=X(J)
290 IF Y(J)>YM THEN YM=Y(J)
300 IF Y(J)<YN THEN YN=Y(J)
310 IF XN>0 AND XN<0 THEN DX=XM-XN
    : GOTO 340
```

```
320 IF XN>=0 AND XM>0 THEN DX=XM
    : GOTO 340
330 DX=-XN
340 IF YN<0 AND YM>0 THEN DY=YM-YN
    : GOTO 370
350 IF YN>=0 AND YM>0 THEN DY=YM
    : GOTO 370
360 DY=-YN
370 IF DY>DX THEN DI=DY
    : GOTO 390
380 DI=DX
390 IF DI<2 THEN DI=2
400 F=INT(140/DI)
    : IF F<1 THEN RUN
410 IF XN<0 THEN U1=-XN
    : GOTO 430
420 U1=0
430 IF YN<0 THEN U2=-YN
    : GOTO 450
440 U2=0
450 X0=U1*F+15
    : Y0=153-U2*F
460 GOSUB 1000
470 K=1
480 IF W=1 THEN K=J
490 IF K>J THEN W=0
    : GOTO 530
500 XS=X(K)*F+X0
    : YS=Y0-Y(K)*F
510 GOSUB 5900
    : REM SYMBOLE
520 K=K+1
    : GOTO 490
530 J=J+1
    : GOTO 240
540 GOSUB 8000
    : REM CALCUL
550 HPLOT X1,Y1 TO X2,Y2
560 S$="-"
570 IF A>0 THEN S$="+"
580 HOME
    : VTAB 22
590 PRINT "LA DROITE LA PLUS PROCHE EST: "
600 IF A=0 AND B=0 OR D=0 THEN INVERSE
    : PRINT "X= "XN
    : GOTO 630
610 IF B=0 THEN INVERSE
    : PRINT "Y= "A
    : GOTO 630
620 INVERSE
    : PRINT "Y= "B " *X "S$ " "ABS(A)
630 NORMAL
640 VTAB 24
650 PRINT "POURSUIVRE? (O/N)"
    : GET A$
660 IF A$="O" THEN RUN
670 TEXT
    : HOME
```

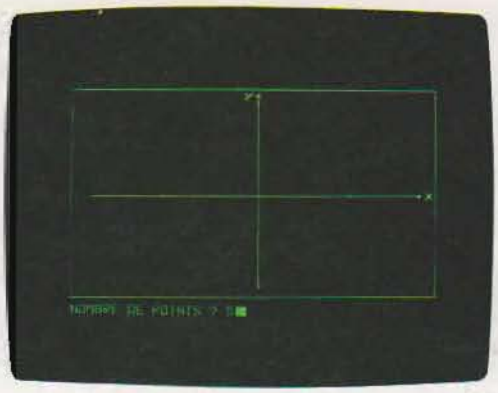
```
    : END
997 REM ---------------------
998 REM DESSINER LES AXES
999 REM ---------------------
1000 IF X0=XP AND Y0=YP AND F=FF THEN W=1
    : RETURN
1010 HGR
    : HCOLOR=3
1020 HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
1030 HPLOT 15,Y0 TO 256,Y0
    : HPLOT X0,153 TO X0,5
1040 N2=5
    : FOR N1=2 TO 0 STEP -1
    : HPLOT X0-N1,N2 TO X0+N1,N2
    : N2=N2-1
    : NEXT N1
1050 N2=265
    : FOR N1=2 TO 0 STEP-1
    : HPLOT N2,Y0-N1 TO N2,Y0+N1
    : N2=N2+1
    : NEXT N1
1060 HPLOT 271,Y0-2 TO 276,Y0+3
    : HPLOT 276,Y0-2 TO 271,Y0+3
1070 HPLOT X0-10,3 TO X0-7,6
    : HPLOT X0-4,3 TO X0-10,8
1080 IF F=1 THEN 1220
1090 FOR G=X0 TO 15 STEP -F
1100 HPLOT G,Y0+1 TO G,Y0-1
1110 NEXT G
1120 FOR G=X0 TO 265 STEP F
1130 HPLOT G,Y0+1 TO G,Y0-1
1140 NEXT G
1150 FOR G=Y0 TO 5 STEP -F
1160 HPLOT X0+1,G TO X0-1,G
1170 NEXT G
1180 FOR G=Y0 TO 153 STEP F
1190 HPLOT X0+1,G TO X0-1,G
1200 NEXT G
1210 XP=X0
    : YP=Y0
    : FF=F
1220 RETURN
5899 REM ---------
5900 REM SYMBOLE
5901 REM ---------
5910 S=F/4
    : IF S>5 THEN S=5
5920 IF S<2 THEN S=2
5930 XA=SX(1)*S+XS
    : YA=SY(1)*S+YS
5940 XC=XA
    : YC=YA
5950 FOR T=2 TO 4
5960 XB=SX(T)*S+XS
    : YB=SY(T)*S+YS
5970 HPLOT XA,YA TO XB,YB
5980 XA=XB
    : YA=YB
5990 NEXT T
6000 HPLOT XB,YB TO XC,YC
6010 RETURN
```

```
7999 REM --------
8006 REM CALCUL
8010 REM --------
8020 C1=0
   : C2=0
     C3=0
   : C4=0
8030 FOR I=1 TO N
8040 C1=C1+X(I)
8050 C2=C2+Y(I)
8060 C3=C3+X(I)^2
8070 C4=C4+X(I)*Y(I)
8080 NEXT I
8090 D=N*C3-C1^2
8100 IF D=0 THEN Y1=Y0-YN*F
   : Y2=Y0-YM*F
   : X1=X0+XN*F
   : X2=X1
   : RETURN
8110 A=(C2*C3-C1*C4)/D
8120 B=(N*C4-C1*C2)/D
8130 IF B=0 THEN Y1=Y0-A*F
   : Y2=Y1
   : X1=X0+XN*F
   : X2=X0+XM*F
     RETURN
8140 IF A=0 AND B=0 THEN Y1=Y0-YN*F
   : Y2=Y0-YM*F
   : X1=X0+XN*F
   : X2=X1
   : RETURN
8150 WS=0
8160 X=(YM-A)/B
   : Y=YM
8170 IF X>=XN AND X<=XM THEN GOSUB 8270
8180 X=(YN-A)/B
   : Y=YN
8190 IF X>=XN AND X<=XM THEN GOSUB 8270
8200 X=XM
   : Y=B*XM+A
8210 IF Y>=YN AND Y<=YM THEN GOSUB 8270
8220 X=XN
   : Y=B*XN+A
8230 IF Y>=YN AND Y<=YM THEN GOSUB 8270
8240 X1=X0+X1*F
   : X2=X0+X2*F
8250 Y1=Y0-Y1*F
   : Y2=Y0-Y2*F
8260 RETURN
8270 IF WS=1 THEN GOTO 8290
8280 X1=X
   : Y1=Y
   : WS=1
   : RETURN
8290 X2=X
   : Y2=Y
   : RETURN
```

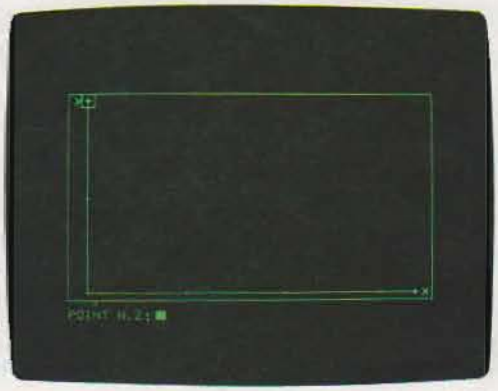
REGRESSION LINEAIRE DES MOINDRES CARRES

Le système présente des axes cartésiens et demande le nombre de points.
Pour vérifier la précision du logiciel, on va introduire 5 points appartenant à la droite Y=3*X+2.

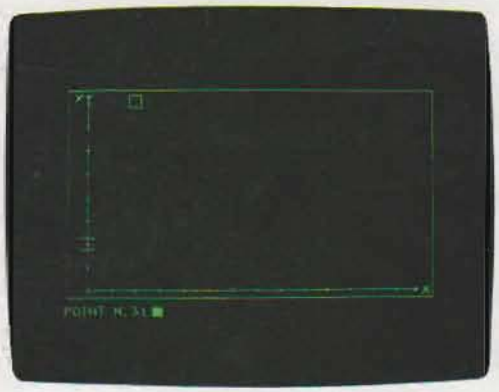
	X	Y
Point 1	0	2
Point 2	2	8
Point 3	3	11
Point 4	5	17
Point 5	10	32



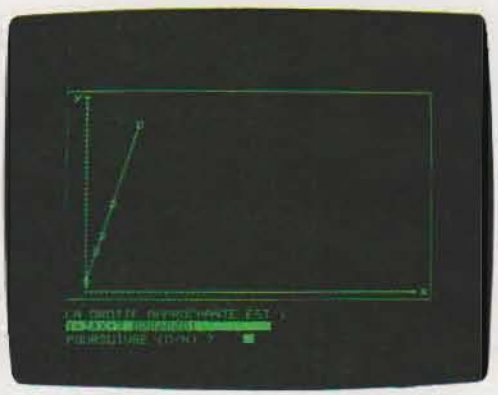
Dans le système, on a déjà introduit le 1er point de coordonnées 0,2. Le programme adapte la position des axes cartésiens de manière à disposer de toute la surface de l'écran pour la représentation des valeurs introduites.
Par rapport aux écrans précédents, l'origine s'est déplacée en bas à gauche, dans la mesure où les valeurs introduites (pour l'instant seulement le point 0,2) ne contiennent pas de nombres négatifs : l'écran est donc entièrement dédié au cadre positif du système de référence, donc la valeur maximale de Y est égale à la valeur maximale entrée.



Après l'introduction du point 2 (de coordonnées 2,8) le programme réduit l'échelle de l'axe Y de manière à l'adapter à la nouvelle abscisse (Y=8). Le point précédent semble avoir été déplacé vers le bas, tout en correspondant aux mêmes coordonnées. En effet, dans l'écran précédent, l'axe Y ne comportait que deux parties (petits traits horizontaux) et le symbole occupait la position 2 (c'est-à-dire l'ordonnée Y=2) ; dans la nouvelle situation, le programme a automatiquement modifié l'échelle en la réglant sur la nouvelle valeur maximale entrée (Y=8).



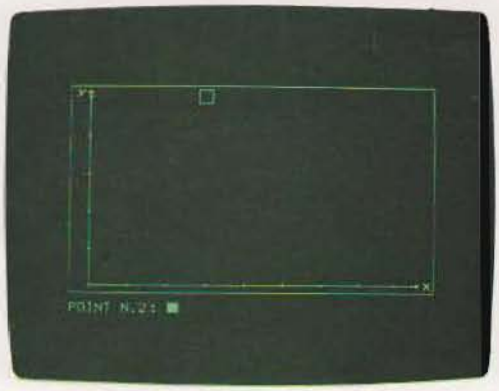
A la fin de l'introduction des données (dont le nombre a été déclaré au début) le programme présente le graphique final, qui comprend les points et la régression. De plus, il visualise l'expression analytique de la droite. Les coefficients calculés ne présentent aucune différence par rapport aux valeurs employées pour déterminer les points. La valeur 1, qui apparaît à la suite d'une série de zéros après la virgule, est due à l'absence d'arrondi dans la présentation des coefficients. Dans les applications pratiques, on tronque le résultat à la 2e ou à la 3e décimale.



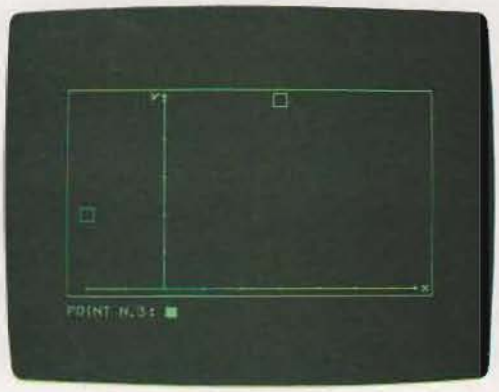
Contrairement au premier, le second exemple se réfère à un calcul de régression proprement dit, effectué sur les valeurs suivantes

	X	Y
Point 1	3	5
Point 2	- 2	
Point 3	10	6
Point 4	- 5	3
Point 5	1	- 2

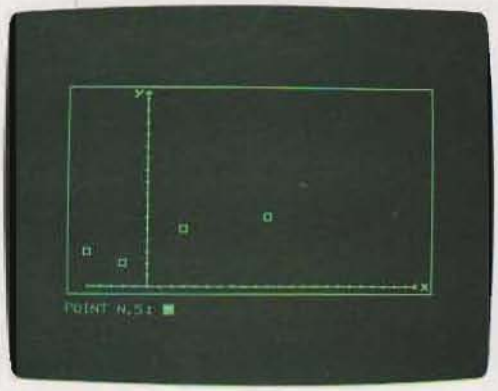
Les coordonnées du premier point sont entrées et le programme demande la 2° donnée.



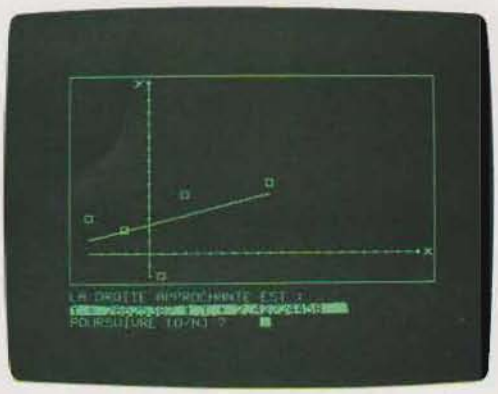
L'introduction de la première valeur négative (point 2, coordonnées -2,2) provoque le déplacement de l'axe Y vers la droite, car on a besoin d'une portion négative de l'axe Y (-2).



Les introductions suivantes déterminent l'adaptation du champ de vision (position de l'origine et facteur d'échelle). Dans la photo, on vient d'introduire le 4° point.



L'introduction du dernier point (1, -2) lance le calcul et la visualisation de la droite de régression.
Comme on peut le remarquer, les valeurs dans ce cas sont très dispersées et la droite de régression fournit une approximation relativement satisfaisante.



Programme du tracé graphique d'une fonction

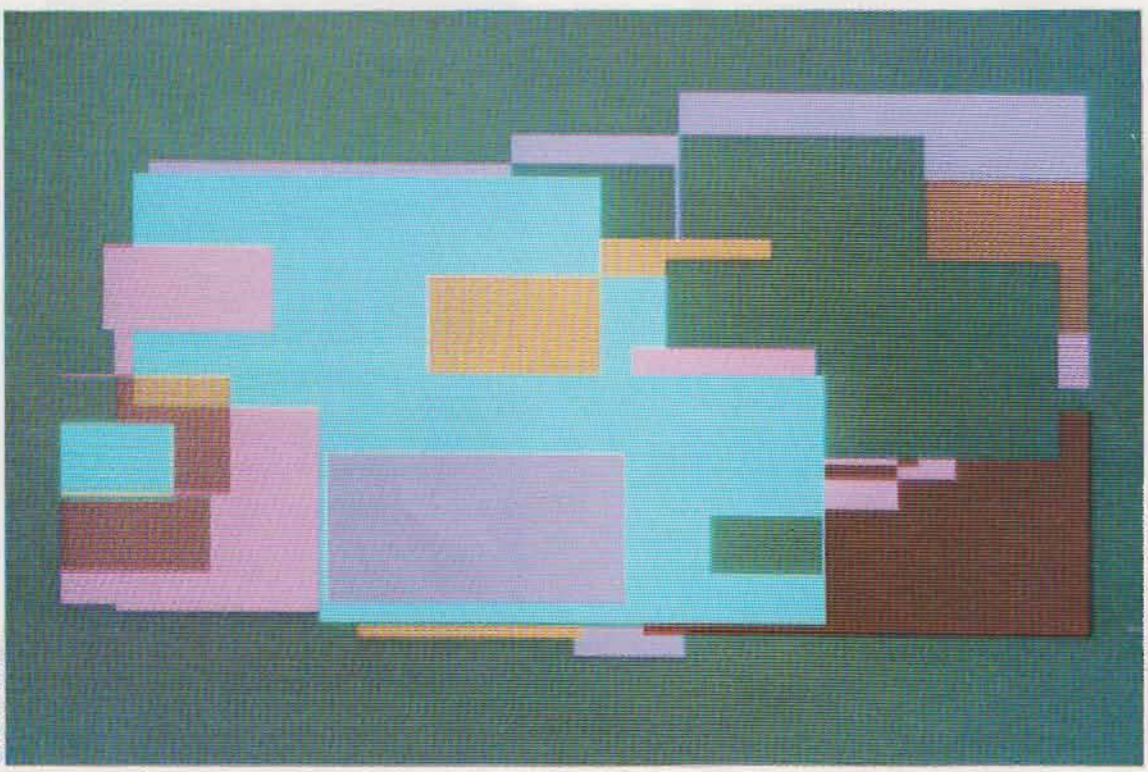
La représentation graphique d'une fonction générale exige plus de précautions que pour une simple droite.
En effet, on doit prévoir les points suivants :

— calcul du facteur d'échelle et sa normalisation
— contrôle et corrections d'erreurs
— choix des symboles par l'usager

Le facteur d'échelle se calcule très simplement comme le quotient entre les dimensions de l'écran (selon les deux axes) et les intervalles de valeurs correspondants.
Ainsi, pour représenter une fonction qui prend des valeurs comprises entre 20 et 70 sur un écran de 200 points sur l'axe Y, le facteur d'échelle sera 200/(70 -20)=200/50=4.
A partir de cette valeur, on calcule la coordonnée Y0 de l'origine qui doit coïncider avec la valeur minimale de la Y(20), plus une marge qui permettra d'éviter que les axes soient dessinés trop près du bord de l'écran.

En général, le résultat du quotient n'est pas un nombre entier. Dans l'exemple précédent, il suffit que les valeurs prises par la fonction varient entre 20 et 71 pour fournir un facteur d'échelle avec des valeurs décimales, donc difficile à interpréter.
Pour pallier cet inconvénient, on prévoit une routine de normalisation qui sélectionne la valeur la mieux adaptée parmi d'autres valeurs définies par l'utilisateur. Dans le cas précédent (valeurs 20 et 71), le facteur d'échelle 200/51=3,9 doit être arrondi à 3. Le résultat est un graphique de dimensions plus réduites (on n'exploite pas tout l'écran) mais plus facile à interpréter.
La fonction prend des valeurs soit négatives, soit positives : il convient par conséquent d'ajuster la position de l'origine (X0, Y0) de manière à permettre l'exploitation optimale de l'écran. La fonction peut, en outre, avoir une forme telle qu'elle fournit des valeurs très petites ou très grandes, supérieures aux limites de présentation. On résout ce problème en introduisant un facteur multiplicateur égal à une puissance de 10.

Visualisation de rectangles superposés sur écran couleur.



Centro THC/Marka

PRINCIPALES VARIABLES EMPLOYEES DANS LE PROGRAMME

V1	= Facteur d'échelle maximale prévu
SX(4), SY(4)	= Déplacements pour dessiner le symbole
VN/10)	= Facteurs d'échelle prévus (pour arrondis)
K$	= Indicateur d'axe (X ou Y)
XS,YS	= Coordonnées exprimées en points écran
XM,XN	= Valeurs maximales et minimales axe X
YM,YN	= Valeurs maximales et minimales de la fonction (Y)
FX,FY	= Facteurs d'échelle axe X et axe Y
DX	= Pas axe X
NP	= Nombre de points à présenter
LX,LY	= Longueur des axes
ER	= Code de la dernière erreur apparue
LOC	= Numéro de la ligne où l'erreur s'est vérifiée
K	= Puissance de 10 dans le facteur d'échelle
S1,S2	= Facteur d'échelle pour les symboles
T$	= Type de graphique (1 pour points, 2 pour segments, 3 pour symboles)
XO,YO	= Coordonnées de l'origine des axes.

Dans le cas du graphique d'une fonction variant entre 0,03 et 0,07 on introduit le facteur $K = 10^2$. Naturellement, sur le graphique il faut indiquer la valeur K utilisée ou, mieux encore, la valeur par laquelle il faut multiplier les coordonnées.

Le contrôle et la correction des erreurs sont nécessaires quand la fonction prend des valeurs indéterminées.

Ainsi, la fonction :

$$Y = \frac{1}{X-3}$$

prendra une valeur infinie au point X = 3. Un programme correct signalera cette anomalie, en évitant la valeur de l'abscisse 3. Sans ces contrôles, il résulterait une erreur de dépassement avec, pour conséquence, un arrêt de programme.

Les symboles définis par l'usager constituent un élément qui, s'il n'est pas indispensable, n'en est pas moins utile pour obtenir une présentation graphique agréable.

Dans le programme que nous présentons dans ces pages, on a prévu 3 systèmes de visualisation :

— par points
— par segments
— par symboles

Dans le premier cas, chaque point de la fonction est représenté par un point écran, et le graphique comme une succession de points plus ou moins rapprochés.

Dans la deuxième forme, chaque point est uni à celui qui le précède et qui le suit par deux segments. L'évolution de la courbe se présente alors comme une ligne brisée.

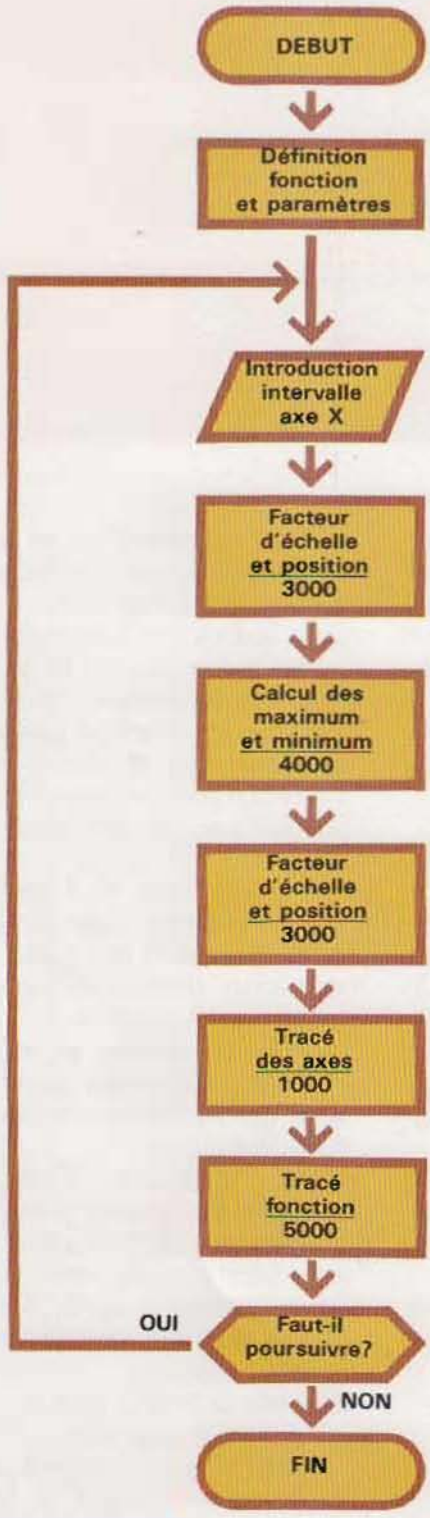
Le dernier type de représentation s'obtient en traçant un petit carré sur le point écran correspondant aux coordonnées de chaque point de la courbe.

Le premier bloc de l'organigramme (page suivante) décrit les paramètres caractéristiques de la machine et de l'application (dimensions de l'écran, facteurs d'échelle...) ainsi que la fonction à représenter. La variation de cette dernière est réalisée en remplaçant la ligne contenant la définition (ligne 100 DEF FN... A noter que ce procédé n'est autorisé qu'avec le Basic interprété).

Ensuite, le programme demande l'intervalle XM, XN des valeurs prises par la variable indépendante ; à partir de ces valeurs, il calcule le facteur d'échelle sur l'axe X (sous-programme 3000). Dans le champ des valeurs XN à XM, on effectue la recherche de la valeur maximale et minimale de la variable dépendante (4000), puis le calcul du facteur d'échelle correspondant.

Les deux dernières fonctions concernent le tracé des axes (1000) et du graphique (5000) avec les symboles sélectionnés.

ORGANIGRAMME DU PROGRAMME DE REPRESENTATION GRAPHIQUE D'UNE FONCTION



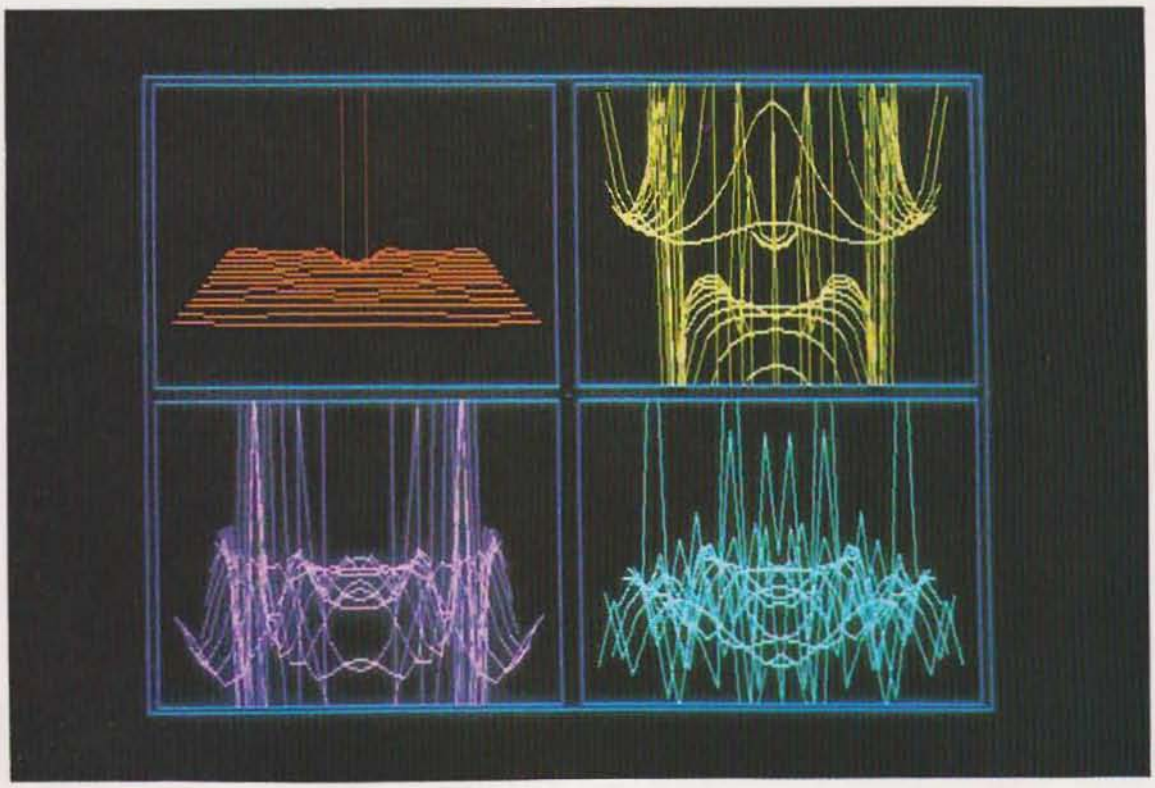
Calcul et arrondi du facteur d'échelle. Le sous-programme de calcul du facteur d'échelle (voir organigramme page suivante) utilise l'indicateur K$ pour sélectionner l'axe (X ou Y) à traiter. Il fournit, en sortie le facteur d'échelle de chacun des axes (FX, FY), les coordonnées X0, Y0 de l'origine et le facteur de multiplication K. Dans ce sous-programme, les valeurs maximale et minimale sont également arrondies en nombres entiers.
L'arrondi s'effectue par appel du sous-programme 6000.

Recherche des valeurs extrêmes de Y (sous-programme 4000) (voir p. 1468).
Le traitement consiste à calculer la valeur de Y en différents points de l'intervalle et à mémoriser, à chaque fois, la valeur la plus élevée et la valeur la plus basse.

Présentation du graphique (sous-programme 5000 et 7000). Elle est obtenue selon l'un des trois modes prévus (points, segments, symboles).
La présentation par points est du ressort du sous-programme 7000, alors que le mode segments ou symboles est exécuté par le sous-programme 5000 (l'organigramme de la page 1469 illustre cette dernière solution).
Le sous-programme 7000 est identique, à l'exclusion de l'appel à la ligne 5900, qui sert à tracer le symbole (voir page 1470).

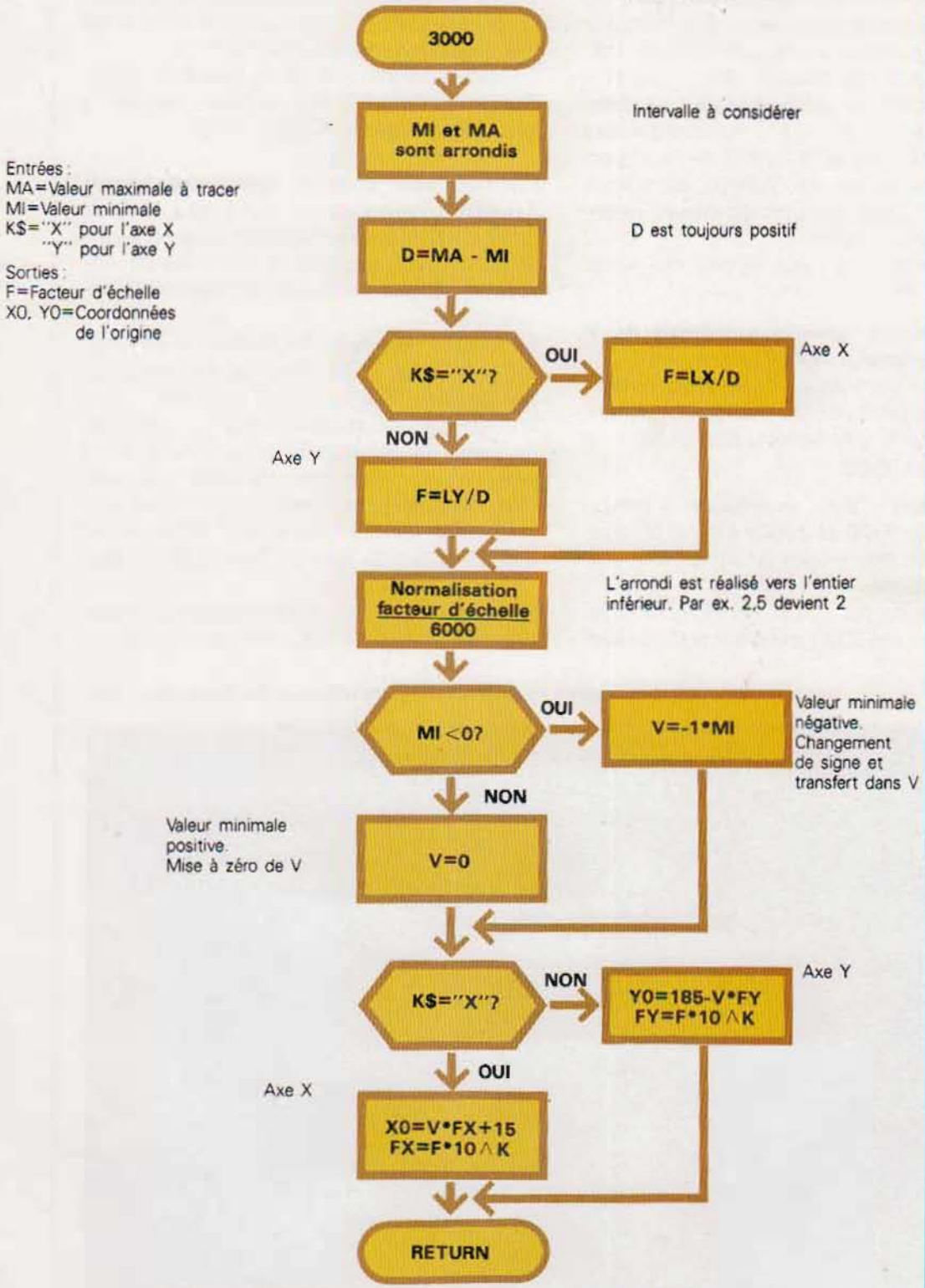
Gestion des erreurs (sous-programme 10000). L'organigramme des pages 1471 et 1472 décrit le sous-programme de gestion des erreurs. C'est la seule partie dépendant étroitement de la machine et donc difficilement généralisable.
La logique employée est propre au Personal Kid, au Siprel 2010, ainsi qu'aux Apple et compatibles. Pour les autres machines, il faut changer certains codes d'erreur, les positions de mémoire et, parfois, adopter une autre logique. Lorsqu'une erreur se produit, le code numérique correspondant est écrit dans une position de mémoire particulière (222), tandis que le numéro de ligne de l'erreur se trouve à une autre position.
Si l'on appelle ER la variable qui devra contenir le code d'erreur et LOC celle qui contiendra

Images graphiques tridimentionnelles utilisant les fenêtres vidéo.



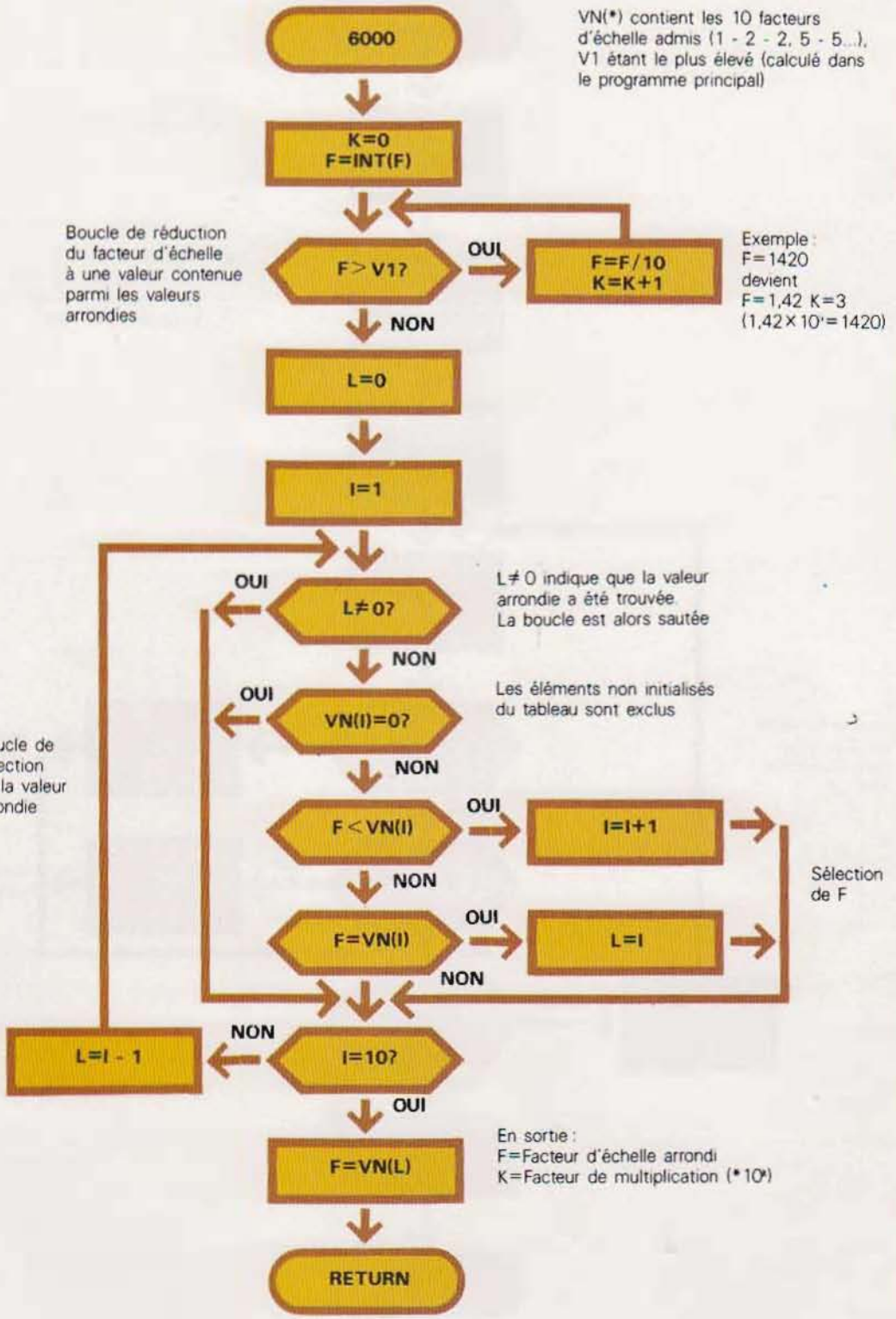
Marka

CALCUL ET ARRONDI DU FACTEUR D'ECHELLE

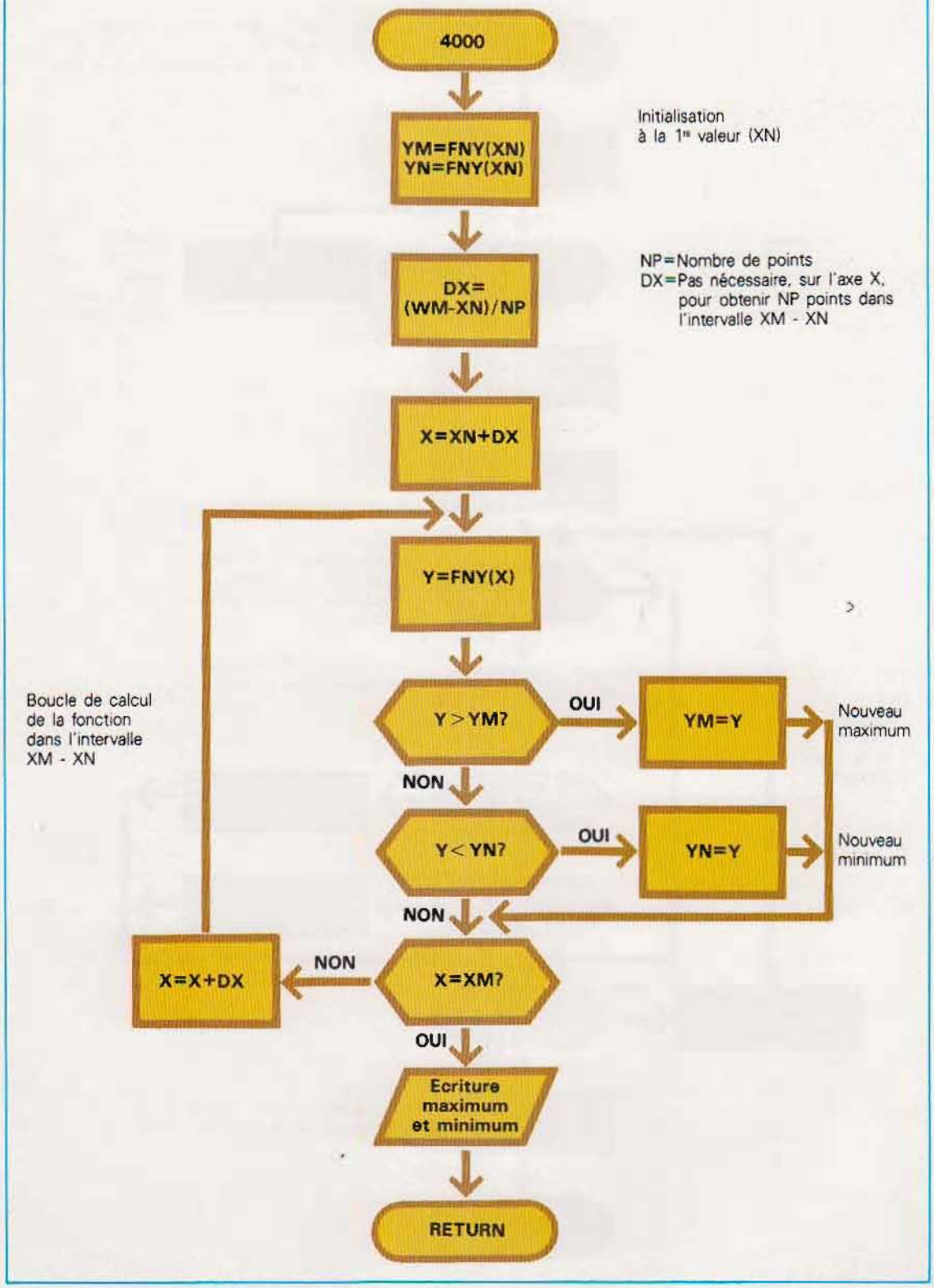


Ce sous-programme est utilisé 2 fois, une pour chaque axe.
La sélection se fait à l'aide de l'indicateur K$

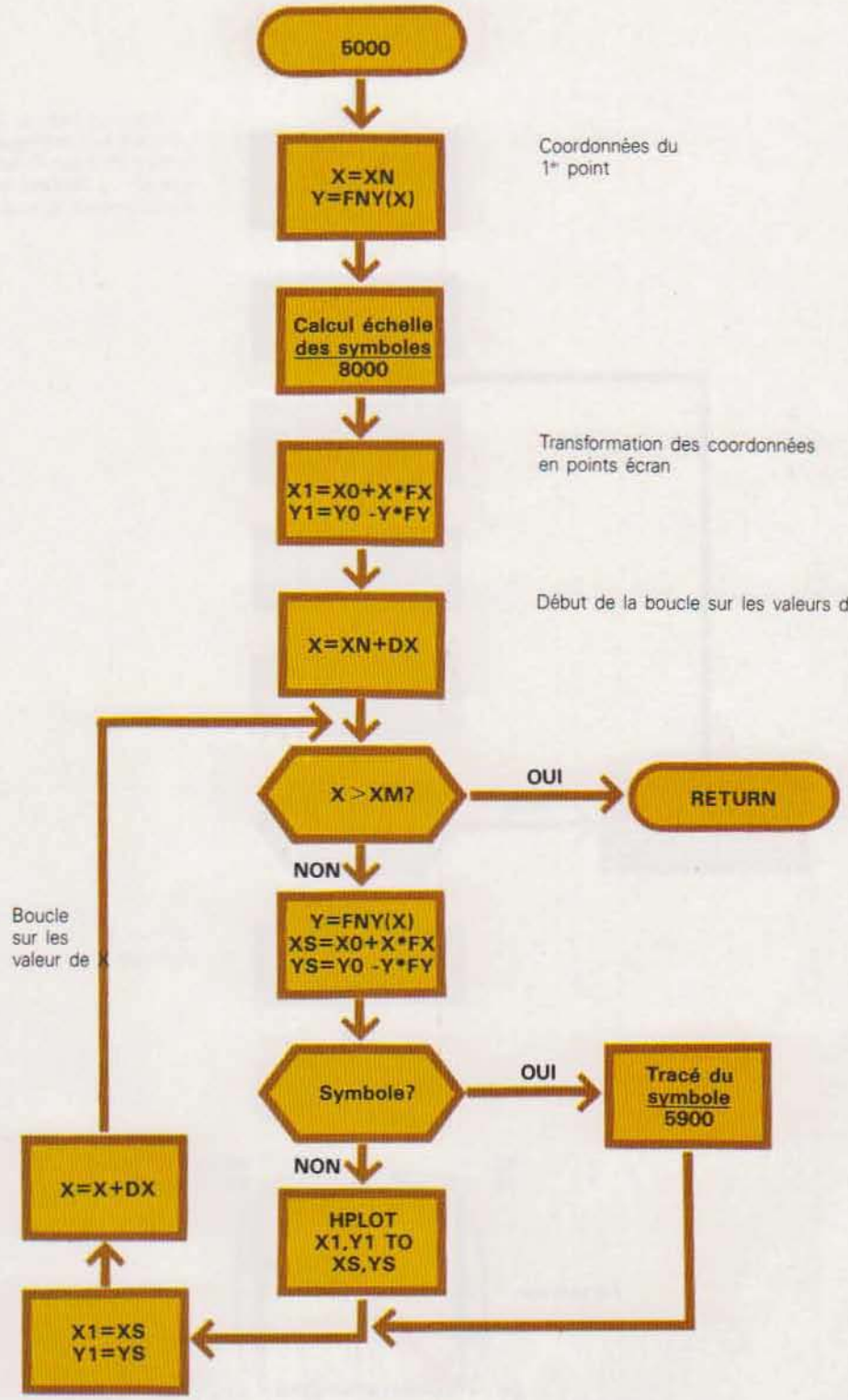
ARRONDI DU FACTEUR D'ECHELLE



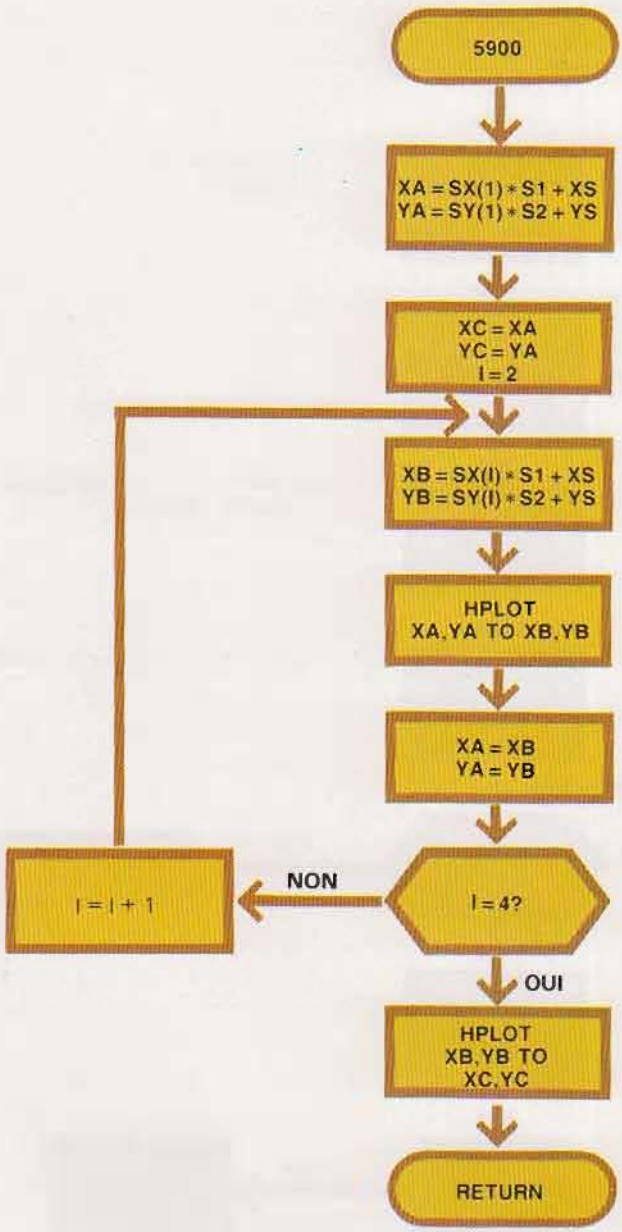
RECHERCHE DU MAXIMUM ET DU MINIMUM



PRESENTATION DE LA COURBE PAR SEGMENTS OU PAR SYMBOLES

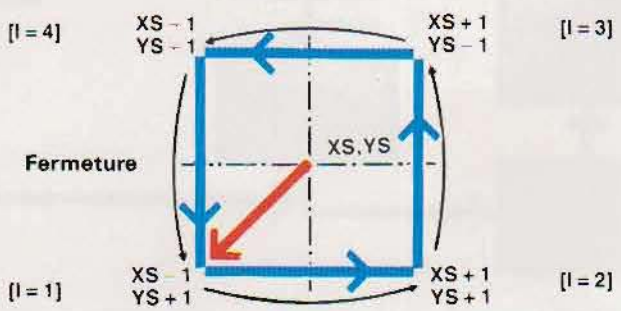


TRACE D'UN SYMBOLE CENTRE EN XS, YS

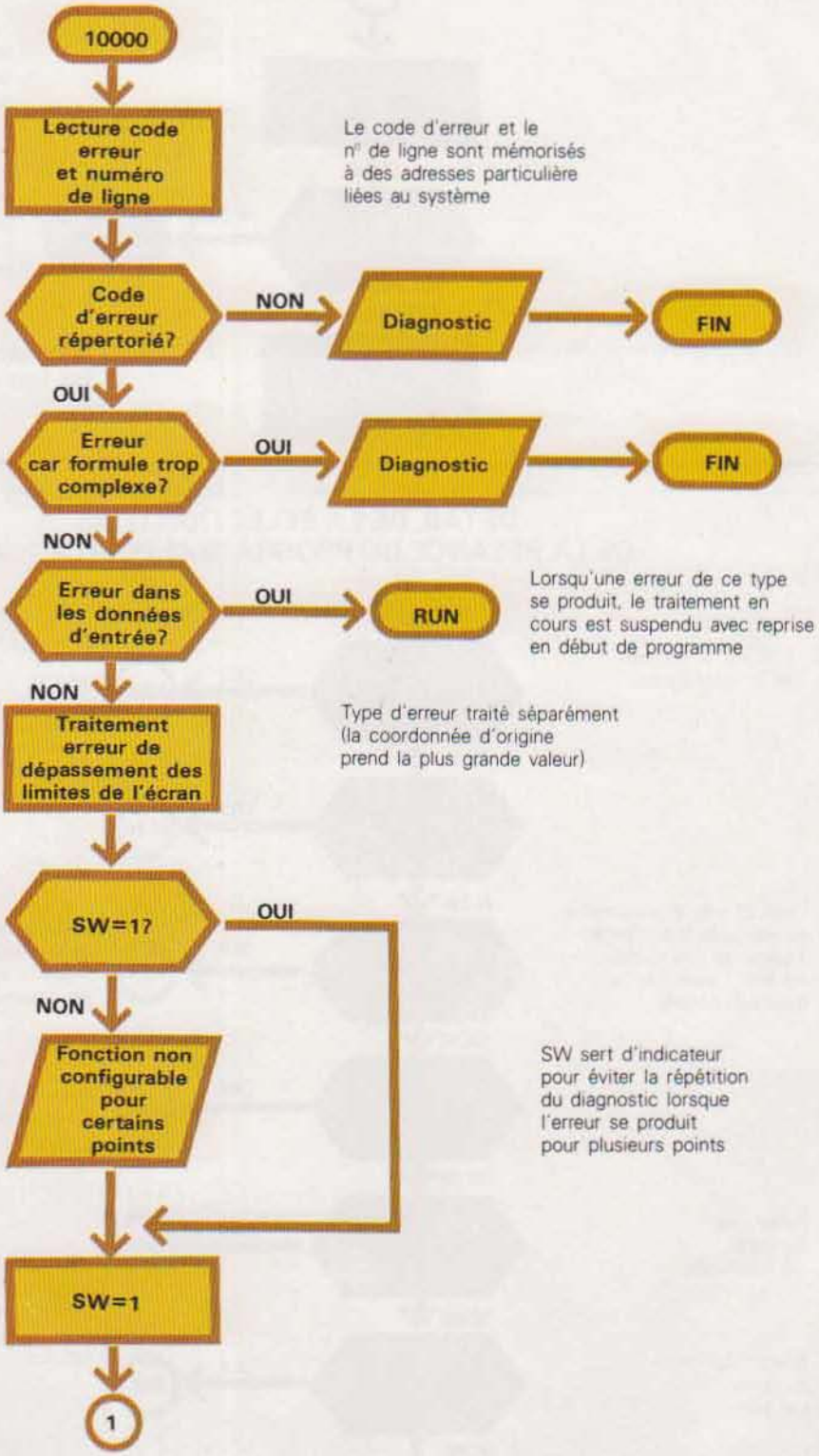


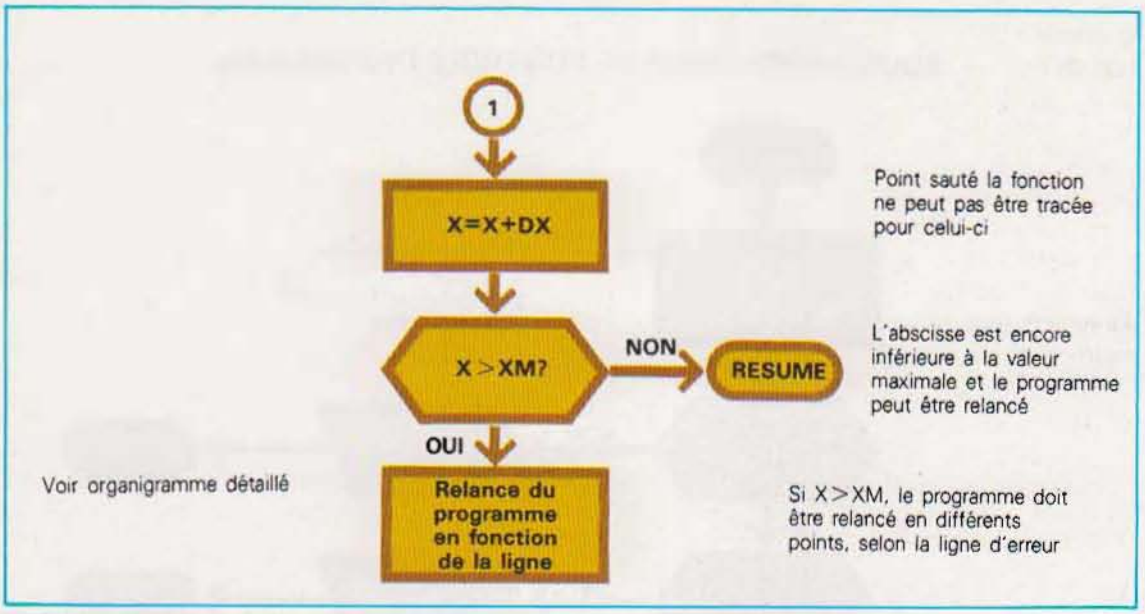
1er point. Le tableau SX ( ) contient le déplacement le long de l'axe X, tandis que SY ( ) contient les déplacements le long de Y

Clôture du symbole

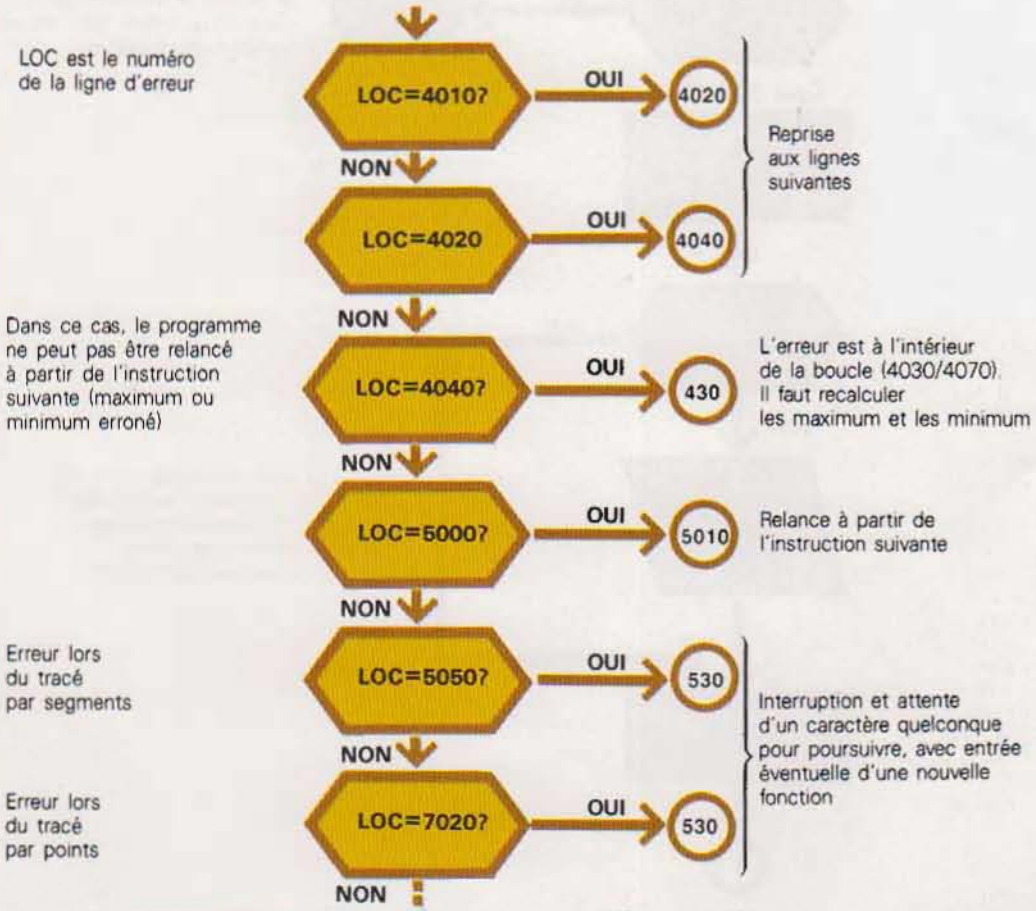


SOUS-PROGRAMME DE CONTROLE DES ERREURS





DETAIL DE LA SELECTION LORS DE LA RELANCE DU PROGRAMME POUR X > XM



le numéro de ligne, les instructions d'acquisition de ces valeurs seront :

```
ER=PEEK (222)
LOC=PEEK (218 )+PEEK (219)*256
```

La première est immédiatement compréhensible : PEEK extrait le contenu de l'adresse mémoire 222 et l'affecte à la variable ER.
Dans la seconde, la donnée (le numéro de ligne) doit être extraite de deux adresses mémoire successives (218 et 219) pour être reconstituée. En effet, l'adresse ne contient que 8 bits alors qu'il en faut 16 pour l'adressage du n° de ligne. Le système divise donc ce numéro en deux parties qu'il range à deux adresses consécutives.

La multiplication par 256 est nécessaire pour donner à la première partie le juste poids par rapport à la seconde (déplacer de 8 bits signifie multiplier par 256).
Les codes d'erreur reconnus, qui ne sont valables que pour la machine spécifiée, sont les suivants :

53=Quantité incorrecte
133=Division par zéro
254=Erreur d'entrée
69=Débordement
191=Formule trop complexe.

Pour ce qui concerne les autres machines, si la gestion des erreurs est proche de celle décrite, il convient de vérifier la correspondance des codes et, éventuellement, de les modifier.

TRACE D'UNE FONCTION

```
 90 REM ----------------------------
 91 REM FONCTIONS MATHEMATIQUES
 92 REM ----------------------------
 93 REM
 94 REM
 95 REM
 96 REM
 97 REM ----------------
 98 REM INITIALISATION
 99 REM ----------------
100 DEF FNY(X)=X*SIN(X)
110 LX=250
    : LY=180
    : SW=0
120 FOR J=1 TO 4
    : READ SX(J),SY(J)
    : NEXT J
130 DATA -1,1,1,1,1,-1,-1,-1
140 ONERR GOTO 10000
150 V1=1
160 FOR J=1 TO 10
    : READ VN(J)
    : IF NV(J)>1 THEN V1=VN(J)
170 NEXT J
180 DATA 1,2,3,4,5,6,7,8,9,0
190 REM ------
200 REM MENU
210 REM ------
220 TEXT
    : HOME
    : GOSUB 2000
230 HTAB 3
    : VTAB 5
    : INPUT "INTERVALLE AXE X? " ; XN,XM
    : HTAB 39
    : VTAB 5
    : PRINT "*"
240 IF XM<=XN THEN 220
250 IF XM-XN>250 THEN 220
260 HTAB 3
    : VTAB 10
    : INPUT "NOMBRE DE POINTS? " ; NP
    : HTAB 39
    : VTAB 10
    : PRINT "*"
```

```
270 HTAB 3
    : VTAB 15
    : PRINT "TYPE DE GRAPHIQUE? "
280 HTAB 3
    : PRINT "1) PAR POINTS"
290 HTAB 3
    : PRINT "2) PAR SEGMENTS"
300 HTAB 3
    : PRINT "3) PAR SYMBOLES"
310 VTAB 15
    : HTAB 30
    : INPUT " " ; T$
    : VTAB 15
    : HTAB 39
    : PRINT "*"
320 GOSUB 9000
330 REM ----------------------
340 REM PROGRAMME PRINCIPAL
350 REM ----------------------
360 IF XN<0 AND XM<=0 THEN MI=XN
    : MA=0
370 IF XN<0 AND XM>0 THEN MI=XN
    : MA=XM
380 IF XN>=0 AND XM>0 THEN MI=0
    : MA=XM
390 IF MA>250 THEN MA=250
400 IF MI<-250 THEN MI=-250
410 K$="X"
    : GOSUB 3000
420 GOSUB 4000
430 IF YN<0 AND YM<=0 THEN MI=YN
    : MA=0
440 IF YN<0 AND YM>0 THEN MI=YN
    : MA=YM
450 IF YN>=0 AND YM>0 THEN MI=0
    : MA=YM
460 IF MA>250 THEN MA=250
470 IF MI<-250 THEN MI=-250
480 K$="Y"
    : GOSUB 3000
490 HGR2
    : HCOLOR=3
500 GOSUB 1000
510 IF T$="1" THEN GOSUB 7000
    : GOTO 530
520 GOSUB 5000
530 PRINT CHR$(7)
    : GET A$
540 TEXT
    : HOME
    : VTAB 10
    : INPUT "VOULEZ-VOUS CHANGER DE FONCTION? (O/N)" ; A$
550 IF LEFT$(A$,1)="N" THEN RUN
560 PRINT
570 INVERSE
    : PRINT "REMPLACER LA FONCTION DE LA LIGNE 100 ET FAIRE <RUN>"
    : NORMAL
580 LIST 100
    : END
997 REM ------------------
998 REM DESSIN DES AXES
```

```
999 REM ------------------
1000 HPLOT 0,0 TO 279,0 TO 279,191 TO 0,191 TO 0,0
1010 IF D>191 THEN Y0=96
1020 HPLOT 15,Y0 TO 265,Y0
     : HPLOT X0,185 TO X0,5
1030 N2=5
     : FOR N1=2 TO 0 STEP -1
     : HPLOT X0-N1,N2 TO X0+N1,N2
     : N2=N2-1
     : NEXT N1
1040 N2=265
     : FOR N1=2 TO 0 STEP -1
     : HPLOT N2,Y0-N1 TO N2,Y0+N1
     : N2=N2+1
     : NEXT N1
1050 HPLOT 271,Y0-2 TO 276,Y0+3
     : HPLOT 276,Y0-2 TO 271,Y0+3
1060 HPLOT X0-10,3 TO X0-7,6
     : HPLOT X0-4,3 TO X0-10,8
1070 IF FX=1 THEN 1140
1080 FOR G=X0 TO 15 STEP -FX
1090 HPLOT G,Y0+1 TO G,Y0-1
1100 NEXT G
1110 FOR G=X0 TO 256 STEP FX
1120 HPLOT G,Y0+1 TO G,Y0-1
1130 NEXT G
1140 IF FY=1 THEN 1210
1150 FOR G=Y0 TO 5 STEP -FY
1160 HPLOT X0+1,G TO X0-1,G
1170 NEXT G
1180 FOR G=Y0 TO 185 STEP FY
1190 HPLOT X0+1,G TO X0-1,G
1200 NEXT G
1210 RETURN
1997 REM ---------------
1998 REM PRESENTATION
1999 REM ---------------
2000 A$="*"
2010 FOR J=1 TO 23
2020 HTAB 1
     : VTAB J
     : PRINT A$
2030 NEXT J
2040 FOR J=2 TO 39
2050 HTAB J
     : VTAB 23
     : PRINT A$
2060 NEXT J
2070 FOR J=23 TO 1 STEP -1
2080 HTAB 39
     : VTAB J
     : PRINT A$
2090 NEXT J
2100 FOR J=39 TO 2 STEP -1
2110 HTAB J
     : VTAB 1
     : PRINT A$
2120 NETX J
2130 VTAB 1
     : HTAB 10
     : INVERSE
     : PRINT "FONCTIONS MATHEMATIQUES"
     : NORMAL
2140 RETURN
2997 REM ---------
2998 REM ARRONDI
2999 REM ---------
3000 MA%=INT(MA)
     : MI%=INT(MI)
```

```
3010 IF MA>0 AND MA%<>(MA) THEN MA=MA%+1
     : GOTO 3030
3020 MA=MA%
3030 IF MI>0 AND MI%<>MI THEN MI=MI%+1
     : GOTO 3050
3040 MI=MI%
3047 REM --------------------------------
3048 REM CALCUL DU FACTEUR D'ECHELLE
3049 REM --------------------------------
3050 D=MA-MI
3060 IF K$="X" THEN F=LX/D
     : GOTO 3080
3070 F=LY/D
3080 GOSUB 6000
     : REM ARRONDI DU FACTEUR D'ECHELLE
3090 IF MI<0 THEN V=-MI
     : GOTO 3110
3100 V=0
3110 IF K$="X" THEN FX=F*10^K
     : X0=U*FX+15
     : RETURN
3120 FY=F*10^K
     : Y0=185-V*FY
     : RETURN
3997 REM ------------------------
3998 REM MAXIMUM ET MINIMUM
3999 REM ------------------------
4000 DX=(XM-XN)/NP
     : X=XN
4010 YN=FNY(X)
4020 YM=FNY(X)
4030 FOR X=XN+DX TO XM STEP DX
4040 Y=FNY(X)
4050 IF Y>YM THEN YM=Y
4060 IF Y<YN THEN YN=Y
4070 NEXT X
4080 REM ECRITURE DU MAXIMUM ET DU MINIMUM
4090 RETURN
4997 REM ----------------------------
4998 REM GRAPHIQUE PAR SEGMENTS
4999 REM ----------------------------
5000 X=XN
     : Y=FNY(X)
5010 GOSUB 8000
5020 X1=X0+X*FX
5030 Y1=Y0-Y*FY
5040 FOR X=XN+DX TO XM STEP DX
5050 Y=FNY(X)
5060 XS=X0+X*FX
5070 YS=Y0-Y*FY
5080 IF T$="3" THEN GOSUB 5900
     : GOTO 5100
5090 HPLOT X1,Y1 TO XS,YS
5100 X1=XS
     : Y1=YS
5110 NEXT X
5120 RETURN
5897 REM ------------------------
5898 REM DESSIN DES SYMBOLES
5899 REM ------------------------
5900 XA=SX(1)*S1+XS
     : YA=SY(1)*S2+YS
5910 XC=XA
     : YC=YA
5920 FOR I=2 TO 4
5930 XB=SX(I)*S1+XS
     : YB=SY(I)*S2+YS
```

```
5940 HPLOT XA,YA TO XB,YB
5950 XA=XB
     : YA=YB
5960 NEXT I
5980 HPLOT XB,YB TO XC,YC
5990 RETURN
5997 REM ---------
5998 REM ARRONDI
5999 REM ---------
6000 K=0
     : F=INT(F)
6010 IF F>V1 THEN F=F/10
     : K=K+1
     : GOTO 6010
6020 L=0
6030 FOR I=1 TO 10
6040 IF L<>0 OR VM(I)=0 THEN 6070
6050 IF F<VM(I) THEN L=I-1
6060 IF F=VM(I) THEN L=I
6070 NEXT I
6080 F=VM(L)
6090 RETURN
6997 REM ------------------------
6998 REM GRAPHIQUE PAR POINTS
6999 REM ------------------------
7000 GOSUB 8000
7010 FOR X=XM TO XM STEP DX
7020 Y=FNY(X)
7030 XS=X0+X*FX
7040 YS=Y0-Y*FY
7050 HPLOT XS,YS
7060 NEXT X
7070 RETURN
7997 REM -------------------------
7998 REM ECHELLE DES SYMBOLES
7999 REM -------------------------
8000 S1=INT(FX/4)
     : IF S1<1 THEN S1=1
8010 S2=INT(FY/4)
     : IF S2<1 THEN S2=1
8020 IF S1<S2 THEN S2=S1
     : RETURN
8030 S1=S2
8040 RETURN
8997 REM ------
8998 REM PAUSE
8999 REM ------
9000 HOME
     : HTAB 12
     : VTAB 12
     : FLASH
     : PRINT "UN INSTANT, SVP"
     : NORMAL
     : RETURN
9997 REM -------------------------
9998 REM GESTION DES ERREURS
9999 REM -------------------------
10000 ER=PEEK(222)
     : LOC=PEEK(218)+PEEK(219)*256
10010 IF ER<>53 AND ER<>133 AND ER<>254 AND ER<>69 AND ER<>191 THEN HOME
     : PRINT "ERREUR N° " ; ER
     : END
10020 IF ER=191 THEN HOME
     : HTAB 12
     : VTAB 12
     : INVERSE
```

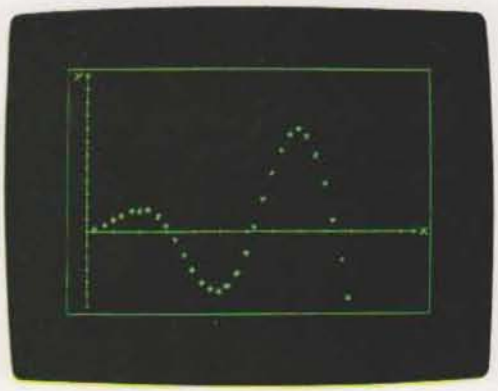
```
      : PRINT "FORMULE TROP COMPLEXE"
      : NORMAL
      : STOP
10030 IF ER=254 THEN RUN
10040 REM GRANDEUR INCORRECTE
10050 IF XS>279 THEN XS=279
      : RESUME
10060 IF XS<0 THEN XS=0
      : RESUME
10070 IF YS>191 THEN YS=191
      : RESUME
10080 IF YS<0 THEN YS=0
      : RESUME
10090 IF XA>279 THEN XA=279
      : RESUME
10100 IF XA<0 THEN XA=0
      : RESUME
10110 IF YA>191 THEN YA=191
      : RESUME
10120 IF YA<0 THEN YA=0
      : RESUME
10130 IF XB>279 THEN XB=279
      : RESUME
10140 IF XB<0 THEN XB=0
      : RESUME
10150 IF YB>191 THEN YB=191
      : RESUME
10160 IF YB<0 THEN YB=0
      : RESUME
10170 IF X1>279 THEN X1=279
      : RESUME
10180 IF X1<0 THEN X1=0
      : RESUME
10190 IF Y1>191 THEN Y1=191
      : RESUME
10200 IF Y1<0 THEN Y1=0
      : RESUME
10210 IF XC>279 THEN XC=279
      : RESUME
10220 IF XC<0 THEN XC=0
      : RESUME
10230 IF YC>191 THEN YC=191
      : RESUME
10240 IF YC<0 THEN YC=0
      : RESUME
10250 IF SW=1 THEN 10300
10260 TEXT
      : HOME
      : VTAB 12
      : INVERSE
      : PRINT "LA FONCTION N'EST DEFINIE EN AUCUN POINT DE L'INTERVALLE"
      : NORMAL
10270 VTAB 23
      : PRINT "APPUYEZ SUR UNE TOUCHE POUR POURSUIVRE"
      : GET A$
10280 GOSUB 9000
10290 SW=1
10300 X=X+DX
      : IF X>XM THEN 10320
10310 RESUME
10320 IF LOC=4010 THEN 4020
10330 IF LOC=4020 THEN 4030
10340 IF LOC=4040 THEN 430
10350 IF LOC=5000 THEN 5010
10360 IF LOC=5050 THEN 530
10370 IF LOC=7020 THEN 530
```

VISUALISATION DU GRAPHIQUE D'UNE FONCTION

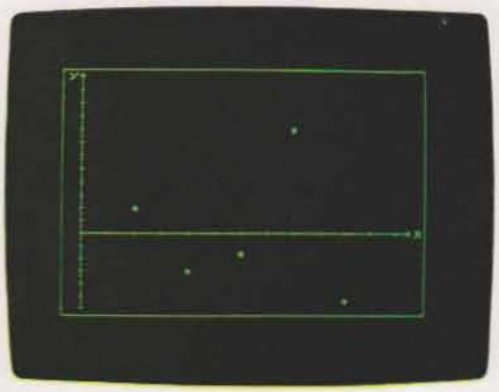
Une fois l'exécution du programme demandée et la partie des initialisations (ligne 100 à 180) terminée, le sous-programme 2000 (ligne 220) est appelé. Celui-ci présente un cadre entouré d'astérisques et le titre en vidéo inversée. En fin de sous-programme 2000, on revient au programme principal qui présente, dans l'ordre, les différentes possibilités proposées. La photographie montre la situation finale, une fois qu'il a été répondu à toutes les questions et avant la sélection du type de graphique. Les données entrées se rapportent à la fonction X*SIN(X) (ligne 100). Les écrans suivants montrent, pour la même fonction, les différents aspects du graphique en fonction des paramètres NOMBRE DE POINTS et TYPE DE GRAPHIQUE.



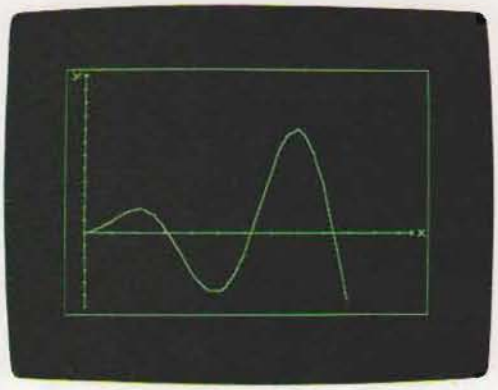
Le graphique par points (type de graphique 1) est le mode sélectionné. Le programme présente les axes cartésiens et les points correspondant aux 30 couples de coordonnées. La valeur de X est entrée au clavier tandis que Y est calculée par le programme.



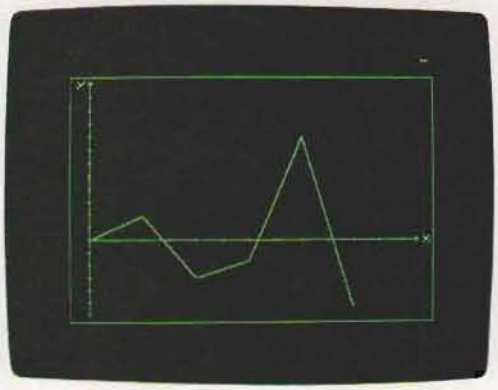
Dans ce cas précis, l'utilisateur a demandé la visualisation du graphique par points pour un nombre de valeurs de la variable X réduit (5). La courbe de la fonction est toujours la même, mais elle est moins nette.



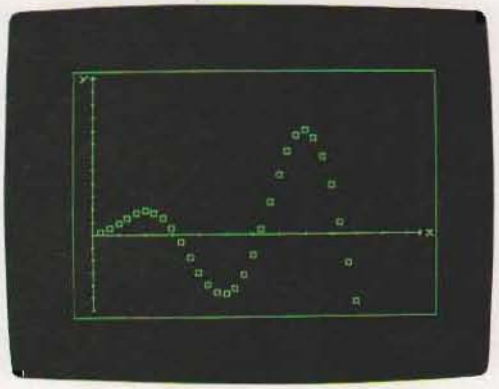
Visualisation par segments de 30 valeurs de la variable X. La définition de la courbe est excellente en raison du grand nombre de données. Le programme a tracé une série de segments joignant deux à deux les points dont les coordonnées sont connues.



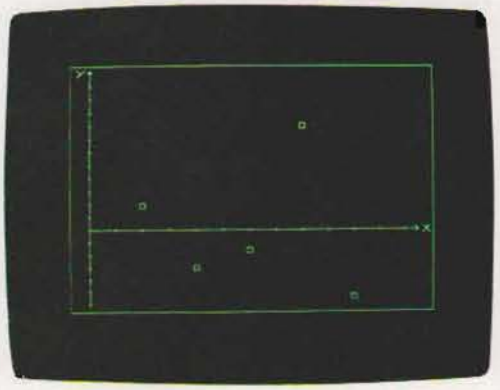
La visualisation se poursuit par segments, mais avec 5 valeurs d'entrée de X. La définition du graphique est très approximative et les segments sont visibles.



L'utilisateur a demandé la visualisation par symboles de 30 valeurs de X. Cette représentation est plus indiquée quand il s'agit de visualiser l'allure d'une grandeur sujette à des erreurs d'approximation. La grandeur du symbole utilisé peut, dans ce cas, couvrir ces erreurs.



Visualisation par symboles pour 5 valeurs de la variable indépendante. L'allure de la courbe est à peine perceptible.



Une fois le graphique visualisé, il suffit d'activer une touche quelconque (ligne 540) pour que le programme fasse évoluer la fonction. Noter la ligne 580, qui permet de présenter le contenu de la ligne 100 et, par suite, de rappeler la fonction traitée. La ligne 580 est suivie de l'instruction END qui met fin au programme. Le système revient alors à l'état commandes (cet état est mis en évidence par l'affichage du caractère guide-utilisateur) et l'utilisateur peut changer la fonction en réécrivant la ligne 100. Pour finir, la commande RUN lance l'exécution du graphique de la nouvelle fonction X/(X-2).



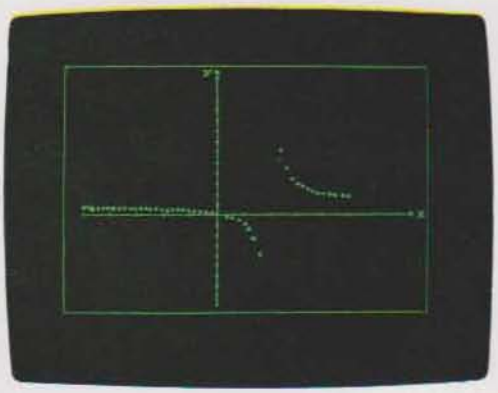
Le programme présente à nouveau le cadre dans lequel il demande à l'utilisateur de luir fournir les paramètres nécessaires à l'exécution.



Une fois le type de graphique sélectionné, le programme passe à la phase de calcul, au cours de laquelle il découvre que la fonction contient un point de discontinuité pour X=2. En effet, pour cette valeur, on a Y=2/(2-2)=2/0.
Le quotient 2/0 donne un nombre infini, qui ne peut être représenté dans le graphique. Le programme signale cette anomalie (il ne s'agit pas d'une erreur mais d'une caractéristique de la fonction).
Le diagnostic qui apparaît est très général et peut même parfois être inexact d'un point de vue mathématique. Ce défaut est dû à la structure du sous-programme d'erreur qui regroupe plusieurs causes en un seul message.



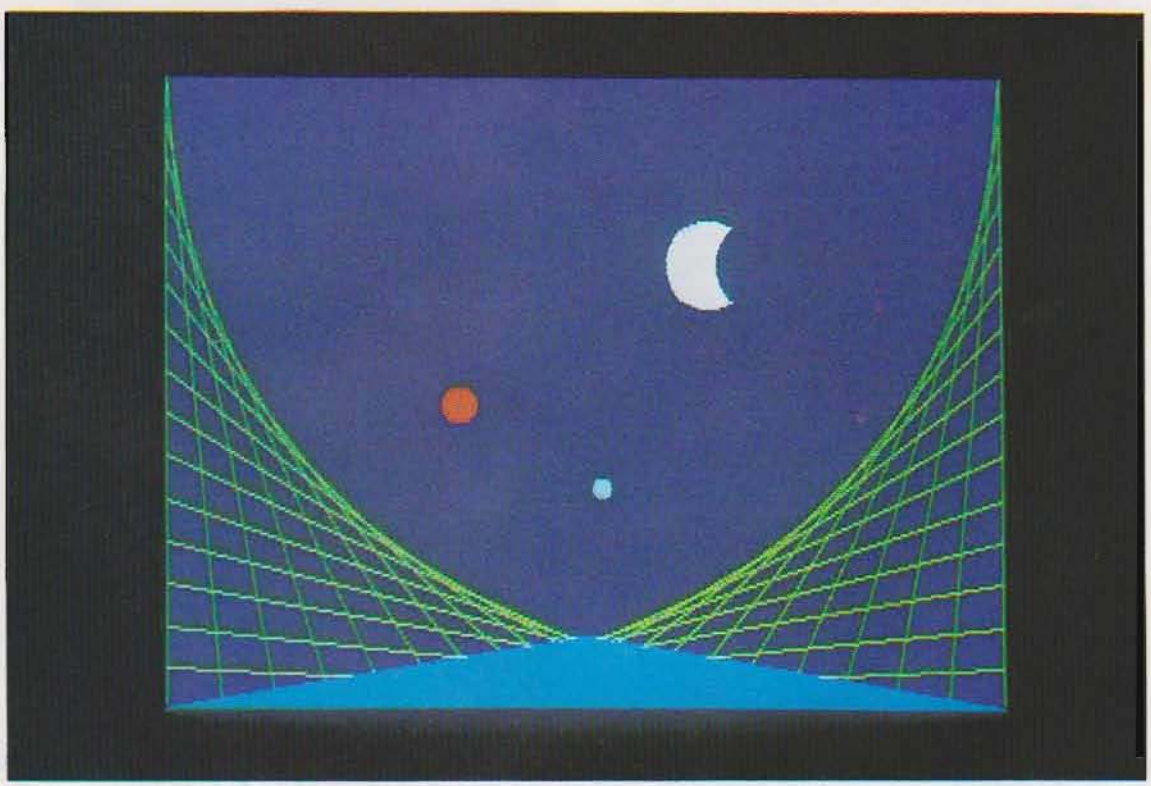
La fonction est tracée une fois que l'utilisateur a confirmé son choix en appuyant sur une touche quelconque (ligne 10250 et suivantes). La présence d'une asymptote pour X=2 (point de discontinuité de la fonction) est indiquée.
La définition de la courbe est améliorée en demandant sa visualisation par segments.



Visualisation des caractères en mode graphique

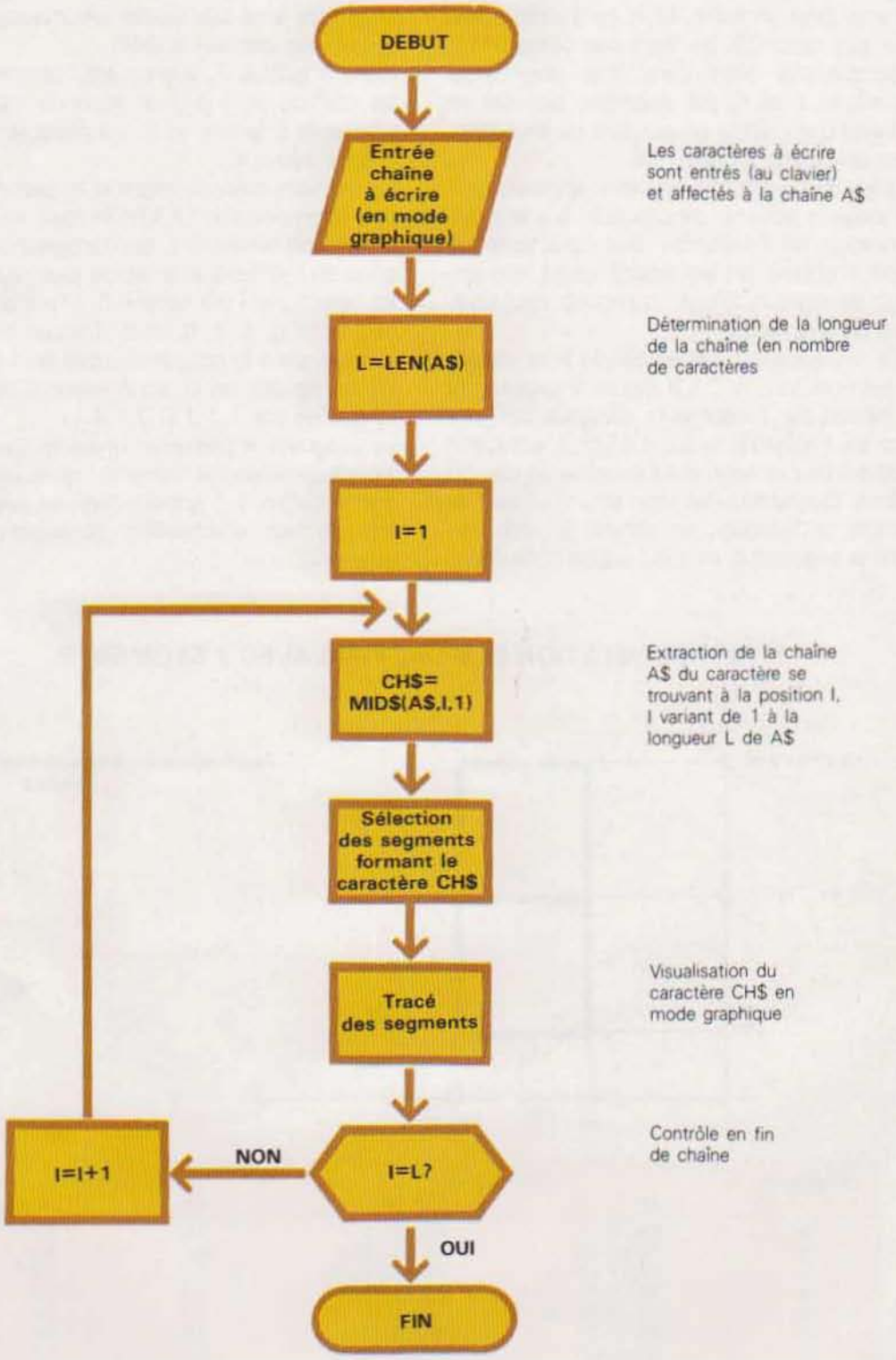
Les applications graphiques demandent souvent, elles aussi, l'insertion de caractères alphabétiques ou numériques dans l'image vidéo. C'est le cas, par exemple, des légendes explicatives associées à un graphique.
Si le moniteur haute résolution est conçu pour gérer séparément chaque point de l'écran, il n'est plus possible de visualiser les caractères ASCII standard utilisés en mode texte. En effet, dans ce dernier mode, l'écran est géré par matrices de caractères : il est donc possible d'y placer un caractère par « case » se trouvant au croisement d'une ligne et d'une colonne. Une case est constituée d'une matrice de points écran qui sont automatiquement activés par le système, en fonction du caractère à visualiser.
En mode graphique, la correspondance entre le caractère et la configuration de points n'existe plus, précisément pour autoriser cette gestion individuelle de tous les points écran.
Le terminal vidéo visualisera donc exclusivement des textes (mode texte) ou des graphiques (mode graphique), sans possibilité de coexistence. Toutefois, une forme de mixité a été mise au point dans certaines machines, réservant quelques lignes (généralement en bas de l'écran) au mode texte, le reste étant affecté au mode graphique.
La visualisation d'un caractère alphabétique ou numérique dans un page graphique nécessite une construction logicielle point par point de ce caractère. La plupart des ordinateurs personnels et des micros ne sont pas dotés de sous-programmes adéquats, laissés au bon soin de l'utilisateur.
La technique de construction consiste à représenter les lettres et les nombres sous des formes schématisables au moyen des déplacements élémentaires d'un crayon imaginaire. Pour obtenir un « tracé » quelconque en mode graphique, il faut lancer le programme correspondant lettre par lettre, ou nombre par nombre (voir organigramme ci-contre).
Le programme ne présente aucune difficulté majeure en-dehors de la sélection des segments composant chacun des caractères. Une

Image graphique obtenue en utilisant des fonctions mathématiques.



Marka

AFFICHAGE DE CARACTERES EN MODE GRAPHIQUE



méthode consisterait à écrire un sous-programme pour chaque caractère : une routine pour écrire la lettre « a », une autre pour « b » et ainsi de suite. Mais ce système, qui n'est pas optimisé, ne tient pas compte des ressemblances entre caractères graphiques. Les lettres F et E, par exemple, peuvent se déduire l'une l'autre, en ajoutant ou en supprimant un segment horizontal.

Pour les autres caractères moins apparentés, il est toujours possible de procéder à une schématisation de l'ensemble des caractères de façon à obtenir un équivalent assez ressemblant de chacun d'eux, composé exclusivement de segments.

Cette méthode est très employée pour visualiser les nombres de 0 à 9 (figure 1 ci-dessous) au moyen de 7 segments désignés par une lettre de l'alphabet (a,b,c,d,e,f,g). L'activation sélective de ces segments visualise un des 10 chiffres. Quand tous les segments sont visibles (comme ci-dessous), on obtient 8 ; en éteignant le segment g, on a 0 ; supprimons encore les segments f, a, b et c et 1 s'affiche, et ainsi de suite.

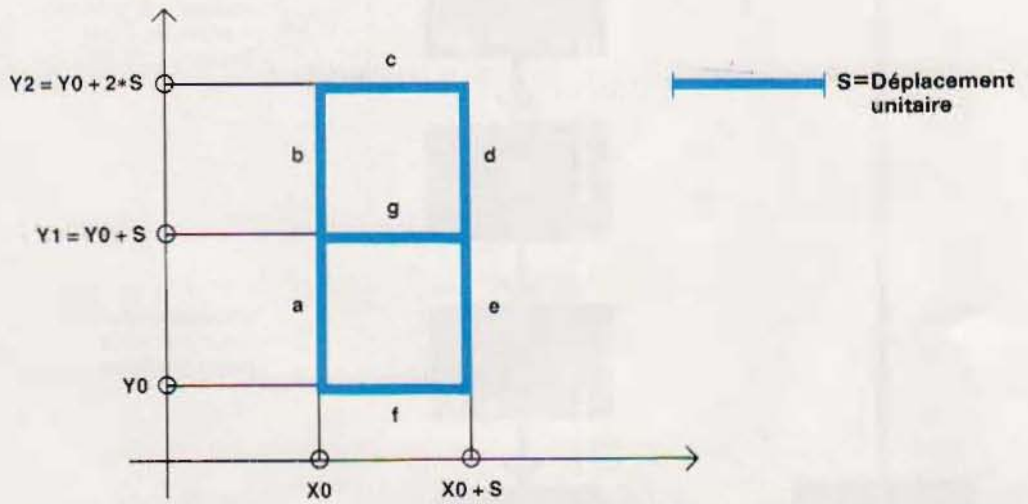
Dans les systèmes d'affichage numériques, les segments sont des diodes électroluminescentes ou des cristaux liquides.

Dans le tableau 2, la première colonne indique les chiffres de 1 à 0, la seconde montre les segments à activer et la troisième le symbole affiché (résultat).

La dernière colonne reprend la seconde, non pas selon notation OUI/NON mais en utilisant la notation binaire 0/1, qui occupe moins d'espace en mémoire et effectue plus rapidement les sélections. Pour obtenir 6, il faut activer les segments g, f, e, b, et a. Chacun d'eux est indiqué, dans le programme par un 1 (OUI), et les autres par un 0. Le nombre 6 sera ainsi symbolisé par 1 1 1 0 0 1 1.

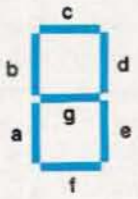
Le programme passe en revue le contenu du tableau position par position ; quand il rencontre le chiffre 1 il appelle l'un des sept sous-programmes d'activation correspondant au segment.

1/REPRESENTATION DES CHIFFRES AVEC 7 SEGMENTS



Segment	Coordonnées de	Coordonnées à	Incréments de	Incréments à
a	X0,Y0	X0,Y0 + S	0,0	0,1
b	X0,Y0 + S	X0,Y0 + 2 * S	0,1	0,2
c	X0,Y0 + 2 * S	X0 + S,Y0 + 2 * S	0,2	1,2
d	X0 + S,Y0 + 2 * S	X0 + S,Y0 + S	1,2	1,1
e	X0 + S,Y0 + S	X0 + S,Y0	1,1	1,0
f	X0 + S,Y0	X0,Y0	1,0	0,0
g	X0,Y0 + S	X0 + S,Y0 + S	0,1	1,1

2/REPRESENTATION DES CHIFFRES AVEC 7 SEGMENTS



Chiffre à représenter	Segment activé 7 g	6 f	5 e	4 d	3 c	2 b	1 a	Symbole tracé	Contenu du tableau
1	—	—	OUI	OUI	—	—	—		0011000
2	OUI	OUI	—	OUI	OUI	—	OUI		1101101
3	OUI	OUI	OUI	OUI	OUI	—	—		1111100
4	OUI	—	OUI	—	—	OUI	—		1010010
5	OUI	OUI	OUI	—	OUI	OUI	—		1110110
6	OUI	OUI	OUI	—	—	OUI	OUI		1110011
7	—	—	OUI	OUI	OUI	—	—		0011100
8	OUI	OUI	OUI	OUI	OUI	OUI	OUI		1111111
9	OUI	—	OUI	OUI	OUI	OUI	—		1011110
0	—	OUI	OUI	OUI	OUI	OUI	OUI		0111111

En réalité, il n'est pas nécessaire d'en avoir sept différents, un seul, correctement paramétré, est suffisant.
La seconde colonne du tableau 1 indique les déplacements nécessaires par rapport au point d'origine (X0, Y0) pour obtenir chacun des segments. Ainsi, le segment est tracé à l'aide du déplacement de X0, Y0 vers X0, Y0 + S (S étant la longueur de chaque segment). Ce système (déplacement unitaire S) permet deux paramétrages importants :

- Tous les déplacements — par conséquent tous les segments — sont identifiés par des coordonnées non pas absolues, mais relatives à un point initial. Pour déplacer la position de visualisation du caractère, il suffit de modifier les coordonnées du point d'origine.
- Les dimensions des caractères sont aisément modifiables au moyen du paramètre S.

Dans le cas du segment a, par exemple, la première position n'a subi aucune variation par rapport à l'origine. Autrement dit, son incrément est de 0 sur l'axe X ainsi que sur l'axe Y.

En revanche, les coordonnées finales du segment (X0 et Y0 + S) ont pour incrément S sur l'axe Y (et 0 sur X). Ce paramétrage permet d'écrire un programme généralisé capable de générer tous les segments.
Si (X1, Y1) est l'origine d'un segment et (X2, Y2) son autre extrémité, on a :

$X1 = X0 + N_X S$ (incrément sur l'axe X)
$X1 = X0 \pm N_Y S$ (incrément sur l'axe Y)

Il en est de même pour les coordonnées X2, Y2 (n'oublions pas que X0 et Y0 sont les coordonnées de référence). Dans ce cas particulier, pour les premiers incréments, on a N = 0 (origine), c'est-à-dire :

$X1 = X0$
$Y1 = Y0$

En revanche, pour le second point nous avons NX = 0 et Ny = 1, d'où :

$X2 = X0$
$Y2 = Y0 \pm S$

X1, Y1 et X2, Y2 sont les coordonnées des extrémités par rapport à l'origine (X0, Y0) prise comme référence. Le signe à utiliser dans la deuxième expression (+ ou -) dépend de l'orientation de l'axe Y. Dans ce cas des terminaux vidéo dont l'origine de l'écran est en bas, le signe est + (les Y augmentent vers le haut) et inversement.

Pour obtenir un programme généralisé, il suffit de mémoriser les incréments (voir figure 1 page 1484) de chaque segment et les segments (figure 2) composant chaque nombre. L'organigramme ci-contre illustre un programme de visualisation d'un nombre aux dimensions désirées (grâce au paramètre S), situé à n'importe quel endroit de l'écran (en faisant varier l'origine X0, Y0).

Les incréments nécessaires pour obtenir un segment sont mémorisés dans 4 tableaux :

XS(I),YD(I)=coordonnées du point initial d'un segment

XA(I),YA(I)=coordonnées du point final d'un segment

Les valeurs sont affectées à la ligne 60 à l'aide des instructions DATA des lignes 910 à 916 (comparer les données à celles du tableau 1, page 1484).

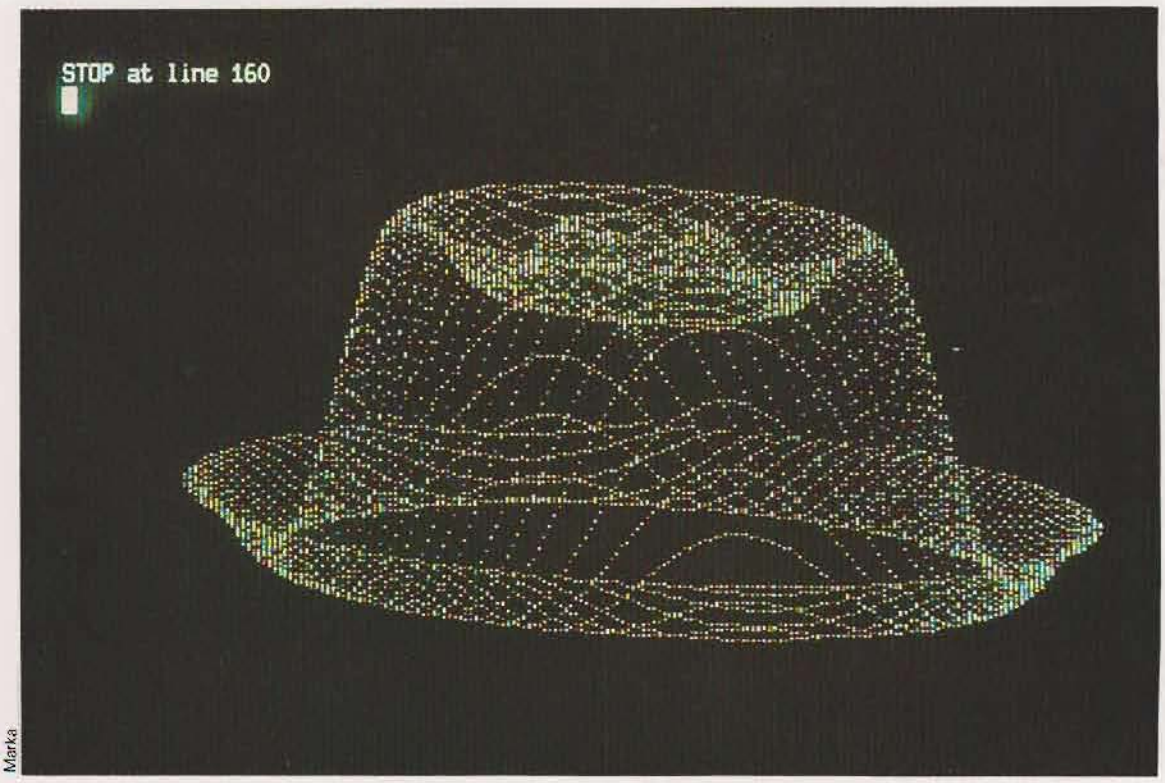
Les segments nécessaires à la visualisation des dix chiffres sont rangés dans le tableau à deux dimensions C% (10,7) dont les cases sont remplies au moyen de l'instruction READ (ligne 76) et des instructions DATA (lignes 918 à 927).

La première dimension (10) indique un des dix chiffres représentables (0,1,2...) tandis que le deuxième (7) représente les segments qui sont à activer.

Le programme utilise un tableau intermédiaire (V, de dimension 7), dans lequel sont transférés les éléments de C% correspondant aux segments à représenter (ligne 116) ; c'est donc le contenu de ce tableau qui est contrôlé (sous-programme 1000, figure inférieure de la page 1488).

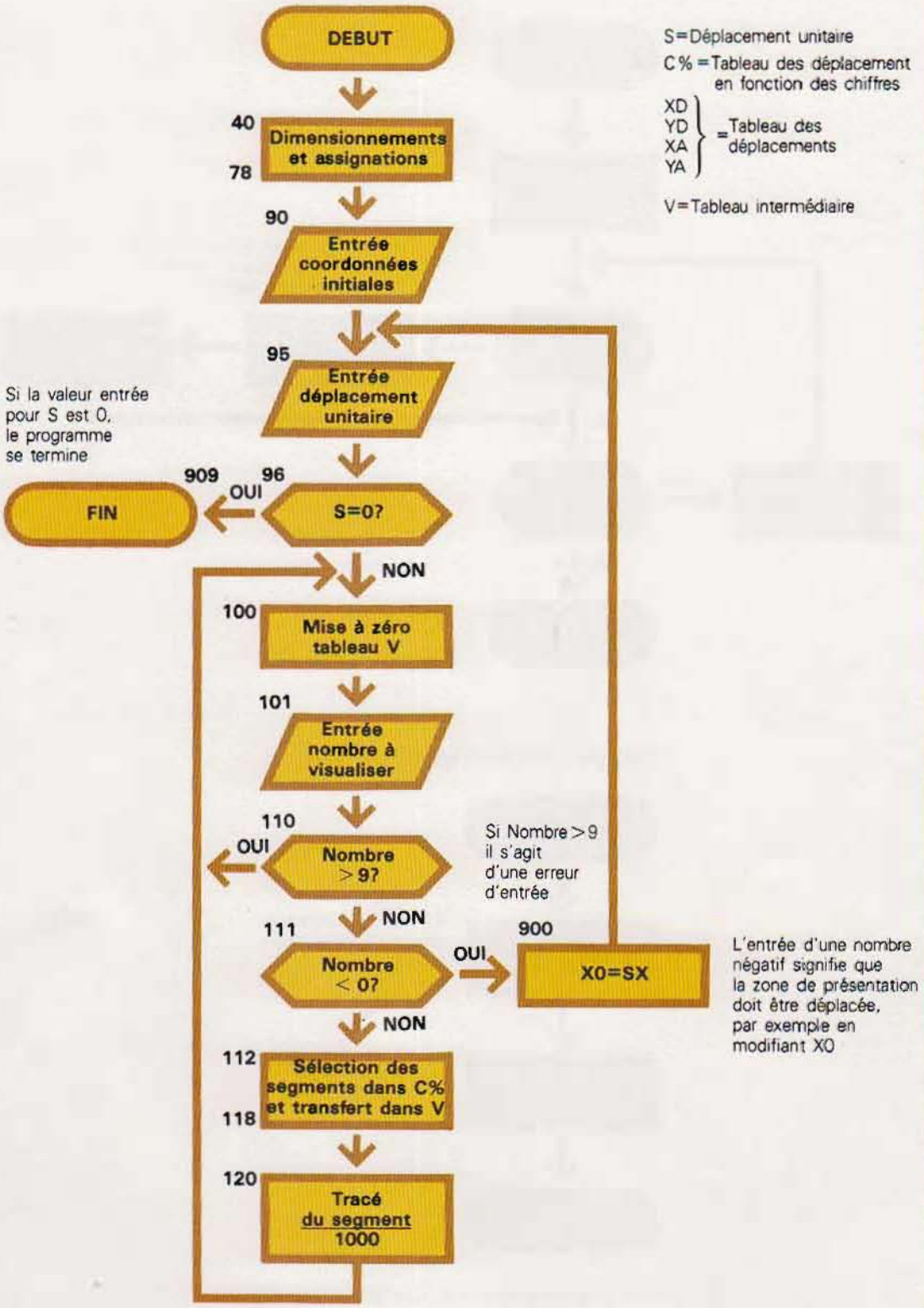
Le sous-programme 1000 (figure 1, page 1488) contient un appel au sous-programrne 2000 (figure 2) qui calcule les positions de début (X1, Y1) et de fin (X2, Y2) du segment et qui, ensuite, le trace.

Graphique tridimentionnel utilisant des courbes de niveau.

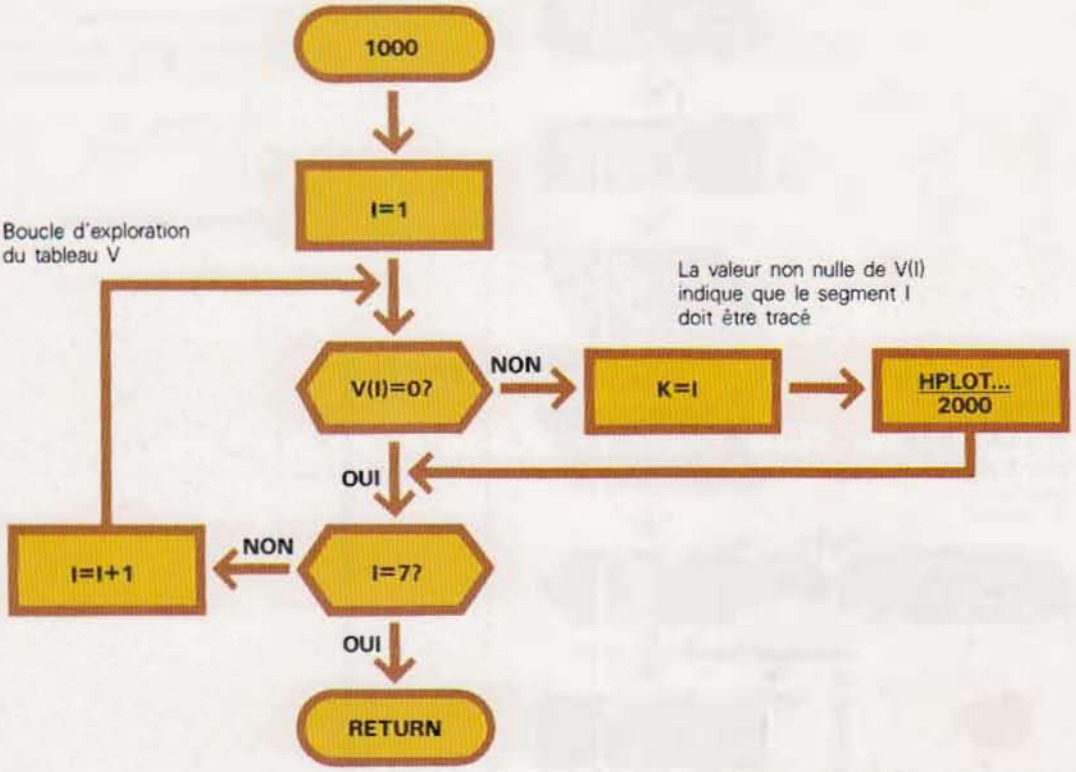


Marka

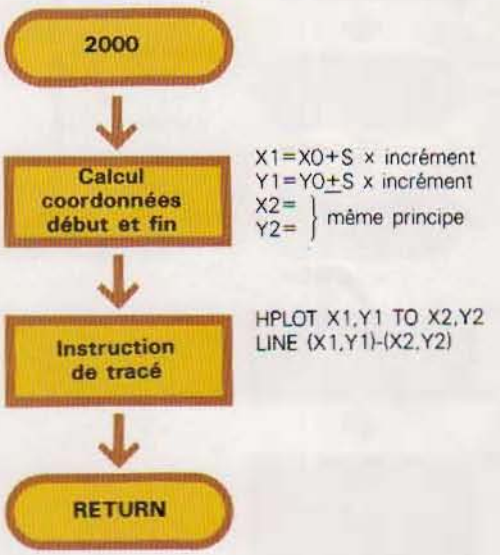
VISUALISATION D'UN NOMBRE EN MODE GRAPHIQUE



1/SOUS-PROGRAMME DE VISUALISATION D'UN SEGMENT



2/SOUS-PROGRAMME DE TRACE



Ce sous-programme dépend du type de machine utilisé

Pour des machines différentes d'Apple et de Siprel 2010 (programme ci-dessous), il suffit de modifier le sous-programme 3000. Dans la version Olivetti M20, le reste demeurant inchangé, les lignes 3010, 3030 et 3040 deviennent :

```
3010 Y1=Y0+YD (K)*S
3030 Y2=Y0+YA (K)*S
3040 LINE (X1,Y1) — (X2,Y2)
```

Le problème de la visualisation des caractères alphabétiques est également résolu sans difficulté grâce à un sous-programme qui représente chaque lettre comme un dessin.
Comme pour les chiffres, il s'agit donc de schématiser les caractères avec une série de segments. Mais l'analogie n'est que formelle. Pour les chiffres on peut définir 7 segments de base qui, selon qu'ils sont éteints ou allumés, génèrent tous les nombres. Pour les lettres, une telle procédure exigerait un nombre trop élevé de segments de base. Il faut donc employer une autre méthode pour concevoir n'importe quel type de figure.
Chaque lettre est schématisée par une série de segments (11 segments au maximum dans notre application) et visualisée en positionnant le pinceau électronique sur le point choisi comme début, puis en traçant les segments nécessaires.

VISUALISATION DES CHIFFRES EN MODE GRAPHIQUE

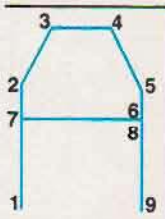
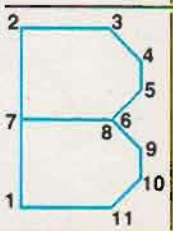
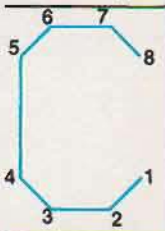
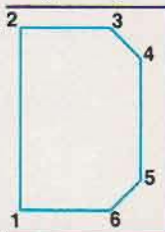
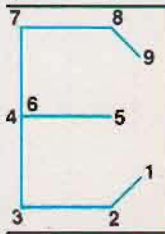
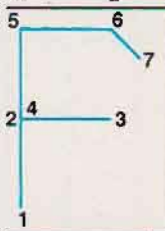
```
10 REM ---------
20 REM NOMBRES
30 REM ---------
40 DIM XD(7),YD(7),XA(7),YA(7)
45 DIM V(7),C%(10,7)
46 HGR
50 FOR I=1 TO 7
60 READ XD(I),YD(I),XA(I),YA(I)
70 NEXT I
72 FOR I=1 TO 10
74 FOR J=7 TO 1 STEP -1
76 READ C%(I,J)
   : NEXT J
78 NEXT I
90 PRINT "COORDONNEES INITIALES"
91 INPUT "X0 = ?" ; X0
92 INPUT "Y0 = ?" ; Y0
93 SX=X0
95 INPUT "DEPLACEMENT UNITAIRE? " ; S
96 IF S=0 GOTO 909
97 Y0=Y0+3*S
100 FOR I=1 TO 7
   : V(I)=0
   : NEXT I
101 INPUT "NOMBRE? " ; N
110 IF N>9 GOTO 100
111 IF N<0 GOTO 900
112 L%=N
   : IF L%=0 THEN L%=10
114 FOR I=1 TO 7
116 V(I)=C%(L%,I)
118 NEXT I
120 GOSUB 1000
160 X0=X0+2*S
170 GOTO 100
900 X0=SX
902 GOTO 95
909 END
```

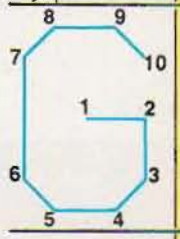
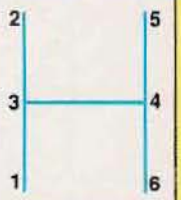
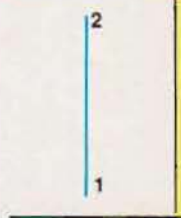
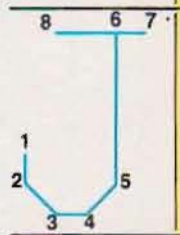
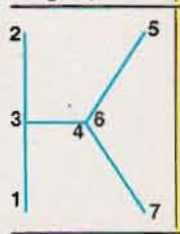
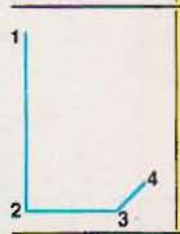
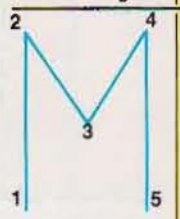
```
910 DATA 0,0,0,1
    : REM =a
911 DATA 0,1,0,2
    : REM =b
912 DATA 0,2,1,2
    : REM =c
913 DATA 1,2,1,1
    : REM =d
914 DATA 1,1,1,0
    : REM =e
915 DATA 1,0,0,0
    : REM =f
916 DATA 0,1,1,1
    : REM =g
918 DATA 0,0,1,1,0,0,0
    : REM =1
919 DATA 1,1,0,1,1,0,1
    : REM =2
920 DATA 1,1,1,1,1,0,0
    : REM =3
921 DATA 1,0,1,0,0,1,0
    : REM =4
922 DATA 1,1,1,0,1,1,0
    : REM =5
923 DATA 1,1,1,0,0,1,1
    : REM =6
924 DATA 0,0,1,1,1,0,0
    : REM =7
925 DATA 1,1,1,1,1,1,1
    : REM =8
926 DATA 1,0,1,1,1,1,0
    : REM =9
927 DATA 0,1,1,1,1,1,1
    : REM =0
1000 FOR I=1 TO 7
1020 IF V(I)=0 GOTO 1040
1025 K=I
1030 GOSUB 3000
1040 NEXT I
1050 RETURN
3000 X1=X0+XD(K)*S
3010 Y1=Y0-YD(K)*S
3020 X2=X0+XA(K)*S
3030 Y2=Y0-YA(K)*S
3040 HPLOT X1,Y1 TO X2,Y2
3050 RETURN
```

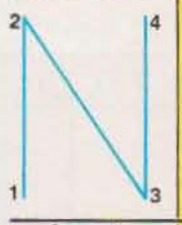
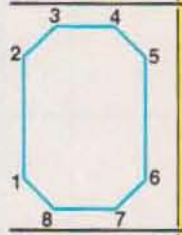
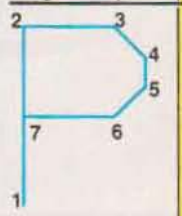
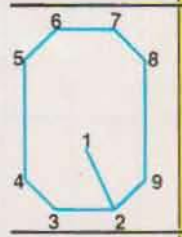
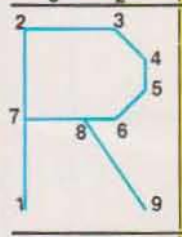
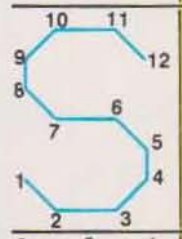
Le tableau ci-contre et les suivants listent les déplacements qui permettent de visualiser les caractères alphabétiques. Chaque lettre est contenue dans un rectangle de base 4 et de hauteur 6 : sa position est définie par l'angle en bas à gauche et le sens des déplacements est positif vers la droite pour l'axe X ; positif vers le haut pour l'axe Y. La première colonne du tableau indique le déplacement à effectuer pour se positionner sur le premier point de départ. Ce déplacement est nécessaire dans la mesure où la position du caractère sur l'écran doit être définie par l'utilisateur. Pour la lettre A par exemple, le tracé commence au point de référence (point 1), le déplacement initial est donc zéro. Pour tracer le C, il faut, au contraire, commencer par un point différent et prévoir un déplacement initial (4,1). Dans les tableaux, les colonnes numérotées (1, 2..., 11) repèrent les déplacements qui ne sont pas égaux à ceux figurant sur chaque lettre. Ils précisent le parcours effectué. Pour la lettre A par exemple, le premier déplacement, qui va du point 1 au point 2, est 0,4, le deuxième est 1,2, etc.

VISUALISATION DES CARACTERES ALPHABETIQUES EN MODE GRAPHIQUE



Lettre	Position	Déplacements 1	2	3	4	5	6	7	8	9	10	11
	0,0	0,4	1,2	2,0	1,-2	0,-1	-4,0	4,0	0,-3	-	-	-
	0,0	0,6	3,0	1,-1	0,-1	-1,-1	-3,0	3,0	1,-1	0,-1	-1,-1	-3,0
	4,1	-1,-1	-2,0	-1,1	0,4	1,1	2,0	1,-1	-	-	-	-
	0,0	0,6	3,0	1,-1	0,-4	-1,-1	-3,0	-	-	-	-	-
	4,1	-1,-1	-3,0	0,+3	3,0	-3,0	0,3	3,0	1,-1	-	-	-
	0,0	0,3	3,0	-3,0	0,3	3,0	1,-1	-	-	-	-	-

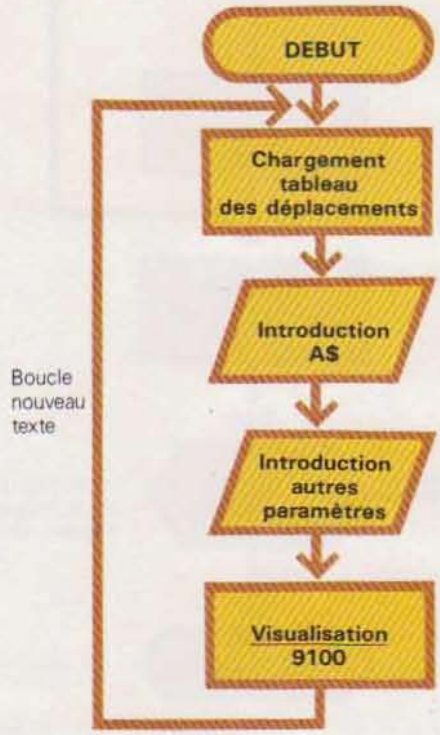
		1	2	3	4	5	6	7	8	9	10	11
	2,3	2,0	0,-2	-1,-1	-2,0	-1,1	0,4	1,1	2,0	1,-1	-	-
	0,0	0,6	0,-3	4,0	0,3	0,-6	-	-	-	-	-	-
	2,0	0,6	-	-	-	-	-	-	-	-	-	-
	0,2	0,-1	1,-1	1,0	1,1	0,5	1,0	-3,0	-	-	-	-
	0,0	0,6	0,-3	2,0	2,3	-2,-3	2,-3	-	-	-	-	-
	0,6	0,-6	3,0	1,1	-	-	-	-	-	-	-	-
	0,0	0,6	2,-3	2,3	0,-6	-	-	-	-	-	-	-

	1	2	3	4	5	6	7	8	9	10	11
 0,0	0,6	4,-6	0,6	-	-	-	-	-	-	-	-
 0,1	0,4	1,1	2,0	1,-1	0,-4	-1,-1	-2,0	-1,1	-	-	-
 0,0	0,6	3,0	1,-1	0,-1	-1,-1	-3,0	-	-	-	-	-
 2,2	1,-2	-2,0	-1,1	0,4	1,1	2,0	1,-1	0,-4	-1,-1	-	-
 0,0	0,6	3,0	1,-1	0,-1	-1,-1	-3,0	2,0	2,-3	-	-	-
 0,1	1,-1	2,0	1,1	0,1	-1,1	-2,0	-1,1	0,1	1,1	2,0	1,-1
 2,0	0,6	-2,0	4,0	-	-	-	-	-	-	-	-



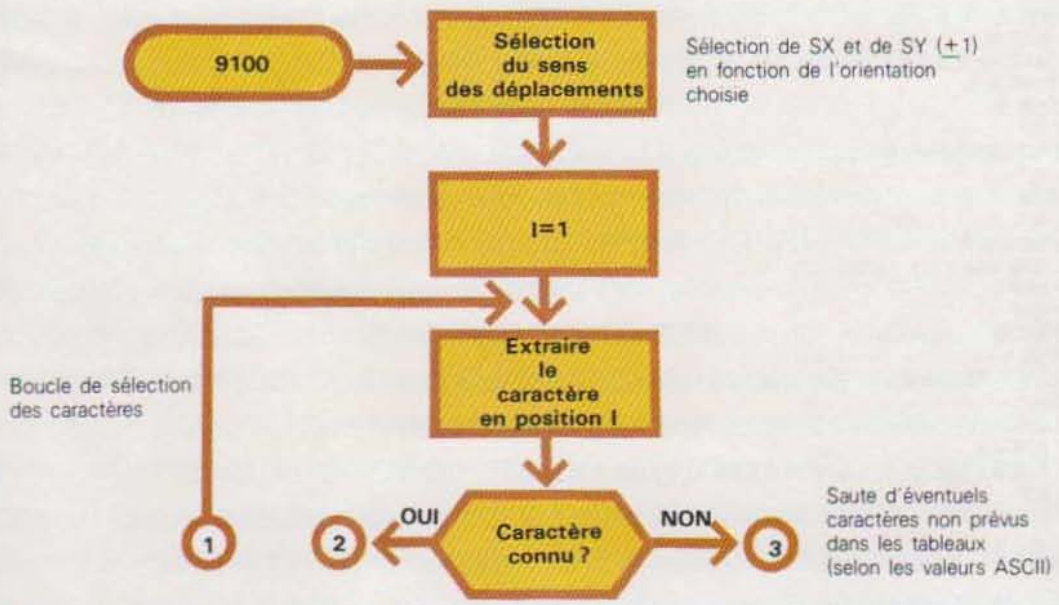
		1	2	3	4	5	6	7	8	9	10	11
1 6 2 5 3 4	0,6	0,-4	1,-2	2,0	1,2	0,4	-	-	-	-	-	-
1 5 2 4 3	0,6	0,-3	2,-3	2,3	0,3	-	-	-	-	-	-	-
8 4 9 7 5 3 6 2 10 1 11	0,0	0,1	4,4	0,1	0,-1	-2,-2	-2,2	0,1	0,-1	4,-4	0,-1	-
3 5 4 2 1	2,0	0,3	-2,3	2,-3	2,3	-		-	-	-	-	-
1 5 3 2 4	0,6	0,-6	2,3	2,-3	0,6	-	-	-	-	-	-	-
5 4 6 1 3 2	4,1	-1,-1	-3,0	4,6	-3,0	-1,-1	-	-	-	-	-	-

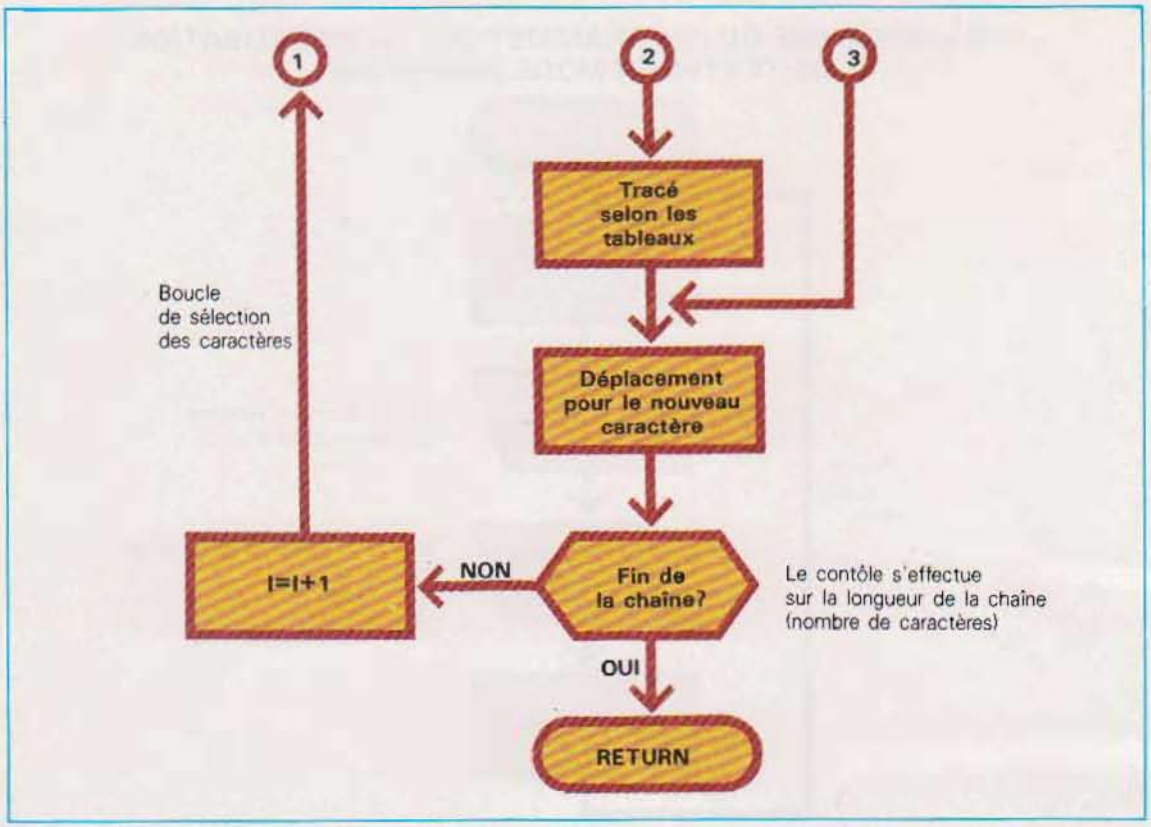
ORGANIGRAMME DU PROGRAMME POUR LA VISUALISATION DE TEXTES EN MODE GRAPHIQUE



Le programme n'a qu'une valeur d'exemple et ne prévoit pas de point de sortie

SOUS-PROGRAMME DE VISUALISATION DES CARACTERES





VISUALISATION DES CARACTERES EN MODE GRAPHIQUE

```
10 HGR
   : HCOLOR=3
20 DIM PO%(26,2),CAR%(26,11,2)
30 GOSUB 350
40 HOME
50 VTAB 22
60 INPUT A$
70 INPUT X0,Y0,S,D,MOV
80 GOSUB 9100
90 GOTO 40
320 REM ----------------
330 REM INITIALISATION
340 REM ----------------
350 FOR I=1 TO 26
360 READ PO%(I,1),PO%(I,2)
370 FOR J=1 TO 11
380 READ CAR%(I,J,1),CAR%(I,J,2)
390 NEXT
400 NEXT
410 RETURN
420 REM ----------------------
430 REM        DONNEES
440 REM ----------------------
450 REM.          A
460 DATA 0,0
470 DATA 0,4,1,2,2,0,1,-2,0,-1,-4,0,4,0,0,-3,0,0,0,0,0,0
480 REM           B
490 DATA 0,0
500 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,3,0,1,-1,0,-1,-1,-1,-3,0
510 REM           C
```

```
520 DATA 4,1
530 DATA -1,-1,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,0,0,0,0,0,0,0
540 REM          D
550 DATA 0,0
560 DATA 0,6,3,0,1,-1,0,-4,-1,-1,-3,0,0,0,0,0,0,0,0,0,0,0
570 REM          E
580 DATA 4,1
590 DATA -1,-1,-3,0,0,3,3,0,-3,0,0,3,3,0,1,-1,0,0,0,0,0,0
600 REM          F
610 DATA 0,0
620 DATA 0,3,3,0,-3,0,0,3,3,0,1,-1,0,0,0,0,0,0,0,0,0,0
630 REM          G
640 DATA 2,3
650 DATA 2,0,0,-2,-1,-1,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,0,0,0
660 REM          H
670 DATA 0,0
680 DATA 0,6,0,-3,4,0,0,3,0,-6,0,0,0,0,0,0,0,0,0,0,0,0
690 REM          I
700 DATA 2,0
710 DATA 0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
720 REM          J
730 DATA 0,2
740 DATA 0,-1,1,-1,1,0,1,1,0,5,1,0,-3,0,0,0,0,0,0,0,0,0
750 REM          K
760 DATA 0,0
770 DATA 0,6,0,-3,2,0,2,3,-2,-3,2,-3,0,0,0,0,0,0,0,0,0,0
780 REM          L
790 DATA 0,6
800 DATA 0,-6,3,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
810 REM          M
820 DATA 0,0
830 DATA 0,6,2,-3,2,3,0,-6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
840 REM          N
850 DATA 0,0
860 DATA 0,6,4,-6,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
870 REM          O
880 DATA 0,1
890 DATA 0,4,1,1,2,0,1,-1,0,-4,-1,-1,-2,0,-1,1,0,0,0,0,0,0
900 REM          P
910 DATA 0,0
920 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,0,0,0,0,0,0,0,0,0,0
930 REM          Q
940 DATA 2,2
950 DATA 1,-2,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,-4,-1,-1,0,0,0,0
960 REM          R
970 DATA 0,0
980 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,2,0,2,-3,0,0,0,0,0,0
990 REM          S
1000 DATA 0,1
1010 DATA 1,-1,2,0,1,1,0,1,-1,1,-2,0,-1,1,0,1,1,1,2,0,1,-1
1020 REM         T
1030 DATA 2,0
1040 DATA 0,6,-2,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
1050 REM          U
1060 DATA 8,6
1070 DATA 8,-4,1,-2,2,0,1,2,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1080 REM          V
1090 DATA 8,6
1100 DATA 8,-3,2,-3,2,3,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1110 REM          W
1120 DATA 8,6
1130 DATA 8,-6,2,3,2,-3,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1140 REM          X
1150 DATA 8,8
1160 DATA 8,1,4,4,4,0,1,0,-1,-2,-2,-2,2,0,1,0,-1,4,-4,0,-1,0,0
1170 REM          Y
1180 DATA 2,0
1190 DATA 8,3,-2,3,2,-3,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1200 REM          Z
1210 DATA 4,1
1220 DATA -1,-1,-3,0,4,6,-3,0,-1,-1,0,0,0,0,0,0,0,0,0,0,0,0
9000 REM ---------------------------------
9001 REM LETTRES EN HAUTE RESOLUTION
9002 REM ---------------------------------
9023 REM ---------------------------------
9100 D=D+4
9110 IF MOV=1 THEN SX=1
     : SY=1
     : M1=1
     : M2=2
9120 IF MOV=2 THEN SX=1
     : SY=-1
     : M1=2
     : M2=1
9130 IF MOV=3 THEN SX=-1
     : SY=-1
     : M1=2
     : M2=2
9140 IF MOV=4 THEN SX=-1
     : SY=1
     : M1=2
     : M2=1
9150 LU=LEN(A$)
9160 FOR I=1 TO LU
9170 LE$=MID$(A$,I,1)
9180 COD=ASC(LE$)
     : NL=COD-64
9190 IF COD<65 OR COD>90 THEN 9260
     : REM A MODIFIER POUR LES NOUVEAUX CARACTERES
9200 X1=X0+(S*PO%(NL,M1)*SX)
     : Y1=Y0-(S*PO%(NL,M2)*SY)
9210 X2=X1+(S*CAR%(NL,J,M1)*SX)
     : Y2=Y1-(S*CAR%(NL,J,M2)*SY)
9230 HPLOT X1,Y1 TO X2,Y2
9240 X1=X2
     : Y1=Y2
9250 NEXT
9260 IF MOV=2 OR MOV=4 THEN Y0=Y0-(D*S*SY)
     : GOTO 9200
9270 X0=X0+(D*S*SX)
9280 NEXT
9290 RETURN
```

Le programme (1495 à 1498) se compose surtout de 3 blocs. Le principal (lignes 10, 90) contient la définition des zones de mémoire (DIM) et les appels de sous-programmes. Parmi ces derniers, la ligne 350 définit, par une série d'instruction DATA, les valeurs des déplacements ; la 9100 les exécute, en visualisant le caractère.

Soit l'exemple d'une chaîne représentée par A$ et des paramètres XO, YO, S, D, MOV lus à partir du clavier. Pour généraliser le programme et le rendre exécutable dans toutes les applications, il suffit de le transformer en sous-programme et d'y entrer les valeurs acquises ou définies ailleurs.

Paramètres utilisés :

XO, YO : Coordonnées du point de référence du rectangle contenant la droite (début de l'écriture).

S : Facteur d'échelle qui détermine les dimensions de visualisation des caractères. En multipliant par S les déplacements on obtient le même caractère agrandi.

D : Espace qui définit la distance entre deux caractères.

MOV : Indique l'orientation du texte. Les orientations codifiées par des numéros allant de 1 à 4 sont précisées dans le listing.

REPRESENTATION DES CARACTERES NUMERIQUES EN MODE GRAPHIQUE

L'écran en bas de page illustre une représentation graphique des caractères numériques.

Ci-contre : phase d'introduction des coordonnées initiales et de la valeur du déplacement unitaire.



Le programme active maintenant le mode graphique, en conservant une fenêtre de 4 lignes pour le dialogue.
L'utilisateur a introduit une série de chiffres à représenter et à afficher sur l'écran.
L'introduction d'un nombre négatif implique une nouvelle sélection du déplacement unitaire.



Après avoir sélectionné un déplacement plus petit, les dimensions des chiffres se réduisent. Notons bien, cependant, que les proportions restent les mêmes.



Dans ce cas, on a changé les valeurs des coordonnées de départ et le déplacement unitaire.



Les chiffres visualisés avec un déplacement 4 sont toujours plus grands que les chiffres ASCII standard, que l'on voit dans la partie réservée au dialogue.



Le dernier écran montre la réduction d'échelle progressive que l'on obtient au fur et à mesure que l'on diminue la valeur du déplacement unitaire.



VISUALISATION DES CARACTERES ALPHABETIQUES EN MODE GRAPHIQUE

Exemples d'emploi de caractères alphabétiques en mode graphique.

Ci-contre, le résultat de l'exécution du programme. On a introduit la chaîne BASIC.



Le deuxième écran montre les différentes possibilités du programme. Le texte peut être positionné sur n'importe quel point de l'écran, avec l'orientation désirée.



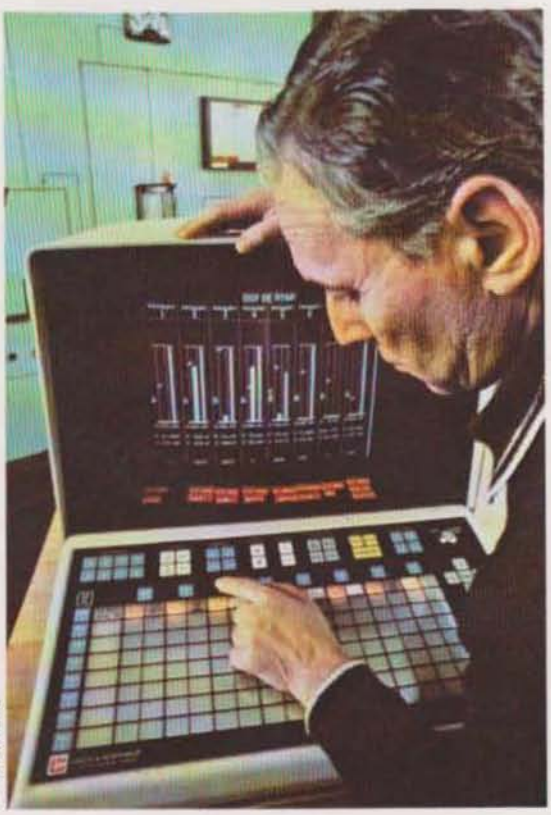
En travaillant davantage le plan de tracé des caractères, il est possible de reproduire divers caractères typographiques. L'écran ci-contre en fournit quelques échantillons.



Histogrammes et diagrammes circulaires

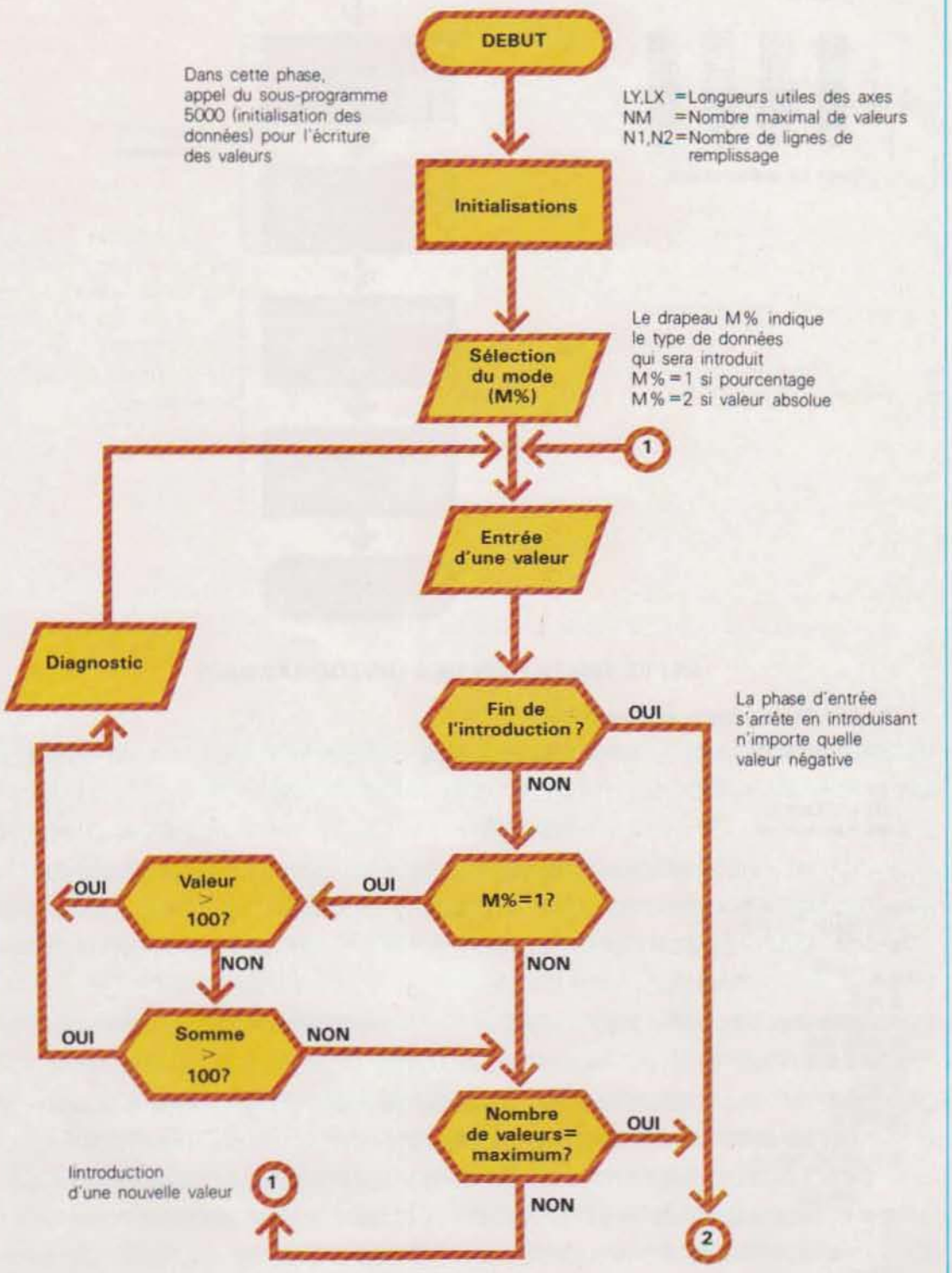
Une représentation graphique quelconque s'obtient en traçant la progression de la fonction mathématique qui le décrit (à condition qu'elle existe). Certains phénomènes ne peuvent pas être décrits au moyen d'une fonction, car la quantité à représenter ne dépend pas d'une variable (indépendante) qui varie de manière continue, ou parce que la variable indépendante n'est pas de type numérique.
Nous avons déjà donné un exemple se rapportant aux ventes d'un article pour différents vendeurs ; il n'était pas possible de considérer, dans ce cas, les vendeurs (A, B, C...) comme des données numériques. Un autre exemple très courant concerne la représentation du bilan mensuel sur une année. Dans ce cas, "la variable indépendante" est le nom du mois (janvier, février...), qui n'est certes pas une variable numérique.
Pour résoudre ces problèmes de représentation graphique, il existe deux méthodes très répandues : les histogrammes et les diagrammes circulaires.
Les histogrammes se présentent comme une série de colonnes d'une hauteur proportionnelle aux valeurs à représenter selon l'axe Y et distantes, sur l'axe X, d'une grandeur proportionnelle aux valeurs de la variable indépendante (X). En général, le pas est constant, dans la mesure où le phénomène est observé à des intervalles réguliers. Dans le cas d'un bilan, la variable indépendante est le temps et la variable dépendante le montant. Elles sont reportées sur une période mensuelle, l'axe X étant divisé en 12 parties (alors que l'axe Y représente le chiffre mensuel, soit en valeur absolue, soit en pourcentage du total). Cette dernière forme est la plus usitée car elle fournit une vision immédiate du phénomène et facilite la construction du diagramme.
Le diagramme circulaire est une représentation proche de l'histogramme, où chaque valeur est convertie en une portion de cercle de grandeur proportionnelle. L'ensemble du cercle représente la valeur totale et chaque secteur la valeur d'une de ses composantes. Exemple : pour représenter le bilan précédent sous cette forme, il faut construire un cercle divisé en 12 secteurs, un par mois, d'une grandeur proportionnelle au pourcentage du montant de ce mois sur le total annuel.
L'histogramme fournit donc une vision de l'évolution du bilan dans le temps, alors que le diagramme circulaire montre comment l'ensemble du bilan se répartit entre les diverses parties (mois). Il s'agit donc de deux représentations qui se complètent. Pages 1503-1504, on a construit l'organigramme d'un programme de représentation d'histogrammes (voir listings en pages 1504-1510). Les données à représenter sont soit des valeurs absolues, soit des pourcentages. Dans le premier cas, le programme les convertit lui-même en pourcentages. L'utilisateur choisit entre 5 formes de représentation différentes sur le même graphique. Dans le programme, les 5 modes de présentation se réfèrent à un écran monochromatique.
Ce fait complique le sous-programme de présentation car il oblige à prévoir divers modes de représentation des colonnes de l'histogramme. Sur un écran couleur on peut associer, à chaque mode de représentation, une couleur différente et dessiner les divers histogrammes.

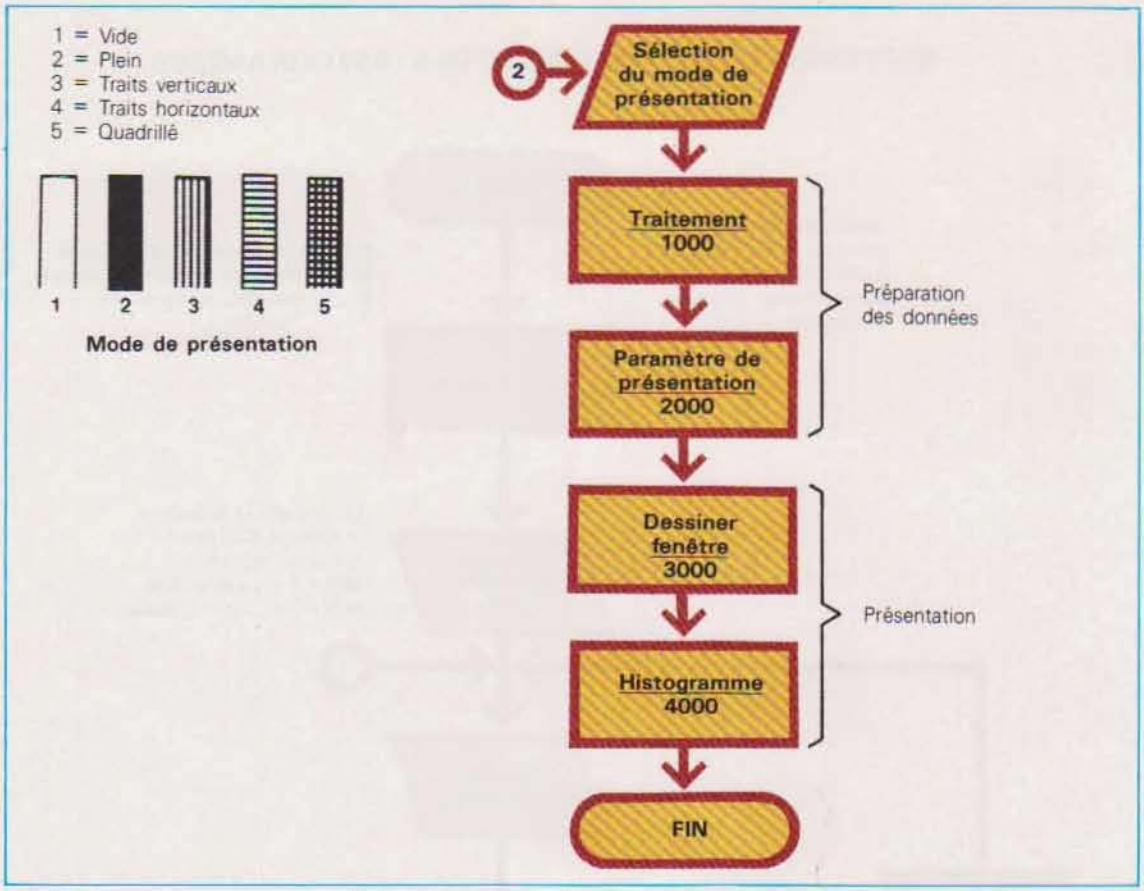
Contrôle de processus sur écran graphique



K. Reese/Marka

REPRESENTATION GRAPHIQUE DES HISTOGRAMMES





REPRESENTATION DES HISTOGRAMMES

Version Siprel, Apple et compatibles

```
4 REM --------------
5 REM HISTOGRAMMES
6 REM --------------
7
    :
8
    :
9 DIM PO%(26,2)CAR%(26,11,2)
10 LX=230
    : LY=150
20 NM=20
30 P%=0
50 DIM D(NM),P(NM),S(NM),ES(NM)
60 GOSUB 5000
65 GOSUB 9120
70 TEXT
    : HOME
80 VTAB 12
    : INPUT "LES DONNEES SONT EN POURCENTAGE? (O/N)" ; A$
90 IF A$="O" THEN M%=1
    : P%=1
    : GOTO 110
100 M%=2
110 I=1
    : TT=0
120 HOME
```

```
    : VTAB 12
130 PRINT "INTRODUIRE LA DONNEE N " ; I
140 INPUT "(NEGATIF POUR FINIR): " ; D$
    : IF D$="" THEN 120
150 D=VAL(D$)
160 IF D>99999 THEN 120
170 IF D<0 THEN 260
180 D(I)=INT(D)
190 TT=TT+D(I)
200 IF M%=2 THEN 220
210 IF D(I)>100 OR TT>100 THEN GOSUB 6000
    : HOME
    : GOTO 110
    : REM          ERREUR
220 VTAB 15
    : INPUT "ETIQUETTE (MAX 3 CARACTERES): " ; E$
230 E$(I)=LEFT$(E$,3)
240 I=I+1
    : IF I>NM THEN 260
250 GOTO 120
260 IF TT=0 THEN RUN
270 NV=I-1
280 HOME
290 VTAB 5
    : PRINT "MODES DE PRESENTATION: "
300 VTAB 10
310 PRINT "1) VIDE"
320 PRINT "2) PLEIN"
330 PRINT "3) TRAITS VERTICAUX"
340 PRINT "4) TRAITS HORIZONTAUX"
350 PRINT "5) INCRUSTE"
360 VTAB 22
    : INPUT "CHOISIR: " ; MP$
    : IF MP$="" THEN 360
370 MP=VAL(MP$)
380 IF MP<1 OR MP>5 THEN 280
1000 REM ------------
1010 REM TRAITEMENT
1020 REM ------------
1030 IF M%=1 GOTO 1140
1040 HOME
    : VTAB 12
1050 PRINT "VOULEZ-VOUS UNE REPRESENTATION EN POURCENTAGE? ";
1060 INPUT "(O/N): " ; R$
1070 IF R$="O" THEN P%=1
    : GOTO 1140
1080 MA=D(1)
1090 FOR K=2 TO NV
1100 IF D(K)>(MA) THEN MA=D(K)
1120 NEXT K
1130 TT=MA
1140 FOR I=1 TO NV
1150 P(I)=D(I)/TT*100
1160 S(I)=INT(LY*P(I)/100)
1170 NEXT I
1180 LA=INT(LX/(2*NV))
2000 REM ----------------------------
2001 REM PARAMETRES DE PRESENTATION
2002 REM ----------------------------
2010 SX=1
    : SY=1
2040 ON MP GOTO 3000,3000,2060,2070,2080
2060 SX=2
    : GOTO 3000
2070 SY=2
    : GOTO 3000
```

```
2000 SY=2
     : SX=2
3000 REM ----------------------
3010 REM DESIN DE LA FENETRE
3020 REM ----------------------
3030 HGR2
     : HCOLOR=3
3040 HPLOT 34,0 TO 279,0 TO 279,159 TO 34,159 TO 34,0
3050 FOR I=1 TO 10
3060 V=159-I*15
3070 HPLOT 34,V TO 39,V
3075 HPLOT 274,V TO 278,V
3080 NEXT
3090 S=3
     : Y0=159
3100 IF P%=0 THEN 3170
3110 FOR NN=0 TO 100 STEP 10
3120 X0=9*S
3130 GOSUB 5500
3140 Y0=Y0-15
3150 NEXT
3160 GOTO 3250
3170 ST=MA/10
     : IF ST<>INT(ST) THEN ST=INT(ST)+1
3175 Y0=9
3180 FOR NN=MA TO 0 STEP -ST
3190 X0=9*S+3
3200 GOSUB 5500
3210 Y0=Y0+15
3220 NEXT
3230 X0=9*S+3
3235 IF NN+ST=0 THEN 3250
3240 NN=0
     : GOSUB 5500
3250 REM
4000 REM ----------------------------
4001 REM DESSIN DES HISTOGRAMMES
4002 REM ----------------------------
4005 N=0
4007 Y1=159
4010 FOR I=1 TO NV
4025 X1=50+N*LA
4027 X2=X1+LA
4030 HPLOT X1,Y1 TO X1,Y1-S(I) TO X2,Y1-S(I) TO X2,Y1
4040 IF MP=1 THEN 4900
4050 IF MP=4 THEN 4300
4060 Y=Y1
     : X=X1+SX
4070 IF X>X2 THEN 4200
4080 HPLOT X,Y TO X,Y-S(I)
4090 X=X+SX
4100 GOTO 4070
4200 IF MP=5 THEN 4300
4210 GOTO 4900
4300 X=X1
     : Y=Y1-SY
4310 IF Y<Y1-S(I) THEN 4900
4320 HPLOT X1,Y TO X2,Y
4330 Y=Y-SY
4340 GOTO 4310
4900 N=N+2
4905 A$=ES(I)
     : H0=X2
     : V0=181
     : S=1
     : D=2
     : MOV=4
     : GOSUB 10270
```

```
4910 NEXT I
4920 GET F$
4930 TEXT
   : HOME
   : VTAB 12
   : INPUT "DESIREZ-VOUS UN AUTRE GRAPHIQUE? (O/N)" ; R$
4940 IF LEFT$(R$,1)="O" THEN 70
4950 END
5000 REM ------------------------------
5010 REM INITIALISATION DES NOMBRES
5020 REM ------------------------------
5030 DIM XD(7),YD(7),XA(7),YA(7)
5040 DIM V(7),C%(10,7)
5050 FOR I=1 TO 7
5060 READ XD(I),YD(I),XA(I),YA(I)
5070 NEXT I
5080 FOR I=1 TO 10
5090 FOR J=7 TO 1 STEP -1
5100 READ C%(I,J)
   : NEXT J
5110 NEXT I
5120 RETURN
5500 REM ---------------------
5510 REM DESSIN DES NOMBRES
5520 REM ---------------------
5530 NN$=STR$(NN)
5540 LN=LEN(NN$)
5550 FOR KK=LN TO 1 STEP -1
5560 N=VAL(MID$(NN$,KK,1))
5570 X0=X0-2*S
5580 FOR I=1 TO 7
   : V(I)=0
   : NEXT I
5590 L%=N
   : IF L%=0 THEN L%=10
5600 FOR I=1 TO 7
5610 V(I)=C%(L%,I)
5620 NEXT I
5630 FOR I=1 TO 7
5640 IF V(I)=0 GOTO 5710
5650 K=1
5660 X1=X0+XD(K)*S
5670 Y1=Y0-YD(K)*S
5680 X2=X0+XA(K)*S
5690 Y2=Y0-YA(K)*S
5700 HPLOT X1,Y1 TO X2,Y2
5710 NEXT I
5720 NEXT KK
5730 RETURN
6000 VTAB 20
   : PRINT "INTRODUCTION ERRONEE: "
6010 PRINT
   : PRINT "LA SOMME DES DONNEES DEPASSE LA VALEUR '100'"
6015 PRINT
   : PRINT "FRAPPER UNE TOUCHE POUR RECOMMENCER"
6030 GET Q$
6040 RETURN
7945 REM --------------
7950 REM    DONNEES
7955 REM --------------
8000 DATA 0,0,0,1
   : REM =a
8010 DATA 0,1,0,2
   : REM =b
8020 DATA 0,2,1,2
   : REM =c
8030 DATA 1,2,1,1
   : REM =d
```

```
8040 DATA 1,1,1,0
    : REM =e
8050 DATA 1,0,0,0
    : REM =f
8060 DATA 0,1,1,1
    : REM =g
8070 DATA 0,0,1,1,0,0,0
    : REM =1
8080 DATA 1,1,0,1,1,0,1
    : REM =2
8090 DATA 1,1,1,1,1,0,0
    : REM =3
8100 DATA 1,0,1,0,0,1,0
    : REM =4
8110 DATA 1,1,1,0,1,1,0
    : REM =5
8120 DATA 1,1,1,0,0,1,1
    : REM =6
8130 DATA 0,0,1,1,1,0,0
    : REM =7
8140 DATA 1,1,1,1,1,1,1
    : REM =8
8150 DATA 1,0,1,1,1,1,0
    : REM =9
8160 DATA 0,1,1,1,1,1,1
    : REM =0
9090 REM ----------------
9100 REM INITIALISATION
9110 REM ----------------
9120 FOR I=1 TO 26
9130 READ PO%(I,1),PO%(I,2)
9140 FOR J=1 TO 11
9150 READ CAR%(I,J,1),CAR%(I,J,2)
9160 NEXT J
9170 NEXT I
9180 RETURN
9190 REM -----------------
9200 REM      DONNEES
9210 REM -----------------
9220 REM          A
9230 DATA 0,0
9240 DATA 0,4,1,2,2,0,1,-2,0,-1,-4,0,4,0,0,-3,0,0,0,0,0,0
9250 REM          B
9260 DATA 0,0
9270 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,3,0,1,-1,0,-1,-1,-1,-3,0
9280 REM          C
9290 DATA 4,1
9300 DATA -1,-1,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,0,0,0,0,0,0,0
9310 REM          D
9320 DATA 0,0
9330 DATA 0,6,3,0,1,-1,0,-4,-1,-1,-3,0,0,0,0,0,0,0,0,0,0,0
9340 REM          E
9350 DATA 4,1
9360 DATA -1,-1,-3,0,0,3,3,0,-3,0,0,3,3,0,1,-1,0,0,0,0,0,0
9370 REM          F
9380 DATA 0,0
9390 DATA 0,3,3,0,-3,0,0,3,3,0,1,-1,0,0,0,0,0,0,0,0,0,0
9400 REM          G
9410 DATA 2,3
9420 DATA 2,0,0,-2,-1,-1,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,0,0,0
9430 REM          H
9440 DATA 0,0
9450 DATA 0,6,0,-3,4,0,0,3,0,-6,0,0,0,0,0,0,0,0,0,0,0,0
```

```
9460 REM          I
9470 DATA 2,0
9480 DATA 0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9490 REM          J
9500 DATA 0,2
9510 DATA 0,-1,1,-1,1,0,1,1,0,5,1,0,-3,0,0,0,0,0,0,0,0,0
9520 REM          K
9530 DATA 0,0
9540 DATA 0,6,0,-3,2,0,2,3,-2,-3,2,-3,0,0,0,0,0,0,0,0,0,0
9550 REM          L
9560 DATA 0,6
9570 DATA 0,-6,3,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9580 REM          M
9590 DATA 0,0
9600 DATA 0,6,2,-3,2,3,0,-6,0,0,0,0,0,0,0,0,0,0,0,0,0
9610 REM          N
9620 DATA 0,0
9630 DATA 0,6,4,-6,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9640 REM          O
9650 DATA 0,1
9660 DATA 0,4,1,1,2,0,1,-1,0,-4,-1,-1,-2,0,-1,1,0,0,0,0,0,0
9670 REM          P
9680 DATA 0,0
9690 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,0,0,0,0,0,0,0,0,0,0
9700 REM          Q
9710 DATA 2,2
9720 DATA 1,-2,-2,0,-1,1,0,4,1,1,2,0,1,-1,0,-4,-1,-1,0,0,0,0
9730 REM          R
9740 DATA 0,0
9750 DATA 0,6,3,0,1,-1,0,-1,-1,-1,-3,0,2,0,2,-3,0,0,0,0,0,0
9760 REM          S
9770 DATA 0,1
9780 DATA 1,-1,2,0,1,1,0,1,-1,1,-2,0,-1,1,0,1,1,1,2,0,1,-1
9790 REM          T
9800 DATA 2,0
9810 DATA 0,6,-2,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9820 REM          U
9830 DATA 0,6
9840 DATA 0,-4,1,-2,2,0,1,2,0,4,0,0,0,0,0,0,0,0,0,0,0,0
9850 REM          V
9860 DATA 0,6
9870 DATA 0,-3,2,-3,2,3,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9880 REM          W
9890 DATA 0,6
9900 DATA 0,-6,2,3,2,-3,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9910 REM          X
9920 DATA 0,0
9930 DATA 0,1,4,4,0,1,0,-1,-2,-2,-2,2,0,1,0,-1,4,-4,0,-1,0,0
9940 REM          Y
9950 DATA 2,0
9960 DATA 0,3,-2,3,2,-3,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9970 REM          Z
9980 DATA 4,1
9990 DATA -1,-1,-3,0,4,6,-3,0,-1,-1,0,0,0,0,0,0,0,0,0,0,0,0
```

```
10000 REM ------------------------------
10010 REM LETTRES EN HAUTE RESOLUTION
10020 REM ------------------------------
10030 REM
10040 REM
10050 REM EN ENTREE
10060 REM
10070 REM A$ - CHAINE DE CARACTERES
10080 REM
10090 REM H0,V0 - POSITION DEBUT ECRITURE
10100 REM
10110 REM S - ECHELLE (>0)
10120 REM
10130 REM D - ESPACEMENT (>0)
10135 REM
10140 REM MOV - TYPE D'ECRITURE:
10150 REM
10160 REM 1=HORIZONTALE
10170 REM 2=DESCENDANTE
10180 REM 3=INVERSEE
10190 REM 4=MONTANTE
10200 REM
10210 REM
10220 REM
10230 REM ------------------------------
10270 D=D+4
10280 IF MOV=1 THEN PX=1
      : PY=1
      : M1=1
      : M2=2
10290 IF MOV=2 THEN PX=1
      : PY=-1
      : M1=2
      : M2=1
10300 IF MOV=3 THEN PX=-1
      : PY=-1
      : M1=1
      : M2=2
10310 IF MOV=4 THEN PX=-1
      : PY=1
      : M1=2
      : M2=1
10320 LU=LEN(A$)
10330 FOR II=1 TO LU
10340 LE$=MID$(A$,II,1)
10345 IF LE$=" " THEN II=LU
      : NEXT II
      : RETURN
10350 COD=ASC(LE$)
      : NL=COD-64
10360 IF COD<65 OR COD>90 THEN 10430
      : REM A MODIFIER POUR LES NOUVEAUX CARACTERES
10370 H1=H0+(S*PO%(NL,M1)*PX)
      : V1=V0-(S*PO%(NL,M2)*PY)
10380 FOR J=1 TO 11
10390 H2=H1+(S*CAR%(NL,J,M1)*PX)
      : V2=V1-(S*CAR%(NL,J,M2)*PY)
10400 HPLOT H1,V1 TO H2,V2
10410 H1=H2
      : V1=V2
10420 NEXT J
10430 IF MOV=2 OR MOV=4 THEN V0=V0-(D*S*PY)
      : GOTO 10450
10440 H0=H0+(D*S*PX)
10450 NEXT II
10460 RETURN
```

L'ordinateur et les aiguilleurs du ciel

Vous avez tous entendu parler des contrôleurs du trafic aérien — les aiguilleurs du ciel — mais peu de gens savent vraiment en quoi consiste leur activité et comment les technologies nouvelles sont devenues indispensables au bon déroulement de leurs tâches. Aujourd'hui, on trouve dans le monde des exemples d'automatisation complète du trafic aérien, depuis le contrôle de procédure jusqu'à l'échange de données entre ordinateurs.

Le choix de tel ou tel système est bien évidemment lié au rapport entre le coût de l'automatisation et les bénéfices que l'on peut en tirer. Dans le cas de la surveillance radar, les coûts d'entretien, la formation d'un personnel hautement qualifié sont tels qu'ils dissuadent encore de nombreux pays de se lancer dans l'aventure. Côté avantages, il faut citer une plus grande sécurité et de meilleures conditions de travail. Tout cela dépend, naturellement, du volume du trafic à contrôler.

Le contrôle du trafic aérien (CTA) est un service dont la finalité est d'assurer un trafic sûr, bien organisé et rapide, dans l'intérêt à la fois des voyageurs et des compagnies aériennes. L'organisation mondiale qui supervise et coordonne les problèmes complexes du contrôle aérien (CTA) est l'ICAO (International Civil Aviation Organization).

Pour faciliter le trafic, l'espace aérien mondial est divisé en FIR (Flight Information Region) dont le contrôle est confié aux nations géographiquement concernées.

Les problèmes de coordination sont réduits par un tracé, dans le ciel, de ***routes de navigation*** *idéales, d'une largeur d'environ 10 miles, et comportant plusieurs niveaux d'altitude.*

Contrairement à ce que l'on pourrait penser, le service de contrôle couvre ***toute la durée du vol****, depuis le décollage jusqu'à l'approche de la piste et l'atterrissage, qui restent toujours les phases opérationnelles les plus délicates.*

La tâche du contrôleur consiste à guider les avions par radio, le long des voies aériennes, en tenant compte des demandes des pilotes eux-mêmes ainsi que de la situation du trafic en cours. Seul le contrôleur se trouve en possession de l'ensemble de ces données car la vision du pilote est limitée par divers facteurs, dont la vitesse (un avion de ligne parcourt environ 10 km par minute).



Les trois aires de vol (FIR) composant l'espace aérien italien.

Pour faciliter le travail du contrôleur, les pilotes sont obligés, avant le départ, de rédiger un ***plan de vol*** *(FPL, Flight Plan) qui consigne toutes les informations concernant l'avion, ses caractéristiques techniques (type), sa destination, ainsi qu'une estimation des horaires. Dans le plan de vol, l'itinéraire est exprimé comme une succession de voies aériennes ou de points caractéristiques (FIX), qui bien souvent correspondent à des balises radio au sol, repérées par des noms conventionnels sur les cartes internationales.*

Le plan de vol est transmis à toutes les régions de vol (FIR) concernées par l'itinéraire, grâce à un réseau de ***télex spécial*** *appelé AFTN (Aeronautical Fixed Telecommunication Network).*

Dans chaque région, les assistants de l'aiguilleur du ciel tirent, de chaque plan de vol, les

informations se rapportant à leur propre région. Pour chaque FIX concerné par l'itinéraire, ils reçoivent une bande télex (strip) avec les données essentielles de l'appareil, l'altitude demandée, le nom du FIX et l'horaire de survol prévu. A titre d'exemple, la FIR de Rome prépare 10 à 15 strips par vol; donc pour une moyenne d'environ 900 vols par jour, on arrive en gros à environ 10 000 strips quotidiens.

Le système traditionnel, qui permet au contrôleur de connaître à tout instant le trafic correspondant à un FIX donné, consiste à classer chronologiquement les strips concernant ce point, afin d'éviter les ***croisements d'avions*** *sur un même niveau.*

Pour un contrôle de ce type (dit de procédure), la distance requise entre deux avions est en général de 10 minutes, s'ils volent à la même altitude, ou de 300 mètres (1 000 pieds), s'ils volent à des altitudes différentes.

Un moment très délicat, du point de vue opérationnel, est celui où un avion franchit la limite entre deux régions de vol.

Un réseau téléphonique spécial permet, aux contrôleurs de régions limitrophes, de coordonner le point, l'altitude et le moment du transfert de contrôle.

Grâce à cette communication, le nouveau contrôleur peut évaluer la correction à effectuer par rapport aux horaires estimés des strips, pour synchroniser la ***trajectoire prévue*** *et la* ***véritable position*** *de l'avion.*

Diverses communications avec le pilote au moment du survol des FIX permettent d'évaluer d'éventuels retards ou avances par rapport aux prévisions.

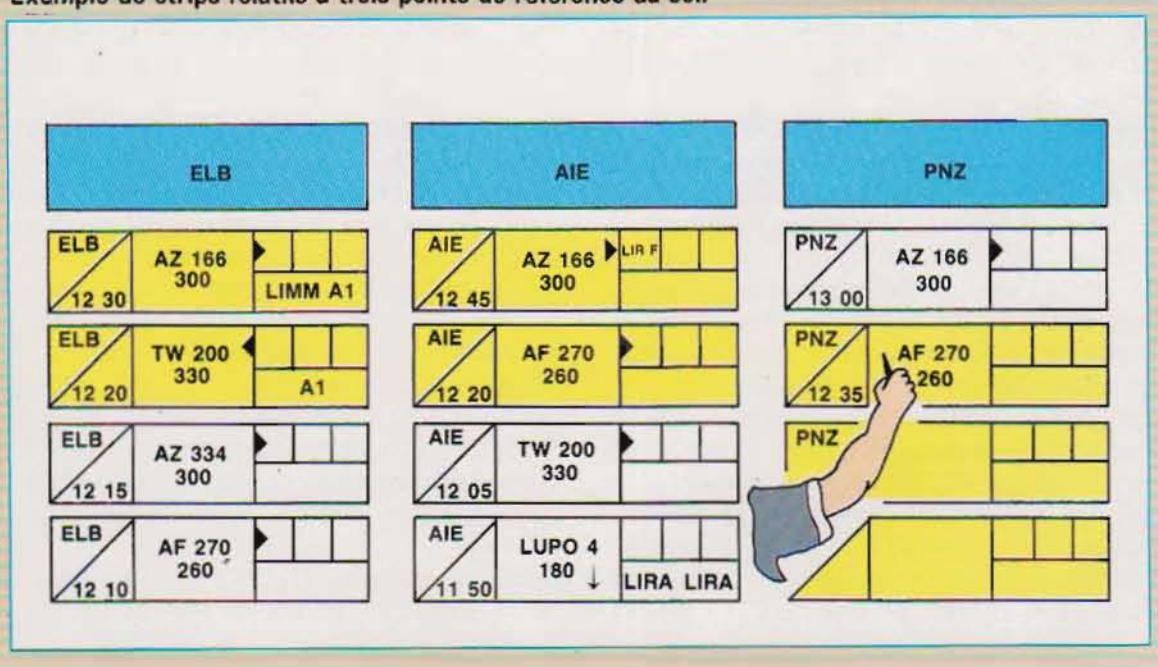
La première application technologique importante du CTA a eu lieu dans les années 40. Il s'agissait du ***radar****. Celui-ci a permis de visualiser le trafic en cours, en générant des images qui n'étaient qu'une série de points lumineux sur un écran (les échos radar).*

Entre 1958 et 1960, on commence à utiliser un second radar dit SSR (Secondary Surveillance Radar), qui interroge un appareil récepteur-émetteur, embarqué dans l'avion, le ***transpondeur****. La réponse numérique que le SSR obtient du transpondeur véhicule une série d'informations concernant l'identité de l'appareil (exprimée avec un code de 64 bits) et l'altitude de vol.*

Ces communications sont traitées par un ordinateur pour l'identification des vols sur écrans PPI (Plan Position Indicator).

Le premier degré d'automatisation est ce que l'on appelle le labelling (étiquetage), qui consiste à associer, à l'écho radar primaire, une étiquette donnant notamment le code de l'avion et son altitude.

Exemple de strips relatifs à trois points de référence au sol.



Après un premier contact radio avec le pilote, le contrôleur remplace le code d'identification (envoyé à l'ordinateur par le transpondeur) par la désignation normale du vol (par exemple AZ 128), qui facilitera la **comparaison** entre la **position** de l'avion, telle qu'elle ressort des informations radar et les **fiches de progression** (voir graphique page précédente).

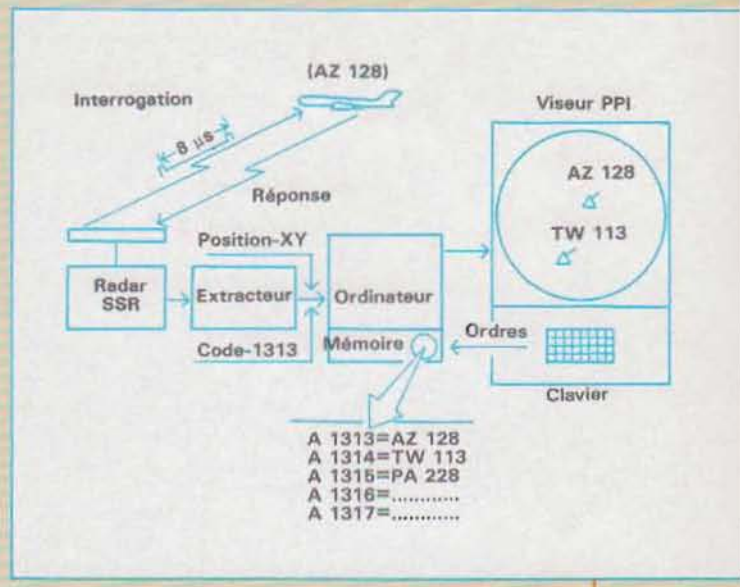
Grâce à l'ordinateur, l'image-écran du moniteur PPI (Plan Position Indicator) est enrichie de plusieurs fonctions destinées au contrôleur, comme par exemple :

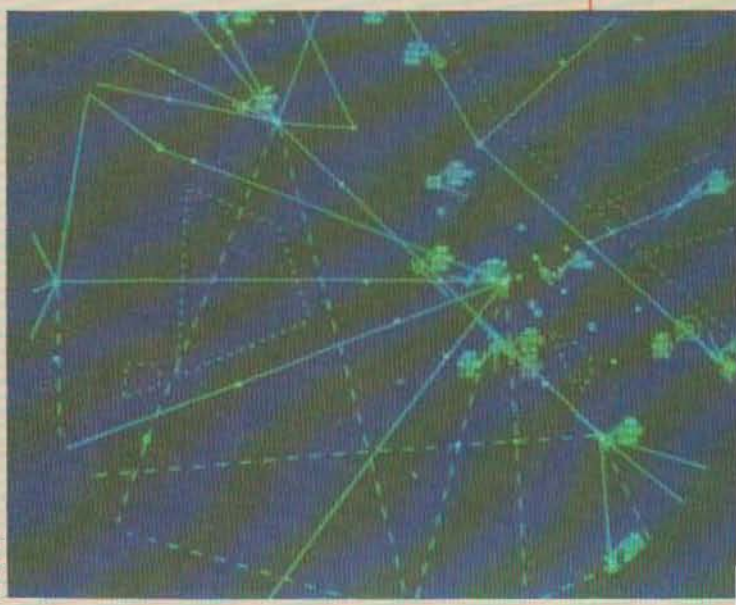
- Une vue schématique des routes aériennes,
- L'ensemble des FIX (points caractéristiques) et des zones particulières (aéroports, zones interdites à la navigation civile, altitudes des principaux reliefs proches d'un aéroport).
- Une carte des itinéraires d'approche des pistes d'atterrissage.

A noter aussi, d'autres fonctions disponibles :

- Calcul de la distance entre deux avions.
- Changement d'échelle de la représentation à l'écran.

La fonction de **tracking** (poursuite), qui réalise le traitement des informations radar, marque une nouvelle phase dans l'automatisation. En effet, les informations fournies par le transpondeur et les échos de réponse aux radars primaire et secondaire (généralement coaxiaux et corotatifs) ne concernent que l'altitude (pour le premier) et la distance (pour les seconds).
Le tracking est la fonction de corrélation de ces données dans le temps pour en déduire une **valeur de vitesse** (intensité, direction, côté) qui est ensuite représentée sur le PPI à l'aide d'un symbole (vecteur) associé au symbole d'identification de l'avion sur l'écran (photo ci-contre).
Le tracking est une fonction complexe, obtenue à partir des données radar, extrapolées avec un taux d'erreurs plus ou moins important. D'autres erreurs tiennent aux équipements et aux programmes quand, par exemple, un espace aérien est couvert par plusieurs





B. Liotta

Identification d'un avion grâce au transpondeur et au tracking vidéo.

radars placés en des points éloignés (multiradar tracking). Si les données en provenance des différents radars ne sont pas erronées, il reste la difficulté de les réduire à la même échelle dans le temps, pour en faire la synthèse.
Une fois calculées la position et la vitesse d'un avion, les données de sorties sont affichées sur le PPI ou réexploitées en entrées pour une opération particulièrement intéressante : l'**analyse des conflits.** En effet, il est possible, à



B. Liotta

Ci-dessus, déroulement d'un strip (fiche de progression) sur l'écran, comportant les divers FIX (balises radio).

Ci-dessous, contrôleur de vol à sa console.

partir des données cinématiques d'un avion, d'effectuer certaines prévisions concernant sa trajectoire, donc des risques de se retrouver à un point déterminé en même temps qu'un autre avion. Cette situation, dangereuse, entraîne le déclenchement d'une alarme et l'intervention urgente du contrôleur.

L'automatisation de cette phase de contrôle assure une aide immense aux contrôleurs et les aide beaucoup à améliorer la sécurité.

Une fois passés les contrôles, les avions volent à des intervalles de moins de 3 minutes, ce qui permet, en phase d'atterrissage, de décongestionner le trafic et de supprimer les ***files d'attente en altitude*** *(holding), très coûteuses pour les compagnies (l'heure peut atteindre 25 000 francs pour un Boeing 747).*

D'autres aspects du service ont également été automatisés pour soulager les contrôleurs aériens. C'est le cas de l'interprétation des plans de vol notamment, et de l'impression des fiches de progression.

Il a donc fallu ajouter, au système, des fonctions et des bases de données qui facilitent le décodage des informations.



S. Senn/EGS

L'une des principales **bases de données** contient la **description géographique** complète des différentes zones de circulation aérienne intégrant les routes aériennes, tous les FIX désignés par leurs noms et leurs coordonnées, les parcours d'approche des pistes des aéroports ainsi que les parcours de décollage, les paramètres des pistes, les caractéristiques concernant le trafic sur les différents segments de route, la liste des régions de vol limitrophes et les FIX correspondants.

Ces informations doivent être facilement modifiables pour tenir compte des aménagements apportés, des ouvertures de nouvelles routes aériennes, etc.

Une autre base de données porte sur les courriers consignant sur 6 mois (il s'agit des RPL, Repetitive Plan) la route et les horaires prévus. Ces prévisions constituent près de 50% des vols quotidiens à contrôler. La base de données est balayée à intervalles réguliers (en général toutes les heures) pour détecter l'entrée d'avions dans la région de vol.

Les plans de vols des 50% restants — hors RPL, donc — sont fournis par le réseau AFTN, connecté à l'ordinateur. Le système lit les télex, les interprète et corrige les éventuelles erreurs. Les messages non corrigés sont présentés à l'opérateur sur l'écran, pour qu'on y apporte les modifications nécessaires (voir photo p. 1517).

Les RPL et les plans de vol AFTN servent d'entrées à une deuxième fonction : la **reconstitution automatique de la route.**

Celle-ci doit, à partir des données géographiques, sélectionner, dans l'ensemble de la route du plan de vol, la partie concernant le contrôleur de la région et la traduire en une suite de FIX à survoler.

Le recours à d'autres bases moins importantes (vitesses et caractéristiques des avions) et aux indications horaires portées dans les plans de vol permet de calculer, pour chaque FIX, le temps de survol prévisible.

Les résultats sont de nouveau traités par la fonction qui pilote les imprimantes (production des fiches de progression tout au long de la journée).

La troisième phase d'automatisation consiste à réunir les deux niveaux précédents en un seul système. Les informations proviennent simultanément **en direct** (du radar) et **par prévision** (des routes déduites du plan de vol). Ce dernier pas n'est possible qu'à condition de disposer d'un gros ordinateur capable d'effectuer des calculs à très grande vitesse.



S. Senni/E.G.S

Salle des opérations de contrôle aérien de Rome-Ciampino.

Certaines informations importantes, comme les temps de survol des FIX théoriques et réels et les prévisions de route, sont présentées aux contrôleurs sur des écrans spéciaux appelés EDD (Electronic Data Display) et placés à côté de l'écran radar afin de compléter les images du PPI (voir graphique p. 1517).

En reliant la base de données radar à celle des routes reconstruites, on peut comparer la position effective de l'avion à sa position théorique d'après la route déclarée, ce qui permet, par conséquent, de déclencher une **alarme en cas d'incohérence.**

Entrées du tracking et vitesse sont ensuite visualisées sous forme numérique sur les EDD, à côté des données déduites du plan de vol, pour fournir au contrôleur, et de façon compacte, le plus grand nombre d'informations.

Cette troisième phase d'automatisation ouvre un champ d'application extrêmement vaste, avec notamment une gamme de commandes permettant d'interroger le système.

Le contrôleur peut ainsi modifier les bases de données pour tenir compte de mises à jour des plans de vol, demander un nouveau calcul des routes particulières et réimprimer les fiches de progression.

La sécurité apportée par l'automatisation a poussé de nombreux pays à adopter le système de CTA.
En Europe, c'est l'échange d'informations entre ordinateurs de différents pays qui assure une nouvelle dimension au contrôle de la navigation aérienne.
L'OACI ou ICAO (International Civil Aviation Organization) a déjà prévu des protocoles pour l'échange normalisé de messages entre ordinateurs, le but étant de construire un réseau informatique spécialisé qui remplacerait l'ancien AFTN.
De telles ***interconnexions*** *pourraient étendre le concept de régulation de la navigation à l'****ensemble du trafic européen*** *en coordonnant les éventuelles modifications de route vers des destinations autres que celles initialement prévues (et non plus en se bornant à chercher une solution à l'intérieur d'une seule FIR ce qui n'est pas toujours aisé, surtout pour des raisons relevant de la météo).*
Ce projet ambitieux, n'a pu, jusqu'ici, être réalisé qu'aux Etats-Unis où une organisation unique, la Federal Aviation Administration, coordonne le contrôle des 51 états.
Du point de vue de la configuration du système, les automatisations radar nécessitent l'usage de ***mini-ordinateurs*** *(les 16 bits peuvent suffire), sans trop de mémoire de masse. En effet, le travail de l'ordinateur est limité aux calculs sur un nombre de données restreint, les données radar étant déjà traitées une première fois par des équipements spécialisés. Le traitement des plans de vol et des plans répétitifs (RPL) n'exige pas non plus de gros ordinateurs, mais d'****importantes bases de données*** *sont indispensables pour stocker les informations géographiques, les caractéristiques des avions et les plans de vol.*
Si l'analyse des RPL et l'impression des fiches de progression sont réalisées séparément du traitement des données radar, il est possible de stocker l'ensemble des données sur disque, car les temps d'E/S ne sont pas critiques.
En revanche les niveaux de contrôle intégré requièrent des sytèmes plus complexes pour augmenter la vitesse de calcul et la capacité mémoire.
Ils constituent une application typique du ***traitement en temps réel,*** *puisqu'ils doivent acquérir périodiquement les tracés et les traiter de telle sorte qu'ils soient présentés de façon synchronisée avec la rotation des radars (moins de 10 secondes par rotation).*
Pour cela, les performances des ordinateurs doivent atteindre 1 Mips (million d'instructions par seconde).
Pour réduire les temps morts d'E/S, les bases de données sont directement chargées en mémoire au démarrage du système, même si cela implique une occupation de 1 à 2 Moctets en mémoire centrale. C'est d'ailleurs le ***coût des mémoires*** *qui explique le retard pris au cours des décennies passées dans le développement de ces systèmes.*
Aujourd'hui, on s'achemine vers une gestion, par l'ordinateur central de la base contenant les données dynamiques sur le trafic, du dialogue avec le contrôleur et du tracking.
C'est un réseau de mini-ordinateurs qui établit les plans de vol, les fiches de progression et commande l'affichage sur les PPI et les EDD.
Les éléments du système sont dédoublés pour éviter les risques de pannes, toujours possibles. Elles sont de trois sortes : pannes d'alimentation, pannes de matériel et pannes des logiciels.
Le premier type couvre les coupures d'électricité. On a alors recours à des groupes électrogènes (eux aussi dédoublés) complétés de batteries de secours pour suppléer au temps mort entre l'interruption de l'alimentation secteur et le démarrage des groupes.
Il suffit, en effet, de quelques dixièmes de seconde pour perdre les informations contenues dans les mémoires.
Le deuxième type de panne concerne l'ordinateur. Le redémarrage est généralement plus long (environ 1 minute). Pour qu'aucune information entrée par les contrôleurs ne soit perdue, une ***copie sur disque*** *des bases de données contenues dans la mémoire centrale est effectuée lors de l'envoi de chaque commande.*
Quand un ordinateur primaire tombe en panne, l'ordinateur secondaire (qui peut accéder aux mêmes disques) en est informé et charge, lors de son démarrage, toutes les informations relatives au trafic en cours.
En cas de panne des logiciels, les informations sont également enregistrées sur disque.
En Italie, la mise au point d'un système de contrôle aérien automatisé remonte à la fin des

années 60, date de la commande, par les militaires, du système ATCAS (Air Traffic Control Automatic System), résultat de la collaboration entre Selenia et IBM, qui devait permettre de contrôler la région aérienne de Rome.
Le programme de développement prévoyait la fourniture en 10 ans (de 1975 à 1985) de tous les niveaux d'automatisation nécessaires. A partir de 1980, la société Datamat Ingegneria dei Sistemi, de Rome, s'est jointe au consortium pour concevoir les logiciels des derniers niveaux d'automatisation.
Dans sa configuration finale, le système prévoit l'emploi d'un gros ordinateur central du type IBM 3033, qui a une vitesse d'exécution de 4,5 mips. Il est connecté à deux GP-16, à un CDG (Selenia) et à un deuxième IBM 3033 servant d'« esclave » (graphique ci-contre).

Les principaux périphériques gérés par l'ordinateur central sont les suivants :

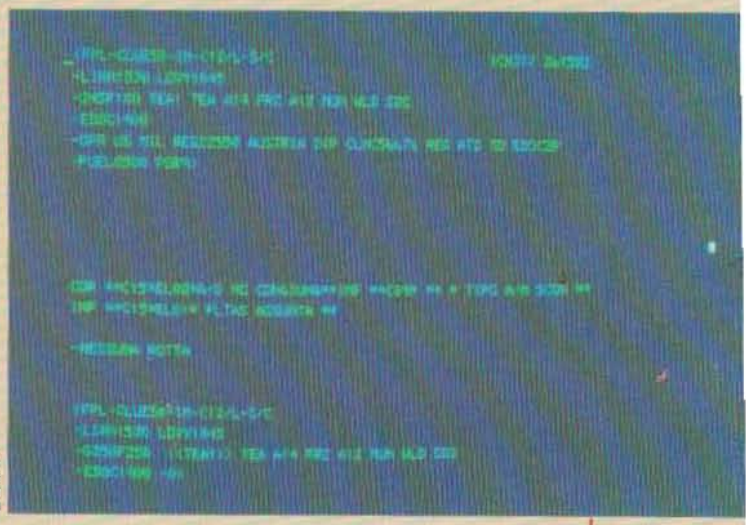
- ***Unités de disque** de 400 Moctets chacun pour enregistrer la situation en cours et les « repetitive plans »*

- ***Dérouleurs de bande magnétiques** pour l'enregistrement légal de toutes les communications entre les divers contrôleurs et le système*

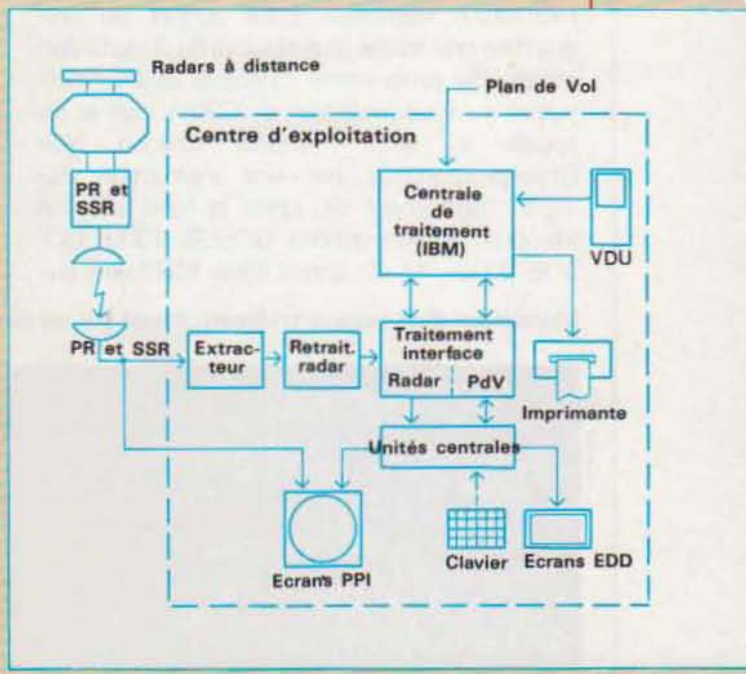
- ***Imprimantes** permettant d'imprimer les fiches de progression et de conserver une copie papier des télex reçus du réseau AFTN*

- ***Unités vidéo** de correction des messages télex et d'entrée des plans de vol ou des repetitives plans*

De plus, l'ordinateur est ***directement connecté au réseau AFTN*** *afin de recevoir l'ensemble des messages concernant les plans de vol.*
La gestion des unités de visualisation (PPI/EDD) dont disposent les contrôleurs est effectuée par les GP-16, qui reçoivent de l'ordinateur central les données traitées. Les mini-ordinateurs font également fonction d'interface entre les opérateurs et le maître en gérant les claviers ordinaires ou les dispositifs d'entrée rapides.



B. Liotta



En haut : télex transmis par l'AFTN. En-dessous, schéma du système de collecte des données.

En cas de panne, qu'elle soit matérielle ou logicielle, de l'IBM 3033, il est prévu que le CDG assurera l'étiquetage des données radar pendant la minute nécessaire au redémarrage automatique.

Umberto Corà et Bruno Liotta,
Datamat Ingegneria dei Sistemi, Rome.

L'écran couleur permet d'associer une couleur différente à chaque mode de représentation. On trace tous les histogrammes sous forme de colonnes pleines dont seule la couleur varie. Avec l'écran monochrome, il faut remplir différemment les colonnes.

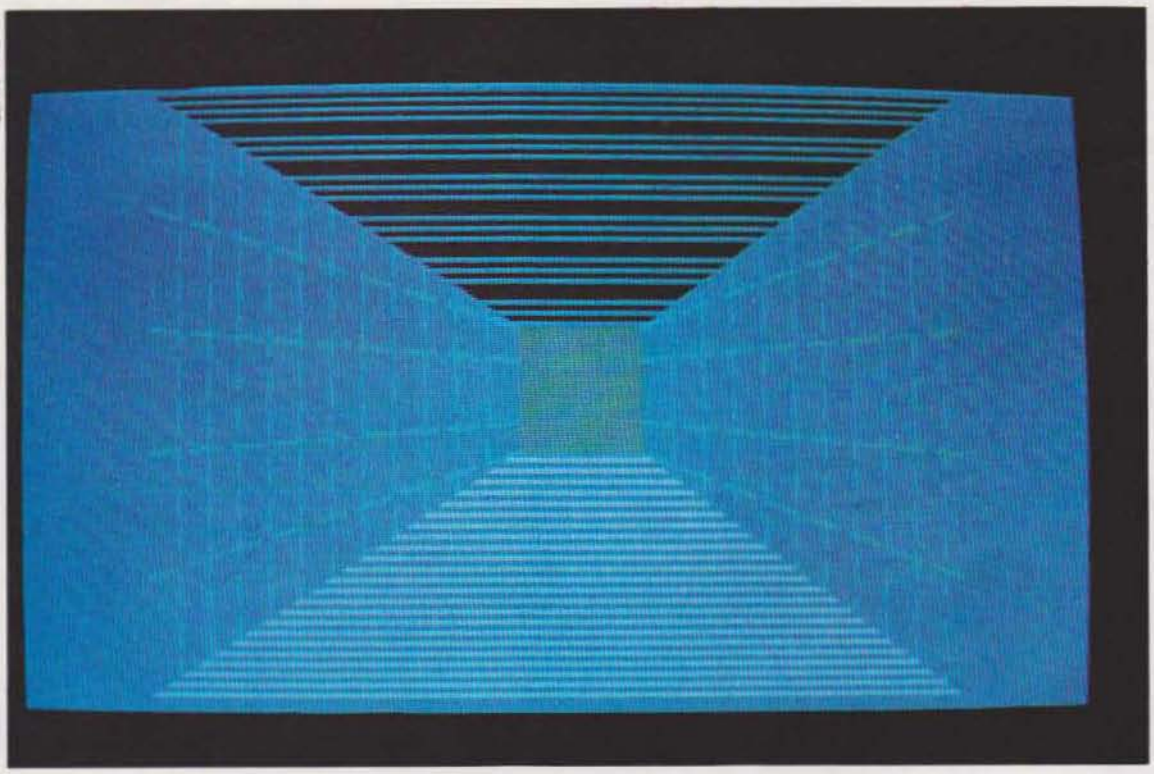
Le programme est divisé en deux parties : la première est consacrée à l'entrée des données, la seconde à leur présentation. L'entrée est insérée dans le programme principal, pour autoriser toutes les adaptations.

La présentation comporte des sous-programmes spécialisés chacun dans une tâche spécifique. La subdivision en sous-programmes ne se retrouve pas dans le listing car les fonctions se suivent : mais les organigrammes (pages 1519-1522) reprennent la numérotation habituelle. Pour donner au programme une forme plus structurée, il faut interrompre le programme principal avant l'exécution du sous-programme 1000, y insérer les appels et enfin terminer chaque bloc (sous-programme) par une instruction RETURN. Autrement dit, après la ligne 380, on introduit les instructions GOSUB 1000, GOSUB 2000... tandis que la ligne 1180 sera suivie de RETURN (de même que la 2085, etc.). Chaque bloc est alors considéré comme un sous-programme.

Le programme contient également les sous-programmes de visualisation des caractères alphanumériques ; les données correspondantes sont initialisées à partir des lignes 60 et 65. Ensuite vient la phase d'entrée des données, au cours de laquelle est effectué un contrôle de validité. Si on a choisi d'entrer les valeurs en pourcentage, aucun chiffre ne peut dépasser 100 (100 %) pas même la somme des valeurs entrées (lignes 210). Noter que le sous-programme 6000 contient simplement l'affichage du diagnostic. La sortie du programme est réalisée en entrant un nombre de données égal au maximum prévu (NM = 20, ligne 20) ou une valeur négative. La limite maximale de 20 données, correspondant à 20 colonnes à répartir sur l'axe X, est due aux dimensions de l'écran utilisé et aux formes particulières de remplissage des histogrammes.

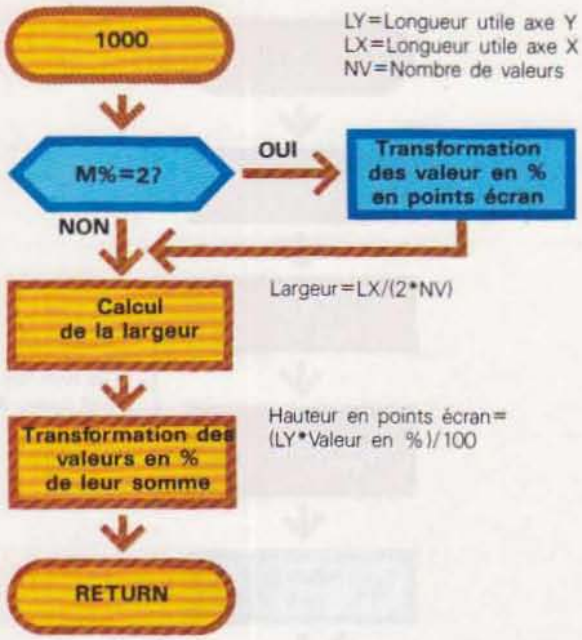
Le transfert de ce programme sur d'autres machines nécessite certaines modifications d'instructions et notamment :

Simulation d'un espace tridimentionnel sur un moniteur couleur.

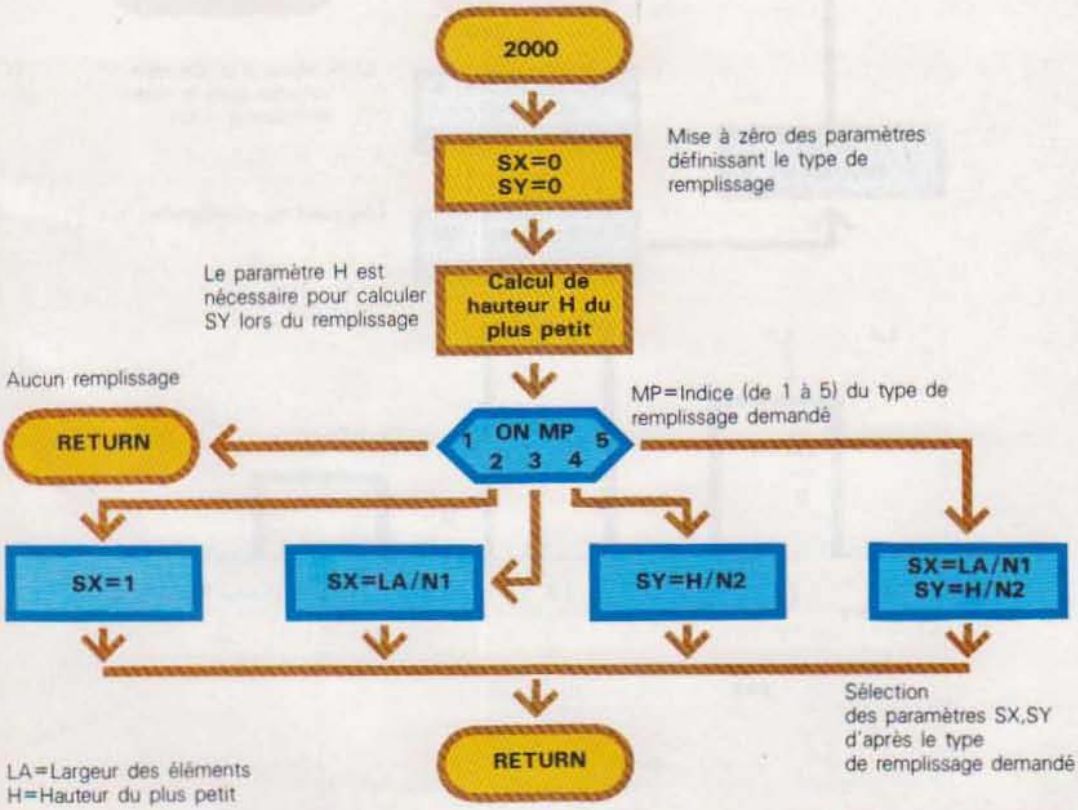


Centro THC

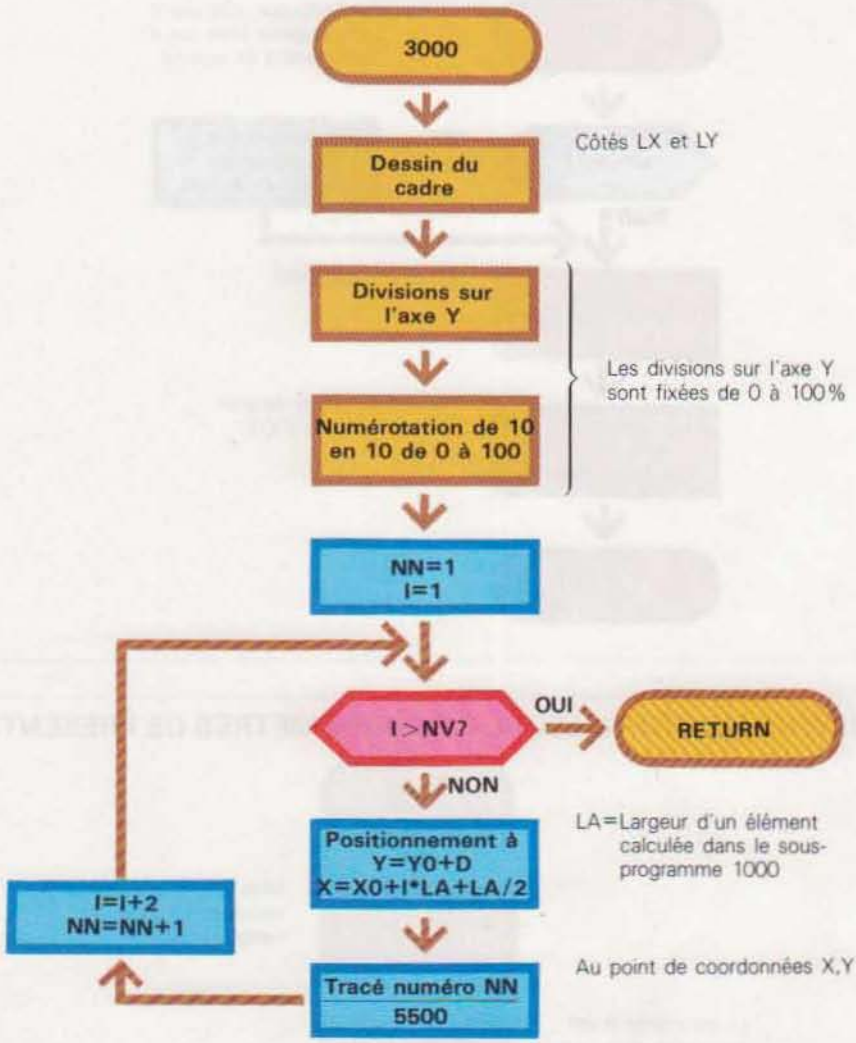
1/ TRAITEMENT DES DONNEES POUR L'HISTOGRAMME



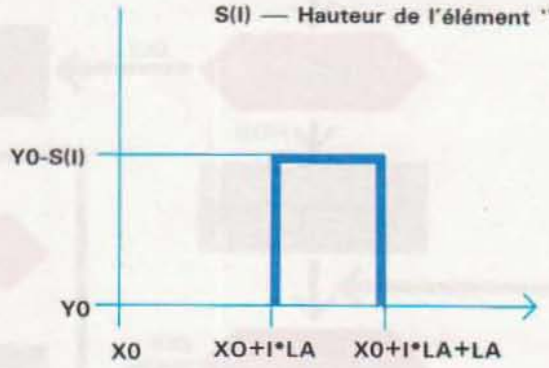
2/ ASSIGNATION DE VALEURS AUX PARAMETRES DE PRESENTATION

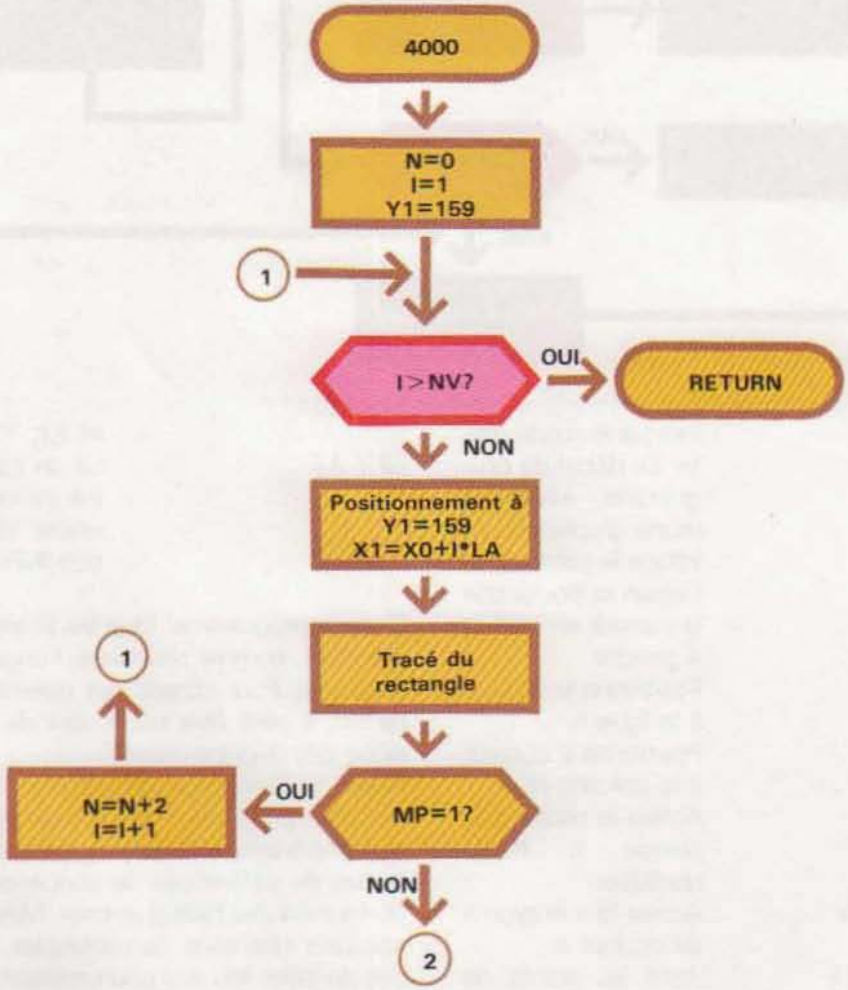


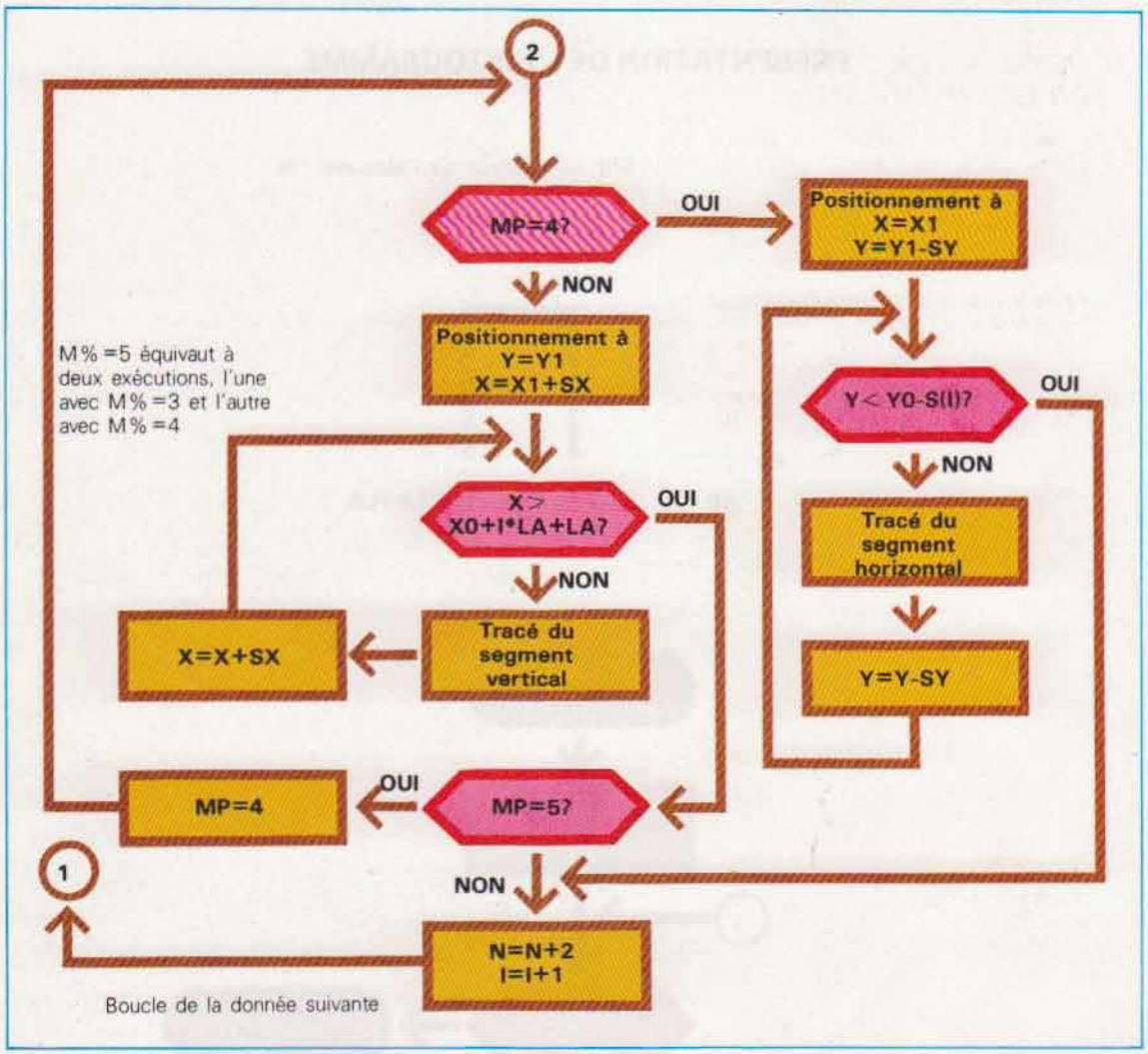
PRESENTATION DE LA FENETRE



PRESENTATION DE L'HISTOGRAMME







TEXT : Indique le mode texte. En début de programme, inhibe le mode graphique.

HOME: Efface le contenu de l'écran et positionne le curseur en haut et à gauche

VTAB n : Positionne le curseur à la ligne n

HTAB m : Positionne le curseur à la colonne m

HGR2 : Active le mode graphique à haute résolution

HCOLOR = n : Active le « crayon » de couleur n

HPLOT X1, Y1 TO X2, Y2 : Joint les points de coordonnées X1, Y1 et X2, Y2.

GET A$: Lit un caractère entré au clavier. Equivalent de l'instruction INPUT $ (1).

Dans le programme, tous les positionnements prennent, comme référence, l'origine, en haut à gauche. Pour obtenir des orientations différentes, il peut être nécessaire de modifier le signe des déplacements.

Pages 1525 et 1526, on trouvera l'organigramme d'un programme de visualisation de diagramme à secteurs. Ce programme ne présente pas de différences de conception par rapport à celui des histogrammes. Mais, au lieu de visualiser une série de rectangles aux valeurs des données (ou aux pourcentages de ces valeurs par rapport au total), le programme doit

REPRESENTATION D'HISTOGRAMMES

Quelques phases de fonctionnement du programme de représentation des histogrammes.



Après la question de la ligne 80, à laquelle l'utilisateur a répondu en indiquant qu'il entrerait des données absolues (et non des pourcentages), le programme est passé à la phase d'entrée des données à représenter. Ici, l'utilisateur a entré la valeur 50 comme première donnée. Elle doit être marquée à l'aide de l'étiquette JAN.

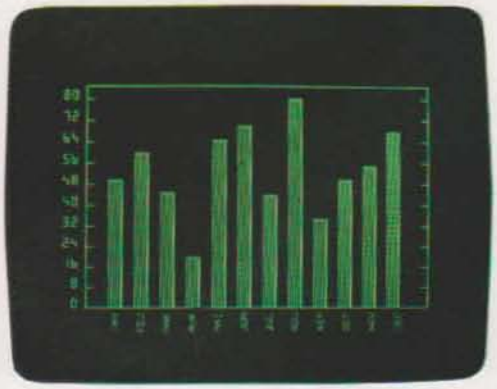
Les lignes 310 à 360 provoquent, au terme de la phase d'entrée, l'affichage du menu des différentes formes de représentation.
L'opérateur à choisi la représentation à colonnes remplies de lignes verticales.



Le programme demande ensuite à l'utilisateur s'il désire représenter les données sous forme de pourcentage.
La réponse est négative.

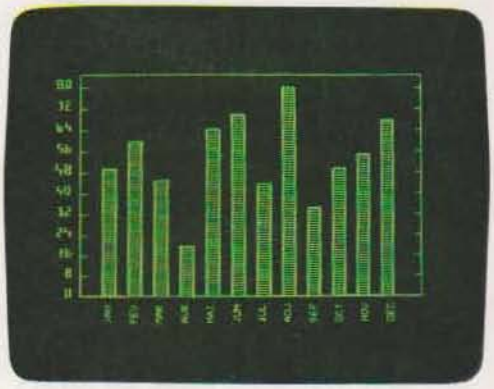


Le programme a activé la représentation en lignes verticales des valeurs absolues entrées.

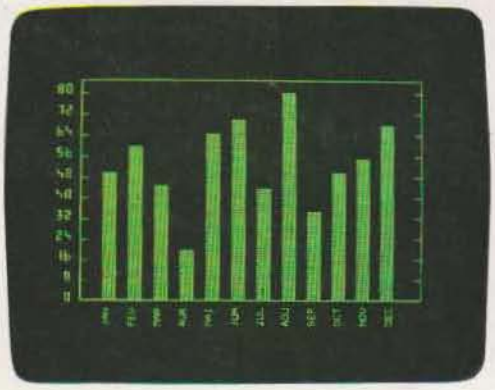


L'écran ci-contre et les 3 autres illustrent les différents modes de visualisation.

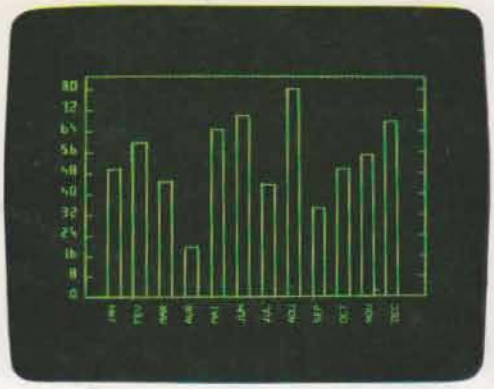
Lignes horizontales...



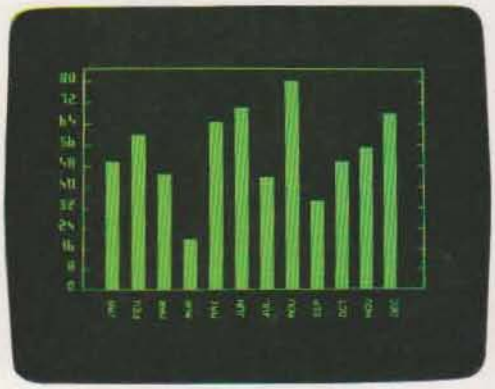
Quadrillage...



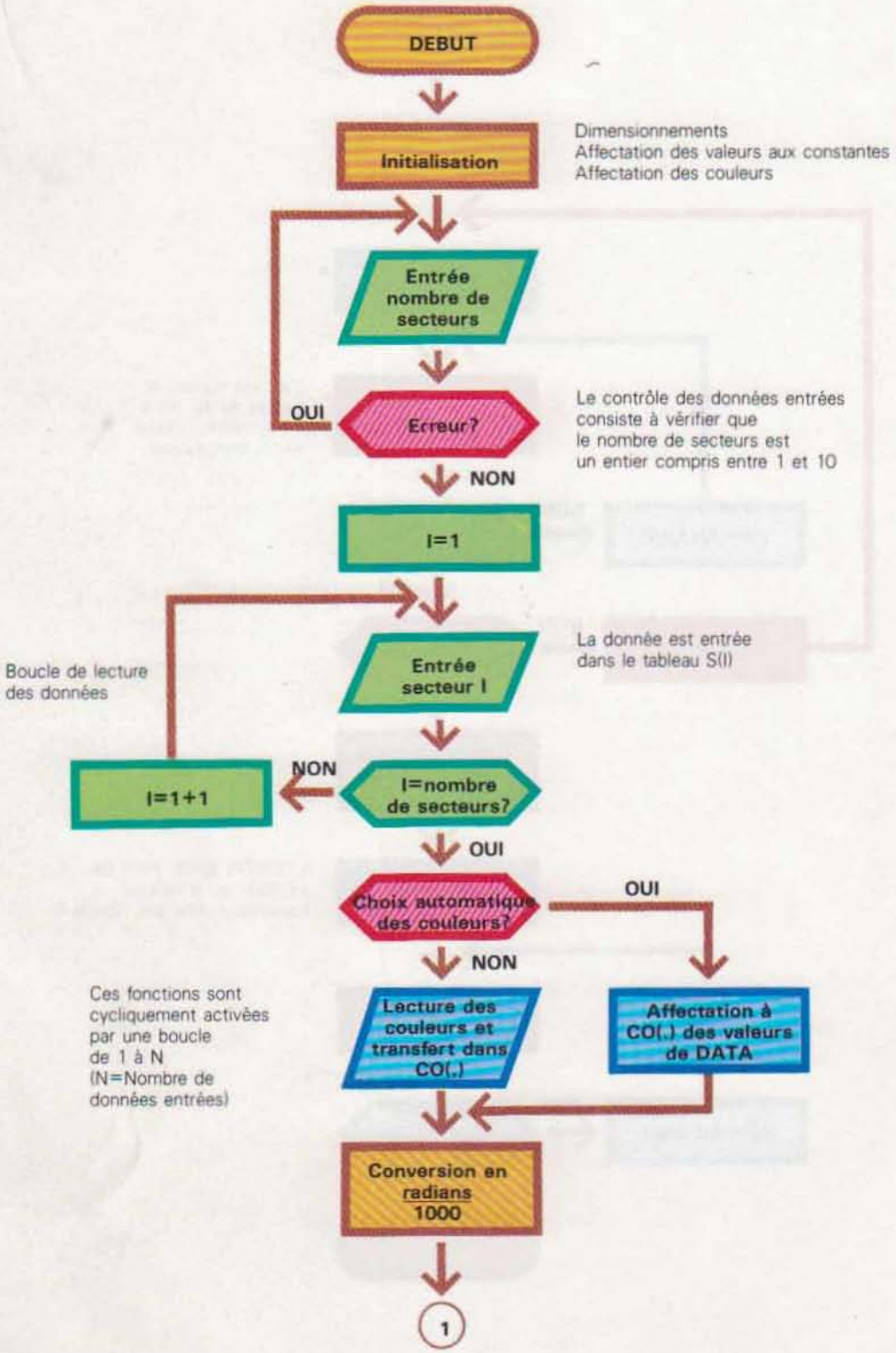
Colonnes vides...

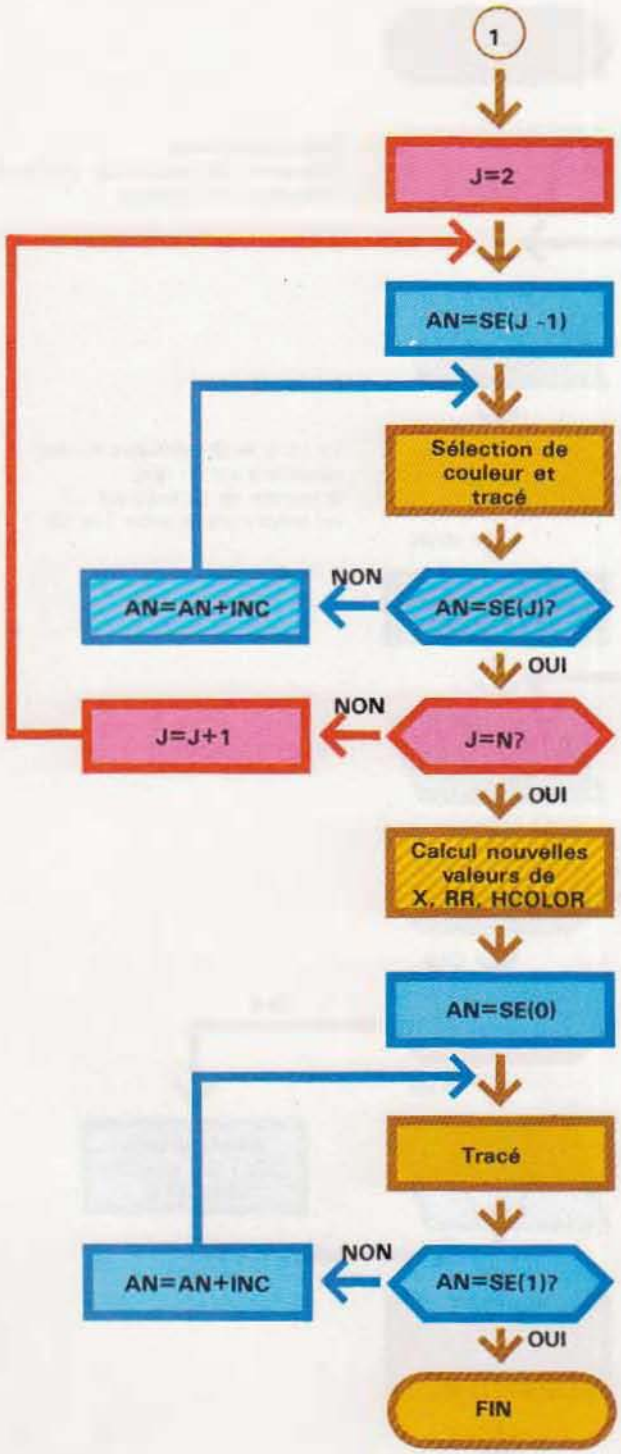


Colonnes pleines...



PROGRAMME DE PRESENTATION DE DIAGRAMMES SECTEURS




1
J=2
AN=SE(J -1)
Sélection de couleur et tracé
L'arc est obtenu en traçant de Xc, Yc à
X=Xc+RR*COS(AN)
Y=Yc-RR*SIN(AN)
AN=AN+INC
NON
AN=SE(J)?
OUI
J=J+1
NON
J=N?
OUI
Calcul nouvelles valeurs de X, RR, HCOLOR
AN=SE(0)
L'OPTION BASE 1 n'a pas été prévue ; le tableau commence donc par l'indice 0
Tracé
AN=AN+INC
NON
AN=SE(1)?
OUI
FIN

VISUALISATION DE DIAGRAMMES A SECTEURS

```
10 REM ---------------------
20 REM DIAGRAMMES A SECTEURS
30 REM ---------------------
40 REM
50 REM
60 XC=130
   :YC=90
   :R=90
   :PI=3.1415926
   :INC=PI/300
70 DIM SE(10),C(10),CO(10),S(10)
80 FOR I=1 TO 10
   :READ C(I)
   :NEXT
90 DATA 5,6,7,1,2,3,5,6,7,1
100 REM
110 REM ------------------
120 REM ENTREE DES DONNEES
130 REM ------------------
140 REM
150 HOME
160 VTAB 4
    :INPUT "Nombre de secteurs (max. 10) "
    ;A$
170 A=VAL(A$)
180 IF A<1 OR A>10 OR INT(A)<>(A) THEN 150
190 VTAB 6
    :PRINT "Entrez les données (nombres
     positifs) de chaque secteur."
200 N=A
    :TT=0
210 FOR I=1 TO N
220 VTAB (10+I)
    :PRINT "Secteur"I;
    :INPUT ": ";A$
230 S(I)=VAL(A$)
    :IF S(I)<0 THEN 220
240 TT=TT+S(I)
    :S(I)=TT
250 NEXT
260 HOME
270 VTAB 4
    :PRINT "Le diagramme contient 'n'
     secteurs."
280 REM
290 REM ----------------------
300 REM SELECTION DES COULEURS
310 REM ----------------------
320 REM
330 VTAB 6
    :INPUT "Voulez-vous choisir vous-même
     les couleurs ou que l'ordinateur s'en
     charge (1/2) ";A$
340 IF A$<>"1" AND A$<>"2" THEN 260
350 IF A$="1" THEN 390
360 FOR I=1 TO N
    :CO(I)=C(I)
    :NEXT
370 IF CO(N)=5 THEN CO(N)=1
380 GOTO 550
390 HOME
400 VTAB 4
    :PRINT "Choisissez entre vert, rouge,
     orange, bleu et blanc en tapant <V>,
     <R>, <A>, <B> ou <W>."
410 FOR I=1 TO N
420 VTAB (6+I)
    :PRINT "Couleur du secteur"I;
    :INPUT ": ";A$
430 IF A$="V" THEN CO(I)=1
    :GOTO 490
```

```
440 IF A$="R" THEN CO(I)=2
    :GOTO 490
450 IF A$="A" THEN CO(I)=3
    :GOTO 490
460 IF A$="B" THEN CO(I)=4
    :GOTO 490
470 IF A$="W" THEN CO(I)=5
    :GOTO 490
480 GOTO 420
490 NEXT
500 REM
510 REM ------------------------
520 REM PRESENTATION DU SECTEUR
530 REM ------------------------
540 REM
550 HOME
560 VTAB 4
    :INPUT "Voulez-vous faire ressortir le
     secteur 1 (O/N) ";A$
570 FL=0
    :IF LEFT$(A$,1)="O" THEN FL=1
580 GOSUB 1000
590 REM
600 REM --------------------
610 REM DESSIN DU DIAGRAMME
620 REM --------------------
630 REM
640 HGR2
650 FOR J=2 TO N
660 FOR AN=SE(J-1) TO SE(J) STEP INC
670 HCOLOR=CO(J)
680 HPLOT XC,YC TO XC+R*COS(AN),YC-R*SIN(AN)
690 NEXT AN,J
700 X=XC+15*FL
    :HCOLOR=CO(1)
    :RR=R-8*FL
710 IF SE(1)>.5 THEN RR=R-3*FL
    :X=XC+12*FL
    :IF SE(1)>1 THEN RR=R
    :XC=XC+9*FL
720 FOR AN=SE(0) TO SE(1) STEP INC
730 HPLOT X,YC TO X+RR*COS(AN),YC-RR*SIN(AN)
470 NEXT
750 END
1000 REM
1010 REM ----------------------
1020 REM CONVERSION EN RADIANS
1030 REM ----------------------
1040 REM
1050 IF TT=360 THEN 1090
1060 FOR I=1 TO N
1070 S(I)=S(I)/TT*360
1080 NEXT
1090 FOR I=1 TO N-1
1100 SE(I)=S(1)-S(1)/2
1110 NEXT
1120 SE(0)=-S(1)/2
     :SE(N)=360+SE(0)
1130 FOR I=0 TO N
1140 SE(I)=SE(I)/180*PI
1150 NEXT
1160 RETURN
```

visualiser un cercle divisé en un nombre de secteurs égal à celui des données à représenter. Il attribue, à chaque secteur, une surface proportionnelle à la donnée représentée.

Le programme (listing pages 1527 et 1528) prévoit l'emploi d'un écran couleur. Les couleurs des secteurs du diagramme sont automatiquement choisies par le programme, sauf

DIAGRAMMES A SECTEURS

Le programme (ligne 160) demande à l'utilisateur combien de secteurs (de données) il doit représenter.
L'entrée du nombre de secteurs (ici 5) est suivie de la phase de demande des données (lignes 190 à 250).



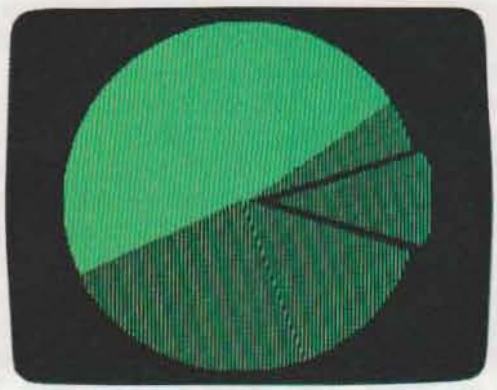
Au terme de l'entrée des données, le programme indique, à l'utilisateur, le nombre de données entrées (ligne 270) et lui demande des précisions sur les choix de couleurs (ligne 330).



L'utilisateur ayant opté pour la sélection automatique des couleurs, le contrôle passe à la ligne 560, qui demande s'il faut mettre en valeur le premier secteur.
La réponse est affirmative.



Présentation du diagramme à secteurs. Le secteur "découpé" est le premier ; les autres suivent dans le sens inverse des aiguilles d'un montre.



indication contraire de l'utilisateur. Seule particularité importante : le sous-programme 1000, qui convertit les valeurs entrées par l'utilisateur en valeurs angulaires (grandeurs des secteurs) exprimées en radians. Pour le reste, les instructions sont analogues à celles utilisées pour les histogrammes.

Gestion de la mémoire en mode graphique

L'écran est l'unité de sortie la plus courante dans les applications graphiques. Or ce dispositif ne garde pas en mémoire les images qui apparaissent. La visualisation d'un texte ou d'un dessin, possible à l'aide d'une seule instruction (PRINT, LINE, etc.) demande, en réalité, une activité continue de régénération de la même image, à des intervalles très courts, afin de réactiver chacun de ses points. C'est pourquoi l'image graphique est rangée dans une zone mémoire spéciale.

A des intervalles réguliers, les programmes du système prélèvent le contenu de cette zone pour le transférer sur l'écran en redessinant continuellement le graphique. Ceci vaut également pour un texte en Basic ou dans tout autre langage. Il faut donc disposer d'une zone de mémoire qui conserve l'état de chaque point de l'écran. La dimension de cette mémoire varie en fonction du type d'écran et du système d'exploitation.

Si l'écran est monochrome, un seul bit par point suffit : à 0, il signifie que le point n'est pas visible ; à 1, que le point est activé.

Avec les écrans couleur, l'extension de la mémoire spéciale est plus grande car, pour chaque point, en plus de l'état (ON/OFF), il faut définir la couleur. Pour l'affichage d'un dessin, on procède donc en deux étapes : chargement de l'image binaire en mémoire graphique, activation du processus de présentation.

Par exemple, les instructions de tracé d'un segment entre 2 points (LINE, HPLOT, etc.) calculent, à partir des coordonnées en points écran, les positions mémoire correspondantes. Pour cela, elles mettent à l'état ON tous les bits relatifs à ce segment. Dans une machine 8 bits, chaque position mémoire contrôle un maximum de 8 points écran. Si la valeur 1 est écrite dans l'une d'elles, le premier des points contrôlés est activé ; par contre, avec la valeur 3, deux points contigus (1 + 2) le seront, et ainsi de suite jusqu'à la valeur maximale prévue.

La qualité graphique d'un dessin dépend fortement de la résolution de l'écran. Plus le nombre de points est élevé, plus la zone mémoire nécessaire est importante ; aussi doit-on, même dans le cas d'une haute résolution, réduire le nombre de points gérés pour éviter l'occupation d'un trop grand espace. Un écran de 280 positions horizontales par 190 positions verticales demande environ 53 000 indicateurs d'état correspondant chacun à un bit. L'occupation mémoire est donc d'environ 7,5 Koctets, en supposant que pour chaque position, seulement 7 bits parmi les 8 possibles sont employés.

L'existence de cette zone mémoire spéciale offre des facilités de programmation remarquables. Pour construire des figures d'une certaine complexité, on peut recourir à l'accès direct aux adresses mémoire sans être limité par les instructions graphiques. Inversement, on lira aisément le contenu de la mémoire pour connaître l'état de chaque point vidéo (allumé ou éteint) et pour traiter ou stocker le dessin. Avec cette méthode, on considère le graphique comme un tableau de valeurs numériques sur lequel on opère toutes sortes de traitements. La présentation à l'écran est soit automatique (au fur et à mesure des modifications du tableau) soit sur commande (quand le mode graphique est activé). Toutefois, cette gestion n'est pas propre à toutes les machines. Dans certaines d'entre elles, la mémoire graphique n'est accessible qu'au système ; il faut donc recourir à des instructions de haut niveau.

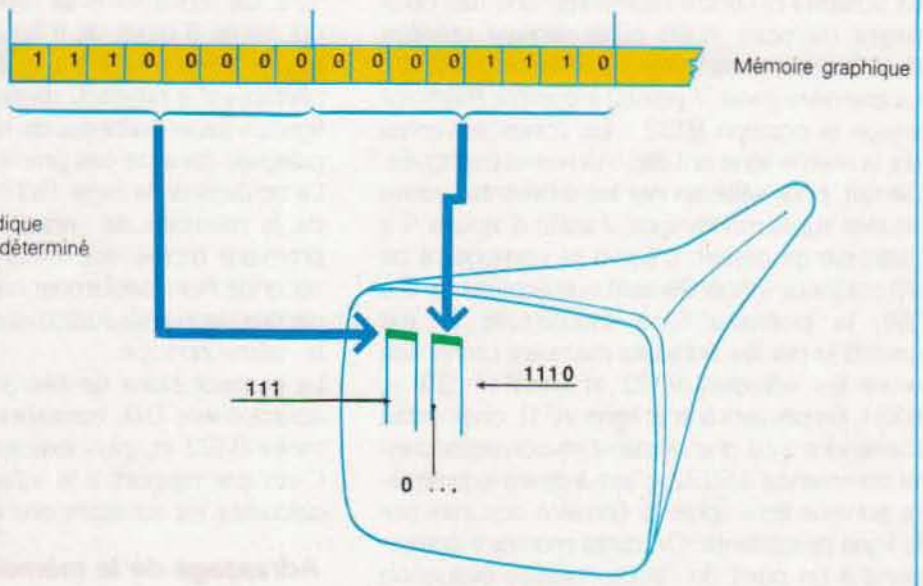
Pour d'autres machines, seule la connaissance de l'organisation interne de la mémoire permet de travailler directement sur l'image binaire.

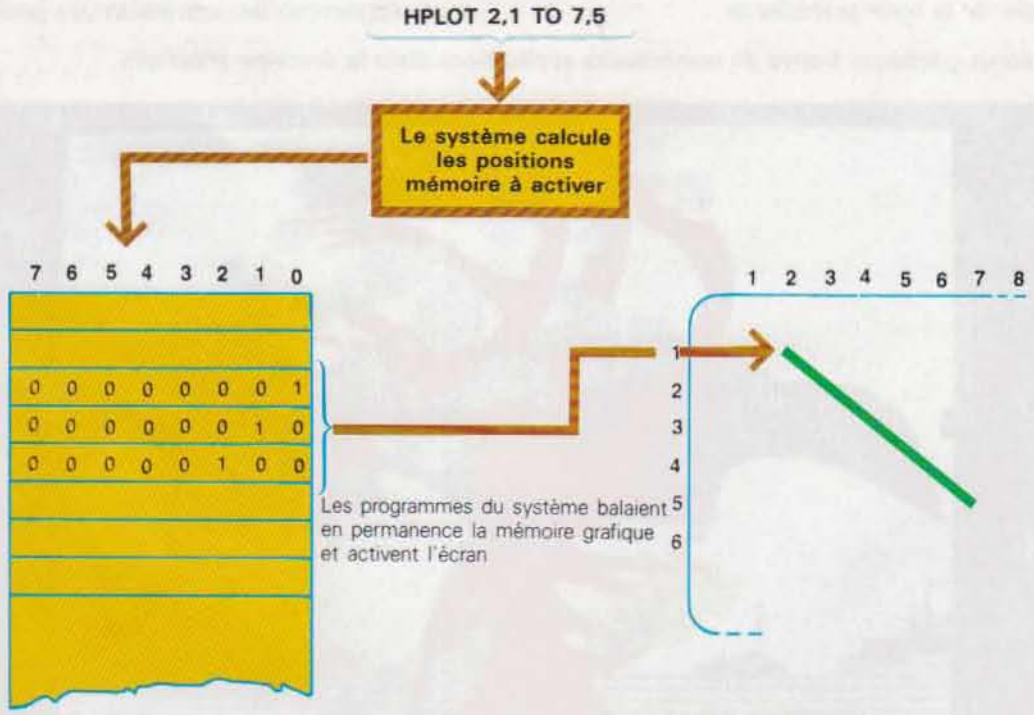
La mémoire graphique

La relation unissant un point écran à l'adresse mémoire qui le gère n'est ni simple, ni linéaire. Le mécanisme de gestion de l'écran est lié aux différents paramètres et, parfois, au type d'unité centrale utilisée. Nous allons étudier un exemple de partition de la mémoire couramment employé dans les ordinateurs personnels et dans les micros dotés d'une unité centrale Rockwell 6502 (machines du type Apple et compatibles).

La zone mémoire correspondant à l'écran n'est

GESTION SIMPLIFIEE DE L'ECRAN EN MODE GRAPHIQUE





pratiquement jamais constituée d'adresses contiguës, pour des raisons d'ordre essentiellement matériel. Ainsi, deux zones distantes de 100 positions ne correspondent pas à deux adresses ayant une différence de 100.
Le schéma ci-contre représente une des deux pages mémoire vidéo généralement utilisées dans les systèmes basés sur le Rockwell 6502. La première zone (7 points) a comme mémoire image la position 8192 ; les zones suivantes sur la même ligne ont des mémoires contiguës. De fait, pour sélectionner les différentes zones situées sur la même ligne, il suffit d'ajouter 1 à l'adresse de départ. L'écran se composant de 40 colonnes (généralement numérotées de 0 à 39), la première ligne (numérotée 0) est contrôlée par les adresses mémoire comprises entre les adresses 8192 et 8192 + 39 = 8231. En passant à la 2e ligne (n° 1), on pourrait s'attendre à ce que la mémoire correspondante commence à 8232, c'est-à-dire à la première adresse libre après la dernière occupée par la ligne précédente. Or, cette mémoire correspond à un point de l'écran localisé beaucoup plus bas. La 2e ligne (n° 1) commence en fait à l'adresse 9216, distante de 1024 de l'adresse initiale de la ligne précédente.

Comme pour les autres lignes, chaque zone de l'écran correspond à une mémoire contiguë à la précédente. L'incrément de 1024 reste vrai pour 7 lignes (nos 1 à 7), tandis que l'adresse de la 8e ligne est égale à celle de la 1re (n° 0) plus 128. Ce mécanisme se répète ainsi 7 fois, ce qui forme 8 blocs de 8 lignes chacun. La mémoire correspondant à la deuxième partie de l'écran est à nouveau divisée en 8 blocs de 8 lignes : seule l'adresse de la 1re position varie, puisque, dans ce cas précis, il s'agit de 8232. Le schéma de la page 1533 indique la partition de la mémoire de l'ensemble de l'écran. La première moitié est visualisée en totalité, la seconde étant seulement suggérée ; les mécanismes de numérotation des adresses suivent le même principe.
Le premier point de l'écran, identifié par les coordonnées 0,0, correspond à l'adresse mémoire 8192 et, plus exactement, à son bit 0. C'est par rapport à la valeur 8192 que sont calculées les adresses des autres points.

Adressage de la mémoire vidéo

Pour expliquer le mécanisme d'adressage, nous prendrons un exemple de calcul et un sous-programme de conversion des positions

Un écran graphique trouve de nombreuses applications dans le domaine artistique.



Centro THC

TOPOGRAPHIE DE LA MEMOIRE VIDEO : SCHEMA GENERAL

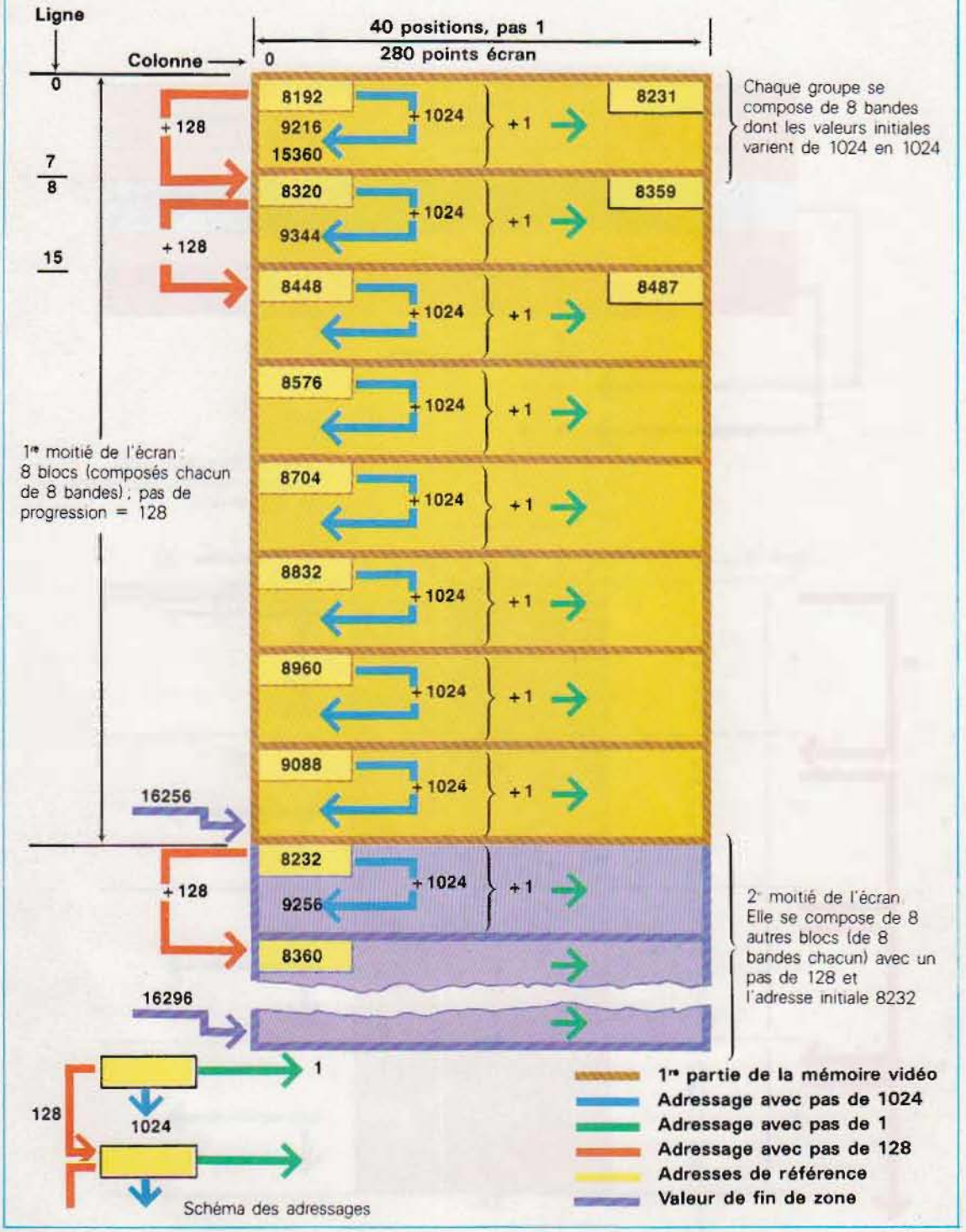


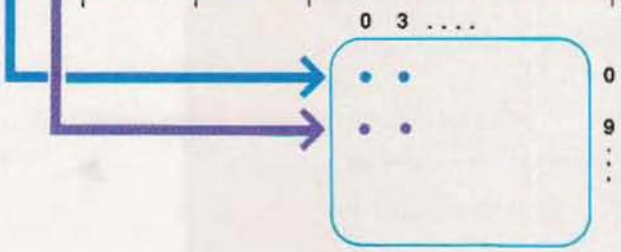
Schéma des adressages

mémoire en points écran, et inversement.

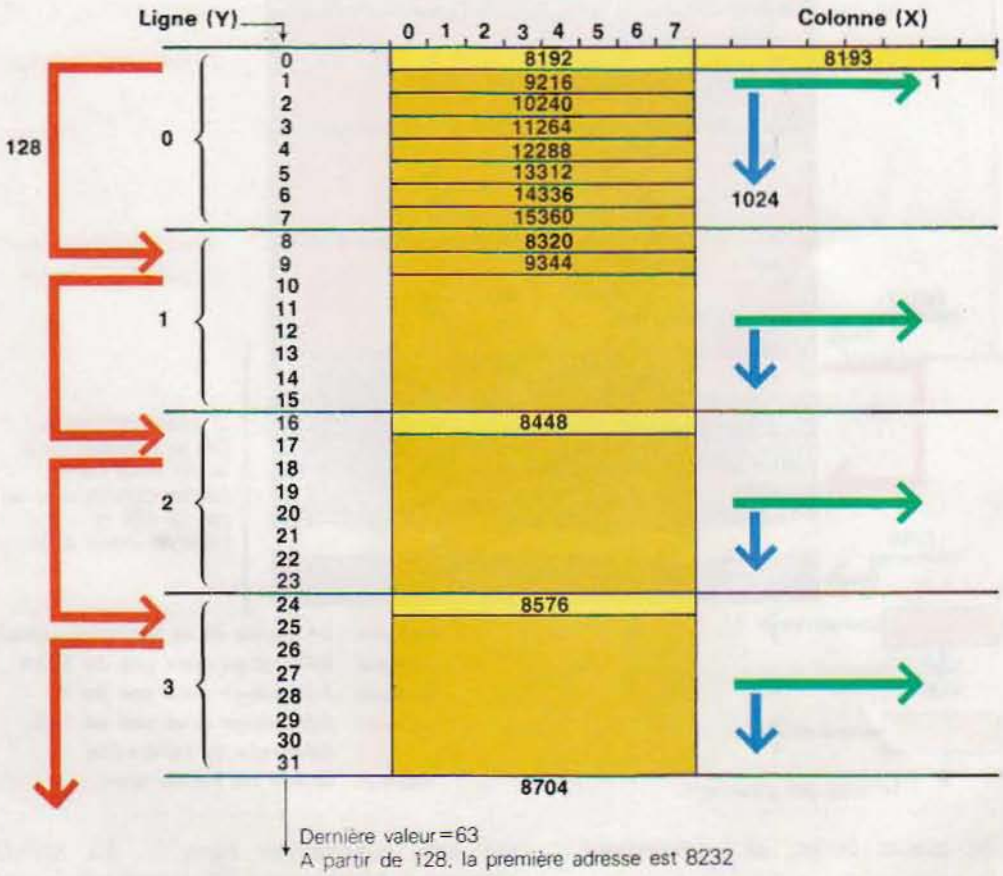
Adressage le long de l'axe. Soit l'adressage d'un point situé dans la colonne zéro (X = 0) sur une quelconque ligne Y. La variable d'entrée est l'ordonnée (Y), à partir de laquelle il faut trouver la position mémoire correspondante. Pour Y = 7, la position est 15360.

ADRESSAGE SUR L'AXE Y

Coordonnées X	Coordonnées Y	Positions mémoire	Contenu de la mémoire 0 1 2 3 4 5 6 7
3	0	8192	0 0 0 1 0 0 0 0
5	0	8192	0 0 0 0 0 1 0 0
3	9	9344	0 0 0 1 0 0 0 0
5	9	9344	0 0 0 0 0 1 0 0



Exemple de rapport entre l'adresse mémoire, son contenu et le point écran



Dernière valeur = 63
A partir de 128, la première adresse est 8232

La partie inférieure de la figure de la page 1534 reprend la partie de la figure précédente concernant les points ayant X=0. Pour des raisons de place, seuls les 4 premiers blocs sont représentés (Y compris entre 0 et 31) ; le mécanisme est identique pour les autres.
Pour déterminer la position initiale correspondant à une valeur de Y donnée, il faut :

A - Déterminer à quel bloc B (groupe de 8 lignes) elle appartient. B s'obtient en divisant Y par 8 (puisque chaque bloc contient 8 valeurs de Y) et en retenant la partie entière du résultat.

B - Déterminer l'adresse initiale du bloc B auquel Y appartient. La position initiale de chaque groupe variant de 128 en 128, l'adresse de B est donnée par la formule 8192+B*128.

C - Déterminer la position de Y dans B.

Chaque groupe commence par une valeur déterminée de Y (0, 8, 16, etc.) et se termine par une valeur +7 (puisqu'il contient 8 lignes, de 0 à 7). Une ligne Y quelconque occupe dans le bloc la position :Y-B*8.

En effet, B est son groupe d'appartenance, et le produit B*8 indique le nombre de lignes précédant la ligne Y et appartenant à d'autres blocs. La différence entre Y et cette dernière valeur fournit la position de la ligne Y dans le bloc B.

Sachant que le pas de progression dans le bloc est de 1024, pour trouver la position mémoire, il faut multiplier la valeur précédente (position de Y dans le bloc B) par 1024 et y ajouter l'adresse initiale du bloc B, calculé au point B. La deuxième moitié de l'écran, dont la mémoire commence à l'adresse 8232, peut être traitée de la même façon, en remplaçant évidemment la valeur initiale utilisée au point B par cette valeur.

TABLEAUX DES POSITIONS DE REFERENCE SUR L'AXE Y

1re partie

Base 8192 - Ordonnée (Y) entre 0 et 63

Bloc	Valeurs de Y de	à	Adresse initiale du bloc
0	0	7	8192
1	8	15	8320
2	16	23	8448
3	24	31	8576
4	32	39	8704
5	40	47	8832
6	48	55	8960
7	56	63	9088

Algorithme : Adresse mémoire=8192+128*Bloc
Bloc =INT (Y/8)

2e partie

Base 8232 - Ordonnée (Y) entre 64 et 127

Bloc	Valeurs de Y de	à	Adresse initiale du bloc
8	64	71	8232
9	72	79	8360
10	80	87	8488
11	88	95	8616
12	96	103	8744
13	104	111	8872
14	112	119	9000
15	120	127	9128

Algorithme : Adresse mémoire=8232+128*(Bloc-8)
Bloc =INT (Y/8)

Pour des raisons pratiques, on a mis en page 1535 deux tableaux, un pour chaque zone. Chacun d'eux fournit, en fonction de la valeur de Y, le bloc d'appartenance et la position initiale de celui-ci. Par exemple, la valeur Y = 35 appartient au bloc 4, qui commence à la position 8704. Il correspond à la position :

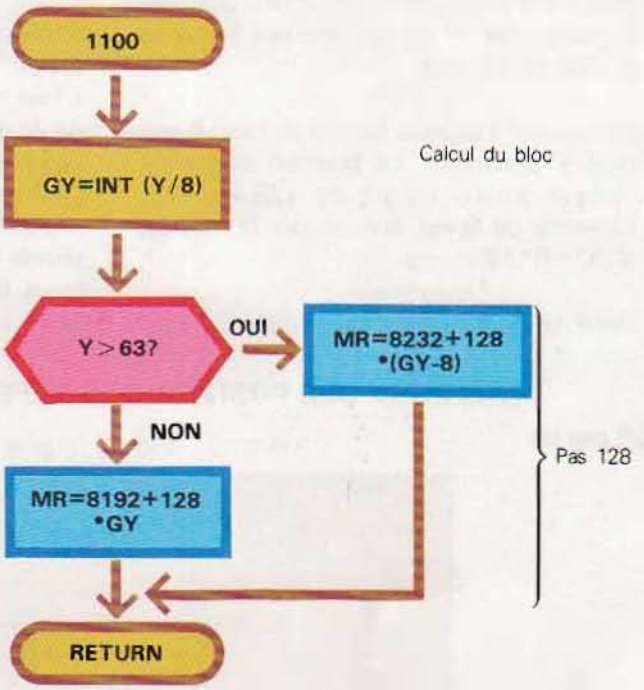
8704+1024*(35-4*8)
=8704+1024*3=11776.

Les organigrammes ci-dessous montrent le déroulement de deux sous-programmes qui, à partir d'une valeur Y trouvent l'adresse correspondante (toujours avec X = 0).

CALCUL D'ADRESSE SUR L'AXE Y

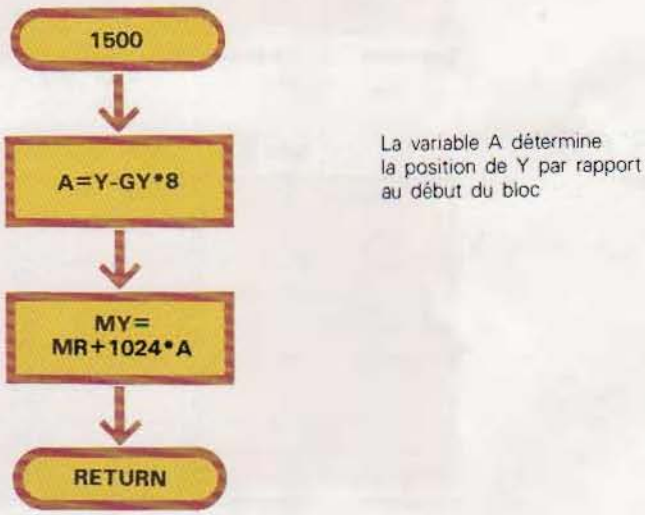
Calcul de l'adresse de référence sur l'axe Y

Entrée : Y=Valeur de l'ordonnée
Sortie : MR=Adresse de référence du bloc
BY=Bloc



Calcul de l'adresse correspondant à l'ordonnée

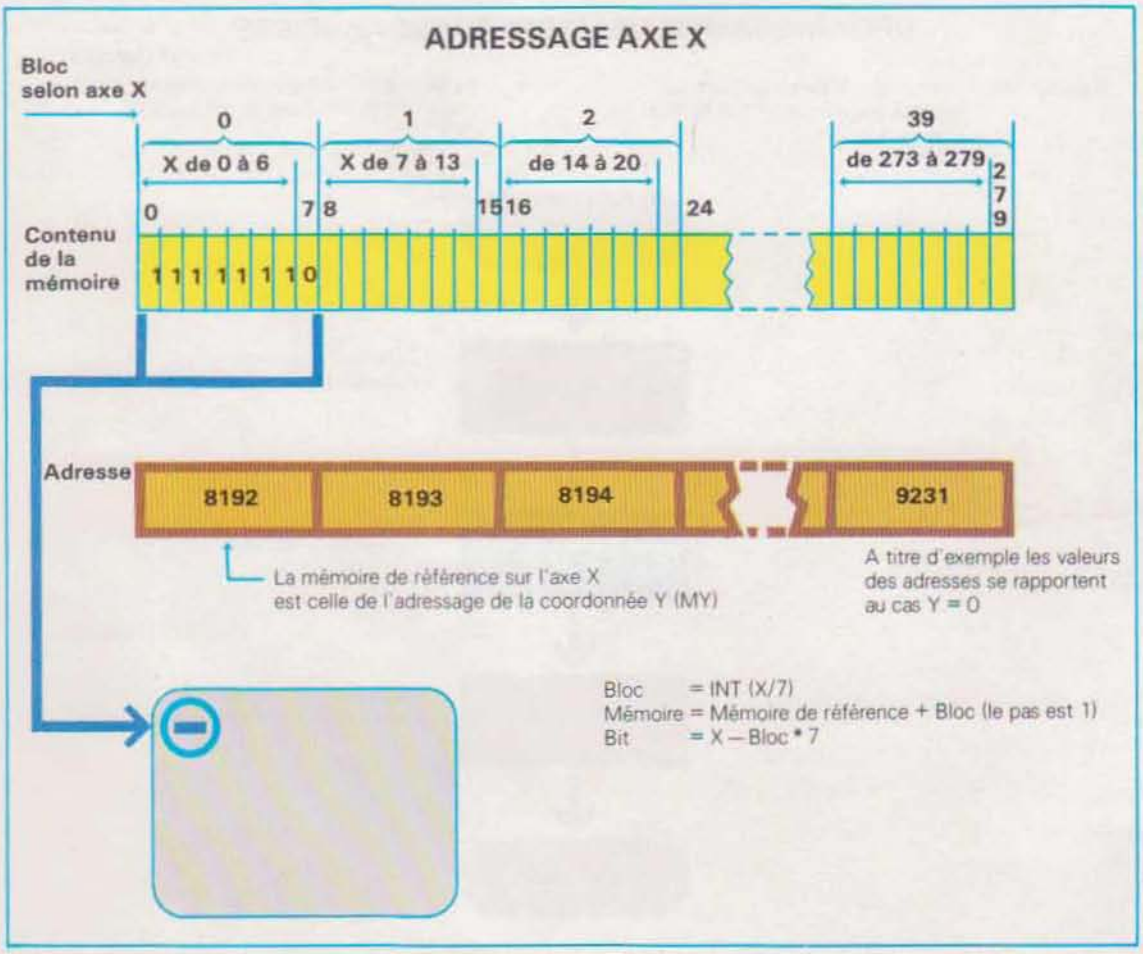
Entrées : Y=Ordonnée
MR=Adresse de référence du bloc
BY=Bloc
Sortie : MY=Adresse mémoire correspondant à Y



Dans le premier sous-programme (1100), on détermine automatiquement la valeur de référence (8192 ou 8232) en fonction de celle de Y. Les opérations, pour parvenir à l'adresse de la mémoire, sont simples, mais la technique n'est pas la plus directe.
On a choisi une méthode fractionnée pour permettre à l'opérateur d'intervenir par variation des valeurs numériques.

Adressage salon l'axe X. Pour chaque ligne (valeur de Y), les colonnes de l'écran (de 0 à 39) correspondent à des mémoires contiguës. Connaissant la valeur initiale correspondant à une ligne Y et à la colonne 0 (ou mieux à l'abscisse X = 0), les autres adresses s'obtiennent de manière séquentielle en ajoutant 1.
Chaque emplacement mémoire contenant 8 bits, il indique l'état de 8 points écran. Le bit 0 correspondant au point X = 0, le bit 1 à X = 1 et ainsi de suite.
Dans notre machine, le bit 7 (la numérotation commence à 0) n'est pas employé ; ainsi, chaque emplacement de mémoire contient l'état (ON/OFF) de 7 points (bits 0 à 6). Dans le graphique ci-dessous, on trouve les adresses des points écran appartenant à la ligne Y = 0. L'adresse de départ de la ligne est 8192. Les suivantes s'obtiennent en ajoutant 1, jusqu'à un maximum de 39, qui est le nombre de colonnes prévues (les colonnes sont, en fait, au nombre de 40, numérotées de 0 à 39). Chaque colonne est divisée en 7 points écran, pour un total de 7 × 40 = 280 points (de X = 0 à X = 279).
Pour adresser un point d'abscisse X, il faut d'abord déterminer son bloc (colonne) ; à partir de cette valeur, calculer l'adresse mémoire et enfin la position du bit dans la mémoire.
Par exemple, le point d'abscisse X = 16 appartient au bloc 2.
En effet :

Bloc = INT (X/7) = 2



1537

et son adresse équivaut au début de ligne (8192) plus 2 soit :

Mémoire = 8192+2 = 8194

Dans l'emplacement ainsi repéré, le 1er bit (numéro 0) représente le point écran X = 14 ; le point étudié étant X = 16, il sera représenté par le 3e bit, c'est-à-dire par le n° 2 (bit 0 = point 14, bit 1 = point 15, bit 2 = point 16). En général, pour le calcul du bit, on a :

Numéro du bit = X — Bloc * 7

L'organigramme ci-dessous et le suivant décrivent deux sous-programmes réalisant ces traitements. Au premier (1800), il faut fournir l'adresse de référence de la ligne (déterminée par la valeur de Y) et la valeur de l'abscisse X. On obtient, en sortie, l'emplacement mémoire MX qui contient l'état du point écran (coordonnées X, Y) et sa position (le n° du bit), à l'intérieur de l'emplacement mémoire.

Le second sous-programme (1900) active un point écran en écrivant la valeur 1 dans le bit correspondant.

L'activation du bit s'effectue avec l'opérateur OU (OR) pour conserver la position précédente, sinon cette activation entraînerait l'effacement des bits précédemment activés. Ce sous-programme lit également l'état d'un point : dans ce cas, il ne contiendra que l'instruction PEEK.

La technique de partition de mémoire est très usitée dans les programmes d'animation de dessins. Avant de déplacer un dessin, on contrôle l'état de l'écran au point d'arrivée pour savoir s'il est déjà activé ou non.

Grâce à cette donnée, le programme d'application peut sélectionner l'opération adéquate, par exemple en refusant le déplacement.

ORGANIGRAMME DE L'ADRESSAGE ABSCISSE

Entrées : MY = Adresse de référence calculée par les sous-programmes 1100 et 1500
X = Valeurs abscisse

Sortie : MX = Adresse de la mémoire axe X
BI = Numéro du bit à activer

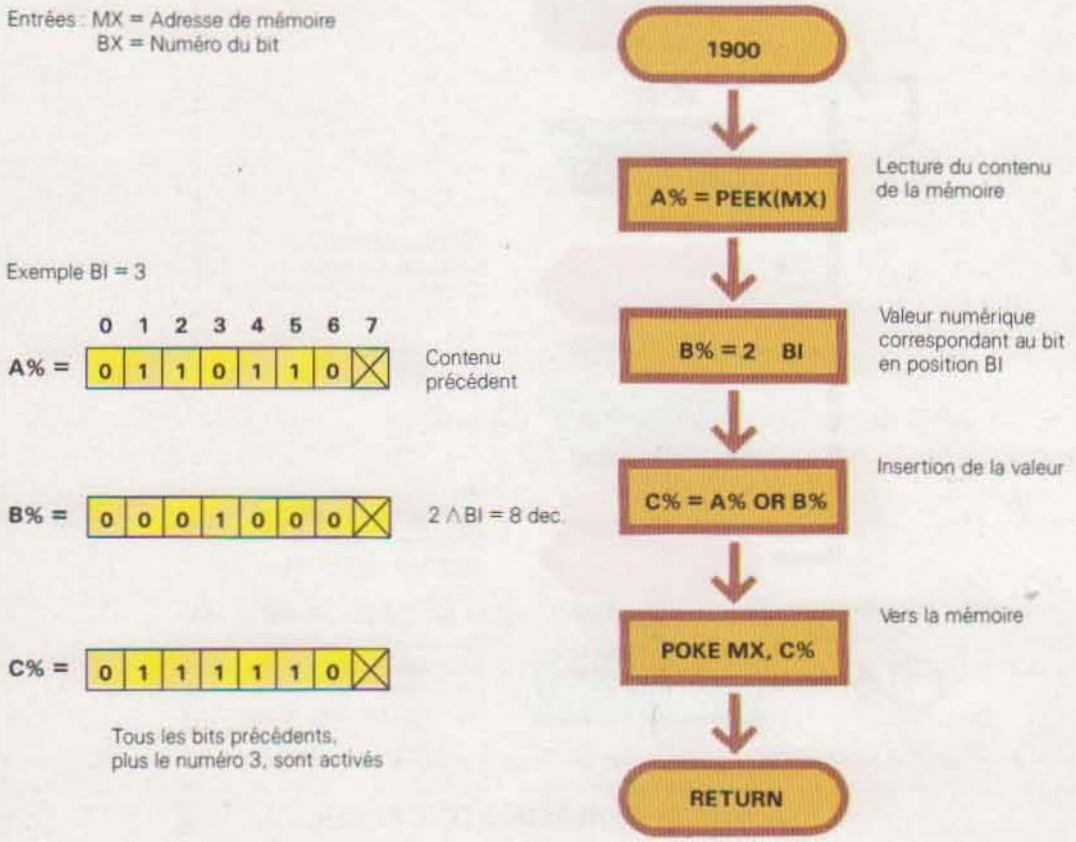


Sélection du groupe appartenant à la coordonnée X

Pas = 1
à partir de la référence MY

Calcul du bit à activer dans la mémoire d'adresse MX

ORGANIGRAMME D'ACTIVATION D'UN BIT



Dans les graphiques des pages 1540 et 1541, on trouvera l'organigramme d'un programme de démonstration qui emploie cette technique. Dans le listing de la page 1542, nous avons omis les sous-programmes dans la mesure où ils ont déjà été présentés. La structure en points écran (caractéristique de la technique « raster ») génère, dans certains cas, une représentation très approximative.

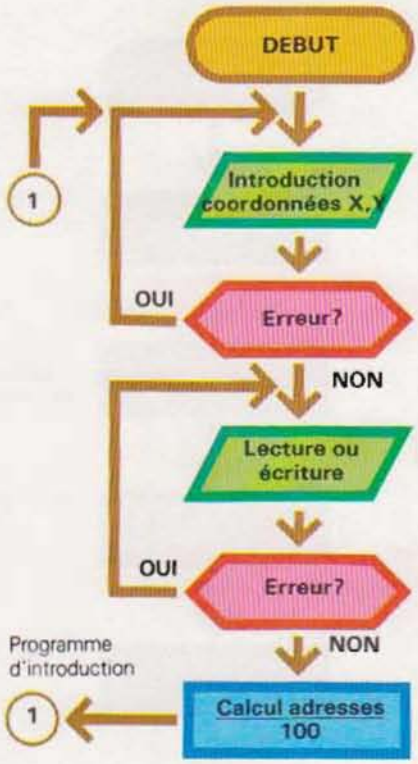
Le résultat est un tracé de segment incliné par rapport aux axes (p. 1543). A cause de cette inclinaison, certaines parties du segment (1 et 3) sont représentées par des traits parallèles à l'un des axes : ils sont respectivement horizontaux et verticaux; en effet, à cause de la pente assez forte, ils concernent plusieurs bits de la même mémoire. Avec des inclinaisons proches de 45°, on a, pour chaque point écran, une ligne et une colonne différentes avec un bien meilleur résultat. Ce défaut est strictement proportionnel à la résolution de l'écran et il ne peut être éliminé.

L'adressage direct de la mémoire

La préparation (ou la reconnaissance) d'une figure, en utilisant l'adressage de la mémoire écran, offre davantage de possibilités que l'usage d'instructions de niveau élevé. Ces dernières sont, en effet, structurées de manière à satisfaire les besoins moyens d'un utilisateur désireux de générer des dessins. Cependant, elles ne prévoient pas de fonctions particulières, comme par exemple la possibilité d'analyser l'écran pour relever la présence d'éventuels dessins.

Pour ce type de fonctions, il est indispensable d'utiliser l'adressage de la mémoire écran pour en lire le contenu ou pour le modifier.

PROGRAMME DE DEMONSTRATION D'ADRESSAGE D'UNE MEMOIRE GRAPHIQUE

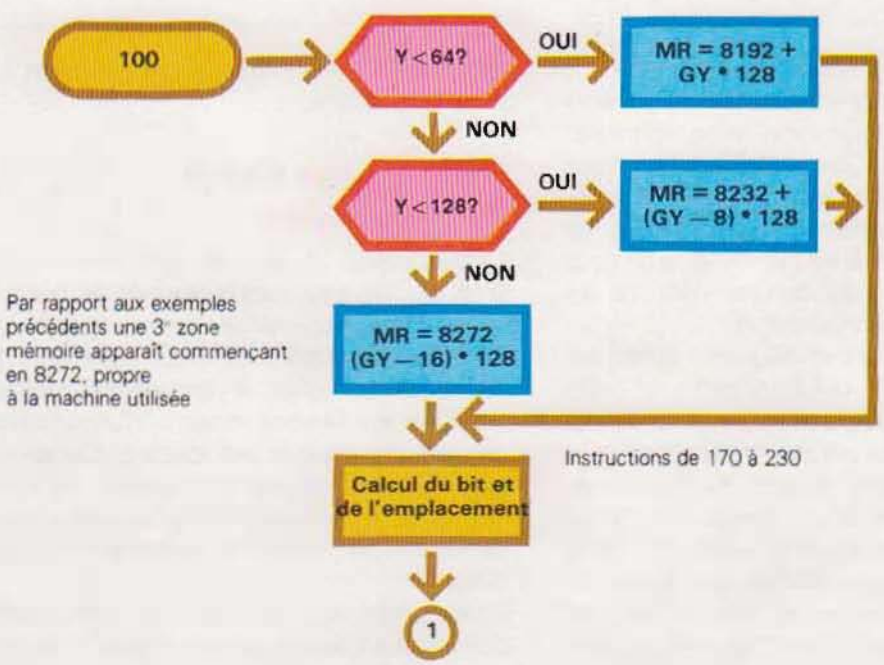


Coordonnées dans les limites de l'écran ($0 < X < 279$ et $0 < Y < 159$)

Réponse L (lecture) ou E (écriture). Les autres caractères sont écartés

Le sous-programme 100 contient les deux calculs (axe Y et X) précédemment rapportés dans deux sous-programmes séparés (1100 et 1500) (voir organigramme détaillé)

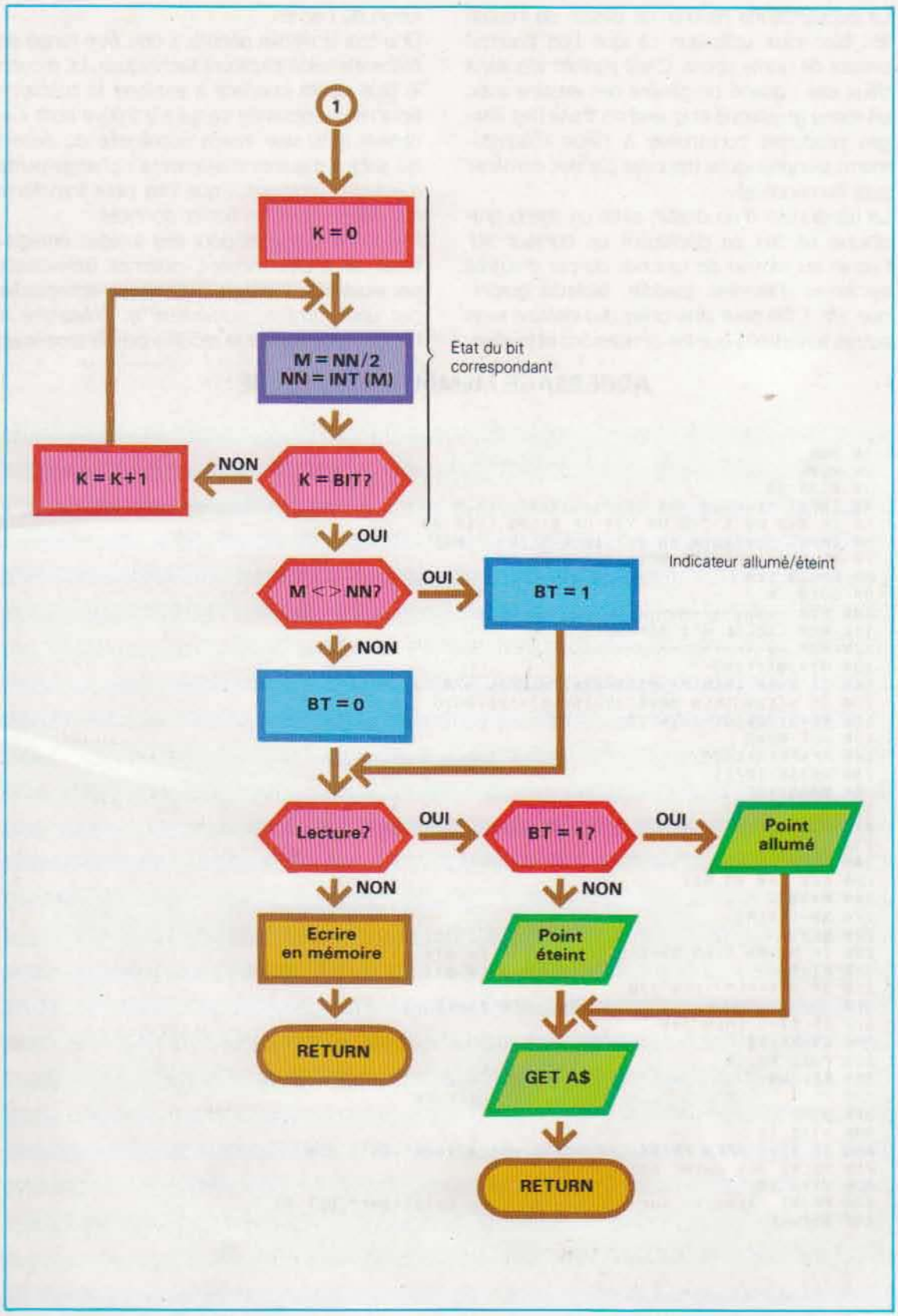
SOUS-PROGRAMME DE CALCUL



Ces deux calculs sont ceux du 1100

Par rapport aux exemples précédents une 3e zone mémoire apparaît commençant en 8272, propre à la machine utilisée

Instructions de 170 à 230


1
K = 0
M = NN / 2
NN = INT (M)
Etat du bit correspondant
K = K+1
NON
K = BIT?
OUI
M <> NN?
OUI
BT = 1
Indicateur allumé/éteint
NON
BT = 0
Lecture?
OUI
BT = 1?
OUI
Point allumé
NON
NON
Ecrire en mémoire
Point éteint
RETURN
GET A$
RETURN

La possibilité de repérer un dessin de l'écran est bien plus utile que ce que l'on pourrait penser de prime abord. C'est surtout vrai dans deux cas : quand on génère des dessins avec un menu graphique et quand on traite des images produites construites à l'aide d'équipements périphériques (reprises par des caméras puis numérisées).

La génération d'un dessin avec un menu graphique se fait en déplaçant un curseur sur l'écran au moyen de touches ou par d'autres systèmes d'entrées (paddle, tablette graphique, etc.). On peut ainsi créer des dessins sans autres limitations que les dimensions et la résolution de l'écran.

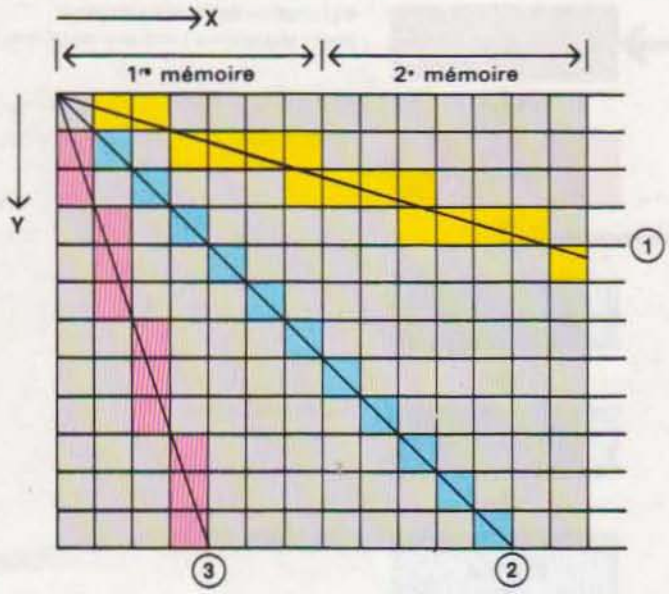
Une fois le dessin généré, il doit être rangé en mémoire selon plusieurs techniques. Le moyen le plus direct consiste à explorer la mémoire écran et à conserver ce qui s'y trouve écrit. On obtient ainsi une image numérisée du dessin qui subira d'autres traitements : changements d'échelle, rotations... que l'on peut transférer sur disque dans un fichier données.

Mêmes traitements pour des images enregistrées avec des moyens externes (télévisuels par exemple). L'image originale est enregistrée par une caméra, numérisée et présentée à l'écran. L'opérateur la modifie puis la stocke en

ADRESSAGE MEMOIRE GRAPHIQUE

```
10 HGR
20 HOME
30 VTAB 23
40 INPUT "Entrez les coordonnées: ";X,Y
50 IF X<0 OR X>279 OR Y<0 OR Y>159 THEN 20
60 INPUT "Lecture ou écriture (L/E) ";MD$
70 IF MD$<>"L" AND MD$<>"E" THEN 20
80 GOSUB 100
90 GOTO 20
100 REM --------------------
110 REM CALCUL DES ADRESSES
120 REM --------------------
130 GY=INT(Y/8)
140 IF Y<64 THEN MR=8192+GY*128:GOTO 170
150 IF Y<128 THEN MR=8232+(GY-8)*128:GOTO 170
160 MR=8272+(GY-16)*128
170 A=Y-GY*8
180 MY=MR+A*1024
190 GX=INT(X/7)
200 MX=MY+GX
210 BIT=X-7*GX
220 A%=PEEK(MX)
230 B%=2BIT
240 NN=A%
250 FOR K=0 TO BIT
260 M=NN/2
270 NN=INT(M)
280 NEXT
290 IF M<>NN THEN BT=1:GOTO 310:REM Le bit est allumé
300 BT=0:                          REM Le bit est éteint
310 IF MD$="L" THEN 370
320                                REM Ecriture
330 IF BT=1 THEN 360
340 C%=A%+B%
350 POKE MX,C%
360 RETURN
370                                REM Lecture
380 HOME
390 VTAB 23
400 IF BT=1 THEN PRINT "Le point est allumé":GOTO 420
410 PRINT "Le point est éteint"
420 VTAB 24
430 PRINT "Appuyez sur une touche pour continuer":GET A$
440 RETURN
```

IMPACT DE LA RESOLUTION DE L'ECRAN VIDEO SUR LA PRESENTATION DES SEGMENTS NON PARALLELES AUX AXES



L'adressage minimum est un bit, auquel correspond un petit point lumineux. Dans le schéma, on a tracé 3 lignes inclinées. Pour une seule d'entre elles (45 degrés) la figure de l'écran se rapproche de la tendance. Pour les autres, on obtient une série de segments parallèles. Dans la figure, chaque point écran est représenté par un carré

mémoire toujours selon la méthode de l'adressage direct. Ces possibilités, associées à l'emploi d'un écran couleur, ont ouvert de nouvelles perspectives au dessin assisté par ordinateur, notamment dans la publicité où la création utilise les effets spéciaux. Dans le domaine de l'art, elles ont donné naissance à une véritable école de peinture grâce au pinceau électronique. L'apport de l'ordinateur consiste en sa grande rapidité et dans la facilité des modifications. La variation d'une couleur ou de la forme d'un détail n'exige d'autres opérations que l'envoi d'une commande à l'ordinateur, là où les moyens traditionnels demanderaient un temps considérable.

Notons aussi la possibilité de mettre en mémoire certaines formes standard qui, lors de la préparation du dessin, seront rappelées et utilisées pour un travail de composition. On peut ainsi créer un menu graphique pour des applications artistiques.

Stockages des images graphiques

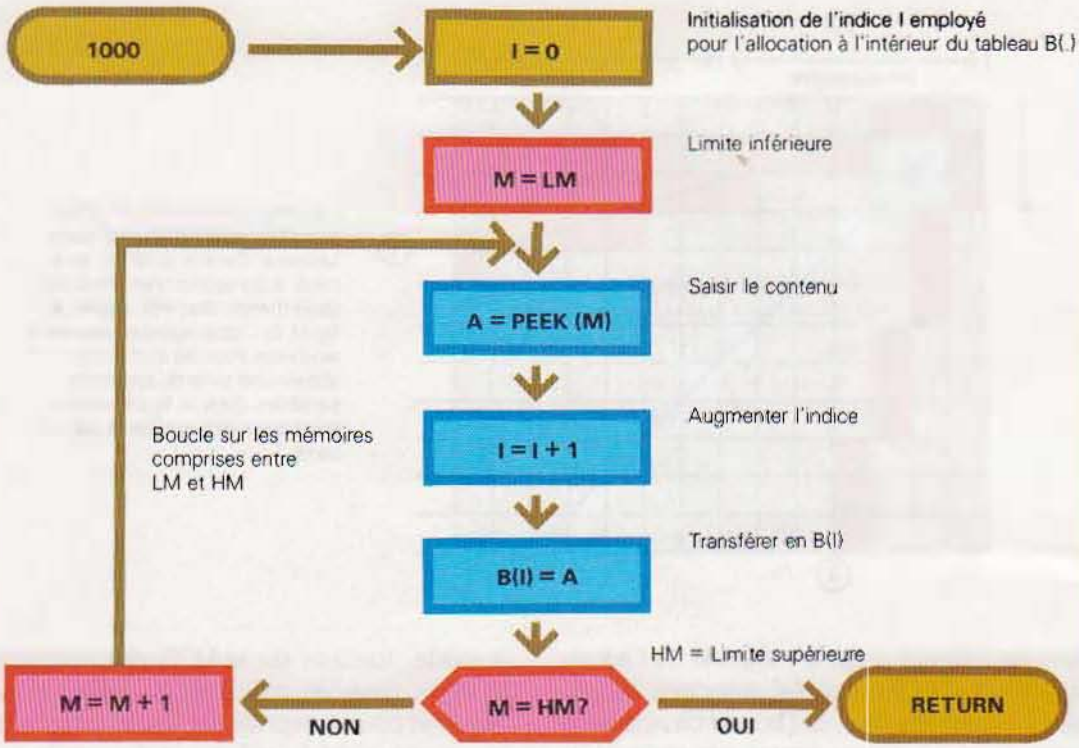
Un traitement graphique commence par la saisie de la figure présentée à l'écran. Pour ce faire, certaines machines disposent d'instructions spéciales. Ainsi, dans l'Olivetti M20, l'instruction POINT (X, Y) restitue la couleur du point de coordonnées X, Y, ainsi que son état.

Il existe, toujours sur le M20, des instructions pour la mise en mémoire de toute la page écran et pour sa représentation. Pour les autres machines, c'est à l'utilisateur d'écrire ses propres sous-programmes.

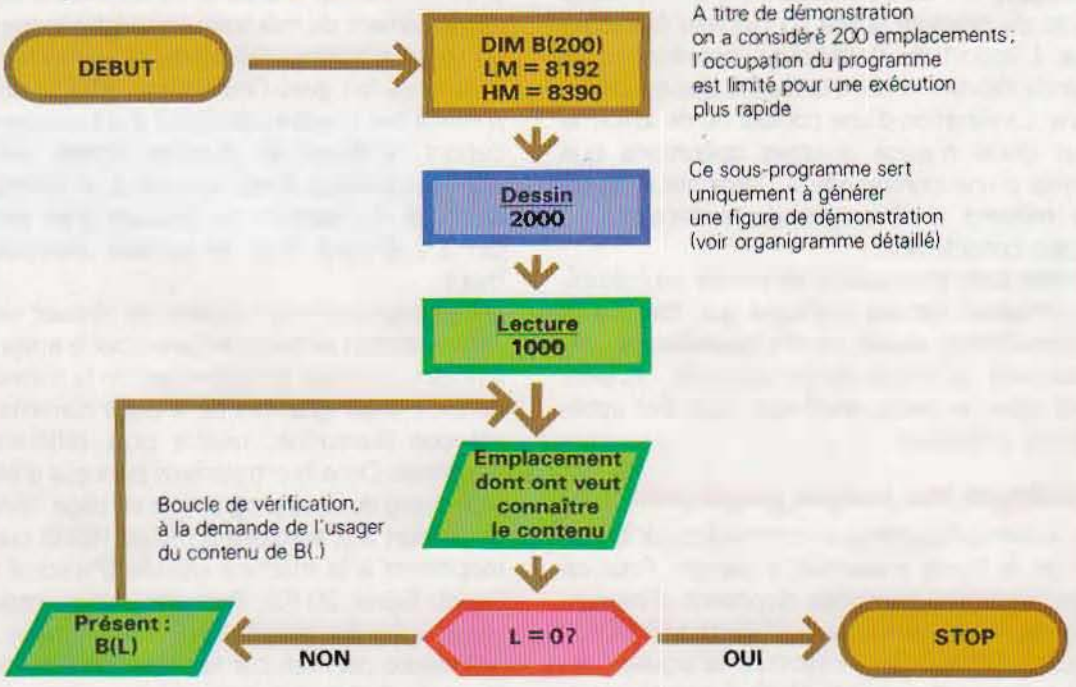
La méthode la plus simple consiste à saisir, avec l'instruction PEEK, le contenu de chaque emplacement de mémoire écran et à le tranférer dans un tableau défini par l'utilisateur. Cette lecture se fait avec l'instruction PEEK quand il n'existe pas d'autre possibilité d'adressage en-dehors du direct. En d'autres termes, aucun nom symbolique n'est associé à la mémoire écran et l'utilisateur, ne pouvant s'en servir, devra l'adresser avec le numéro d'emplacement.

Le sous-programme capable de réaliser cette tâche est tout simplement une boucle entre les limites maximales et minimales de la mémoire écran. L'organigramme de la page suivante en est une illustration, valable pour différentes machines. Dans le programme principal d'essai (voir listing du sous programme en page 1546), on recourt aux adresses 8192 et 16383 qui se rapportent à la machine utilisée (Personal Kid Siprel, Siprel 2010). Pour les autres ordinateurs, il faudra modifier les valeurs selon les indications données par le manuel d'utilisation. Dans le listing, pourraient apparaître deux

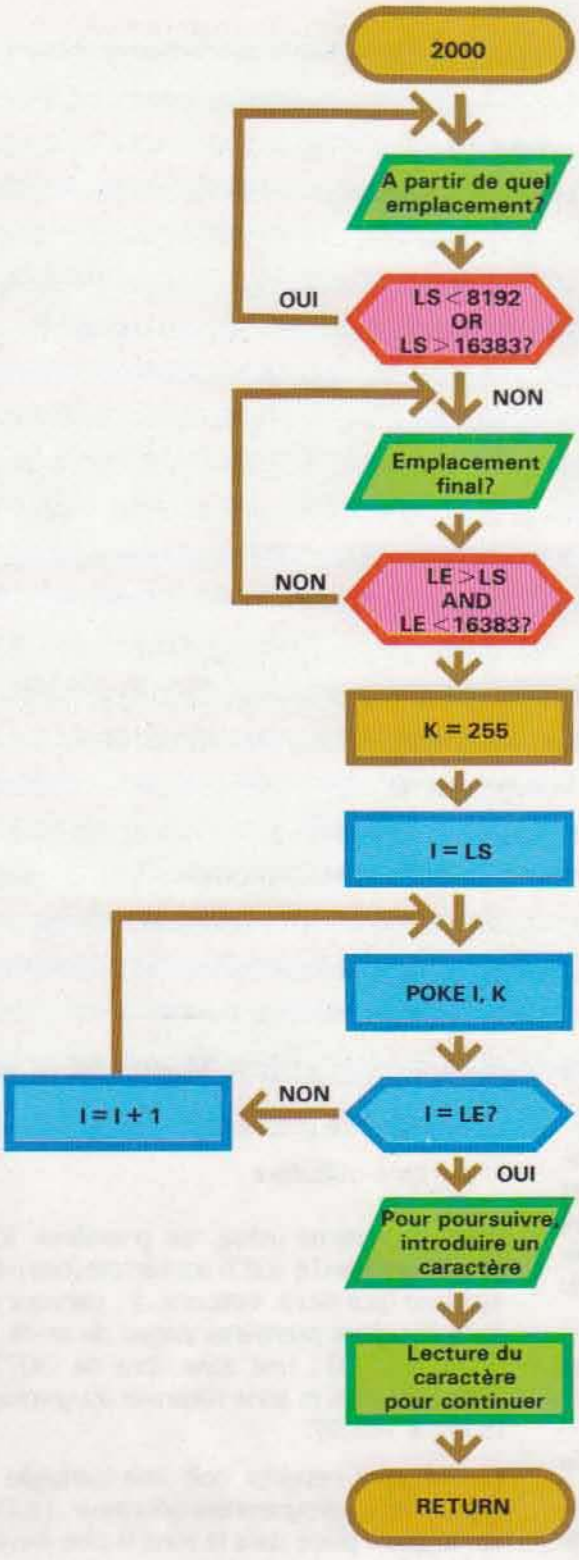
LECTURE DE LA MEMOIRE VIDEO



Programme principal d'essai



PREPARATION D'UN SEGMENT D'ESSAI



Le sous-programme n'a qu'un but pédagogique et se limite à la présentation de segments horizontaux. Les deux emplacements de mémoire doivent appartenir à la même ligne.
En introduisant des adresses n'appartenant pas à la même ligne, on peut activer tous les points écran compris entre un minimum et un maximum

Active les bits de 0 à 7

Boucle qui écrit la valeur 255 dans les mémoires graphiques choisies

LECTURE ET VISUALISATION DU CONTENU DE LA MEMOIRE

VERSION Siprel 2010, Apple et compatibles

```
50 HGR
100 DIM B(200):LM=8192:HM=8390
110 GOSUB 2000
120 GOSUB 1000
130 HOME:VTAB 22:INPUT "Entrez localisation ";L
140 IF L=0 THEN TEXT:END
150 HOME:VTAB 22:PRINT "Contenu = "B(L)
155 PRINT "Appuyer une touche":GET Q$
160 GOTO 130
1000 REM ===========
1001 REM * Lecture *
1002 REM ===========
1003 :
1010 I=0
1015 FOR M=LM TO HM
1020 A=PEEK(M):I=I+1
1030 B(I)=A
1040 NEXT M
1050 RETURN
2000 REM ==========
2001 REM * Dessin *
2002 REM ==========
2003 REM
2010 HOME:VTAB 22:INPUT "Localisation initiale ";LS
2015 IF LS<8192 OR LS>16383 THEN 2010
2020 INPUT "Localisation finale ";LE
2025 IF LE>LS AND LE<16383 THEN 2040
2030 GOTO 2020
2040 K=255
2045 FOR I=LS TO LE
2050 POKE I,K
2060 NEXT I
2070 HOME:VTAB 22:PRINT "Frappez une touche":GET Q$
2080 RETURN
```

instructions reliées étroitement à la machine : HIMEM et LOMEM. Ce type d'instructions indique la zone occupée par le programme. Elle est donc limitée par HIMEM (adresse de mémoire la plus élevée) et par LOMEM (adresse de mémoire la plus basse) selon la configuration du matériel et du logiciel du système.

La RAM du système est divisée en 3 fonctions ; en voici un résumé :

— système d'exploitation, par exemple le DOS

— fonctions particulières (par exemple mémoire écran de texte)

— mémoire graphique

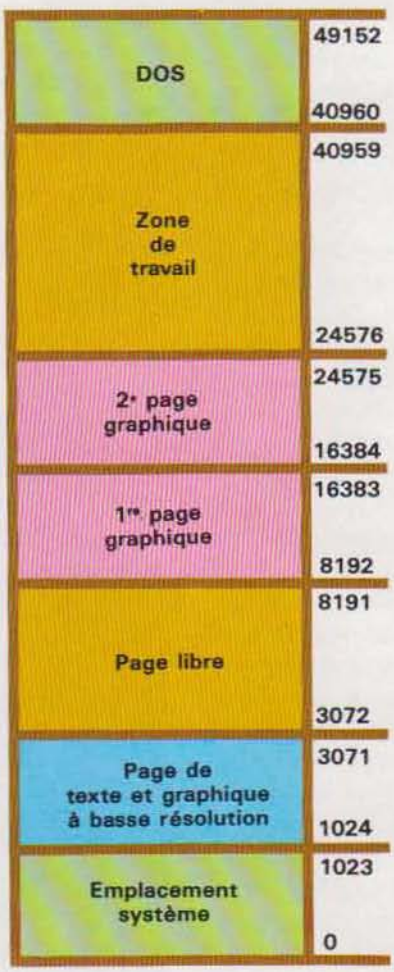
— zone utilisateur

Dans le système utilisé, les premières 1024 allocations (de 0 à 1023) sont employées par le système (pointeurs, tampons...) ; viennent ensuite les deux premières pages de texte (de 1024 à 3071) ; une zone libre de (3072 à 8191) et enfin la zone réservée au graphique (8192 à 16383).

La mémoire restante doit être partagée en 3 DOS et en programmes utilisateur. Le DOS est toujours placé dans la zone la plus élevée ; donc, reste pour le programme utilisateur, une

zone comprise entre la fin de la page graphique et le début du DOS (schéma ci-dessous).
Ce système offre, en outre, une 2e page graphique gérée et présentée indépendamment de la première (emplacements 16384 à 24575). Le programme utilisateur part donc de l'allocation 24576 et s'achève avant le début du DOS (emplacement 40960).
Les deux paramètres HIMEM et LOMEM établissent ces bornes pour éviter d'empiéter sur la zone réservée au graphique.
Dans ce cas particulier, les deux paramètres ne sont pas nécessaires, étant donné le faible encombrement du programme : nous ne les avons mentionnés que pour faire ressortir ce problème d'encombrement de mémoire, particulièrement important dans les applications graphiques. Notre programme a cependant besoin de certaines fonctions supplémentaires pour la mise en mémoire et le rappel de pages graphiques.
La méthode la plus avantageuse consiste à utiliser un tampon qui ne contient qu'une seule ligne à la fois (40 emplacements de mémoire) ; à chaque nouvelle ligne, son contenu est transféré sur disque ; le tampon est donc à nouveau libre pour recevoir la ligne suivante.
Cette méthode se trouve illustrée par l'organigramme de la page 1551.
Puisqu'il faut disposer d'un dessin sur l'écran pour procéder au test du programme (dont le sous-programme saisit les points), nous

PARTITION DE LA MEMOIRE RAM DANS LE SIPREL 2010



Radioscopie de l'ordinateur personnel

Bien souvent, on trouve chez l'utilisateur d'un ordinateur personnel une bonne connaissance du langage de programmation. En revanche, il ignore la structure de sa machine. Le châssis de l'ordinateur est souvent considéré comme une barrière impénétrable.

L'attitude d'embarras à l'égard de la machine et de ses composants, risque cependant d'avoir des conséquences fâcheuses, car l'apparition d'une panne ou d'un mauvais fonctionnement du système prend souvent l'utilisateur au dépourvu.

Toute personne qui s'est trouvée dans la triste nécessité de devoir faire réparer une défaillance matérielle de son ordinateur personnel aura certainement expérimenté le manque de diligence avec lequel les centres d'assistance émettent un diagnostic, ainsi que la longueur des délais de réparation.

C'est le moment de rappeler avec insistance que la structure interne d'un ordinateur personnel est très simple, bien plus simple que celle d'un circuit de téléviseur ou de radio. En effet, ceux-ci traitent des signaux analogiques et non numériques.

L'utilisateur courageux, qui décide un beau jour de retirer le capot qui recouvre les circuits, reste toujours surpris par le peu d'éléments et par la modularité du système. Il peut constater que chaque fonction est réalisée par des composants (cartes) bien séparés, facilement identifiables, dont les connections sont, à première vue, élémentaires.

Sur chaque carte, les circuits intégrés ne sont pas soudés mais insérés par pression sur des plots prévus à cet effet.

Dans ces conditions, une réparation ne devrait pas demander plus de 15 minutes pour le diagnostic et moins d'une minute pour le changement du circuit intégré en cause, lequel coûte rarement plus de quelques dizaines de francs. Donner un coup d'oeil à la structure interne de votre ordinateur vaut donc la peine, ne serait-ce que pour se rendre compte que ce que nous venons de dire est vrai.

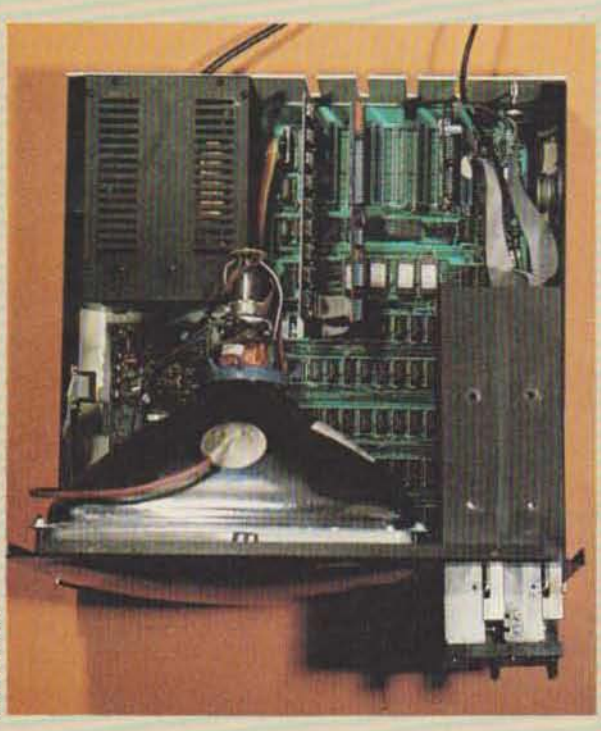
Le système Siprel 2040 (Personal Kid) se prête fort bien à une étude générale de l'architecture d'un ordinateur personnel. Il s'agit d'une machine où l'écran, les lecteurs et les cartes se partagent une même entité physique à laquelle est relié un clavier séparé.

Même s'il existe des versions de ce système où les unités sont séparées, nous allons examiner cette configuration justement parce que, au premier abord, elle apparaît inabordable.

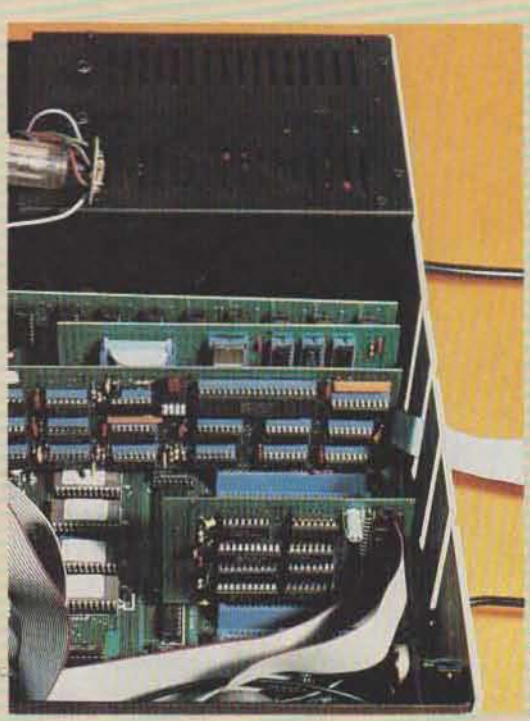
Dans la première photo ci-dessous, nous avons enlevé le couvercle du châssis pour extraire les deux lecteurs : il nous a suffi de retirer 8 vis. La seconde (haut de la page 1549) montre le système vu du dessus. Outre les lecteurs de type « slim », on y voit clairement le tube à rayons cathodiques et, au-dessus, la boîte d'alimentation, avec les fentes pour la ventilation du transformateur. Sur le fond on entrevoit la fiche mère, avec les slots d'insertion des diverses cartes fonctionnelles (visibles dans la partie haute de la photo). La troisième photo montre ces cartes montées et, au premier plan, la fiche d'interface avec les lecteurs et les deux câbles plats qui en sortent (un pour chaque unité de disque). Au second plan, on distingue une partie de la carte qui contient l'unité centrale (Zilog Z80) ; puis la carte interface avec l'écran et une carte extension de la mémoire RAM. Les mêmes cartes sont visibles dans la quatrième



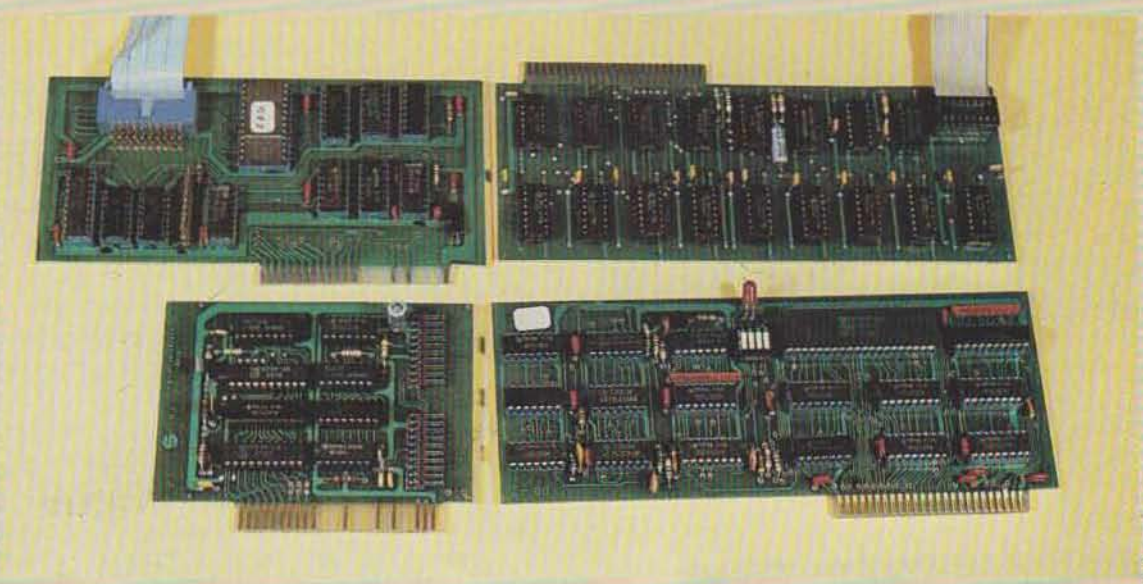
Il Dagherotipo



Il Dagherrotipo



Il Dagherrotipo



Il Dagherrotipo

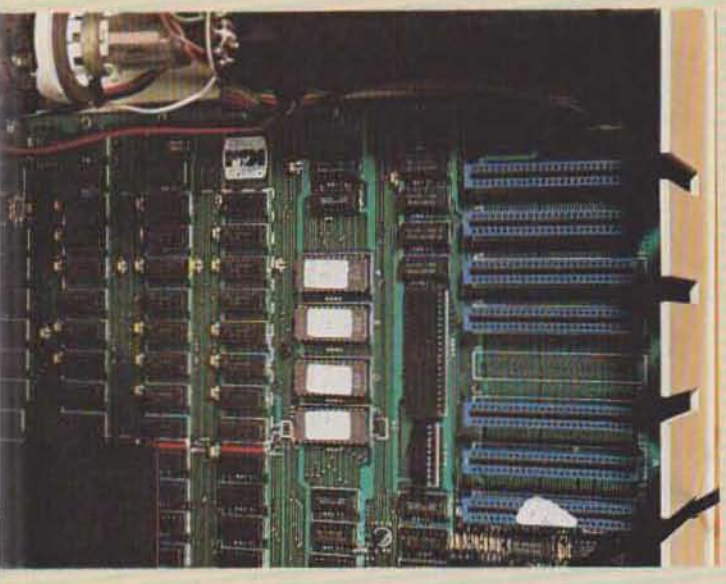
photo : en haut à gauche, la carte de gestion d'écran 80 colonnes et, à côté, l'extension mémoire ; en bas à gauche, la carte interface avec les lecteurs et, à côté, la carte avec le processeur Z80.

La cinquième photo (page 1550) met en évidence un détail interne après séparation des cartes fonctionnelles et de la fiche mère. Les 7 slots sont maintenant bien visibles et on peut entrevoir, dans le bas, le sabot pour la connexion des dispositifs externes (joystick, tablette graphique, paddle, etc.)

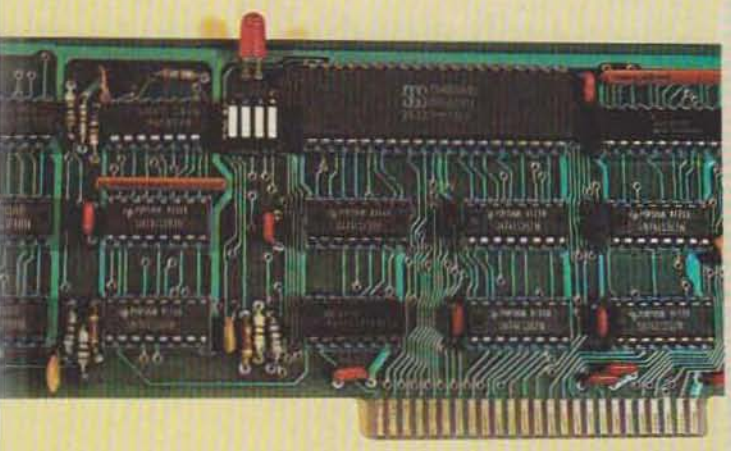
Une tablette graphique est reliée au sabot dans notre exemple : le câble de connexion sort du côté droit de la photo.

Le circuit intégré de grande taille, qui se trouve dans le sens contraire des autres, est l'UC (Rockwell 6502), alors que les circuits intégrés plus petits, occupant la partie gauche de l'image, sont des mémoires RAM.

Les quatre circuits intégrés recouverts par des étiquettes sont les EPROM (mémoires de



Il Daghenotipo



Il Daghenotipo



Il Daghenotipo

lecture seulement, qui sont programmables et effaçables) où se trouve le système d'exploitation. Les étiquettes qui recouvrent les fenêtres permettent d'effacer le contenu des mémoires au moyen d'une exposition aux rayons ultraviolets et ont pour but d'empêcher l'effacement accidentel à la suite d'une exposition prolongée à la lumière solaire.

Le système Siprel utilise, comme base, l'unité centrale Rockwell 6502 (la même qu'Apple et que Commodore 64) qui travaille en DOS mais peut être complétée par un processeur Zilog Z80, travaillant normalement sous CP/M (photo centrale).

Le processeur 6502 est contenu dans la carte mère sur laquelle ont peut insérer une deuxième qui contient la Z80. En insérant la carte optionnelle, toutes les unités fonctionnelles (mémoires, dispositifs d'E/S, etc.) peuvent être gérées par la nouvelle unité centrale et on obtient ainsi un système qui travaille sous CP/M. Le passage d'un mode de fonctionnement à l'autre est automatique. A la mise en route le système est prêt à fonctionner sous DOS, avec le processeur 6502. En insérant la disquette CP/M dans le lecteur et en tapant, à partir du clavier, la commande DOS de chargement (IN = 6) le DOS est « remplacé » par le CP/M et le système passe sous le contrôle de l'unité Z80.

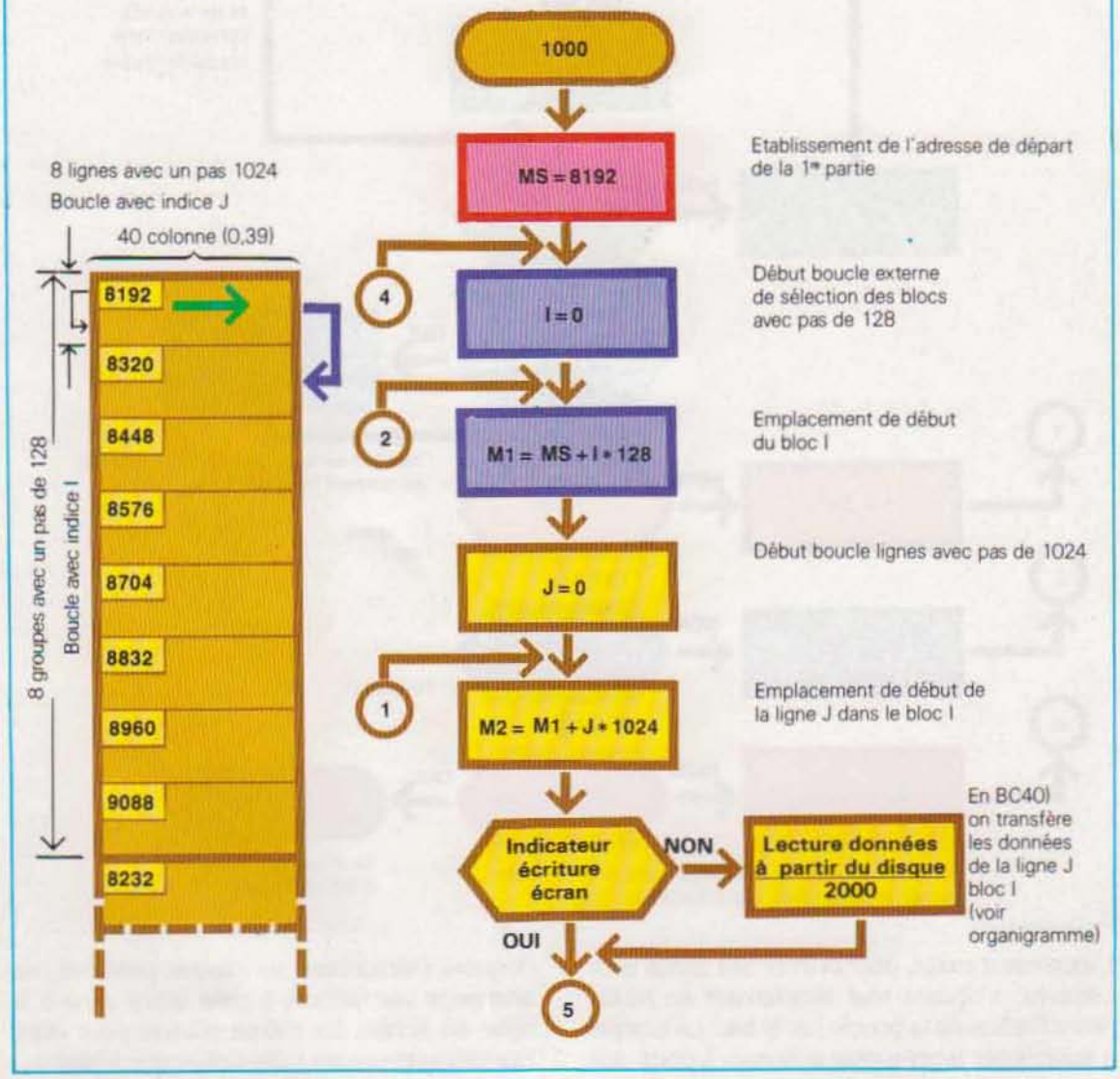
Cette dualité est très pratique dans la mesure où elle permet de faire exécuter par la machine tous les programmes écrits sous l'un ou l'autre des systèmes d'exploitation qui sont parmi les plus utilisés.

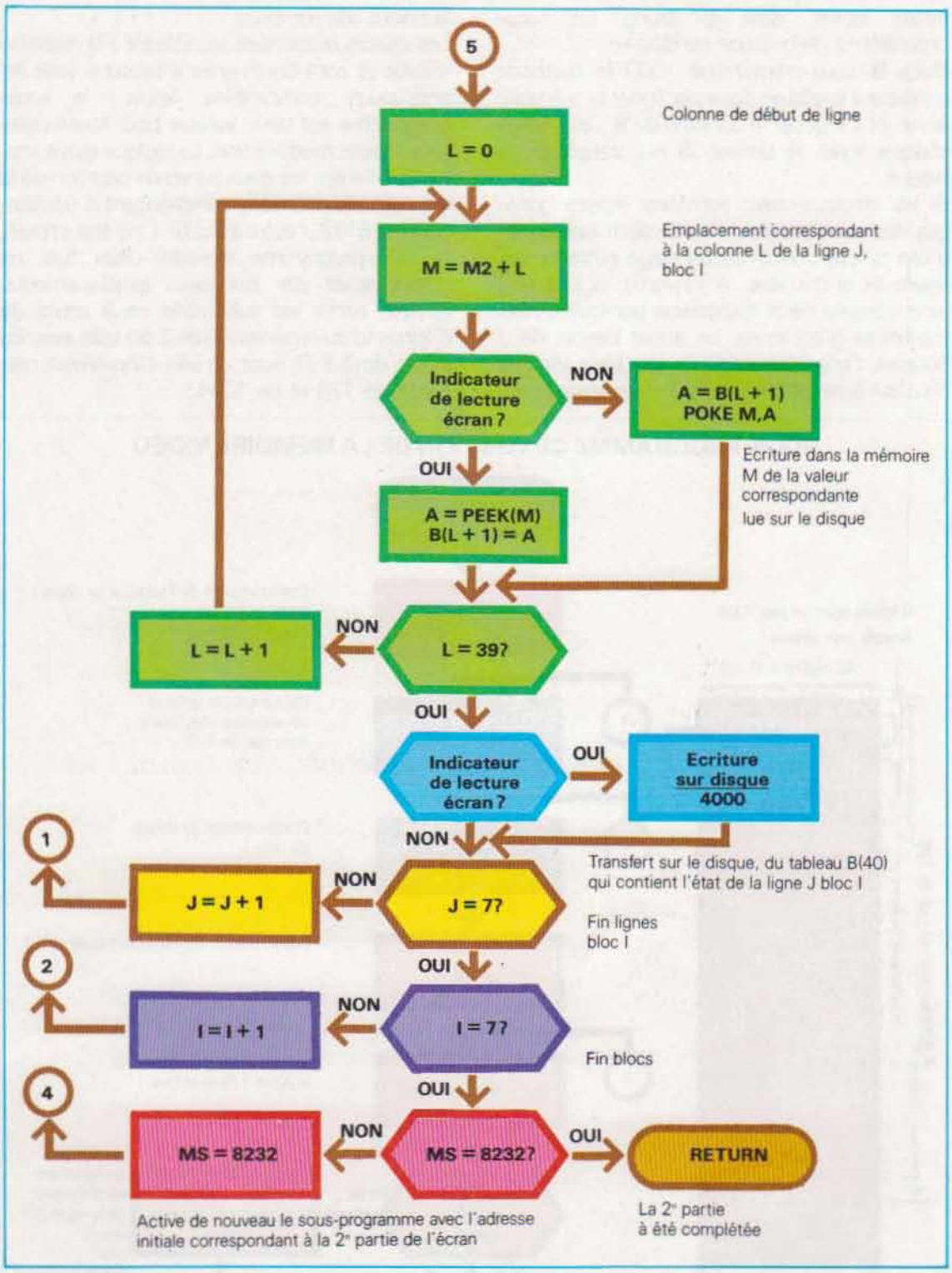
La dernière photo montre le clavier du système. Dans la partie centrale on distingue le circuit intégré destiné à la gestion du dispositif. Ajouter que le circuit intégré plus petit, à côté de l'attache du câble, sert à isoler le circuit principal de la ligne.

Nous pensons ainsi avoir illustré la transparence d'un système de traitement bien conçu et surtout d'avoir donné envie aux utilisateurs d'approfondir, de leur côté, l'étude de l'anatomie du matériel. De cette expérience, on pourra prendre conscience du fait que chacun de nous est en mesure de résoudre seul bien des problèmes qui apparaissent normalement dans la vie d'un ordinateur ou, au moins, d'orienter de manière claire un éventuel travail de réparation.

avons inclus, dans le listing, un sous-programme de création de dessin.
Dans le sous-programme 1000 la méthode consiste à explorer, ligne par ligne, la mémoire écran et à stocker le contenu de B ; à la fin de chaque ligne, le tableau B est transféré sur disque.
Si les emplacements mémoire étaient contigus, les sous-programmes seraient constitués d'une simple boucle de balayage entre la première et la dernière. A l'inverse, si l'on doit tenir compte de la disposition particulière des mémoires graphiques, on aurait besoin de 3 boucles, l'une pour explorer une ligne (de 0 à 39 c'est-à-dire 40 mémoires) et les autres pour le calcul des adresses.
Les valeurs rapportées se réfèrent à la machine utilisée et sont communes à presque tous les ordinateurs compatibles Apple ; le sous-programme est donc valable pour toute cette famille sans modification. La logique suivie traite séparément les deux zones de partition de la mémoire, la première commençant à l'emplacement 8192, l'autre à 8232. Une fois appelé, le sous-programme travaille deux fois en commençant par ces deux emplacements. Chaque partie est subdivisée en 8 zones de 8 lignes (d'où la nécessité de 2 boucles avec un indice de 0 à 7), avec un pas d'incrément respectif de 128 et de 1024.

SOUS-PROGRAMME DE GESTION DE LA MEMOIRE VIDEO

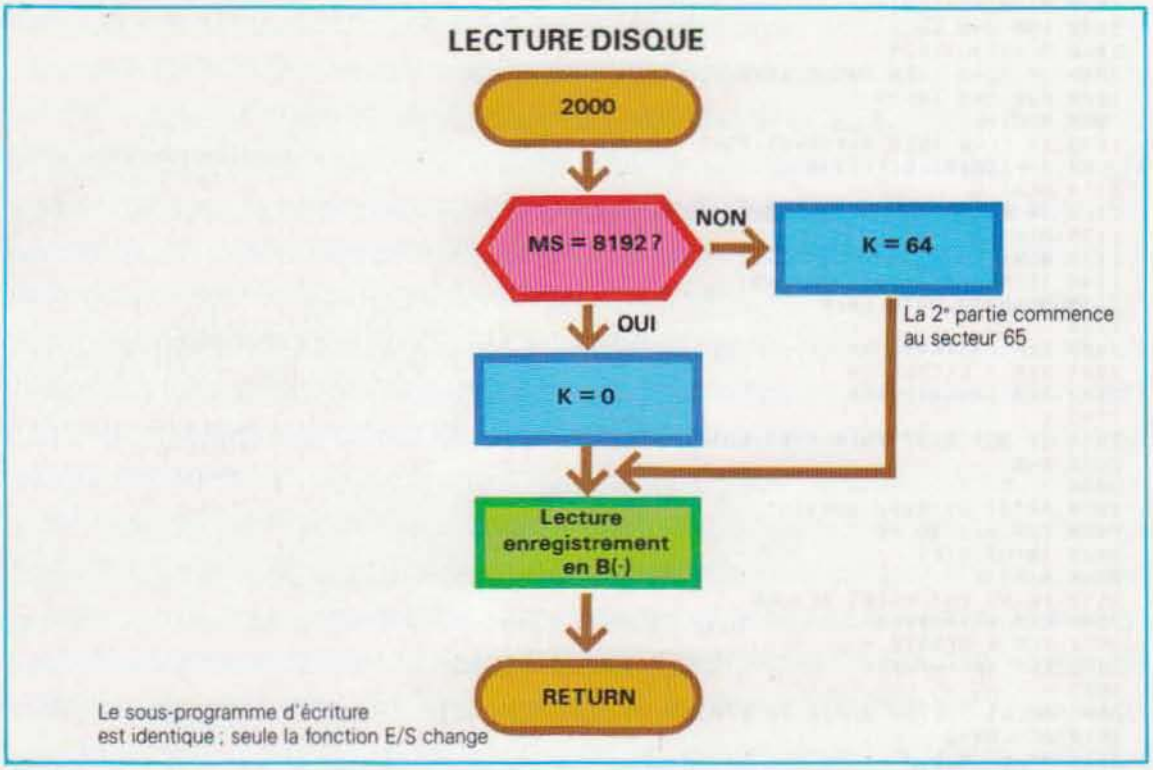




L'incrément exact, pour pointer des zones successives, s'obtient tout simplement en multipliant l'indice de la boucle par le bas. La logique à suivre pour recomposer le dessin, à partir des données mémorisées sur disque, présente une altenance par rapport à celle ayant servi à la mise en fichier. La même routine peut donc alors être utilisée pour deux objectifs différents :

l'un ou l'autre mode de fonctionnement est indiqué par la valeur d'un indicateur activé lors de l'appel.

L'emploi de ce sous-programme sur d'autres machines exige parfois d'importantes modifications ; par exemple, si on dispose d'une ins-



MEMORISATION DES IMAGES GRAPHIQUES SUR DISQUE

```
1 REM ====================
2 REM *     PROGRAMME     *
3 REM *     DE DESSIN     *
4 REM * ET MEMORISATION *
5 REM *     SUR DISQUE    *
6 REM ====================
7 :
30 :
40 HIMEM:8191
45 :
50 DIM B(40):Y=90:X=140:S=5
55 D$=CHR$(4)
60 PRINT D$"Open dessin"
65 HCOLOR=3
100 HOME:VTAB 22:INPUT "Lecture/Ecriture (L/E) ";A$
110 IF A$="L" THEN FL=0:HGR:GOTO 150
120 IF A$="E" THEN FL=1:HGR:GOTO 140
130 IF A$="" THEN 200
135 GOTO 100
140 GOSUB 3000
150 GOSUB 1000:GOTO 100
200 PRINT D$"Close dessin":TEXT:END
999 :
1000 REM ==================
```

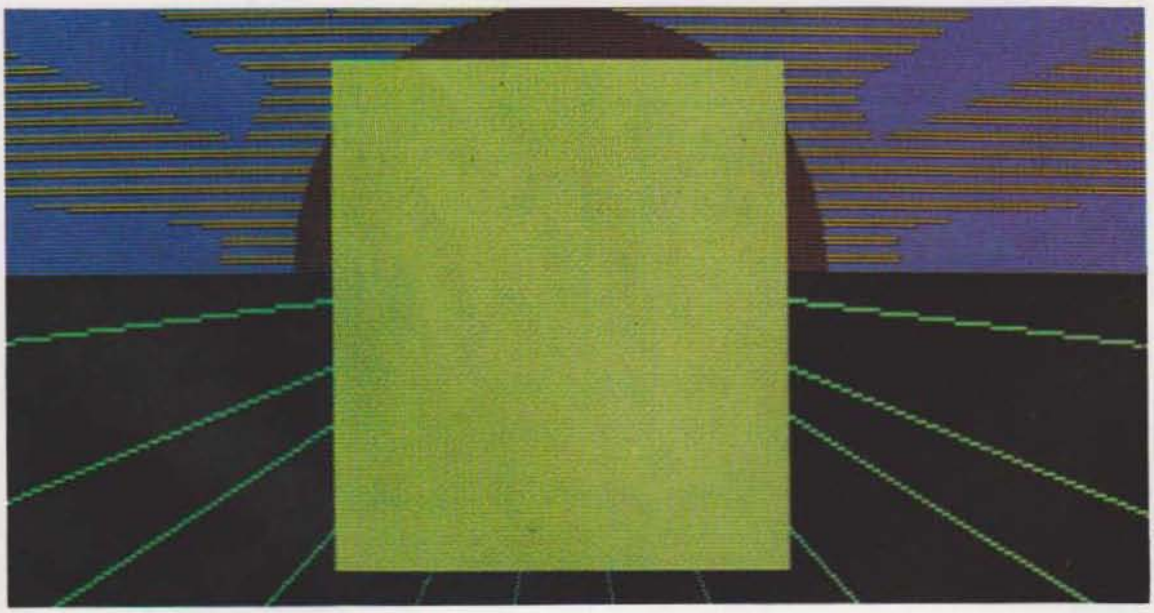
```
1001 REM * MEMOIRE ECRAN *
1002 REM ==================
1003 :
1010 MS=8192
1020 FOR I=0 TO 7
1030 M1=MS+I*128
1040 FOR J=0 TO 7
1050 M2=M1+J*1024
1060 IF FL=0 THEN GOSUB 2000
1070 FOR L=0 TO 39
1080 M=M2+L
1090 IF FL=0 THEN A=B(L+1):POKE M,A:GOTO 1110
1100 A=PEEK(M):B(L+1)=A
1110 NEXT L
1120 IF FL=1 THEN GOSUB 4000
1130 NEXT J
1135 NEXT I
1140 IF MS=8232 THEN RETURN
1150 MS=8232:GOTO 1020
1160 :
2000 REM ============
2001 REM * LECTURE *
2002 REM ============
2003 :
2010 IF MS<>8192 THEN K=64:GOTO 2030
2020 K=0
2030 :
2050 PRINT D$"Read dessin"
2060 FOR P=1 TO 40
2070 INPUT B(P)
2080 NEXT P
2150 PRINT D$":PRINT:RETURN
3000 REM ===========
3001 REM * DESSIN *
3002 REM ===========
3003 :
3005 HPLOT 1,1 TO 278,1 TO 278,159 TO 1,159 TO 1,1
3010 HCOLOR=3
3015 HPLOT X,Y
3020 HOME:VTAB 22:PRINT "I=haut / M=bas / K=droite / J=gauche"
3025 PRINT "S=sauvegardez"
3030 VTAB 1:GET Q$:PRINT
3040 IF Q$="I" THEN Y=Y-S:GOTO 3200
3050 IF Q$="M" THEN Y=Y+S:GOTO 3200
3060 IF Q$="K" THEN X=X+S:GOTO 3200
3070 IF Q$="J" THEN X=X-S:GOTO 3200
3080 IF Q$="C" THEN 3010
3090 IF Q$="S" THEN PRINT:RETURN
3100 GOTO 3030
3200 HPLOT TO X,Y:GOTO 3030
4000 REM ============
4001 REM * ECRITURE *
4002 REM ============
4003 :
4010 IF MS<>8192 THEN K=64:GOTO 4030
4020 K=0
4030 :
4050 PRINT D$"Write dessin"
4060 FOR P=1 TO 40
4070 PRINT B(P)
4080 NEXT P
4150 PRINT D$:PRINT:RETURN
```

truction capable de lire l'état d'un point (ou la couleur), il faudra restructurer les boucles en éliminant la boucle externe et en conservant les deux autres, une pour les lignes, l'autre pour les colonnes.

Calcul de zones et de périmètres

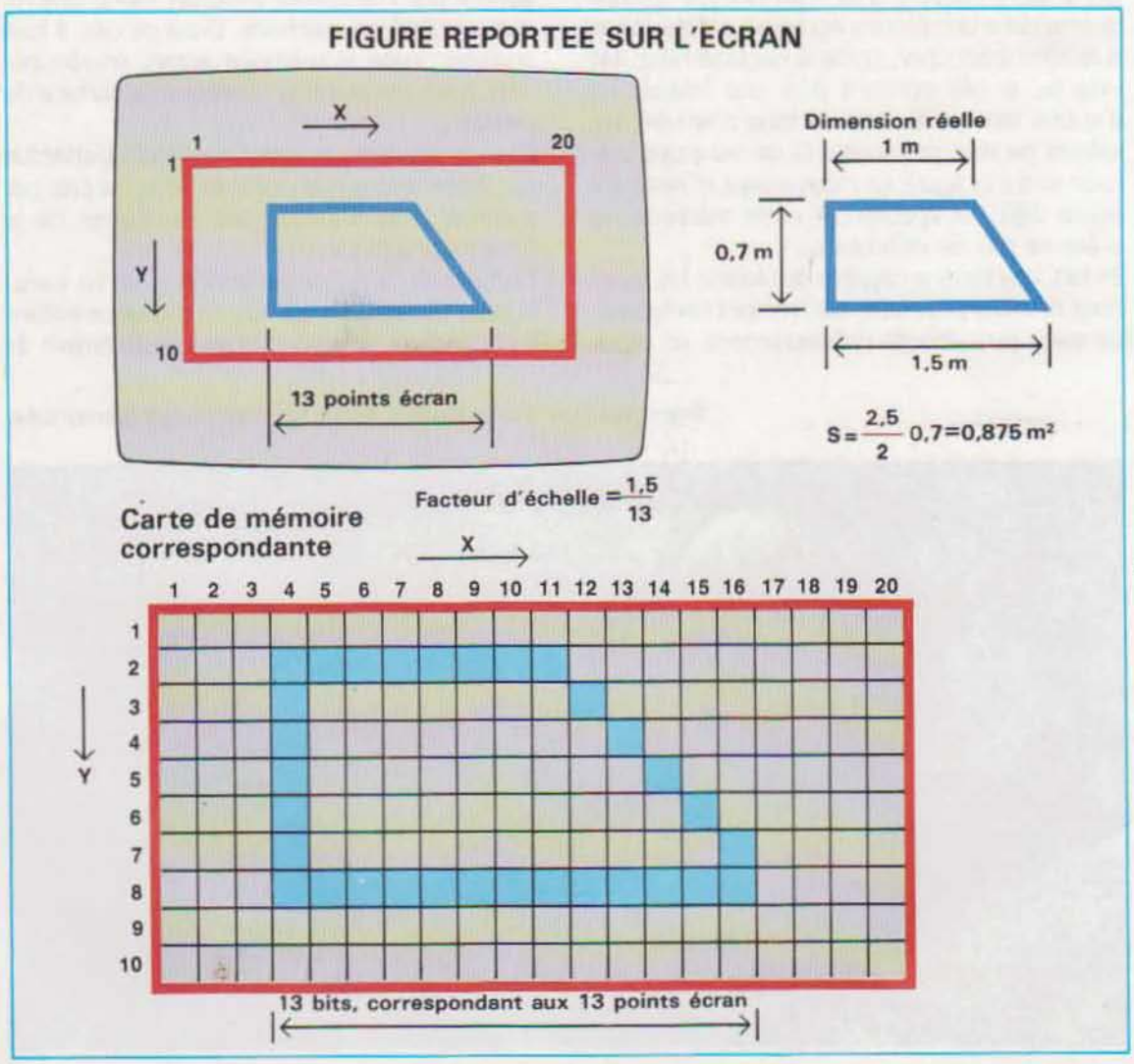
La lecture de l'état d'un point, ou mieux, de la mémoire qui le contient, peut se révéler indispensable dans certaines applications comme l'écriture de logiciels pour les jeux vidéo. Dans ces programmes, il est fréquent de déplacer une figure en vérifiant d'éventuelles collisions avec d'autres figures présentes à l'écran.
Dans certaines machines, ce contrôle s'effectue avec une instruction Basic, qui existe rarement dans les ordinateurs personnels, d'emploi plus général.
On a alors recours à la méthode de lecture directe dans la mémoire écran pour déterminer si la zone à occuper, après le déplacement, est vide ou si elle contient déjà une image. En d'autres termes, si, dans la zone d'arrivée, les valeurs ne sont pas toutes 0, on aura une collision entre la figure en mouvement et celle qui existe déjà. En apparence, cette méthode ne présente pas de difficultés.
En fait, une routine capable de réaliser les fonctions décrites peut être relativement complexe. Ce sujet sera abordé ultérieurement, en montrant la différence entre la réalisation de la fonction au moyen du logiciel d'application et ce que l'on peut obtenir de machines disposant de dispositifs spéciaux.
Un problème semblable à celui de la collision, mais bien plus simple, concerne la mesure des surfaces et des périmètres. Dans beaucoup d'applications graphiques, après avoir exécuté le dessin, on doit procéder à une phase de traitement pour mesurer les dimensions de l'objet représenté. Le dessin une fois terminé, toutes les dimensions sont présentes dans la mémoire graphique et il n'est donc pas difficile de concevoir quelques sous-programmes pour le calcul des caractéristiques géométriques.
Le problème peut être abordé de deux façons, selon le mode de saisie du dessin.
La première méthode prévoit que le dessin, généré par l'utilisateur avec un menu graphique, est déjà en mémoire. Dans ce cas, il faut analyser toute la mémoire écran, en récupérant, point par point, le contour et la surface du dessin.
Dans la seconde, le calcul du dessin s'effectue en même temps que sa saisie : c'est le cas, par exemple, des introductions au moyen de la tablette graphique.
Comme on l'a vu, ce périphérique est un transducteur de position qui fournit à chaque instant à l'ordinateur les coordonnées de l'élément de

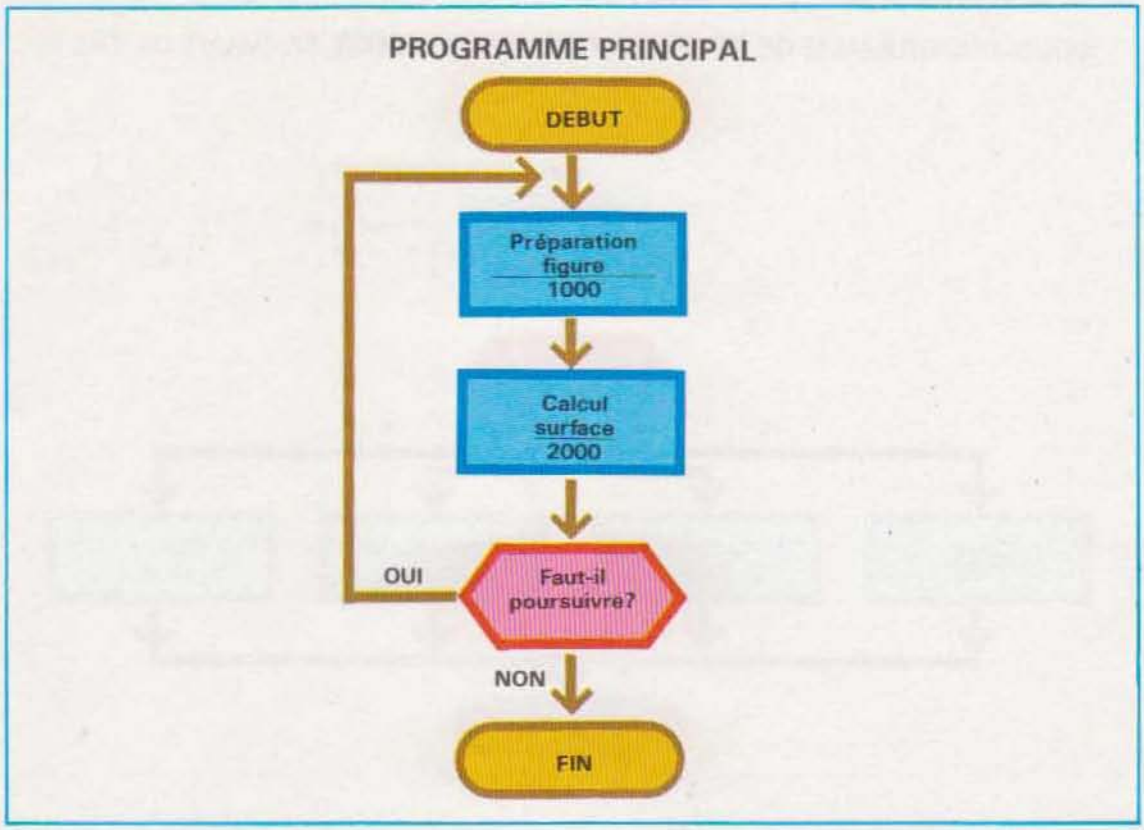
Superposition d'une fenêtre écran sur une image mémorisée.



Centro THC

contact ; le dessin est saisi en déplaçant cette référence le long du contour de la figure.
De cette manière, le calcul du périmètre est immédiat. Il consiste à mesurer la distance entre les divers points, en schématisant chaque déplacement par un segment de droite. Il est donc d'autant plus précis que les déplacements sont petits.
En réalité, l'utilisateur ne se rend pas compte de ce caractère discret le long du contour, le mouvement de référence est continu : c'est le rôle du logiciel de le diviser en éléments suffisamment petits pour assurer une bonne précision, sans toutefois trop ralentir l'exécution du programme.
La surface est calculée une fois que le dessin est terminé. Dans ce cas aussi, la méthode la plus simple consiste à examiner la mémoire écran et à effectuer le calcul sur l'image mémorisée (voir graphique en bas de page).
Pour illustrer la méthode, nous avons choisi un trapèze dont la surface calculée mathématiquement vaut 0.875 m² (les dimensions sont en mètres). Supposons l'image saisie et représentée avec la grande base (13 points écran) et un facteur d'échelle de 1,5/13 ; chaque point écran vaut 0,11...m et la hauteur en points écran 6,06... (hauteur en point écran = 13/1.5 * 0.7).
Ce dimensionnement, choisi pour des raisons graphiques, se révèle inefficace. En réalité, si l'on doit représenter ce dessin, l'échelle la plus adaptée est 100 ; ainsi la base devient 1.5 × 100 = 150 points écran et la hauteur 70 ; on obtient une meilleure précision.
Dans la partie inférieure du schema ci-dessous,





on montre qu'on obtient une carte mémoire correspondant à l'image. Les numérotations ne sont pas celles de l'écran, mais sont relatives à une zone de l'écran autour de la figure ; cependant, la présentation de la méthode n'en demeure pas moins valable. Le calcul de la surface consiste à examiner l'aire de mémoire concernée et à déterminer si chaque point se trouve, ou non, à l'intérieur de la figure. La somme des points intérieurs fournit la surface à l'échelle.

L'exploration de la mémoire se fait selon deux boucles, la plus externe selon l'axe Y (dans l'exemple de 1 à 10) et la plus interne selon l'axe X (de 1 à 20).

En posant Y = 1, et en examinant tous les emplacements de mémoire appartenant à cette ligne, on ne repère aucun bit actif ; donc sur la ligne Y = 1 il n'existe pas de points dessinés. Pour Y = 2, on rencontre en colonne 4 le 1er bit actif, puisque c'est le 1er mémorisé (dans la variable X1) comme début d'une ligne interne de la figure. En continuant l'exploration de la ligne, chaque fois que l'on rencontre un bit actif (après le 1er) sa position est rangée dans la variable XF. A la fin de la boucle interne (X de 1 à 20), on aura XF = 11 et la différence XF — XI fournit la longueur de la 1re ligne interne (en particulier pour Y = 2 : cette ligne est la petite base du trapèze examiné).

L'échelle du dessin étant 1,5/13, la longueur du segment en dimensions réelles est (1.5/13) * (XF-XI) = (1.5/13) * 8 = 0.92... m.

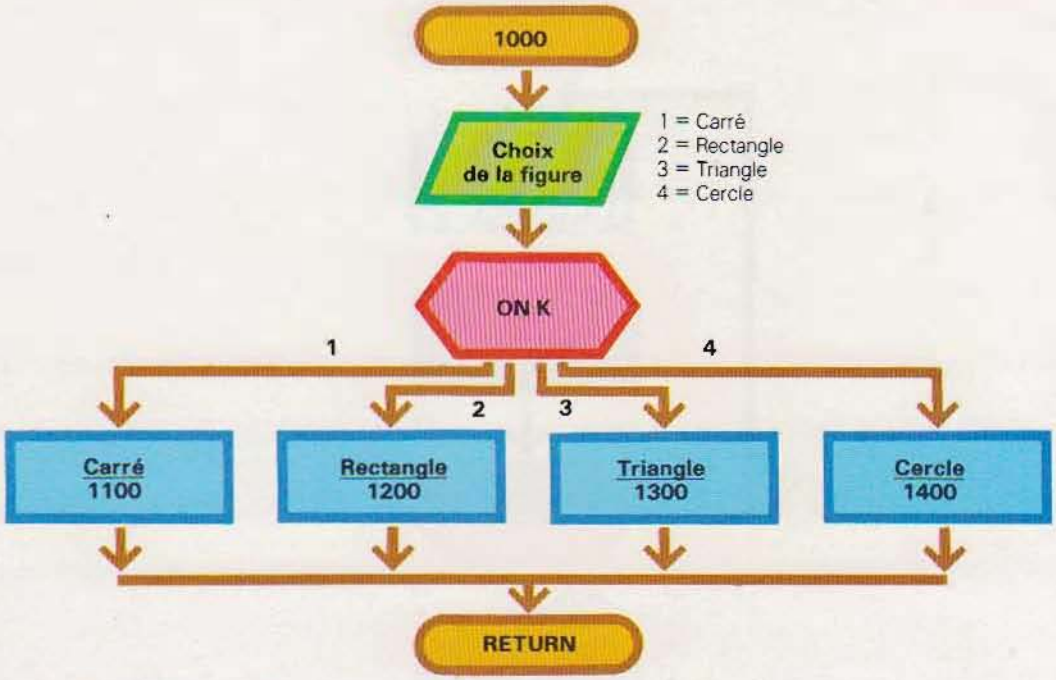
Remarquons la différence avec la mesure réelle de la petite base (1 m), due au facteur d'échelle adopté. Avec la valeur 1,5/13, la petite base devrait être longue 8.66... points écran, mais elle doit être ramenée à 8, avec l'erreur qui en découle. (o)

En choisissant une échelle mieux adaptée, cette erreur devient négligeable. Le segment ainsi mesuré (longueur : 0,92) a une dimension, selon l'axe Y, égale à un point écran, c'est-à-dire, 1,5/13 mètres. En multipliant cette quantité par la longueur, on obtient la valeur du premier élément de la surface.

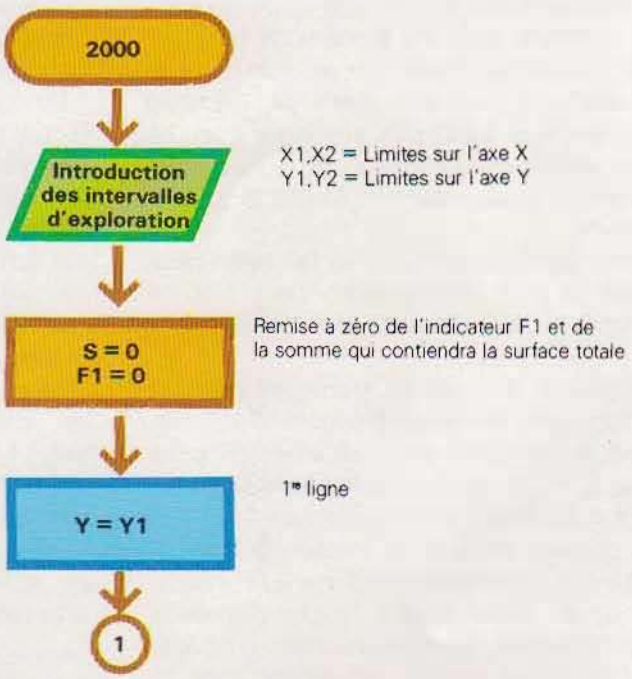
Le processus se poursuit pour toutes les lignes en additionnant chaque élément de surface ainsi trouvé.

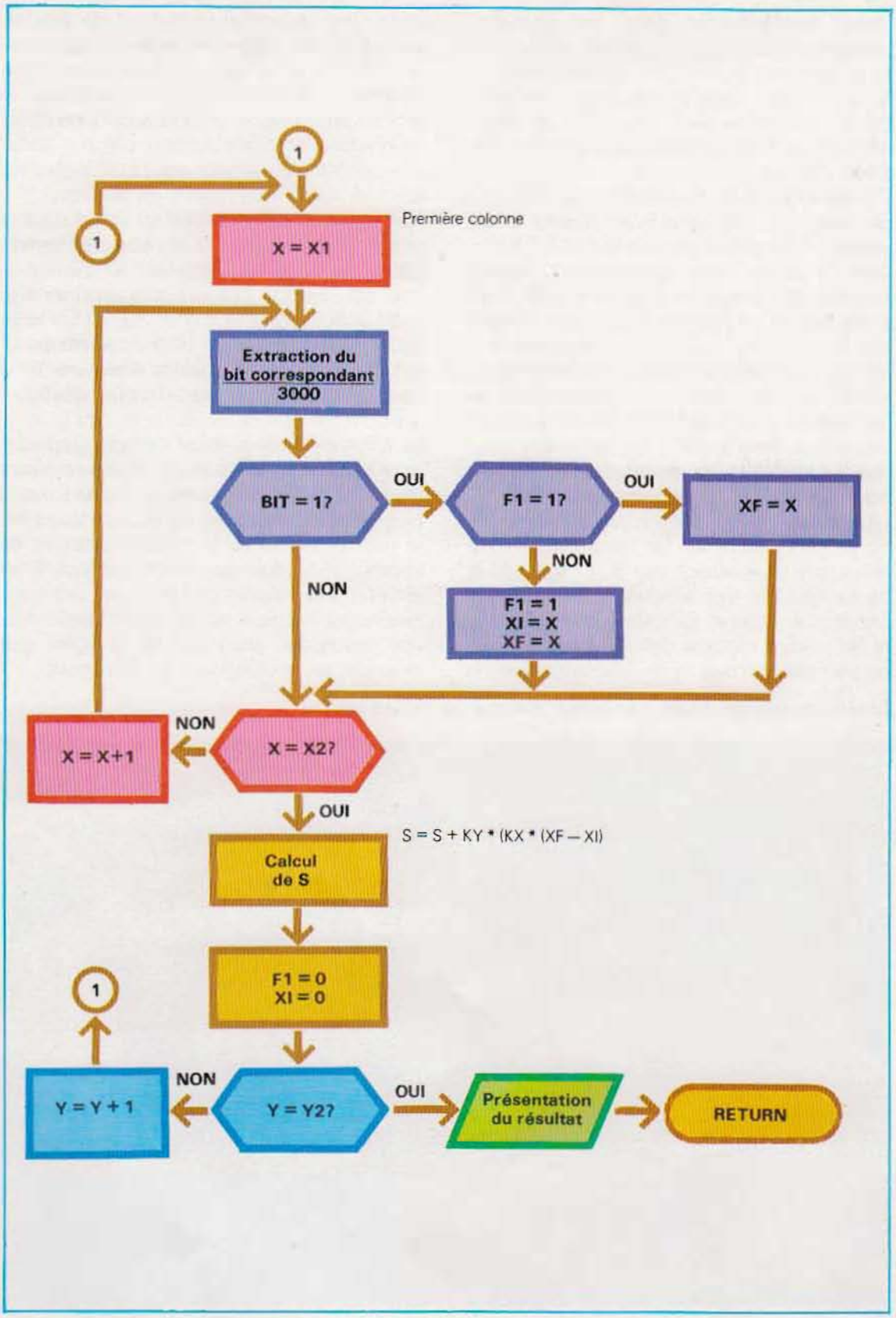
(o) La mémoire de l'écran et l'écran ne peuvent être adressés avec des valeurs réelles.

SOUS-PROGRAMME DE PREPARATION DE LA FIGURE SERVANT DE TEST



SOUS-PROGRAMME DE CALCUL DE LA SURFACE



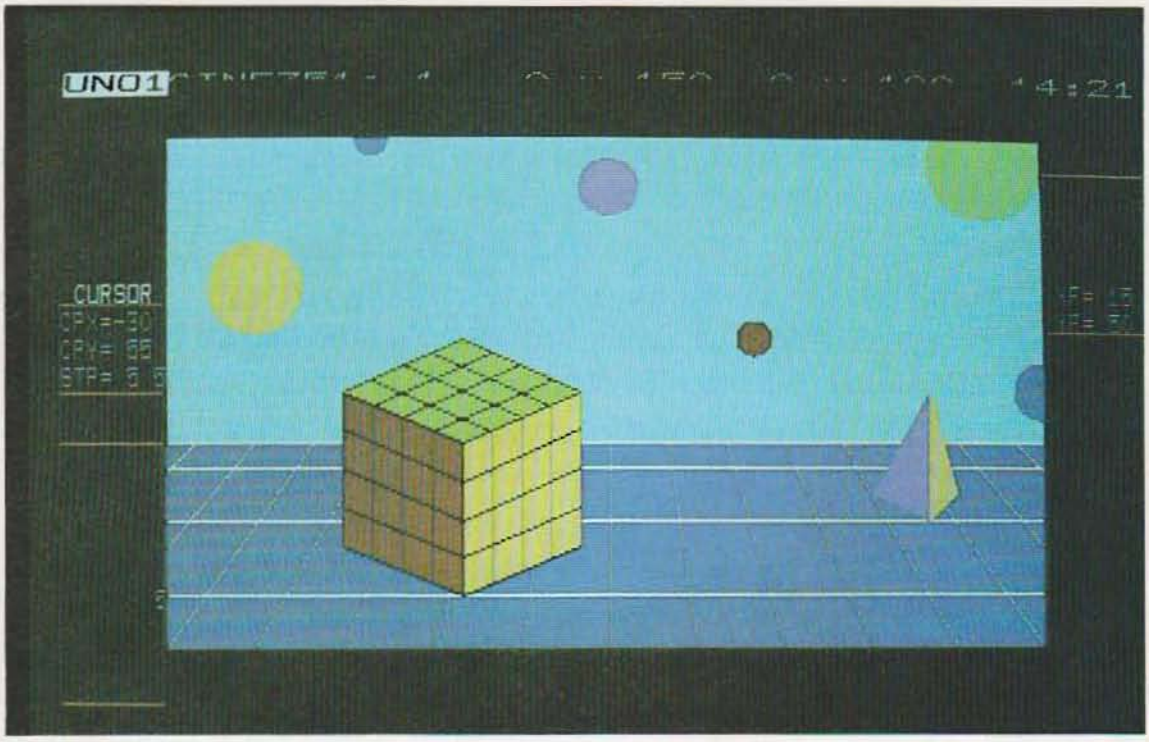

1
1
Première colonne
X = X1
Extraction du bit correspondant 3000
BIT = 1?
OUI
F1 = 1?
OUI
XF = X
NON
NON
F1 = 1
XI = X
XF = X
X = X+1
NON
X = X2?
OUI
S = S + KY * (KX * (XF — XI)
Calcul de S
F1 = 0
XI = 0
1
Y = Y + 1
NON
Y = Y2?
OUI
Présentation du résultat
RETURN

Notre problème est traité par le sous-programme 2000, les autres étant chargés de la préparation de certaines figures géométriques à l'écran. Dans le programme, on a recours, avec quelques variantes, au sous-programme déjà présenté pour le calcul de l'emplacement de mémoire.
Contrairement à la précédente qui avait un but démonstratif, cette version est réduite à l'essentiel à l'exception des lignes 3142 à 3150. Dans ce cas particulier, le temps de traitement est important surtout en Basic interprété. Pour éviter de longues attentes, nous avons introduit ces lignes (3142 à 3150) qui remplissent le contour de l'image au fur et à mesure du traitement. Toujours en suivant la même logique, on a introduit l'indicateur F2 qui peut être supprimé, même dans la 2071. Cet indicateur interrompt l'activation des points écran quand les coordonnées sont extérieures à la figure.
La logique employée est volontairement simple et peu fonctionnelle. Le remplissage n'est exact que pour autant que la 1re ligne de la figure n'est pas trop éloignée de la verticale.
De plus, les figures géométriques présentées et les valeurs relatives des surfaces calculées ne sont pas précises et en adéquation avec le reste du programme. Le but est de montrer comment se servir simplement du sous-programme 2000 tout en évitant des complexités en d'autres points. En particulier, le dessin des figures et la surface calculée par le programme ne correspondent pas aux surfaces calculées analytiquement. La différence est liée à la manière de réaliser les dessins.
Pour le carré, par exemple, en introduisant la valeur 10 pour le côté, et, comme origine de la figure, le point de coordonnées 10, 10, le premier côté vertical s'obtient en traçant un segment entre l'origine et le point 10, 20. De cette manière, la longueur du côté en points écran est 11 ; la surface résultante diffère de 10% de celle calculée : cet écart disparaît si la figure est correctement dessinée.
La méthode présentée est de type graphique dans la mesure où la surface obtenue provient du traitement graphique de la figure. Il existe aussi les méthodes analytiques avec lesquelles le résultat (calcul de la surface) s'obtient en appliquant des formules mathématiques. Mais elles ne sont valables que dans des cas particuliers (par exemple, polygônes) et demandent une description analytique de la figure (par exemple, les coordonnées des sommets).

Représentation graphique d'un cristal cubique sur ordinateur.

J. Pickerell/Marka





Centre THC

Image graphique tridimensionnelle (3 D) sur écran couleur.

PROGRAMME DE CALCUL DE SURFACES

Version Siprel 240, Apple et compatibles

```
10 REM
20 REM fichier=surfaces
30 REM
40 KY=1:KX=1
50 GOSUB 1000
60 GOSUB 2000
70 REM
80 INPUT "Poursuivre (O/N) ";A$
90 IF A$="O" GOTO 50
92 END
1000 REM figure d'essai
1005 REM
1010 HGR
1015 PRINT "Sélection(1-2-3-4)?"

1020 GET R
1025 IF R<1 OR R>4 GOTO 1015

1030 ON R GOSUB 1100,1200,1300
     ,1400
1040 RETURN
1050 REM
1052 REM ■■■■■
1054 REM
1100 REM carré
1105 REM
1110 INPUT "Coté ";L
1115 INPUT "Position (X,Y) ";
     X0,Y0
1120 HPLOT X0,Y0 TO X0+L,Y0
     TO X0+L,Y0-L TO X0,Y0-L
     TO X0,Y0
1121 A=L*L
1122 PRINT "Surface = ";A
1125 RETURN
1130 REM
1132 REM ■■■■■
1134 REM
1200 REM rectangle
1210 INPUT "Côtés (L,l) ";LX,LY
1215 INPUT "Position (X,Y) ";X0
     ,Y0
1220 HPLOT X0,Y0 TO X0+LX,Y0
     TO X0+LX,Y0-LY TO X0,Y0-LY
     TO X0,Y0
1221 A=LX*LY
1222 PRINT "Surface = ";A
1225 RETURN
1230 REM
1232 REM ■■■■■
1234 REM
1300 REM triangle
1305 REM
1310 INPUT "Base et hauteur ";B
     ,H
1315 INPUT "Position (X,Y) ";X0
     ,Y0
1320 HPLOT X0+B,Y0 TO X0,Y0 TO
     X0,Y0+H TO X0+B,Y0
1321 A=B*H/2
1322 PRINT "Surface = ";A
1325 RETURN
1330 REM
1332 REM ■■■■■
```

```
1334 REM
1400 REM cercle
1405 REM
1410 INPUT "Rayon ";R
1415 INPUT "Centre (X,Y) "
     ;X0,Y0
1520 X1=X0+R:Y1=Y0
1525 FOR A=-0.05 TO 6.5 STEP
     0.05
1530 X2=X0+R*COS(A)
1535 Y2=Y0+R*SIN(A)
1540 HPLOT X1,Y1 TO X2,Y2
1545 X1=X2:Y1=Y2
1550 NEXT A
1555 A=3.14*R^2
1556 PRINT "Surface = ";A
1560 RETURN
2000 REM
2005 REM surfaces
2010 REM
2015 INPUT "Limites axe X "
     ;X1,X2
2020 INPUT "Limites axe Y "
     ;Y1,Y2
2025 S=0:F1=0:XI=0:XF=0
2030 FOR Y=Y1 TO Y2
2035 FOR X=X1 TO X2
2040 GOSUB 3000
2045 IF BIT=0 GOTO 2050
2050 IF F1=1 THEN XF=X:GOTO
     2060
2055 XI=X:XF=X:F1=1
2060 NEXT X
2062 PRINT "Y = ";Y;"XI = "
     ;XI;"XF = ";XF
2065 S=S+KY*(KX*(XF-XI))

2070 F1=0:XI=0.XF=0
2071 F2=0
2075 NEXT Y
2080 PRINT "Surface = ";S
2082 GET D$
2085 RETURN
2090 REM
2092 REM *****
2094 REM
3000 REM extraction du bit
3005 REM
3010 REM entrées:
3015 REM     X,Y       X,Y
3016 :
3020 REM
3025 REM sorties:
3030 REM    BIT=0,1   BIT=0,1
3035 REM
3036 :
3040 GY=INT(Y/8)
3045 IF Y<64 THEN MR=8192+GY*
     128:GOTO 3070
3050 IF Y<128 THEN MR=8232+(GY
     -8)*128:GOTO 3070
3060 MR=8272+(GY-16)*128
3070 A=Y-GY*8
3080 MY=MR+A*1024
3090 GX=INT(X/7)
3100 MX=MY+GX:BIT=X-7*GX

3105 A%=PEEK(MX):B%=2^BIT

3110 NN=A%
3115 FOR K=0 TO BIT
3120 M=NN/2
3125 NN=INT(M)
3130 NEXT K
3135 BIT=0
3136 IF M<>NN THEN BIT=1
3142 IF F1=0 THEN RETURN
3143 C%=A%+B%
3145 IF F1=1 AND BIT=1 THEN
     F2=1
3146 IF F2=1 THEN RETURN
3147 IF BIT=1 THEN RETURN
3150 POKE MX,C%
3152 RETURN
3160 REM
3162 REM *****
3164 REM
```

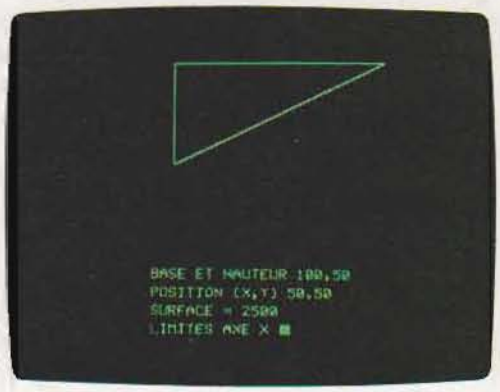
Les méthodes analytiques concernent donc plus la programmation scientifique que graphique car elles exigent quelques connaissances scientifiques.
La méthode présentée maintenant n'utilise aucune formule et s'applique à toute figure plane mais demande un temps de traitement certain des boucles de contrôle de la page graphique. Pour réduire ce temps d'attente on peut :

1/ se servir de la version compilée du programme
2/ et/ou circonscrire la zone de mémoire à explorer à l'intérieur de la figure.

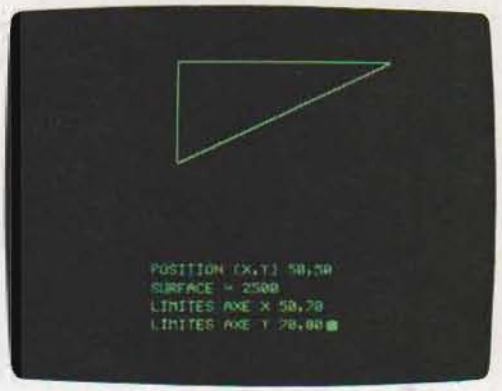
C'est pourquoi le sous-programme 2000 interroge sur les limites à poser dans l'exploration de l'écran le long des deux axes : ainsi les zones sans figures ne sont pas analysées. Autre avantage : ces limites permettent d'analyser plusieurs figures simultanées à l'écran. Il suffit d'appeler le sous-programme autant de fois qu'il y a de figures, en posant pour chacune d'elles les limites d'exploration appropriées. Attention : l'entrée des données doit bénéficier d'une attention particulière, car un mauvais calcul entraînerait de graves erreurs. Là encore, une série de contrôles s'avère nécessaire. Le plus simple est de dessiner, sur l'écran, une grille indiquant les coordonnées. Pour cela, il suffit de dessiner un rectangle en suivant les bords de l'écran et en marquant les côtés de référence, par exemple tous les dix points. Le programme exécutant ce tracé est le même que pour le tracé des axes cartésiens ou du cadre des histogrammes.
Une autre technique (analytique) consiste à mémoriser les coordonnées maximales et minimales au fur et à mesure que le dessin est créé.

PROGRAMME DE CALCUL DE SURFACES

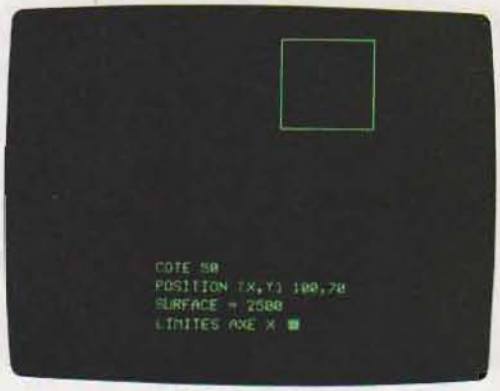
Examinons quelques exemples d'application du programme présenté pages 1561 et 1562.
L'écran montre ce qui apparaît après l'introduction de la valeur 3 en réponse à la ligne 1015. Le contrôle passe au sous-programme 1300, qui demande la base, la hauteur et la position du triangle et fournit la valeur analytique de la surface (1321, 1322).



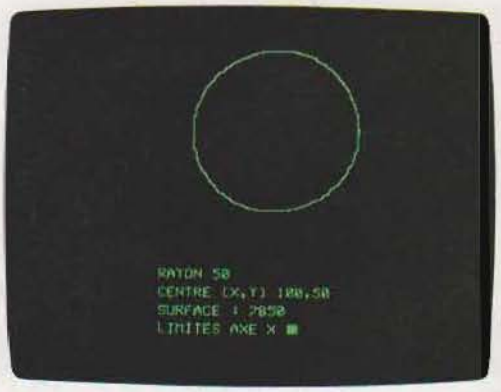
On passe ensuite la main au sous-programme 2000, qui calcule la surface en balayant la mémoire vidéo. Il demande les limites de balayage de l'écran le long des axes X et Y. Cet écran et le précédent illustrent la phase d'entrée de ces limites : l'axe X est exploré de la position 50 à 70, et l'axe Y de la 70 à la 80. A la fin s'affiche la valeur de la surface, déterminée par balayage.



Le 3e écran représente la même situation pour un carré positionné en 100,70 dont le côté mesure 50. La surface calculée analytiquement vaut 2500 ; la valeur déterminée par le programme sera affichée une fois entrées les limites d'analyse de l'écran.



Le dernier écran (cercle). Le système peut, même en mode graphique, réserver les 4 lignes inférieures pour dialoguer avec l'utilisateur.



Pour finir, on explore la surface rectangulaire, limitée par les valeurs maximales et minimales, avec une éventuelle marge. Dans le cas de plusieurs figures c'est à l'utilisateur de signaler au programme quand chacune d'elles est terminée.
Le fait de poser des limites permet d'éviter un autre type d'erreur. C'est le cas, notamment, d'un dessin aux contours mal définis. N'étant pas fermés, l'exploration et le calcul ultérieur du dessin ne s'achèvent qu'au moment où le programme rencontre le bord de l'écran. Résultat : la surface attribuée à cette partie serait bien supérieure à la réalité.
C'est pourquoi a été créée, dans le sous-programme 1400, la boucle utilisée pour tracer la circonférence.
Le cercle devrait être tracé en partant de l'angle 0 jusqu'à l'angle $2*\pi = 6{,}28$ (ce qui équivaut à un tour complet), mais la boucle utilise les limites - 0,05 et 6,3 afin d'assurer la clôture totale de la figure.

Instructions Basic pour l'archivage des figures

La méthode exposée pour obtenir une figure ne fait intervenir aucune instruction particulière. Elle est appliquée à toutes les machines dont les adresses mémoire graphiques sont connues.
Dans certains cas il existe des instructions de haut niveau et surtout transparentes, c'est-à-dire qui ne nécessitent aucun adressage, du moins en-dehors de la définition des coordonnées graphiques de la zone à mémoriser ou à redessiner.
Ce type d'instructions est étroitement lié à la structure de la machine. De ce fait, leur syntaxe, parfois même les fonctions exécutées, sont particulières.
Voici par exemple les instructions disponibles sur l'Olivetti M20 :

POINT : Enregistre la couleur d'un point
GET : Mémorise une zone écran
PUT : Visualise l'image enregistrée avec une instruction GET

Les instructions GET et PUT utilisées ici sont différentes — dans leur syntaxe et dans leur objet — de celles employées avec des fonctions d'E/S.

POINT. Son format est :
A% =POINT(X,Y)

A l'exécution, le programme assigne à la variable entière A% une valeur numérique dépendant de la couleur du point de coordonnées X,Y. Cette valeur est 0 ou 1 pour un écran monochrome (0 = éteint, 1 = allumé), ou comprise entre 0 et 3 pour un écran 4 couleurs ou 0 et 7 pour le 8 couleurs.

Les valeurs des coordonnées X,Y peuvent aussi bien se rapporter au matériel, c'est-à-dire correspondre à des points de l'écran, qu'être définies par l'utilisateur. Avec ce type de machine, il est en effet admis de recourir à un système de référence différent de celui qui définit les points écran ; il emploie alors la **référence utilisateur**.

POINT s'emploie également dans une autre instruction graphique, mais dans un autre sens pour avoir deux curseurs, l'un de texte et l'autre graphique. La gestion du premier est identique à celle des autres machines, bien que les instructions soient formellement différentes. Cependant, celle du curseur graphique est peu courante.
Le format le plus simple de cette instruction est :

CURSOR POINT(X,Y)N,M

N et M indiquant un point visible (N=1) ou non visible (N=0) ainsi que la fréquence du clignotement du curseur (pour M=0, il ne clignote pas).

GET. Son format simplifié est :

GET(X_1,Y_1) - (X_2,Y_2),A%(0)

X_1,Y_1 et X_2,Y_2 définissant les sommets de la portion rectangulaire d'écran à mémoriser, et A%(0) le premier élément du tableau dans lequel seront rangées les valeurs. (N'oublions pas qu'en Basic, l'indice initial d'un tableau est 0). Une fois l'instruction exécutée, tous les bits correspondant à la section de l'écran définie par les coordonnées sont stockés dans A%(.). Outre l'état de chacun des bits, A%(.) contient les informations suivantes :

A%(0) : Base du rectangle défini par X_1,Y_1 et X_2, Y_2
A%(1) : Hauteur du rectangle
A%(2) : Ecran monochrome ou couleur.

A partir de A% (3), les adresses sont utilisées pour stocker les bits de la zone graphique décrite. Le dimensionnement du tableau A%(.) dépend de l'extension, en pixels, de la zone à archiver ; elle se calcule comme suit :

Dimension=((⌈base/16⌉*hauteur)*K)+3

K valant 1 pour un écran monochrome, 2 pour un écran 4 couleurs, et 3 pour un écran 8 couleurs, tandis que le quotient base/16 doit toujours être arrondi par excès (vers le haut).
L'ordinateur M20 admet une partition de l'écran en portions appelées fenêtres. Chacune d'elles est utilisable indépendamment des autres, par simple indication de son numéro.
La syntaxe complète de l'instruction GET est :

GET W% (X_1,Y_1) - (X_2,Y_2), A%(0)

W% étant la valeur numérique de la fenêtre (ce sujet sera traité plus loin).

PUT. Son format est :

PUT W%(X_1,Y_1) - (X_2,Y_2), A%(0),V

Là encore, le paramètre W% (fenêtre) est optionnel. Par rapport à l'instruction précédente, PUT contient en plus le paramètre V, qui exprime une opération logique à effectuer entre les numéros de couleur contenus dans le tableau A%(.) et ceux des couleurs présentes à l'écran dans le rectangle défini par les coordonnées spécifiées.
Quand V est omis, le tableau A%(.) est visualisé tel qu'il a été mémorisé. Si V est spécifié, la visualisation obtenue sera le résultat de l'opération logique exprimée par V entre les couleurs prédéfinies et celles de A%(.). L'option V prend les valeurs AND, OR, XOR, NOT, PSET et PRESET. AND, OR et XOR ont leur sens habituel. NOT indique le complément des numéros des couleurs présentes à l'écran. PSET ressort le tableau tel qu'il a été enregistré. PRESET donne le complément des numéros de couleur du tableau.

Les fenêtres

Dans bon nombre d'applications, il n'est pas inutile d'avoir un écran divisé en zones gérées séparément les unes des autres. Appelées **fenêtres,** ces zones se comportent, du point de vue du programme d'application, comme autant de petits écrans séparés. Elles sont le plus souvent utilisées pour présenter, simultanément, un dessin d'ensemble et des détails de celui-ci (grossissements).
Par exemple, pour tracer le graphique d'une fonction, on peut diviser l'écran en deux ou en trois parties dont une, généralement la plus grande, pour le graphique entier, et les autres pour l'analyse détaillée de petites portions du graphique.
Dans les systèmes capables de générer des fenêtres, l'opération se déroule généralement en deux phases. On commence par assigner des valeurs aux paramètres définissant les fenêtres (dimensions, position, etc.) ; on passe ensuite à la phase de travail sur la fenêtre adressée à l'aide du nombre ou du symbole qui lui a été associé précédemment.
Les possibilités offertes par la gestion des fenêtres varient d'une machine à l'autre. En fait, il en existe 3 grandes catégories :

- les machines dans lesquelles les fenêtres ne sont pas prévues
- les ordinateurs ne permettant de créer que 2 fenêtres, l'une pour le graphique, et l'autre pour le texte
- les ordinateurs permettant une gestion complète des fenêtres, tant du point de vue de leur nombre que du mode (graphique ou texte)

Les deux premières catégories sont pratiquement équivalentes. En effet, la visualisation d'une fenêtre de texte, généralement limitée à quelques lignes, ne peut être utile que dans les cas où l'on veut avoir simultanément un dialogue avec l'utilisateur et l'affichage de la page graphique. C'est le cas, par exemple, des ordinateurs compatibles Apple qui conservent une fenêtre de texte d'une hauteur de 4 lignes (instruction HGR). Il ne s'agit donc pas d'une division en fenêtres graphiques, laquelle reste néanmoins réalisable par l'utilisateur à l'aide de programmes relativement simples.

Programme de génération de fenêtres

Une fenêtre est une zone de l'écran, le plus souvent carrée ou rectangulaire, identifiée à l'aide d'un symbole ou d'un numéro. Pour l'utiliser le programme graphique doit être paramétré, c'est-à-dire qu'il doit gérer tous les positionnements et les déplacements en fonction des paramètres de la fenêtre.

Ce paramétrage n'est évidemment nécessaire que lorsque la partition de l'écran est obtenue à l'aide de programmes écrits par l'utilisateur. Avec les machines prévoyant des fenêtres au niveau du système, le paramétrage est transparent à l'utilisateur; il n'a qu'à appeler la fenêtre de travail.

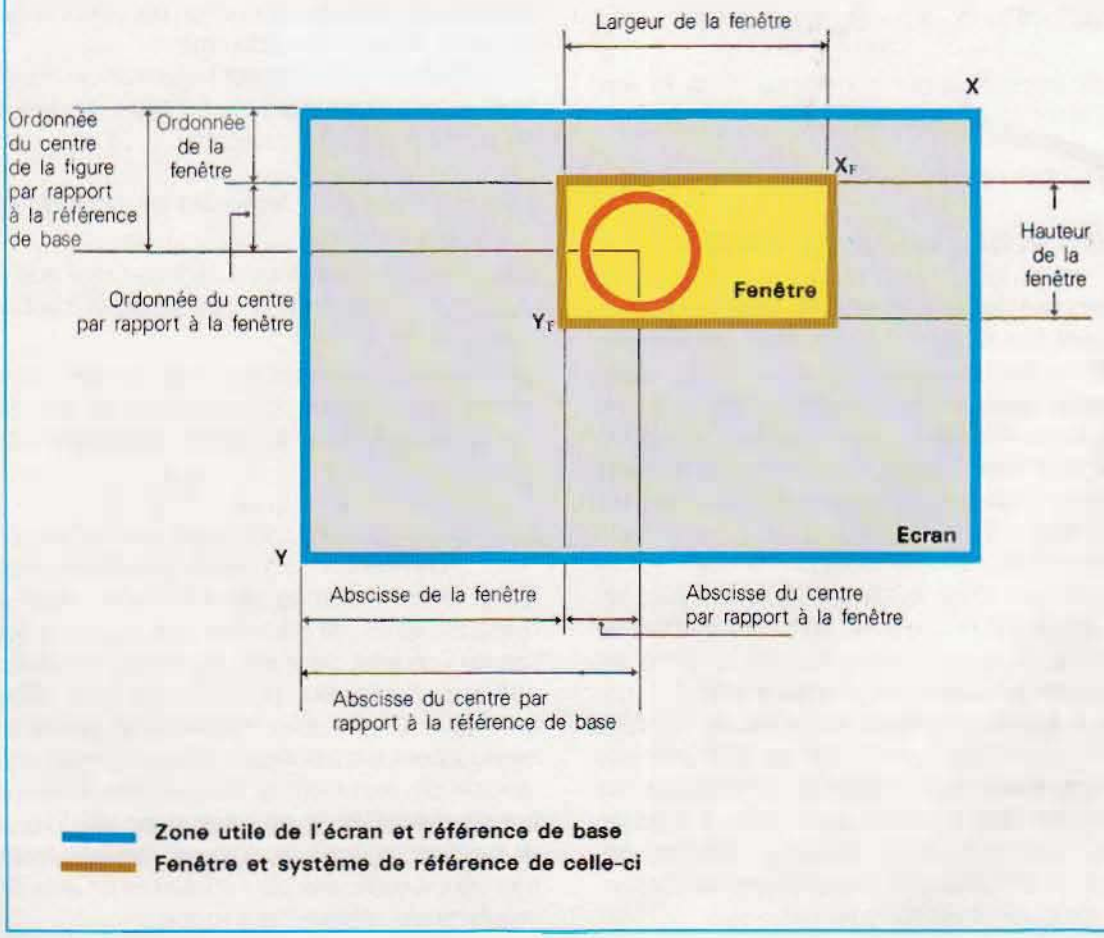
On a représenté ci-dessous le dessin d'une circonférence tracée à la fois par référence au système de base et à un système de référence coïncidant avec deux côtés d'une fenêtre. La gestion d'un dessin, à l'intérieur d'une fenêtre, demande que les coordonnées de construction soient exprimées par rapport au système de référence de la fenêtre.

La présentation d'un cercle de rayon R est obtenue à l'aide d'une boucle qui calcule les coordonnées de chaque point au moyen des formules suivantes :

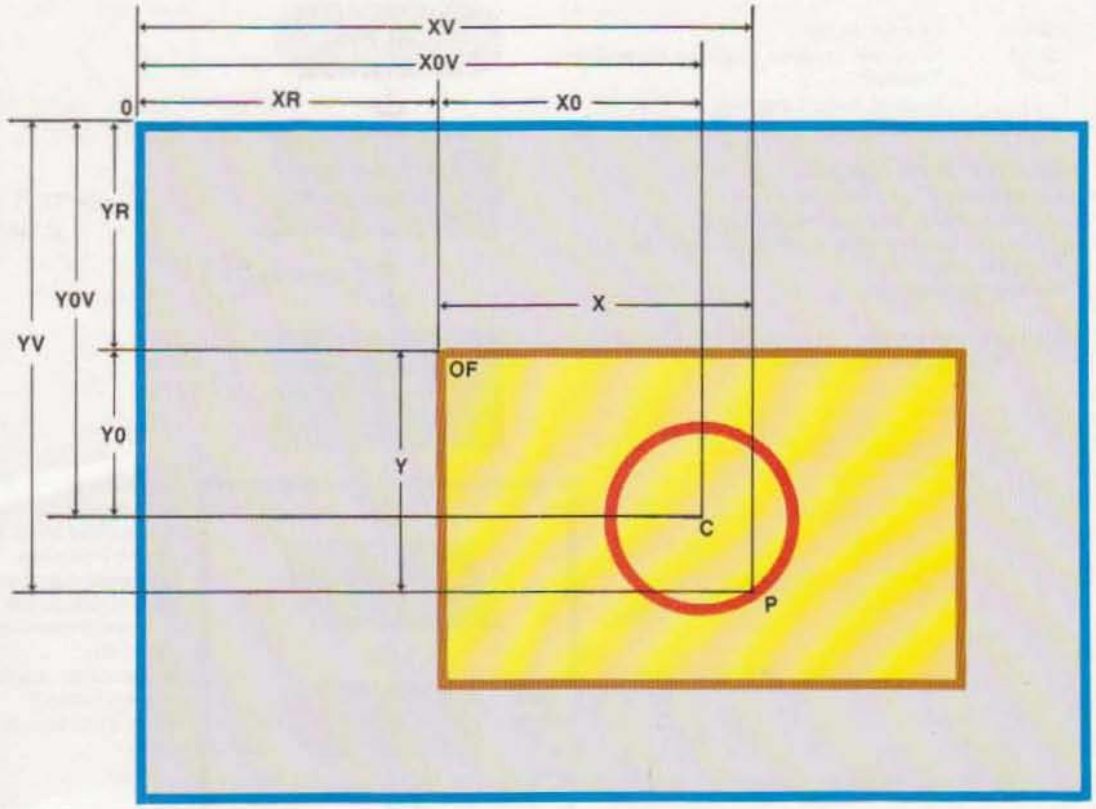
X=XO+R*COS(A)
Y=YO+R*SIN(A)

XO et YO étant les coordonnées du centre.

GESTION D'UNE FIGURE DANS UNE FENETRE



SYMBOLES UTILISES DANS LA GESTION DES FENETRES



Pour transférer le cercle dans la fenêtre, il suffit de définir les coordonnées du centre par rapport à celles de la fenêtre. Les symboles utilisés figurent ci-dessus. Les coordonnées X, Y d'un point quelconque du cercle par rapport à la fenêtre doivent être fournies sous forme de coordonnées XV, YV par rapport au système de base. Celui-ci est utilisé par l'ordinateur pour adresser les déplacements du crayon électrique. Donc :

XV=XR+X
YV=YR+X

De même, si elles se réfèrent au système de base, les coordonnées du centre deviennent :

Y0V=XR+X0
Y0V=YR+Y0

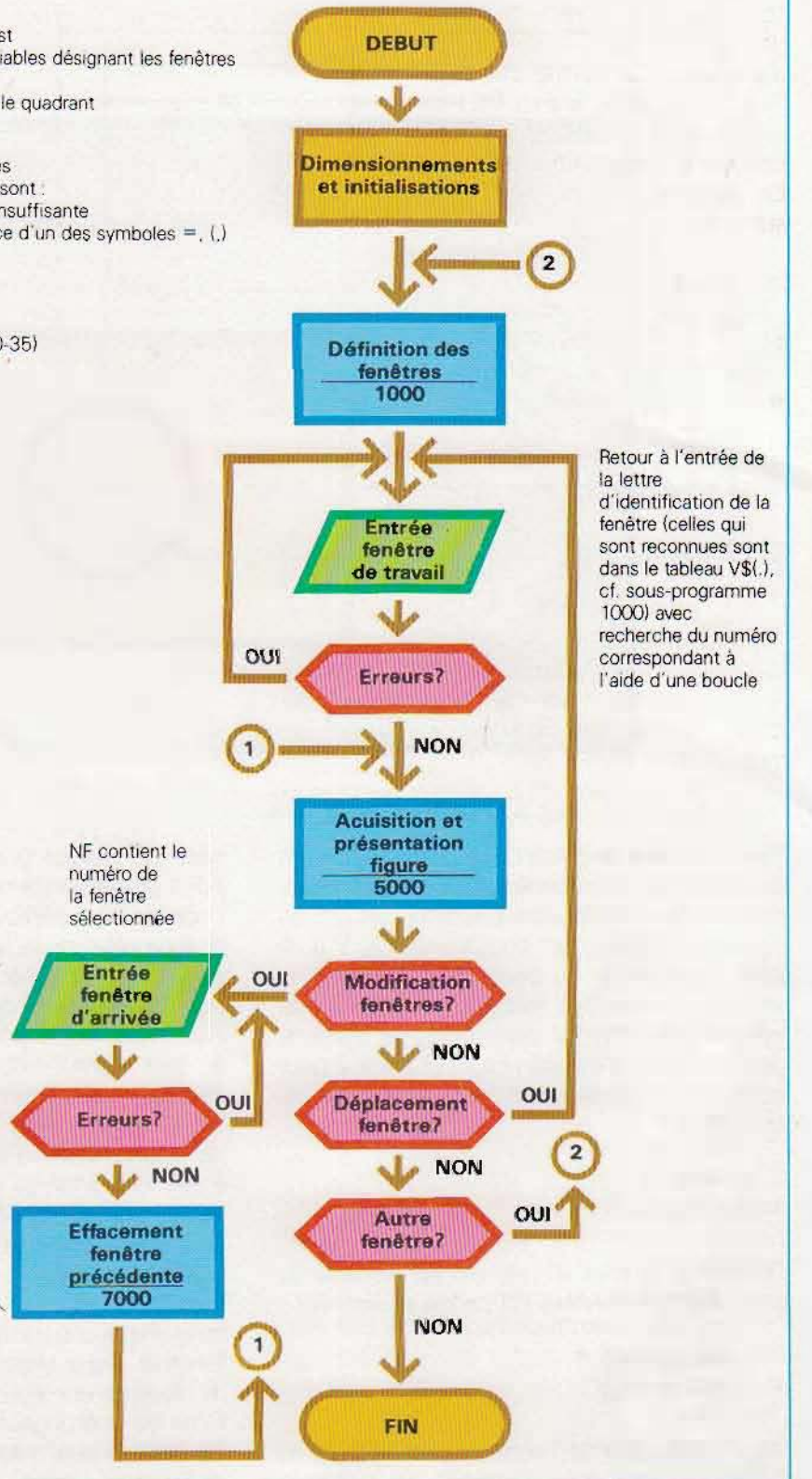
Pour positionner le cercle à l'intérieur de la fenêtre, il suffit donc d'ajouter, aux coordonnées de chaque point, celles de l'origine du point de référence de la fenêtre.
Dans ce cas particulier, les coordonnées de chaque point de la figure renvoient à celles de son centre, et la somme s'effectue aussi bien pour chaque point que pour le centre seul.
Autrement dit, le positionnement d'une figure au sein d'une fenêtre se réduit à une simple translation du système de référence. Ces deux façons de procéder (sur chaque point ou sur les coordonnées d'une seule référence) sont également valables pour des figures n'ayant pas de centre, à condition que les coordonnées de chaque point se réfèrent à un point fixe comme un sommet.
L'organigramme de la page suivante est un programme d'illustration qui manipule plusieurs fenêtres. Dans chacune d'elles, il est possible de dessiner une figure par référence à un système de coordonnées, solidaire de la fenêtre. On dispose ainsi d'autant d'écrans séparés que de fenêtres créées.

GENERATION DES FENETRES

D$(4)	Chaînes de test
V$(16)	Noms des variables désignant les fenêtres
Q(16)	Quadrant
P(16)	Position dans le quadrant
H(16),L(16)	Dimensions

Les chaînes de test, chargées dans une instruction DATA, sont :
1/ commande de longueur insuffisante
2/ erreur de syntaxe, absence d'un des symboles =, (,)
3/ mot-clé erroné
4/ erreur de paramètre

Format de l'instruction :
F=WINDOW(1-3-70-35)



Ce programme est structuré comme un interprétateur, puisque la définition des fenêtres est formulée sous forme de chaîne : le sous-programme 1000 l'interprète en déduisant les paramètres. La forme choisie coïncide avec l'instruction Basic la plus courante pour générer une fenêtre dans les systèmes ayant prévu ce type d'implémentation.
La syntaxe que l'on veut simuler est la suivante :

- une première lettre quelconque identifie la fenêtre en cours de génération
- le symbole =, informe le système que le mot suivant est le mot-clé
- un mot-clé (WINDOW) définit la fonction désirée (création d'une fenêtre)
- une série de paramètres entre parenthèses dessinent la géométrie de la fenêtre.

Ces paramètres sont :

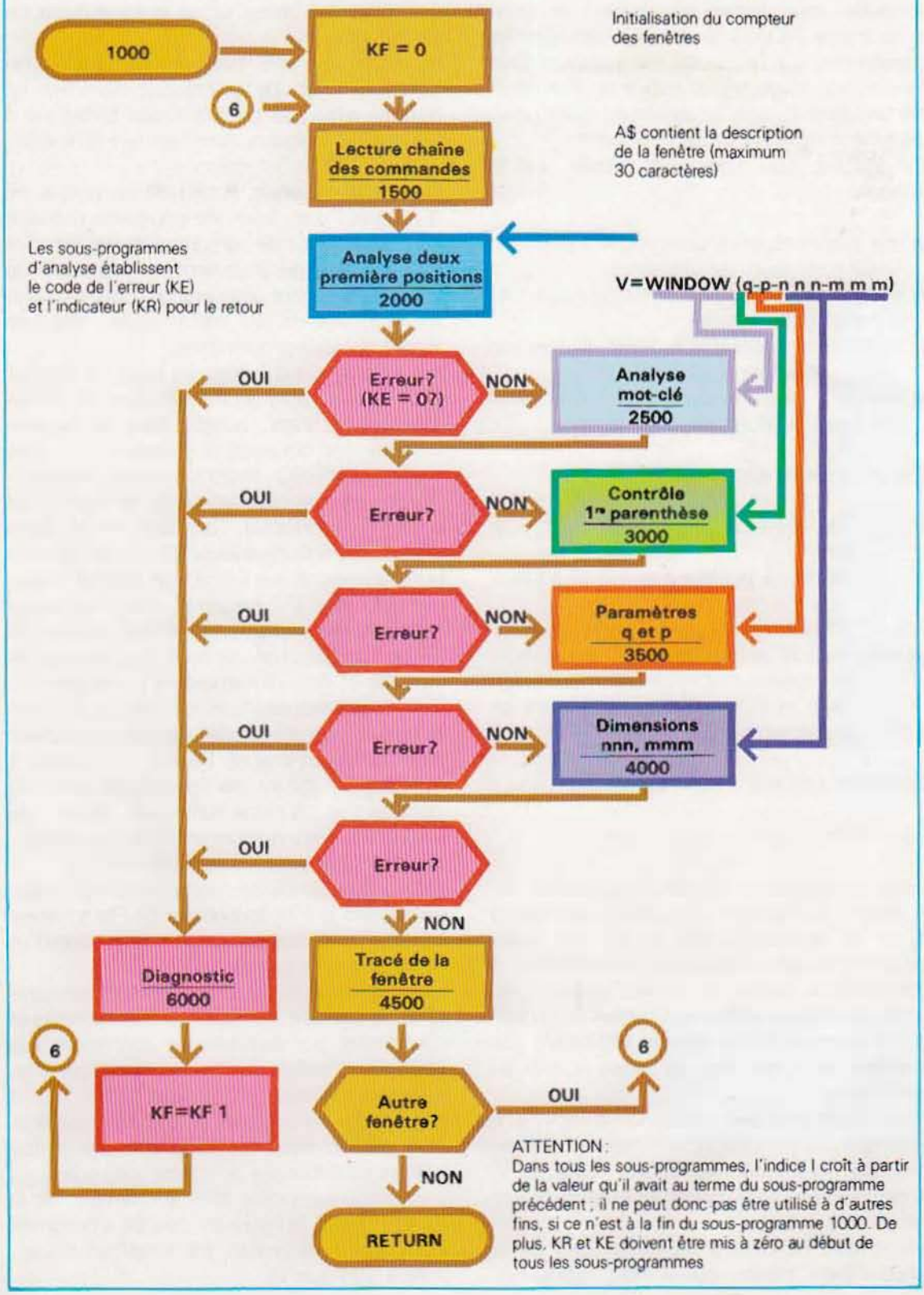
q : Identifie le quadrant de l'écran (1, 2, 3, 4)
p : Donne la position (de 1 à 4) à l'intérieur du quadrant
nnn, mmm : Valeurs numériques constituées chacune d'un maximum de trois chiffres et représentant respectivement la largeur et la hauteur de la fenêtre en points écran.

La chaîne à fournir a donc la forme :

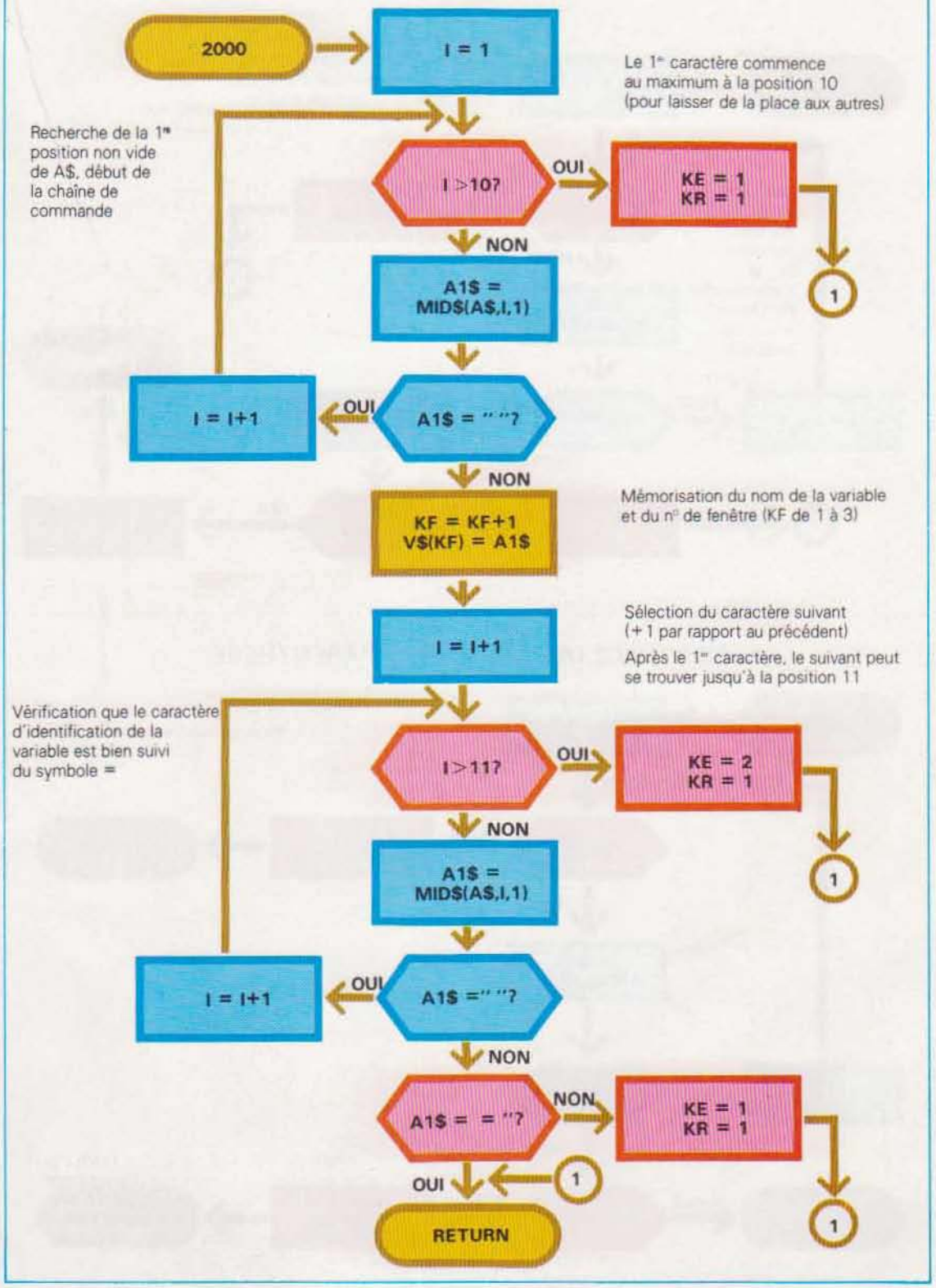
V=WINDOW(1 - 4 - 50 - 20)

Cette formule décrit une fenêtre identifiée par la lettre V, occupant le 1er quadrant en position 4, et de dimensions 50 × 20. Les sous-programmes qui prélèvent les paramètres sont structurés de manière à éliminer les éventuels espaces entre les différentes parties de la chaîne de commande : le mot-clé WINDOW, par exemple, peut être plus ou moins accollé au symbole =.
Seule limite imposée : l'absence d'espaces à l'intérieur des parenthèses contenant la définition des paramètres (ce choix résulte de la nécessité de ne pas compliquer excessivement l'exemple).
Les valeurs définissant les dimensions de la fenêtre (nnn, mmm) contiennent indifféremment de 1 à 3 caractères. Elles seront identifiées grâce à la position du symbole — séparation entre les zones et de la parenthèse de fermeture.
En réalité, une telle méthode n'est pas nécessaire pour créer une fenêtre. Cet exemple a été formulé ainsi afin d'illustrer une technique à laquelle on a recours dans l'écriture d'un interpréteur.
Dans notre exemple, le mot-clé est unique, ce qui permet d'en faire une constante utilisable comme élément de comparaison lors des diagnostics. Plus généralement, on peut créer un tableau contenant une série de mots-clés permettant, lors de leur identification, d'activer autant de sous-programmes.
Les modifications à apporter sont, en théorie, très simples : lors de l'initialisation, les mots-clés prévus sont chargés dans un tableau (comme les diagnostics d'erreur). Le sous-programme 1500 contient une boucle d'identification du mot-clé introduit. Si le résultat est positif (mot identifié), les autres termes de la chaîne sont ensuite analysés, la chaîne se comportant comme une instruction dans un langage particulier à l'utilisateur. Toutefois, cette méthode présente un inconvénient : elle entraîne une prolifération de sous-programmes de contrôle et de prélèvement de paramètres.
Dans le cas examiné (mot-clé unique), les paramètres suivant le mot-clé sont obligatoirement dans un format imposé. Dans le cas général, il faut appeler autant de sous-programmes de contrôle et d'extraction qu'il existe de possibilités ; leur nombre est donc directement proportionnel à celui des mots-clés.
Même tres simplifiée, cette structure reflète néanmoins le fonctionnement de l'interpréteur Basic et est utilisable dans la construction d'un langage orienté graphique.
Le programme fourni est purement démonstratif et ne contient pas les contrôles nécessaires concernant, par exemple, les dimensions des fenêtres et celles du dessin ; ou encore les variations d'échelle. L'utilisateur doit fournir des valeurs correctes en entrée. Les modifications sont toutefois faciles : il suffit de vérifier que les coordonnées de chacun des points du dessin correspondent bien à l'intérieur de la fenêtre. Sinon, la figure est coupée à proximité des bords de la fenêtre (cette opération s'appelle « écrêtage »).

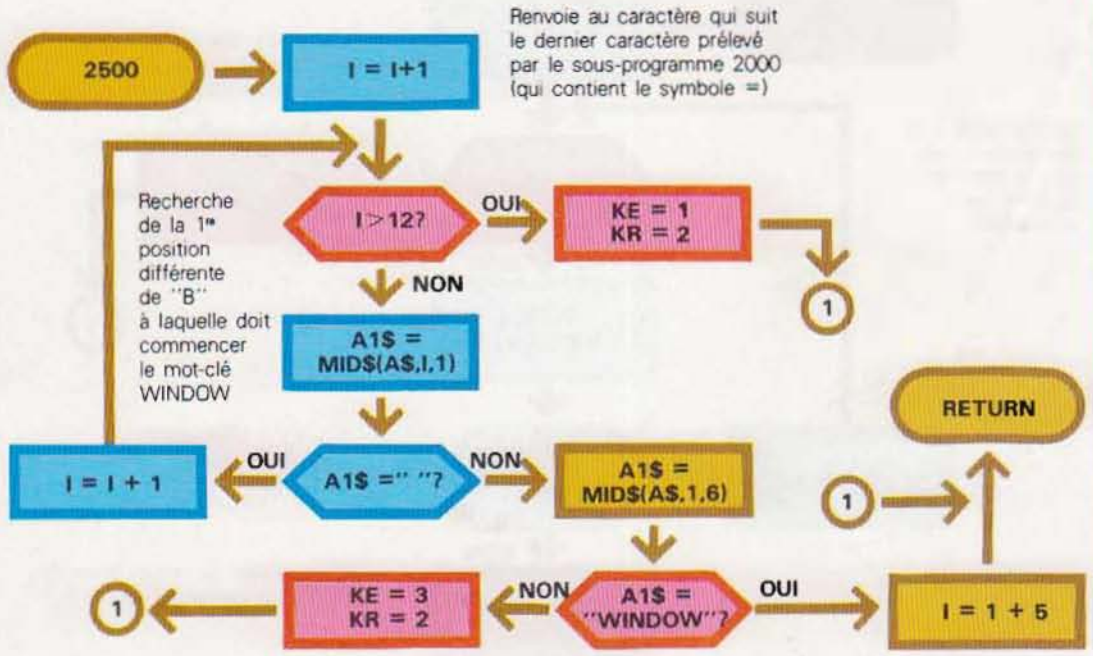
SOUS-PROGRAMME DE DEFINITION DES FENETRES



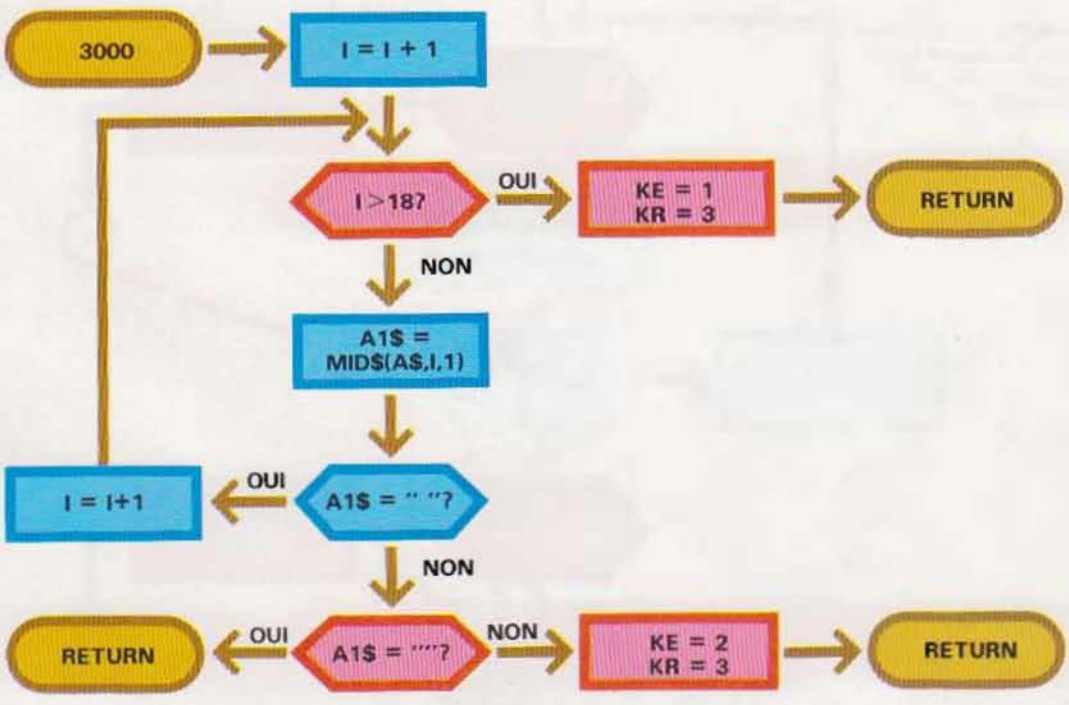
SOUS-PROGRAMME D'ANALYSE DES 2 PREMIERS CARACTERES



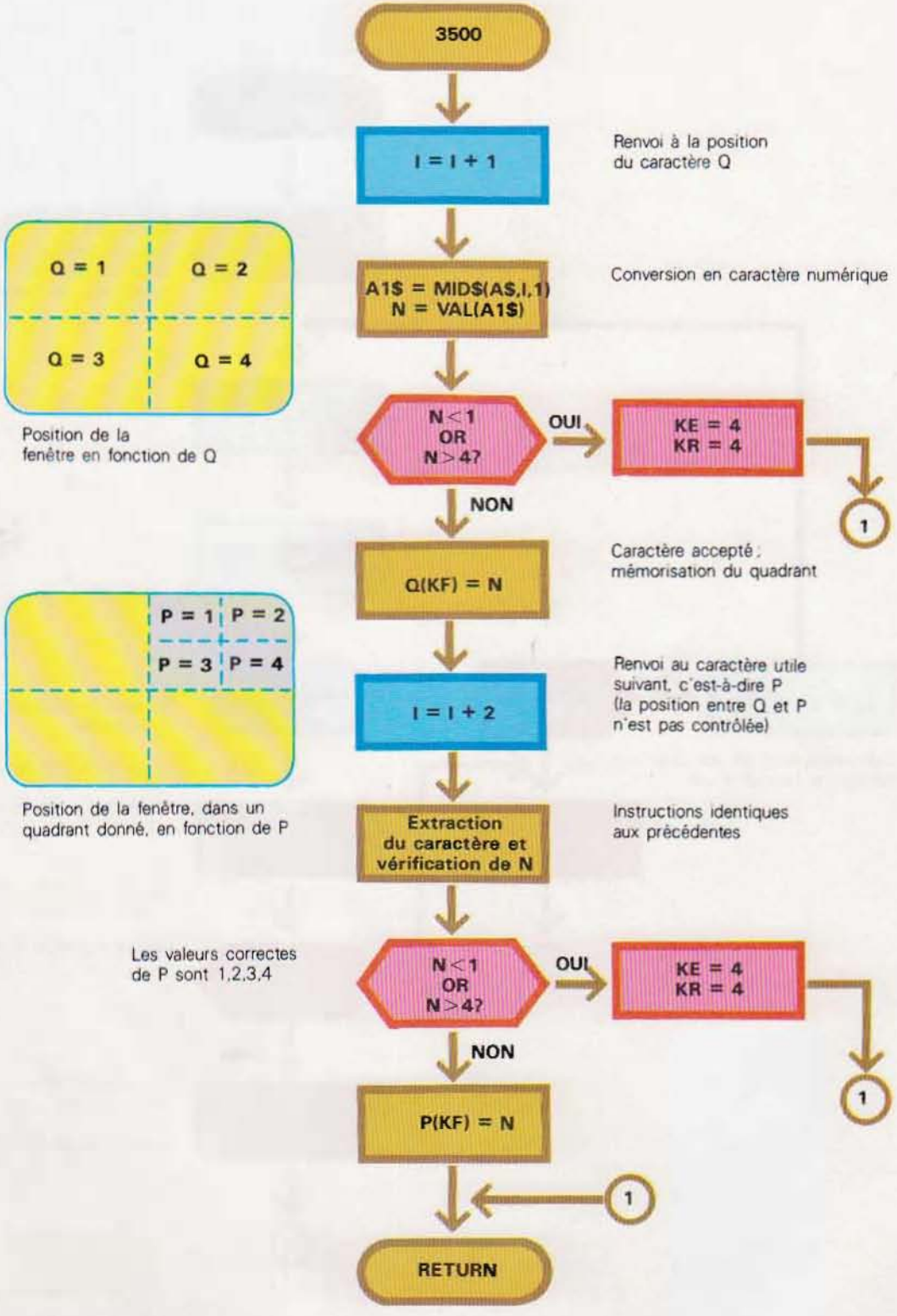
ANALYSE DU MOT-CLE



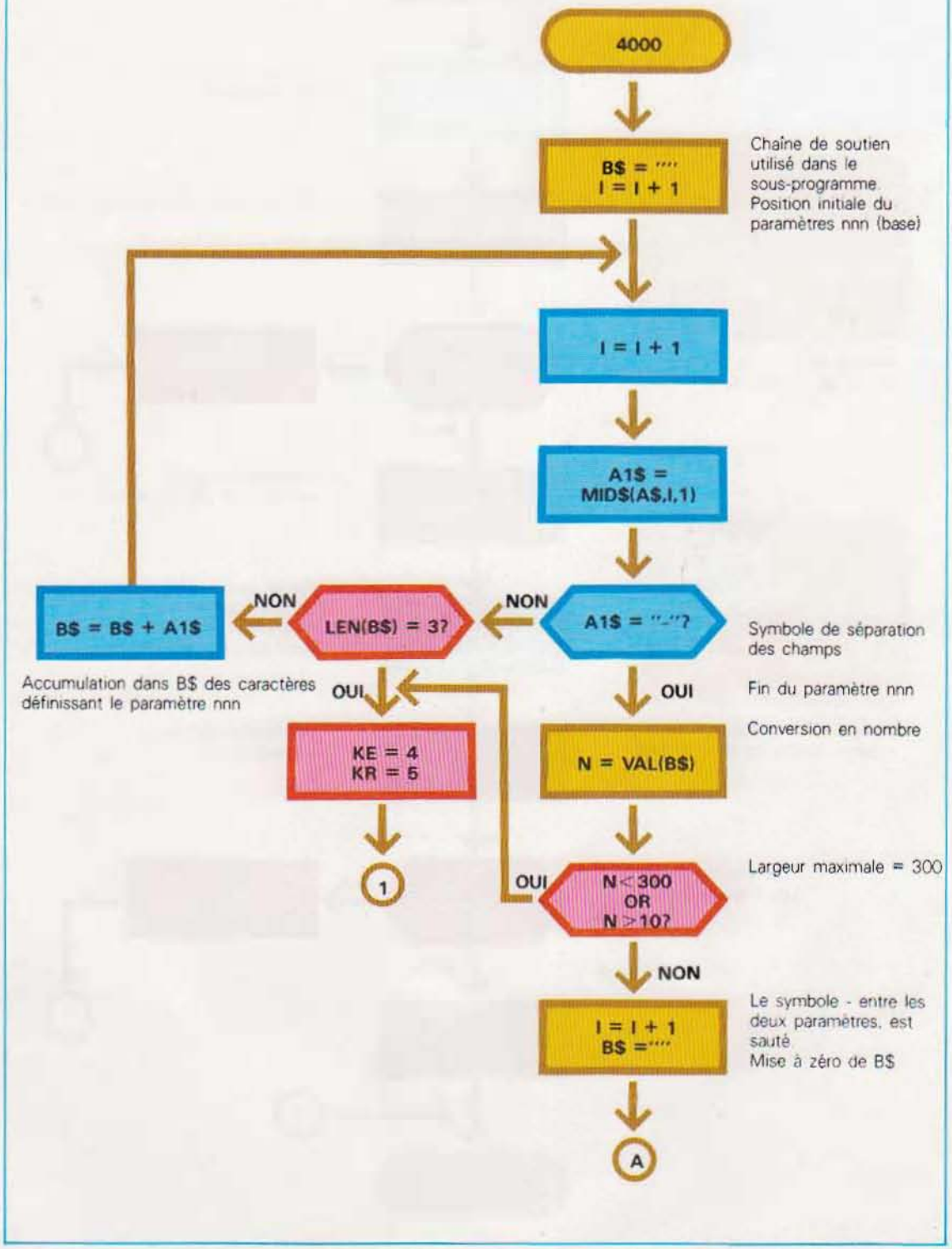
EXISTENCE DE LA PREMIERE PARENTHESE

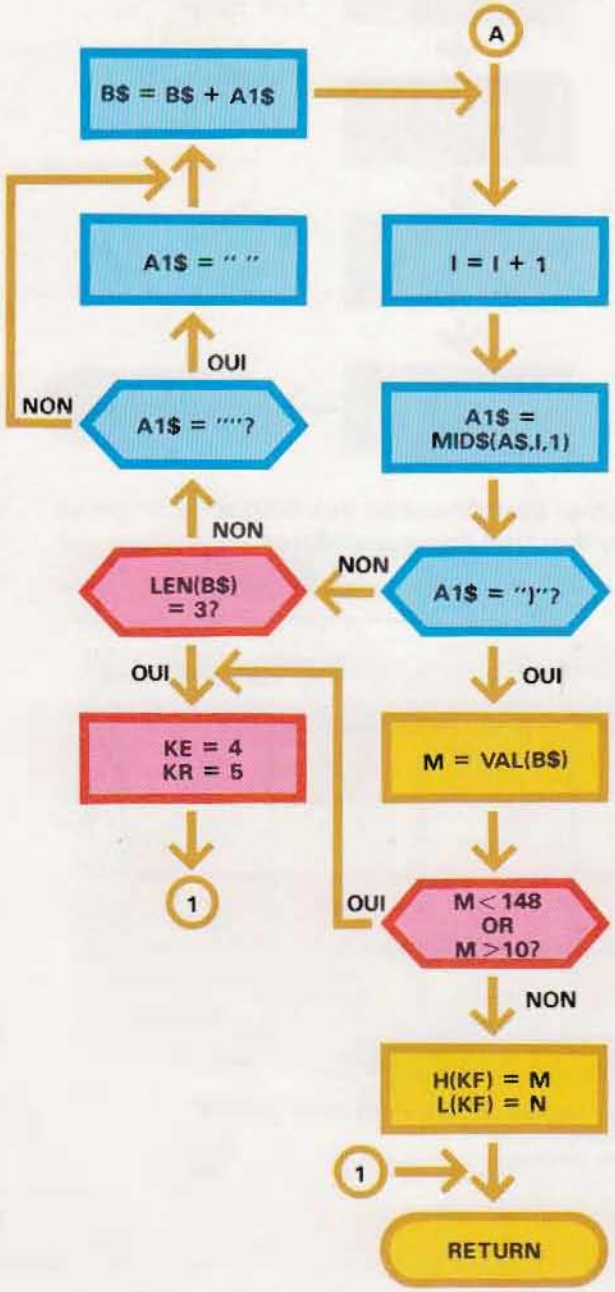


CONTROLE DES PARAMETRES Q ET P (QUADRANT ET POSITION)



CONTROLE DES DIMENSIONS (PARAMETRES nnn, mmm)

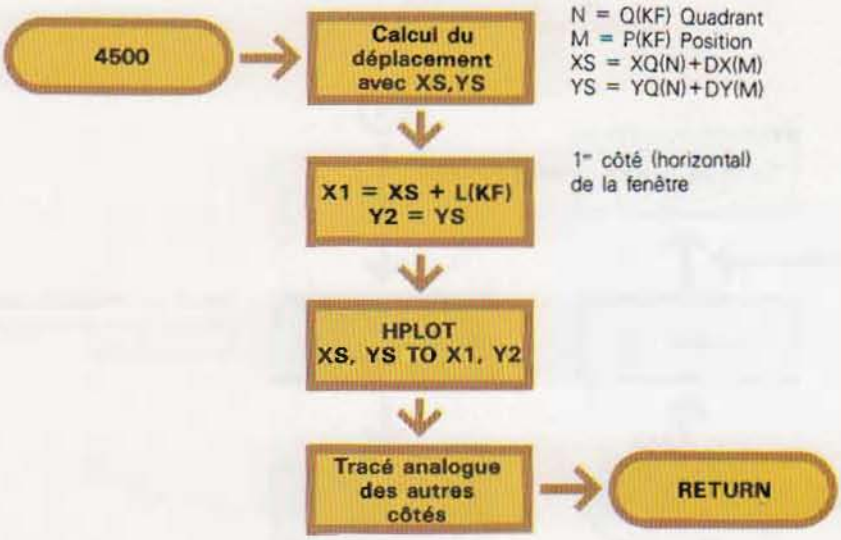




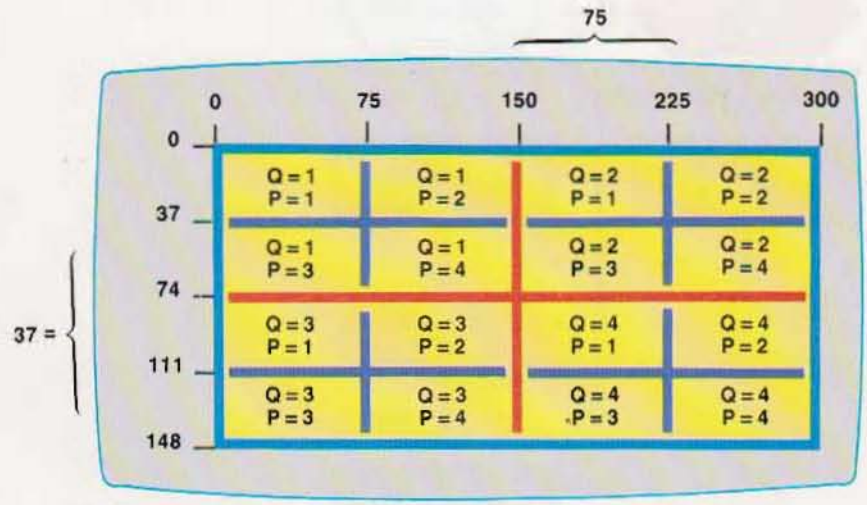
Nouvel incrément nécessaire pour la boucle d'extraction du 2e paramètre

Fin du paramètre mmm (hauteur)

SOUS-PROGRAMME DE PRESENTATION DE LA FENETRE



Division de l'écran et coordonnées des nouvelles origines en fonction des paramètres



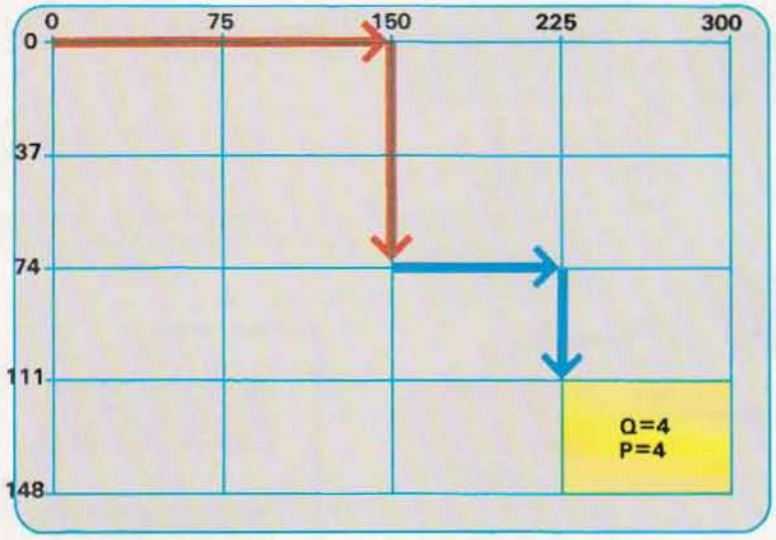
Le point de référence de chaque quadrant (subdivisions rouges) est le point extrême en haut à gauche. On obtient donc les valeurs suivantes :

Quadrant	Coordonnées du sommet
1	0,0
2	150,0
3	0,74
4	150,74

qui sont mémorisées (avec une instruction DATA) dans XQ(4) et YQ(4). De même, l'origine de la position, à l'intérieur d'un quadrant, est définie par les coordonnées du sommet en haut à gauche (mémorisée dans DX(4) et DY(4) par rapport à celle du quadrant. Les valeurs sont :

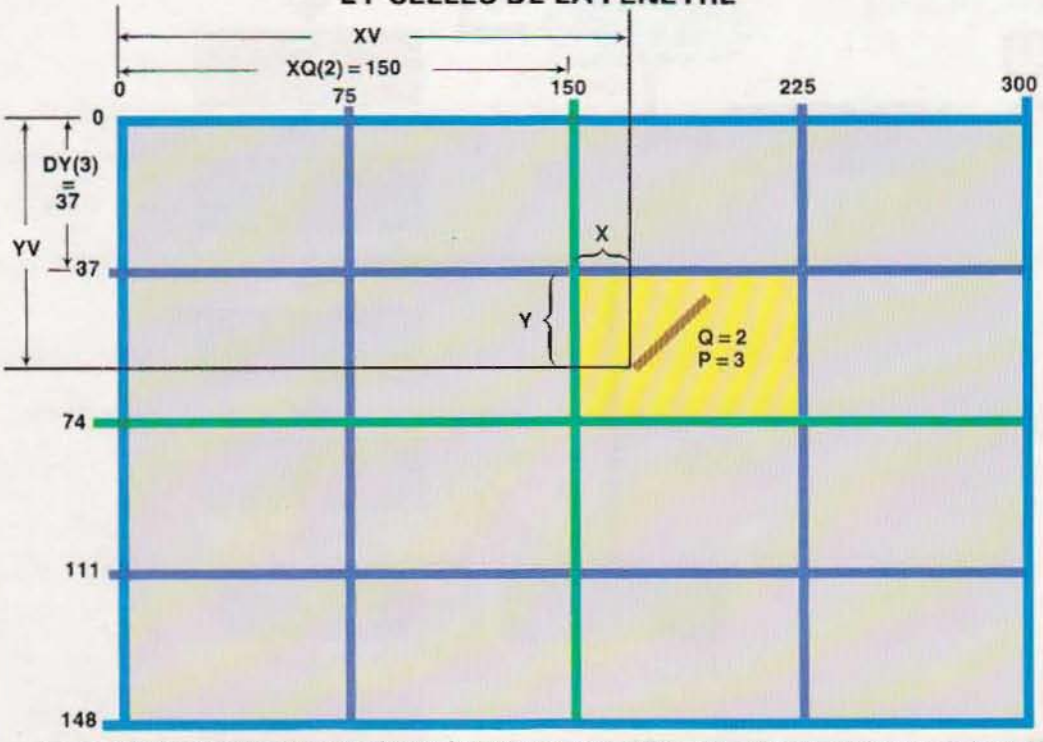
XQ(.) = 0,150,0,150
YQ(.) = 0,074,74
DX(.) = 0,75,0,75
DY(.) = 0,0,37,37

UTILISATION DES COORDONNEES DE QUADRANT ET DE POSITION



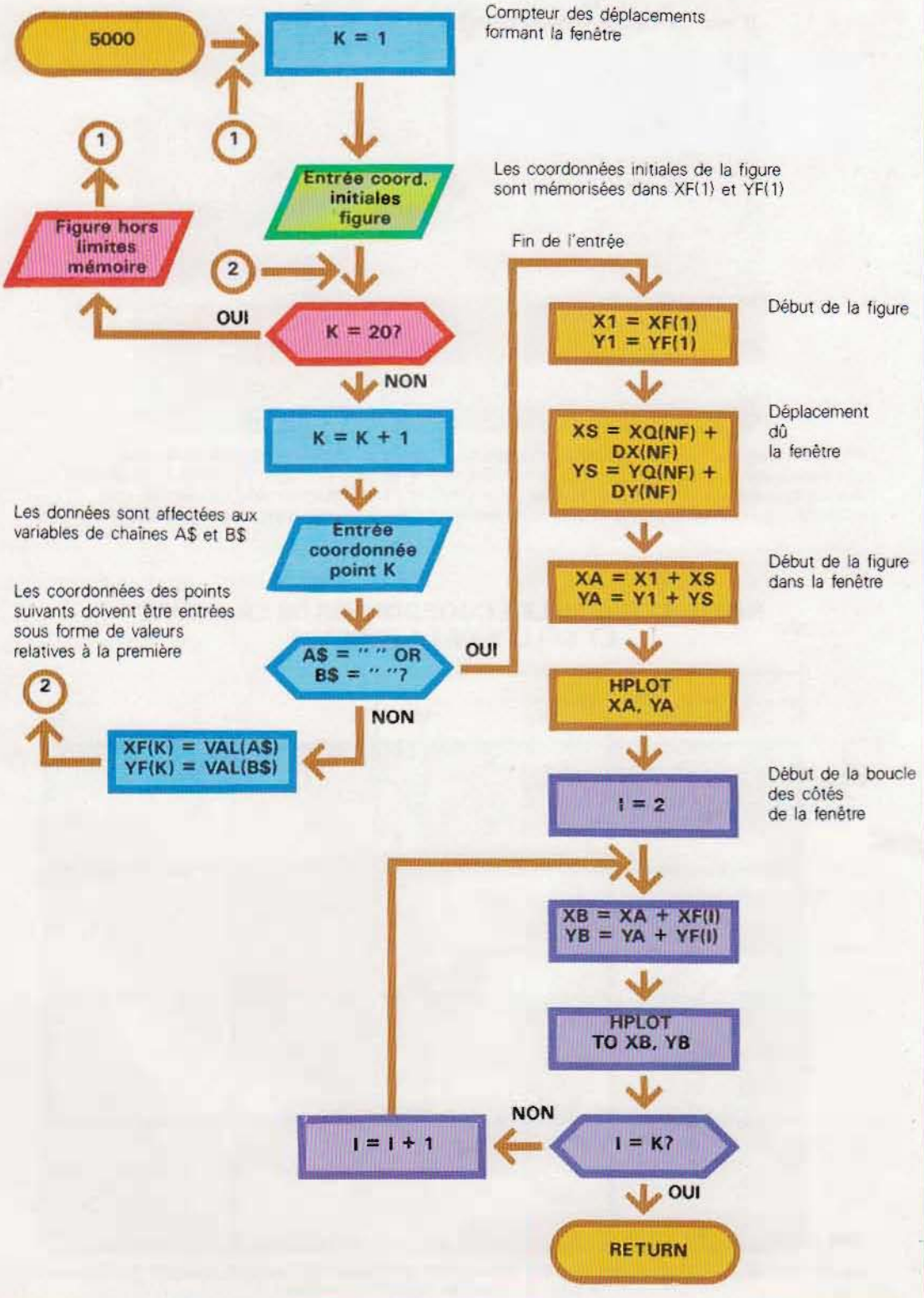
La fenêtre en position 4 dans le quadrant 4 s'obtient tout d'abord par déplacement jusqu'au quadrant, c'est-à-dire jusqu'au point de coordonnées X = XQ(4) et Y = YQ(4) (trait rouge). Il faut ensuite se déplacer, à l'intérieur du quadrant, jusqu'aux coordonnées X1 = DX(4) et Y1 = DY(4) trait bleu). On identifie ainsi l'extrémité supérieure gauche de la fenêtre. Si les dimensions de l'écran changent, il faut également changer les DATA.

RAPPORT ENTRE LES COORDONNES DE LA FIGURE ET CELLES DE LA FENETRE



Les coordonnées réelles d'un point (XV, YV) s'obtiennent en additionnant celles du quadrant et de la position à celles du point par rapport à la fenêtre

SAISIE DES COORDONNEES ET PRESENTATION DE LA FIGURE



CREATION ET GESTION DES FENETRES

```
1 REM ==================
2 REM *   PROGRAMME    *
3 REM * DE SIMULATION  *
4 REM *  DE FENETRES   *
5 REM * EN PAGE  HGR2  *
6 REM ==================
7 :

8 :

10 HOME
100 GOSUB 10000
110 GOTO 500
115 :

120 REM ==================
130 REM * APPEL FENETRE *
140 REM ==================
145 :

147 TEXT
150 HOME:PRINT "Fenetres
    disponibles"
160 VTAB 3:PRINT I$

170 FOR J=1 TO KF
180 PRINT V$(J);:HTAB 11:PRINT
    Q(J);:HTAB 23:PRINT P(J);
    :HTAB 31:PRINT H(J)"*"L(J)

200 NEXT J
210 VTAB 22:INPUT "Introduisez
    le nom de la fenetre ":NM$

220 FOR J=1 TO KF
230 IF NM$=V$(J) THEN NF=J:J=KF
    :NEXT J:GOTO 250

240 NEXT J
245 GOTO 210
250 RETURN
499 :

500 REM ========================
501 REM *APPEL SOUS-PROGRAMMES*
502 REM ========================
503 :

505 TEXT
510 GOSUB 1000
520 GOSUB 120
530 GOSUB 5000
540 TEXT:HOME:VTAB 10:INPUT
    "Voulez-vous déplacer la
    figure (O/N) ":RS$

550 IF LEFT$(RS$,1)="O" THEN 640
560 VTAB 10:INPUT "Voulez-vous
    changer le dessin (O/N) "
    :RS$
570 IF LEFT$(RS$,1)="O" THEN 520
580 VTAB10:INPUT "Voulez-vous
    modifier les fenetres (O/N) "
    :RS$
590 IF LEFT$(RS$,1)="O" THEN 500

600 HOME:VTAB 22:END

640 GOSUB 120:GOSUB 7000

645 IF Q$=" " THEN 655
650 GOSUB 5125
655 GOTO 540
1000 :

1001 REM ========================
1002 REM *DEFINITION DES FENETRES*
1003 REM ========================
1004 :

1010 KF=0:HGR2

1020 GOSUB 1500
1030 GOSUB 2000
1040 IF KE<>0 THEN 6000
1050 GOSUB 2500
1060 IF KE<>0 THEN 6000
1070 GOSUB 3000
1080 IF KE<>0 THEN 6000
1090 GOSUB 3500
1100 IF KE<>0 THEN 6000
1110 GOSUB 4000
1120 IF KE<>0 THEN 6000
1130 GOSUB 4500
1140 TEXT
1150 HOME:VTAB 10:INPUT "Autre
     fenetre (O/N) ":RS$

1160 IF LEFT$(RS$,1)="O" THEN 1020
1170 RETURN
1499 :

1500 REM ========================
1501 REM * DESCRIPTION FENETRE *
1502 REM ========================
1503 :

1505 TEXT
1510 HOME:VTAB 10:INPUT "Fenetre "
     ;A$

1520 IF LEN(A$)>30 THEN 1510
1530 RETURN
2000 :

2001 REM ==================
2002 REM * CTRL POS. 1-2 *
2003 REM ==================
2004 :

2010 KR=0:KE=0:I=1

2020 IF I>10 THEN KE=1:KR=1
     :RETURN

2030 A1$=MID$(A$,I,1)
2040 IF A1$=" " THEN I=I+1
     :GOTO 2020
2050 KF=KF+1:V$(KF)=A1$

2060 I=I+1
2070 IF I>11 THEN KE=1:KR=1
     :RETURN
```

```
2080 A1$=MID$(A$,I,1)
2090 IF A1$=" " THEN I=I+1
     :GOTO 2070
2100 IF A1$="=" THEN RETURN
2110 KE=2:KR=1:RETURN

2499 :

2500 REM ================
2501 REM * CTRL WINDOW *
2502 REM ================
2503 :
2510 KR=0:KE=0:I=I+1

2520 IF I>12 THEN KE=1:KR=2
     :RETURN
2530 A1$=MID$(A$,I,1)
2540 IF A1$=" " THEN I=I+1
     :GOTO 2520
2550 A1$=MID$(A$,I,6)
2560 IF A1$="WINDOW" THEN
     I=I+5:RETURN
2570 KE=3:KR=2:RETURN

3000 :

3001 REM =============
3002 REM * CTRL "(" *
3003 REM =============
3004 :
3010 KE=0:KR=0:I=I+1

3020 IF I>18 THEN KE=1:KR=3
     :RETURN

3030 A1$=MID$(A$,I,1)
3040 IF A1$=" " THEN I=I+1
     :GOTO 3020
3050 IF A1$="(" THEN RETURN
3060 KE=2:KR=3:RETURN

3500 :

3501 REM =============
3502 REM * CTRL Q/P *
3503 REM =============
3504 :
3510 KR=0:KE=0:I=I+1

3520 A1$=MID$(A$,I,1)
3530 N=VAL(A1$)
3540 IF N<1 OR N>4 THEN KE=4
     :KR=4:RETURN
3550 Q(KF)=N:I=I+2

3560 A1$=MID$(A$,I,1)
     :N=VAL(A1$)
3570 IF N<1 OR N>4 THEN KE=4
     :KR=4:RETURN

3580 P(KF)=N:RETURN

4000 :

4001 REM ==================
4002 REM * CTRL PARAMET. *
4003 REM ==================
4004 :

4010 KE=0:KR=0:B$="":I=I+1

4020 I=I+1:A1$=MID$(A$,I,1)

4030 IF A1$="-" THEN 4070
4040 IF LEN(B$)=3 THEN KE=4
     :KR=5:RETURN

4050 B$=B$+A1$:GOTO 4020

4070 N=VAL(B$):IF N>300 OR
     N<10 THEN KE=4:KR=5
     .RETURN

4080 B$=""
4090 I=I+1:A1$=MID$(A$,I,1)

4100 IF A1$=")" THEN 4130
4110 IF LEN(B$)=3 THEN KE=4
     :KR=5:RETURN

4115 IF A1$="" THEN A1$=" "
4120 B$=B$+A1$:GOTO 4090

4130 N=VAL(B$):IF N>140 OR
     N<10 THEN KE=4:KR=5
     :RETURN

4140 H(KF)=N:L(KF)=N:RETURN

4500 :

4501 REM ==================
4502 REM * DESSIN FENETRE *
4503 REM ==================
4504 :
4505 POKE -16304,0:POKE -16297,0
     :POKE -16299,0

4507 HCOLOR=3
4510 N=Q(KF):M=P(KF)

4520 XS=XQ(N)+DX(M):YS=YQ(N)
     +DY(M)
4530 X1=XS+L(KF):Y2=YS

4535 IF X1>279 THEN X1=279
4537 IF Y2>191 THEN Y2=191
4540 HPLOT XS,YS TO X1,Y2
4550 Y2=YS+H(KF)
4555 IF Y2>191 THEN Y2=191
4560 HPLOT TO X1,Y2
4570 X1=XS
4575 IF X1>279 THEN X1=279
4577 HPLOT TO X1,Y2
4580 Y2=YS
4585 IF Y2>191 THEN Y2=191
4587 HPLOT TO X1,Y2
4600 GET Q$:RETURN

5000 :
```

```
5001 REM ================
5002 REM * REPR. FIGURE *
5003 REM ================
5004 :

5005 TEXT
5010 K=1
5020 HOME:VTAB 10:PRINT
     "Introduire les coordonnées
     initiales"

5030 VTAB 10:HTAB 30:PRINT "X/Y"

5040 VTAB 11:HTAB 31:INPUT "":A$

5050 VTAB 11:HTAB 35:INPUT "":B$

5052 IF A$="" OR B$="" THEN 5020
5055 GOTO 5110
5060 IF K=20 THEN 5700
5070 K=K+1
5080 VTAB 10:HTAB 21:PRINT "Point
     "K

5090 VTAB 11:HTAB 31:INPUT "":A$

5100 VTAB 11:HTAB 35:INPUT "":B$

5110 IF A$="" OR B$="" THEN K=k-1
     :GOTO 5126

5120 XF(K)=VAL(A$):YF(K)=VAL(B$)
     :IF XF(K)>270 OR XF(K)<1
     THEN 5080

5121 IF YF(K)>191 OR YF<1 THEN
     5080
5122 GOTO 5060
5123 :

5124 REM >> DESSIN <<
5125 :

5126 HCOLOR=3
5127 POKE -16304,0:POKE -16297,0
     :POKE -16299,0

5130 X1=XF(1):Y1=YF(1)

5140 XS=XQ(Q(NF))+DX(P(NF)):YS
     =YQ(Q(NF))+DY(P(NF))
5145 CF=NF
5150 XA=X1+XS:YA=Y1+YS:I=2


5155 HPLOT XA,YA
5160 XB=XF(I)+XS:YB=YF(I)+YS

5170 REM
5175 IF XB>=XS+L(NF) THEN XB=XS
     +L(NF)-1
5176 IF YB>=YS+H(NF) THEN YB=YS
     +H(NF)-1
5180 HPLOT TO XB,YB
5190 IF I=K THEN GET Q$:RETURN

5200 I=I+1:GOTO 5160

5700 TEXT:HOME:VTAB 10


5710 PRINT "Max. 10 points":GET
     Q$
5720 GOTO 5010
6000 :

6001 REM ===============
6002 REM * DIAGNOSTIC *
6003 REM ===============
6004 :

6010 PRINT CHR$(7)
6020 VTAB 20:PRINT D$(KE):GET Q$

6030 VTAB 20:PRINT BK$

6040 KF=KF-1:GOTO 1020

6999 :

7000 REM ======================
7001 REM * EFFACEMENT FIGURE *
7002 REM ======================
7003 :

7010 NF=NF:NF=CF

7020 HCOLOR=0:GOSUB 5127

7030 NF=NF:RETURN

10000 :

10001 REM ===================
10002 REM * INITIALISATION *
10003 REM ===================
10004 :

10005 HIMEM 16383:HGR2:TEXT

10010 DIM D$(4),V$(16),Q(16)
      ,P(16),H(16),L(16),XF(20)
      ,YF(20)
10020 I$="Nom/Quadrant/Position
      /Dim."
10025 BL$="":FOR J=1 TO 30:BK$
      =BK$+" ":NEXT J

10030 FOR J=1 TO 4:READ D$(J)
      :NEXT J

10050 RETURN

10100 DATA 0,0,0,0,150,0,75,0,0
      ,74,0,3,7,150,74,75,37

10110 DATA "Données insuffisant
      es","Erreur de syntaxe"
      ,"Mot-clé erronné","Erreur
      de paramètres"
```

GENERATION ET GESTION DES FENETRES VIDEO

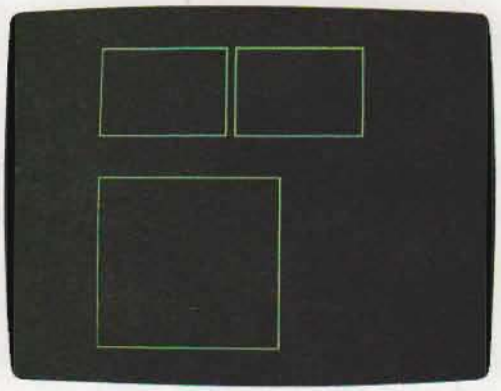
Voici quelques exemples de fonctionnement du programme présenté aux pages 1579 à 1581.
Sur l'écran ci-contre, on peut voir la phase de définition d'une fenêtre A, de 50 × 70 points écran, positionnée au point 1,1 (en haut à gauche). Le mot-clé WINDOW est suivi de ces paramètres dans un ordre inversé.



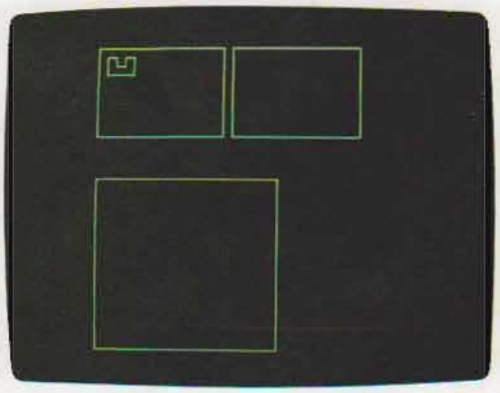
Après en avoir défini 2 autres, le programme liste les fenêtres.
L'activation d'une fenêtre se fait en introduisant le nom qui l'identifie.



A la demande de l'opérateur, 3 fenêtres définies au préalable sont affichées.



Une figure graphique a été dessinée. Elle peut être insérée dans n'importe laquelle des 3 fenêtre activées à l'aide de commandes de positionnement.



L'ordinateur et la voile

Qui ne connaît, ou tout du moins n'a entendu parler de la fameuse Coupe de l'America , ce trophée très convoité qu'a conquis, le 22 août 1851, la goélette américaine America lors d'une régate au large de l'île de Wight.

Ce fut la première victoire de la navigation à voile du Nouveau continent sur le Vieux. La coupe, qui s'appelait alors Coupe des Cent guinées, fut rebaptisées Coupe de l'America et, pendant 132 années consécutives, en dépit d'une suite de défis épiques qui ont renforcé sa renommée, elle est restée aux mains des Américains.

Avec l'édition de 1983, Coupe de l'America est devenue un symbole, bien plus important que le simple trophée. De l'avis de tous, cette année a représenté un ***tournant*** *dans l'histoire de cette compétition. Il y a d'abord eu la victoire d'Australia II sur Liberty, qui a quelque peu entamé l'invincibilité américaine.*

Il faut parler ensuite de l'intérêt de plus en plus marqué d'un plus grand public qui n'est pas forcément impliqué par la course.

Autre facteur ayant fortement contribué à ce changement de vitesse de cette régate : ***l'utilisation croissante*** *de l'****électronique*** *et de l'****informatique.***

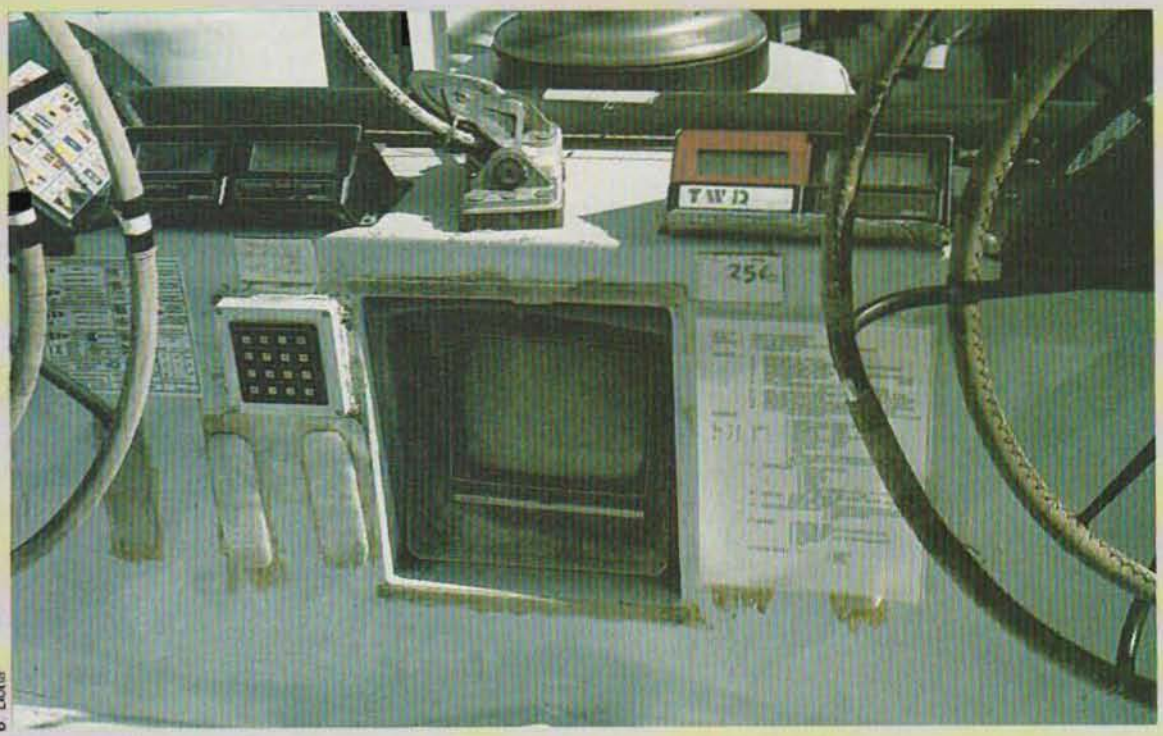
D'abord simples instruments de calcul ou outils d'aide à l'architecture et à la conception des voiliers, les ordinateurs sont « montés à bord » pour des essais en mer et en régate, afin de collecter et de traiter les données de bord. Nous assistons à un renouvellement complet de la compétition avec des voiliers de plus en plus sophistiqués.

Précisément, l'ordinateur a fait sa première apparition en 1963 sur la scène de la Coupe de l'America, avec l'embarquement d'un appareil capable de calculer l'angle réel du vent. En effet, les instruments d'alors ne fournissaient que des ***informations*** *— sens et vitesse du vent, notamment — sur le vent apparent (vent réel combiné au vent de vitesse engendré par le déplacement même de l'embarcation). Or, les données relatives au vent réel s'obtiennent en effectuant des* ***calculs vectoriels****. C'est pourquoi on est passé de l'emploi des ordinateurs comme simples instruments de calcul*

Départ de la Coupe de l'America.



B. Lotta



© Lotta

Le pont de Victory. Au premier plan, le "puits" du timonier : entre les timons, on aperçoit l'écran de l'ordinateur de bord, le clavier de commande et les viseurs à cristaux liquides permettant de visualiser les données.

pour résoudre des problèmes de navigation, à leur utilisation au cœur d'un système complexe de collecte, de traitement et de contrôle des données afin de déterminer les différents paramètres de performance.

Le développement technologique considérable réalisé dans le domaine électronique — et notamment en matière de ***miniaturisation,*** *de* ***puissance*** *de calcul et de* ***coûts*** *des ordinateurs — a été pour beaucoup dans l'accélération du processus d'intégration de l'ordinateur comme instrument indispensable à bord des 12 mètres.*

L'édition 1980 de la Coupe de l'America, à Newport, a marqué le début de l'emploi de l'ordinateur, la nouveauté n'étant plus la présence de celui-ci à bord, mais le support externe constitué par les ***puissants systèmes de calcul installés à terre et reliés par radio.*** *Pendant les entraînements et les régates, toutes les données intéressantes de l'embarcation — la vitesse et la direction du voilier, la vitesse et le sens du vent, etc. — relevées par des capteurs de bord étaient d'abord affichées sur les viseurs des instruments de bord pour une exploitation immédiate par l'équipage, ensuite recueillies et traitées par le calculateur de bord, et, enfin, intégralement enregistrées sur disque ou bande magnétique. A la fin de la journée, les données étaient transmises à terre et entrées dans l'ordinateur central, afin d'être analysées pour qu'on puisse en tirer le plus grand nombre d'informations possible sur les performances de l'embarcation.*

Toujours à l'occasion de l'édition 1980, une autre nouveauté est apparue : l'introduction de la ***télématique****, qui permet de transmettre directement les données du voilier au centre de traitement à terre. L'Indépendance Syndicate (un « syndicate » est l'équivalent d'un consortium) du New York Yacht Club, propriétaire du Defender Clipper (defender, signifiant défenseur, est le nom que l'on donne — ou plutôt que l'on donnait — aux 12 mètres américains, qui devaient défendre la coupe détenue), avait adopté cette nouvelle méthode afin de rendre le traitement des données plus* ***rapide*** *et plus* ***efficace.***

Il n'y avait donc plus à attendre le retour à la base de l'embarcation pour disposer des données recueillies, celles-ci étant ***directement traitées à bord.***

Les données sont transmises en permanence par radio, de l'ordinateur de bord à celui du centre terrestre. Leur traitement est, quasiment, instantané. Au retour de l'embarcation, les données, ainsi analysées par le calculateur, sont immédiatement disponibles.

Cette innovation s'est avérée une aide précieuse, notamment lors de la mise en service des nouveaux bateaux. En effet, l'équipage doit apprendre à connaître les performances du voilier en mer : l'**ordinateur** constitue un auxiliaire de grande valeur et un moyen d'**accélérer** au maximum la **phase d'apprentissage**.

L'épreuve de 1983 a confirmé la tendance au recours de plus en plus fréquent à l'ordinateur de bord et des systèmes de traitement de soutien à terre, non seulement pour les defenders américains, qui sont, historiquement et techniquement, les mieux lotis en la matière, mais aussi pour presque tous les 12 mètres engagés. Il faut d'ailleurs dire que l'un des systèmes les plus sophistiqués jamais utilisés au cours de cette course était installé à bord de l'embarcation anglaise Victory 83.

A travers cet exemple, nous examinerons en détail l'emploi d'un ordinateur à bord d'un 12 mètres. Ce système se décompose en plusieurs éléments :

■ ***Le système de relevé, de préparation et de visualisation des données*** (transducteurs et capteurs) qui mesure les données liées au pilotage de l'embarcation (par exemple, vitesse et direction du vent, vitesse et route de l'embarcation, angle du timon, position, etc.).

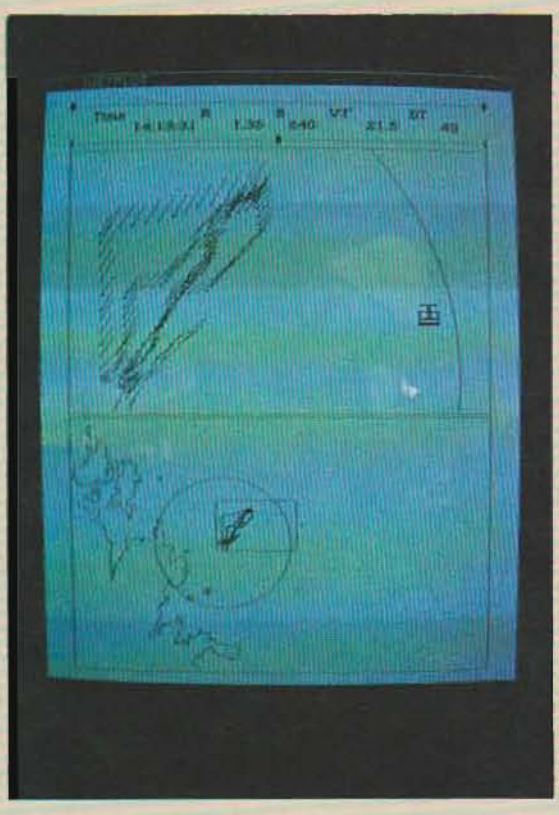
Il faut y ajouter des unités d'interface ainsi qu'un microprocesseur dont la tâche est de préparer les données brutes et d'effectuer quelques calculs simples. Les résultats sont affichés sur des viseurs à cristaux liquides. Ils sont donc immédiatement exploitables par l'équipage, mais, de toute façon, transmis à l'unité centrale pour traitement.

■ ***Le système d'acquisition et de traitement des données*** comporte un ordinateur de bord qui effectue les traitements parti-

Le système d'acquisition, de traitement et de mémorisation des données à bord de l'embarcation auxiliaire de Victory.



B. Lotz



B. Liotta

Présentation, à l'écran, de la zone de régate et reconstitution de la route suivie par Victory. En haut de l'écran, la zone est agrandie.

culiers, comme la corrélation des données, la production de nouveaux résultats à partir de données transmises par les capteurs, la visualisation et la comparaison des données et des résultats avec ceux qui ont été mémorisés au cours des essais ou des précédentes régates.

Jusque-là, pour des raisons d'ordre plus historique que technique, la puissance du système de bord était limitée. On a préféré confier le travail d'analyse statistique, de production de fonctions, de mémorisation... à un autre système, plus important, installé à terre ou, comme dans le cas du Victory, à bord d'une vedette chargée d'assister le 12 mètres jusqu'aux minutes précédant le début de la régate. Les données ne sont donc pas confinées à bord de l'embarcation engagée, mais sont ***transmises par radio*** *au système principal. Ce type de transmission est désormais admis dans le monde des 12 mètres.*

Outre l'ordinateur, le noyau du système intègre une unité de disque, un écran vidéo, une imprimante graphique et un clavier. Le système possède une banque de données à la disposition de l'embarcation ; celle-ci constitue une aide réelle pour l'équipage qui peut, grâce aux instruments embarqués, contrôler efficacement les performances du voilier.

Les données prélevées par les capteurs de bord sont enregistrées ***en continu*** *et, le cas échéant, traitées* ***en temps réel.*** *L'opérateur se trouve ainsi en mesure de « piloter » les événements, par exemple de signaler par radio à l'équipage les éventuelles anomalies... et d'enregistrer les données s'il le juge utile. Ceci lors des essais seulement, car il n'est pas permis de communiquer avec les 12 mètres pendant la régate.*

La prochaine édition de la Coupe de l'America se déroulera à Perth, en Australie, en 1987. On peut déjà assurer, sans risque d'erreurs, que d'autres systèmes de bord se trouvent d'ores et déjà à l'étude.

Prenons le cas d'un pays européen. L'Italie, par exemple, a déjà lancé trois défis pour l'année 1987.

Le premier l'a été par le Yacht Club Costa Smeralda, soutenu par le Consorzio Azzurra, Sfida Italian America's Cup. Le second est celui du Yacht Club Italiano, avec le Consorzio Italia, qui a acheté Victory 83 pour s'en servir comme « lièvre » du nouveau 12 mètres qui sera conçu et construit par le consortium (l'embarcation-lièvre est utilisée pour l'entraînement d'une autre). Le règlement de la Coupe prévoit, en effet, que l'embarcation et son équipement doivent obligatoirement être conçus et réalisés par la nation engagée.

Le dernier défi est celui du Club Nautico Marina di Carrara, parrainé par le Consorzio Futura.

De nouvelles idées et de nouveaux projets entourés du plus grand secret seront probablement mis à jour. Il est néanmoins certain que la ***capacité des systèmes de bord*** *installés sur les 12 mètres sera* ***augmentée,*** *et que les études météorologiques de la zone de régate (analyse statistique des vents et d'autres données, à l'aide de satellites météorologiques du type Météostat) seront améliorées.*

Alors vivent le sport et la technologie ! Conjointement, car tous deux retirent d'appréciables avantages réciproques de leur étroite coopération.

Bruno Liotta

Instructions Basic de gestion des fenêtres vidéo

Le Basic standard, implémenté sur certaines machines, contient les instructions nécessaires à la gestion des fenêtres vidéo. On procède en deux étapes :

— génération de la fenêtre, en en définissant les dimensions et la position ;
— adressage de graphiques ou de texte par appel du nom symbolique associé à la fenêtre.

Une fenêtre est décrite à l'aide d'instructions proches de l'instruction WINDOW de l'exemple précédent. Toutefois, tous les sous-programmes nécessaires au décodage de l'instruction et à son exécution font partie intégrante du langage et sont totalement transparents à l'utilisateur.
Dans notre exemple, les sous-programmes de décodage sont écrits en Basic et génèrent des appels à d'autres sous-programmes, également en Basic. Ces derniers sont traduits par l'interpréteur avant d'être exécutés.
Quand les instructions nécessaires font partie du langage, le décodage fournit directement la version finale.
En revanche, si c'est l'utilisateur qui doit les implémenter, le temps d'exécution sera plus long (sauf si le programme est compilé).
Nous allons maintenant examiner la syntaxe des instructions de gestion des fenêtres dans l'Olivetti M20.
Dans cette machine, l'écran vidéo offre deux modes de visualisation : 512 × 256 pixels ou 480 × 256 pixels à sélectionner au moment de la configuration du système. Chacun des points de l'écran (pixel) est identifié, à l'aide du système de coordonnées de base dont l'origine se trouve en bas à gauche. Ce système de référence peut également être défini par l'utilisateur.
Dans ce dernier cas, il peut arriver que les coordonnées d'un point ne correspondent à aucun pixel (la position de chacun des pixels est définie au niveau du matériel). Le système active alors le pixel le plus proche des coordonnées indiquées.
Ce mode de fonctionnement n'oblige pas à recourir aux références des points écran, mais peut entraîner des imprécisions parfois importantes dans la présentation des images graphiques.
L'instruction qui définit une fenêtre à la forme :

A=WINDOW (q,p,v,h)

comprenant les éléments suivants :

A : C'est la variable (entière) associée à la fenêtre. Le système lui affecte une valeur croissante de 2 à 16, 16 étant le plus grand nombre de fenêtres susceptibles d'être ouvertes simultanément, tandis que 1 désigne l'intégralité de l'écran. La numérotation croissante commence à la première valeur disponible et progresse au fur et à mesure de la création de fenêtres mais en réutilisant les numéros laissés libres par d'éventuelles fermetures. Par exemple, si les fenêtres 2, 3, et 4 sont créées, la suivante sera la 5, à moins que l'une des précédentes ne soit effacée (fermée). La prochaine fenêtre prendra alors la valeur dégagée et non pas 5.

q : Cet élément indique la position occupée par la fenêtre en cours de création à l'intérieur de la fenêtre « mère ». Cette fonction se distingue de celle de l'exemple précédent, puisque la fenêtre ne pouvait être positionnée que par rapport à l'écran tout entier.
A l'inverse, avec cette machine, une fenêtre est considérée comme un écran (l'écran tout entier est la fenêtre 1). Les valeurs admises pour le paramètre sont les suivantes :

0 Positionnement de la nouvelle fenêtre dans la partie supérieure de la fenêtre mère
1 Positionnement dans la partie inférieure
2 Positionnement à gauche
3 Positionnement à droite

p : Il sert à noter la position de subdivision de la fenêtre mère (ou de l'écran entier) pour créer les nouvelles fenêtres.
La valeur à affecter au paramètre p dépend de celle de q. Si q vaut 0 ou 1 (fenêtre dans la partie supérieure ou inférieure de la fenêtre mère), la valeur de p variera de 1 à 255 pour créer une fenêtre ; elle indique donc le nombre de lignes de balayage de la

nouvelle fenêtre*. Un exemple de subdivision de l'écran est montré ci-dessous. Si le paramètre q (quadrant) vaut 2 ou 3, la division est verticale. Le paramètre p (position) peut alors varier entre 1 et la longueur de la fenêtre mère est décrémentée de 1.

v : C'est le paramètre optionnel, indiquant le nombre de lignes de balayage par ligne de texte. Il varie de 1 à 16. La valeur par défaut est celle de la fenêtre mère.

h : Il sert à indiquer la distance, en pixels, entre deux caractères de texte. Il peut prendre les valeurs 6 ou 8. Dans le premier cas, chaque ligne comporte un maximum de 80 caractères, contre 64 dans le second. Ce paramètre est optionnel. Sa valeur par défaut est celle de la fenêtre mère.

* L'expression « ligne de balayage » désigne une ligne de l'écran graphique. Elle ne doit pas être confondue avec les termes « rangée » ou « ligne », qui se réfèrent au mode texte. Dans ce cas, une ligne est séparée des autres de la hauteur d'un caractère au moins, et elle se compose de plusieurs lignes de balayage, généralement de 7 à 12.

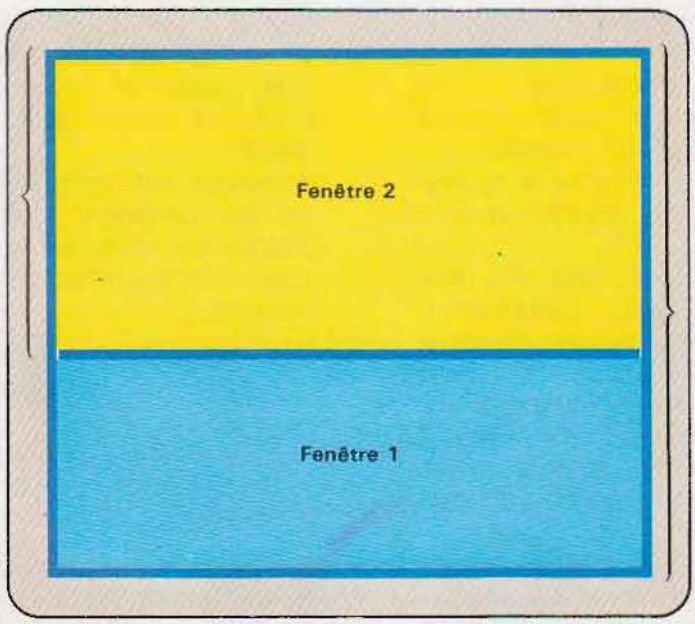
La page ci-contre montre quelques exemples de l'instruction WINDOW, en fonction des variations de v et de h. Les autres paramètres sont omis, ce qui signifie que le système prend en compte les valeurs de la fenêtre mère. Les paramètres v et h peuvent être modifiés à tout moment (même une fois la fenêtre ouverte) à l'aide de la même instruction WINDOW. Elle prend alors la forme :

A=WINDOW(0,0,v,h)

Dans ce cas, la variable A est fictive, car le système lui affecte automatiquement la valeur correspondant à la fenêtre sélectionnée.
La sélection d'une fenêtre reprend une fois encore l'instruction WINDOW, sous la forme :

WINDOW %N

EXEMPLE DE PARTAGE DE L'ECRAN EN FONCTION DES PARAMETRES q et p

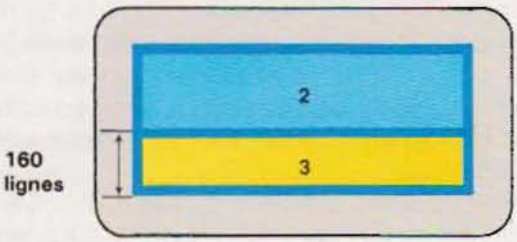


Quand une fenêtre (par ex. la n° 2) est ouverte selon les valeurs :

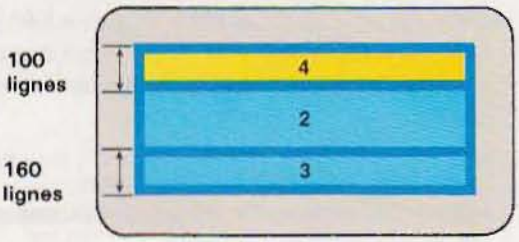
q = 0 la fenêtre est positionnée dans la partie haute de la fenêtre mère (ici l'écran dans son ensemble) et la division est horizontale

p= 180 la hauteur de la fenêtre, à partir du bord supérieur, est de 180 lignes

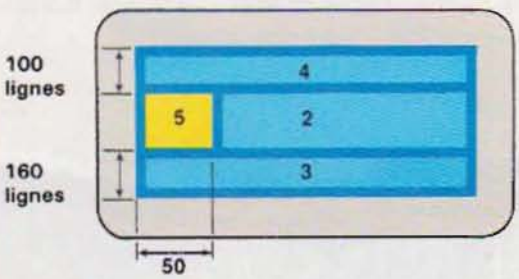
EXEMPLES D'APPLICATION DE L'INSTRUCTION WINDOW



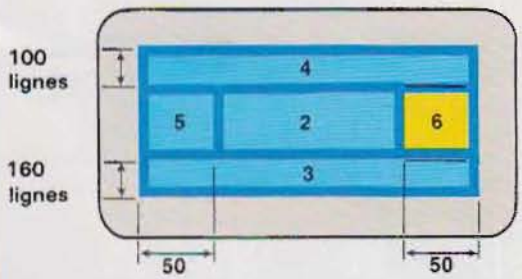
WINDOW (1,160). Séparation de la partie basse (q = 1), d'une hauteur de 160 lignes de balayage



WINDOW (0.100). Séparation de la partie haute, d'une hauteur de 100 lignes



WINDOW (2,50). Division dans la partie gauche. Dans ce cas, la valeur 50 du paramètre p indique la largeur de la fenêtre



WINDOW (3,50). Division analogue opérée dans la partie droite (q = 3) de la fenêtre mère

La fenêtre choisie correspond à la valeur numérique de la variable entière N.
Par exemple, les instructions :

```
N=2
WINDOW %N
```

provoquent la sélection de la fenêtre n° 2.

La valeur numérique peut également être fournie directement ; les instructions sont alors regroupées dans WINDOW %2.
WINDOW ferme une fenêtre devenue inutile, et rend cet espace disponible pour d'autres applications.

Sa forme devient alors :

CLOSE WINDOW %A

A indiquant la fenêtre à effacer. Si A est omis, toutes les fenêtres présentes à l'écran sont fermées.

Vecteurs graphiques et tables des figures

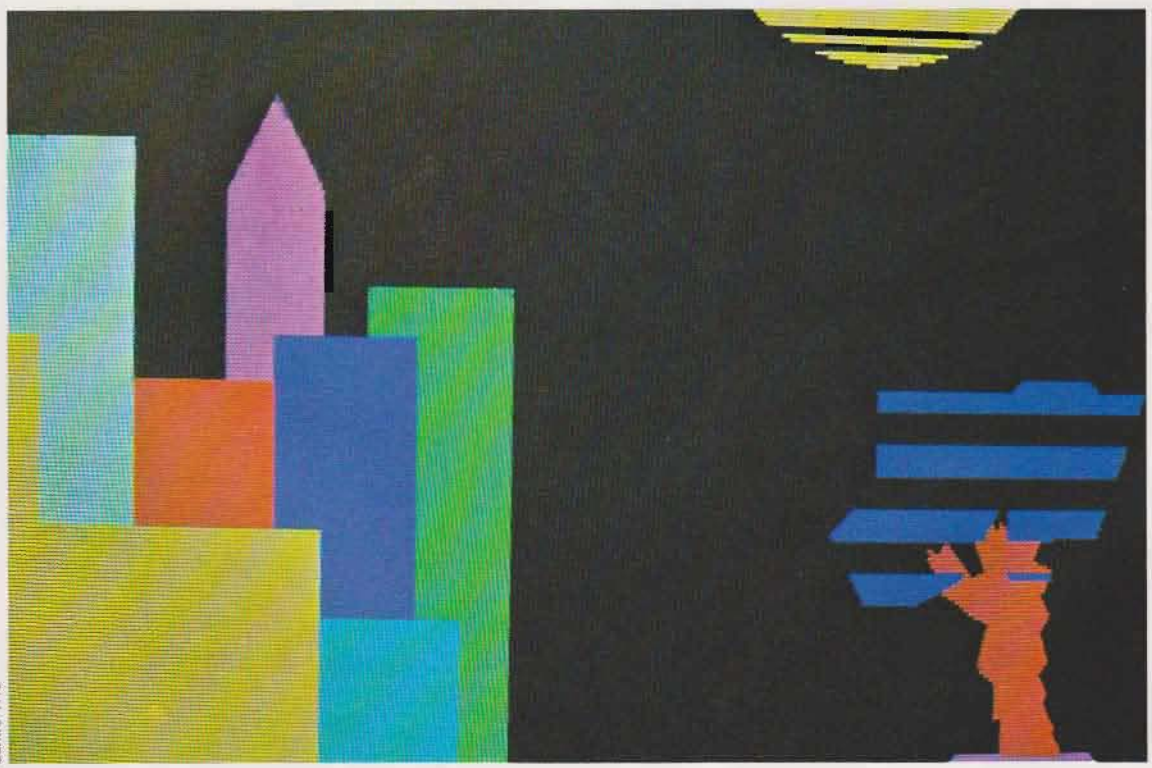
Tout dessin se décompose en une série de traits linéaires élémentaires. Cette technique permet, avec certains systèmes, de définir un ensemble de déplacements élémentaires du stylet constituant la figure. Ces ensembles sont généralement désignés par **tables des figures** à l'image des tables des déplacements servant à générer des chiffres ou des lettres.
Dans les systèmes qui y font appel, les tables sont gérées par le logiciel de base. Celui-ci contient des instructions de changement d'échelle ou de rotation.
Il est toutefois possible de créer un logiciel très simple, assurant les mêmes fonctions, avec des machines pour lesquelles ces instructions n'ont pas été initialement prévues.

Codage des vecteurs de déplacement

Pour produire une table des figures, il suffit de mémoriser, à l'échelle appropriée, tous les déplacements nécessaires au tracé du dessin. Il y a quatre types de déplacements (haut, bas, droite, gauche), les lignes inclinées ou courbes étant représentées à l'aide de lignes brisées dégradées.
Une figure de la table n'est pas mémorisée à ses dimensions réelles, mais selon un système qui assimile un déplacement à la longueur du côté le plus court.
Ainsi, pour mémoriser un rectangle dont la hauteur est double de sa base, il faudra utiliser la table suivante (dans notre machine, l'origine des axes se trouve en haut à gauche de l'écran) :

1 / Un déplacement horizontal (base)
2 / Deux déplacements verticaux (hauteur = 2 × base)
3 / Un déplacement horizontal (dans le sens opposé au précédent)
4 / Deux déplacements verticaux (dans le sens contraire des précédents)

Une ville "interprétée" par l'ordinateur en mode graphique.



Centro THC

Si les symboles suivants :

↑ = déplacement unitaire vertical négatif (haut)

↓ = déplacement vertical positif (bas)

→ = déplacement horizontal positif (droite)

← = déplacement horizontal négatif (gauche)

sont utilisés pour indiquer les déplacements, ce rectangle sera décrit par la suite de symboles :

→ ↑↑ ← ↓↓

Chaque déplacement est appelé **vecteur graphique ;** le rectangle précédent sera donc constitué de 6 vecteurs. Lors de la reconstitution du dessin, chacun d'eux indique la direction du déplacement à effectuer pour tracer un côté, tandis que le pas de déplacement est donné par le facteur d'échelle. Les changements d'échelle sont donc très simples à effectuer : il suffit de suivre le même tracé, mais en multipliant les quantités en fonction de la nouvelle échelle.

Si chaque figure est schématisée comme une combinaison des quatre déplacements élémentaires, le gain d'espace obtenu à la mémorisation est considérable. Les choix possibles à chaque pas sont effectivement limités à quatre (↑ , →, ← , ↓ ,) et sont représentés à l'aide de deux bits, par exemple comme il est indiqué dans le tableau ci-dessous :

Déplacement	Valeur décimale	Valeur binaire
↑	0	0 0
→	1	0 1
↓	2	1 0
←	3	1 1

Le codage indiqué est le plus courant dans les machines reprenant cette technique (Apple et compatibles). Dans tous les cas, les vecteurs peuvent toujours être représentés avec deux bits.

Le tableau 1 montre un codage possible des vecteurs graphiques mais n'indique pas leurs éventuels attributs. Il faut notamment que soit indiqué si le déplacement spécifié par le vecteur est seulement un positionnement (sans que rien n'apparaisse sur l'écran) ou la visualisation d'un segment. Dans le premier cas, le déplacement décrit par le vecteur doit être tracé à l'aide d'une couleur identique à celle du fond ; dans le second cas (visualisation), la couleur doit être visible. Cette information nécessite le recours à un 3e bit indiquant si le segment est visible ou non. Si on lui associe la valeur O pour indiquer qu'il s'agit d'un simple déplacement et la valeur 1 pour signifier que le trait est visible, on obtient les correspondances présentées dans le tableau 2 en bas de page. Comme on peut le voir, pour passer d'un déplacement non visible au même déplacement visible, il suffit d'ajouter 4. Par exemple, le déplacement horizontal positif (→) a le code décimal 1 s'il est invisible et 5 dans le cas contraire. D'après ce décodage, le rectangle considéré précédemment est représenté par les valeurs décimales 5 (→), 4 (↑), 4 (↑), 7 (←), 6 (↓), 6 (↓) et par les valeurs binaires correspondantes :

101 (5, →)
100 (4, ↑)
100 (4, ↑)
111 (7, ←)
110 (6, ↓)
110 (6, ↓)

La mémorisation de cette table nécessiterait

CODAGE DES VECTEURS DE DEPLACEMENT

Déplacement (vecteur)	Visible			Non visible			Valeur décimale visible	Valeur décimale non visible
	2	1	0	2	1	0		
↑	1	0	0	0	0	0	4	0
→	1	0	1	0	0	1	5	1
↓	1	1	0	0	1	0	6	2
←	1	1	1	0	1	1	7	3

6 positions de mémoire, soit une par déplacement. L'espace mémoire serait inutilement occupé car chaque position se compose de 8 bits, alors qu'un vecteur graphique n'en utilise que 3. Il est donc normal de rechercher une forme de compactage permettant de mémoriser plusieurs vecteurs à la même adresse.

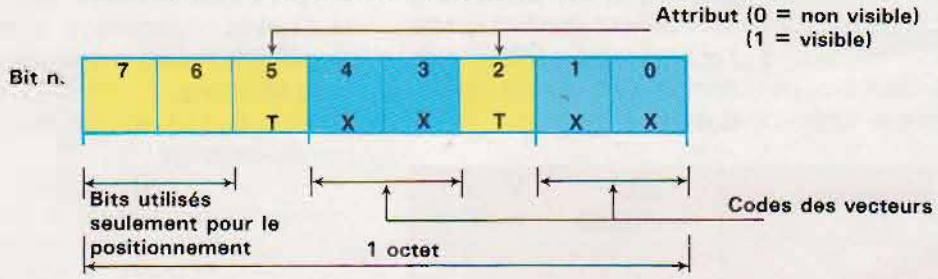
La plus usitée (figure 1 de cette page) consiste à occuper, avec deux vecteurs successifs, 6 des 8 bits d'une adresse mémoire ; les bits 3, 4 et 5 pour le premier et 0, 1 et 2 pour le second. Les bits 6 et 7 de l'adresse restent donc inactifs. Ils pourront, dans certains cas, servir à désigner un vecteur de déplacement sans affichage (il y manque le bit indiquant l'attribut). Leur emploi ne peut toutefois pas être généralisé.

La figure 2 de cette page illustre une autre forme de mémorisation d'un rectangle, exploitant le système de compactage précédemment décrit. Les bits 6 et 7 sont toujours à 0 tandis que le 5 et le 2 sont toujours à 1 (ils indiquent un déplacement visible). Pour signaler à la machine que la table est finie, une adresse tout entière est mise à 0 (position 4). Une fois la table de la figure mémorisée, il suffit, pour exécuter le tracé, d'appeler un sous-programme spécifique. Celui-ci analyse le contenu de chaque adresse et dessine un segment dans la direction indiquée par les bits 3, 4 et 0, et dans la couleur représentée par les bits 5 et 2.

La boucle est interrompue quand le contenu de la mémoire est à zéro (valeur choisie, conventionnellement, pour indiquer la fin de la table). On trouvera pages 1593 et 1595 les organigrammes d'un programme d'illustration et le listing correspondant (page 1596). Dans cet exemple, la table de la figure (un rectangle), introduite à l'aide d'une instruction DATA, est fixe, tandis que la phase de présentation (sous-programme 1000) peut être utilisée pour des emplois généraux.

Le programme n'est valable que du point de vue qualitatif. Il y manque la possibilité d'inclure plusieurs figures dans la même table, ensuite les sous-programmes de générations et de mémorisation des figures.

COMPACTAGE DES CODES CORRESPONDANT AUX VECTEURS GRAPHIQUES



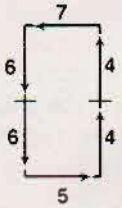
MEMORISATION D'UN RECTANGLE

Position mémoire	7	6	32 5 T	16 4	8 3	4 2 T	2 1	1 0	Valeur décimale
1	0	0	1	0	1	1	0	0	44
2	0	0	1	0	0	1	1	1	39
3	0	0	1	0	1	1	0	1	45
4	0	0	0	0	0	0	0	0	0

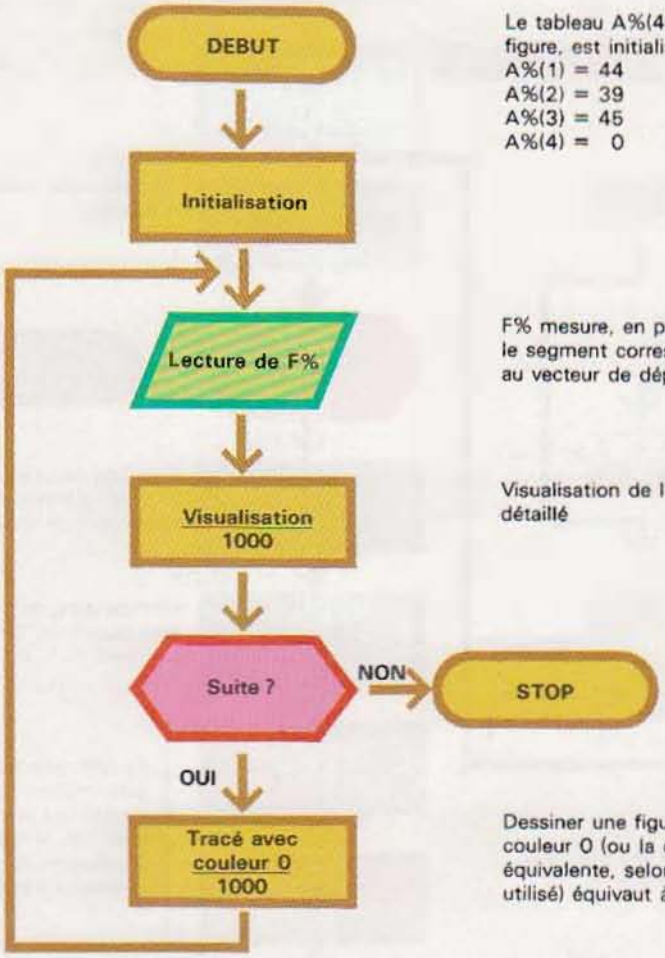
5(→) + 4(↑)

4(↑) + 7(←)

6(↓) + 6(↓)



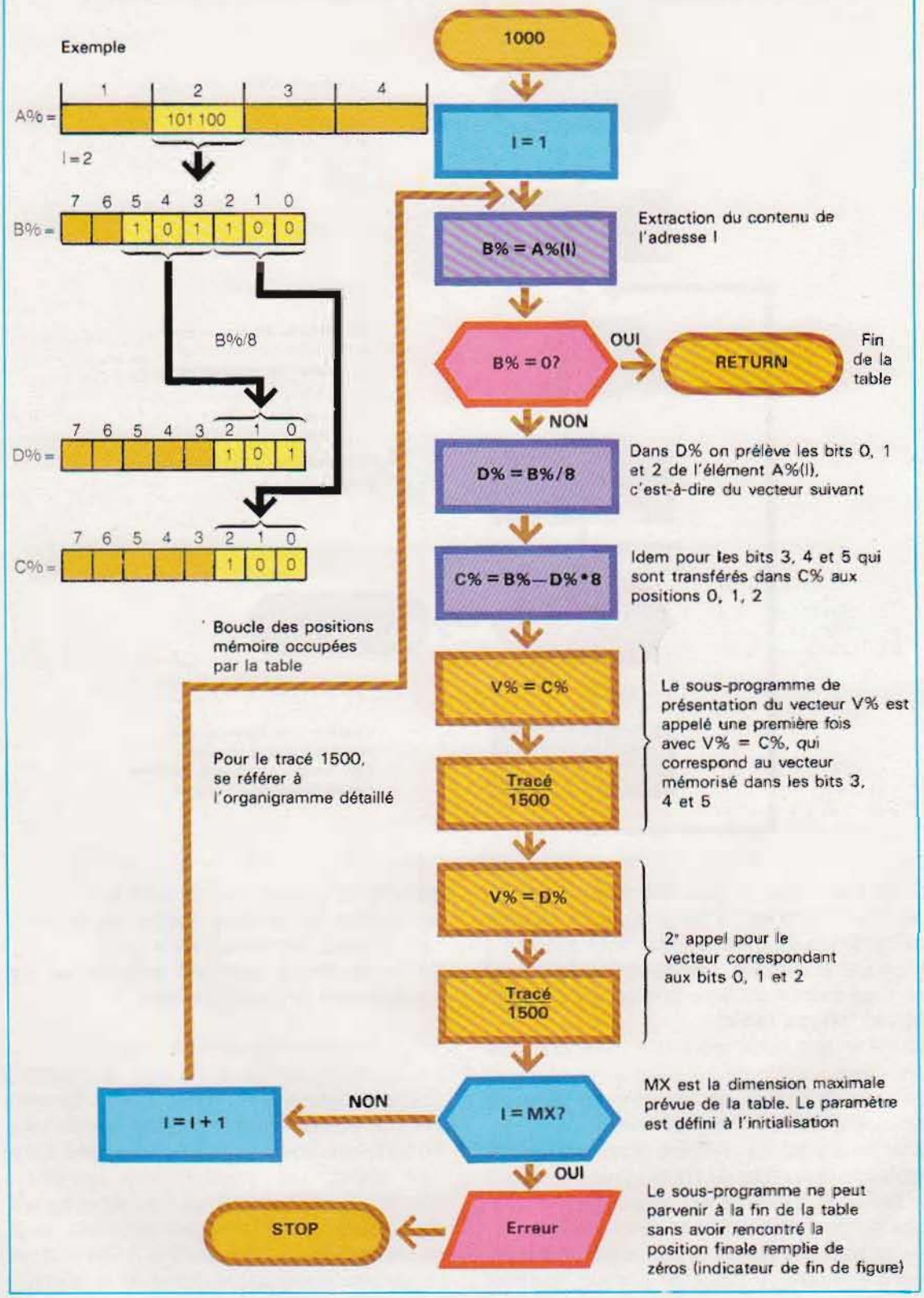
ORGANIGRAMME DE PRESENTATION D'UNE TABLE DE FIGURES



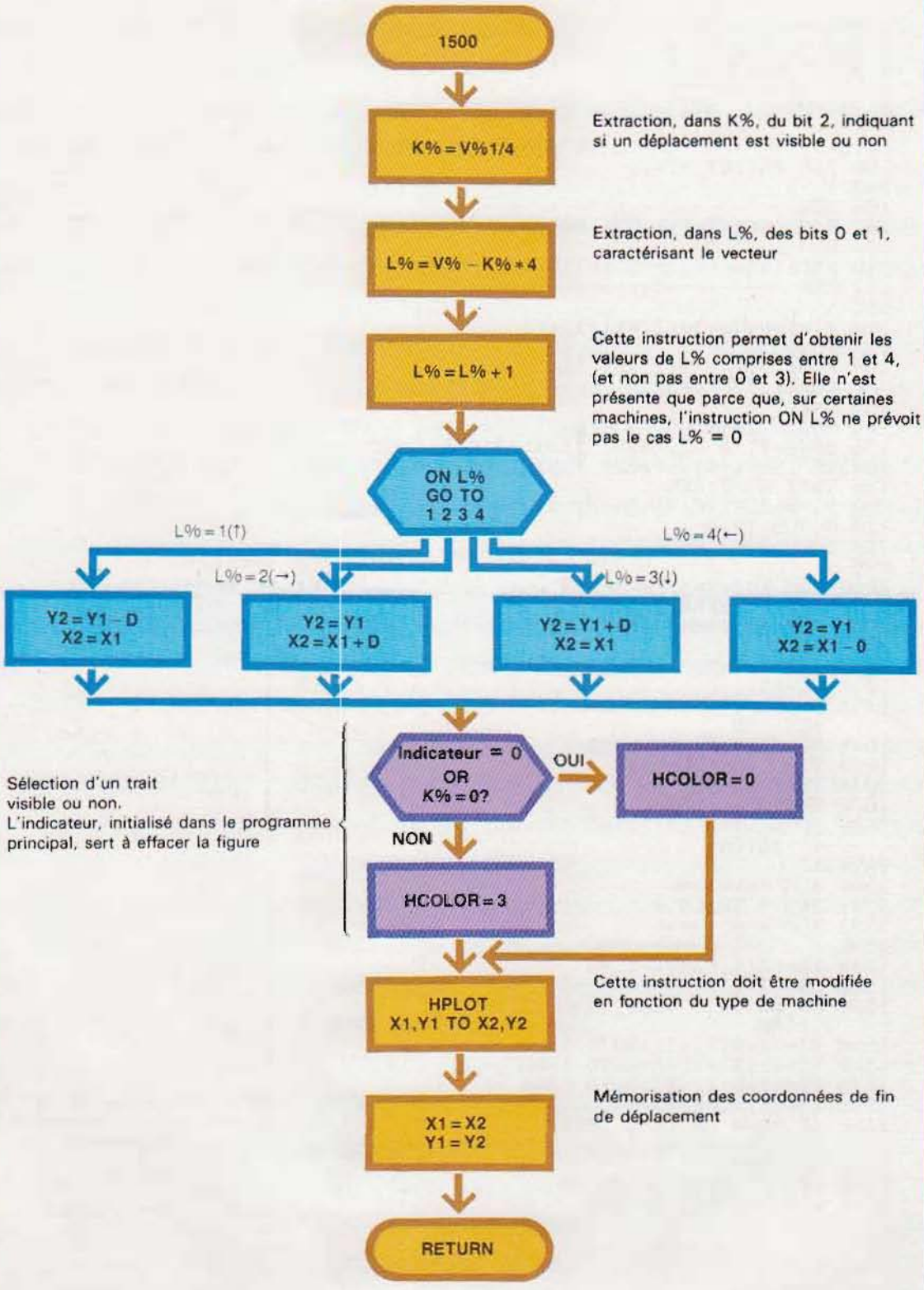
Dans la pratique, on peut difficilement réaliser un dessin complexe à partir d'une seule série de symboles. La meilleure approche consiste à découper le dessin en figures simples (**shapes**) et à les mémoriser dans une unique table de figures (**shape table**).
La table doit avoir alors une structure plus complexe.
La figure de la page 1597 montre une structure typique utilisée dans les machines dotées de ce type de logiciel. La première adresse (octet 0) renferme le nombre de figures présentes dans la table ; la deuxième est toujours nulle.
Les autres sont divisées en deux groupes :
— la zone des adresses, qui contient les pointeurs indiquant le début de chaque figure et constituant le répertoire de la table,
— la zone des vecteurs graphiques représentant chacune des figures de la table.
La fin de chaque figure est indiquée par une position dont le contenu est nul.

Voici comment fonctionne la table.
Le système examine le contenu de l'octet 0 (première adresse) et s'informe ainsi du nombre des figures présentes (3 dans l'exemple). Le deuxième octet étant toujours à zéro, il doit être sauté. Les adresses des différentes figures, au nombre de 3, se trouveront à partir de l'adresse 2, en avançant d'un pas de 2. Autrement dit, les adresses 2 et 3 contiendront le pointeur désignant le début de la première

SOUS-PROGRAMME DE PRESENTATION



SOUS-PROGRAMME DE TRACE

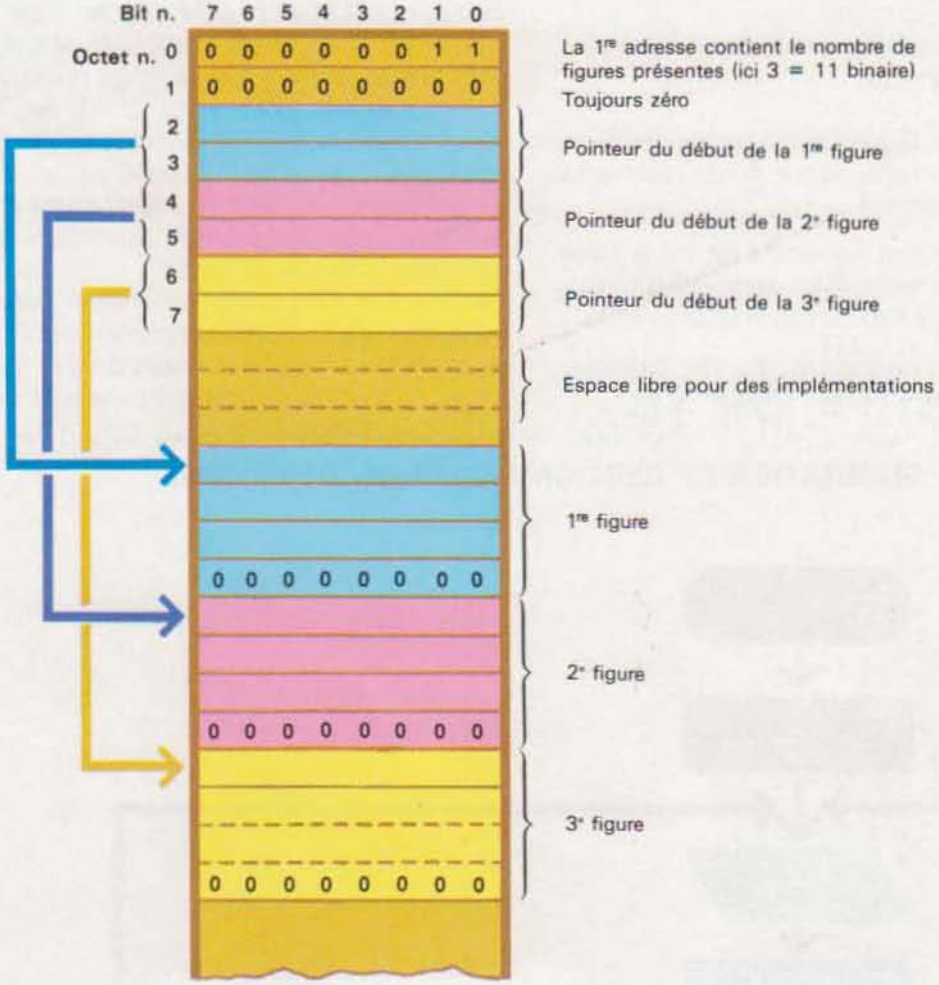


GESTION D'UNE TABLE DE FIGURES

Version Siprel, Apple et compatibles

```
1 REM ===========
2 REM *  TABLE  *
3 REM * FIGURES *
4 REM ===========
5 :
6 :
50 HGR
100 DIM A%(20):MX=20
103 :
106 REM ----------------------
107 REM CHARGEMENT DES VECTEURS
108 REM :
110 A%(1)=36:A%(2)=55:A%(3)=46
112 REM ------------------------
115 :
120 X1=100:Y1=50:X2=X1:Y2=Y1
125 FL=1
150 HOME:VTAB 22:INPUT "Longueur
    segment? ";F%
155 D=F%
160 GOSUB 1000
170 HOME:VTAB 22:INPUT "Suite? ";A$
180 IF LEFT$(A$,1)="O" THEN 200
190 TEXT:HOME:END
200 FL=0:XI=100:Y1=50:X2=X1:Y2=Y1
210 GOSUB 1000
220 GOTO 125
999 :
1000 REM ================
1001 REM * PRESENTATION *
1002 REM ================
1003 :
1010 FOR I=1 TO MX
1020 B%=A%(I)
1030 IF B%=0 THEN I=MX:NEXT I
     :RETURN
1040 D%=B%/8:C%=B%-D%*8
1050 V%=C%:GOSUB 1500
1060 V%=D%:GOSUB 1500
1070 NEXT I
1080 HOME:VTAB 22:PRINT "Erreur
     !":RETURN
1499 :
1500 REM =========
1501 REM * TRACE *
1502 REM =========
1503 :
1510 F%=V%/4:L%=V%-K%*4
1520 L%=L%+1
1530 ON L% GOTO 1550,1560,1570
     ,1580
1550 Y2=Y1-D:X2=X1:GOTO 1600
1560 Y2=Y1:X2=X1+D:GOTO 1600
1570 Y2=Y1+D:X2=X1:GOTO 1600
1580 Y2=Y1:X2=X1-D
1600 IF FL=0 OR K%=0 THEN
     HCOLOR=0:GOTO 1620
1610 HCOLOR=3
1620 HPLOT X1,Y1 TO X2,Y2
1630 X1=X2:Y1=Y2:RETURN
```

STRUCTURE D'UNE TABLE DE FIGURES



figure, les 4 et 5 le pointeur de la deuxième, et les 6 et 7 celui de la troisième. Une adresse est généralement laissée à zéro entre les figures et les tables pour permettre l'introduction de nouveaux pointeurs si les dimensions et la complexité du dessin augmentaient.

Programme de génération et de gestion de tables des figures

Il y a deux méthodes de création d'une table de figures. La première (et la plus simple) prévoit un dialogue avec l'utilisateur qui indique directement les déplacements à effectuer. La seconde (plus complexe du point de vue de la programmation, mais beaucoup plus utile) associe 4 touches aux 4 déplacements principaux ; de plus, chacun des déplacements est visualisé et mémorisé au fur et à mesure qu'il est activé.

Du point de vue de la production de la table, le résultat est le même, mais l'utilisateur voit progressivement le dessin se construire.

Examinons un programme de génération et d'utilisation de tables de figures utilisables dans les systèmes dépourvus de ces fonctions.

Avec les modifications adéquates, ce même programme peut servir dans les systèmes prévoyant la gestion des tables de déplacements. Dans ce type de machines, les fonctions accessoires, comme le changement d'échelle et les translations de la figure, seront obtenues à l'aide d'instructions spéciales.

Les principales fonctions exécutées par ce programme concernent :

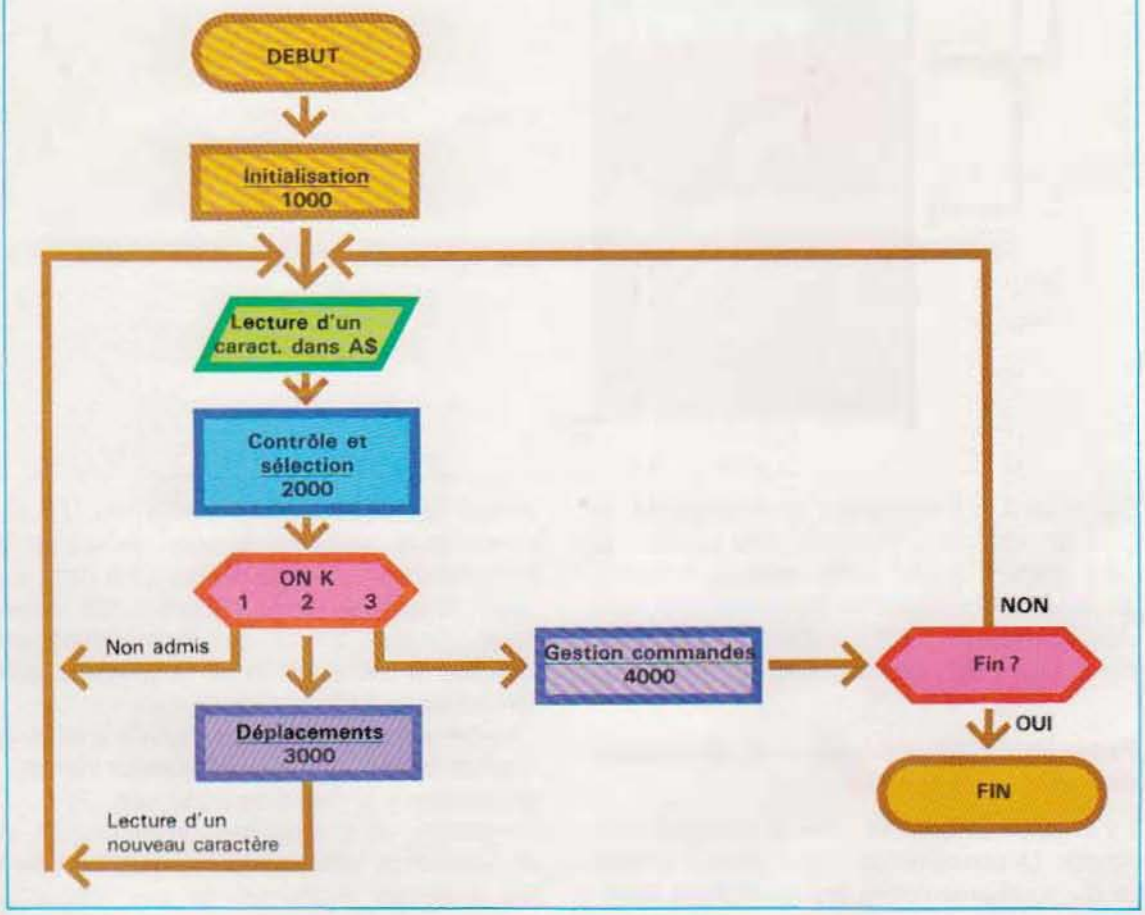
- La saisie des figures constituant le dessin, la mémorisation sous forme de déplacements et la visualisation.
- L'enregistrement sur disquette de la table des déplacements, avec possibilité de relecture pour des modifications ultérieures.
- La possibilité de changer l'échelle, de déplacer ou de faire tourner chaque figure.

L'organigramme du programme principal (ci-dessous), dont les variables sont énumérées dans le tableau de la page ci-contre, comporte quatre sous-programmes essentiels ou blocs fonctionnels.

Le premier (1000) initialise les variables, les instructions DATA et les indicateurs. Vient ensuite la lecture d'un caractère (A$), effectuée dans le programme principal, et qui doit utiliser l'instruction INPUT$(1) (sous CP/M) ou GET (ou un équivalent). Ces instructions, n'ayant pas d'écho, permettent d'entrer un caractère seul, sans avoir à fermer l'enregistrement comme avec l'instruction INPUT normale (il faut actionner la touche RETURN).

Le caractère ainsi entré est analysé dans le sous-programme 2000, qui répond par 3 valeurs contenues respectivement dans K, TS et TP. La valeur de K indique la touche enfoncée :

GENERATION ET GESTION DE TABLES DE FIGURES



VARIABLES ET SYMBOLES EMPLOYES DANS LE PROGRAMME

A$	= Caractère lu. Indique un caractère ou une commande.
A1$	= Deuxième caractère d'exécution de la commande
NM$	= Nom du fichier mémorisé ou rappel de la figure
K	= Indicateur du type de commande activée (non valide, déplacement, ordre)
TS	= Type de déplacement demandé
TP	= Type de commande
S(10)	= Tableau contenant les codes ASCII des touches de déplacement
O(10)	= Tableau contenant les codes ASCII des touches de commande
X1, Y1	= Coordonnées avant le déplacement
X2, Y2	= Coordonnées après le déplacement
DX, DY	= Valeurs du déplacement unitaire (points écran)
HC	= Indicateur de déplacement visible ou non visible. Activé par TP=1 ou 2
B(100)	= Tableau des déplacements constitutifs de la figure
IN	= Indice de la dernière position occupée dans B(100)
X0, Y0	= Coordonnées de début du dessin
FL	= Indicateur de déplacement
FR	= Indicateur de rotation
FS	= Indicateur d'échelle

Touches de commande

S = Déplacement	D = Disque (sauvegarde)
R = Rotation	C = Effacement
L = Chargement	+ = Agrandissement
	— = Réduction

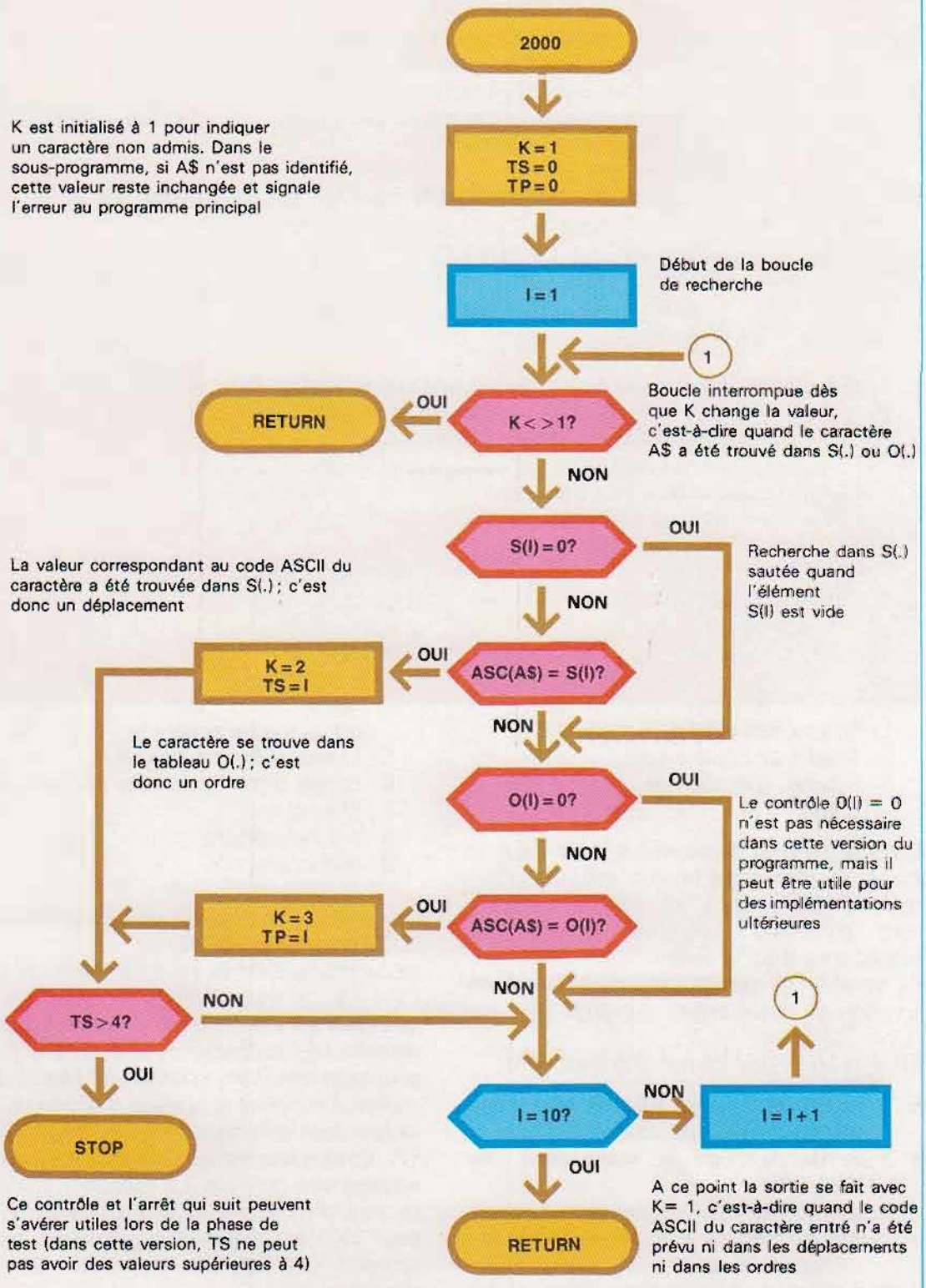
1 : Touche non prévue
2 : Touche de déplacement
3 : Touche d'entrée de commandes et ordres

Dans le cas d'un déplacement (pour tracer la figure) ou d'un ordre (mémorisation, changement d'échelle, etc.), les valeurs associées sont contenues dans les variables TS (déplacement) et TP (ordre).
La variable TS prendra la valeur 1 à 4 en fonction du déplacement demandé (↑, →, ↓, ←);
TP aura une valeur comprise entre 1 et 9 :

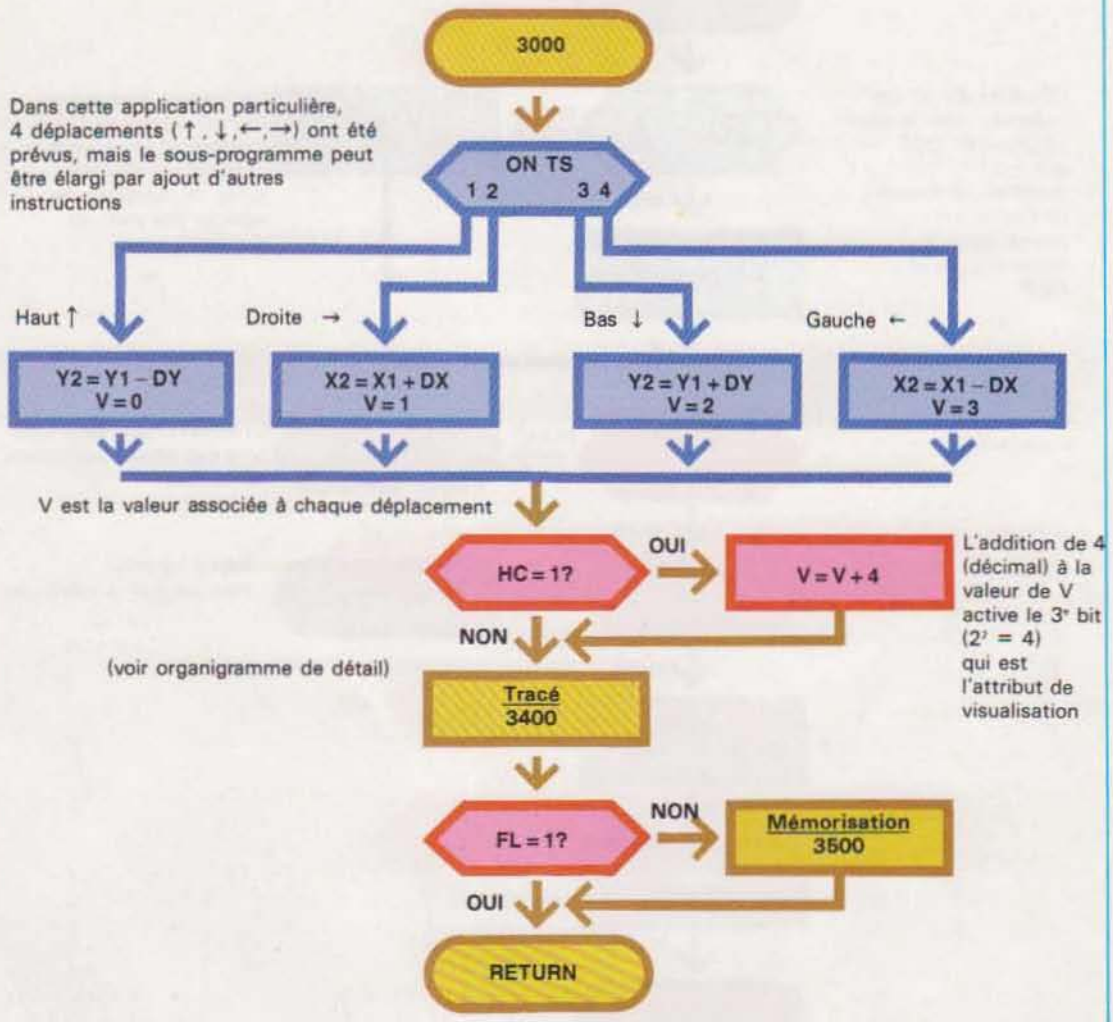
1 : Les déplacements suivant cette commande ne sont pas visibles
2 : Restauration de la visualisation des déplacements
3 : Déplacement de la figure dans la direction indiquée par la touche enfoncée immédiatement après
4 : Rotation de la figure dans le sens indiqué par la touche actionnée
5 : Mémorisation de la figure
6 : Rappel d'une figure mise en mémoire
7 : Effacement
8 : Agrandissement
9 : Réduction

L'organigramme de détail du sous-programme 2000 est en page 1600.
Le paramètre d'entrée est le caractère contenu dans A$. Le sous-programme vérifie que ce caractère est l'un de ceux qui ont été prévus pour les déplacements (dans le tableau S) ou pour les ordres (dans le tableau O). Dans l'affirmative, il transfère la position d'origine du caractère dans la variable correspondante (TS ou TP). Cette valeur indique également le type de déplacement ou d'ordre à exécuter.
Le sous-programme 3000 (organigramme en page 1601) est chargé de la gestion des déplacements. Les fonctions exécutées sont les suivantes :

SOUS-PROGRAMME DE CONTROLE DU CARACTERE ENTRE



ORGANIGRAMME DU SOUS-PROGRAMME DE GESTION DES DEPLACEMENTS



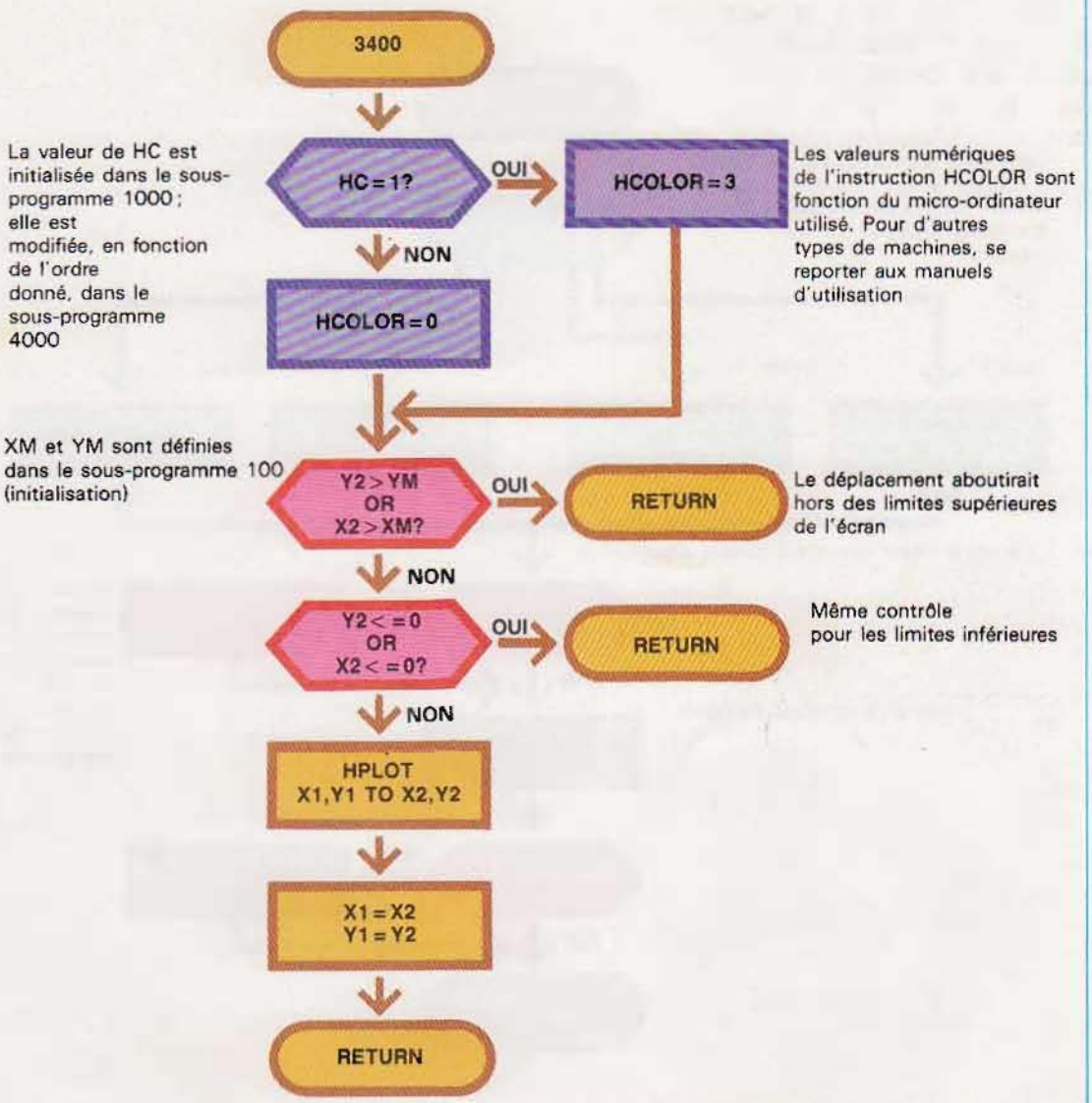
- Calcul des coordonnées après le déplacement et de la valeur V à mémoriser dans la table
- Transfert du déplacement dans la table (sous-programme 3500)
- Tracé du déplacement (sous-programme 3400).

Le calcul des coordonnées après le déplacement s'obtient en ajoutant à X1, Y1 les valeurs DX et DY. Les valeurs initiales de X1 et Y1 doivent être introduites dans le sous-programme 1000.

La valeur de V à mémoriser dans la table des déplacements n'est montrée que pour des raisons de clarté. On pourrait, en effet, l'obtenir simplement à l'aide de l'équation V = K—1, ou en faisant varier K de 0 à 3.

La nouvelle variable V ne serait alors plus nécessaire (K se trouve dans la plage de mémorisation et pourrait être utilisé directement). Mais des complications se produiraient au niveau du programme (par exemple, dans l'instruction ON K... certaines machines ne prévoient pas le cas K = 0).

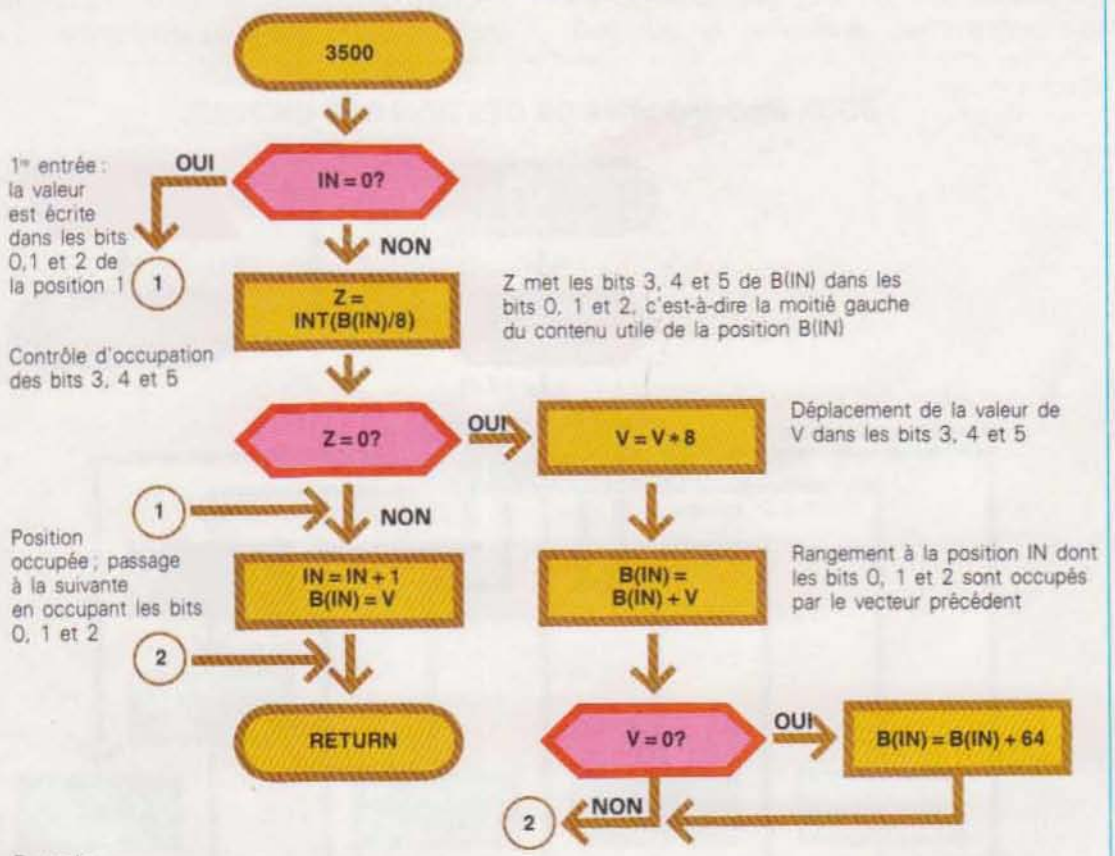
SOUS-PROGRAMME DU TRACE D'UN DEPLACEMENT ELEMENTAIRE



La condition de déplacement visible ou non est testée à l'aide de l'indicateur HC, qui prend la valeur 1 quand le caractère correspondant à TP = 2 est entré, et la valeur O pour TP = 1. Pour mettre à 1 le 3e bit du déplacement (attribut), il suffit d'ajouter 4 à la valeur de V. Au cours de ce même sous-programme (3000) sont appelés les sous-programmes 3500, qui mémorise le déplacement (V) dans le tableau B (100) et, 3400, qui visualise le déplacement.
Le sous-programme 3400 contrôle également les valeurs de X2 et Y2. Au terme de cette opération, si les valeurs excèdent les limites de l'écran, le déplacement n'est pas visualisé mais seulement mémorisé (avec éventuellement une indication non portée dans l'organigramme). Cette méthode a été choisie parce que, comme nous allons le voir, ce programme permet de changer d'échelle. Si un déplacement sort des limites de l'écran, il suffit d'activer l'état commandes et de modifier l'échelle de représentation pour y remédier.
Le sous-programme 3500 est, quant à lui, plus complexe car il tient compte de « l'encombre-

ment » des valeurs prises successivement par la variable V (voir ci-dessous l'organigramme correspondant).
L'indice IN contient la dernière position occupée dans le tableau B(100). Sa valeur initiale est de zéro. De ce fait, au premier déplacement, le sous-programme saute les instructions intermédiaires et mémorise + 1 dans IN, c'est-à-dire dans la partie droite de B(1).
Si l'on pose B(1) = V, la valeur V occupe la position lui correspondant, soit les bits 0 à 2. Lors du deuxième appel, IN ne vaut plus zéro mais il est activé par la partie centrale.
Ces instructions analysent le contenu de B(IN) divisé par 8. Si la donnée contenue dans B(IN) occupe seulement les bits 0, 1 et 2, la valeur entière de la division est zéro ; dans les autres cas, elle est différente de zéro.
Si le résultat de la division est 0, c'est que la partie droite (bits 0 à 2) est déjà occupée (il est impossible que la mémoire ne contienne que des zéros, car IN est incrémenté avant

SOUS-PROGRAMME DE MEMORISATION SOUS FORME DE TABLE DE FIGURES



Exemple :

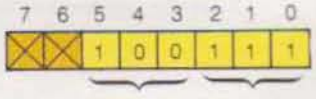
V=7 (déplacement visible à droite)

La mémoire est vide. La représentation binaire de 7 est 111 (écrite dans les bits 0, 1 et 2)

V=4 (déplacement visible vers le haut)

La mémoire utile est occupée dans sa partie droite (INT(B(.)/8=INT(7/8)=0) ; V doit alors être déplacé de 3 bits : V=V*8=32=1 0 0 0 0 0 ($32=2^5$). Le contenu de la mémoire correspond au tracé.

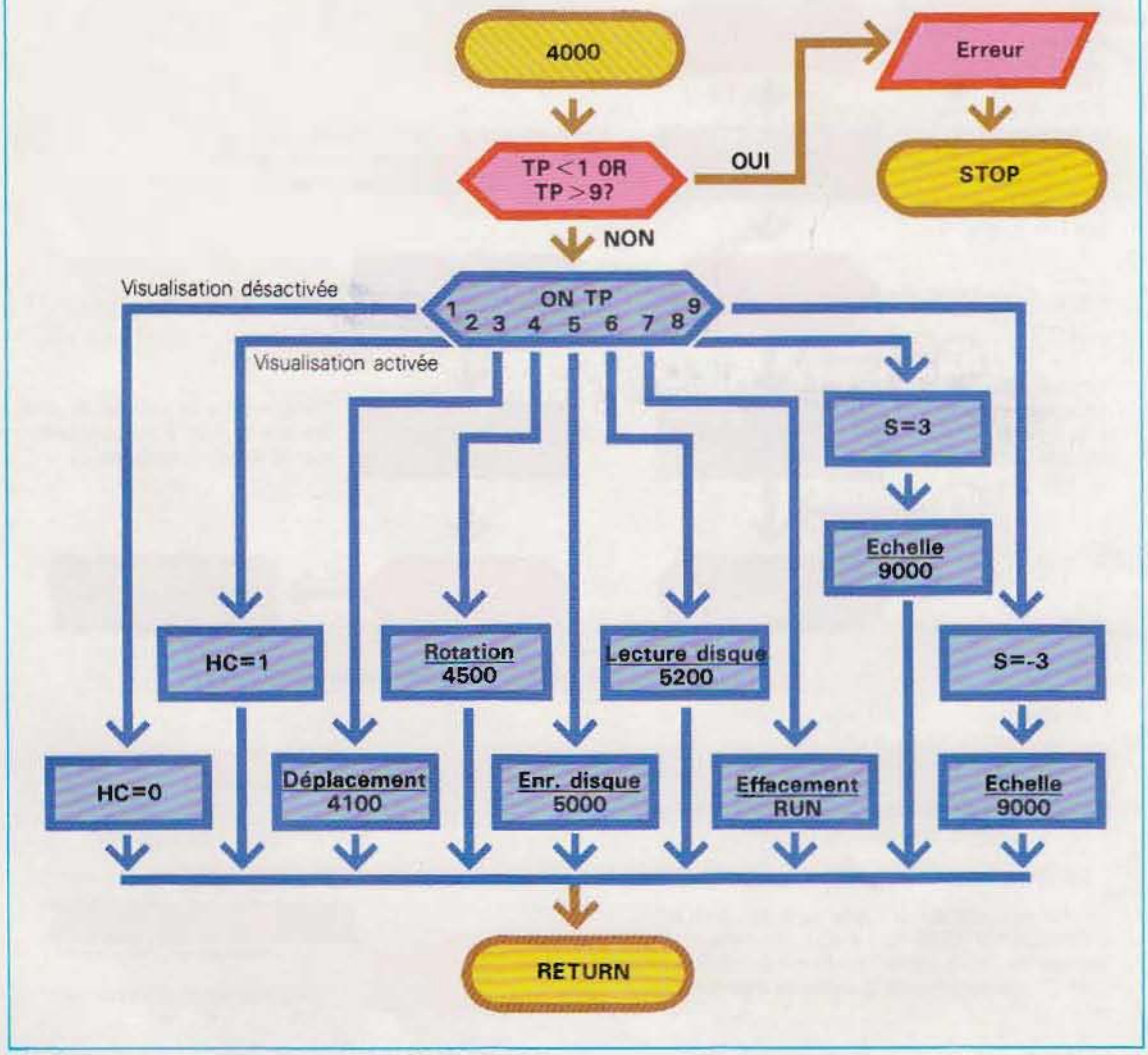


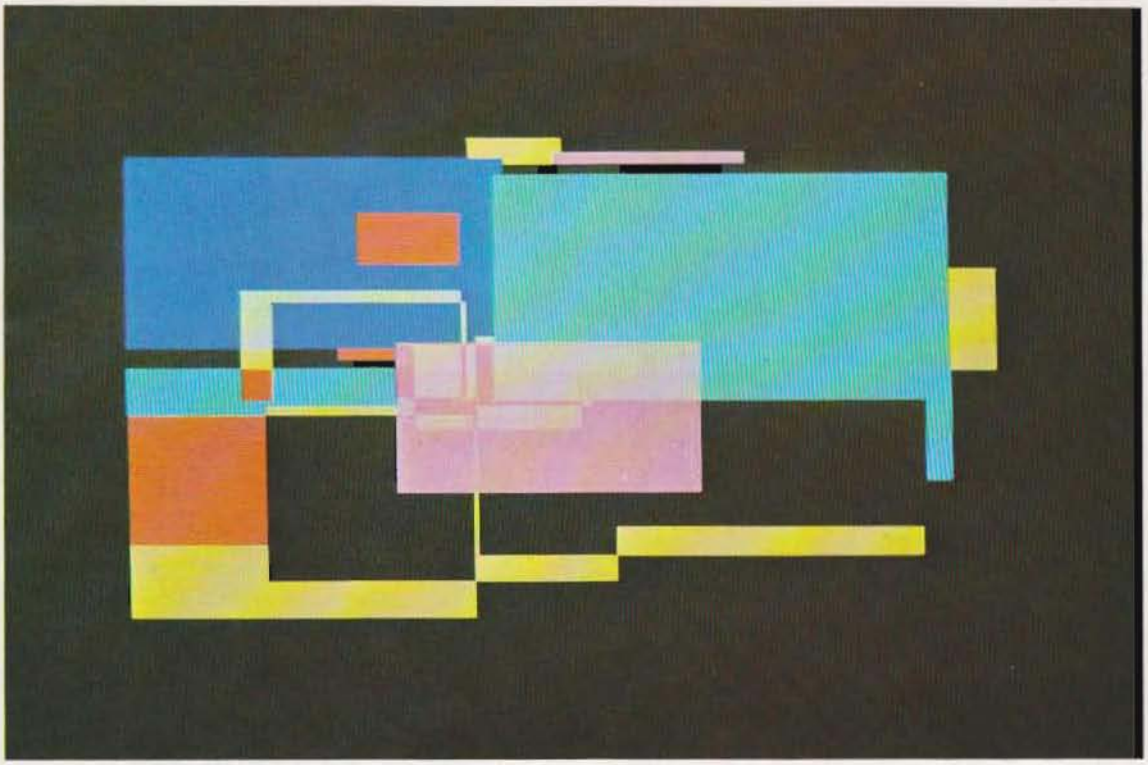


l'écriture). Il faut alors mémoriser V dans les bits 3, 4 et 5, ce qui s'obtient en multipliant sa valeur par 8 et en ajoutant le résultat au contenu précédent. Remarquons toutefois que, de ce fait, le déplacement entré à une adresse en premier se trouve à droite, alors que le suivant est à gauche.

On trouvera ci-dessous l'organigramme du sous-programme 4000 (gestion des ordres) qui se compose uniquement d'appels à d'autres sous-programmes dont chacun est consacré à un des neuf ordres possibles. De nouvelles fonctions seront éventuellement implémentées en introduisant d'autres instructions dans l'appel adressé (ON TP...) et, bien entendu, les sous-programmes nécessaires à leur exécution. Les fonctions de déplacement visible ou invisible nécessitent seulement l'attribution de la valeur 0 ou 1 à l'indicateur HC ; elles sont exécutées dans le sous-programme principal. Cette méthode de subdivision poussée à l'extrême est intéressante. En effet, le sous-programme 4000 aurait pu contenir l'ensemble des autres instructions et éliminé la nécessité de désigner les indicateurs à transmettre aux sous-programmes suivants. Ce sous-programme aurait été cependant beaucoup plus complexe et difficile à modifier. De plus, l'utilisation de sous-programmes consacrés à une tâche bien délimitée permet d'économiser des instructions en réutilisant le même sous programme en plusieurs points du programme.

SOUS-PROGRAMME DE GESTION DES ORDRES





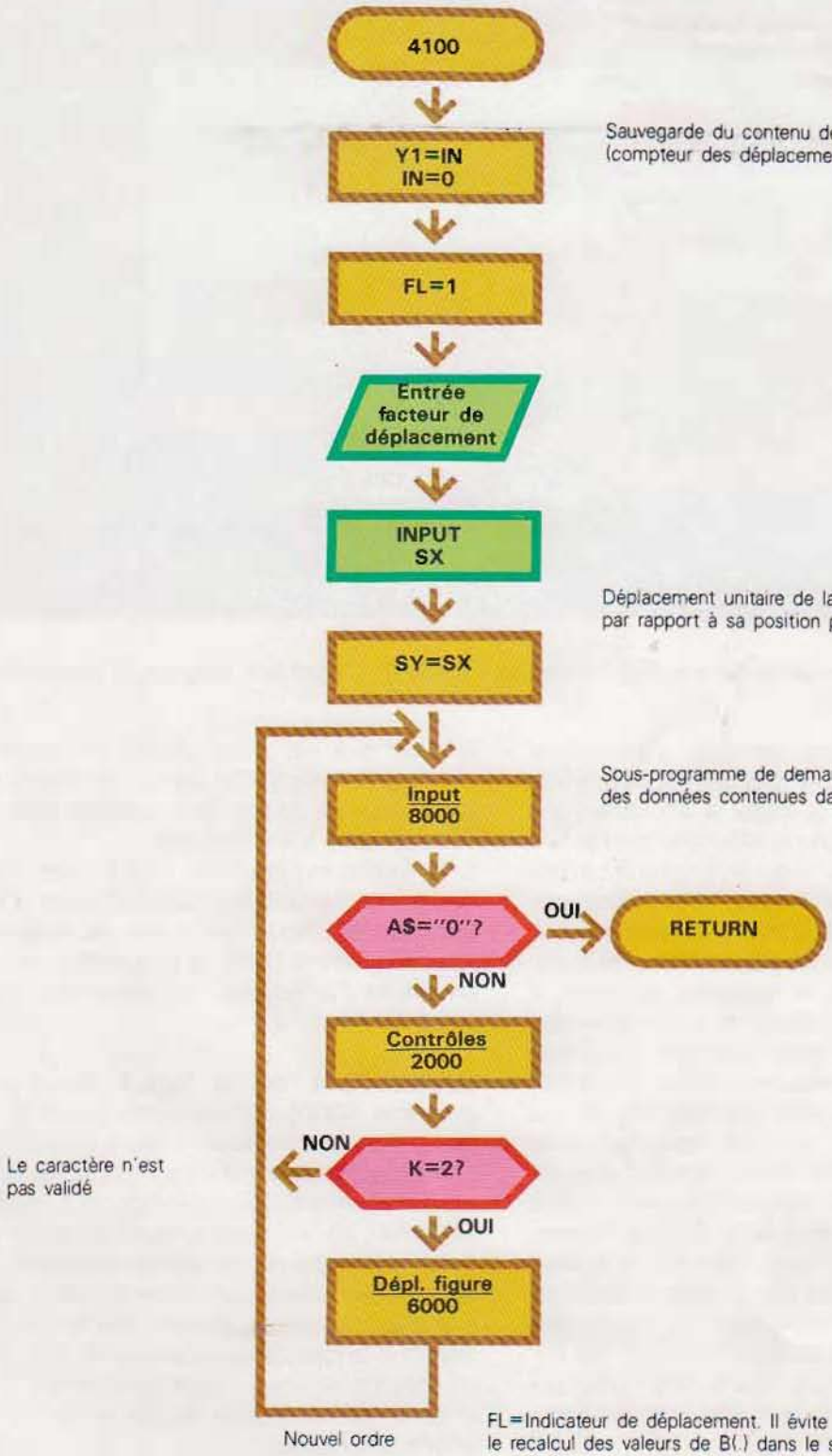
Centre THC

Composition graphique obtenue à l'aide d'instructions simples de haut niveau.

Gestion des départements d'ensemble (sous-programme 4100). Ce sous-programme a pour fonction de déplacer la totalité d'une figure dans une zone quelconque de l'écran. (Ne pas confondre avec le sous-programme 2000, qui se charge de la gestion des déplacements lors de la génération du dessin.) Il prévoit la sauvegarde de la table des déplacements B(100) avant la translation du dessin. Il suffit d'utiliser un tableau de correspondance pour que l'original reste inchangé. Il est toujours possible de retrouver son état précédent si les résultats ne sont pas satisfaisants. La première fonction à exécuter est la lecture du type de déplacement désiré. Il s'agit d'une partie du programme semblable à celle utilisée pour la génération de la table, la seule variante étant l'identification de la valeur 0, qui indique la fin de la translation. Les touches activant les déplacements du dessin sont les mêmes que celles qui servent à produire les tables des déplacements formant la figure : c'est pourquoi ils sont également contrôlés par le sous-programme 2000.
Dans ce cas, toutefois, il n'y a pas d'ordre : si la valeur de K est différente de 2 à la sortie du sous-programme 2000, elle est ignorée (K = 2 signifie qu'une des touches prévues pour les déplacements a été enfoncée).
Si la touche est identifiée (et s'il s'agit d'une des quatre prévues), le sous-programme 6000, qui déplace la figure, est activé. Au début du sous-programme 6000, le programme se met en attente d'un nouveau déplacement ou de la valeur 0.

Déplacement de la figure (sous-programme 6000). Ce sous-programme lit les déplacements constituant la figure dans B(100) et les dessins à partir d'un point dont les coordonnées sont modifiées par rapport à l'origine par défaut X0, Y0, d'une quantité SX ou SY, en fonction du type de translation demandée. La fonction est obtenue en extrayant, de la table des déplacements, l'attribut (visible ou non visible) et le type de déplacement (0, 1, 2, 3) et en utilisant les sous-programmes prévus pour la génération de la table. Il s'agit, en fait, d'une entrée simultanée.
Pour ne pas surcharger l'organigramme et le

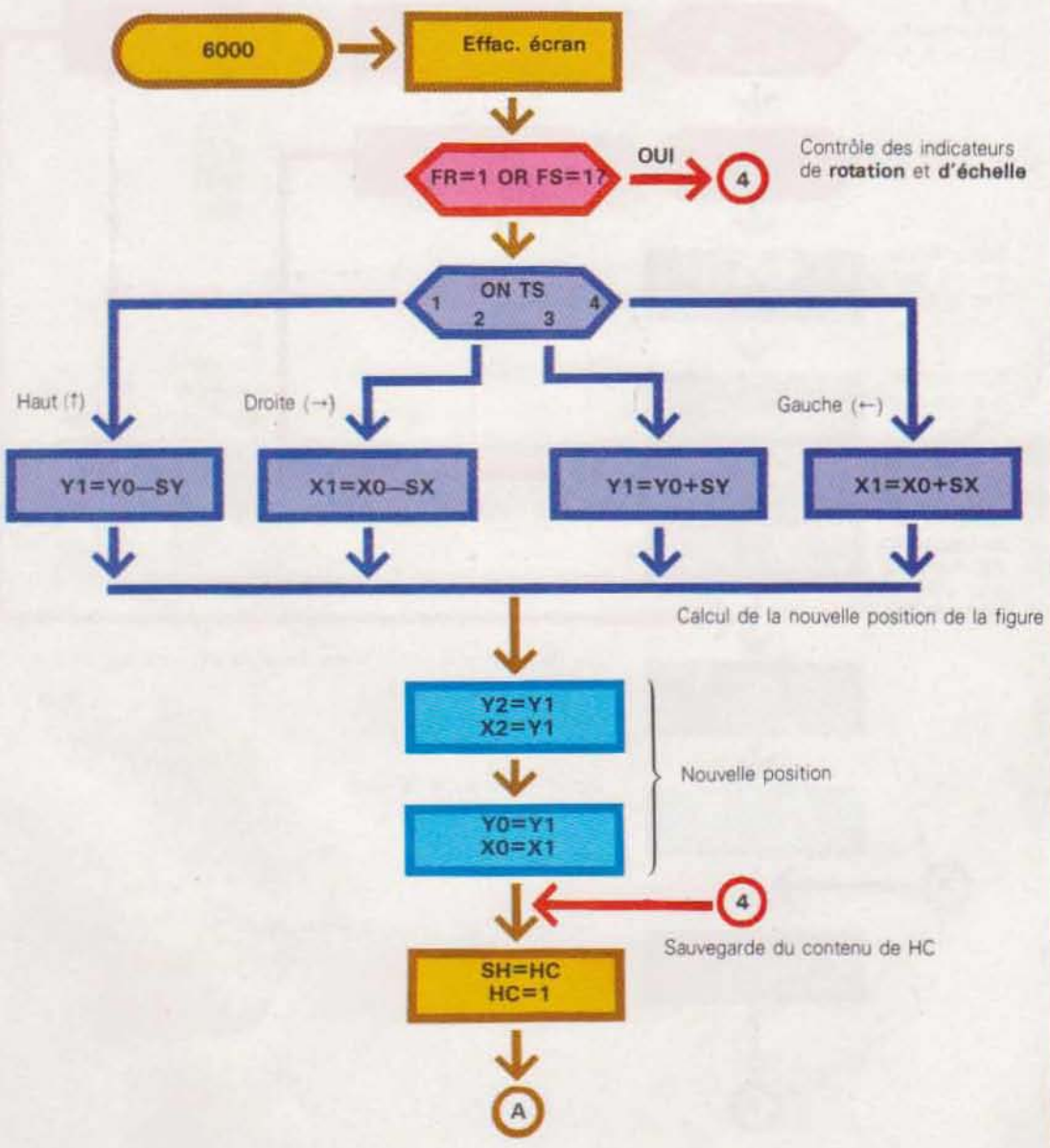
GESTION DES DEPLACEMENTS D'ENSEMBLES

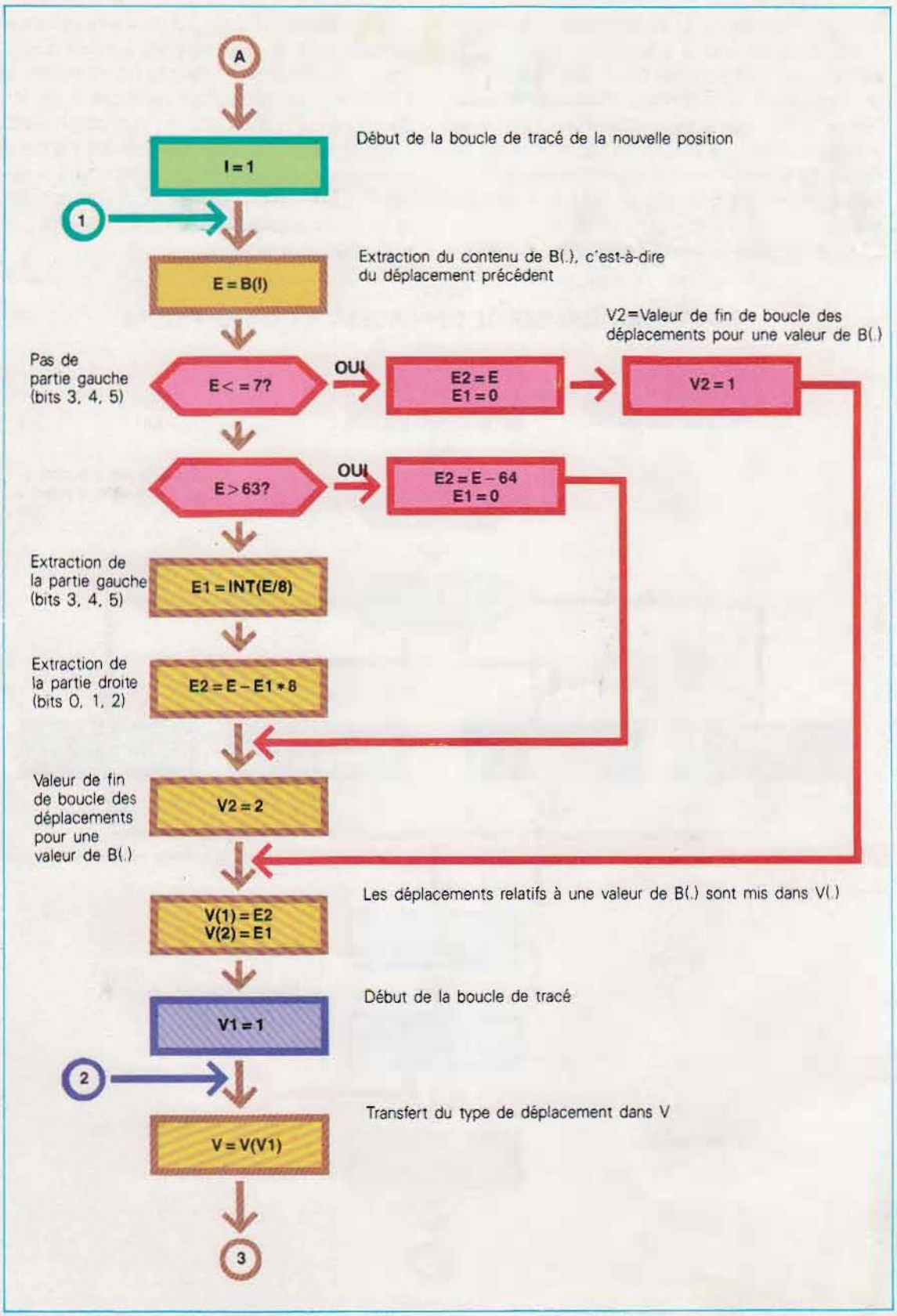


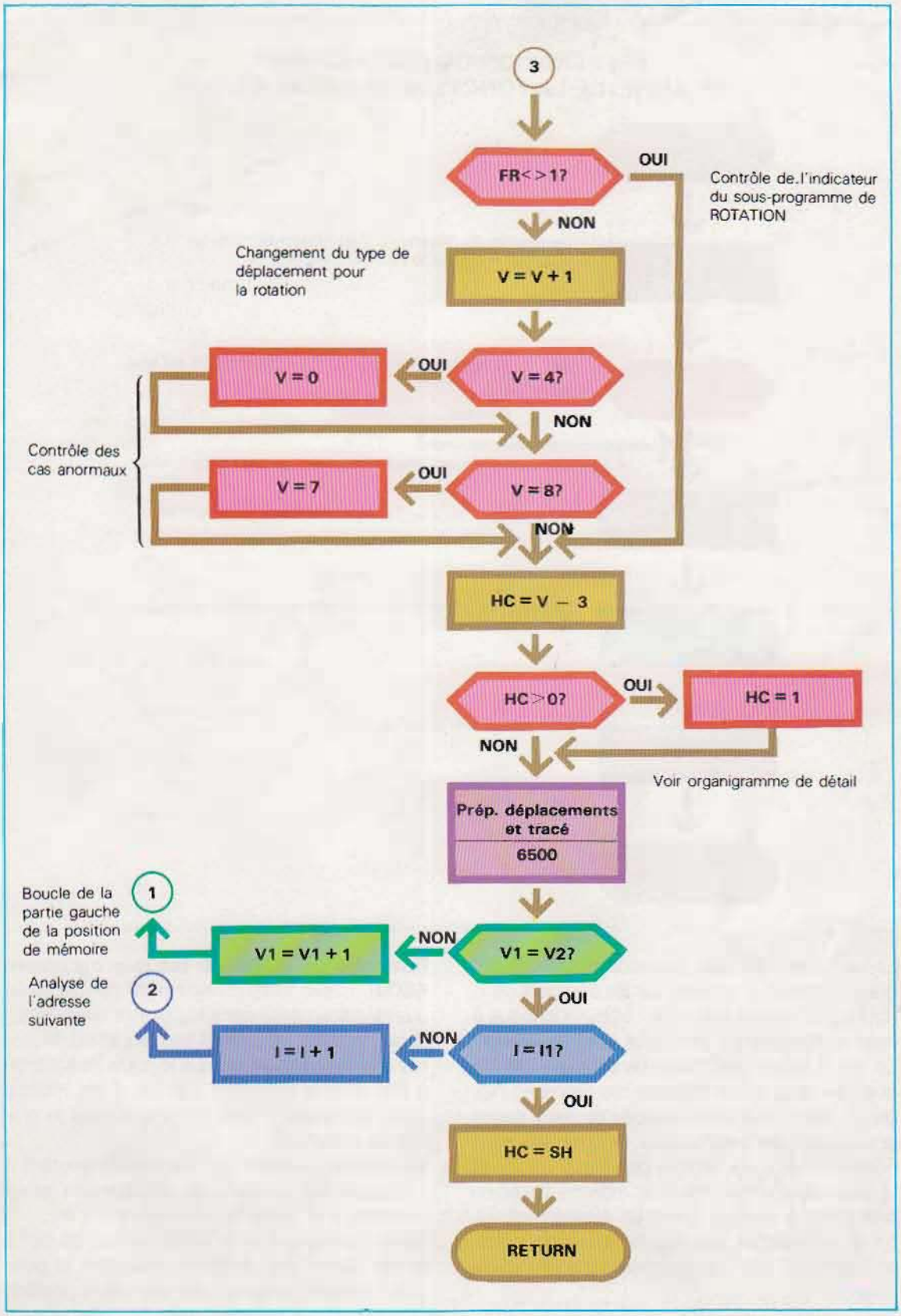
listing, l'effacement du déplacement précédent a été omis. Quand la translation d'une figure est activée, cette même figure est répétée en un autre point. Si la position finale est obtenue (après 10 passages intermédiaires par exemple), il y aura autant de répétitions de la figure. Chacune d'elles sera déplacée de 1/10 du déplacement final par rapport à la précédente.

Deux méthodes peuvent permettre de remédier à cet inconvénient. La première consiste à effacer toute la page graphique avant d'effectuer le déplacement (mais la figure initiale est également effacée). Pour remédier à cet état de fait, il suffit d'introduire l'instruction d'effacement de la mémoire vidéo avant l'appel du sous-programme 6000. La reconstruction de la figure à la position originale peut être activée à la fin du sous-programme 6000, quand il n'y a plus d'écran à vider.

SOUS-PROGRAMME DE DEPLACEMENT D'UNE FIGURE

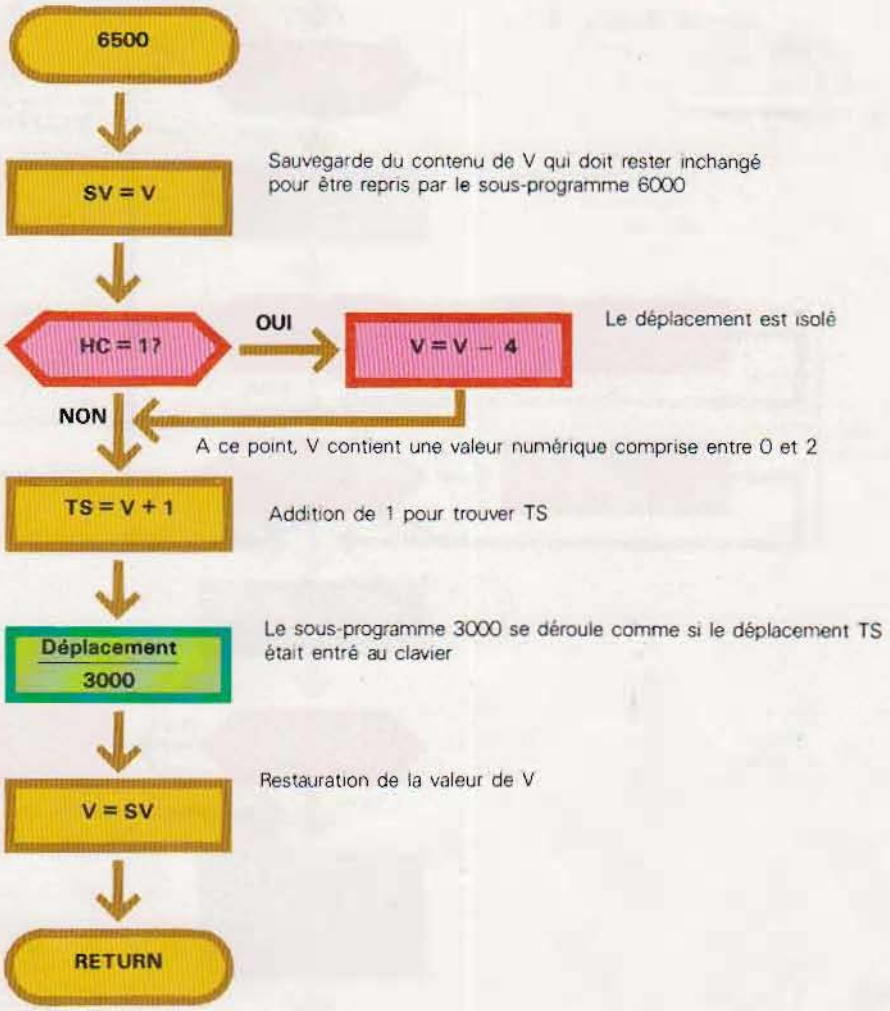



A
I = 1
Début de la boucle de tracé de la nouvelle position
1
E = B(I)
Extraction du contenu de B(.), c'est-à-dire du déplacement précédent
V2 = Valeur de fin de boucle des déplacements pour une valeur de B(.)
Pas de partie gauche (bits 3, 4, 5)
E < = 7?
OUI
E2 = E
E1 = 0
V2 = 1
E > 63?
OUI
E2 = E – 64
E1 = 0
Extraction de la partie gauche (bits 3, 4, 5)
E1 = INT(E/8)
Extraction de la partie droite (bits 0, 1, 2)
E2 = E – E1 * 8
Valeur de fin de boucle des déplacements pour une valeur de B(.)
V2 = 2
Les déplacements relatifs à une valeur de B(.) sont mis dans V(.)
V(1) = E2
V(2) = E1
Début de la boucle de tracé
V1 = 1
2
Transfert du type de déplacement dans V
V = V(V1)
3


3
FR<>1?
OUI
Contrôle de.l'indicateur du sous-programme de ROTATION
NON
Changement du type de déplacement pour la rotation
V = V + 1
V = 4?
OUI
V = 0
NON
Contrôle des cas anormaux
V = 8?
OUI
V = 7
NON
HC = V – 3
HC > 0?
OUI
HC = 1
NON
Voir organigramme de détail
Prép. déplacements et tracé
6500
Boucle de la partie gauche de la position de mémoire
1
V1 = V2?
NON
V1 = V1 + 1
Analyse de l'adresse suivante
2
OUI
I = I1?
NON
I = I + 1
OUI
HC = SH
RETURN

1609

PREPARATION DU DEPLACEMENT ET APPEL DE LA FONCTION DE VISUALISATION



La seconde méthode, beaucoup plus compliquée, consiste à repasser sur les contours de la figure précédente avec une couleur identique à celle du fond, ce qui provoque son effacement. En fait, il existe une troisième solution, valable pour certains systèmes comme Siprel et Apple. Quand l'ordinateur dispose de deux pages graphiques, on peut utiliser la première pour conserver la figure dans sa position d'origine et la seconde pour les positions intermédiaires en procédant à chaque fois à un effacement. En fin de translation, les figures (position initiale et déplacée) sont regroupées dans une seule page.

Rotation de la figure (sous-programme 4500). Cette fonction se révèle peu adaptée aux déplacements dans les quatre directions : le programme ne permet pas, en effet, de tracer des lignes inclinées. La logique de rotation à 90° dans le sens des aiguilles d'une montre n'est fournie qu'à titre d'exemple dans le graphique ci-contre.

La rotation s'obtient simplement en ajoutant 1 à chacun des vecteurs de déplacement et en revenant à 0 quand la somme atteint 4.

Dans l'exemple donné, la rotation est de 90° à droite. Selon une méthode analogue, la position initiale de la figure basculée est la position

par défaut (X0, Y0) plus une constante (10). Pour extrapoler à partir de ce programme, il faut y ajouter la possibilité, pour l'utilisateur, d'introduire sa sélection.

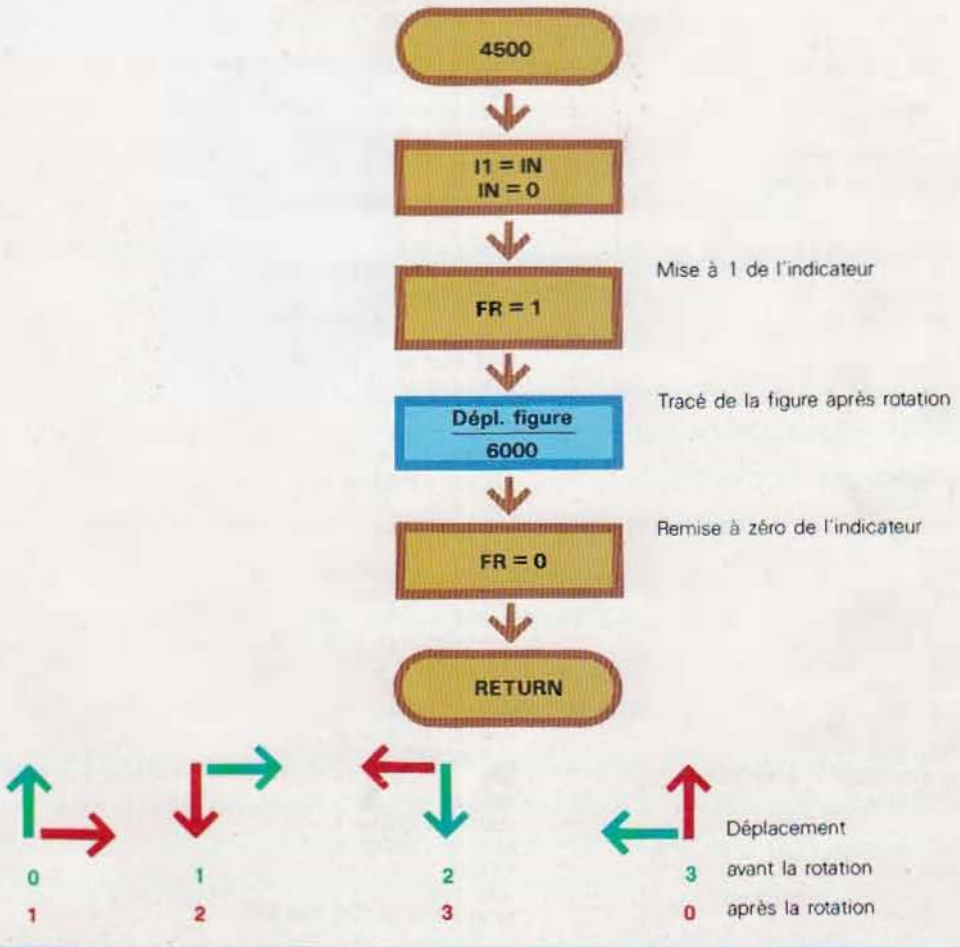
Enregistrement sur le disque (sous-programme 5000). Le sous-programme comprend uniquement l'entrée du nom du fichier qui devra contenir B (100) et son enregistrement sur le disque. C'est pourquoi nous n'en donnerons que le listing (pages 1612 et 1614).

Lecture du disque (sous-programme 5200).
Ce sous-programme exécute les fonctions inverses du précédent : une fois entré le nom du fichier contenant les données du dessin à visualiser, il transfère son contenu dans B (100) et appelle le sous-programme de présentation 6000. (Voir listing page 1612 à 1614).



Une des phases du fonctionnement du programme (voir listing p. 1612 à 1614). C'est la page graphique 1, réservant 4 lignes pour le texte, qui a été sélectionnée. Ici, la fenêtre de texte sert à l'affichage du menu des commandes (voir ligne 118).

SOUS-PROGRAMME DE ROTATION



CREATION ET GESTION DES TABLES DE FIGURES

```
1 REM ==================
2 REM *  CREATION ET   *
3 REM *    GESTION     *
4 REM * FIGURES HAUTE  *
5 REM *   RESOLUTION   *
6 REM ==================
7 :
8 :
100 GOSUB 1000
110 :
111 REM ===============
112 REM *  LECTURE     *
113 REM * CARACTERES   *
114 REM ===============
115 :
117 HGR:HOME
118 VTAB 21:PRINT "I=HAUT/M=BAS
    /K=DROITE/J=GAUCHE/F=INV/V
    =VIS":PRINT "S=DEPL/R=ROT/D
    =DISQUE/L=CHARG/C=EFFAC"
120 GOSUB 8000
140 GOSUB 2000
150 ON K GOTO 160,170,190
160 GOTO 120
170 GOSUB 3000
180 GOTO 120
190 :
192 GOSUB 4000
200 GOTO 120
300 STOP
1000 :
1001 REM ===================
1002 REM * INITIALISATION *
1003 REM ===================
1004 :
1010 DIM B(100)
1020 DX=10:DY=10:Y0=50:X0=100
     :XM=270:YM=159:Y1=Y0:X1=X0

1025 X2=X1:Y2=Y1:S=1
1030 S(1)=73:S(2)=75:S(3)=77
     :S(4)=74
1035 D(1)=70:D(2)=86:D(3)=83
     :D(4)=82:D(5)=68:D(6)=76

1040 D(7)=67:D(8)=43:D(9)=4
     5
1045 D$=CHR$(4)
1100 RETURN
2000 :
2001 REM ==============
2002 REM * CONTROLE   *
2003 REM * CARACTERE  *
2004 REM ==============
2005 :
2010 K=1:TS=0:TP=0:I=1
2020 IF K<>1 THEN RETURN
2030 IF S(I)=0 THEN 2050
2040 IF ASC(A$)=S(I) THEN K=2
     :TS=I:GOTO 2200
2050 IF D(I)=0 THEN 2070
2060 IF ASC(A$)=D(I) THEN K=3
     :TP=I:GOTO 2200
2070 IF I=10 THEN RETURN
2080 I=I+1:GOTO 2020
2100 :
2200 :
2210 GOTO 2070
3000 :
3001 REM ==================
3002 REM *    GESTION    *
3003 REM * DEPLACEMENTS  *
3004 REM ==================
3005 :
3010 ON TS GOTO 3020,3030,3040
     ,3050
3011 :
3012 REM  < HAUT >
3013 :
3020 Y2=Y1-DY:V=0:GOTO 3100

3021 :
3022 REM  < DROITE >
3023 :
3030 X2=X1+DX:V=1:GOTO 3100

3031 :
3032 REM  < BAS >
3033 :
3040 Y2=Y1+DY:V=2:GOTO 3100

3041 :
3042 REM  < GAUCHE >
3043 :
3050 X2=X1-DX:V=3
3060 :
3100 :
3105 IF HC=1 THEN V=V+4
3110 GOSUB 3400
3115 IF FL=1 THEN RETURN
3150 GOSUB 3500
3160 RETURN
3400 :
3401 REM ================
3402 REM *    TRACE     *
3403 REM * DEPLACEMENT  *
3404 REM ================
3405 REM :
3420 IF HC=1 THEN HCOLOR=3
     :GOTO 3440
3430 HCOLOR=0
3440 IF Y2>YM OR X2>XM THEN
     RETURN
3450 IF Y2<=0 OR X2<=0 THEN
     RETURN
3460 HPLOT X1,Y1 TO X2,Y2
3470 X1=X2:Y1=Y2
3480 RETURN
3500 :
3501 REM ==================
3502 REM * MEMORISATION  *
3503 REM *    TABLES     *
3504 REM ==================
3505 :
3510 IF IN=0 THEN 3540
3520 Z=INT(B(IN)/8)
3530 IF Z<>0 THEN 3540
3535 V=V*8:B(IN)=B(IN)+V:IF
     V=0 THEN B(IN)=B(IN)+64
     :GOTO 3550
3537 GOTO 3550
3540 IN=IN+1:B(IN)=V
3550 RETURN
4000 :
4001 REM ============
4002 REM * GESTION   *
4003 REM * ORDRES    *
4004 REM ============
4005 :
4010 IF TP<1 OR TP>9 THEN STOP
```

```
4011 :
4020 ON TP GOTO 4030,4040,4050
     ,4060,4070,4080,4085,4090
     ,4095
4030 HC=0:RETURN
4040 HC=1:RETURN
4050 GOSUB 4100:RETURN
4060 GOSUB 4500:RETURN
4070 GOSUB 5000:RETURN
4080 GOSUB 52o0:RETURN
4085 RUN
4090 S=3:GOSUB 9000:RETURN
4095 S=-3:GOSUB 9000:RETURN

4100 :
4101 REM ================
4102 REM *    GESTION    *
4103 REM * DEPLACEMENTS *
4104 REM ================
4105 :
4106 FL=1
4140 I1=IN
4145 VTAB 23:HTAB 12:INPUT
     "Facteur de déplacement? " ;SX

4147 SY=SX
4148 VTAB=23:HTAB=22:PRINT "
4150 GOSUB 8000
4160 IF A$="0" THEN 4300
4170 GOSUB 2000
4180 IF K<>2 THEN 4150
4190 GOSUB 6000
4200 GOTO 4150
4300 :
4310 :
4320 :
4325 IN=I1
4327 FL=0
4330 RETURN
4500 :
4501 REM =============
4502 REM * ROTATION *
4503 REM =============
4504 :
4510 I1=IN:IN=0
4520 FR=1:GOSUB 6000
4530 FR=0
4540 RETURN
5000 REM ==================
5001 REM * ENREG. DISQUE *
5002 REM ==================
5003 :
5010 VTAB 23:HTAB 12:PRINT "Nom
     :"
5020 VTAB 23:HTAB 19:INPUT " "
     ;NM$
5030 VTAB 1:PRINT D$"Open "NM$
5040 PRINT D$"Write "NM$
5045 PRINT IN
5050 FOR I=1 TO IN: PRINT B(I)
     :NEXT I
5060 PRINT D$"Close "NM$:PRINT
     D$
5070 RETURN
5200 REM ===============
5201 REM * CHARGEMENT *
5202 REM ===============
5203 :
5210 VTAB 23:HTAB 12:PRINT "Nom
     "
5220 VTAB 23:HTAB 19:INPUT " "
     ;NM$
5230 VTAB 1:PRINT D$"Open "NM$
5240 PRINT D$"Read "NM$
5250 INPUT I1
5255 IN=I1
5260 FOR I=1 TO I1:INPUT B(I)
     :NEXT I
5270 PRINT D$"Close "NM$:PRINT
     D$
5273 X0=100:Y0=50:X1=X0:Y1=Y0:X2
     =X1:Y2=Y1:FL=1
5275 GOSUB 6105:FL=0
5200 RETURN
6000 :
6001 REM ================
6002 REM * DEPLACEMENT *
6003 REM *   FIGURE    *
6004 REM ================
6005 :
6007 HGR
6008 IF FR=1 OR FS=1 THEN 6105

6010 ON TS GOTO 6020,6030,6040
     ,6050
6020 Y1=Y0-SY:GOTO 6100
6030 X1=X0+SX:GOTO 6100
6040 Y1=Y0+SY:GOTO 6100
6050 X1=X0-SX
6100 X2=X1:Y2=Y1:Y0=Y1:X0=X1

6105 SH=HC:HC=1
6110 FOR I=1 TO I1
6120 E=B(I)
6130 IF E<=7 THEN E1=0=E2=E:V2=1
     :GOTO 6155
6135 IF E>63 THEN E2=E-64:E1=0
     :GOTO 6153
6140 E1=INT(E/8)
6150 E2=E-E1*8
6153 V2=2
6155 V(1)=E2:V(2)=E1
6160 V1=1
6165 V=V(V1)
6167 IF FR<>1 THEN 6175
6169 V=V+1
6170 IF V=4 THEN V=0
6172 IF V=8 THEN V=4
6175 HC=V-3:IF HC>0 THEN HC=1

6180 GOSUB 6500
6185 VTAB 23:HTAB 35:PRINT TS
6190 :
6200 IF V1<>V2 THEN V1=V1+1
     :GOTO 6165
6210 NEXT I
6220 IF I<I1 THEN I=I1:NEXT I

6230 HC=SH:RETURN
6500 :
6501 REM ================
6502 REM * PREPARATION *
6503 REM * D"PLACEMENT *
6504 REM ================
6505 :
6510 SV=V:IF HC=1 THEN V=V-4
6520 TS=V+1:GOSUB 3000
6530 V=SV:RETURN
8000 :
8001 REM ==========
8002 REM * INPUT *
```

```
8003 REM ==========
8004 :
8010 VTAB 23:HTAB 12:PRINT
     "Entrez (?) commande "
8020 VTAB 23:HTAB 22:GET A$
8021 VTAB 23:HTAB 35:INVERSE
     :PRINT A$:NORMAL
8022 VTAB 23:HTAB 1:PRINT "
                "
8024 VTAB 23:PRINT "X="X1"Y="Y1

8025 IF ASC(A$)=13 THEN TEXT
     :HOME:END
8030 RETURN
9000 REM ===========
9001 REM * ECHELLE *
9002 REM ===========
9003 :
9010 FS=1
9015 DY=DY+S
9020 DX=DX+S:I1=IN:IN=0
9030 GOSUB 6000
9040 FS=0:RETURN
```

Programmes utilitaires pour la création de formes graphiques

L'exemple donné pour l'établissement d'une table archivant les déplacements illustre une méthode de création d'images graphiques avec des systèmes ne disposant pas d'instructions spéciales.

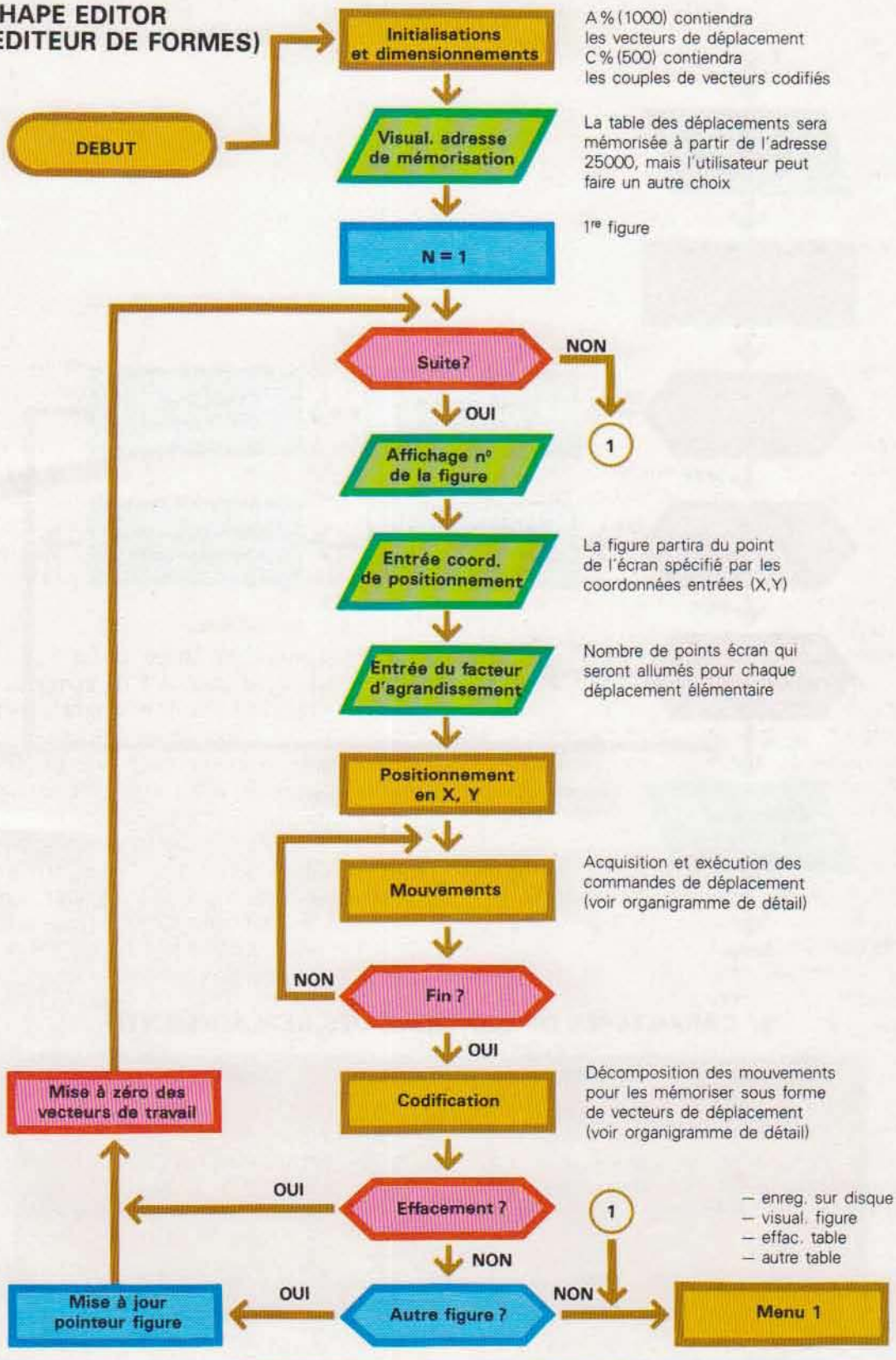
Les micro-ordinateurs — qui prévoient des tables de formes — recourent à deux types d'instructions.
— Le premier (ordinateurs basés sur l'unité centrale 6502 de la famille Apple et compatibles) reprend les déplacements élémentaires, les symboles et les conventions décrits dans notre exemple.
— Le second dispose d'une chaîne de commandes dans laquelle les déplacements sont indiqués les uns à la suite des autres.

Pour générer une table de formes avec le premier type, la meilleure solution consiste à écrire directement en mémoire les vecteurs des déplacements, à l'aide de l'instruction POKE. Il faut donc tout d'abord dessiner la table sur une feuille de papier, en détaillant les déplacements élémentaires, puis la charger en mémoire comme nous l'avons déjà vu.
Mais on peut également simuler les instructions de haut niveau de gestion et de visualisation de formes complexes, même sur des machines dans lesquelles elles n'ont pas été prévues.
Nous présentons ici deux exemples très pratiques.
— Le premier programme, que nous avons appelé « Shape Editor » (éditeur de forme) en reprenant une terminologie très courante, permettra de construire directement à l'écran des formes parfois très complexes et de sauvegarder sur disque, sous un nom conventionnel, les tables des déplacements.
— Le second programme, qui est le « Shape Loader » (chargeur de forme) permettra, à l'inverse, de charger et de visualiser à partir du disque une figure précédemment élaborée.

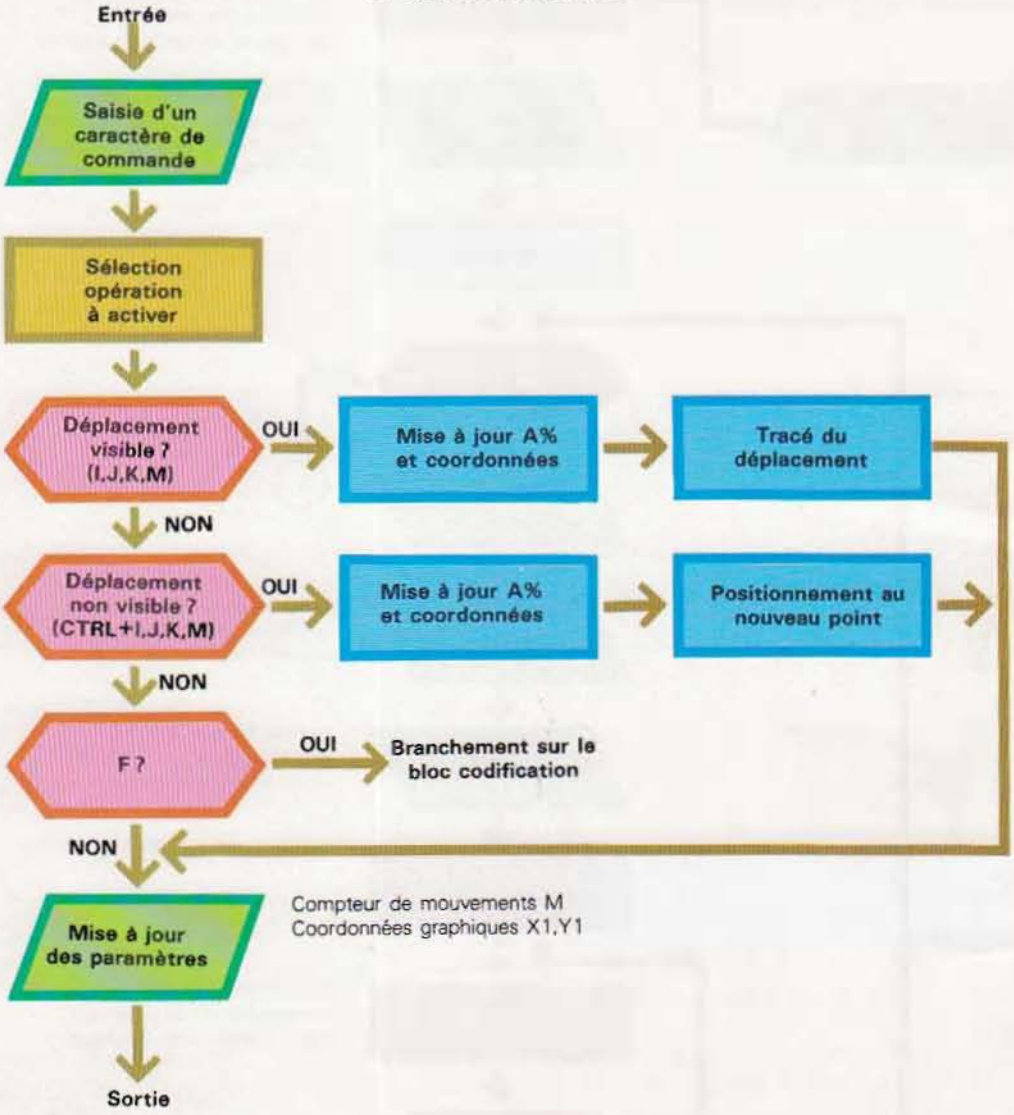
L'éditeur de formes.

L'organigramme du Shape Editor (page ci-contre) prévoit la possibilité de composer un dessin à l'écran à l'aide d'un ensemble de touches, activant le déplacement, visible ou non, d'une pointe, traçante imaginaire. Le bloc 1 détaille l'acquisition et l'exécution des commandes de déplacement (page 1616).
Une fois tracée la forme graphique, on parvient au bloc (3) de codification (page 1617) : les déplacements de la pointe enregistrés dans le tableau A%, sont convertis en un ensemble de vecteurs graphiques et transférés dans le tableau C%. En fin de codification, les vecteurs de C% sont chargés en mémoire. A ce stade, l'utilisateur peut décider de composer une autre figure, qui sera traitée de la même manière par le programme.
Quand le tracé des formes désirées est achevé, on retourne au menu et on commande les enregistrements sur disque sous un nom d'ensemble désignant la table des figures. Dans chaque table, les différentes formes sont numérotées par ordre croissant.
Ce programme est listé dans sa version Siprel, Apple et compatibles, en pages 1618 et 1621.
Les variables employées ont été regroupées dans le tableau de la page 1623.

SHAPE EDITOR (EDITEUR DE FORMES)



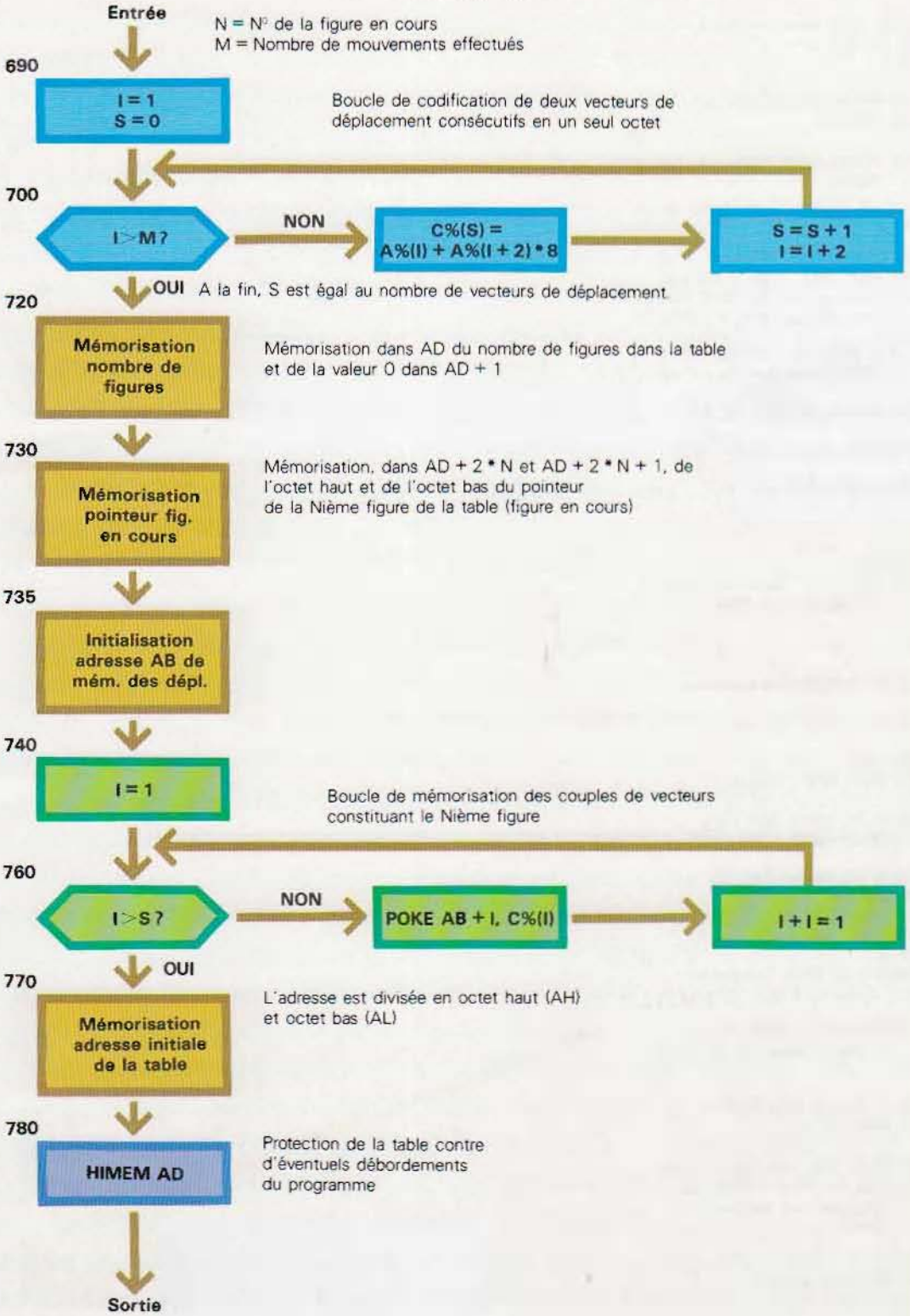
1/ MOUVEMENTS



2/ CARACTERES DE CONTROLE DES DEPLACEMENTS

Caractère	Mode	Direction	Codification dans A%
CTRL + I	Non visible	Vers le haut	0
CTRL + K	Non visible	A droite	1
CTRL + M	Non visible	Vers le bas	2
CTRL + J	Non visible	A gauche	3
I	Visible	Vers le haut	4
K	Visible	A droite	5
M	Visible	Vers le bas	6
J	Visible	A gauche	7

3/ CODIFICATION



L'EDITEUR DE FORMES

Version Siprel, Apple et compatibles

```
70 REM Peut contenir jusqu à
80 REM 30 figures
90 :

100 DIM A%(1000),C%(500)
110 TEXT:HOME

120 INVERSE:PRINT "Editeur de figures"
    :NORMAL

140 PRINT:PRINT

150 AD=25000
160 PRINT "Adresse de la table de
    figures = 25.000":PRINT "Voulez
    -vous changer (O/N) ?";:INPUT Z$

170 IF LEFT$(Z$,1)="O" THEN PRINT
    :INPUT "Nouvelle valeur ?";AD

180 PRINT:PRINT "OK !";AD

190 AH=INT(AD/256):AL=AD-AH*256

200 BL=62:BH=0:BB=62


210 N=1
220 HOME
230 PRINT:PRINT "Numéro de figure ?"
    :INVERSE:PRINT N:NORMAL


240 PRINT:PRINT "<1> Poursuivre"

250 PRINT:PRINT "<2> Stop"

260 PRINT
270 PRINT;INPUT "Choisir";A$

280 IF VAL (A$)=2 THEN N=N-1
    :BB=BB-5:GOTO 1000

290 IF VAL (A$)<>1 THEN 220
300 HGR
310 HOME
320 S=1
330 M=1
340 VTAB 22:INPUT "Coordonnées
    initiales ?";X,Y

350 HOME:VTAB 22:INPUT "Facteur
    d'agrandissement (0-10) ?";IN


360 IF IN<1 OR IN>10 THEN 350
370 HOME:VTAB 21

380 PRINT "<I>, <J>, <K>, <M> pour
    dessiner (avec <CTRL> pour se
    déplacer sans dessiner), <F>
    pour finir.";

390 :
400 REM Point initial
```

```
420 HCOLOR=3
430 HPLOT X,Y
440 X1=X:Y1=Y

450 :

460 REM Mouvements
470 :

480 HTAB 1:VTAB 1:GET A$


490 HTAB 1:VTAB 24:PRINT "Mouvements"
    :INVERSE:PRINT M:NORMAL



500 IF A$="I" THEN A%(M)=4
    :Y=Y-IN:GOTO 600

510 IF A$="M" THEN A%(M)=6
    :Y=Y+IN:GOTO 600

520 IF A$="J" THEN A%(M)=7
    :X=X-IN:GOTO 600

530 IF A$="K" THEN A%(M)=5
    :X=X+IN:GOTO 600

540 IF A$=CHR$(9) THEN A%(M)=0
    :Y=Y-IN:GOTO 610

550 IF A$=CHR$(13) THEN A%(M)=2
    :Y=Y+IN:GOTO 610

560 IF A$=CHR$(10) THEN A%(M)=3
    :X=X-IN:GOTO 610

570 IF A$=CHR$(11) THEN A%(M)=1
    :X=X+IN:GOTO 610

580 IF A$="F" THEN 670
590 GOTO 480
600 HPLOT X1,Y1 TO X,Y:GOTO 630

610 IF IN=1 THEN 630
620 HPLOT X,Y
630 M=M+1
640 X1=X:Y1=Y

650 GOTO 460
660 :

670 REM Codification
680 :

690 FOR I=1 TO M STEP 2
700 C%(S)=A%(I)+A%(I+1)*8
    :S=S+1
710 NEXT
720 POKE AD,M:POKE AD+1,0

730 POKE AD+2*M,BL
    :POKE AD+2*M+1,BH
735 AB=AD+BB-1
740 FOR I=1 TO S
```

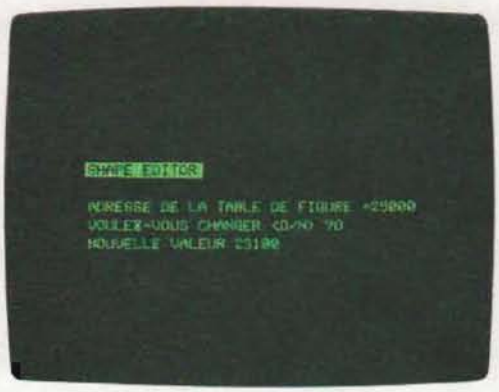
```
750 POKE AB+1,C%(I)
760 NEXT
770 POKE 232,AL:POKE 233,AH

780 HIMEM AD
790 :

800 REM Menu 2
810 :

820 HOME:VTAB 21

830 PRINT "Numéro de figure"
    :PRINT
840 PRINT "<1> J'annulle, <2>
     Autre, <3> Stop"
850 PRINT
860 PRINT "Choisir":GET Z$

870 Z=VAL(Z$)
    :IF Z<1 OR Z>3 THEN 800
880 ON Z GOTO 900,910,1000
890 GOTO 910
900 N=N-1:BB=BB-S

910 N=N+1:BB=BB+S

920 BH=INT(BB/256):BL=BB-BH*256)

930 FOR I=1 TO S:C%(I)=0:NEXT


940 FOR I=1 TO N:A%(I)=0:NEXT


950 TEXT:HOME:PRINT "Faire la
     figure numéro :";N .


960 VTAB 5:GOTO 230

970 :

980 REM Menu 1
990 :

1000 TEXT:HOME:VTAB 5

1010 FI$="E":IF N=1 THEN FI$="A"

1020 PRINT:PRINT "Vous avez dessiné "
     :N:" figures":FI$

1030 PRINT:PRINT "Adresse de la
      table :";AD

1040 BT=BB+S
1050 IF N=0 THEN BT=0
1060 PRINT "Longueur :";BT
1070 PRINT:PRINT

1080 PRINT "<1> Sauvegarder la table
      sur disque"
1090 PRINT "<2> Visualisation dans l'
     ordre"
1100 PRINT "<3> Annulation de la
      table"
 1110 PRINT "<4> Autre table"
 1120 PRINT "<5> Fin"
 1130 PRINT
 1140 PRINT "Choisissez :";GET Z$
 1150 Z=VAL(Z$)
```

```
1160 IF Z<1 OR Z>5 THEN 1000
1170 ON Z GOTO 1210,1260,1490,1190
     ,1100
1180 HOME:END

1190 CLEAR:GOTO 100

1200 :

1210 REM Sauvegarde sur disque
1220 :

1230 IF N=0 THEN HOME:VTAB 12:HTAB 10
     :FLASH:PRINT "Il n'y a pas de
      figure":NORMAL:GET Z$:GOTO 1000



1240 PRINT:INPUT "Sous quel nom ";A$

1250 PRINT CHR$(4)"BSAVE";A$",A";AD;
     ",L";BR+S
1260 GOTO 1000
1270 :

1280 REM Présentation de la table
1290 :

1300 FOR I=1 TO N
1310 RO=0:SC=1

1320 HGR:HCOLOR=3:ROT=RO:SCALE=SC



1330 HOME
1340 VTAB 22:PRINT "Figure n° "
     ;I;"Echelle = ";SC;"Rot = ";RO

1350 VTAB 23:PRINT "Pour changer: <S>
      Echelle, <R> Rot, <F> Figure,
      (avec <CTRL><TOUCHE>, on revient
      en arrière)";
1360 DRAW 1 AT 140,80
1370 GET A$
1380 IF A$="S" THEN SC=SC+1
     :IF SC>255 THEN SC=255
1390 IF A$="R" THEN RO=RO+1
     :IF RO>255 THEN RO=255
1400 IF A$=CHR$(18) THEN RO=RO-1
     :IF RO<0 THEN RO=0

1410 IF A$=CHR$(19) THEN SC=SC-1
     :IF SC<1 THEN SC=1

1420 IF A$="F" THEN 1440
1430 GOTO 1320
1440 NEXT
1450 GOTO 1000
1460 :

1470 REM Effacement de la table
1480 :

1490 PRINT:PRINT "Vraiment (O/N) ?"
     ;GET Z$


1500 IF Z$="O" THEN CLEAR:GOTO 100

1510 GOTO 1000
```

SHAPE EDITOR

Ces écrans apparaissent pendant le déroulement du programme Shape Editor.

Le programme commence par calculer l'adresse mémoire initiale à laquelle sera rangée la table créée.
L'utilisateur peut néanmoins la modifier.



Au départ, la table est vide et le programme présente le n° de la forme en cours de traitement.



Une fois entrés les paramètres (coordonnées initiales, facteurs d'agrandissement), le programme passe en mode graphique et présente le menu des commandes. Ici, le symbole d'un condensateur a été dessiné.



La figure est achevée et l'on peut passer à une autre. La table complète est ensuite enregistrée sur disque.



PRINCIPALES VARIABLES DU PROGRAMME SHAPE EDITOR

A%	: Tableau contenant les déplacerments codifiés
C%	: Tableau des couples de vecteurs graphiques codifiés
AD	: Adresse de départ (en mémoire) de la table des formes (partie basse = AL ; partie haute = AH)
BB	: Adresse de départ de la nouvelle forme (partie basse = BL ; partie haute = BH)
IN	: Facteur d'agrandissement
M	: Nombre de mouvements composant la figure
N	: Numéro de la figure
RO	: Indicateur de rotation d'une figure visualisée
SC	: Indicateur d'échelle d'une figure visualisée
X,Y X1,Y1	} Coordonnées graphiques

Le chargeur de formes. La mémorisation d'une table des figures sur disque n'a de sens que s'il existe un programme capable de la rappeler et de s'en servir.

La définition d'un certain nombre de figures graphiques, regroupées en une même table, permet, en effet, de disposer d'un menu de symboles qui peuvent être combinés de différentes façons pour former un dessin, même complexe.

Les performances du programme Shape Editor doivent donc être complétées par un programme capable d'effectuer les opérations de lecture d'une table de figures et de composition de ses éléments à l'écran, (voir organigramme 1 page suivante).

L'utilisateur fait son choix à partir d'un menu initial. En réponse, le programme passe la main à la partie du programme appelée. Le sous-programme 300, (organigramme 2, p. suiv.) charge en mémoire une table précédemment enregistrée sur disque. L'écran affiche alors le menu.

Avec le sous-programme 900 (organigramme 3, page 1625), on visualise une des figures de la table. Après ces opérations, le programme réaffiche le menu. Il est ensuite possible d'entrer dans la phase de composition d'une image exécutée avec l'option 3 (sous-programme 500 p. 1625). Le sous-programme 500 (organigramme 3 bis) sélectionne le mode graphique et demande le n° de la figure à afficher ainsi que le point de l'écran à partir duquel elle doit être visualisée.

Une fois ces paramètres entrés, l'image demandée est présentée et le programme passe au bloc 580 de saisie et d'exécution des commandes de composition des figures (voir tableau 4 de la page 1626).

Les fonctions décrites sont complétées par la possibilité de demander le répertoire (directory) du disque et par la sauvegarde de l'image composée.

Le programme de changement des formes est listé pages 1626 à 1630 (les variables utilisées sont résumées dans le tableau 4, page 1630).

Notons que les deux programmes présentés ont été développés sur un ordinateur Siprel. Ils ne fonctionneront sous cette forme que sur les ordinateurs personnels de la famille Apple et compatibles. Pour les transporter sur d'autres machines, il faut recourir aux règles de transposition désormais bien connues.

Instructions Basic de gestion des figures graphiques

Une fois mémorisée, une figure graphique peut, sur certaines machines, être traitée comme une donnée d'ensemble à l'aide d'instructions de haut niveau comme celles-ci.

DRAW	: Tracé de la figure
XDRAW	: Effacement
ROT	: Rotation
SCALE	: Changement de facteur d'échelle

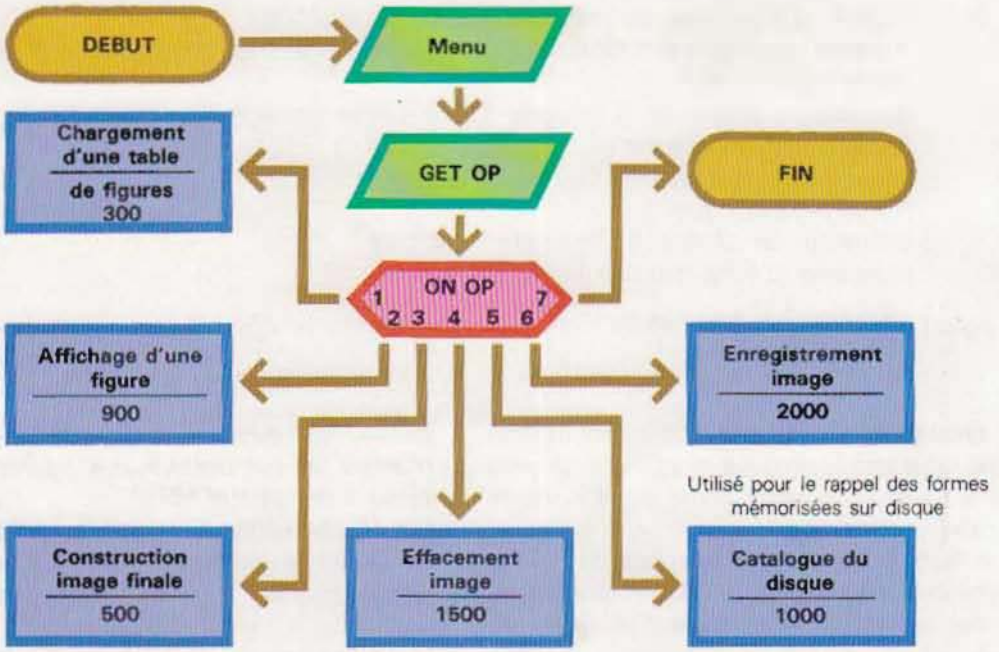
Dans ce qui suit, nous citerons les instructions utilisées par les systèmes Apple.

DRAW. La syntaxe de cette instruction est :

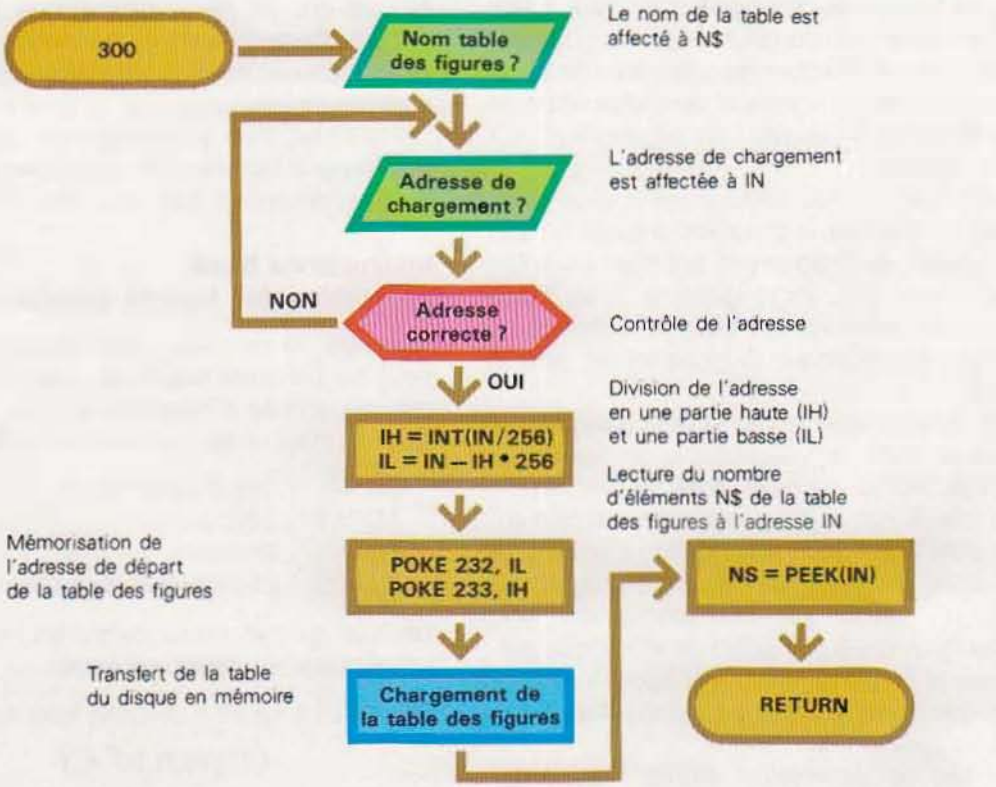
DRAW N AT X,Y

dans laquelle :

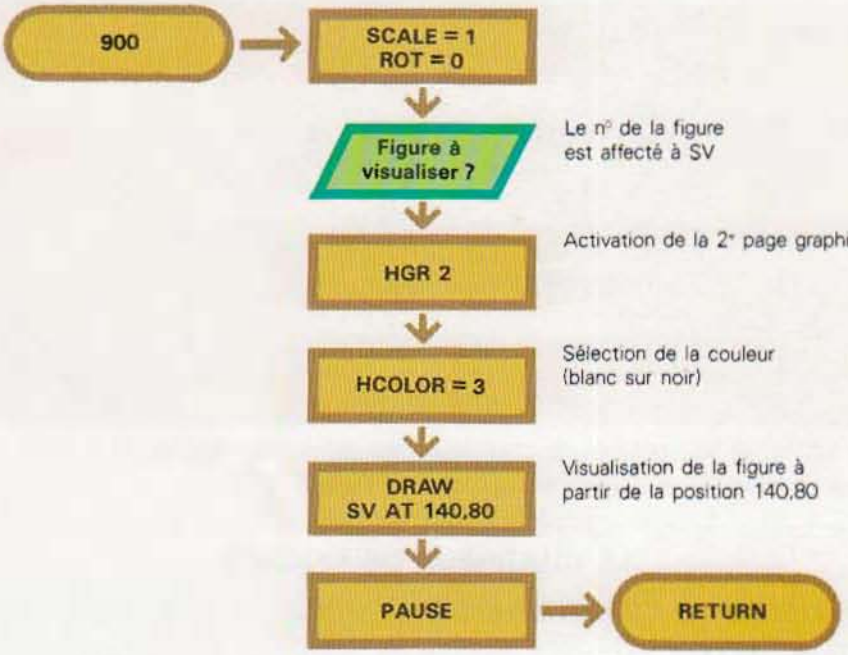
1/ SHAPE LOADER (CHARGEUR DE FORMES)



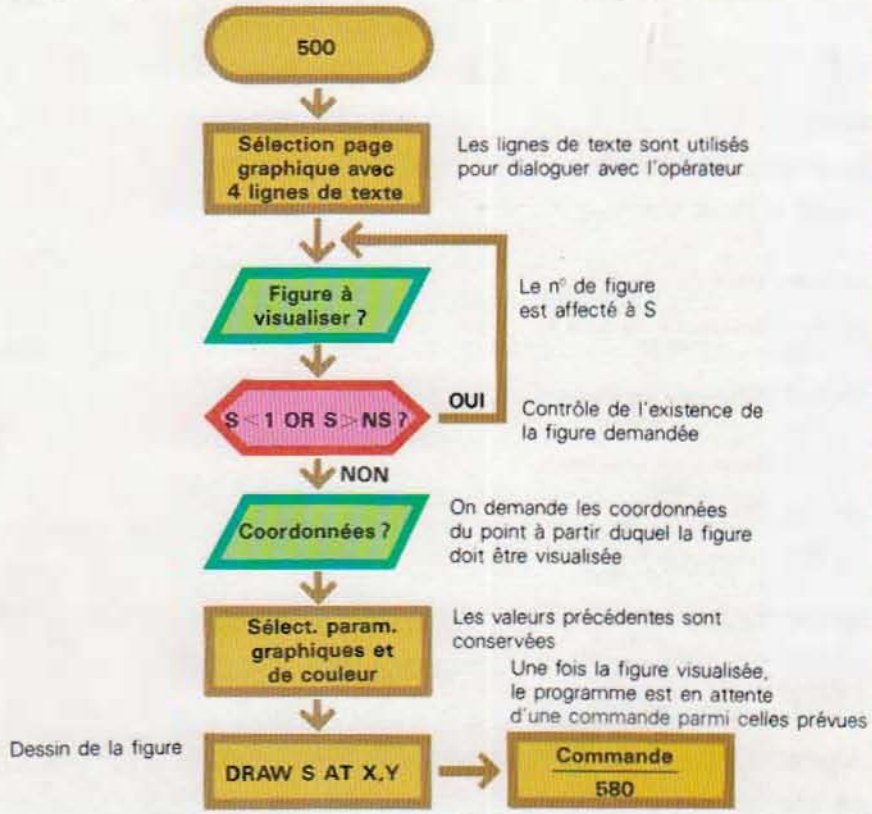
2/ SOUS-PROGRAMME DE CHARGEMENT D'UNE TABLE DE FIGURES



3/ SOUS-PROGRAMME DE VISUALISATION D'UNE FIGURE



3 bis/ Sous-programme de construction de l'image finale



4/ COMMANDES DU SOUS-PROGRAMME 580

Commandes	Signification
/	Effacement de la fenêtre texte
?	Visualisation de la fenêtre texte
I	Déplacement de la figure d'un pas vers le haut
J	Déplacement de la figure d'un pas à gauche
K	Déplacement de la figure d'un pas à droite
M	Déplacement de la figure d'un pas vers le bas
R	Rotation de la figure de 90° à droite
S	Agrandissement de la figure
CTRL + R	Rotation de la figure de 90° à gauche
CTRL + S	Réduction de la figure
C	Effacement
ESC	Retour au programme principal

Le sous-programme 580 permet de composer les différentes figures de la table chargée en mémoire jusqu'à ce qu'un dessin complexe soit formé.

LE CHARGEUR DE FIGURES

```
10 REM ************************
20 REM *                      *
30 REM * CHARGEUR DE FIGURES *
40 REM *                      *
50 REM ************************
60 :

70 REM MENU
80 :

90 TEXT:HOME

100 HI=PEEK(115)+PEEK(116)*256

110 HTAB 18:INVERSE:PRINT "Menu"

120 NORMAL:VTAB 6

130 PRINT "Chargement d'une table de
    figures ... 1":PRINT

140 PRINT "Affichage d'une figure
    ............... 2":PRINT

150 PRINT "Construction image finale
    ............ 3":PRINT
160 PRINT "Effacement image
    ....................... 4":PRINT
170 PRINT "Catalogue du disque
    ................... 5":PRINT

180 PRINT "Enregistrement image
    ................... 6":PRINT

190 PRINT "Fin ..............

    ...................... 7":PRINT
200 VTAB21

210 INPUT "Quelle option choisissez-
    vous ";OP$
```

```
220 OP=VAL(OP$):IF OP<1 OR OP>7
    THEN 90
230 ON OP GOSUB 300,900,500,1500,1000
    ,2000
240 IF OP<>7 THEN 90
250 HOME:END

300 :

310 REM Chargement d'une table de
    figures
320 :
330 TEXT:HOME

340 INPUT "Nom de la table de figures
    ?";n$
345 IF ASC(LEFT$(N$,1))<65 OR
    ASC(LEFT$(N$,1))>90 THEN 330
350 PRINT:PRINT

360 PRINT "Adresse de chargement :"
    :print "(24576-";HI;")" 370 VTAB
    4:HTAB 26:PRINT SPC(14)



380 VTAB 4:HTAB 26:INPUT " ":IM$


390 IN=VAL(IN$)
400 IF IN<24576 OR IN>HI THEN 370

410 IH=INT(IN/256):IL=IN-IH*256

420 POKE 232,IL:POKE 233,IH

425 VTAB 16:INVERSE:PRINT "Chargez un
    disque et tapez <RETURN>":NORMAL
    :GET R$



427 PRINT CHR$(4)
430 PRINT CHR$(4):"BLOAD";N$;", A";IN

435 MS=PEEK(IN)
440 RETURN
500 :

505 REM Construction de l'image
510 :

515 IF MS=0 THEN HOME:VTAB 12:FLASH
    :PRINT "Il n'y a pas de table en
    mémoire":NORMAL:GET A$:RETURN



520 POKE -16297,0:POKE -16300,0
    :POKE -16301,0:POKE -16304,0
    :POKE 230,32

525 HOME:VTAB 21

530 PRINT "Figure à visualiser (1 - "
    ;NS;")":INPUT " ?";S$
535 IF S$="" THEN RETURN

540 S=VAL(S$)
545 IF S<1 OR S>NS THEN 525
```

```
546 VTAB23:INPUT "Coordonnées (X,Y) "
    ;X$,Y$:X=VAL(X$):Y=VAL(Y$)


547 IF X<0 OR X>279 OR Y<0 OR Y>191
    THEN 546
550 RO=0:SC=1

555 X1=X:Y1=Y=R1=RO:S1=SC


557 HCOLOR=3
560 SCALE=SC:TOR=RO

565 DRAW S AT X,Y
570 HOME:VTAB 23

575 INVERSE:PRINT "Commandes: I, J, K,
     M R, ^R S, ^S /, ? C ESC";
    :NORMAL

580 HTAB 1:VTAB 21:PRINT SPC(79)


585 HTAB 1:VTAB 21

590 PRINT "Figure = ";:INVERSE:PRINT S
    :NORMAL


595 PRINT "Echelle = ";:INVERSE:PRINT
     SC;:NORMAL


600 PRINT "Rot = ";:INVERSE:PRINT RO;
    :NORMAL


605 PRINT "X = ";:INVERSE:PRINT X
    :NORMAL


610 PRINT "Y = ";:INVERSE:PRINT Y;
    :NORMAL


615 VTAB 1:HTAB 1:GET T$


620 IF T$="/" THEN POKE -16302,0
625 IF T$="?" THEN POKE -16301,0
630 IF T$="I" THEN Y=Y-1
    :IF Y<0 THEN Y=191
635 IF T$="J" THEN X=X-1
    :IF X<0 THEN X=279
640 IF T$="K" THEN X=X+1
    :IF X>279 THEN X=0
645 IF T$="M" THEN Y=Y+1
    :IF Y>191 THEN Y=0
650 IF T$="R" THEN RO=RO+1
    :IF RO>255 THEN RO=255
655 IF T$=CHR$(18) THEN RO=RO-1
    :IF RO<0 THEN RO=0

660 IF T$="S" THEN SC=SC+1
    :IF SC>255 THEN SC=255
665 IF T$="C" THEN GOSUB 1000
670 IF T$=CHR$(19) THEN SC=SC-1
    :IF SC<1 THEN SC=1
```

```
675 IF T$=CHR$(27) THEN RETURN

677 HCOLOR=0

680 SCALE=S1:ROT=R1

685 DRAW S AT X1,Y1

687 HCOLOR=3

690 SCALE=SC:ROT=RO:DRAW S AT X,Y

695 S1=SC:R1=RO:X1=X:Y1=Y

700 GOTO 580

900 :

905 REM Affichage d'une figure

910 :
915 IF N$=0 THEN HOME:VTAB 12:FLASH
    :PRINT "Il n'y a pas de table en
    mémoire":NORMAL:GET A$:RETURN

920 TEXT:HOME

925 SCALE=1:ROT=0

930 VTAB 12:HTAB 1

935 PRINT "Figure à visualiser (1 - "
    ;n$;") ";INPUT SV$:SV=VAL(SV$)

940 IF SV<1 OR SV>NS THEN 920

945 HGR2:HCOLOR=3

950 DRAW SV AT 140,80

955 GET R$

960 RETURN

1000 :
1010 REM Catalogue

1020 :
1030 HOME:PRINT CHR$(4);"Catalogue"

1040 GET A$:RETURN
1075 PRINT CHR$(4)

1500 :
1510 REM Effacement

1520 :
1530 HOME:VTAB 12:INPUT "Certain (O/N)
     ?";R$
1540 IF R$<>"O" AND R$<>"N" THEN 1530

1550 IF R$="O" THEN HGR
1560 RETURN

1800 :
1810 REM Mode effacement

1820 :
1830 X0=X:Y0=Y:X=140:Y=80
```

```
1835 VTAB 23:HTAB 36:PRINT "C"

1840 HCOLOR=3:HPLOT X,Y

1850 VTAB 1:HTAB 1:GET A$

1855 HCOLOR=0:HPLOT X,Y

1860 IF A$="I" THEN Y=Y-1
     :IF Y<0 THEN Y=191
1870 IF A$="M" THEN Y=Y+1
     :IF Y>191 THEN Y=0
1880 IF A$="J" THEN X=X-1
     :IF X<0 THEN X=279
1890 IF A$="K" THEN X=X+1
     :IF X>279 THEN X=0
1900 IF A$="C" THEN 1920
1910 GOTO 1840
1920 HCOLOR=3:X=X0:Y=Y0

1930 VTAB 23:INVERSE

1940 PRINT "Commandes: I, J, K, M R^R
      S, ^S /, ? C ESC";:NORMAL

1950 RETURN
2000 :

2010 REM Sauvegarde de l'image
2020 :

2030 HOME:VTAB 12

2040 INPUT "Sous quel nom ";NM$
2050 IF ASC(LEFT$(NM$,1))<65 OR
     ASC(LEFT$(NM$,1))>90 THEN 2030

2060 VTAB 14:PRINT "C'est bon !";NM$
     ", PTC"
2070 VTAB 16:INVERSE:PRINT "Chargez un
     disque et tapez <RETURN>":NORMAL
     :GET R$


2080 PRINT CHR$(4)
2090 PRINT CHR$(4);"BSAVE";NM$;", PTC
     , A$2000,L$1FFF"
2100 PRINT CHR$(4)
2110 RETURN
```

VARIABLES EMPLOYEES DANS LE PROGRAMME SHAPE LOADER

HI	= Limite de la mémoire réservée au graphique
IN	= Adresse initiale de la table des figures
N$,NM$	= Nom de la table des figures
NS	= Numéro de la figure dans la table
R1,RO	= Valeurs des ROT (rotations)
S$	= Figure en cours
S1,SC	= Valeurs de SCALE (échelle)
X,Y,XO,YO,X1,Y1	= Coordonnées à l'écran

EXEMPLE D'APPLICATION DU PROGRAMME SHAPE LOADER

Cette série d'écrans illustre une application de composition d'une image utilisant des formes élémentaires (shapes) créées et mémorisées à l'aide du programme Shape Editor

Le programme Shape Loader affiche le menu. L'utilisateur demande le chargement à partir du disque d'une table de figures contenant des symboles électriques



Le sous-programme de composition de l'image finale a été désigné et est exécuté. On peut voir, en bas, la ligne de dialogue et le menu des commandes ; au centre de l'écran, se trouve le dessin en cours de formation. L'élément de gauche est la figure qui va être positionnée.



La construction de la figure composée est terminée. L'utilisateur devra la mémoriser sur disque en lui donnant un nom particulier pour pouvoir la rappeler en cas de besoin



N : C'est le n° de la figure à visualiser
X,Y : Sont les coordonnées de départ de l'affichage de la figure.

Par exemple, l'instruction :

DRAW 2 AT 30,50

présente la figure n° 2 en commençant le dessin à partir du point de coordonnées X = 30 et Y = 50. Si les coordonnées sont omises, la figure commence au dernier point rappelé. Cela signifie que cette instruction peut également se présenter sous la forme DRAW N, qui lance l'affichage de la figure N à partir des dernières coordonnées utilisées dans une précédente instruction DRAW ou HPLOT (pour le positionnement).

XDRAW. Sa forme est identique à celle de l'instruction précédente :

XDRAW N AT X,Y

Elle commande l'effacement de la figure n° N, positionnée au point de coordonnées X,Y.

ROT. La syntaxe est très simple :

ROT M

où M est une valeur numérique exprimant l'angle de rotation, généralement en degrés.
La validité du paramètre M dépend de la valeur définie par l'instruction précédente SCALE. Si le facteur d'échelle est 1, le paramètre M aura une des valeurs suivantes : 0 = 0°, 16 = 90°, 32 = 180°, 48 = 270°.
En revanche, si ce facteur d'échelle est supérieur à 4, il est possible d'avoir 64 rotations différentes à des intervalles réguliers par rapport aux valeurs précédentes (ces valeurs ne sont qu'un exemple lié au matériel considéré et ne constituent donc pas une règle).

SCALE. Cette instruction doit être utilisée en priorité, avant toute autre instruction de ce groupe, puisqu'elle définit l'échelle du dessin.
Sa syntaxe est :

SCALE = Q

dans laquelle le paramètre numérique Q indique le nombre de tracés pour chaque vecteur graphique. Par exemple : SCALE = 1 provoque l'affichage d'une figure dans laquelle chaque vecteur n'est tracé qu'une fois, alors qu'avec SCALE = 2, il le sera deux fois.
Sur certaines machines, il est possible de préparer un dessin composé de déplacements élémentaires à l'aide d'une chaîne de commandes. Le mot-clé reste généralement DRAW, mais sa syntaxe et la signification de ses éléments sont différents, bien que proches.
Cette instruction, peut prendre deux formes :

— Dans la première, les commandes de déplacement sont contenues dans une chaîne associée à l'instruction.

— Dans la seconde, elles sont mémorisées dans une table résidente en mémoire ; l'instruction n'admet alors comme paramètre que le n° correspondant à la figure demandée :

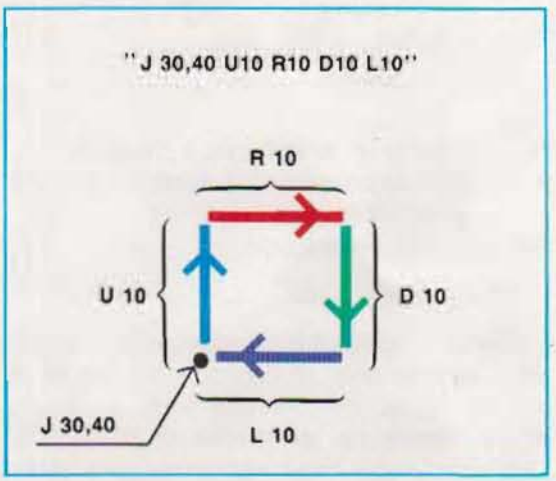
DRAW : « chaîne de commandes »
DRAW : N avec N = n° de la figure à visualiser.

Dans la chaîne de commandes, il est nécessaire d'expliquer certains symboles qui représentent des actions élémentaires à exécuter.

Ces paramètres sont :

M DX,DY : Positionnement du curseur au point de coordonnées X = DX et Y + DY en partant du point X,Y.
J,X,Y : Déplacement au point X,Y
U DY : Déplacement de DY vers le haut
L DX : Déplacement de DX vers la gauche.
R DX : Déplacement de DX à droite
C : Indication de la couleur du tracé de la figure ; s'il est omis, le système prend par défaut la dernière couleur utlisée dans une instruction DRAW.

Les paramètres ci-dessus correspondent au système Olivetti M20 ; ils peuvent être différents pour les autres machines, mais si cette forme de l'instruction DRAW existe, la logique reste la même.
La figure ci-dessous montre, à titre d'illustration, la chaîne des commandes nécessaires au tracé d'un carré de côté 10 positionné au point 30,40 (sommet inférieur gauche). La chaîne adresse = 30 et Y = 40 (code J) indique les quatre déplacements nécessaires pour obtenir les côtés.
Telle qu'elle est présentée, l'instruction trace également une ligne entre la position d'origine du curseur et le premier sommet du carré puisque la commande J 30,40 est exécutée en mode visible (dans la couleur spécifiée ou la dernière utilisée). Pour éviter que ce segment (non désiré) soit visualisé, il faut précéder la commande de la lettre B, qui invalide l'affichage, la forme exacte de la chaîne est donc :
"BJ30, 40 U10 R10 D10 L10".



On peut utiliser une des options suivantes avec chacune des commandes :

AND : La couleur résultante est obtenue par intersection logique de 2 couleurs, la précédente et celle spécifiée dans la chaîne

XOR, OR : Même principe, mais avec une union logique (OU exclusif ou inclusif)

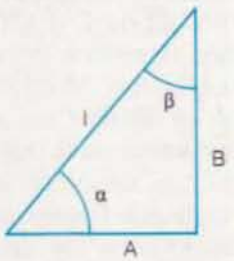
NOT : Indique que la couleur de la figure est le complément de celle utilisée comme fond.

Graphisme tridimensionnel (3D)

La préparation d'un graphique en trois dimensions (3D) exige le recours à des formules mathématiques parfois complexes, qui seront développées plus loin. Les plus employées sont celles de la trigonométrie appliquée aux problèmes de triangles rectangles.

La figure ci-dessous indique, dans le tableau central, comment calculer les éléments inconnus d'un triangle rectangle dans les différents cas de figure.

RESOLUTION DES TRIANGLES RECTANGLES



$B = I * \sin(\alpha)$
$A = I * \cos(\alpha)$

$\frac{B}{A} = \frac{\sin(\alpha)}{\cos(\alpha)} = tg(\alpha)$

Eléments connus*	Calcul des autres
I, α	Appliquer les formules suivantes
A, B	$\alpha = \text{cotg}\ (B/A)$ $I = A/\cos(\alpha) = B/\sin(\alpha)$
A, α	$I = A/\cos(\alpha)$ $B = I.\sin(\alpha) = A.\frac{\sin(\alpha)}{\cos(\alpha)} = A.tg(\alpha)$
B, α	$I = B/\sin(\alpha)$ $A = I.\cos(\ ) = B.\frac{\cos(\alpha)}{\sin(\alpha)} = \frac{B}{tg(\alpha)}$
I, A	$\cos(\alpha) = A/I$ $\sin(\alpha) = \sqrt{1 - \cos^2(\alpha)}$ $B = I.\sin(\alpha)$
I, B	$\sin(\alpha) = B/I$ $\cos(\alpha) = \sqrt{1 - \sin^2(\alpha)}$ $A = I.\cos(\alpha)$

Toutes les formules peuvent être traduites en langage symbolique pour être utilisées dans un programme.
Ne pas oublier que :
— l'angle α doit être en radians
— les noms de fonctions sont généralement :

```
sin(α)=SIN(ALPHA)
cos(α)=COS(ALPHA)
tg(α)=SIN(ALPHA)/COS(ALPHA)

√ =SQR(..)
COTG (B/A) =ATN(B/A)
```

*Le cas où c'est l'angle β qui est connu, au lieu de l'angle α, n'est pas pris en considération. En effet, étant donné que $\alpha = 90 - \beta$, ce cas peut être résolu à l'aide d'une des figures exposées.

Les fonctions trigonométriques sont (notation mathématique) :

sinus=sin(α)
cosinus=cos(α)
tangente=tg(α)
cotangente=cotg(α)

Pour obtenir les côtés de l'angle droit, à partir de l'hypoténuse I et de l'angle α, il suffit d'appliquer les formules :

A=I*cos(α)
B=I*sin(α)

Les principales fonctions trigonométriques ont été prévues dans les langages de programmation comme le Fortran et le Basic.
Les instructions nécessaires pour effectuer les calculs précédents sont :

A=I*COS(ALPHA)
B=I*SIN(ALPHA)

à condition de convertir l'angle en radians, selon l'expression :

$$\text{angle en radians} = \frac{\pi * \text{angle en degrés}}{180} = \frac{3{,}14 * \text{angle en degrés}}{180}$$

Certains langages ne comportant pas la fonction tg(α), il faut la calculer à l'aide de l'expression :

TG(ALPHA)=SIN(ALPHA)/COS(ALPHA)

Dans le programme, la fonction tangente peut être définie par l'utilisateur (au moyen de l'instruction DEF...) et assimilée à une fonction particulière.
La résolution des triangles rectangles est plus difficile, quand les seuls éléments connus sont les côtés, à l'exclusion de tout angle. On peut alors recourir au théorème de Pythagore, mais aussi aux fonctions trigonométriques inversées. Par exemple, si l'on connaît les deux côtés de l'angle droit, on peut calculer l'angle α en utilisant la fonction cotangente (ATN) :

ALPHA=ATN(B/A)

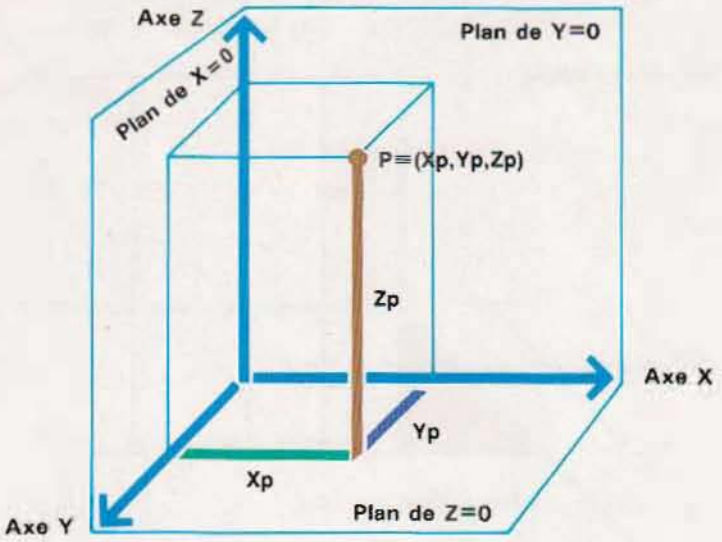
A et B étant les côtés de l'angle droit.

La fonction cotangente fait généralement partie du langage de programmation.

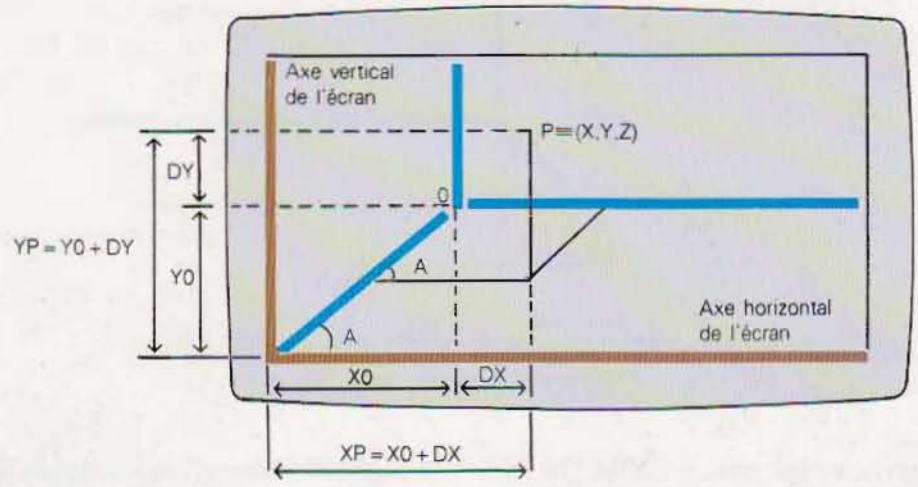
Les fonctions à deux variables

Jusque-là, les fonctions rencontrées se présentaient sous la forme Y=f(X) où Y est fonction d'une seule variable indépendante, X. Les fonctions à une variable sont représentables sur un plan cartésien, puisque chaque point est, en effet, repéré par un couple de valeurs (coordonnées). Les courbes qu'elles représentent sont facilement visualisables en faisant coïncider les axes de référence avec ceux de l'écran.
Mais il existe des fonctions à plusieurs variables et les plus fréquemment utilisées comportent deux variables indépendantes, du type Y=f(X,Z) ou Z=f(X,Y).
Nous reprenons là les symboles habituels : f désigne une fonction générique, c'est-à-dire une loi unissant les valeurs de la variable dépendante à celles des variables indépendantes. Le choix de la forme, ici Y=f(X,Z), est plus proche des formes déjà connues.
Une fonction de ce type ne peut pas être visualisée sur un plan, puisqu'il faut reporter les valeurs d'une 3e coordonnée (Z). Le système de référence adopté doit donc comporter trois axes. La figure 1 (page ci-contre) montre un système de référence à 3 axes X, Y et Z. Chacun des points de l'espace est défini par leurs coordonnées relatives.
De toute évidence, une représentation à deux variables laisse de côté la possibilité d'un 3e axe sur l'écran, pour, par exemple, figurer un objet tridimensionnel sur un plan (feuille de dessin). Pour corriger cette lacune, on a donc imaginé de simuler la profondeur et, pour cela, construire des algorithmes analogues à ceux employés par un dessinateur dans la réalisation d'une perspective ou d'une figure axonométrique.
Les méthodes exposées ici ne sont pas de véritables applications de graphisme tridimensionnel. Il s'agit plutôt de simulations s'adaptant à un grand nombre d'utilisations mais qui sont toutefois limitées par l'environnement matériel.
Pour réaliser un véritable logiciel graphique 3D, il faut disposer de machines très puissantes.

1/SYSTEME DE REFERENCE DANS L'ESPACE



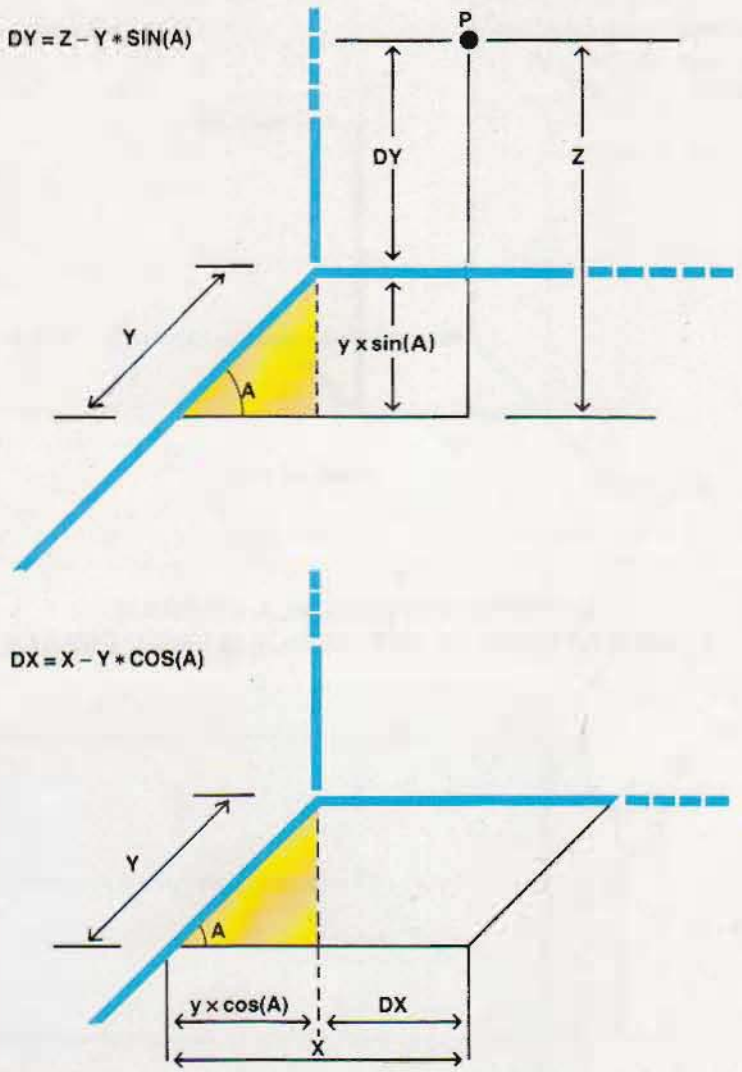
2/REPRESENTATION A L'ECRAN D'UN SYSTEME DE REFERENCE DANS L'ESPACE



Représentation 3D. Le procédé utilisé, pour représenter à l'écran un point quelconque P de l'espace, est schématisé par la figure 2.
L'origine du système de référence, désigné par la lettre O, est positionnée sur un point de l'écran. Pour des raisons de simplicité, nous considérerons que l'origine des axes de l'écran se trouve en bas à gauche (dans certains systèmes, elle est en haut à gauche).
Par rapport à l'écran (du point de vue strictement graphique), le point P (défini par les coordonnées X, Y, Z) a pour coordonnées XP et YP.
Ces coordonnées s'obtiennent en ajoutant respectivement DX et DY aux coordonnées de l'origine des axes.
On a donc :

XP=XO+DX
YP=YO+DY

CALCUL DES COORDONNEES A L'ECRAN



Une fois connues les valeurs DX et DY, on calcule les coordonnées écran de tous les points de la fonction 3D. En dernière analyse, le problème se réduit donc au calcul des quantités DX et DY.
La première phase consiste à calculer l'angle A en utilisant les formules trigonométriques précédentes (pour des raisons pratiques, l'angle est désigné par la seule lettre A, alors qu'il est appelé ALPHA dans les formules). La valeur de A est déterminée par les coordonnées X0, Y0. Celles-ci constituent les côtés de l'angle droit d'un triangle rectangle ayant pour hypoténuse le segment unissant l'origine (écran) au point 0. Le résultat est: A=ATN(Y0/X0).
Une fois cet angle déterminé, on procède au calcul de DX et DY, toujours à l'aide des règles relatives aux triangles rectangles.
De cette façon, on obtient :

DY=Z-Y*SIN(A)
DX=X-Y*COS(A)

Pour comprendre les formules précédentes, il faut observer les figures ci-dessus, sans prendre en compte l'aspect 3D du système de référence.

Voyons, maintenant, comment tracer une fonction tridimensionnelle :

1/Affectation des valeurs arbitraires aux variables indépendantes X et Z
2/Calcul de la valeur correspondante de Y en utilisant l'expression mathématique de la fonction
3/Calcul des coordonnées XP et YP en points écrans à l'aide des formules :

```
YP=XO+DX
YP=YO+DY
```

ou avec les formules équivalentes :

```
XP=XO+(X-Y*COS(A))
YP=YO+(Z-Y*SIN(A))
```

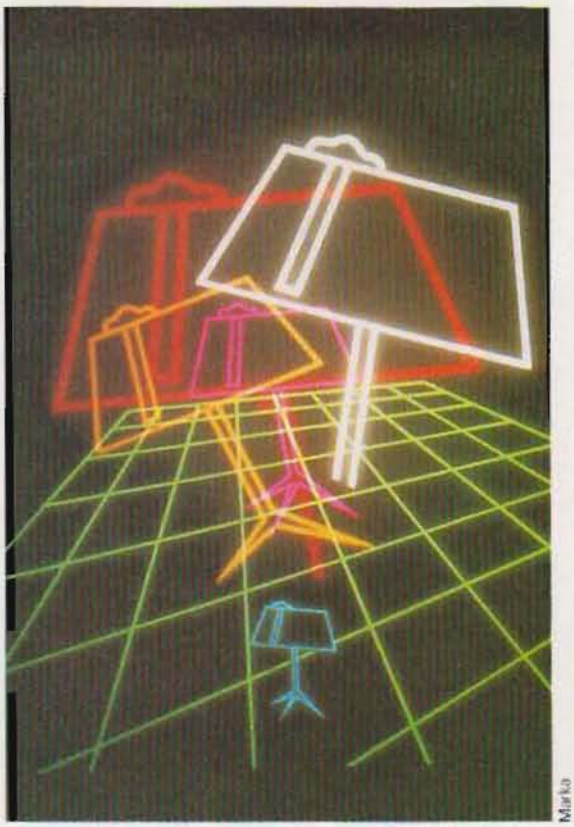
dans lesquelles DX et DY ont été remplacés par :

```
DX=X-Y*COS(A)
DY=Z-Y*SIN(A)
```

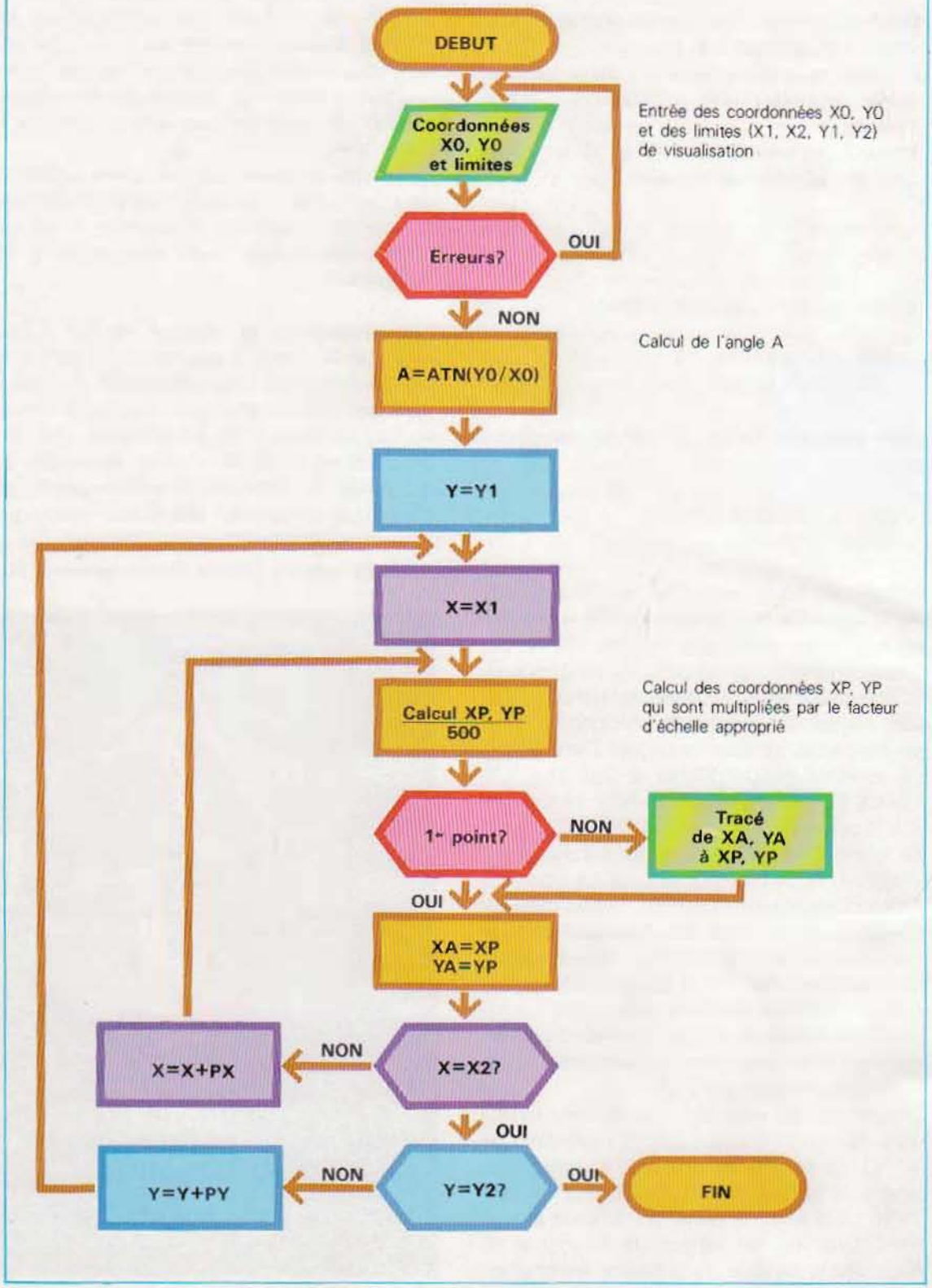
Notons que les coordonnées XP, YP en points écran n'ont rien de commun avec les coordonnées X, Y des points des courbes. Par conséquent, si la fonction est de la forme Z=f(X,Y), et non plus Y=f(X,Z), toutes les formules restent valides, à condition de changer les coordonnées utilisées dans les calculs. Dans ce cas, les variables indépendantes ne sont plus X,Z, mais X,Y ; la fonction permet donc de calculer Z à la place de Y.
La forme Z=f(X,Y) est la plus courante. Pour simplifier, nous avons omis tous les contrôles et les changements d'échelle utilisés dans les programmes de tracé des fonctions dans le plan. La bouche externe est construite à partir de l'incrémentation de Y (considérée comme coordonnée de la fonction), et la boucle interne avec des valeurs de X. Cela équivaut à présenter le tracé de la fonction par sections successives, parallèles au plan XZ.
Supposons, par exemple, que la valeur initiale de la coordonnée Y soit zéro, c'est-à-dire que Y1=0. La bouche interne (sur X) montrerait alors tous les points de la courbe pour lesquels Y=0, c'est-à-dire la partie située dans le plan Y=0. Une fois les valeurs de X pour Y=0 épuisées, la variable de la boucle externe est incrémentée. Y passe donc à une nouvelle valeur, par exemple 1. Dans ce cas, la boucle interne présenterait la courbe de la fonction dans le plan Y=1 (parallèle au précédent). En continuant ainsi à incrémenter Y de 1, on obtient le graphique 3D de la fonction, par superposition d'autant de graphiques plans dont chacun montre la courbe dans un plan particulier (Y=0, Y=1, Y=2...).
Ces représentations sur des plans parallèles sont des sections de la figure solide représentée par la courbe. Le mécanisme décrit est schématisé en page 1640 suivi du listing du programme.

Les problèmes du dessin en 3D. Il faut distinguer le véritable graphisme 3D de la simple représentation tridimensionnelle. En effet il y a là une différence considérable qui se retrouve tant au niveau des équipements, que des programmes et de leurs coûts respectifs. Le graphisme 3D demande la mémorisation des attributs de chacun des points de l'espace que

Image obtenue à l'aide d'un programme 3D.

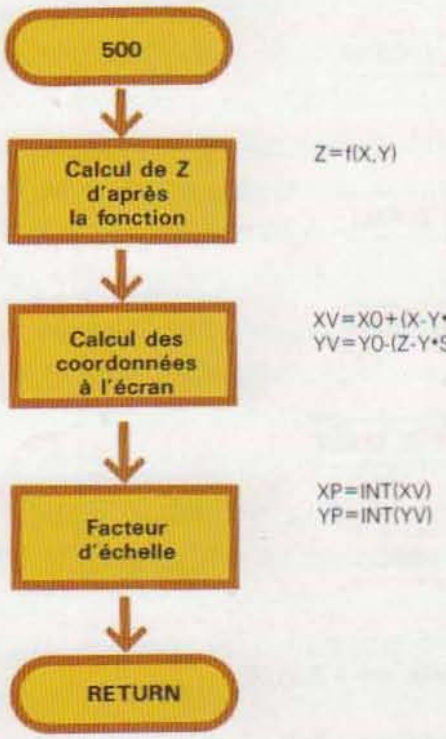


VISUALISATION D'UNE FONCTION A 2 VARIABLES

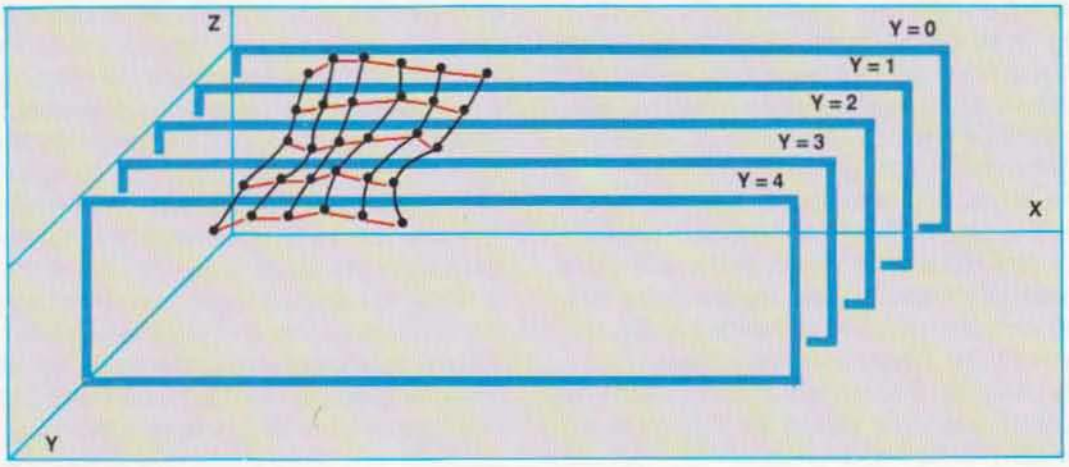


constitue le solide à représenter. On peut alors obtenir des projections, des vues en coupe ou tout autre traitement graphique d'un objet.
Mais un tel logiciel ne peut fonctionner que sur des machines très sophistiquées, notamment en terme de capacité mémoire et en résolution du terminal vidéo ou des autres périphériques (traceur de courbes par exemple).
Dans le cas de la représentation 3D, en revanche, le problème est nettement plus simple et peut être réduit à un simple traitement d'une vue de l'objet donnant une impression de tridimensionnalité, ce qui ne demande pas des machines et des programmes complexes.
Bien entendu, il n'est nullement question de visualiser une vraie coupe, en l'absence des attributs relatifs à la 3e dimension.
Dans la méthode utilisée pour la représentation 3D (voir graphique p. 1642), la 3e dimension est obtenue par translation de la figure de base (vue frontale de l'objet) qui est inclinée de 45° (de ce fait, les déplacements de chaque point sont égaux sur les deux axes). Les sommets de la figure de base étant numérotés de 1 à 6, l'impression d'épaisseur est obtenue en traçant la même figure après avoir incrémenté les coordonnées des sommets de la quantité S ; une fois que le contour déplacé est réalisé, il faut unir les points homologues. Il reste toutefois à résoudre le problème de l'effacement (ou mieux du non tracé) des segments de la vue déplacée qui sont couverts par la figure de base. Nous touchons là à un des problèmes les plus complexes du graphisme 3D : comment déterminer les lignes à éliminer. En effet, les surfaces qu'elles circonscrivent sont cachées par d'autres parties de l'objet représenté.
Il existe de nombreux algorithmes de résolution de cette question, dont la plupart sont utilisables dans le tracé de fonctions. Mais il s'agit là généralement de solutions très spécialisées et difficilement généralisables aux dessins dont les lignes ne sont pas des fonctions mathématiques.
Autrement dit, il faut que le logiciel soit capable de trouver les lignes cachées et d'éviter qu'elles ne soient tracées.
La méthode utilisée dans le programme que

SOUS-PROGRAMME DE CALCUL DES COORDONNEES A L'ECRAN



SCHEMA DE LA METHODE DE REPRESENTATION



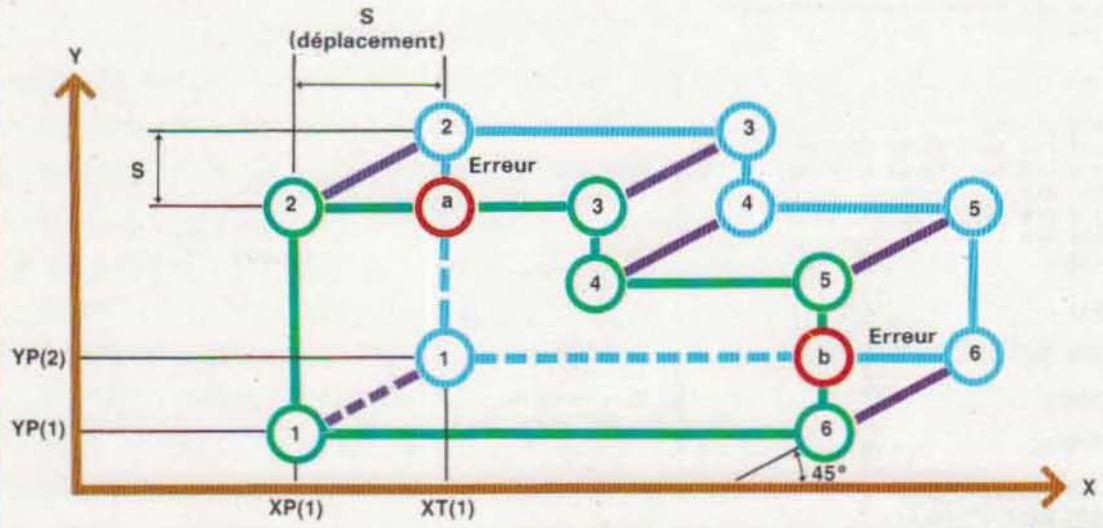
GRAPHIQUE D'UNE FONCTION A DEUX VARIABLES

```
10 REM ******************************
20 REM *                            *
30 REM *                            *
40 REM * GRAPHIQUES A 3 DIMENSIONS *
50 REM *                            *
60 REM *                            *
70 REM ******************************
80 :

90 :

100 REM ------------------
110 REM LIMITES DE L'ECRAN
120 REM ------------------
130 XN=0
      :XM=279
      :YN=0
      :YM=191
140 REM ------------------
150 REM FACTEURS D'ECHELLE
160 REM ------------------
170 FX=5
      :FY=10
180 REM ------------------
190 REM DENSITE DES POINTS
200 REM ------------------
210 PX=.2
      :PY=.2
220 REM --------------------
230 REM COORDONNEES DU CENTRE
240 REM --------------------
250 X0=129:
      Y0=61
270 REM ------------------
280 REM ENTREE DES DONNEES
290 REM ------------------
300 TEXT
      :HOME
310 VTAB 12
320 INPUT "Intervalle axe X ";X1,X2

330 VTAB 14
340 INPUT "Intervalle axe Y ";Y1,Y2
```

```
350 A=ATN(Y0/X0)
      :REM ANGLE
360 HGR2
      :HCOLOR=3
370 GOSUB 2000
380 :

390 :

400 REM ************************
410 REM **                    **
420 REM ** BOUCLE PRINCIPALE **
430 REM **                    **
440 REM ************************
450 :

460 :

470 FLAG=1
480 FOR Y=Y1 TO Y2 STEP PY
490 FOR X=X1 TO X2 STEP PX
500 REM ----------------
510 REM CALCUL DE XP, YP
520 REM ----------------
530 :

540 :

550 Z=(SIN(X)/X)*(5*SIN(Y))

560 :

570 :

580 DX=X-Y*COS(A)
590 DY=Z-Y*SIN(A)
600 XV=X0+DX*FX
610 YV=Y0-DY*FY
620 XP=INT(XV)
630 YP=INT(YV)
640 REM ------------------
650 REM TRACE DU GRAPHIQUE
660 REM ------------------
670 IF FLAG THEN FLAG=0
      :GOTO 690
680 HPLOT XA,YA TO XP,YP
690 XA=XP:
      YA=YP
700 NEXT X
710 FLAG=1
720 NEXT Y
730 :

740 :

750 REM **********************
760 REM ** FIN DE LA BOUCLE**
770 REM **********************
780 :

790 :

800 GET D$
810 TEXT
     :HOME
820 END
2000 REM ---------------
2010 REM AXES CARTESIENS
2020 REM ---------------
2030 HPLOT XN,YN TO XM,YN TO XM,YM
      TO XN,YM TO XN,YN
```

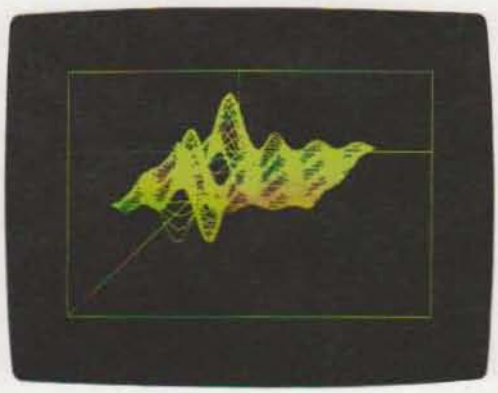
```
2040 HPLOT X0,Y0 TO X0,YN
2050 HPLOT XN,Y0 TO XM,Y0
2060 HPLOT X0,Y0 TO XN,YM
2065 HCOLOR=0
2070 FOR I=X0-3 TO 0 STEP -3
2080 HPLOT I,Y0
2090 NEXT
2100 HCOLOR=3
2110 RETURN
```

METHODE EMPLOYEE POUR LA REPRESENTATION 3D



Les segments affectés de la mention Erreur sont tracés parce que le programme ne dispose pas de toutes les fonctions nécessaires.

- Contour de référence (c'est le premier à être tracé)
- Contour déplacé
- Lignes de liaison des deux contours (épaisseur)
- Sommets de la base
- Sommets de la base déplacés
- Points d'intersection



Résultat final de la visualisation d'une fonction à 2 variables.

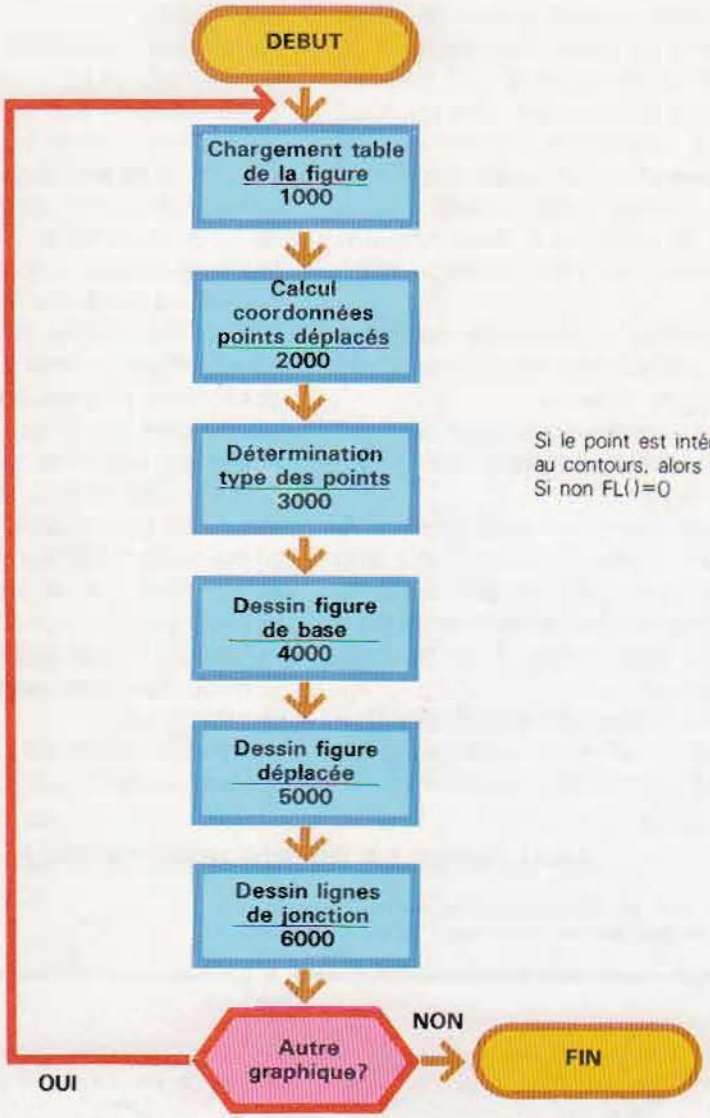
nous présentons (voir organigramme page ci-contre) est fondée sur la seule analyse de la figure de base, et elle peut, dans certains cas, aboutir à des résultats incorrects.

C'est pourquoi cette méthode ne doit être considérée que comme un exemple d'approche de ce problème et non pas comme une méthode universelle.

Les différentes fonctions employées sont :

- Acquisition des coordonnées des sommets de la figure de base (sous-programme 1000) dont la translation engendre la représentation 3D.
- Calcul des coordonnées des points déplacés (sous-programme 2000). La technique consiste tout simplement à incrémenter toutes les coordonnées de la quantité S.

ORGANIGRAMME DU PROGRAMME DE SIMULATION 3D



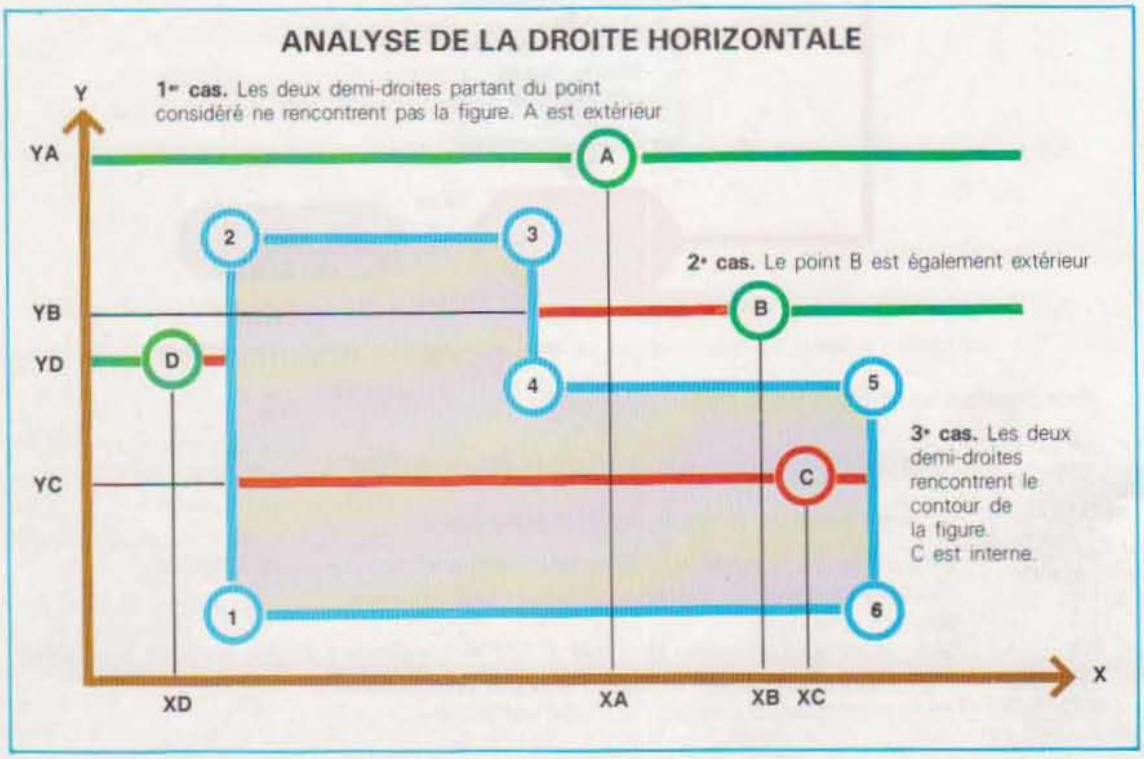
Par rapport au listing, les organigrammes de détail qui suivent ont, parfois, été simplifiés.

Principales variables utilisées

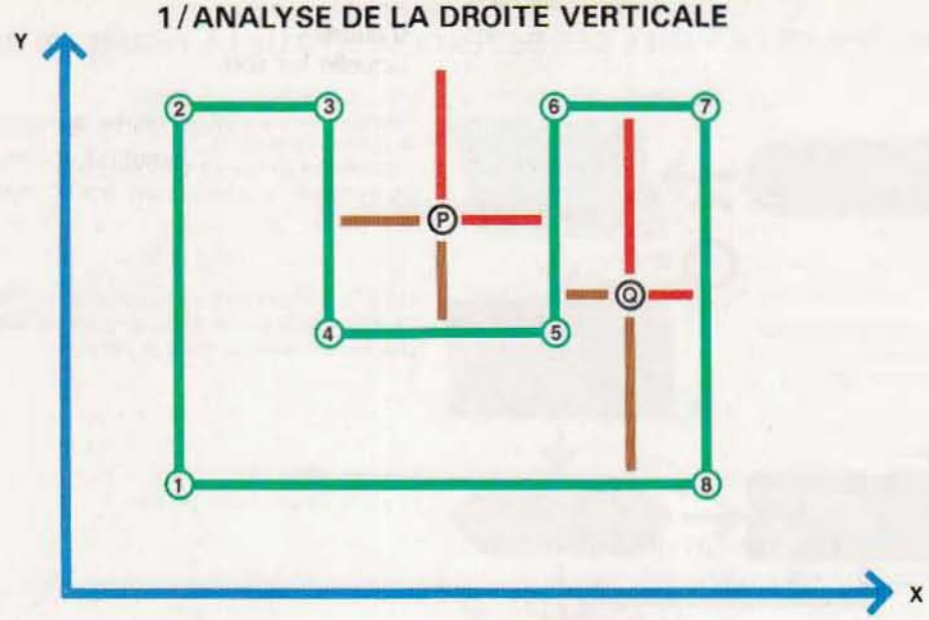
XP(21) } YP(21) }	Coordonnées de chaque point de la figure de base
XT(21) } YT(21) }	Coordonnées de la figure après translation
FL(20)	Indicateur de condition (1 si le point est interne, 0 s'il est externe)
N1	Nombre des points entrés
FS	Facteur d'échelle
OY	Indicateur d'orientation de l'axe Y (OY=-1 indique l'origine en haut à gauche)
S(2, 20)	Indicateur gauche/droite dans l'analyse horizontale
T(2, 20)	Indicateur haut/bas dans l'analyse verticale

■ Détermination du type de chaque point de la figure translatée. Une fois calculées les coordonnées des sommets, il faut vérifier si le segment unissant deux à deux les points correspondants est visible. Dans la figure de la page 1642, par exemple, le côté translaté 1 est uni au côté 2 par une ligne qui n'est pas visible ; à l'inverse, le segment qui unit les points translatés 2 et 3 doit l'être.
Le sous-programme 3000 analyse chacun des couples de sommets et détermine si le côté les unissant doit être affiché ou non.

La méthode consiste à déterminer quand un point du contour déplacé tombe à l'intérieur du contour de la figure de base.
Par exemple, le point translaté 1 doit être non visible puisqu'il se trouve à l'intérieur de la figure de base qui le recouvre.
Cette technique a ses propres limites. Si l'épaisseur (donc le déplacement S) est trop important, les coordonnées du point translaté se trouveront hors du contour de base, le point translaté sera lui aussi visible, et les résultats obtenus erronés.
Nous présentons ci-dessous l'algorithme utilisé pour déterminer si un point quelconque est interne ou non à un contour donné. Cette méthode, qui peut également être exploitée à d'autres fins, est fondée sur l'hypothèse selon laquelle les sommets ont été entrés de façon séquentielle.
Dans notre représentation, les numéros de 1 à 6 désignent les sommets du contour de base, et les lettres A à D les points dont on désire connaître la position par rapport à la figure. Un point est interne à la figure quand chacune des deux demi-droites horizontales, tracées à partir de ce point, coïncide avec le contour.
La figure 1 (page ci-contre) représente un type de contour particulier avec lequel l'analyse de la droite horizontale se révèle insuffisante. En effet, partant du point P, les deux demi-droites horizontales couperaient bien la figure de base, alors même que P est externe. L'analyse est également être effectuée le long de l'axe Y.
Une fois que le type de chaque point a été déterminé (interne/externe), le contour translaté peut être tracé (sous-programme 5000), et les sommets unis par les segments, créant ainsi une illusion d'épaisseur (sous-programme 6000). Le sous-programme 5000 contient un contrôle qui, dans certains cas, produit des résultats faux.
En effet, comme nous l'avons vu, il suffit, pour éliminer un côté, que l'une de ses deux extré-

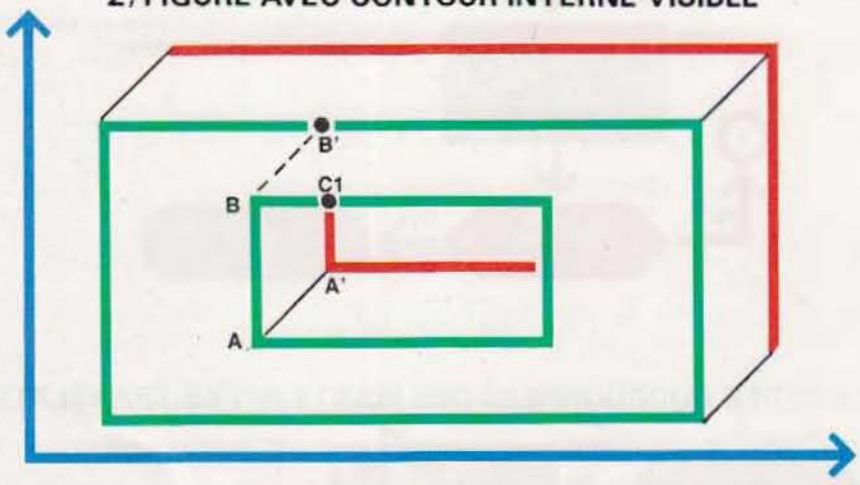


1/ANALYSE DE LA DROITE VERTICALE



Il s'agit d'un cas anormal qui rend nécessaire l'analyse selon l'axe Y. Avec uniquement l'analyse précédente, le point P serait vu intérieur à la figure (1ᵉ cas). Par contre, le second contrôle, le long de l'axe Y, indique qu'en réalité, il est extérieur. A l'inverse, le point Q relève du 1ᵉ cas d'après les deux analyses.

2/FIGURE AVEC CONTOUR INTERNE VISIBLE

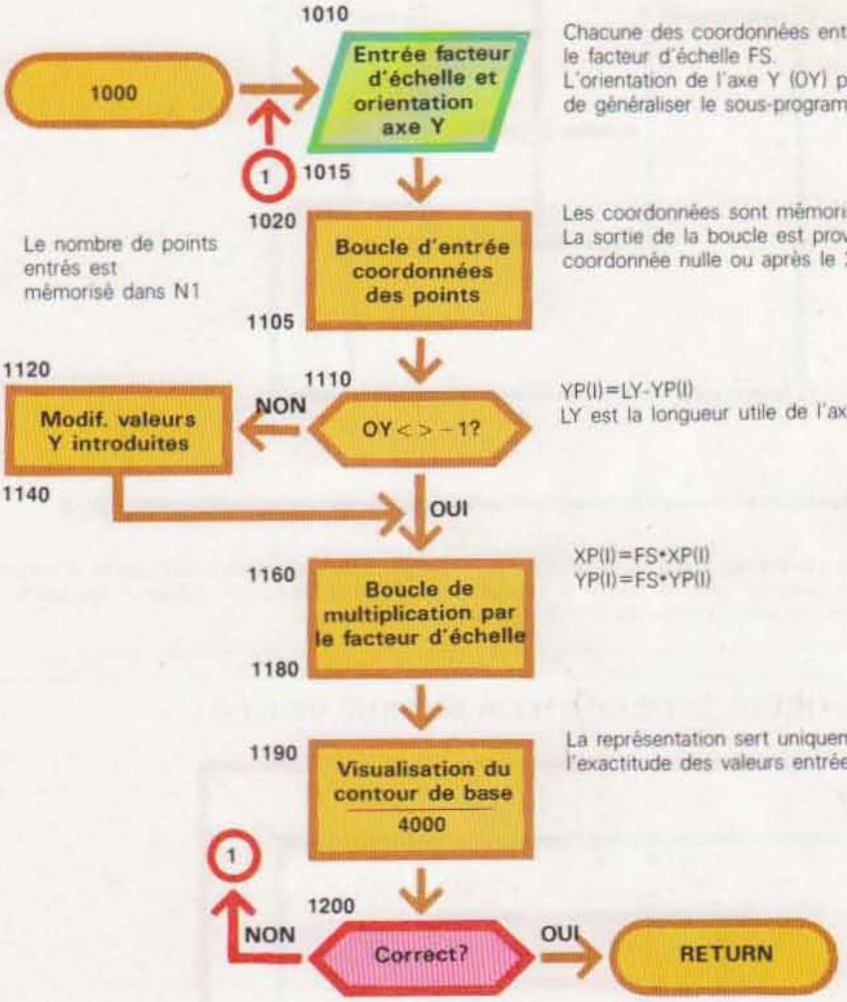


mités soit interne. D'après cette méthode, le côté unissant les sommets translatés 1 et 2 ne serait pas visible.

Il existe pourtant des cas où la logique décrite entraînerait l'omission de parties entières du dessin. Par exemple, dans celui de la figure 1, la partie interne serait totalement occultée. Il a donc fallu prévoir la possibilité de déterminer quand il existe des portions de côté visibles (sous-programme 5500). Là encore, certaines zones particulières du dessin peuvent toutefois être fausées (segments 2-a et 6-b de la figure, page 1642). L'erreur proviendrait de la sélection des points, dont la position n'est analysée que par rapport à la figure de base.

Le sous-programme 5500 illustre une méthode souvent employée pour identifier les points encore à l'intérieur de la fenêtre utilisée. Les pages 1646 à 1657 présentent les organigrammes et le listing de l'ensemble de la procédure.

ENTREE DE LA TABLE DES DEPLACEMENTS DE LA FIGURE DE BASE



Chacune des coordonnées entrées sera multipliée par le facteur d'échelle FS.
L'orientation de l'axe Y (OY) permet de généraliser le sous-programme

Le nombre de points entrés est mémorisé dans N1

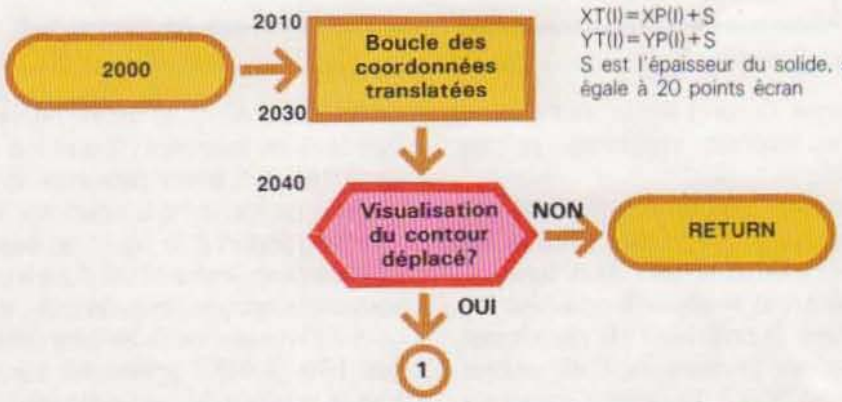
Les coordonnées sont mémorisées dans XP(I) et YP(I). La sortie de la boucle est provoquée par l'entrée d'une coordonnée nulle ou après le 20e point

YP(I)=LY-YP(I)
LY est la longueur utile de l'axe Y

XP(I)=FS*XP(I)
YP(I)=FS*YP(I)

La représentation sert uniquement à contrôler l'exactitude des valeurs entrées

CALCUL DES COORDONNEES DES POINTS APRES TRANSLATION

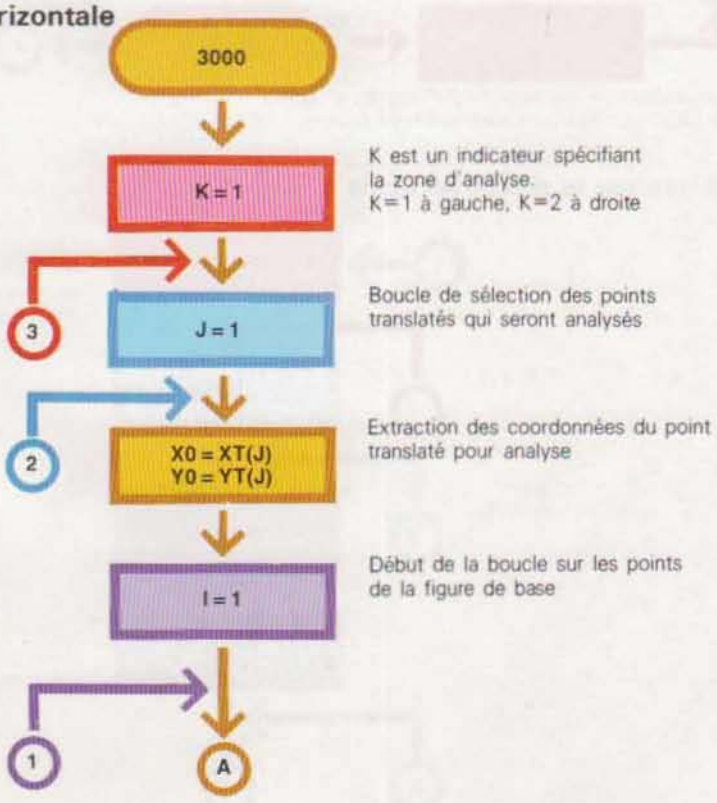


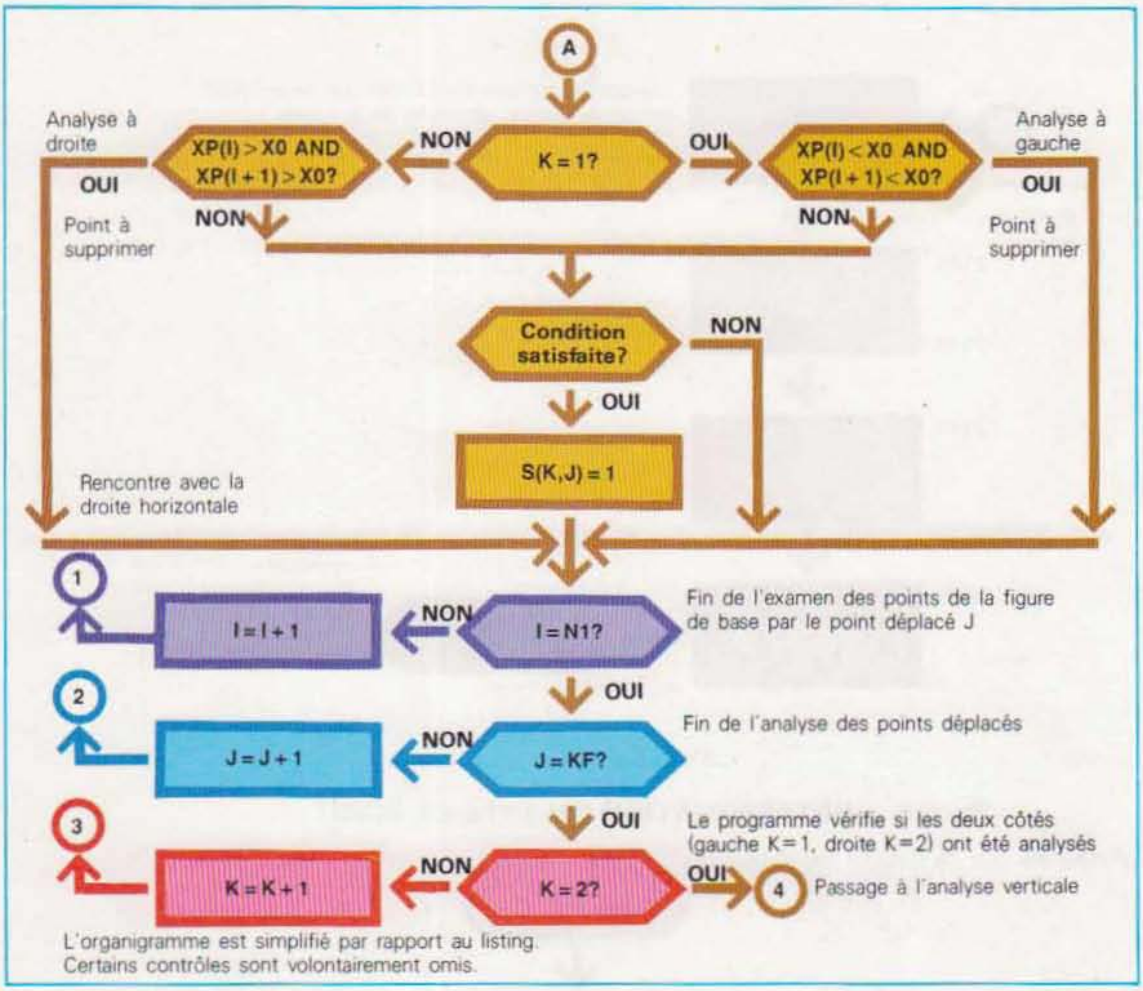
XT(I)=XP(I)+S
YT(I)=YP(I)+S
S est l'épaisseur du solide, supposée égale à 20 points écran



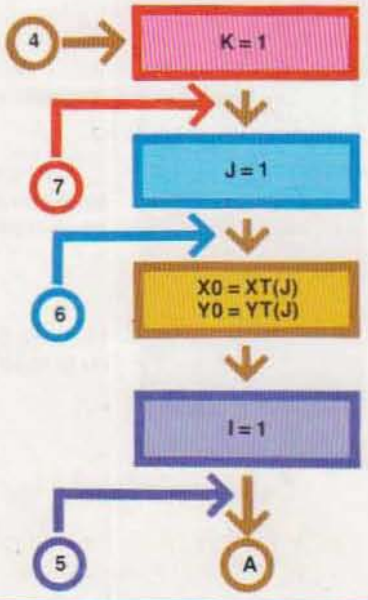
DETERMINATION DU TYPE DE POINT

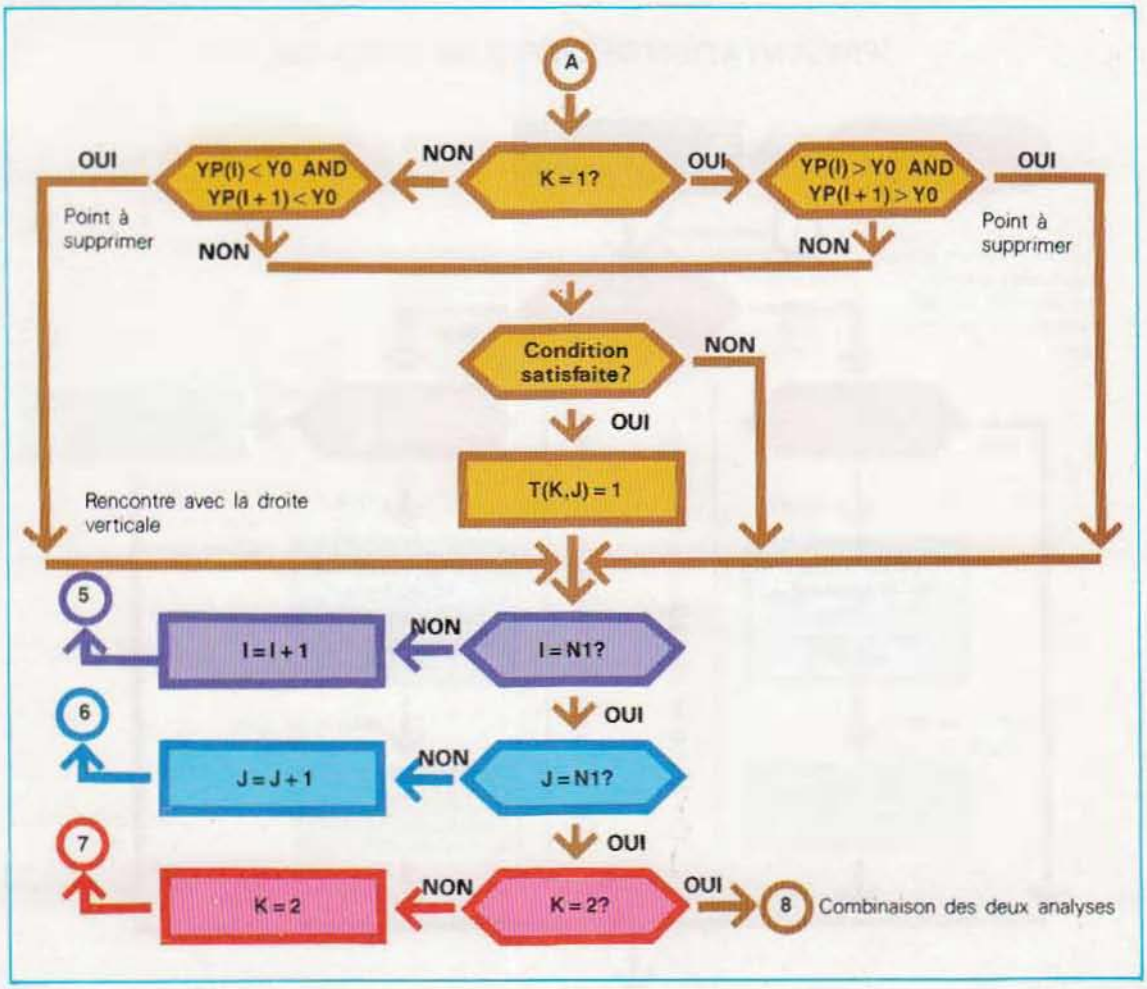
Analyse sur la droite horizontale

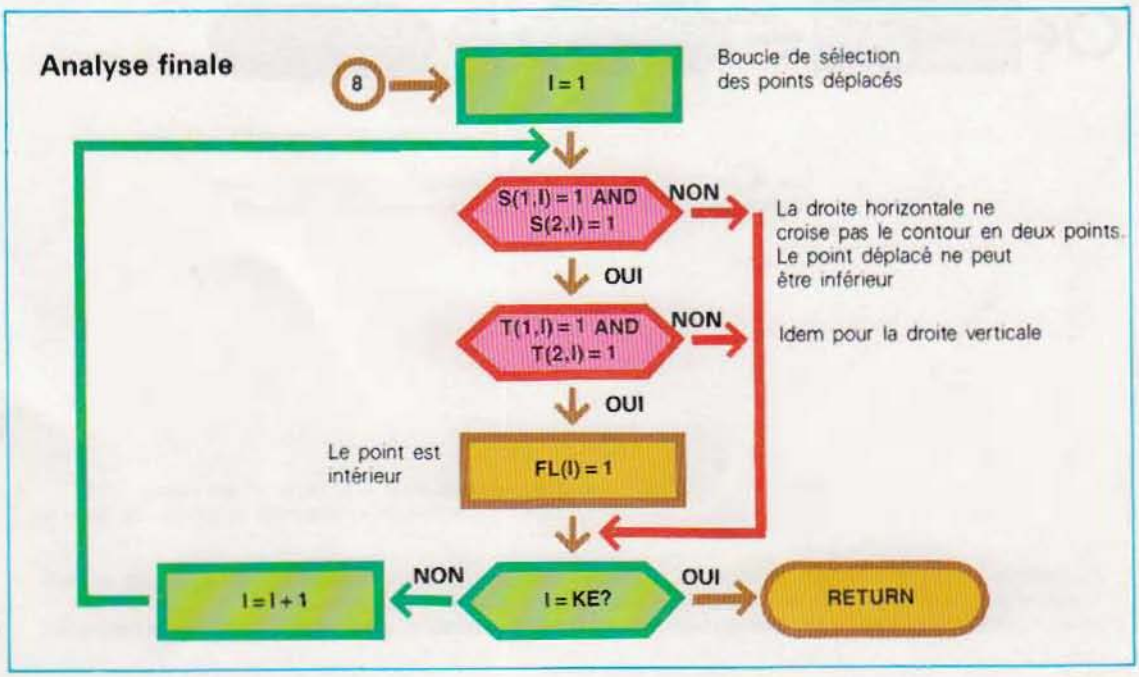




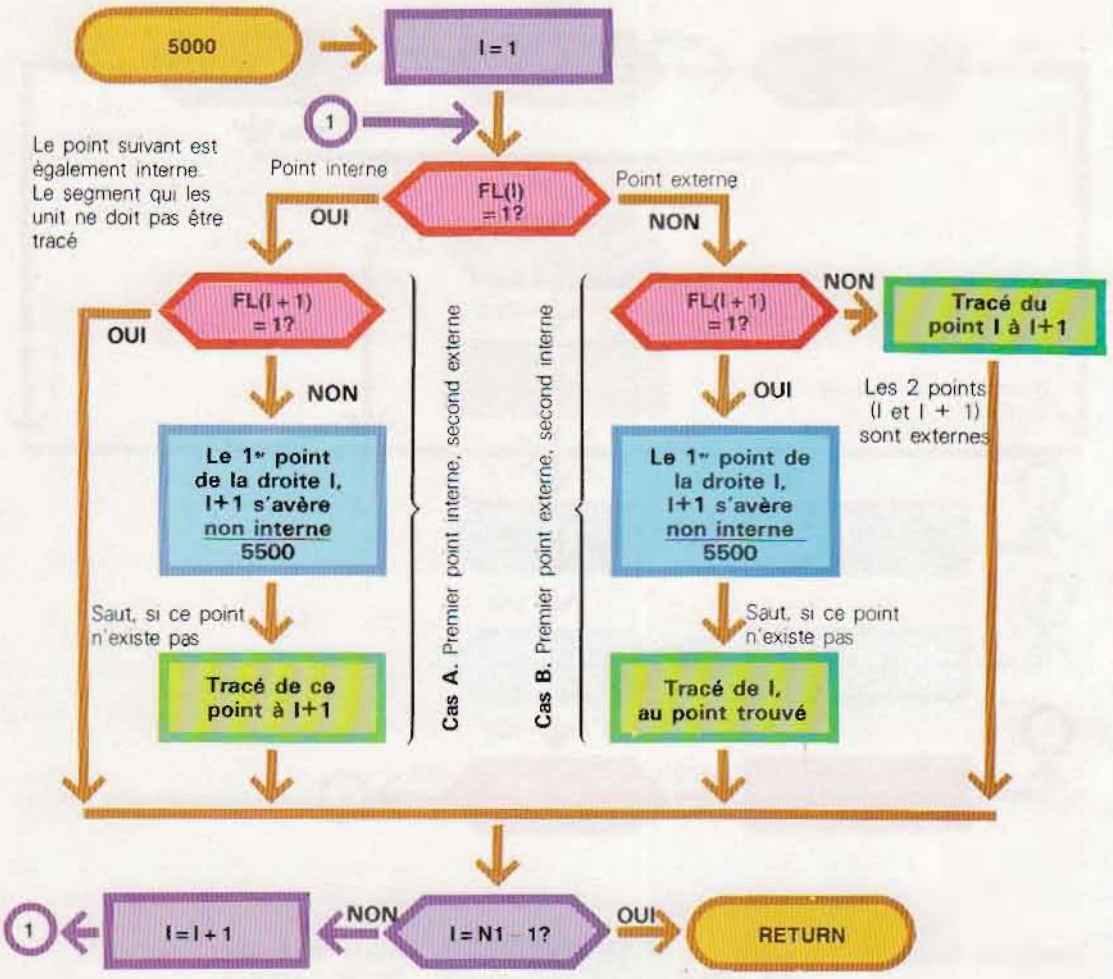
Analyse sur la droite verticale

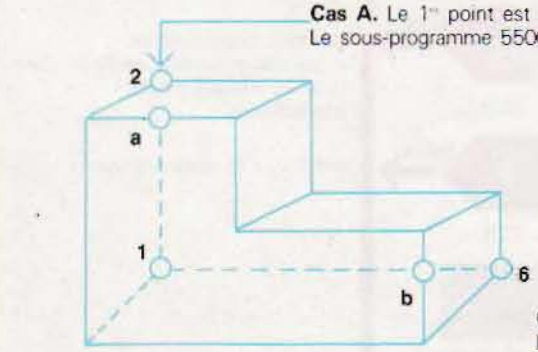



A
K = 1?
NON
OUI
YP(I) < Y0 AND YP(I+1) < Y0
OUI
Point à supprimer
NON
YP(I) > Y0 AND YP(I+1) > Y0
OUI
Point à supprimer
NON
Condition satisfaite?
NON
OUI
T(K,J) = 1
Rencontre avec la droite verticale
5
I = I + 1
NON
I = N1?
OUI
6
J = J + 1
NON
J = N1?
OUI
7
K = 2
NON
K = 2?
OUI
8
Combinaison des deux analyses


Analyse finale
8
I = 1
Boucle de sélection des points déplacés
S(1,I) = 1 AND S(2,I) = 1
NON
La droite horizontale ne croise pas le contour en deux points. Le point déplacé ne peut être inférieur
OUI
T(1,I) = 1 AND T(2,I) = 1
NON
Idem pour la droite verticale
OUI
Le point est intérieur
FL(I) = 1
I = I + 1
NON
I = KE?
OUI
RETURN

PRESENTATION DE LA FIGURE DEPLACEE



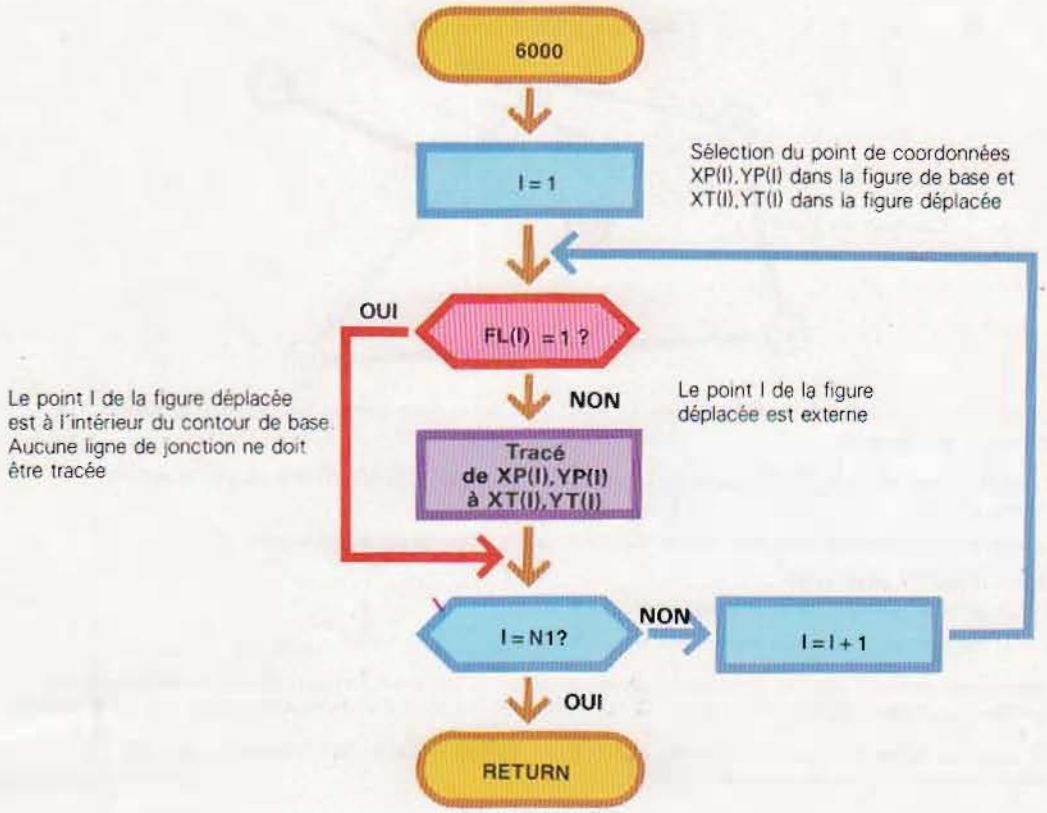


Cas A. Le 1er point est interne alors que le 2e point est externe. Le sous-programme 5500 détermine les coordonnées du point a

Cas B. Le point 6 est externe, le 1er est interne. Le sous-programme 5500 détermine la position du point b

La méthode présentée analyse la position des points déplacés uniquement par rapport à la figure de base et trace quelques segments erronés, comme les traits a,2 et 6,b. Pour avoir une vision complète, il faut aussi analyser chaque point déplacé par rapport aux surfaces générées par la translation, c'est-à-dire en fonction de l'épaisseur

LIGNES DE JONCTION



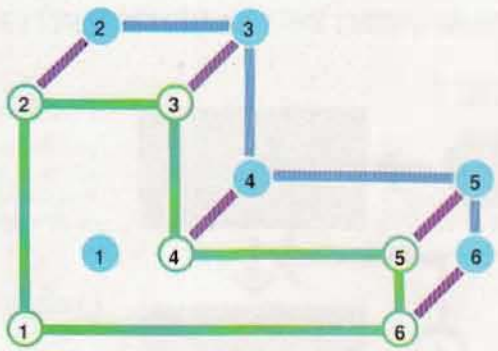


Figure de base

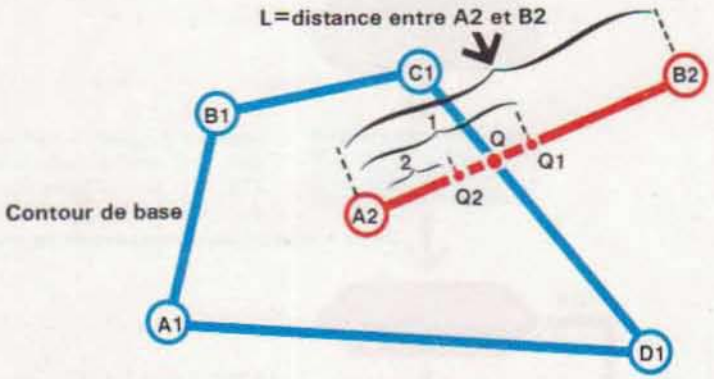
Points déplacés XT(I),YT(I)

Contour de la figure déplacée, présenté par le sous-programme 5000 (après suppression du sous-programme 5500)

Segments de jonction tracés par ce sous-programme

Les deux points 1 ne sont pas unis parce que celui de la figure déplacée se trouve à l'intérieur du contour de base

ALGORITHME UTILISE PAR LE SOUS-PROGRAMME 5500



A1, B1, C1 et D1 forment le contour de base. A2 et B2 sont deux points quelconques du contour déplacé, l'un interne et l'autre externe.

Il faut déterminer les coordonnées du point Q, appartenant au segment A2,B2, à partir duquel on sort du contour de base.

Le segment L, qui unit les deux points A2 et B2, est divisé en deux, ce qui définit le point Q1.

Deux cas peuvent se présenter :

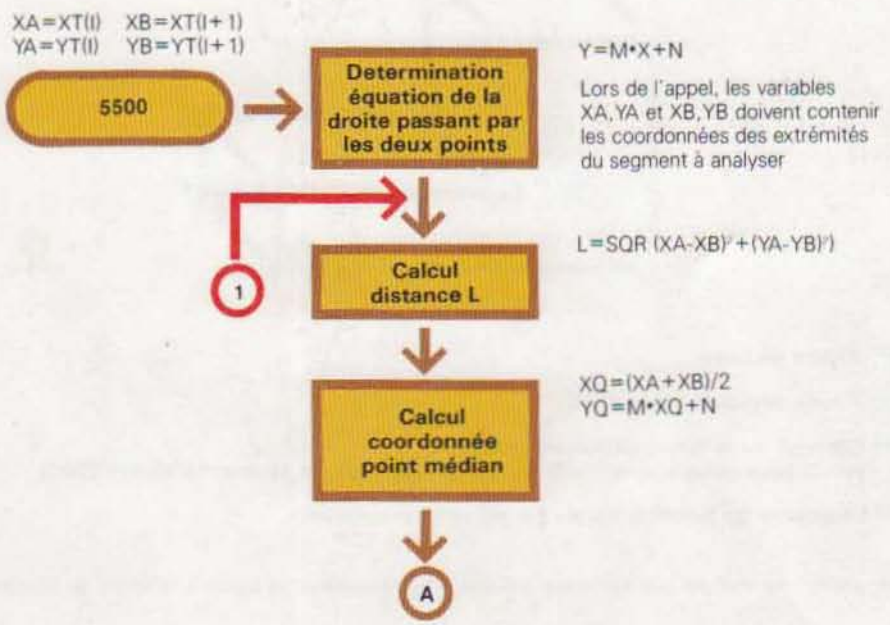
1 - Q1 est externe : l'analyse se poursuit sur la portion A2 Q1

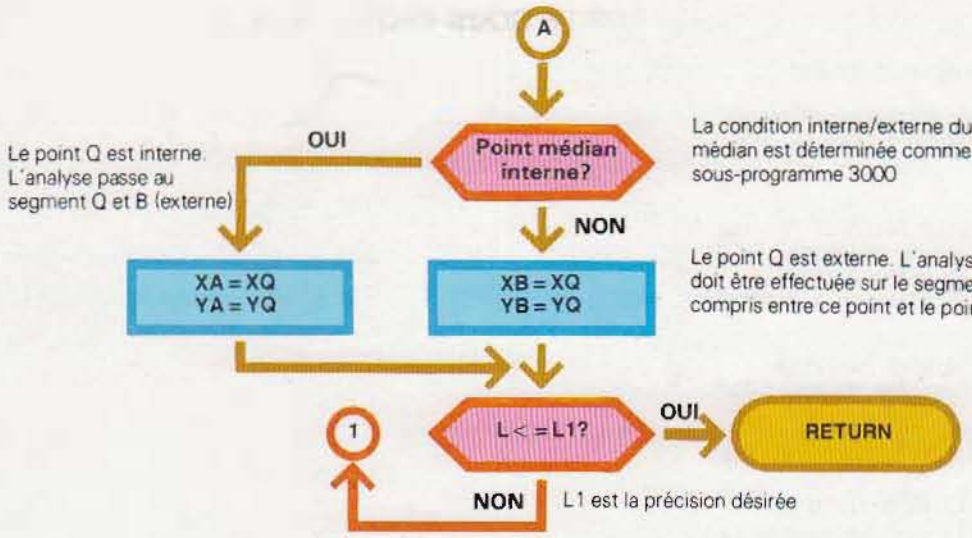
2 - Q1 est interne : l'analyse se poursuit sur la portion Q1 B2

Une fois sélectionné le segment sur lequel l'analyse va se poursuivre, le programme retourne au premier pas en remplaçant les deux extrémités du segment (A2,B2) par celles qui ont été trouvées au cours des pas précédents.

La boucle se répète quand la distance entre deux points Q successifs (Qn et Qn+1) est inférieure à une valeur prédéterminée (précision que l'on veut obtenir).

DETERMINATION DU POINT DE TRANSITION INTERNE/EXTERNE





La méthode illustrée ici donne des résultats approximatifs. En effet, la recherche est interrompue quand la précision requise (L1) est obtenue, alors qu'elle peut ne pas être suffisante du point de vue graphique. Un meilleur résultat peut être obtenu avec un autre système basé sur la lecture de la mémoire vidéo.
Il s'agit, en bref, de se déplacer le long du segment A2 B2 en déterminant l'état vidéo de chaque point. Les coordonnées de Q sont celles pour lesquelles le pixel est actif. Cette méthode donne de meilleurs résultats du point de vue graphique, mais elle demande plus de temps. En effet, le contenu de chacune des adresses correspondant aux points du segment A2 B2 doit être analysé. Il est possible de l'optimiser en la combinant avec la précédente. Ainsi, la première méthode détermine dans quel cadre se trouve Q, et la deuxième analyse ensuite ce cadre en vérifiant le contenu de la mémoire vidéo.

Le sous-programme 5500 peut être remplacé par : 5500 XQ=XB:YQ=YB:RETURN

De cette façon, le programme n'est que partiellement exécuté ; par contre, l'erreur est évitée.

GRAPHIQUE TRIDIMENSIONNEL AVEC EFFACEMENT DES LIGNES CACHEES

```
1 REM ====================
2 REM,* PROGRAMME  HAUTE *
3 REM * RESOLUTION  POUR *
4 REM * FIGURES EN 3D ET *
5 REM *  EFFACEMENT DES  *
6 REM *  LIGNES CACHEES  *
7 REM ====================
8 :
9 :
50 S=20:L1=5:LY=191
55 DIM YP(20),XP(20),YT(20),XT(20)
   ,FL(20),S(2,20),T(2,20)
60 ONERR GOTO 10000
65 CN=1
100 GOSUB 1000
110 GOSUB 2000
120 GOSUB 3000
130 GOSUB 4000
140 GOSUB 5000
150 GOSUB 6000
200 HOME:VTAB 21:INPUT "Autre
    graphique (O/N) ";RS$
210 IF LEFT$(RS$,1)="O" THEN RUN
```

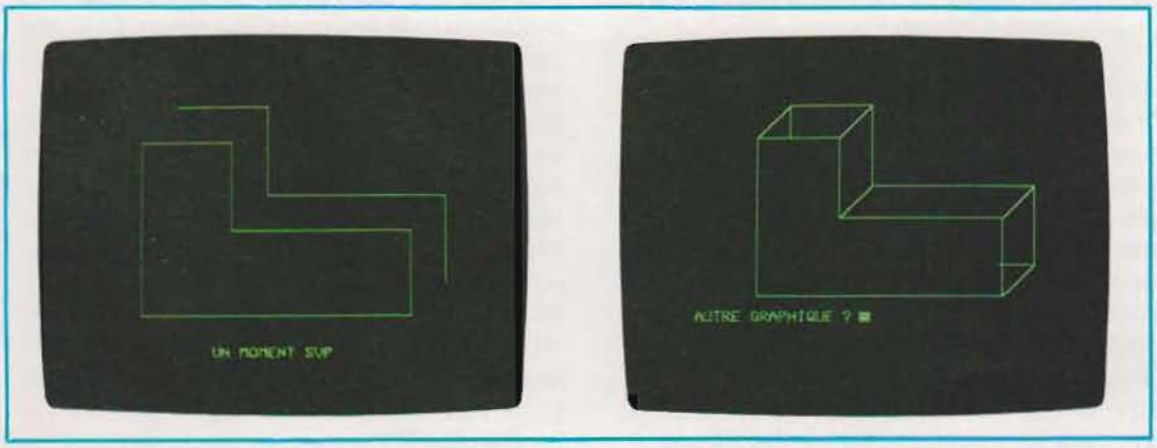
```
220 TEXT:HOME:VATB 22:END
999 :
1000 REM ======================
1001 REM ENTREE FIGURE DE BASE
1002 HGR
1003 :
1005 TEXT
1010 HOME:VTAB 10:INPUT "Entrez
      le facteur d'échelle ";F$
1015 VTAB 10:HTAB 10:INPUT
      "Orientation de l'ordonnée "
     ;OY
1020 I=1
1030 VTAB 10:HTAB 10:PRINT
      "Coordonnées du point "I"   "
1040 VTAB 10:HTAB 31:INPUT "X=
     ";XP(I)
1050 IF XP(I)*FS>278 THEN 1040

1060 VTAB 12:HTAB 31:INPUT "Y=
     ";YP(I)
1070 IF YP(I)*FS>191 THEN 1100

1080 IF YP(I)=0 OR XP(I)=0 THEN 1100

1090 IF I=20 THEN 1100
1095 I=I+1:GOTO 1030
1100 N1=I-1
1105 XP(N1+1)=XP(N1):YP(N1+
     1)=YP(N1)
1110 IF OY<>-1 THEN 1160
1120 FOR I=1 TO N1
1130 YP(I)=LY-YP(I)
1140 NEXT I
1160 FOR I=1 TO N1
1170 XP(I)=FS*XP(I):YP(I)=F
     S*YP(I)
1180 NEXT I
1190 GOSUB 4000
1200 VTAB 22:INPUT "Dessin correct
      (O/N) ";RS$
1210 IF LEFT$(RS$)="O" THEN
     RETURN
1220 GOTO 1000
2000 :
2001 REM ======================
2002 REM * CALCUL DES POINTS *
2003 REM *     DEPLACES      *
2004 REM ======================
2005 :
2010 FOR I=1 TO N1
2020 XT(I)=S+XP(I):YT(I)=YP
     (I)-S
2030 NEXT I
2035 XT(I)=XT(N1)+S:YT(I)=Y
     T(N1)-S
2040 HOME:VTAB 22:INPUT "Je présente
      la figure déplacée (O/N) ";RS$

2050 IF LEFT$(RS$,1)="O" THEN
     2070
2060 RETURN
2070 :
2090 FORI=1 TO N1-1
```

```
2100 HPLOT XT(I),YT(I) TO XT(I+1)
     ,YT(I+1)
2110 NEXT I
2120 HPLOT XT(N1-1),YT(N1-1) TO
      XT(N1),YT(N1)
2130 VTAB 22:PRINT "Enfoncez une
      touche pour effacer";:GET Q$

2140 HGR
2150 GOSUB 4000
2160 RETURN
2999 :
3000 REM --------------------
3001 REM * DETERMINATION DU *
3002 REM *  TYPE DE  POINT  *
3003 REM --------------------
3004 :
3005 KE=N1:HOME:VTAB 22:HTAB 10
     :PRINT "Un moment, svp"
3006 REM ----------------------
3007 REM :   ANALYSE  DE LA   :
3008 REM : DROITE HORIZONTALE :
3009 REM ----------------------
3010 FOR K=1 TO 2
3020 FOR J=1 TO KE
3025 IF KE=1 THEN 3040
3030 X0=XT(J):Y0=YT(J)
3040
      IF K=1 THEN 3070
3060 IF XP(I)>=X0 AND XP(I+1)<=X0
      THEN 3100
3065 GOTO 3080
3070 IF (YP(I)<=Y0 AND YP(I+1)>=Y0)
      OR (YP(I)>=Y0 AND YP(I+1)<=T0)
      THEN S(K,J)=1



3100 NEXT I,J,K
3105 :
3106 REM --------------------
3107 REM :  ANALYSE  DE LA  :
3108 REM : DROITE VERTICALE :
3109 REM --------------------
3110 FOR K=1 TO 2
3120 FOR J=1 TO KE
3125 IF KE=1 THEN 3140
3130 X0=XT(J):Y0=YT(J)
3140 FOR I=1 TO N1
3150 IF K=1 THEN 3170
3160 IF YP(I)>=Y0 AND YP(I+1)>=0
      THEN 3200
3165 GOTO 3180
3170 IF YP(I)<=Y0 AND YP(I+1)<=Y0
      THEN 3200
3180 IF (XP(I)<=X0 AND XP(I+1)>=X0)
      OR (XP(I)>=X0 AND XP(I+1)<=X0)
      THEN T(K,J)=1

3200 NEXT I,J,K
3205 :
3206 REM ------------------
3207 REM : ANALYSE FINALE :
3208 REM ------------------
3210 FOR I=1 TO KE
3220 IF S(1,I)=1 AND S(2,I)=1 THEN
      3240
```

```
3230 GOTO 3250
3240 IF T(1,I)=1 AND T(2,I)=1 THEN
      FL(I)=1
3250 NEXT I
3260 RETURN
3999 :
4000 REM ======================
4001 REM * REPRESENTATION DE *
4002 REM * LA FIGURE DE BASE *
4003 REM ======================
4004 :
4005 HGR:HCOLOR=3
4010 FOR I=1 TO N1-1
4020 HPLOT XP(I),YP(I) TO XP(I+1)
     ,YP(I+1)
4030 NEXT I
4035 GOTO 4060
4040 HPLOT XP(N1-1),YP(N1-1) TO
      XP(N1),YP(N1)
4060 RETURN
4999 :
5000 REM ======================
5001 REM * REPRESENTATION  DE *
5002 REM * LA FIGURE DEPLACEE *
5003 REM ======================
5004 :
5010 FOR P=1 TO N1-1
5020 IF FL(P)=CN THEN 5200
5030 IF FL(P+1)=CN THEN 5050

5040 HPLOT XT(P),YT(P) TO XT(P+1)
     ,YT(P+1)
5045 GOTO 5300
5050 XB=XT(P):XA=XT(P+1):YB=YT(P)
     :YA=YT(P+1)
5055 GOSUB 5500
5060 HPLOT XT(P),YT(P) TO X0,Y0
5070 GOTO 5300
5100 :
5200 IF FL(P+1)=CN THEN 5300

5210 XB=XT(P):XA=XT(P+1)
5211 YA=YT(P):YB=YT(P+1)
5215 GOSUB 5500
5220 HPLOT XQ,YQ TO XT(P+1),YT
     (P+1)
5300 NEXT P
5310 RETURN
5499 :
5500 REM ========================
5501 REM * POINTS DE TRANSITION *
5502 REM *    INTERNE/EXTERNE   *
5503 REM ========================
5504 :
5505 IF XA=XB THEN X=1:GOTO
     5515
5506 REM -------------
5507 REM : Y0=XQ*M+N :
5508 REM -------------
5510 X=XB-XA
5515 M=(YB-YA)/X:N=INT(
     YA-(XA*M))
5520 L=INT(SQR((XA-XB)^2+(YA-YB)^2))
```

```
5525 IF X=1 THEN X0=XA
     :Y0=INT((YA+YB)/2):GOTO 5540
5530 X0=INT((XA+XB)/2):Y0=INT(M*X0+N)
5540 X0=XQ:Y0=YQ
5545 S(1,1)=0:S(2,1)=0:T(1,1)=0
     :T(2,1)=0
5550 CF=FL(1):KE=1:FL(1)=0:GOSUB 3010
5580 IF FL(1)=CN THEN XA=XQ:YA=YQ
     :GOTO 5600
5590 XB=XQ:YB=Y0
5600 FL(1)=CF
5605 IF L<=L1 THEN SW=0:RETURN

5610 GOTO 5520

5999 :
6000 REM ==============
6001 REM * LIGNES DE *
6002 REM * FONCTION  *
6003 REM ==============

6004 :
6010 FOR I=1 TO N1
6020 IF FL(I)=CN THEN 6040
6030 HPLOT XP(I),YP(I) TO XT(I),YT(I)
6040 NEXT I
6050 RETURN

10000 :
10001 REM ==========
10002 REM * ERREUR *
10003 REM ==========

10004 :
10005 TEXT:HOME
10010 ER=PEEK(222)
10020 IF ER=53 THEN VTAB 10:PRINT
      "Coordonnées fausses "
     :GOTO 10050
10030 VTAB 10:PRINT "Données fausses "
10050 VTAB 22:HTAB 30:PRINT ">>";
10060 GET Q$:RUN
```

Exemple de fonctionnement du programme présenté.
A gauche, le programme a analysé les points de la figure de base et a présenté, après translation, ceux qui sont visibles. A droite, le dessin a été complété. Remarquer les deux segments erronés.



L'animation d'images par ordinateur

L'animation d'images graphiques constitue une activité bien particulière dans l'informatique graphique. Les techniques d'animation sont déjà largement utilisées dans les programmes de jeux vidéo. Il existe de très nombreux exemples dans lesquels l'animation d'une figure à l'écran peut simplifier considérablement la résolution d'un problème. C'est le cas quand on désire connaître l'effet d'une force sur une structure solide, ou observer le mouvement d'un organe mécanique.

Les objets animés

Nous avons vu que la visualisation d'une image graphique n'est pas une opération élémentaire. Elle passe par le tracé d'un grand nombre de segments qui serviront à la reconstitution de l'image. Le déplacement de l'objet correspondra à une succession d'effacements et de nouveaux tracés dans une autre position

Un **objet animé** (on dit « **sprite** », lutin, en anglais), est, à l'inverse, une image graphique qui est gérée comme telle par le programme, en choisissant une fois pour toutes sa forme et ses différentes positions à l'écran.

L'objet animé, qui est limité en fonction des dimensions de l'écran vidéo, possède trois caractéristiques principales :

— il est complètement programmable par l'utilisateur ;
— il se déplace dans les quatre directions du plan ;
— il mémorise d'éventuelles collisions avec d'autres objets animés ou d'autres figures présentes à l'écran.

Chacune de ces caractéristiques peut être mise en œuvre à l'aide de sous-programmes spéciaux, assurant la simulation d'objets animés même dans les systèmes ne les prévoyant pas. Les autres machines possèdent, quant à elles, des instructions spéciales de déplacement, de contrôle de collisions, etc. Elles sont donc dotées de certaines implémentations qui ne se trouvent pas dans le Basic Standard.

La gestion des objets animés est généralement commandée par des composants matériels ad hoc. Pour les machines plus banales il faut écrire des programmes qui vont simuler ces composants. Les fonctions exécutables sont toutefois plus limitées et la vitesse moins élevée, tout au moins en Basic interprété.

Pilotage des objets animés par logiciel

L'emploi des objets animés est étroitement lié à la production de dessins animés. Certaines des difficultés, rencontrées dans le pilotage par programme, sont justement dues à la nature de cette application.

Un objet animé peut être créé comme une table de figures et déplacé par variation de l'origine. Cette technique, bien connue, consiste à reproduire la figure à déplacer (l'objet animé) comme une série de déplacements (paramétrables) par rapport à une origine.

Si les coordonnées du point choisi comme référence sont modifiées, la figure se trouve déplacée (il faut naturellement prévoir son effacement).

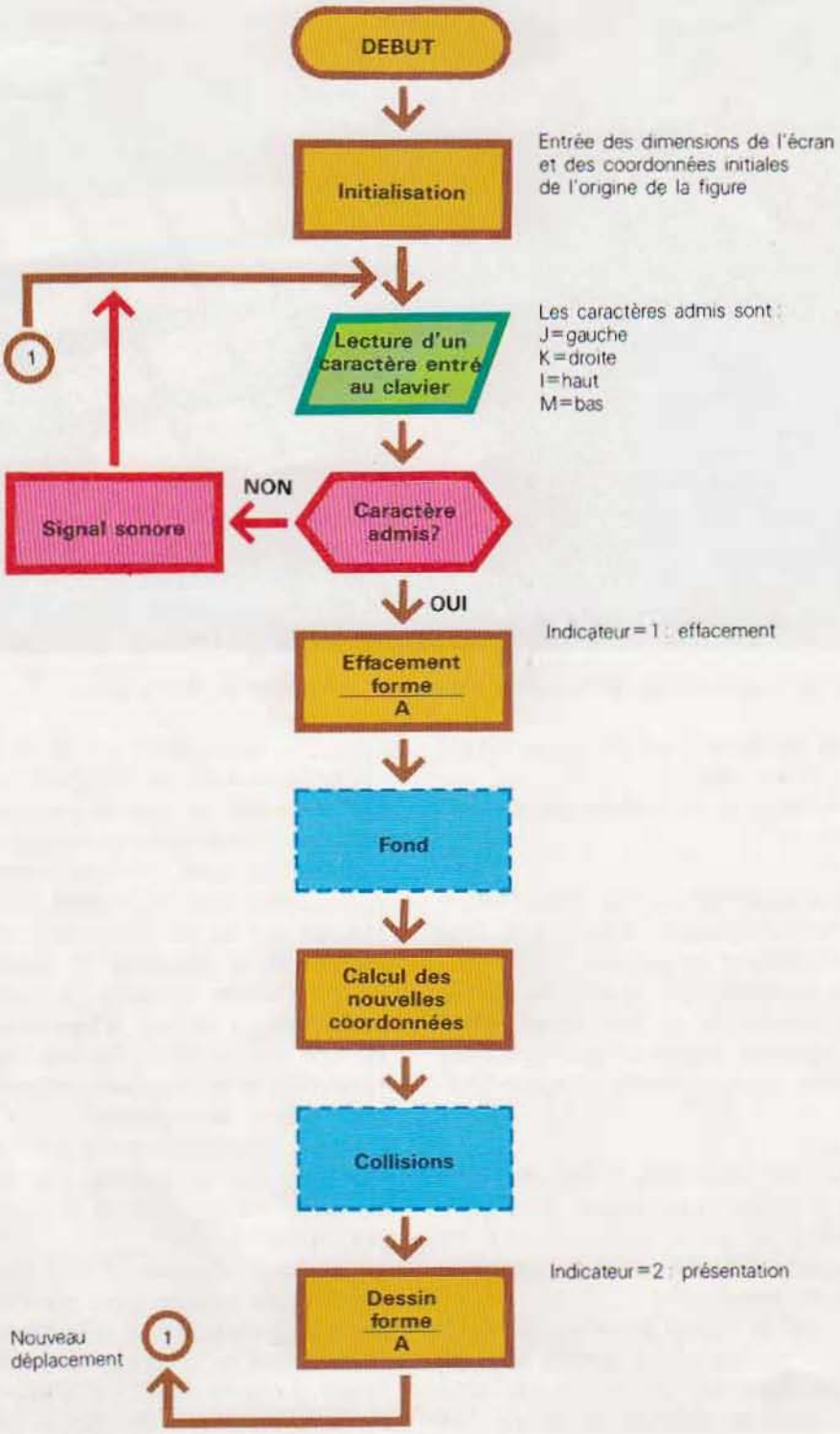
Page ci-contre, se trouve l'organigramme d'un programme qui déplace une figure à l'écran en suivant les commandes entrées au clavier. Les différentes phases de ce programme apparaissent immédiatement. Le sous-programme marqué A mérite, toutefois, une observation : il exécute aussi bien l'effacement que la visualisation. Il est déclenché par l'introduction d'un indicateur spécifiant laquelle de ces deux fonctions est désirée.

L'organigramme comporte également deux blocs (Fond et Collisions) dont l'emploi est spécifique à cette application.

L'animation d'images graphiques. L'animation par ordinateur est une technique très proche de celle employée au cinéma. La scène contient des images fixes (le fond) par rapport auxquelles se déplace l'objet de l'animation.

Au cinéma, la technique consiste à dessiner uniquement le fond, plus autant de tables, sur support transparent, que de mouvements désirés. Sur chacune de ces tables, la figure à déplacer est reproduite dans une position légèrement modifiée par rapport à la précédente. En posant ces différentes tables sur le fond et en photographiant toutes les scènes intermédiaires obtenues, on obtient un fond, qui, à la projection, donnera une réelle impression de mouvement.

DEPLACEMENT D'UNE FORME





P.A. Simon/Grazia Neri-Photolike

Un moment de la conception assistée par ordinateur des image du film « Tron ».

En réalité le problème n'est pas aussi simple qu'on pourrait le croire.
Noter que la figure qui se déplace est gérée de deux façons :

1 / Lors du déplacement par rapport au fond, la figure reste inchangée ; il faut donc faire varier uniquement sa position.
2 / La figure se déplace par rapport au fond et subit des modifications. Dans ce cas, il faut ajouter au déplacement un nouveau tracé de la figure, partiel ou total, à chaque position.

Le deuxième cas, très facile à résoudre avec une feuille de papier et un crayon, pose quelques difficultés au niveau de l'ordinateur. La première grande difficulté, c'est de rétablir le fond après un mouvement.
Un point donné de l'écran appartient au fond, du moins tant que sa zone n'est pas touchée par la figure déplacée. De fait, il est actif (visible) ou éteint en fonction de ce que l'on doit montrer en fond.

Quand le même point est caché par la figure déplacée, il subit les impératifs de tracé de celle-ci et doit être rétabli à sa position initiale après un déplacement le dégageant.
Par exemple, si un petit bonhomme qui se promène passe devant un arbre, l'ordinateur devra, de pas en pas, remplacer une partie de l'arbre par la silhouette du bonhomme, puis rétablir l'image d'origine de l'arbre dès qu'il sera passé. Si, de plus, la figure est susceptible de subir des modifications, il ne s'agit alors plus seulement de déplacer un graphique, mais aussi de prévoir ses variations.
Dans les machines assurant l'animation, le fond est traité par le système : la figure mobile constitue l'objet animé et peut être déplacée par rapport au fond.
La sensation obtenue est celle de mouvement d'une figure statique. Pour obtenir une animation plus réaliste, il faut alors également modifier la forme de la figure, par exemple en dessinant quelques détails en différents points.
Avec la technique des objets animés, deux possibilités sont offertes : modifier la forme de

l'objet animé avant déplacement ou créer autant d'objets animés qu'il y a de positions à simuler et les présenter les uns à la suite des autres.
Dans ce cas, chacune des figures représente un objet, et l'animation consiste à les activer successivement. Ce système est alors tout à fait analogue à l'animation traditionnelle.
Les mêmes problèmes se posent dans la gestion des objets animés avec des programmes écrits par l'utilisateur. Il est nécessaire d'y prévoir un sous-programme de reconstruction du fond et un autre pour le contrôle des collisions. Avec les machines dédiées à ce type d'applications, cette dernière fonction est automatiquement prise en charge.

Gestion du fond. Il existe différentes méthodes de gestion du fond d'un dessin animé. Leur degré de complexité dépend principalement de deux paramètres :

- La vitesse d'exécution
- L'occupation de la mémoire

Pour donner une impression de déplacement continu, l'animation d'un dessin demande une vitesse d'exécution élevée.
Si le programme n'est pas suffisamment rapide, l'œil réussit à percevoir les différentes phases du déplacement (effacement, reconstruction du fond, nouvelle présentation), ce qui n'est pas agréable ; de plus, il faut effectuer des déplacements relativement importants pour ne pas obtenir un mouvement trop lent, ce qui serait inacceptable dans les jeux vidéo par exemple.
La méthode et les exemples exposés ci-après ne servent qu'à illustrer ces aspects de la gestion du fond. Pour rester suffisamment généraux, ils n'exploitent pas les particularités matérielles de certaines machines, qui pourraient permettre de réduire les problèmes de lenteur d'exécution et d'espace mémoire. Il est en tout cas pratique d'utiliser la version interprétée des programmes dans la phase d'écriture et de contrôle, puis de recourir à la version compilée pour l'application, afin d'obtenir des vitesses d'exécution acceptables.
Une autre solution consisterait à écrire certains sous-programmes en Assembleur.

Le fond peut être considéré comme un dessin faisant appel à l'écran tout entier. La gestion la plus simple, lors d'une animation, serait d'enregistrer le dessin dans une zone mémoire spéciale puis de le rappeler au moment de sa reconstruction.
Voici les fonctions impliquées dans le déplacement d'un objet animé :

1 / Transfert du fond de la mémoire de soutien à l'écran (visualisation)
2 / Présentation de l'objet dans sa position initiale
3 / Présentation du fond (entraînant l'effacement de l'objet)
4 / Présentation de l'objet dans la position suivante

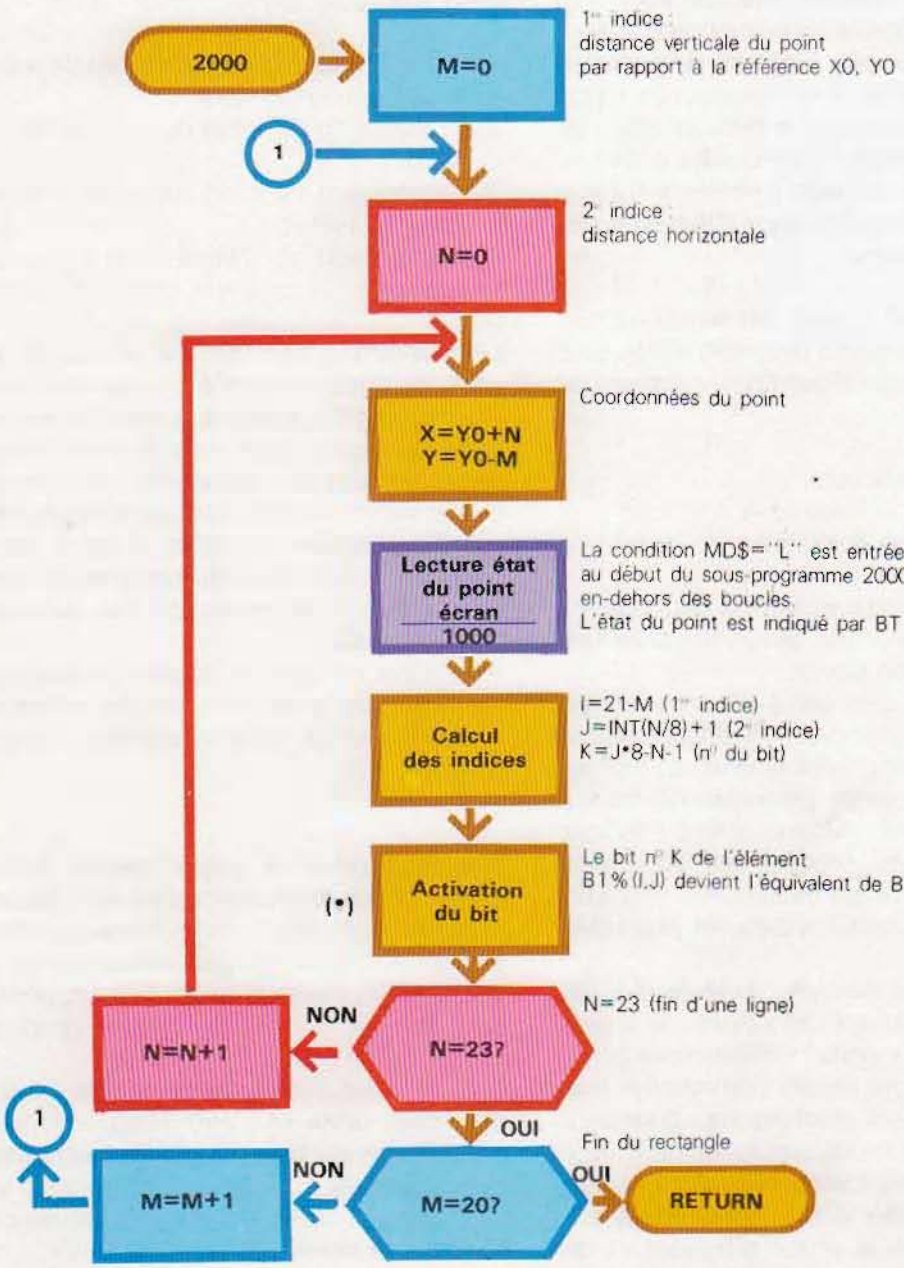
Le mouvement de l'objet est obtenu en activant, de façon itérative et rapide, ces quatre phases, jusqu'à reproduire le déplacement tout entier au moyen d'une série de petits mouvements donnant une impression de continuité.
L'effet visuel de cette méthode n'est toutefois pas des meilleurs. En effet, la faible vitesse d'exécution, liée à la quasi-totalité des interpréteurs Basic, rend perceptible l'enchaînement des opérations.
Il n'y a pas de solution logicielle généralisable, car une bonne animation nécessite obligatoirement l'emploi de certains éléments matériels particuliers.

Visualisation d'un objet animé. Voyons maintenant de près comment on peut résoudre le problème d'une machine. Nous voulons visualiser à l'écran un petit rectangle de 24 (base) sur 21 (hauteur) points écran *, position au point X0, Y0 sur une image graphique constituant le fond.
Chacun des déplacements de l'objet (rectangle) devra être accompagné de la reconstruction de la partie du fond précédemment masquée.
Ce processus comporte les trois phases successives suivantes :

*Ces valeurs ont été choisies pour conserver les dimensions de l'objet, utilisées par les systèmes prévoyant la gestion matérielle des objets animés.

Phase 1. Avant le positionnement de l'objet, la zone de l'écran concernée (rectangle ayant les mêmes dimensions que l'objet) est sauvegardée dans une zone mémoire (tableau bidimensionnel). Cette fonction consiste à calculer, à partir des coordonnées à l'écran de chaque point à sauvegarder, l'adresse correspondante dans la mémoire vidéo.

MEMORISATION DE LA FENETRE CONTENANT L'OBJET



• Ce bloc d'instructions est identique à celui utilisé dans le sous-programme 1000 pour activer un point écran. Cette méthode concerne les machines n'admettant pas d'employer les opérateurs logiques dans la création de masquages à l'intérieur de l'octet. Quand cette possibilité existe (par exemple sous CP/M), il suffit d'utiliser l'opérateur logique approprié.

Le contenu de cette mémoire est lu et transféré dans une mémoire tampon. Dans l'organigramme ci-contre, le tampon utilisé est B1%(21,3) (ce dimensionnement sera expliqué plus loin).

Phase 2. L'objet est introduit à partir du point X0, Y0. Le contenu de la mémoire vidéo correspondante est remplacé par le contenu du tableau S%(21,3) qui représente, sous une forme spéciale, l'objet.

Phase 3. Le déplacement de l'objet est obtenu en rétablissant le fond, c'est-à-dire en transférant B1%(21,3) dans la zone mémoire vidéo occupée par l'objet et en reprenant le cycle après avoir modifié les valeurs de X0,Y0.
Les phases 2 et 3 peuvent être décrites dans le même organigramme. Le sous-programme correspondant, qui est paramétré, présente, à partir des coordonnées X0,Y0, l'objet S%(21,3) ou le fond B1%(21,3).

Le transfert de données, à partir et à destination de la mémoire vidéo, nécessite le calcul de la position de mémoire correspondant aux coordonnées en points écran et du bit qu'elle contient. Sur ce point, nous avons repris le sous-programme 100 déjà présenté en l'adaptant.
Ce sous-programme demande, en entrée, les coordonnées X et Y du point écran et la valeur affectée à MD$ (MD$="L" pour la lecture de l'état allumé/éteint, MD$="S" pour l'écriture). Il fournit, en sortie, BT=1 si le point est allumé et BT=0 s'il est éteint.
Il est illustré, page suivante, par le tableau B1%(21,3). Chacun des éléments se composant de 8 bits, il peut contenir l'état d'autant de points écran. Les 24 points horizontaux sont donc mémorisés dans 3 cellules mémoire seulement. La même logique est utilisée dans certaines machines prévoyant la gestion des objets animés.
L'organigramme de reconstruction du fond ou de présentation de l'objet en fonction de l'indicateur F (F=1 fond, F=2 objet) est en page 1665. Les fonctions exécutées sont tout à fait analogues à celles du sous-programme 2000 avec, pour seule différence, l'utilisation de deux tableaux différents selon la valeur de F (F=1 pour B1%, F=2 pour S%).

Page 1666 enfin, c'est l'organigramme d'un programme dans lequel les sous-programmes 5000 et 6000 servent uniquement à préparer les données (voir listing en pages 1667 à 1669).

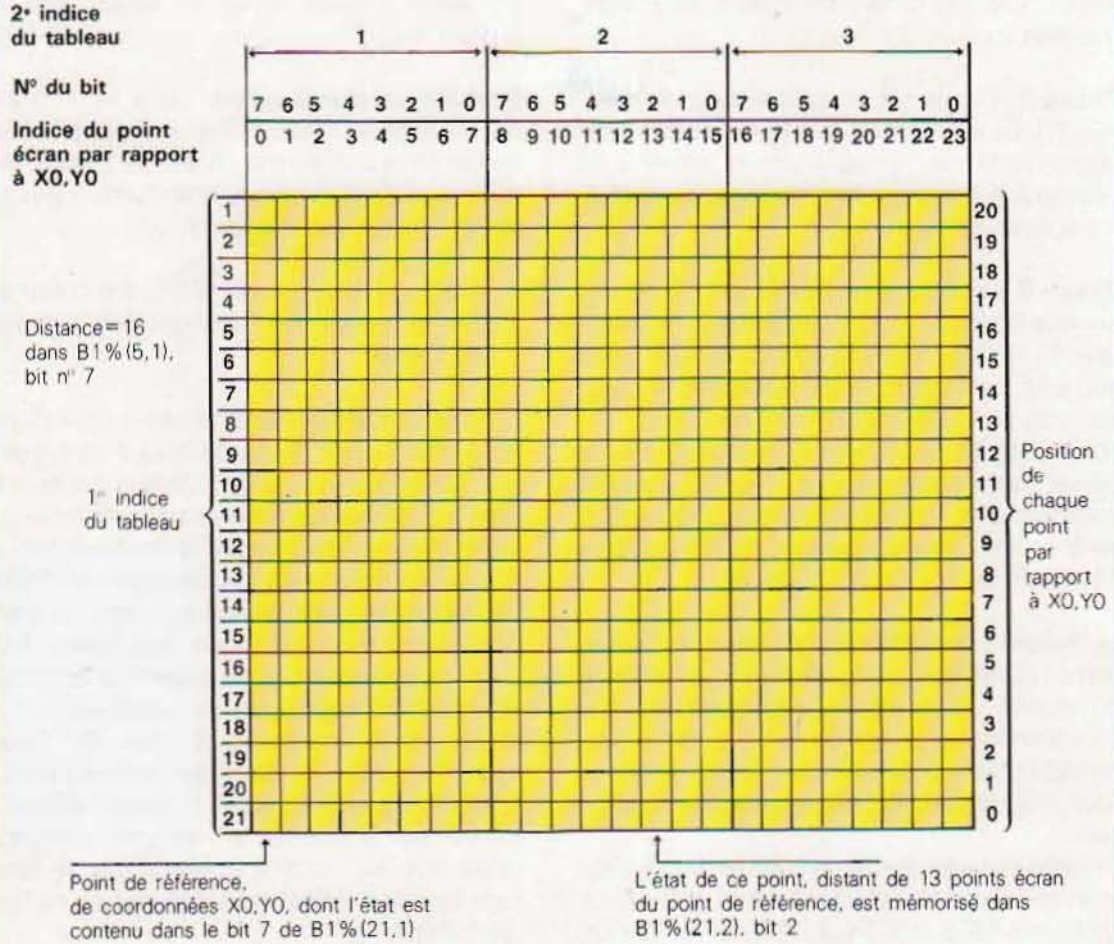
Problèmes d'animation. Dans le programme de la page 1666, l'objet est assimilé à un rectangle qui, converti en image binaire, est mémorisé dans le tableau S%. Cette méthode présente deux inconvénients :

1 / Elle ne permet pas de définir des contours.
2 / Elle ne prévoit pas d'animations à l'intérieur de l'objet.

Le premier oblige à remplir toute la zone disponible pour l'objet, faute de quoi il se créerait une zone toujours éteinte. Celle-ci correspondrait à la différence entre la zone du rectangle effectivement occupée par le dessin (objet) et la totalité de la zone à la disposition de l'objet. Ce défaut peut être éliminé à l'aide d'un sous-programme de présentation plus évolué. Il ne s'agit plus simplement de présenter le rectangle formant l'objet, mais de construire l'intersection entre ses points et ceux de l'écran. Autrement dit, si un point de l'objet est actif, il doit être visualisé à l'écran (il couvre le fond) ; s'il est inactif (éteint), la mémoire vidéo doit rester intacte, comme couverte par le fond. Cela équivaut à dessiner l'objet sur un rectangle transparent.
Cette fonction peut être obtenue très simplement à l'aide d'un IF lors du transfert en mémoire vidéo de l'image de l'objet. Si le point concerné n'est pas actif, il n'y aura pas de transfert, de manière à garder le fond inchangé (voir schéma en page 1671).
Le deuxième défaut (pas d'animation dans l'objet) est nettement plus difficile à traiter. En fait, même avec des machines dédiées, l'animation reste une fonction difficile. Il s'agit, en effet, de modifier la forme de l'objet au fur et à mesure que le rectangle qui le contient se déplace.
On crée ce mouvement de deux façons :

1 / En mémorisant certaines positions intermédiaires.
2 / En schématisant les déplacements à l'aide de règles mathématiques.

ADRESSAGE DE LA MEMOIRE DE L'OBJET



Le balayage commençant par la dernière rangée (élément de 1e indice=21), les points écran de la rangée 21 peuvent être divisés en trois groupes :

— le premier, (2e indice=1) est contenu dans les bits 0 à 7 de l'élément B1%(21,1)
— le deuxième est contenu dans les bits 0 à 7 de B1%(21,2)
— le troisième est contenu dans B1%(21,3)

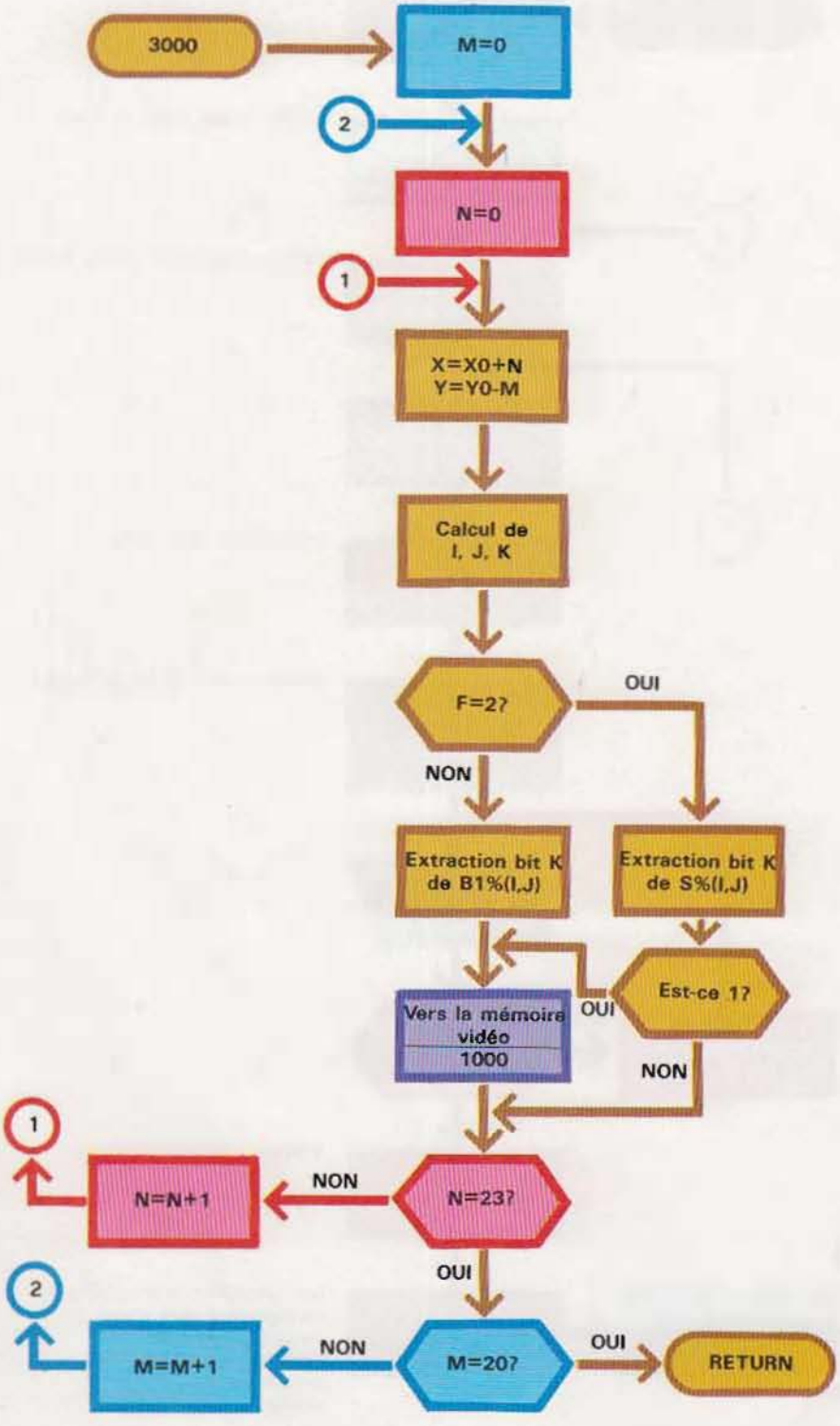
La même procédure permet d'obtenir l'adresse (indices et bits) de tout autre point. Par exemple, le point N=12, M=13 est contenu dans l'élément B1%(8,2) bit 3, selon la règle :

1e indice=21-M
2e indice =INT(N/8)+1
3e du bit=Second indice*8-N-1

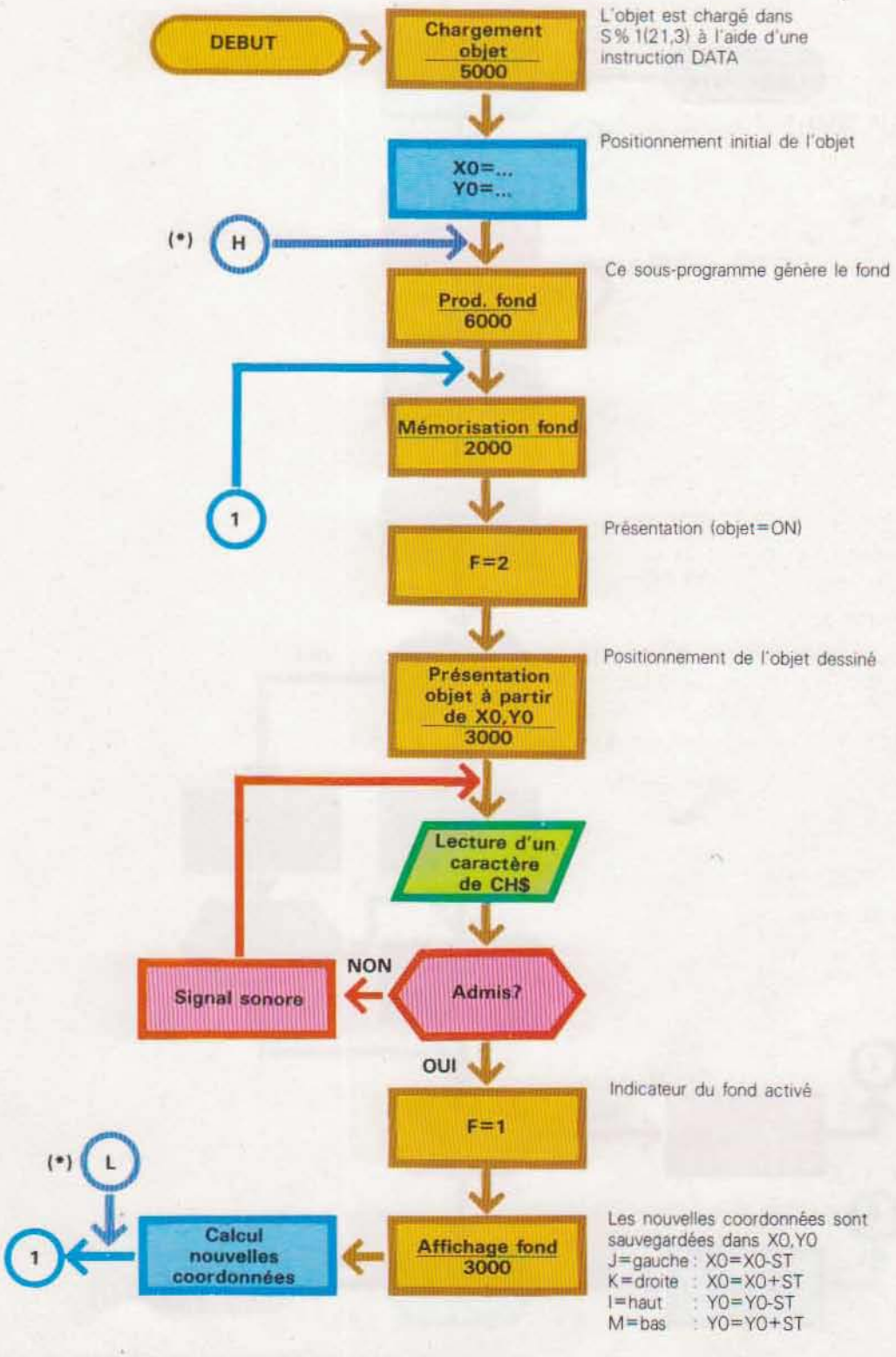
Noter que le point de référence (point X0,Y0) est distant de lui-même des quantités N=0, M=0. Par conséquent :

1e indice=21-0=21
2e indice =1
ne du bit =7

PRESENTATION DU FOND OU DE L'OBJET



EXEMPLE DE GESTION DES OBJETS ANIMES



(*) : Il sera fait référence aux connecteurs H et L plus loin.

EXEMPLE DE GESTION DES OBJETS ANIMES PAR LE LOGICIEL

```
10 REM ------------------------
20 REM * PROGRAMME PRINCIPAL *
30 REM ------------------------
40 DIM S%(21,3),BI%(21,3)
50 ST=5
60 GOSUB 5000
   :REM Chargement de l'objet animé
70 X0=10:Y0=180
80 GOSUB 6000:REM Dessin du fond
90 GOSUB 2000
   :REM Lecture de la fenêtre

100 F=2
110 GOSUB 3000
    :REM Ecriture de l'objet animé
120 GET CH$
130 IF CH$<>"I" AND CH$<>"J" AND
     CH$<>"M" AND CH$<>"K"
     AND CH$<>"CHR$(27) THEN 120
140 F=1
150 GOSUB 3000
    :REM Ecriture de la fenêtre
160 IF CH$="I" THEN Y0=Y0-ST
170 IF CH$="M" THEN Y0=Y0+ST
180 IF CH$="J" THEN X0=X0-ST
190 IF CH$="K" THEN X0=X0+ST
200 IF CH$=CHR$(27) THEN END

210 GOTO 90

1000 REM ------------------------
1010 REM * CALCUL DES ADRESSES *
1020 REM ------------------------
1030 GY=INT(Y/8)
1040 IF Y<64 THEN MR=8232+(GY-8)*128
     :GOTO 1070
1050 IF Y<128 THEN MR=8232+(GY-8)*128
     :GOTO 1070
1060 MR=8272+(GY-16)*128
1070 A=Y-GY*8
1080 MY=MR+A*1024
1090 GX=INT(X/7)

1100 MX=MY+GX
1110 BIT=X-7*GX
1120 NN=PEEK(MX)
1130 BY=NN
1140 FOR KK=0 TO BIT
1150 MS=NN/2
1160 NN=INT(MS)
1170 NEXT
1180 IF MS<>NN THEN BT=1:GOTO 1200
     :REM Le point écran est allumé
1190 BT=0
     :REM Le point écran est éteint

1200 RETURN

2000 REM ------------------------
2010 REM * LECTURE DE LA FENETRE *
2020 REM ------------------------
2030 MD$="L"
2040 FOR M=0 TO 20
2050 FOR N=0 TO 23
2060 X=X0+N:Y=Y0-M
```

```
2070 GOSUB 1000
2080 I=21-M:REM Premier indice
2090 J=INT(N/8)+1:REM Deuxième indice

2100 K=J*8-N-1
     :REM Numéro du point écran
2110 BI%(I,J)=BI%(I,J)+BT*2^K
2120 NEXT N
2130 NEXT M
2140 RETURN

3000 REM --------------------------
3010 REM * ECRITURE DE LA FENETRE *
3020 REM --------------------------
3030 MD$="S"
3040 FOR M=0 TO 20
3050 FOR N=0 TO 23
3060 X=X0+N:Y=Y0-M
3070 I=21-M:REM cf. 2080
3080 J=INT(N/8)+1:REM cf. 2090
3090 K=J*8-N-1:REM cf. 2100

3100 IF F=2 THEN N0=S%(I,J):GOTO 3120
3110 N0=BI%(I,J)
3120 FOR N1=0 TO K
3130 N2=N0/2
3140 N0=INT(N2)
3150 NEXT N1
3160 IF N2=N0 THEN BB=0:GOTO 3180
3170 BB=1
3180 GOSUB 1000
3190 IF BT=BB THEN 3220

3200 IF BT>BB THEN POKE MX,BY-2^BIT
     :GOTO 3220
3210 POKE MX,BY+2^BIT
3220 NEXT N
3230 NEXT M
3240 RETURN

5000 REM --------------------------
5010 REM * CHARGEMENT OBJET ANIME *
5020 REM --------------------------
5030 FOR I=1 TO 21
5040 FOR J=1 TO 3
5050 READ S%(I,J)
5060 NEXT J,I
5070 DATA 0,0,0
5080 DATA 0,0,0
5090 DATA 0,0,0

5100 DATA 0,0,0
5110 DATA 0,0,0
5120 DATA 0,0,0
5130 DATA 0,0,0
5140 DATA 28,0,248
5150 DATA 62,1,252
5160 DATA 103,3,254
5170 DATA 199,7,158
5180 DATA 143,15,28
5190 DATA 30,15,28

5200 DATA 60,30,60
5210 DATA 60,30,108
5220 DATA 31,252,248
5230 DATA 15,248,144
```

```
5240 DATA 7,240,48
5250 DATA 0,0,0
5260 DATA 0,0,0
5270 DATA 0,0,0
5280 RETURN

6000 REM ----------------------
6010 REM * CHARGEMENT DU FOND *
6020 REM ----------------------
6030 HOME
6040 INPUT "Nom du fond ";NS$

6050 HGR:POKE -16302,0:HCOLOR=3
6060 PRINT CHR$(4);"BLOAD";NS$
     ;"A$2000"
6070 RETURN
```

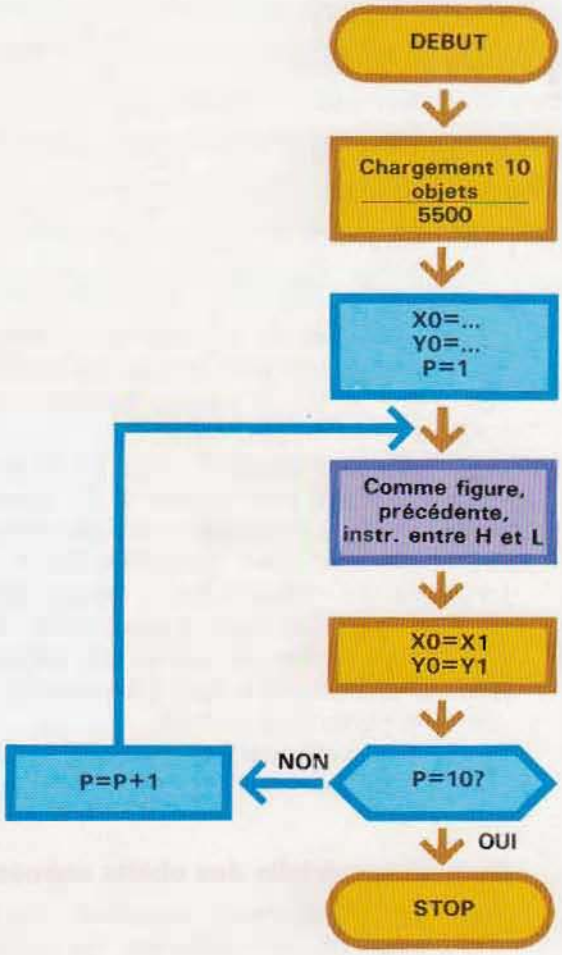
La seconde solution est la plus pratique, tant en raison de sa vitesse d'exécution que de la moindre occupation de l'espace mémoire. Elle n'est toutefois pas toujours applicable, les déplacements d'une figure étant difficiles à schématiser avec des formules mathématiques.
Quant à la première solution, elle ne nécessite qu'un compteur et une zone de mémoire contenant toutes les positions intermédiaires que l'on veut utiliser. Par exemple, si l'on range 100 positions intermédiaires dans T% (100,21,3), il suffit de les transférer (les unes après les autres, avec le compteur) dans S% (21,3) pour obtenir un mouvement également à l'intérieur de l'objet.
Une autre possibilité, enfin, consiste à ne dessiner que les positions initiales et finale, et éventuellement quelques intermédiaires si les deux précédentes sont très différentes. On laissera à l'ordinateur le soin de calculer tous les passages intermédiaires nécessaires à l'animation. Cette technique, qui présente des aspects extrêmement complexes, est encore à l'étude mais elle sera de plus en plus employée pour les productions cinématographiques.
On trouvera, à la page suivante, l'organigramme donné en page 1666, modifié de façon à permettre également l'animation à l'intérieur de l'objet. Cette animation s'obtient en préparant une dizaine de formes qui seront successivement présentées. Le déplacement de l'objet étant commandé par touches, l'animation est ralentie par la phase d'entrée. Pour obtenir un résultat plus réaliste, on définit préalablement un parcours qui sera exécuté automatiquement. La méthode la plus simple, pour décrire ce parcours consiste à définir un tableau (à 10 valeurs) dans lequel seront introduits (à l'aide d'une instruction DATA) les codes correspondant aux touches de déplacement. L'instruction de lecture sera alors activée par l'instruction CH$ = SP (P), SP étant un tableau contenant les codes de placement.
Une boucle dans laquelle P va de 1 à 10 simulera les pressions consécutives de 10 touches de déplacement et génèrera automatiquement le parcours défini dans l'instruction DATA.
La gestion des collisions est un élément indispensable pour ce type d'applications. Elle consiste à chercher si, lors de son déplacement, un objet animé en rencontre un autre ou une zone particulière du fond.
Ce contrôle s'effectue par lecture de la zone mémoire occupée par l'objet : il y a collision si elle est active.

Gestion matérielle des objets animés

Beaucoup d'ordinateurs personnels, certes plus proches des jeux vidéo que des applications professionnelles, sont équipés de composants matériels dédiés à la gestion des objets animés.
Ces composants peuvent mémoriser, déplacer et contrôler simultanément plusieurs objets. Il ne reste, à l'utilisateur, qu'à produire ces objets et le fond, puis à donner les instructions de mouvement, sans avoir à s'occuper ni d'effacement ni de reconstruction.
Dans ce type de machine, chaque objet animé se compose d'un ensemble de points divisés en rangées et colonnes ; on peut constituer la forme désirée en allumant certains d'entre eux.

Mémorisation de l'objet animé. Pour chaque objet il est prévu plusieurs centaines de points écran. Par exemple, la valeur utilisée par le Commodore 64 est de 504 points divisés en 21 rangées de 24 colonnes. L'occupation

GESTION DES OBJETS AVEC ANIMATION INTERNE



Les objets préparés (10) sont des formes différentes de la même figure. Présentées les unes après les autres, elles permettent d'animer l'intérieur du rectangle. Les 10 formes sont mémorisées dans T(10,21,3) à l'aide d'une instruction DATA

P est le compteur réglant la succession des 10 formes de l'objet

La seule différence se situe au niveau du sous-programme 3000, qui doit concerner le fond, alors qu'un autre (3700) gère l'objet.

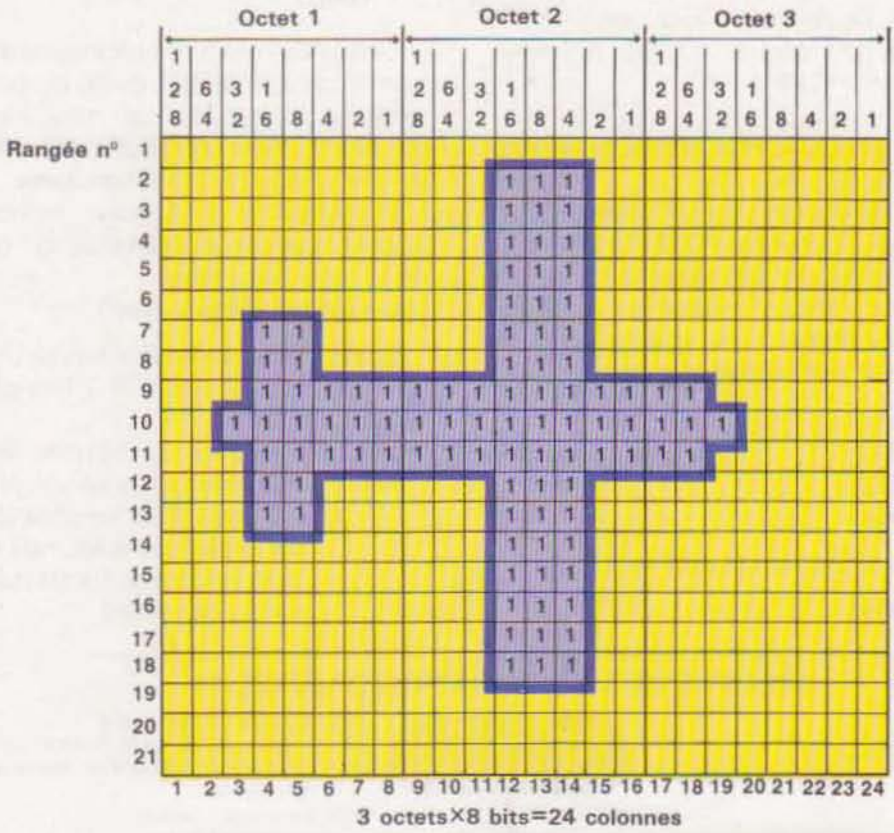
Le sous-programme 3700 est identique au 3000, à ceci près :
— une boucle initiale transfère T%(P,21,3) dans S%(21,3)
— le test de F et le bloc d'extraction du bit K de B1%(I,J) sont éliminés ; le sous-programme n'est plus paramétré et n'a donc pas à gérer le fond

en mémoire est donc de 504 bits soit 63 octets. La construction d'un objet consiste à mémoriser, dans 63 cellules de mémoire contiguës, l'état allumé/éteint de chacun des 504 bits. Pour allumer un point, il faut écrire, dans l'octet correspondant, une valeur décimale particulière qui active le bit sélectionné en le mettant à 1. Par exemple, pour activer le point se trouvant au croisement entre la première rangée et la première colonne, il faut écrire la valeur 128, tandis que pour la colonne 4, cette valeur sera 16 (ne pas oublier que les valeurs décimales correspondant à la fonction du bit sont 1, 2, 4, 8, 16, 32, 64, 128). Les valeurs pouvant être additionnées en écrivant le chiffre décimal 7, on active simultanément les bits se trouvant aux positions 1, 2 et 3.

Dans le schéma ci-contre, la 1ère rangée ne contient aucun point actif : les points qui lui sont associés sont donc 0, 0, 0 (octets 1, 2 et 3 respectivement). A la 2e rangée, sont actifs les points 12, 13, 14 qui appartiennent à l'octet 2 et valent 16+8+4=28 ; elle est donc représentée par le groupe 0, 28, 0.

De même, pour le calcul des valeurs de la rangée 10, on a :

SCHEMA DE MEMORISATION D'UN OBJET



3 octets×8 bits=24 colonnes

Rangée n°	Valeurs		
	Octet 1	Octet 2	Octet 3
1	0	0	0
2	0	28	0
3	0	28	0
4	0	28	0
5	0	28	0
6	0	28	0
7	24	28	0
8	24	28	0
9	31	255	192
10	63	255	224
11	31	255	192
12	24	28	0
13	24	28	0
14	0	28	0
15	0	28	0
16	0	28	0
17	0	28	0
18	0	28	0
19	0	0	0
20	0	0	0
21	0	0	0

Points actifs de l'octet 1 : 3, 4, 5, 6, 7, 8 = =32+16+8+4+2+1=63
Points actifs de l'octet 2 : tous, soit =255
Points actifs de l'octet 3 : 17, 18, 19 = =128+64+32=224

La rangée 10 est donc représentée par le groupe de valeurs 63, 255, 224.
L'archivage de la table ainsi constituée est réalisé à l'aide de l'instruction POKE. En supposant que la mémorisation commence à la position 896 (valeur prise uniquement à titre d'exemple), il faudra 3 instructions pour mémoriser la première rangée (une instruction par octet) :

POKE 896,0	1er octet
POKE 897,0	2e octet
POKE 898,0	3e octet

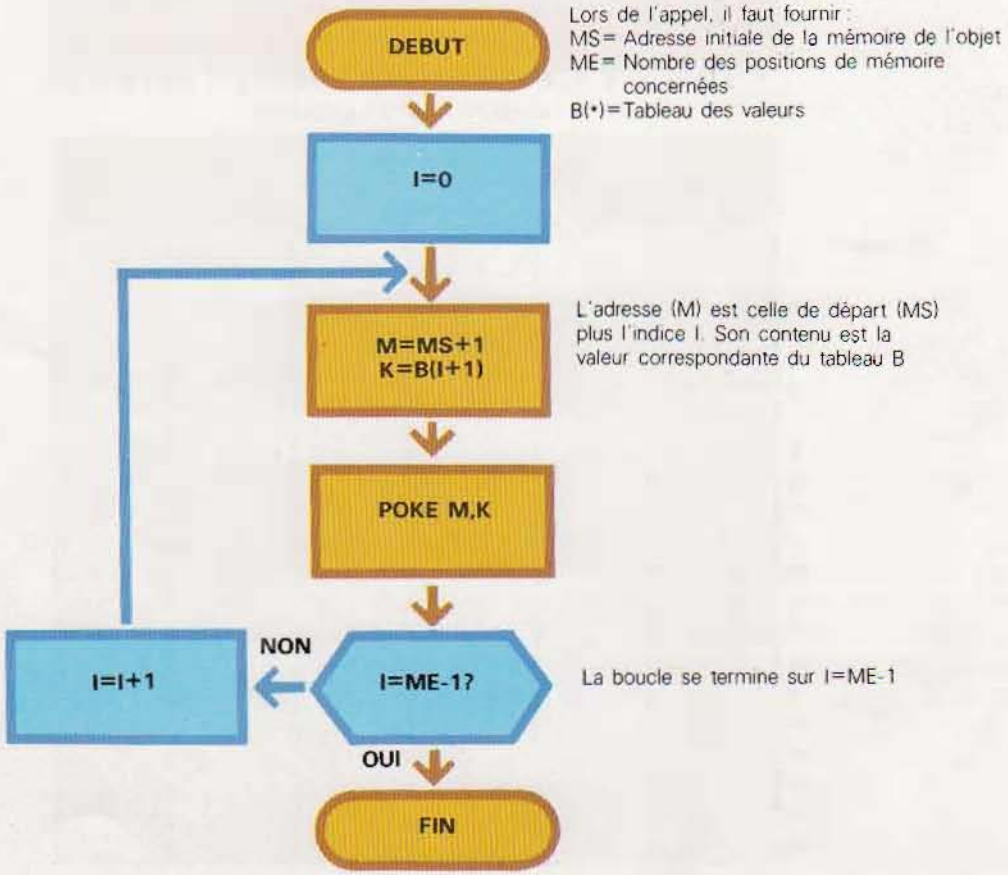
tandis que la 2e rangée sera entrée à l'aide de :

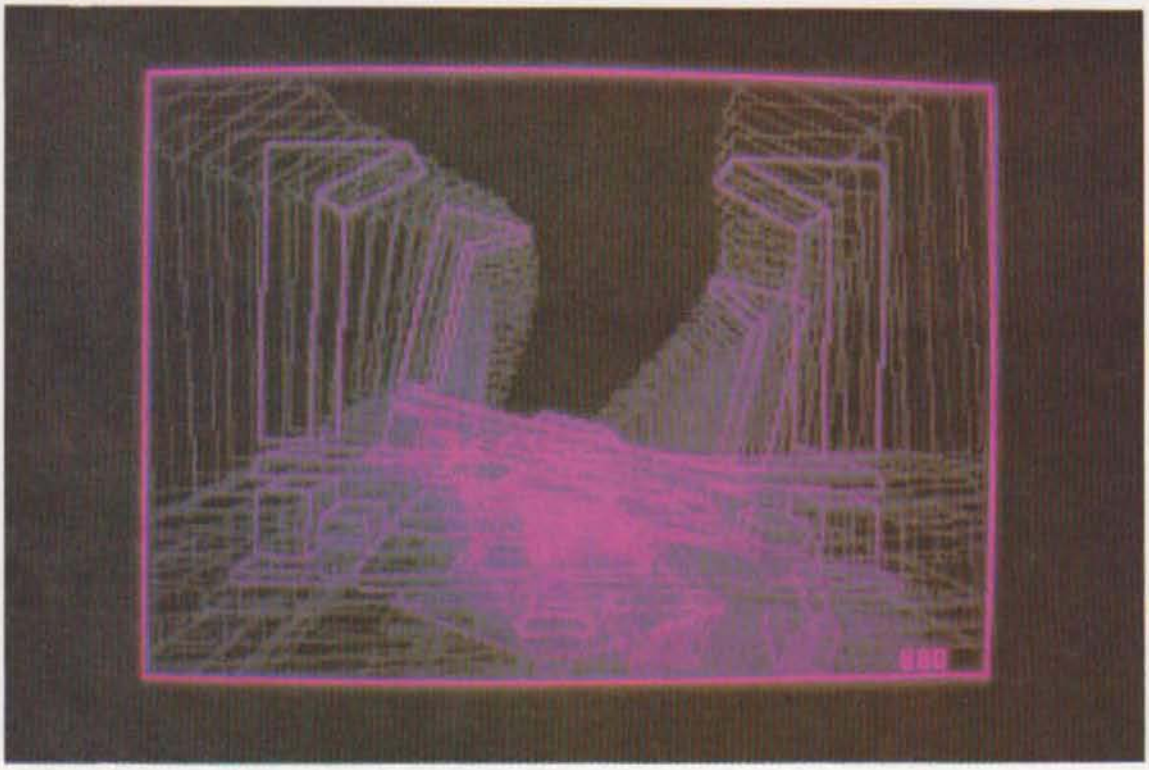
POKE 899,0
POKE 900,28
POKE 901,0

Il n'est évidemment pas nécessaire d'écrire autant d'instructions qu'il existe de positions de mémoire à activer. La meilleure solution consiste à utiliser une boucle depuis la position de départ jusqu'à la position finale.
L'organigramme ci-dessous représente un sous-programme de mémorisation de la table de la page précédente.
Les variables utilisées sont :

B (63) = Contient les 3 octets de chaque rangée (3 octets × 21 rangées = 63 octets)
MS = Position de mémoire de départ
ME = Nombre de positions de mémoire à activer. Pour l'ensemble de la table, il faut entrer ME=63 (64 si l'on tient compte de l'octet nul qui achève la description).

EXEMPLE DE MEMORISATION D'UN OBJET





P.A. Simon/Grazia Neri-Photodake

Une image animée réalisée pour le film « Tron ».

La table de difinition est rangée dans n'importe quelle zone de la mémoire vive (RAM) et pas nécessairement à proximité immédiate des définitions d'autres objets.
En effet, un registre pointeur permettra de retrouver chacune d'entre elles. Dans le Commodore 64, qui gère jusqu'à 8 objets différents, les pointeurs des définitions des objets occupent les positions 2040 à 2047.
Dans les applications, il est toutefois utile de réserver, à la mémorisation des tables de description des objets, une zone unique divisée en autant de blocs de 64 octets qu'il y a d'objets à gérer.
En supposant que la zone mémoire réservée aux objets commence à la position 832, l'objet défini précédemment (et chargé à la position 896) se trouvera dans le bloc 2. Après son activation, il faudra donc écrire la valeur 14 (12 + 2) à la position réservée à l'adressage (2040).
Le système peut ainsi retrouver sa position en effectuant la multiplication 14×64= 896.

Visualisation de l'objet animé. Une fois mémorisée la table, on doit activer la logique de visualisation de l'objet. La gestion des objets (présentation, déplacement, etc.) se fait généralement en activant ou en désactivant des positions de mémoire spéciales et préprogrammées.
Le système étant capable d'accepter plusieurs objets il faut lui indiquer sur lequel on désire travailler. Cette fonction est généralement activée en mettant à 1 un bit d'une position mémoire particulière. Par exemple, en écrivant 3 (décimal, c'est-à-dire 11 binaire) à cette adresse, on déclare actifs les objets n° 0 et n° 1.
Dans le Commodore 64, le registre de contrôle de la visualisation des objets se trouve à l'adresse 53269. Si le bit n° 0 de ce registre est mis à 1, la visualisation de l'objet n° 0 est activée. Il en sera de même pour tous les autres objets.
Toutefois il n'y aura visualisation que si le point de l'écran où se trouve l'objet est spécifié. Pour cela, chaque objet est contrôlé par deux registres dans lesquels il est nécessaire d'entrer les coordonnées (en points écran) du point désiré. Par exemple, la position de l'objet 0 est contrôlée par les registres 53248 et 53249 .
Dans le premier, c'est la valeur de la coordon-

née X qui devra être chargée (POKE), alors que dans le second c'est celle de la coordonnée Y. La visualisation de l'objet 0 (mémorisé dans le bloc 2) au point 100, 150 s'obtiendra par :

```
30 POKE 2040,2
40 POKE 53248,100
50 POKE 53249,150
60 POKE 53269,1
```

Déplacement de l'objet. Pour déplacer l'objet visualisé, il faut, à l'aide d'une boucle, faire varier les coordonnées de sa posotion contenues dans les deux registres de contrôle.
Dans le cas considéré, le déplacement horizontal de l'objet 0, s'obtiendra par :

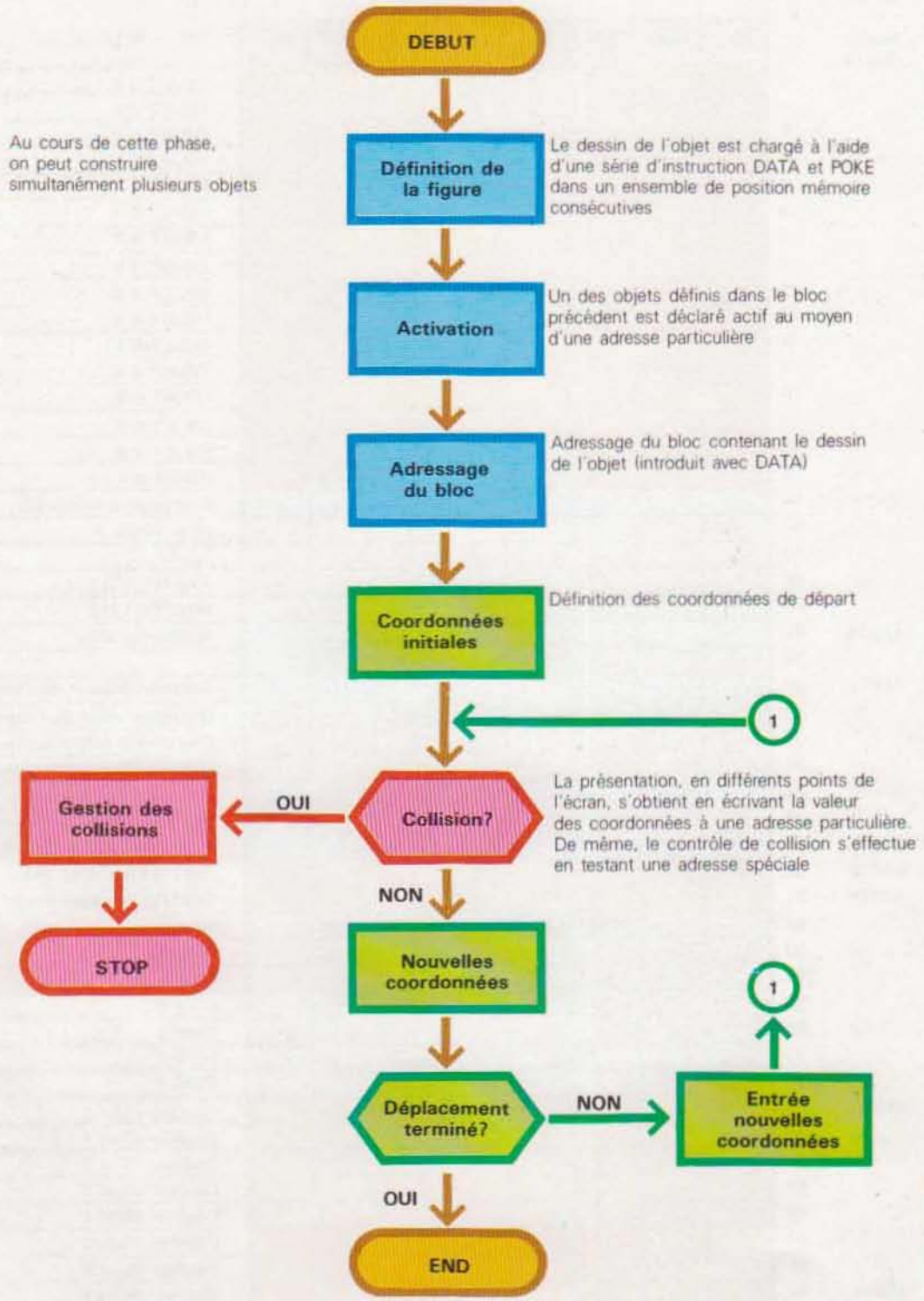
```
100 POKE 53249,100
110 FOR I = 50 TO 150
120 POKE 53248,I
130 NEXT I
```

La ligne 100 fixe la valeur de Y, tandis que la boucle attribue les valeurs 50 à 150 à la coordonnée X.
Il est souvent nécessaire d'associer un contrôle de collisions au déplacement d'un objet. Là encore, il existe des positions mémoire spéciales dont le contenu indique si une collision s'est produite.
Dans le Commodore 64, la collision entre deux objets est contrôlée par le registre 53278 et celle objet-fond par le registre 53279 (voir organigramme ci-contre).

Utilisation de la mémoire pour la gestion des objets animés. La zone mémoire du Commodore 64 chargée de la gestion des objets est schématisée en page 1676 ; elle commence à la position 53248.
Les 16 premières cellules sont les registres de positionnement des 8 objets que la machine peut gérer et dans lesquelles seront chargées les coordonnées de leur positionnement.
Vient ensuite un registre (53264) qui stockera les bits les plus significatifs des coordonnées X (1 bit par objet). Les 4 octets suivants ne concernent pas les objets animés. Le registre d'activation/désactivation est à l'adresse 53269 (adresse relative 21). En mettant à 1 un bit de ce registre, on active l'objet correspondant.
Aux adresses 53271 et 53277 se trouvent deux registres qui permettent de doubler les dimensions de l'objet, le premier le long de l'axe X et le second le long de l'axe Y, en mettant à 1 le bit correspondant (0 pour l'objet 0, 1 pour l'objet 1, etc).
Le registre 53275 établit la priorité d'un objet par rapport au fond. Par exemple, si le bit n° 3 de ce registre a la valeur 0, l'objet 3 a la priorité sur le fond (c'est-à-dire qu'il couvrira le fond). S'il vaut 1, ce sera l'inverse.
Les positions 53278 et 53279 contiennent les indicateurs de collision objet-objet et objet-fond. Si l'objet 2 entre en collision avec l'objet 3, les bits 2 et 3 du registre 53278 seront mis à 1 ; il en sera de même en cas de contact avec le fond.
Pour finir, les registres 53287 à 53294 permettront de choisir la couleur des objets en chargeant les codes numériques correspondants (0 = blanc, 1 = noir, etc).
Les autres registres, assez nombreux, sont utilisés pour exécuter des fonctions particulières sur lesquelles nous ne nous arrêterons pas.
En revanche, voyons plus en détail la question de la mémorisation des objets.
Nous avons dit que la description d'un objet se compose d'une suite de 63 + 1 octets consécutifs (le dernier contenant toujours la valeur 0) mémorisés (instruction POKE) à partir d'une adresse de départ.
L'adresse de cette position doit ensuite être mémorisée dans le registre pointeur correspondant à l'objet en question.
En fait, ce qui y est réellement mémorisé ce n'est pas une adresse mémoire, mais un numéro d'ordre d'un bloc de mémoire de 64 octets. En multipliant cette valeur par 64, le système obtient l'adresse réelle à laquelle commence la description de l'objet.
Cette logique est détaillée page 1677.

Exemple d'application. Pages 1678 et 1680, on trouvera l'organigramme d'un jeu utilisant 4 objets animés multicolores.
Le jeu consiste à déplacer horizontalement un parachutiste afin d'éviter les collisions avec une série d'objets qui apparaissent de façon aléatoire. Le jeu est gagné si le parachutiste réussit à toucher le sol.
Pour compliquer sa descente et faire perdre son contrôle au joueur, un vent latéral déplacera le parachute de gauche à droite.

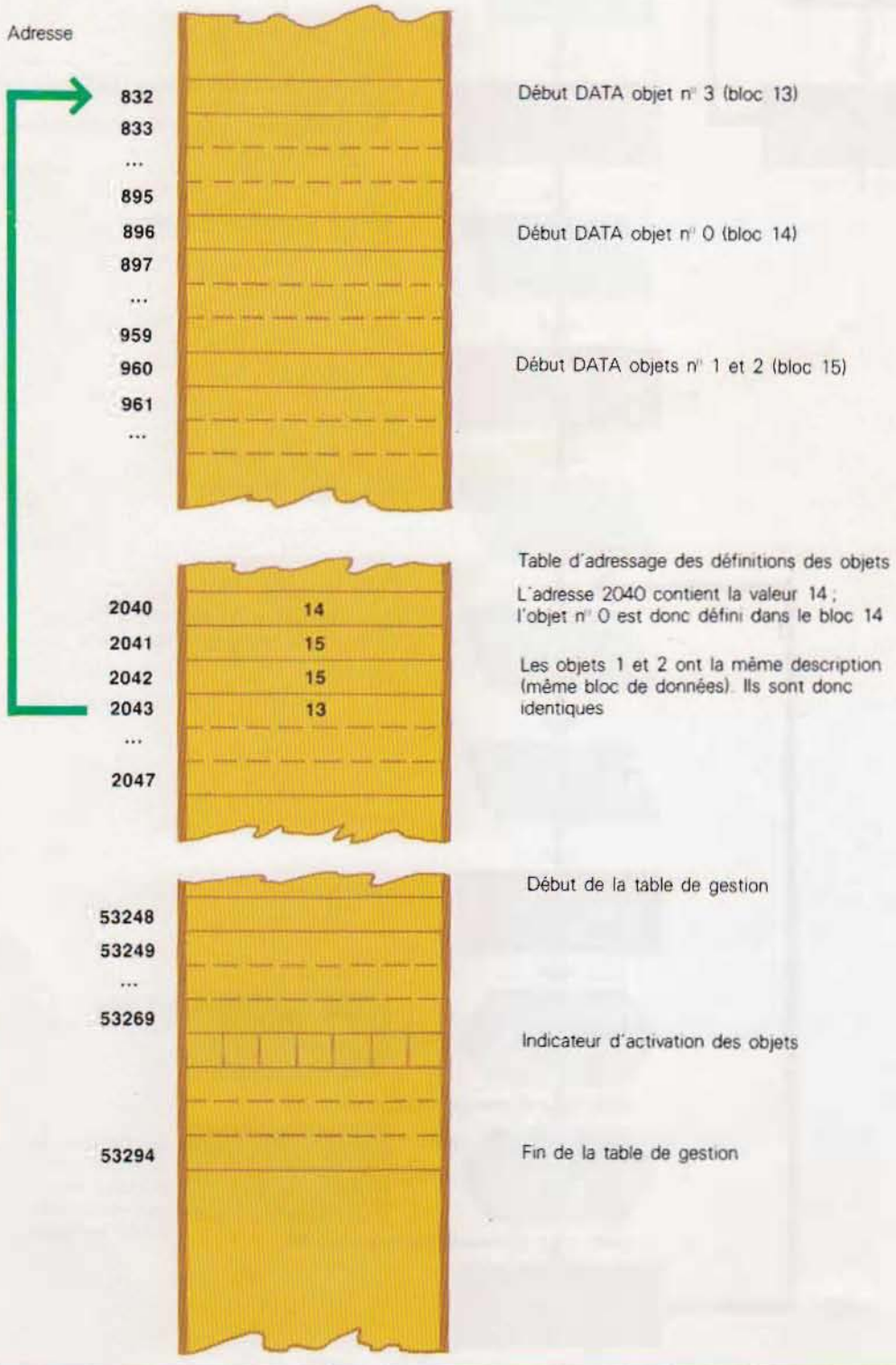
SCHEMA LOGIQUE DE GESTION DES OBJETS ANIMES



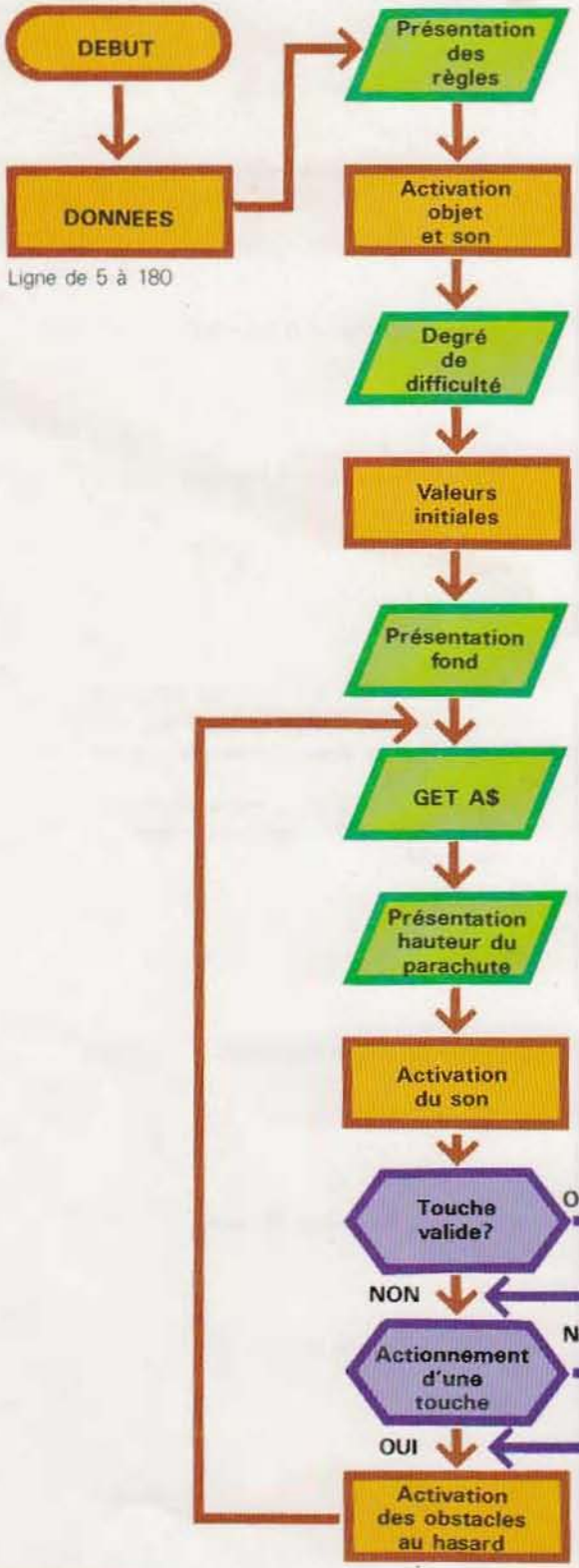
ORGANISATION DES REGISTRES DES OBJETS DANS LE COMMODORE 64

Bit n°		7	6	5	4	3	2	1	0	
Base = 53248		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
	0									OBJET 0 X
	1									OBJET 0 Y
	2									OBJET 1 X
	3									OBJET 1 Y
	4									OBJET 2 X
	5									OBJET 2 Y
	6									OBJET 3 X
	7									OBJET 3 Y
	8									OBJET 4 X
	9									OBJET 4 Y
	10									OBJET 5 X
	11									OBJET 5 Y
	12									OBJET 6 X
	13									OBJET 6 Y
	14									OBJET 7 X
	15									OBJET 7 Y
53264	16									Bits de plus fort poids
	17									de la valeur X
	18									TABLE
	19									PHOTOSTYLE X
	20									PHOTOSTYLE Y
53269	21									Validation objet
	22									
53271	23									Augmentation Y de l'objet
	24									Mémoire vidéo de l'objet
	25									Demandes d'interruption
	26									MASQUES interruptions
53275	27									Priorité fond de l'objet
	28									Sélection objet multicolore
53277	29									Augmentation X de l'objet
53278	30									COLLISION Objet-Objet
53279	31									COLLISION Objet-Fond
	32			INFORMATIONS COULEUR						Marge
	33									Fond 0
	34									Fond 1
	35									Fond 2
	36									Fond 3
	37						Objets multicolores			OMC 0
	38									OMC 1
53287	39									Couleur objet 0
	40									Couleur objet 1
	41									Couleur objet 2
	42									Couleur objet 3
	43									Couleur objet 4
	44									Couleur objet 5
	45									Couleur objet 6
53294	46									Couleur objet 7

UTILISATION DE LA MEMOIRE DANS LA GESTION DES OBJETS ANIMES



ORGANIGRAMME DU PROGRAMME PARACHUTES



Gestion évoluée des objets animés

Les objets animés se retrouvent dans d'autres applications graphiques que les jeux vidéo, par exemple dans la préparation de symboles élémentaires servant à former des dessins complexes. Cet emploi particulier nécessite toutefois un logiciel de gestion évolué permettant d'appeler les fonctions nécessaires directement en Basic, sans s'occuper de questions aussi complexes que l'adressage de la mémoire ou que l'affectation des tables.

Les machines les plus modernes, notamment dans la catégorie des ordinateurs personnels, acceptent des versions de Basic avancé qui prévoient non seulement des instructions graphiques, mais également une gestion des objets animés. Les principales fonctions implémentées sont les suivantes :

— Définition des objets animés comme variables
— Présentation
— Contrôle des collisions

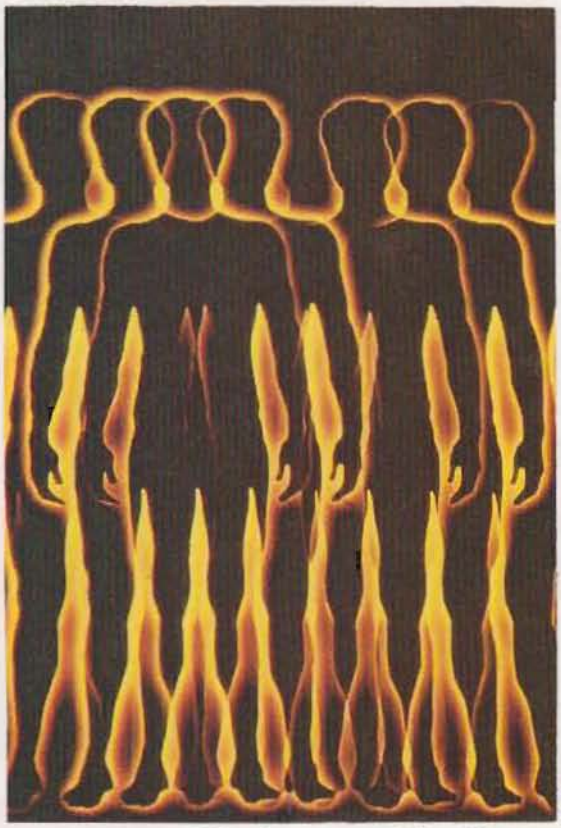
Définition des objets animés. La méthode reste la même que celle qui vient d'être décrite. Elle consiste à réserver une zone de mémoire dans laquelle l'image binaire de l'objet est transférée. Mais, contrairement aux formes précédentes, chaque objet est, dans ce cas, défini comme une variable de chaîne et sera donc chargé à l'aide d'une seule instruction DATA, sans passer par l'écriture de tables d'adressage.

Voici une forme très employée de cette instruction :

```
SPRITE$(N)=B$
```

Elle définit l'objet N comme égal au contenu de la variable de chaîne B$, qui est à son tour chargée au moyen d'une instruction DATA.

Cette instruction, comme les suivantes, est propre au système Philips VG8000, mais fonctionne également sur tous les ordinateurs utilisant le Basic Standard MSX. Cette forme de Basic est très proche du Basic 80 sous CP/M, tout en étant plus riche en instructions spécifiques. Le Basic MSX est d'une diffusion récente mais, du fait de sa remarquable facilité d'emploi (notamment dans les applications graphiques) ; il a été adopté dans la plupart des machines basées sur le microprocesseur Z80.



Marka

Effet de superposition d'images produit par l'ordinateur.

Visualisation. La forme la plus simple de cette instruction est

```
PUT SPRITE P, (X, Y), C, N
```

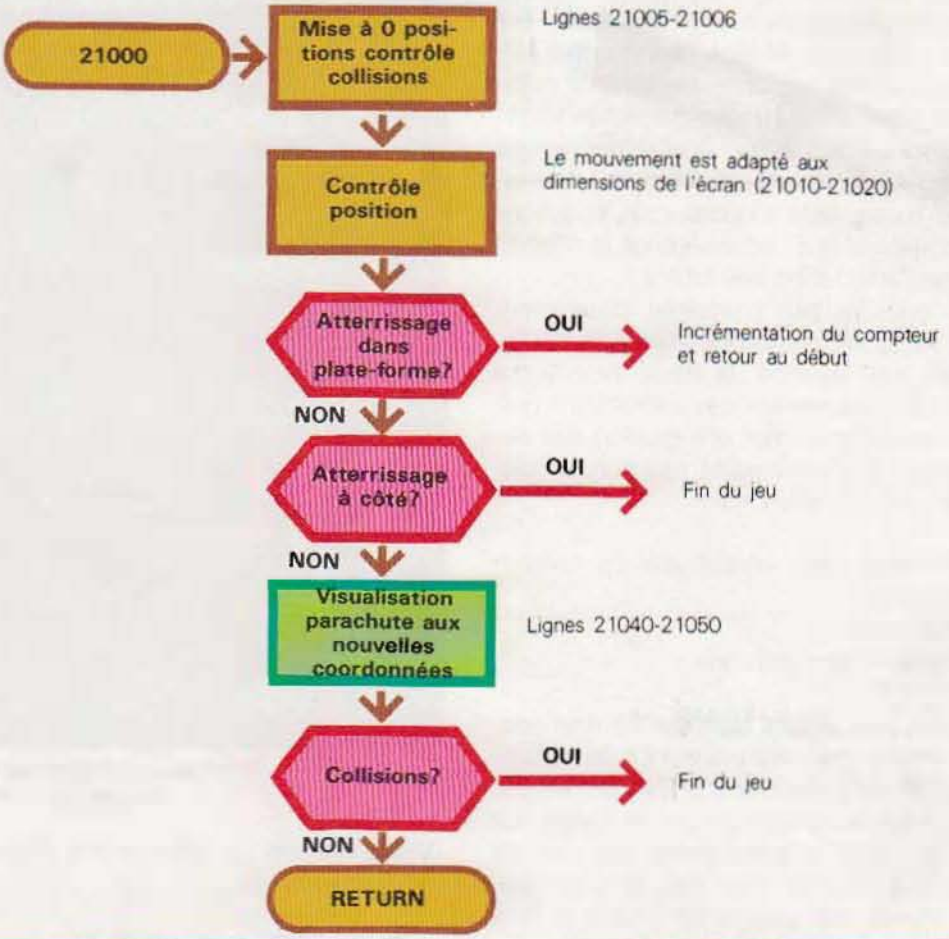
dans laquelle :

P : Indique la priorité de l'objet (pour éviter d'éventuels conflits en cas de présentation de deux objets au même point de l'écran)
X, Y : Sont les coordonnées
C : Est le nombre qui désigne la couleur choisie, de 0 à 15
N : Est le n° d'identification de l'objet, défini dans l'instruction SPRITE$(N)

La syntaxe présentée est, là aussi, propre au système Philips, également utilisable avec les ordinateurs acceptant le Basic MSX. Mais cette instruction n'établit pas les dimensions de l'objet, qui doivent être préalablement définies au moyen de l'instruction SCREEN,

```
SCREEN A, B, C, D
```

SOUS-PROGRAMME DE DEPLACEMENT ET DE CONTROLE



CARACTERES PARTICULIERS UTILISES DANS LE PROGRAMME PARACHUTE VERSION CMB 64

CHR $	Fonction	CHR $	Fonction
147	Vider l'écran	144	Ecrire en noir
18	ON inversé	31	Ecrire en bleu
146	OFF inversé	5	Ecrire en blanc
159	Ecrire en bleu clair	17	Déplacer le curseur vers le bas
28	Ecrire en rouge		

Positions de mémoires

650,128	Répétition automatique
53248+28,8	Définition objet multicolore
53248+42 53248+37 53248+38	Couleurs de l'objet
53248+29,1	Doubler la dimension de l'objet n° 0 (décimal 1) sur l'axe X
53278 53279	Collisions

Instruction d'activation de l'objet N.''n'':
POKE S+21, PEEK (S+21) AND ''n''

EXEMPLE D'UTILISATIONS DES OBJETS ANIMES CBM 64

```
5 REM *** Micro Resque CBM 64 ***
10 REM ** parachutes **
20 DATA 0,170,0,2,170,128,2,40,128,14,0,176
30 DATA 47,130,248,43,235,232,170,255,170,170,190,170
40 DATA 186,170,174,168,0,42,161,0,74,1,0,64
50 DATA 0,113,0,0,243,0,0,40,0,0,40,0
60 DATA 0,60,0,0,60,0,0,60,0,0,20,0,0,0,0
70 REM ** nuage **
80 DATA 0,60,0,0,255,0,3,255,128,15,191,192
90 DATA 63,255,240,255,255,184,239,254,126,243,253,255
100 DATA 253,243,255,127,247,255,62,111,255,28,63,239
110 DATA 0,31,222,0,15,204,2,7,248,12,15,224
120 DATA 16,31,128,8,127,192,1,239,224,0,126,112,0,56,0
130 REM ** faucon **

140 DATA 192,0,3,240,0,15,254,0,127,255,129,255
150 DATA 255,129,255,255,129,255,127,195,254,63,231,252
160 DATA 15,231,240,7,255,224,0,255,0,0,126,0
170 DATA 0,60,0,0,90,0,0,126,0,0,213,0
180 DATA 1,129,128,3,0,192,6,0,96,10,0,80
300 REM ** présentation **
301 GOSUB 3000
310 PRINT CHR$(147):POKE 53281,1:CLR
315 PRINT CHR$(18);CHR$(159);"                                    ";
320 PRINT CHR$(18);CHR$(28);"            Micro Resque             ";
325 PRINT CHR$(18);CHR$(159);"                                    ";
330 PRINT CHR$(17);CHR$(144)"Vous devez faire atterrir un groupe de"
331 PRINT "parachutistes sur le tremplin."
340 PRINT "Attention toutefois à ne pas heurter"
350 PRINT "les mines (;"CHR$(120);") pendant la descente."
360 PRINT "Et attention à éviter aussi les"
370 PRINT "buses géantes et les nuages toxiques."
380 PRINT "Le parachutiste descendra seul":PRINT "sous l'effet de la gravité"
390 PRINT "mais vous devrez le guider à droite et":PRINT "à gauche"
400 PRINT CHR$(17);CHR$(18);CHR$(31)"Touches de direction :";CHR$(146);CHR$(144)
410 PRINT TAB(20);"(";CHR$(18);CHR$(28);"X";CHR$(146);CHR$(144);")... droite"
430 PRINT TAB(20);"(";CHR$(18);CHR$(28);"Z";CHR$(146);CHR$(144);")... gauche"
435 PRINT TAB(12);CHR$(17);CHR$(17);CHR$(18);"F1 pour commencer"
440 GET A$:IF A$CHR$(133) THEN 440
443 PRINT TAB(12);CHR$(18);CHR$(28);"Un instant... "
445 POKE 650,128:REM ** répétition automatique pour toutes les touches **
450 REM ** adresse VIC-II **
460 S=53248
480 GOSUB 1000
490 GOSUB 3000
500 GOSUB 2000
545 PRINT CHR$(147);CHR$(144):INPUT "Difficulté :";DF
550 GOTO 5000:REM ** bloc principal **
1000 REM ** lecture et activation des objets animés **
1010 FOR D=0 TO 62:REM ** lecture parachutes **
1020 READ W
1030 POKE 832+D,W:REM ** objet n°3 bloc 13 **
1040 NEXT D
1050 FOR D=0 TO 62:READ W:REM ** lecture nuage **
1060 POKE 960+D,W:REM ** objet n°0 bloc 14 **
1070 NEXT D
1080 FOR D=0 TO 62:READ W:REM ** lecture faucon **
1090 POKE 960+D,W:REM ** objet n°1E2 bloc 15 **
1100 NEXT D
1120 REM ** activation des objets animés **
1140 POKE S+21,15:REM ** (décimales = 1+2+4+8) **
1150 POKE 2040,14:REM ** nuage au bloc 14 **
1160 POKE 2041,15
1170 POKE 2042,15:REM ** faucons au bloc 15 **
1180 POKE 2043,13:REM ** parachutistes au bloc 13 **
1200 RETURN
2000 REM ** définition des objets animés **
2010 POKE S+28,8:REM ** objet n°3 multicolore **
```

```
2020 POKE S+42,8:REM ** bit "10" orange **
2030 POKE S+37,0:REM ** bit "01" noir **
2040 POKE S+38,1:REM ** bit "11" blanc **
2050 REM ** objet n°0 noir et double en X **
2060 POKE S+39,0:POKE S+29,1
2070 REM ** objet n°1 couleur au hasard **
2080 POKE S+40,INT(RND(1)*2)+1
2090 REM ** objet n°2 couleur au hasard **
2100 POKE S+41,INT(RND(1)*7)+7
2140 RETURN
3000 REM ** activation registres du son **
3010 FOR RS=54272 TO 54296:POKE RS,0:NEXT RS
3020 POKE 54274,0:POKE 54275,8:POKE 54278,240:POKE 54277,0
3030 POKE 54296,15
3040 RETURN
4999 REM *** bloc principal ***
5000 SP=0:CS=0:X1=20:X2=20:X3=255
5001 REM
5002 PRINT CHR$(147):PRINT CHR$(142);:GOSUB 10000
5003 POKE 53278,0:POKE 53279,0
5005 X=INT(RND(1)*150)+100:Y=70:GOSUB 21000
5006 V1=0:V2=0:V3=0:NC=0
5010 GET A$
5011 IF Y<100 THEN B$=" ":GOTO 5013
5012 B$=" "
5013 PRINT CHR$(19);CHR$(18);CHR$(158);"Aterris: ";PAR;
5014 PRINT TAB(23);CHR$(18);CHR$(5)"Altitude: ";(200-Y);CHR$(157);B$;
5015 LB=8:HB=97:HZ=129:GOSUB 30000
5020 A=PEEK(197)
5030 IF A=23 THEN X=X+5:Y=Y+3:GOSUB 21000
5040 IF A=12 THEN X=X-5:Y=Y+3:GOSUB 21000
5045 IF A=64 THEN GOSUB 22000
5050 NC=INT(RND(1)*10)+1
5055 NC1=INT(RND(1)*50)+1
5070 IF NC=20RNC=60RNC=9 THEN GOSUB 23000
5080 IF NC=30RNC=70RNC=10 THEN GOSUB 23100
5085 IF NC1=17 AND Y>159 THEN 35000
5090 IF NC=40RNC=8 THEN GOSUB 23200
5210 GOTO 5010
10000 REM ** fond **
10003 PRINT CHR$(147);:POKE 53281,14:POKE 53280,6
10020 FOR MI=1 TO (DF*3)
10025 W=INT(RND(1)*500)+1
10030 POKE 1224+W,88:POKE 55496+W,INT(RND(1)*2)+1
10035 NEXT MI
10036 FOR I=1 TO 21:PRINT CHR$(17);:NEXT I
10037 PRINT CHR$(18);CHR$(30);"                    ";
10038 PRINT CHR$(18);CHR$(5);"     ";CHR$(30);"              ";
10040 FOR I=1 TO 2:PRINT CHR$(18);CHR$(30)"                    ";

10042 NEXT I
10043 PRINT CHR$(19)
10050 RETURN
21000 REM ** contrôle des coordonnées et visualisation de l'objet n°3
21005 POKE 53278,0
21006 POKE 53279,0
21010 IF X<25 THEN X=X+5:RETURN
21020 IF X>240 THEN X=X-5:RETURN
21030 IF X>170 AND X<176 AND Y>197 THEN 24000
21035 IF (Y>197 AND X<170) OR (Y>197 AND X<176) THEN 24100
21040 POKE S+6,X
21050 POKE S+7,Y
21060 REM ** collisions **
21065 IF (PEEK(53278) AND 9)=1 THEN RETURN
21070 IF (PEEK(53278) AND 9)=9 THEN 24500
21080 IF (PEEK(53278) AND 10)=10 THEN 24520
21090 IF (PEEK(53278) AND 12)=12 THEN 24520
21095 IF (PEEK(53279) AND 8)=8 THEN 26000
21097 IF PEEK(53279)<>8 THEN RETURN
21100 REM
```

```
21105 POKE 53278,0
21110 POKE 53279,0
21150 RETURN
22000 REM ** chute libre **
22010 Y=Y+3
22020 CS=INT(RND(1)*2)+1
22030 IF CS=1 THEN X=X+3
22040 IF CS=2 THEN X=X-3
22050 GOSUB 21000
22100 RETURN
23000 REM ** mouvement du nuage **
23005 Y1=INT(RND(1)*50)+100
23007 LB=1:HB=12:HZ=65:GOSUB 30000
23010 X1=X1+INT(RND(1)*10)+20
23012 IF X1>=24 AND V1>4 THEN SP=254:GOSUB 25000
23014 IF X1>=24 AND V1<4 THEN X1=20:GOTO 23005
23020 POKE S+0,X1:POKE S+1,Y1:GOSUB 21060
23035 RETURN
23100 REM ** mouvement buse 1 **
23102 IF V2<>0 THEN 23115
23107 LB=1:HB=12:HZ=65:GOSUB 30000
23110 Y2=INT(RND(1)*130)+50:V2=V2+1
23115 X2=X2+INT(RND(1)*10)+10
23117 IF X2>240 AND V2>4 THEN SP=253:GOSUB 25000
23119 IF X2>=240 AND V2<4 THEN X2=20:GOTO 23110
23120 POKE S+2,X2:POKE S+3,Y2:GOSUB 21060
23125 RETURN
23200 REM ** mouvement buse 2 **
23202 IF V3<>0 THEN 23215
23204 Y3=INT(RND(1)*150)+50:V3=V3+1
23207 LB=1:HB=12:HZ=65:GOSUB 30000
23215 X3=X3-10
23217 IF X3<=35 AND V3>4 THEN SP=251:GOSUB 25000
23219 IF X3<=35 AND V3<4 THEN X3=255:GOTO 23204
23220 POKE S+4,X3:POKE S+5,Y3:GOSUB 21060
23235 RETURN
24000 REM ** il a atterri **
24030 GOSUB 60000
24050 PRINT CHR$(19);CHR$(18);CHR$(144)"     Parachute atterri     ";
24055 PRINT TAB(15);CHR$(18);CHR$(158);"Félicitations"
24059 FOR T=1 TO 800: NEXT T
24060 PRINT CHR$(147):PAR=PAR+1:POKE 53279,247:GOTO 5001
24100 REM ** hors cible **
24105 PRINT CHR$(19);CHR$(18);CHR$(5);"     Hors cible     ";
24110 GOTO 35065
24500 REM ** collision avec d'autres objets **
24505 POKE 53278,0:SP=247:GOSUB 25000:SP=251:GOSUB 25000:SP=253:GOSUB 25000
      :SP=254:GOSUB 25000
24510 PRINT CHR$(147);CHR$(18);CHR$(5);"Le parachutiste est entré dans"
24515 PRINT CHR$(18);CHR$(5);"le nuage de ";CHR$(158);"trioxine";CHR$(5)
      ;"et est mort"
24516 PRINT CHR$(17);CHR$(5);"Total atterris: ";CHR$(18);CHR$(158);PAR
24517 LB=134:HB=24:HZ=65:FOR T=1 TO 3:GOSUB 30000:NEXT T
24519 GOTO 50000
24520 POKE 53278,0:SP=247:GOSUB 25000:SP=251:GOSUB 25000:SP=253:GOSUB 25000
      :SP=254:GOSUB 25000
24525 PRINT CHR$(147);CHR$(18);CHR$(5);"Le parachutiste a été"
24526 PRINT CHR$(18);CHR$(5);"tué par une";CHR$(158);" buse géante"
24528 LB=1:HB=12:HZ=17:FOR T=1 TO 3:GOSUB 30000:NEXT T
24529 PRINT CHR$(17);CHR$(15);"Total atterris: ";CHR$(18);CHR$(158);PAR
24530 GOTO 50000
25000 REM ** désactivateur **
25010 POKE S+21,PEEK (S+21) AND SP
25020 RETURN
26000 REM ** choc avec les mines **
26005 SP=251:GOSUB 25000:SP=247:GOSUB 25000:SP=253:GOSUB 25000:SP=254
      :GOSUB 25000
26010 PRINT CHR$(147);CHR$(18);CHR$(5);"Le parachutiste a heurté une "
26012 PRINT CHR$(158);CHR$(18);"mine anti-personnel"
26015 LB=1:HB=12:HZ=65:FOR T=1 TO 3:GOSUB 30000:NEXT T
```

```
26020 GOTO 50000
26025 PRINT CHR$(17);CHR$(5);"Total atterris: ";CHR$(18);CHR$(158);PAR
30000 REM ** sous-programme son **
30010 POKE 54276,HZ
30020 POKE 54272,LB
30030 POKE 54273,HB
30040 FOR T=1 TO 5:NEXT T
30050 POKE 54272,0:POKE 54273,0
30060 RETURN
35000 REM ** vent **
35010 SP=254:GOSUB 25000:SP=253:GOSUB 25000:SP=251:GOSUB 25000
35020 FOR K=X TO 255 STEP 9
35030 PRINT CHR$(19);CHR$(17);CHR$(18);CHR$(5);"Vent de 60 noeuds !!"
35035 LB=134:HB=24:HZ=65:GOSUB 30000
35040 POKE S+6,K:POKE S+7,Y
35045 PRINT CHR$(19);CHR$(17);"                "
35060 FOR T=1 TO 100:NEXT T:NEXT K
35065 FOR I=1 TO 7:PRINT CHR$(17):NEXT I:PRINT TAB(14);CHR$(18);CHR$(144)
      ;"Tu es mort !"
35070 SP=247:GOSUB 25000
35080 PRINT CHR$(19);CHR$(5)
35100 END
50000 REM
50010 PRINT CHR$(17);CHR$(5);"Une autre partie (O/N) ?";
50020 INPUT R$
50030 IF R$="O" THEN RUN
50040 PRINT TAB(15);CHR$(17);CHR$(18);CHR$(158);"Au revoir"
50050 FOR T=1 TO 1000:NEXT T:SYS770
60000 REM ** bruit atterrissage **
60005 POKE 54276,65
60010 POKE 54272,33:POKE 54273,134:POKE 54272,35:POKE 54273,132
60020 POKE 54272,37:POKE 54273,161:POKE 54272,42:POKE 54273,60
60030 POKE 54272,44:POKE 54273,191:POKE 54272,47:POKE 54273,104
60035 POKE 54272,47:POKE 54273,104:POKE 54272,44:POKE 54273,191
60040 POKE 54272,42:POKE 54273,60:POKE 54272,37:POKE 54273,60
60045 POKE 54272,35:POKE 54273,132:POKE 54272,33:POKE 54273,134
60060 POKE 54272,0:POKE 54273,0
60100 RETURN
```

Deux moments de l'exécution du programme « Parachutes ».



Les paramètres de cette instruction ont la signification suivante :

A : Mode d'utilisation de l'écran

A=1 texte, 40 caractères par ligne (cpl)
A=2 texte, 32 cpl
A=3 mode graphique 1 (256×192 pixels)
A=4 mode graphique 2 (même nombre de pixels, les figures étant constituées de rectangles de 4×4 pixels)

B : Dimension des objets animés ; ses valeurs vont de 0 (objets de 8×8 pixels) à 3 (32×32 pixels)

C : Quand il est à 1, il se produit un cliquetis chaque fois qu'une touche est actionnée

D : Vitesse de transmission lors de l'échange de données avec l'enregistreur. Les valeurs admises sont 1 (1200 baud) et 2 (2400 baud)

Contrôle des collisions. Les instructions de gestion de cette fonction sont :

SPRITE ON/OFF/STOP
pour activer, désactiver, exclure le contrôle
ON SPRITE GOSUB...
appelle le sous-programme spécifié quand une collision se produit.

Le programme ci-dessous et à la page suivante illustre l'emploi de ces instructions.
On se rend compte immédiatement de l'extrême facilité d'écriture des programmes qu'apporte ce type de Basic.

GESTION DES OBJETS ANIMES SUR MSX

```
4 CLS:CLEAR
5 COLOR 5,4,1
7 PLAY "L406CDEFEDCREFGAGFEREDEDGAG"
10 SCREEN 2,2
11 GOSUB 9000
13 C1=15:C2=1:C3=11:P=0
15 REM ** objet araignée **
20 DATA 1,3,7,63,79,159,47,79
25 DATA 147,161,35,5,3,2,4,2
30 DATA 128,192,224,252,242,249,244,242
35 DATA 201,133,196,160,192,64,32,64
40 REM ** objet phalène haut et bas **
45 DATA 0,0,1,1,3,7,15,31
50 DATA 63,119,235,221,255,127,61,24
55 DATA 0,0,128,128,192,224,240,248
60 DATA 252,238,215,187,255,254,188,24
65 REM ** phalène gauche **
70 DATA 0,0,0,1,3,7,15,63
75 DATA 63,15,7,3,1,0,0,0
80 DATA 56,124,238,233,191,222,235,254
85 DATA 254,235,222,191,223,238,124,56
90 REM ** phalène droite **
95 DATA 28,62,119,251,253,123,55,127
100 DATA 127,55,123,253,251,119,62,28
105 DATA 0,0,0,128,192,224,240,252
110 DATA 252,240,224,192,128,0,0,0
140 REM ** objet oeufs **
145 DATA 0,0,1,3,3,7,15,31
150 DATA 31,31,31,15,7,0,0,0
155 DATA 0,0,128,192,192,224,240,248
160 DATA 248,248,248,240,224,0,0,0
170 A$=" ":B$=" ":C$=" ":D$=" ":E$=" "
200 FOR I=1 TO 32:READ A:A$=A$+CHR$(A):NEXT I
205 SPRITE$(1)=A$
210 FOR I=1 TO 32:READ B:B$=B$+CHR$(B):NEXT I
215 SPRITE$(2)=B$
220 FOR I=1 TO 32:READ C:C$=C$+CHR$(C):NEXT I
225 SPRITE$(3)=C$
230 FOR I=1 TO 32:READ D:D$=D$+CHR$(D):NEXT I
235 SPRITE$(4)=D$
240 FOR I=1 TO 32:READ E:E$=E$+CHR$(E):NEXT I
245 SPRITE$(5)=E$
263 X=100:Y=50:PUT SPRITE 2,(X,Y),C1,2
264 X=1 INT(RND(1)*250)+1:Y1=INT(RND(1)*100)+50:PUT SPRITE 1,(X1,Y1),C2,1
265 J=STICK(1)
```

```
267 IF J=1 THEN Y=Y-7:GOSUB 5000
268 IF J=2 THEN X=X+7:Y=Y-7:GOSUB 5000
269 IF J=5 THEN Y=Y+7:GOSUB 5000
270 IF J=4 THEN X=X+7:Y=Y+7:GOSUB 5010
271 IF J=6 THEN X=X-7:Y=Y+7:GOSUB 5020
272 IF J=3 THEN X=X+7:GOSUB 5010
275 IF J=7 THEN X=X-7:GOSUB 5020
276 IF J=8 THEN X=X-7:Y=Y-Y:GOSUB 5020
280 IC=INT(RND(1)*4)+1:IF IC=1 THEN X1=X1-20
282 IF IC=2 THEN X1=X1+20
283 IF IC=3 THEN Y1=Y1-20
284 IF IC=4 THEN Y1=Y1+20
290 GOSUB 1000
295 GOSUB 4000
297 IF X1=X OR Y1=Y THEN 3000
310 ON SPRITE GOSUB 2000
320 SPRITE ON
405 T=T+1
900 GOTO 265
1000 REM ** mouvement araignée **
1003 IF X1>250 THEN X1=X1-20:RETURN
1004 IF X1<10 THEN X1=X1+20:RETURN
1005 IF Y1>190 THEN Y1=Y1-20:RETURN
1006 IF Y1<10 THEN Y1=Y1+20:RETURN
1010 PUT SPRITE 1,(X1,Y1),C2,1
1050 RETURN
3000 REM
3005 SPRITE OFF
3010 PLAY "L20ZBDDC"
3017 COLOR 15,1,1
3018 LOCATE 15,10
3020 PRINT "Tu es mort"
3030 LOCATE 19,12
3030 PRINT "Total points: ";P
3050 LOCATE 1,1
3060 INPUT "Une autre partie (O/N) ";
3070 R$=INKEY$:IF R$="" THEN 3060
3080 IF R$="O" THEN RUN
3200 END
4000 REM ** apparition oeuf **
4010 X2=INT(RND(1)*250)+1
4015 Y2=INT(RND(1)*160)+1
4017 IF T>50 THEN 4070
4019 IF T<1 AND T<50 THEN RETURN
4020 PUT SPRITE 5,(X2,YZ),C3,5
4065 RETURN
4070 T=0:RETURN
5000 REM
5001 IF X>253 OR Y>187 OR X<0 OR Y<0 THEN RETURN
5002 PUT SPRITE 3,(X,209),C1,3:PUT SPRITE 4,(X,209),C1,4
5006 PUT SPRITE 2,(X,Y),C1,2:RETURN
5010 REM
5011 IF X>253 OR Y>187 OR X<0 OR Y<0 THEN RETURN
5013 PUT SPRITE 2,(XX,209),C1,2:PUT SPRITE 3,(X,209),C1,3
5016 PUT SPRITE 4,(X,Y),C1,4
5018 RETURN
5020 REM
5022 IF X>253 OR Y>187 OR X<0 OR Y<0 THEN RETURN
5023 PUT SPRITE 2,(X,209),C1,2:PUT SPRITE 4,(X,209),C1,4
5026 PUT SPRITE 3,(X,Y),C1,3
5029 RETURN
9000 REM ** dessin du fond **
9003 LINE (0,0)-(126-98),1:LINE (128,102)-(255,191),1
9005 LINE (128,89)-(255,0),1:LINE (126,102)-(0,191),1
9007 CIRCLE (126,100),2,1
9010 LINE (126,100)-(0,75),1:LINE (128,100)-(255,75),1
9013 LINE (126,101)-(0,125),1:LINE (128,101)-(255,125),1
9015 LINE (127,98)-(86,0),1:LINE (129,98)-(171,0),1
9017 LINE (125,102)-(86,191),1:LINE (129,102)-(171,191),1
9020 FOR I=S TO 140 STEP 6 CIRCLE (126,100),1,1:NEXT I
10000 RETURN
```



Marka

GLOSSAIRE

A

Accès. Opération d'entrée/sortie sur l'unité de mémoire de masse (disque). Elle est réalisée en deux phases : la recherche et la lecture ou l'écriture. Dans le cas d'un accès disque, qui n'est normalement pas déclenché, une autre phase vient s'y ajouter : le temps nécessaire pour atteindre la vitesse de rotation correcte. C'est d'ailleurs cette phase qui fait la différence entre deux systèmes.

Accès aléatoire (Direct). Méthode de lecture ou d'écriture dans une zone quelconque de la mémoire. Cette méthode ne permet pas de prévoir quelle sera la prochaine position adressée. Elle peut s'appliquer à une zone mémoire RAM ou à un fichier.

Accès direct (ou aléatoire). Méthode permettant d'exécuter directement la fonction d'accès à une donnée. Un fichier est à accès direct (ou aléatoire) quand un enregistrement peut être lu ou écrit indépendamment des autres enregistrements.

Accès séquentiel. Méthode de lecture/écriture qui suit l'ordre normal des enregistrements sur le support (disque, bande magnétique ...). Lors d'une opération d'écriture, chaque élément est physiquement positionné après l'élément précédent ; lors de la lecture, l'ordre de sortie des données est le même que l'ordre d'introduction. Il s'oppose donc à l'accès direct (ou aléatoire).

Accès sériel. Expression généralement synonyme d'accès séquentiel. L'accès à une donnée n'est réalisé qu'une fois traitées les données précédentes. Dans un fichier séquentiel, par exemple, un enregistrement n'est lu ou écrit qu'après que les enregistrements antérieurs ont été effectivement traités.

Accumulateur. Position de mémoire ou registre dans lequel sont temporairement rangés les résultats d'un calcul.

Activation. Opération qui lance l'exécution d'une procédure ou d'un sous-programme.

Adaptateur. Interface de nature mécanique qui établit une liaison entre des dispositifs ayant des connecteurs différents.

ADM. Accès Direct à la Mémoire (en anglais : DMA, Direct Memory Access). Transfert de données entre un périphérique et la mémoire sans passer par l'unité centrale.

Adressage calculé. Technique d'adressage consistant à convertir la clé de l'enregistrement en une valeur dont l'adresse est déduite.

Adressage indirect. Méthode d'adressage selon laquelle une position contenant une nouvelle adresse est indiquée. La donnée désirée se trouve à l'adresse ainsi spécifiée.

Adresse. Registre, position de mémoire ou dispositif générique. Une adresse se compose généralement de 8 bits, mais elle peut en avoir 16 ou 32 dans les plus grosses machines. Sa longueur indique l'extension de mémoire utilisable par le système.

Adresse d'une piste. Adresse à laquelle une donnée devrait logiquement se trouver. Ce terme ne s'utilise qu'avec les gros systèmes pour lesquels la gestion des données prévoit une zone dite de débordement. Cette zone est utilisée pour stocker temporairement les enregistrements quand les zones qui leur sont allouées sont occupées.
Il peut également s'agir d'une zone d'adressage (toujours avec une mémoire de masse).

Affectation. Transfert d'une valeur dans une variable. (La valeur précédente est généralement perdue).

Afnor. Association française de normalisation.

Aiguillage. Indicateur dont la valeur provoque une variation du flux de traitement.

Aléatoire. Qualifie une méthode d'adressage par référence à la position. Synonyme d'**Accès direct**.

Aléatoire. Qualifie des nombres produits au hasard à l'aide d'un algorithme particulier contenu dans presque tous les langages.

Algol. Algorithmic Oriented Language. Langage de programmation à usage scientifique, généralement utilisé dans les grands systèmes.

Algorithme. Série de règles permettant de résoudre un problème avec un nombre fini d'opérations de passages.

Alimentation (module d'alimentation). Circuit permettant de convertir une tension de manière à l'adapter aux niveaux requis par d'autres circuits. Il convertit, par exemple, un courant alternatif en courant continu.

Allocation. Désigne l'opération par laquelle une ressource est attribuée (par exemple un périphérique ou une zone de mémoire) à un utilisateur particulier ou à un programme (dans les systèmes autorisant la multiprogrammation).

ALU. Voir **UAL**.

Amorce. Version de base d'un programme capable de s'automodifier. Désigne également un programme capable de s'autocharger. L'amorce est généralement le premier programme chargé dans un ordinateur ; il permet à son tour de charger le système d'exploitation. Il arrive, très souvent, que l'amorce soit résidente et automatiquement activée lors de la mise en marche de la machine.

Amplitude. Valeur d'un signal à un instant déterminé.

Analogique. Qualifie un dispositif dont le fonctionnement est fondé sur l'analogie. Les calculateurs analogiques, à l'inverse des calculateurs numériques (les plus répandus), utilisent tous les niveaux

possibles d'un signal. Dans ce contexte, même les données sont représentées par ces niveaux.

APL. A Programming Language. Langage de programmation à usage scientifique.

Appel. Opération permettant de lancer un programme ou un sous-programme. C'est également, dans bon nombre de langages, un code reconnu (CALL).

Appel. Activation d'une procédure ou d'un sous-programme d'un de ses points de lancement.

APT. Automatically Programmed Tools. Langage de programmation spécial, utilisé pour préparer des données destinées à la commande numérique.

Area. Voir **Zone**.

Arbre. Structure logique ramifiée qui permet, en partant d'un point quelconque, de remonter à l'origine (le parcours suivi est univoque).

Argument. Valeur d'une variable affectée à une fonction fournie par le système ou par l'utilisateur. Dans certains cas, elle est fictive (dummy) et n'a qu'une utilité formelle. L'instruction BASIC FRE(n), par exemple, qui sert à connaître l'espace mémoire disponible, utilise « n » comme argument fictif. Le résultat obtenu sera valide quelle que soit la valeur de « n ».

Array. Voir **Tableau**.

Arrière-plan. Zone de l'écran entourant une figure ou un caractère (voir **Premier plan)**.

Arrondi. Méthode permettant d'éliminer quelques chiffres d'un nombre. Elle permet, en particulier, d'obtenir le nombre le plus proche d'une valeur réelle à gérer par l'ordinateur.

ASCII. American Standard Code for Information Interchange. C'est le code le plus usité pour l'échange d'informations entre un ordinateur et ses périphériques. Dans certains cas, il est également utilisé comme code de mémorisation des données ou de programmes sur disque.

Assemblage. Langage très proche de la logique de la machine, mais qui utilise néanmoins des codes mnémoniques. Il permet d'obtenir une version exécutable d'un programme.

Assembleur. Programme qui traduit, en langage machine, les instructions en langage d'assemblage.

Asynchrone. Désigne un processus dans lequel l'axe du temps n'est pas découpé d'une manière régulière. Qualifie le plus souvent une forme particulière de transmission de données ou un processus dont l'activation n'est pas liée à l'exécution des instructions d'un programme.

Attribut. Propriété caractéristique d'un ou de plusieurs éléments. Un point de l'écran peut avoir, comme attribut, sa couleur ; une zone de données d'un fichier peut avoir, comme attribut, son caractère numérique, le nombre de chiffres la composant ou le nombre de positions décimales de cette valeur.

Autohide. Voir **Elimination des faces cachées**.

Autoplacement. Fonction d'informatique graphique appliquée à la conception des circuits électroniques et permettant de déplacer un composant afin d'optimiser la conception du circuit.

Avant. En parlant d'un dérouleur de bande ou d'un fichier séquentiel, indique le sens de défilement vers l'arrière, de point actuel vers le début. Dans le cas des dérouleurs de bande, indique le rebobinage après lecture.

Avant-plan. Zone de l'écran occupée par un dessin ou par un caractère (voir aussi **Arrière-plan)**.

B

Background. Voir **Arrière plan**.

Backspace. Voir **Rappel arrière**.

Balayage. Opération par laquelle une zone déterminée est explorée séquentiellement. C'est notamment ce que réalise le faisceau électronique d'un terminal vidéo sur l'écran (voir **Balayage récurrent)**.

Balayage récurrent (TV). Méthode de formation de l'image basée sur l'excitation de certains points de l'écran au fur et à mesure que le faisceau électronique le parcourt. Cette méthode est semblable à celle qui est utilisée pour la formation des images de télévision.

Bande perforée. Ruban de papier sur lequel sont enregistrées des données, sous forme de perforations. La présence d'un trou à une position déterminée indique l'état du bit correspondant.

Banque de données. Ensemble de données relatives à un même sujet.

Barker (Code Barker). Forme particulière de codification dans la transmission des données.

Base de données. Ensemble de données en rapport les unes avec les autres, et pouvant servir à différentes applications. Dans les systèmes évolués, le type de mémorisation utilisé permet de les rendre indépendantes du programme qui les exploite. L'ensemble des programmes nécessaires à leur gestion prend le nom de SGBD (Système de Gestion de Base de Données). Ces programmes utilisent généralement des langages spécialisés pour leur description et leur manipulation.

Base de numération. Unité de base d'un système de numération. Les plus utilisées sont la base décimale (base 10), binaire (base 2), octale (base 8) et hexadécimale (base 16).

Batch. Voir **Traitement par lots**.

Baud. Unité d'expression du nombre d'états ou de conditions se produisant en une seconde lors d'une transmission des données. 1 200 bauds indiquent que la transmission s'effectue à une vitesse égale à 1 200 bauds par seconde (un état, ou une condition peut être un bit). Dans les systèmes utilisant 10 bits pour transmettre un caractère, la vitesse de transmission est égale à 1 200/10 = 120 caractères par seconde.

Baudot (Code Baudot). Code de transmission de données antérieur au code ASCII et qui utilise 5 bits par caractère.

Bibliothèque. Index des programmes (source, objet) généralement utilisés par le système d'exploitation.

Binaire. Se dit d'un dispositif ne pouvant avoir que deux états ou valeurs. Fait référence au système de numération en base 2.

Bistable. Circuit ayant deux états stables. Un signal spécial permet de le faire passer de l'un à l'autre et d'obtenir ainsi un compteur en base 2.

Bisynchrone. Méthode de communication permettant de repérer des erreurs multiples.

Bit. Contraction de binary digit (numérotation binaire). Information élémentaire dans un système de numération binaire. Il peut avoir les valeurs 0 ou 1.

Bit de parité. Bit de contrôle dont la présence ou l'absence indique si les bits précédents, qui constituent la donnée, doivent être en nombre pair ou impair.

Blanc. Espace non occupé par des caractères. Dans un ordinateur, un espace blanc est considérée comme un caractère ; il existe donc un caractère « blanc ». A ne pas confondre avec l'espace qui indique un véritable espacement. Ce dernier terme désigne généralement une position de la tête d'écriture d'une imprimante ou du curseur, mais il ne correspond à aucune donnée en mémoire. A l'inverse, « blanc » peut indiquer une quantité mise en mémoire.

Blinking. Voir **Clignotement**.

Blocking. Voir **Groupage**.

BOM. Bill Of Materials. Terme utilisé dans les applications de conception et de fabrication assistées sur ordinateur (CFAO) pour désigner la liste des composants nécessaires à la réalisation d'un appareil.

Bogue. Issu de l'anglais « bug » (« vermine ») et désignant une erreur contenue dans un programme. Termes dérivés : déboguer (mise au point) qui est l'opération de recherche d'erreurs dans un programme avant d'être opérationnel. Un programme spécial est déroulé pour ce faire. La mise au point permet normalement de localiser les failles en examinant, instant par instant, le contenu de la mémoire et l'état des variables.

Booléen. Se dit d'une valeur numérique interprétable comme « vraie » ou « fausse ». L'état vrai est généralement représenté par une valeur non nulle et par zéro. Il peut toutefois arriver qu'une logique inverse soit adoptée.

Bootstrap. Voir **Amorce**.

Boucle. Ensemble d'instructions exécutables récursivement jusqu'à ce qu'une condition déterminée soit vraie.

Boucle de courant. Procédé physique de transmission de données. Le signal est constitué d'un courant généralement compris entre 20 et 50 mA.

Boule roulante. Dispositif d'entrée constitué d'une sphère que l'on fait rouler pour déplacer le curseur. Son fonctionnement est identique à celui du manche à balai.

BPI. Bits Per Inch. Unité de mesure de densité d'enregistrement sur ruban magnétique (bit par pouce). Elle indique la quantité de données binaires contenues dans 27,5 mm de ruban.

BPS. Bits par seconde. Vitesse de transmission en bits. Pour obtenir la vitesse en caractères, il faut diviser la valeur en BPS par le nombre de bits contenues dans un caractère (par 8, si le caractère est un octet, par exemple).

Bruit. Signal, d'origine non déterminée, qui se superpose au signal utile et qui crée des perturbations.

B-Spline. Représentation mathématique d'une courbe plane.

Bubble Sort. Voir **Tri « à bulle »**.

Bucket. Voir **Compartiment**.

Buffer. Voir **Mémoire tampon**.

Bug. Voir **Bogue**.

By-pass. Elément de circuit évitant le passage du courant électrique dans d'autres circuits.

Byte. Voir **Octet**.

C

Cabestan. Mécanisme de réglage de la vitesse dans les enregistrements magnétiques des gros systèmes.

Câblé. Adjectif qualifiant un ensemble de liaisons électriques de transfert et de traitement des signaux.

Canal. Liaison par laquelle transitent les données.

CAO (anglais CAD, Computer Aided Design). Conception Assistée par Ordinateur. Procédé de conception informatisé. Cette expression désigne également un système, compris comme ensemble

matériel et logiciel, orienté vers la résolution de problèmes graphiques et de conception à l'aide de l'ordinateur.

Capacité. Appliqué à la mémoire, ce terme indique sa taille, généralement exprimée en Koctets. Plus généralement, désigne la capacité d'un circuit à emmagasiner une charge électrique, ou encore ce circuit lui-même. En cas d'utilisation de courant alternatif, il sert de pont. La capacité se mesure en micro-farads.

Caractère. Tout symbole utilisé comme partie de données ou pour leur commande et leur organisation. Un caractère n'est pas forcément alphabétique puisque sont également considérés comme caractères des codes spéciaux utilisés pour la gestion des périphériques ou pour la commande d'échanges avec d'autres systèmes (le code ASCII en présente tout un tableau).

Caractère de sollicitation. Symbole émis par l'ordinateur dans l'attente d'une donnée ou d'une instruction.

Caractère alphabétique. Lettre de l'alphabet. Les ordinateurs reconnaissent généralement l'alphabet latin (anglais).

Caractères alphanumériques. Ensemble de symboles constitué à la fois de lettres de l'alphabet et de chiffres, comme symboles et non pas comme valeurs.

Caractère de commande (ou de fonction). Tout caractère (au sens large, et non pas seulement alphabétique) utilisé pour exécuter des fonctions particulières comme l'activation, la modification ou l'exécution d'une opération de commande (transmission de données, enregistrement sur disque, etc.).

Carte de circuits imprimés. Support de réalisation de liaisons électriques. Elles sont fabriquées par photogravure d'une plaque d'un matériau isolant et par dépôt de cuivre.

Catalogue. Liste de tous les fichiers présents (données et programmes).

CCITT. Comité Consultatif International Télégraphique et Téléphonique. Organisme chargé de la normalisation internationale dans le domaine de la transmission.

Cellule. Groupe de positions de mémoire contiguës ayant généralement des contenus homogènes. Dans les bases de données, un ensemble de cellules est défini de telle sorte que son appellation ne recoupe pas celles des subdivisions mécaniques de la mémoire de masse, comme par exemple les secteurs.

CFAO. Conception et fabrication assistées par ordinateur.

Champ. Partie d'un enregistrement de données ayant un sens particulier. Dans un enregistrement d'un fichier d'identification, par exemple, il y a un champ nom de famille, un champ rue... (Voir **Zone**).

Chargeur (cartouche de disques). Ensemble de disques magnétiques utilisés uniquement dans les gros systèmes.

Checkpoint. Voir **Point de reprise**.

Chip. Voir **Puce**.

Circuit intégré. Circuit électronique miniaturisé dans un seul boîtier.

Clavier. Ensemble des touches, tout comme sur une machine à écrire, servant à entrer des données dans l'ordinateur.

Clé. Champ de données utilisé pour rechercher un enregistrement et pour y accéder. La clé peut être primaire, quand elle identifie un enregistrement de façon univoque, ou secondaire si elle est commune à plusieurs enregistrements.

Clé. Partie de l'enregistrement qui en permet l'identification. Dans les opérations de tri, la clé est le champ servant d'élément de comparaison entre les valeurs des enregistrements pour obtenir une sortie ordonnée.

CL File (Cutter Location File : fichier de positionnement de l'outil de coupe). Ensemble de données de sortie d'un système graphique constituant les coordonnées de commande d'une machine à commande numérique.

Clignotement. Variation de la luminosité d'un texte à l'écran. Cette forme de présentation sert à attirer l'attention de l'opération sur un message particulier. Le curseur à l'écran est souvent indiqué par un carré blanc clignotant.

Clipping. Voir **Ecrêtage**.

Clock. Voir **Horloge** et **Signal d'horloge**.

Cobol-Ansi. Langage de programmation développé par l'American National Standard Institute (ANSI) et orienté vers la gestion.

Cocktail Shaker Sort. Voir **Tri « à bulle »**.

Codasyl. (Conference of Data Systems Languages). C'est l'organisation chargée de mettre à jour les spécifications relatives au Cobol ou d'en émettre de nouvelles.

Code. Représentation de données ou de programmes intelligible pour l'ordinateur (par ex. le code ASCII). De ce terme, dérive « codage », qui désigne l'opération d'écriture d'un programme, c'est-à-dire la traduction des fonctions à exécuter sous forme binaire (voir **Indicateur**).

Code. Circuit servant à coder des données.

Code Huffman. Forme particulière de codage selon

laquelle les caractères les plus fréquemment utilisés sont représentés par un nombre de bits inférieur à la normale. Cette technique très complexe permet de diminuer l'espace occupé en mémoire de masse.

Collection de données. Ensemble de données groupées sous un nom collectif unique. Il en existe de deux types : les vecteurs et les groupes répétitifs. Un vecteur est un ensemble avec une seule dimension et des caractéristiques identiques. Le second type désigne un ensemble de données susceptible d'être présent dans le même enregistrement.

Commande. Directive demandant l'utilisation d'une fonction particulière du système d'exploitation ou du langage employé.

Commande numérique et **machines à commandes numériques.** Les commandes numériques sont des instructions mémorisées générées par les systèmes de FAO (fabrication assistée par ordinateur), par exemple pour piloter des machines à l'aide d'un ordinateur. Les machines à commandes numériques sont des machines de production automatisées ayant pour objet d'exécuter un travail particulier commandé par des instructions sous forme de données numériques, généralement produites pas les systèmes de FAO.

Commentaire. Ligne de programme ayant pour objet de fournir des explications. La possibilité d'inclure des lignes de commentaire existe avec tous les langages symboliques. Le compilateur est généralement capable de les éliminer de la version exécutable, de façon à ne pas occuper inutilement de l'espace mémoire (la version exécutable étant écrite en code numérique, les commentaires n'auraient donc aucun sens).

Communication. Opération de transmission ou de réception d'informations. La communication est directe dans le cas de distances faibles, par exemple entre un ordinateur et une imprimante ou un autre terminal local. A l'inverse, sur les grandes distances, l'emploi de dispositifs auxiliaires, comme les modems, est nécessaire.

Compactage. Méthode de réduction de la longueur du champ de la clé en utilisant des algorithmes de transformation appropriées. Désigne aussi la réduction de l'espace occupé par des données, par exemple en éliminant les expaces vides.

Compactage. Terme utilisé principalement en Basic pour désigner l'opération de réduction de la zone mémoire réservée aux chaînes. Elle permet à l'ordinateur d'éliminer celles qui ne sont plus exploitées, de regrouper toutes les zones occupées en une seule, de rendre disponible pour d'autres valeurs l'espace restant.

Compartiment. Zone de mémoire pouvant contenir plusieurs enregistrements et être exploitée globalement.

Compilateur. Programme capable de traduire des programmes utilisant des codes symboliques en une forme compréhensible pour l'ordinateur. Contrairement à l'interpréteur, le compilateur effectue la traduction de la totalité du programme et produit une forme intermédiaire, dite translatable, qui ne peut pas encore être exécutée.

Complément. Indique généralement un inverse. C'est le nombre obtenu par soustraction d'un nombre à un autre. Par exemple, le complément à 10 de 7 est 3 (10-7=3). En informatique, on utilise plus fréquemment le complément à 2, le système de numération étant binaire.

Complément à 2. Forme de représentation des nombres négatifs dans le système binaire.

Compteur. Circuit pouvant compter les impulsions reçues.

Concaténation. Union de deux chaînes pour en former une troisième constituant la somme des deux autres. Ce terme désigne également l'union de deux programmes ou le regroupement de données dans les mémoires de masse.

Constante. Toute valeur fixe, numérique ou alphabétique.

Conversationnel. Qualifie une méthode d'introduction de données selon laquelle l'ordinateur pose une série de questions à l'opérateur et attend les réponses, qu'il valide au fur et à mesure.

Coordonnées. Série de nombres permettant d'identifier un point par rapport à une référence. Les coordonnées peuvent se référer au plan et, dans ce cas, elles sont au nombre de deux (x, y), ou bien encore à l'espace, avec 3 valeurs. Elles servent généralement à identifier la position du curseur à l'écran ou celle de la pointe du traceur.

Coordonnées absolues. Coordonnées indiquant la position d'un point par rapport à l'origine du système de référence utilisé.

Coordonnées relatives. Coordonnées permettant d'identifier un point en indiquant son déplacement par rapport à un autre point.

Copie d'écran (hardcopy). Copie sur papier de ce qui apparaît sur un périphérique de sortie, généralement l'écran.

Corrélateur. Circuit dont la sortie vaut 1 si toutes les entrées sont 0. Sert à former des codes.

Couche. Dans les dessins complexes, l'ensemble des éléments représentés peut être divisé en sous-ensembles portant le nom de couches, afin de fournir une vue logique fractionnée permettant de composer le dessin par groupes.

CPU. Voir **UC**.

CR (Carriage Return : retour chariot). Lorsque ce code (ASCII) est envoyé à une imprimante ou à un écran, la

tête d'écriture ou le curseur va se placer à la première position utile de la même ligne. Pour obtenir un changement de ligne, il faut que le code LF (Line Feed), soit envoyé en même temps. A la fin d'un texte, les codes CR et LF sont donc presque toujours présents.

CRT (Cathode Ray Tube : tube à rayons cathodiques). Composant de l'écran permettant d'afficher les données de façon visible en exploitant les propriétés de certains luminophores excités par un faisceau d'électrons.

CRT. Désigne, par extension, le terminal de visualisation lui-même (écran).

Culot. Logement en matériau isolant protégeant des composants en évitant qu'ils ne soient soudés au circuit imprimé.

Curseur. Elément d'identification d'une position à l'écran. Il est généralement représenté par un tiret ou par un petit rectangle lumineux à l'écran.

Cycle. Désigne une phase de traitement de l'unité centrale, par exemple l'extraction d'une donnée de la mémoire. Une instruction se compose généralement de plusieurs cycles.

Cylindre. Zone pouvant être lue sans mouvement du mécanisme d'accès. Un cylindre désigne généralement l'ensemble de deux traces, chacune sur une face du disque, lisibles à partir d'une seule position des têtes de lecture.

D

DASD. (Direct Access Storage Device : unité de stockage à accès direct). Dispositif de mémoire à accès direct.

Data. Voir **Données**.

Débit. Vitesse d'exécution d'un traitement.

Debugger, debugging. De bug : vermine. Voir **Bogue**.

Décalage (retrait). Insertion d'un certain nombre d'espaces non significatifs au début d'une ligne de programme pour augmenter la lisibilité du listing. Ce décalage n'est possible qu'avec certains langages (Pascal et certaines formes du Basic).

Déclaration. Ensemble d'instructions définissant le type ou les dimensions des variables, ou encore les valeurs des constantes.

Déclaration. Opération d'indication du type ou des dimensions des variables. Elle peut être explicite, si elle se trouve dans les lignes du programme, ou implicite, si elle est automatiquement présumée par le langage utilisé. Par exemple, la déclaration d'un dimensionnement en Fortran est explicite, alors qu'en Basic elle peut, dans certaines limites, être implicite.

Décodeur. Circuit servant à décoder des données.

Décoder. Transposer une valeur de sa forme codée à sa forme normale.

Défaillance. Condition accidentelle d'erreur au niveau d'un périphérique. Elle entraine généralement sa mise hors service par le système d'exploitation.

Défaut (valeur par). Valeur optionnelle considérée par le système à défaut d'autre indication.

Défilement. Mouvement horizontal ou vertical d'une partie ou de la totalité de l'image à l'écran afin de faire apparaitre de nouvelles données. Dans les systèmes les plus complexes, dotés d'une mémoire vidéo spéciale, le défilement peut se faire dans les deux sens (on peut rappeler des rangées ou des colonnes déjà vues).

Délai d'attente. Temps nécessaire pour positionner la tête de l'unité de disque au-dessus de la surface en rotation.

Dépassement de capacité. Dans le cas d'un programme, cette expression signifie que la capacité de la mémoire a été dépassée.

Dépendance fonctionnelle. Il y a dépendance fonctionnelle quand un certain attribut d'une relation dépend d'un autre attribut de la même relation et que cette dépendance est univoque. On dit alors que le premier attribut identifie le second.

Déverminage, déverminer. Pour debugging et debugger. Voir **Bogue.**

Diagnostic. Opération par laquelle l'opérateur est informé des causes possibles d'une erreur. On reconnait deux types de diagnostic : celui du système et celui des programmes d'application. Le premier est géré par le système ou par les compilateurs et fait apparaitre les erreurs de forme ou de syntaxe. Le second doit être préparé par le programmeur en fonction de son application.

Dictionnaire. Liste de tous les types de données avec leurs noms et leurs structures.

Digital. Voir **Numérique**.

DIP. Dual In line Package. Boîtier dont les broches se trouvent sur deux côtés parallèles.

Directory. Voir **Répertoire**.

Disponibilité. Mesure l'exploitation d'un système sans défaillance.

Disque souple (floppy). Disque magnétique souple.

Disquette (mini disque). Ce terme désigne généralement les disquette de 5 pouces 1/4, d'une capacité de l'ordre de 160 Kockets et plus. Il peut également s'appliquer aux disques de 3 1/2 pouces adoptés par certains constructeurs de micro-ordinateurs professionnels.

Diviseur. Circuit logique à la sortie duquel est présentée

une impulsion pour toutes les « n » impulsions d'entrée. Il est généralement réalisé selon une configuration de compteur particulière.

DL/1. Langage particulier, implémenté sur machine IBM, pour la définition logique et physique des structures de données.

DMA (Direct Memory Access). Voir **ADM**.

Domaine. Ensemble des données du même type liées par une relation déterminée.
Espace sur disque réservé à un fichier.

Données. Informations à fournir avant le traitement. Toutes les données doivent être connues.

DOS (Disk Opérating System : système d'exploitation sur disque). C'est l'ensemble des programmes et sous-programmes, enregistrés sur disque, qui constituent un système d'exploitation. L'ensemble de ces programmes peut être divisé en deux groupes fondamentaux : les **transitoires** et les **résidents.** Les premiers ne sont chargés en mémoire qu'au moment de leur activation, tandis que les autres sont toujours résidents, par exemple, les modules de gestion des périphériques qui se retrouvent dans tous les programmes doivent être résidents.

Double précision. Méthode de mémorisation des données utilisant un espaces de mémoire deux fois plus grand que la normale afin d'utiliser plus de chiffres significatifs.

Drapeau. Voir **Indicateur**.

DRC (Design Rules Checking : contrôle des règles de conception). Procédé de contrôle des données servant à faire apparaître des types d'erreur prédéterminés, comme une incompatibilité des tolérances.

DTL (Diode Transistor Logic : logique à transistor et diode). Circuits obtenus en combinant des diodes et des transistors.

Duplex intégral. Mode de transmission simultané dans les deux sens. Est dit intégral par opposition au semi-duplex, système de transmission non simultanée. (Voir **Simplex**).

Dynamique. Qualifie un événement se produisant en cours d'exécution d'un programme.

E

EAO. Enseignement Assisté par Ordinateur. Système d'enseignement autodidactique où l'ordinateur joue le rôle de professeur.

EBCDIC. Extended Binary Coded Decimal Interchange Code. Code spécial, parfois utilisé dans la transmission de données à la place du code ASCII, plus connu. Dans le code EBCDIC, chaque caractère est représenté par 8 bits. La première valeur est 0000 0000, qui représente l'espace en blanc (SP), tandis que la dernière est 1111 1001, qui correspond au caractère 0.

Echange. Processus au cours duquel deux variables échangent leurs contenus.

Echelle. Rapport entre les dimensions d'une image et celles de l'objet réel.

ECL. Emitter-Coupled Logic (Logique à couplage d'émetteurs). Famille particulière de circuits intégrés assez peu utilisée car difficilement interconnectables à d'autres.

Eclatement cellulaire. Technique particulière utilisée pour ranger les enregistrements ajoutés à un fichier. Les enregistrements sont placés dans une cellule qui est divisée en deux chaque fois qu'elle est remplie.

Echo. Retransmission de la donnée reçue. Par exemple, les caractères transmis par le clavier sont présentés à l'écran, en écho.

Ecrêtage. En électronique, il s'agit de l'élimination de la partie la plus élevée du signal. Dans les applications graphiques sur ordinateur, ce terme désigne parfois le procédé permettant d'éliminer une partie du dessin pour visualiser à l'écran des objets qu'il ne contiendrait pas autrement.

Edition. Modification de données. Ce terme est plus particulièrement utilisé pour désigner les opérations de correction (insertions ou modification) de programmes.

Edition. Opération de préparation des données pour l'impression ou pour l'opération de modification d'un programme.

Effet de bord. Il y a effet de bord quand une erreur générée dans une partie de programme par des causes déterminées réapparaît dans d'autres parties du programme.

Elément. Donnée particulière d'une liste ou d'un tableau.

Elimination des faces cachées (autohide). Option existant dans les systèmes graphiques les plus perfectionnés et permettant de visualiser une figure sous un angle particulier en effaçant les faces non visibles.

Emboîtement. Opération par laquelle une structure est englobée dans une autre. Ainsi, une boucle peut être emboîtée dans une autre ou un sous-programme imbriqué dans un autre.

En ligne (en direct). Qualifie un système dans lequel les données peuvent être entrées ou prélevées directement. C'est le contraire de hors ligne, ou différé, qui s'applique à des données accessibles uniquement après des actions particulières, comme la copie sur bande d'archives historiques.

Enregistrement. Ensemble d'informations liées entre elles, traitées comme une seule entité.

En-tête. Désigne généralement une partie initiale contenant la description ou la clé d'interprétation des parties suivantes. Ce terme est surtout employé pour désigner l'enregistrement initial d'un fichier contenant des informations sur ceux qui le suivent.

En tête (de gauche). Qualifie la partie initiale d'un champ de données. On peut, par exemple, parler des zéros en tête avant les valeurs numériques ou des espaces en tête avant les caractères d'une chaîne.

Entier. Nombre sans décimale.

Entrance. Terme désignant la charge produite par un circuit à son entrée. L'entrance donne un ordre de grandeur de la puissance nécessaire pour piloter ce circuit. (Voir **Sortance**).

EOF. End of File. Signal d'identification de la fin d'un fichier.

E/S. Entrée/Sortie. (anglais I/O). Désigne toutes les opérations d'entrée ou de sortie de données.

Escape. Code particulier utilisé dans de nombreux systèmes d'exploitation pour indiquer une action à exécuter. Correspond généralement à une touche spéciale.

Esclave. Dispositif faisant partie d'un réseau mais n'ayant pas une fonction dirigeante, par exemple un terminal vidéo.

Espace. Etat 0 (bas) lors de la transmission de données.

Etiquette (label). Terme désignant, dans de nombreux langages symboliques, le libellé d'adressage d'une instruction. Il peut également se référer au disque , dans ce cas, il s'agit de son nom.

Evénement. Action généralement extérieure pouvant provoquer une interruption de programme afin que des programmes particuliers soient lancés. Dans certains ordinateurs, des tests fonctionnels, gérés sous interruption, produisent de tels événements. Ces tests fonctionnels peuvent également être gérés par programme, auquel cas il n'y a pas d'interruption mais intégration continue de l'état des touches par le programme.

Exécution. Réalisation d'une instruction. Désigne le cycle d'exécution d'une instruction qui suit généralement l'extraction de son code en mémoire.

Extraction. Cycle de prélèvement en mémoire du code de l'instruction qui doit être ensuite exécutée.

F

Faces cachées. Plans à effacer parce que couverts par d'autres parties dans les graphiques d'objets en trois dimensions (3D).

Facteur de groupage. Nombre maximal d'enregistrements prévus dans un bloc. Est généralement utilisé en Cobol.

Facteur d'échelle. Valeur de multiplication des dimensions d'un dessin pour le ramener aux dimensions de l'écran ou du facteur.

FAO. Fabrication Assistée par Ordinateur. Système de conception et de préparation des données pour le fonctionnement des machines à commande numérique assistées par ordinateur. C'est un stade ultérieur de la CAO.

Fenêtre. Zone rectangulaire de l'écran sélectionnée par l'opérateur.

Fenêtre. En informatique graphique, et dans certains cas également en Mode texte, désigne une zone particulière de l'écran gérée indépendamment du reste, comme s'il s'agissait d'un écran séparé. En Pascal, désigne un ensemble d'identificateurs formant le « window diagram ».

FF. Form Feed. Caractère de commande (voir code ASCII) provoquant le saut à la page suivante.

Fichier circulaire. Fichier particulier dans lequel les nouveaux enregistrements remplacent les précédents.

Fichier hiérarchique. Fichier dans lequel certains enregistrements dépendent d'autres enregistrements selon une structure arborescente.

Fichier inversé. Structure particulière d'un fichier. Elle permet de rechercher des informations n'ayant pas été décrites précédemment. Un fichier inversé est obtenu à partir de listes d'indices inversées.

Fichier logique. Structure d'un fichier tel que le programme d'application le voit. Elle peut être totalement différente de sa structure physique.

Fictif. Qualifie une valeur utilisée uniquement pour suivre des règles formelles déterminées, sans être prises en compte. Par exemple, un argument d'une fonction peut être fictif s'il n'est pas réellement utilisé pour effectuer des calculs mais seulement pour respecter une règle formelle (en réalité, l'argument fictif est utilisé par le système, mais pas en tant que valeur).

File. Voir **Fichier**.

File d'attente. Liste d'éléments en instance de traitement. Il peut s'agir, par exemple, de la liste des demandes de service de plusieurs périphériques ou de procédures à exécuter.

Filtre. Composant permettant d'éliminer une grandeur en gardant les autres inchangées.

Flag (Drapeau). Voir **Indicateur**.

Floppy. Voir **Disque souple**.

Fonction. Procédure qui, à partir d'autres valeurs (paramètres) et d'un algorithme donné, rend une valeur dépendante. Les fonctions déjà prévues dans un langage (par exemple les fonctions mathématiques du Fortran) sont dites intrinsèques.

Foreground. Voir **Premier plan**.

Format. Forme de sortie ou d'entrée de données. Ce terme désigne également, parfois, la structure de mémorisation sur disque. A ne pas confondre avec le « formatage » du disque tout entier.

Formatage. Opération consistant à « préparer » le support (disque, par exemple), selon les spécifications de chaque système d'exploitation.

FORWARD. En Pascal, mot réservé, utilisé dans les déclarations de certaines procédures particulières.

Fréquence. Indique le nombre de fois par seconde qu'un signal prend la même valeur ; c'est l'inverse de la période.

FSK (Frequency Shift Keying). Méthode de modulation des signaux consistant à modifier la fréquence d'une onde porteuse en fonction des valeurs du signal modulant.

Fusion. Procédure selon laquelle deux listes en produisent une seule. Elle peut également s'appliquer à deux programmes.

G

Gamme. Limites entres lesquelles se trouve la valeur d'un élément donné. Par exemple, la gamme des nombres entiers d'une machine 8 bits va de - 32 000 à + 32 000.

Groupage. Combinaison de deux blocs (ou plus) de manière à faciliter leur lecture/écriture avec une seule instruction.

Guide-opérateur. Se dit d'un message ou d'un symbole utilisé par l'ordinateur pour guider l'utilisateur.

H

Haut niveau (évolué). Qualifie les langages symboliques orientés utilisateur.

Hertz. Unité de mesure de la fréquence. Ses multiples sont : kHz (kilo-Hertz = 1 000 Hz), MHz (méga-Hertz = 1 000 000 Hz), GHz (giga-Hertz = 1 000 MHz). Il n'a pas de sous-multiples.

Heuristique. Méthode de résolution d'un problème qui consiste à effectuer différents essais et à en analyser ensuite les erreurs.

Hiérarchie. Structure d'éléments subdivisée en différents niveaux ayant plusieurs degrés d'importance ou de priorité. La hiérarchie des opérations, par exemple, établit les règles de priorité de leur exécution arithmétique. Pour ce qui concerne les fichiers ou les structures de données, ce terme indique une subdivision et les lignes de dépendance de tous les éléments de niveau inférieur à un autre et liés à celui-ci.

Histogramme. Graphique à barres. La longueur de chaque barre représente la valeur de la variable.

Horloge. Composant capable de produire un signal périodique de synchronisation. Chaque signal est une impulsion d'horloge ; il est caractérisé par une amplitude et une période. L'amplitude est la valeur maximale atteinte par le signal à l'état actif, tandis que la période indique sa durée en secondes. On utilise aussi parfois la caractéristique de fréquence (mesurée en Hz ou MHz) dont la valeur numérique l'inverse de la période : elle représente le nombre d'impulsions émises en une seconde.

Hors ligne (en différé). Se dit d'un appareil sous le contrôle direct de l'ordinateur dirigeant le système. Quand elle s'applique à un traitement, cette expression signifie qu'il est exécuté dans le flot principal.

Hôte. Ordinateur central d'une installation à plusieurs machines (réseau).

Huffman. Voir **Code Huffman**.

I

Identificateur : Nom donné à un programme ou à une de ses parties.

Incrément. Valeur modifiant un compteur, comme le STEP (pas) dans les instructions FOR... du basic ou DO... du Fortran.

Indexé. Mode d'adressage dans lequel une position est obtenue en additionnant le contenu du registre d'index à l'adresse contenue dans l'instruction.

Indicateur (flag). S'utilise en un point donné d'un programme pour mémoriser un état ou un événement particulier. Certains langages, comme le RPG, utilisent beaucoup les indicateurs pour déterminer par exemple le type d'enregistrement ou indiquer le résultat d'une comparaison. Dans ces langages, on utilise généralement plutôt le terme de code, mais les fonctions ou la logique d'utilisation sont les mêmes. Les opérations activées en fonction de l'état de ces indicateurs sont dites « en rupture de codes ».

Indicateur. C'est un registre élémentaire qui permet d'afficher l'état d'un évènement. Il indiquera, par exemple, les débordements, le signe...

Indice. Valeur qui détermine la position d'un enregistrement ou un élément dans un tableau.

Inductance. Concerne un circuit constitué d'un certain nombre de bobines. Elle se mesure en (micro-Henry).

Infini machine. Le plus grand nombre qui peut être représenté selon le format interne d'un ordinateur.

Informatique répartie. Technique permettant d'obtenir un traitement simultané de données sur différents systèmes connectés en un réseau.

Initialisation. Dans un programme, affectation des valeurs initiales aux indicateurs, compteurs, chaînes...

Instable. Qualifie un circuit n'ayant pas d'état stable. L'exploitation de cette caractéristique permet d'utiliser ce type de circuit comme générateurs.

Instruction. Ensemble de mots réservés et de données formant une commande et indiquant une action particulière à exécuter par l'ordinateur. Un programme se compose d'une suite d'instructions (voir également **Mot réservé** et **Mot-clé**).

Instruction composée. Instruction complexe non prévue dans le langage et obtenue en réunissant des instructions élémentaires. Le bloc obtenu doit normalement commencer et finir par des mots-clés particuliers. Les instructions composées n'existent que dans certains langages évolués.

Intégré. Qualifie une procédure faisant partie intégrante du système d'exploitation.

Intégrité. Caractéristique indiquant l'absence d'erreurs dans un ensemble de données.

Interactif. Se dit d'un programme établissant un dialogue avec l'utilisateur (voir **Conversationnel**).

Interface. Dispositif qui adapte les signaux aux exigences de différents circuits. Dans les communications entre plusieurs ordinateurs (ou entre un ordinateur et des périphériques), les signaux doivent être adaptés tant à l'émission qu'à la réception.

Interpréteur. Programme capable d'interpréter et d'exécuter un programme source.

Interpréteur. Programme spécifique chargé de traduire un langage symbolique en codes exécutables. Contrairement au compilateur, l'interpréteur effectue la traduction et l'exécution instruction par instruction. La traduction d'une instruction n'étant pas mémorisée, une opération donnée doit être exécutée autant de fois que l'instruction est répétée.

Interruption. Suspension du programme en cours d'exécution, avec sauvegarde de tous les paramètres, de telle sorte que l'exécution puisse reprendre ultérieurement à partir du point d'interruption. L'interruption est dite « armée » quand le système, à l'état de validation, mémorise la demande. Elle est dite « traitée », ou « prise en charge » lorsque le système a activé un sous-programme spécial chargé d'exécuter les tâches demandées par l'interruption.

Intersection. Union de deux ou de plusieurs groupes de données, obtenue en prélevant seulement les éléments communs à ces groupes.

Intersection. Création d'un ensemble de données à partir de deux autres. Il regroupe tous les éléments communs aux deux premiers, et exclusivement ceux-là.

Invalidation. Etat interdisant certaines fonctions. Par exemple, quand les interruptions sont invalidées, le programme ne peut pas être interrompu (voir **Validé**).

Inversé. L'opposé d'un état quelconque. En parlant d'affichage vidéo, ce terme qualifie un libellé en noir sur fond blanc, généralement obtenu en éteignant les pixels composant les lettres. Normalement, c'est le contraire : l'écran est éteint et les lettres sont obtenues en activant certaines zones).

Inversé (fichier). Voir **Fichier inversé**.

Invitation à émettre. Phase de la transmission de données pendant laquelle le maître invite un esclave à envoyer ses données.

I/O. Voir **E/S**.

IPS. Inches Per Second. Unité de mesure de la quantité de ruban (exprimée en pouces) pouvant être traitée (lue ou écrite) en une seconde.

ISAM. Index Sequential Access Method. Méthode d'accès aux données d'un fichier à l'aide d'indices.

Itératif. Se dit d'une méthode de calcul consistant à chercher une solution par des passages successifs.

J

Jeu. Ensemble de symboles reconnus dans un langage donné ou par un ordinateur.

Jeu d'essai. Ensemble d'instructions utilisé pour contrôler la qualité d'un logiciel.

Joystick. Voir **Manche à balai**.

Jump. Voir **Saut**.

Justification. Opération de déplacement des valeurs à l'intérieur d'une zone ayant pour résultat de mettre tous les espaces du même côté. D'une façon générale, les valeurs numériques sont justifiées à droite, les éventuels espaces étant placés à gauche des valeurs, tandis que les données alphanumériques sont justifiées à gauche.

K

K. Abréviation de kilo, et plus généralement symbole du facteur de multiplication 1 000. Dans le cas des mémoires d'ordinateur, un K vaut 1 024.

L

Label. Voir **Etiquette**.

Langage machine. Ensemble des codes numériques que l'ordinateur interprète et exécute directement, c'est-à-dire sans passer par des traductions comme avec les langages dits évolués.

LF. Abréviation de Line Feed. Caractère de commande qui provoque le déplacement d'une ligne, mais sans changement de colonne.

LFU. Least Frequently Used (le moins fréquemment utilisé). Algorithme consistant à remplacer les données les moins utilisées par de nouvelles (voir également **LRU**).

Ligne. Donnée permettant de définir une position horizontale (par opposition à la colonne). On peut également parler de rangée.

Ligne. Quand il se rapporte à une sortie, ce terme désigne un ensemble de données contiguës et terminées par un CR. En entrée, il s'agit d'un groupe de données considérées comme un seul bloc (par ex. une chaîne). En transmission, il désigne tout moyen permettant d'envoyer des signaux : câbles électriques ou fibres optiques (pour la transmission des données sous forme de signaux lumineux).

Listing. Copie, sur papier ou à l'écran, des instructions constituant un programme.

Logiciel. Programmes, sous-programmes et, d'une façon générale, tout de qui n'est pas matériel. Traduction de Software.

Longueur fixe (fichier). Qualifie généralement un type particulier de fichier dans lequel tous les enregistrements ont la même longueur, par exemple les fichiers à accès direct en Basic.

Loop. Voir **Boucle**.

Lots. Voir **Traitements par lots**.

LRU. Least Recently Used (le moins récemment utilisé). Algorithme dit d'ancienneté consistant à remplacer les données utilisées le plus récemment par de nouvelles données (voir aussi **LFU**).

LSI. Large Scale Integration. Désigne des composants ayant un niveau élevé d'intégration. Il s'agit généralement de circuits composant un système tout entier.

M

M. Abréviation de Méga (10^6).

Macro. Groupe de commandes ou d'instructions considérées par l'ordinateur comme une seule instruction. Elles sont particulièrement utilisées en Assembleur ou dans certains progiciels comme les tableurs.

Macro-instruction. Instruction source qui génère une série d'actions équivalentes à un certain nombre d'autres instructions.

Manche à balai (joystick). Mécanisme constitué d'un levier et de transducteurs de position. Ses déplacements permettent de commander la position du curseur ou d'un dessin à l'écran.

Mantisse. Dans la représentation des nombres en virgule flottante, désigne la partie numérique qui doit être multipliée par l'exposant. Par exemple, le nombre 5 600 écrit sous la forme $5.6 \times E^3$ a pour mantisse 5.6 et pour exposant 3.

Marque. Etat 1 (haut) pendant la transmission de données.

Masque. Ensemble de caractères utilisé pour contrôler l'acquisition ou l'élimination de certains caractères appartenant à un autre ensemble. Le masque peut être représenté par un seul caractère ; dans ce cas, il sert à la sélection des bits individuels.

Matériel. Partie physique d'un système de traitement.

Matrice. Tableau à deux dimensions ou plus.

Mémoire associative. Mémoire dont les positions sont identifiées d'après leur contenu et non par leurs adresses. Elle permet d'augmenter sensiblement la vitesse de recherche (à l'aide des attributs du champ considéré).

Mémoire-tampon. Zone de mémoire utilisée pour stocker temporairement des données. On recourt généralement au tampon lors d'opérations d'E/S pour servir de « poumon » entre deux dispositifs échangeant des données à des vitesses différentes. En impression, par exemple, la présence d'un tampon permettra à l'ordinateur d'envoyer une certaine quantité de données et de se consacrer à d'autre tâches pendant que l'imprimante imprime ces données.

Menu. Liste des fonctions prévues dans un programme donnant la possibilité d'en sélectionner une.

Méthode d'accès. Technique de transfert de données entre un ordinateur et un périphérique. Les principales sont les suivantes : accès série, direct (aléatoire), à distance, séquentiel virtuel (VSAM) et indexé hiérarchique séquentiel (HISAM). Ces deux dernières sont caractéristiques des grands systèmes.

Méthode des éléments finis. Méthode de calcul utilisée essentiellement en génie mécanique.

Micrologique (microprogrammes). Fonctions logicielles mémorisées de façon permanente, par ex. sur ROM.

Micro-ordinateur. Ordinateur basé sur des circuits

intégrés, voire parfois sur un circuit unique.

Migration. Opération de transfert des données dans une zone permettant d'augmenter la vitesse d'accès.

Mini-ordinateur. Ordinateur basé sur des circuits intégrés et, quelquefois, sur un seul circuit. Il est vrai que les micro-ordinateurs professionnels ont de plus en plus la puissance d'un mini.

Mise à jour. Opération de modification des données, généralement dans un fichier.

Mise au point. Opération de recherche et d'élimination des erreurs (voir déboguer).

Mode d'accès. Technique utilisée dans la lecture ou pour l'écriture d'un enregistrement logique d'un fichier. Les principaux sont les suivants : le mode séquentiel, dans lequel les enregistrements sont traités l'un après l'autre, et le mode direct, où l'on accède uniquement à l'enregistrement désiré.

Mode point (graphique). L'expression graphique en mode point (bit map graphics) désigne la représentation prenant en compte l'état de chaque point de l'écran.

Modèle. Les programmes qui simulent un certain phénomène ou un processus déterminé utilisent un modèle pour simuler le comportement du système réel.

Modèle conceptuel. Schéma d'ensemble d'une base de données.

Modem. Abréviation de Modulateur-Démodulateur. Appareil utilisé pour moduler et démoduler les signaux dans les transmissions à grande distance.

Module. Partie d'un programme logiquement séparée du reste et exécutant des fonctions définies et autonomes.

Moniteur. Synonyme d'écran.

Monitoring. Ce terme est parfois utilisé pour désigner l'opération par laquelle un programme présente l'état des variables tout en poursuivant le traitement.

Morgan. Voir **Théorème de Morgan**.

Mot. Ensemble constitué d'un nombre de bits égal à celui pouvant être inséré dans une adresse mémoire. Les longueurs les plus usitées sont : 8, 16 et 32 bits.

Mot-clé. Mot prédéfini et réservé, dans le langage de programmation utilisé, à un emploi particulier.

Mot réservé. Terme utilisé dans un langage (ou une commande) pour activer une fonction déterminée. Il ne peut généralement pas être utilisé à une autre fin, par exemple comme nom de variable.

MSI. Medium Scale Integration. Technologie particulière qui permet de contenir dans un seul boîtier un nombre de composants supérieur à celui de la TTL mais inférieur à celui de la LSI.

Multiplexeur. Dispositif permettant d'établir un dialogue entre plusieurs unités en utilisant un nombre réduit de liaisons. Deux types sont utilisés : temporel et fréquentiel.

Multipoint. Type de configuration de réseau dans lequel un maître dirige l'échange avec différents esclaves.

Multiprocesseur. Ordinateur utilisant simultanément plusieurs unités centrales.

N

Nœud. En parlant d'un réseau, il s'agit du point terminal d'une branche ou de la connexion de plusieurs dérivations.

Nœud. Point de branchement dans un organigramme. Ce terme est principalement employé en programmation structurée et désigne un point de branchement d'un diagramme en arborescence.

Nœud terminal. Terme utilisé, en programmation structurée, pour désigner un bloc n'ayant pas de branchement vers le bas.

Normalisation. Alignement de chiffres composant un nombre pour l'obtenir dans la forme requise par les règles arithmétiques (syntaxe).

Notation. Forme de représentation des données. En informatique, on utilise la notation binaire, octale, hexadécimale...

Nul. Qualifie un mot en blanc qui, d'une façon générale, ne génère aucune action. Il est intercalé entre des mots contenant les données des transmissions synchrones.

Numérique. Qualifie un dispositif ne pouvant reconnaître que des états définis et non pas les valeurs intermédiaires. Les deux états possibles sont généralement identifiés par 0 et 1. Noter cependant que certains dispositifs électroniques sont à trois états ; dans ce cas, tout de même, le troisième n'est pas un état logique mais seulement une condition électrique particulière dans laquelle le dispositif est comme branché.

Numériseur. Appareil permettant de convertir des coordonnées en une forme numérique compréhensible pour l'ordinateur.

O

Octal. Qualifie un système de numération utilisant la valeur 8 comme base.

Octet. Bloc de 8 bits. En binaire, un caractère est contenu dans un octet.

Off-line. Voir **Hors ligne**.

Offset. Désigne une valeur indiquant le décalage d'une donnée par rapport à une origine. Dans un fichier, il peut s'agir du nombre d'octets à partir du début d'un enregistrement, ce qui donne la position d'une donnée. Dans le cas de la mémoire, c'est une valeur constante à déduire d'une adresse pour connaître une position.

Offset. En électricité, désigne une tension constante à partir de laquelle le signal est appliqué.

On-line. Voir **En ligne**.

Opérande. Elément sur lequel doit être effectuée une opération.

Opérateur. Symbole particulier indiquant des opérations déterminées à exécuter sur les opérandes. Les principaux opérateurs sont de deux types : arithmétique et logique.

Opération. Toute action qui, appliquée à une combinaison d'éléments connus, génère un nouvel élément appartenant au même ensemble.

Ordinateur central. Elément central de traitement dans une configuration de grandes dimensions (voir **Hôte**).

Ordre. Indique la logique utilisée pour former une liste de données particulière. En parlant d'un bit, il s'agit de sa position à l'intérieur du mot.

Organigramme. Représentation graphique des fonctions à exécuter dans un programme.

Oscillateur. Circuit capable de produire un signal périodique (voir **Horloge**).

Overflow. Voir **Dépassement de capacité**.

Overlay. Voir **Zone de débordement**.

P

Page. Zone de mémoire généralement composée de 1 024 positions. Dans certains systèmes, il faut distinguer la page zéro et la page en cours. La page zéro contient des informations de chaînage utilisées par le système, alors que la page en cours est la zone de mémoire contenant l'instruction en cours d'exécution.

Page active. Dans certains systèmes, l'unité vidéo peut utiliser plusieurs pages de mémoire. La page active indique celle qui contient des informations. Elle est généralement différente de celle présentée à l'écran.

Pagination. Technique de segmentation utilisée par les systèmes d'exploitation les plus évolués et consistant à diviser les programmes ou les données en pages normalement résidentes sur disque et chargées en mémoire uniquement au moment opportun. C'est une façon d'augmenter l'utilisation de la mémoire par rapport à sa vraie capacité (la même zone est partagée, à des moments différents, par plusieurs groupes de données).

Paliers (par). Technique de développement d'un programme consistant à obtenir le résultat final par étapes de plus en plus approfondies et précises.

Paquet. Ensemble de valeurs constituant un bloc de données.

Paramètre. Nom symbolique utilisé pour transférer des valeurs, comme argument d'une fonction, lors de l'appel d'un sous-programme, ou encore entre procédures. En voici quelques types : paramètres effectifs, paramètres formels, paramètres à appel par nom, paramètres à appel par valeur.

Parité. Etat des données à contrôler avant leur transmission. Ce contrôle consiste à vérifier le nombre d'états ON en envoyant, en même temps que la donnée, un bit indiquant si ce nombre doit être pair ou impair. La parité paire est la logique selon laquelle le bit de parité est actif si le nombre de bits ON est pair, alors que dans le cas de la parité impaire (ou imparité), le bit de parité n'est actif que quand ce nombre est impair.

Pas à pas. Type de déroulement d'un programme. On recourt à la technique pas à pas (anglais step by step) pour effectuer des recherches d'erreurs. Toutes les machines n'offrent pas cette possibilité.

Passation. Opération de transfert des paramètres entre un programme (ou un sous-programme) appelant et un sous-programme ou une fonction appelé(e).

Période. Intervalle de temps entre deux points d'un signal de même amplitude.

Périphérique. Tout dispositif utilisé par l'ordinateur pour communiquer avec son environnement.

PEP (Parametric Element Processor). Langage évolué, utilisé pour résoudre des problèmes de graphique en trois dimensions (3D).

Phase. Représente l'origine d'un signal périodique.

Photostyle. Dispositif sensible à la lumière utilisé pour détecter un point de l'écran d'après sa luminosité.

Piping. Terme utilisé en informatique graphique (infographie) pour désigner la conception de tuyaux, lignes électriques ou éléments linéaires en général.

Piste. Division circulaire du disque, divisée en secteurs.

Pixel. Un point de l'écran (PICture Element) en mode graphique. Il peut également s'agir du bit précis de la zone mémoire contenant l'information sur l'état du point écran correspondant (voir **Point**).

Plaquette de circuits imprimés (PCB, Printed Circuit Board). Support isolant, par exemple la vétronite, sur lequel est gravé, à l'aide de procédés

photographiques, l'ensemble des liaisons à réaliser.

Plot. Petite zone d'un circuit imprimé sur laquelle sont soudés les composants.

Poids. Valeur multiplicative d'un chiffre dépendant de sa position dans un nombre écrit selon un système de numération positionnel. Par exemple, le 7 du nombre 275 a le poids 10.

Point. Désigne en général un point de l'écran (synonyme de pixel).

Point adressable. En informatique graphique, l'écran est divisé en un grand nombre de points formant une trame. Un dessin se construit en activant certains de ces points. D'une façon générale, l'écran ne peut pas être géré dans sa totalité et les zones activables sont appelées « points adressables ».

Point de reprise. Utilisé avec une méthode particulière consistant à écrire les paramètres d'un programme dans un fichier au fur et à mesure de son exécution. Si le programme s'interrompt pour des causes accidentelles, il peut être relancé à partir du point défini lors du dernier enregistrement (point de reprise). Cette technique est utilisée dans les procédures demandant un temps de traitement très long afin d'éviter de devoir repartir du début en cas d'interruption imprévue.

Pointeur. Variable dont la valeur indique un élément particulier d'un ensemble. Il s'agit soit d'un élément d'un tableau ou d'un enregistrement, soit d'un octet déterminé à l'intérieur de l'enregistrement.

Pointeur intégré. S'utilise dans un type de gestion particulière qui prévoit que les pointeurs se trouvent dans l'enregistrement des données auxquelles ils donnent accès et non pas dans le répertoire.

Port. Voir **Porte**.

Porte (d'accès). Point d'accès, c'est-à-dire d'entrée ou de sortie, des données. Les portes sont dites numériques quand elles permettent le transfert de signaux numériques (ON/OFF), ou analogiques. Dans ce dernier cas, elles doivent être équipées d'un convertisseur analogique/numérique.

Position. Désigne une position en mémoire ou un espace sur disque. Plus généralement désigne toute zone servant à la mémorisation d'une donnée.

Position. Position repérable à l'aide d'une valeur numérique. Par exemple, un enregistrement au début d'un fichier ou un caractère dans une chaîne.

Postprocesseur. Programme capable d'interpréter des données en sortie d'un autre logiciel afin de les convertir en un format adapté à des usages ultérieurs, par exemple pour produire des dessins avec un traceur de courbes.

Précision. Désigne, d'une façon générale, la capacité de relever des différences.

Précision. Indique la « qualité » de la machine. Elle est donnée par la différence entre la valeur réelle d'un nombre et celle que la machine est capable de gérer. Par exemple, dans une machine utilisant 9 chiffres significatifs (pour les nombres réels), la précision est d'environ $1/1\ 000\ 000\ 000^e$.

Premier plan. Zone de l'écran occupée par un dessin ou par un caractère.

Préparation. Voir **Pré-traitement**.

Pré-traitement. Méthode de traitement des données permettant de les rendre conformes aux exigences d'un logiciel particulier (synonyme de **Préparation**).

Procédure. Ensemble des programmes exécutant une tâche particulière. Le sens de ce terme est différent en Pascal : il désigne une partie de programme qui peut être exécutée , il est alors synonyme de sous-programme.

Progiciel : Logiciel complexe.

Programmation structurée. Méthode logique de développement d'un programme qui consiste à diviser le problème en plusieurs problèmes de moindre importance et à résoudre chacun d'eux à l'aide d'un module conceptuellement séparé des autres.

Programme d'application. Programme écrit en vue de résoudre un problème particulier.

Protégé. Se dit d'une zone mémoire à accès contrôlé. Peut également qualifier une disquette. Dans ce cas, il indique la présence ou l'absence du dispositif matériel permettant l'écriture (la protection est généralement réalisée en couvrant, ou dans certains cas en découvrant, une encoche de la disquette).

Protocole. Ensemble de règles concernant l'échange de messages entre deux unités.

Puce. Synonyme de circuit intégré.

Puck. Dispositif manuel servant à entrer des coordonnées.

R

Racine Valeur mathématique particulière. En programmation structurée, c'est le nœud de départ d'un organigramme en arbre.

Rafraîchissement (régénération). Méthode de maintien de l'image vidéo consistant à la retracer à intervalles réguliers avant qu'elle ne disparaisse.

RAM (Random Access Memory). Mémoire vive : mémoire à accès direct, ou sélectif, également dite mémoire vive.

Randomisation (algorithme de). Algorithme de calcul des nombres aléatoires.

Rappel arrière. Déplacement en arrière d'un caractère. Une touche d'espacement arrière est prévue dans de nombreux claviers. Quand elle n'existe pas, cette fonction est exécutée par une combinaison particulière d'autres touches.

Raster display. Voir **Balayage récurrent**.

Read only. Expression qualifiant un dispositif, le plus souvent des mémoires, qui ne peut être utilisé qu'en lecteur, donc non modifiable.

Recherche. Examen, par prélèvement d'un élément d'une liste, si satisfaction de conditions déterminées.

Recherche dichotomique. Algorithme particulier utilisé pour accélérer la procédure de recherche. L'ensemble à examiner est divisé en deux à chaque phase de la recherche.

Récursif. Qualifie une procédure telle que chaque pas utilise les résultats du précédent. Exemple, la technique permettant la solution d'une équation par des approximations successives, ou des types de programmes particuliers (également dits « réentrants »), qui peuvent s'appeler eux-mêmes.

Recouvrement. Utilisation de la même zone de mémoire par plusieurs programmes à tour de rôle. Dans les petits systèmes, le recouvrement est géré par le programme d'application ; dans les systèmes plus évolués, il est pris en charge par le système d'exploitation en fonction de l'activité en cours.

Réel. Adjectif qualifiant une variable ou une constante appartenant à l'ensemble des nombres réels.

Relationnelle : Se dit d'une base de données construite en fonction des relations de ses éléments. Sa gestion demande l'utilisation d'un logiciel capable de traiter ses éléments de façon à produire de nouvelles relations, ce qui donne une grande souplesse d'emploi au système.

Remark. Voir **Commentaire**.

Remplissage. Bourrage d'un bloc avec des données factices comme des zéros ou des espaces.

Répertoire (Directory). Table d'identification ou de correspondance d'un ensemble de données. Le répertoire d'une disquette est la zone de ce disque réservée au système. Elle contient le nom de chacun des fichiers et sa position (et d'autres informations spécifiques à chaque système).

Réseau. Ordinateurs connectés entre eux.

Résistance. Composant d'un circuit offrant une résistance au passage du courant. Celle-ci se mesure en Ohms et ses multiples (K=x1 000, M=x1 000 000).

Résolution. Quantité minimale mesurable entre deux points de l'écran. En informatique graphique, elle se mesure d'après le nombre de lignes comprises dans l'unité de longueur. En parlant des transducteurs, il s'agit de la plus petite quantité de la grandeur considérée pouvant engendrer un signal exploitable.

Retracer. Reconstruire un dessin pour y ajouter les dernières modifications.

Return. Fonction de retour dans le programme au terme de l'exécution d'un sous-programme. Dans de nombreux langages, c'est aussi le mot-clé de cette fonction.

RGB. Abréviation désignant un écran couleur fonctionnant selon la technique additive (Red, Green, Blue : Rouge, Vert, Bleu).

ROM. (Read Only Memory). Mémoire morte : mémoire à lecture seule (voir également **Read Only**).

Router. Logiciel capable de déterminer automatiquement les interconnexions des composants d'un circuit imprimé.

Routine. Programme.

RS 232. Sigle désignant une interface de communication utilisée dans les liaisons de transmission de données entre ordinateurs ou entre un ordinateur et des périphériques.

RTL (Resistance Transistor Logic). Logique obtenue en combinant des résistances et des transistors.

S

Saut. Rupture dans le traitement séquentiel normal.

Sauvegarde. Copie des données destinée à être utilisée pour une éventuelle reconstitution en cas d'erreur ou de panne.

Scalaire. Qualifie une grandeur non orientée (à l'inverse d'une grandeur vectorielle) ou ne figurant pas dans un tableau.

Schéma canonique. Modélisation d'un ensemble de données représentant la structure de ces données, indépendamment de l'usage qui en sera fait et du matériel ou du logiciel qui les exploitera.

Schéma d'implantation. Dessin représentant une entité physique grandeur nature.

Scissor (programme de découpage). Logiciel de division de graphiques en parties plus petites dans un écran.

Secteur. Quantité minimale adressable par la mémoire de masse.

Segment. Désigne une zone mémoire définie comme une extension, par exemple de 64 Koctets.

Sélection. Phase de transmission des données au cours de laquelle le maître invite un esclave à recevoir (on dit aussi « invitation à recevoir »).

Sémantique. Ensemble de règles de description d'un langage de programmation.

Semi-duplex. Méthode de transmission dans les deux sens en alternance. C'est la méthode la plus utilisée pour relier un écran ou un clavier, et l'unité centrale.

Séparateur. Caractère de séparation entre des valeurs. Dans certains langages, la virgule, par exemple, servira de séparateur lors de la saisie des données.

Séquentiel. Se dit d'un type de fichier dans lequel les enregistrements sont mémorisés par ordre croissant d'après le contenu d'un champ particulier de la clé.

Série. Expression contenant un ensemble d'éléments produits à l'aide de règles mathématiques précises.

Signal. Impulsion électrique indentifiable par l'ordinateur.

Signal d'horloge. Signal, de forme généralement carrée, par rapport auquel les composants d'un système sont synchronisés.

Simplex (unidimensionnel). Se dit d'une méthode de transmission de données. Un « canal » ou une voie simplex ne permet qu'un sens de communication, par exemple de l'ordinateur au périphérique, ou inversement.

Software. Voir **Logiciel**.

Sortance. Terme désignant la capacité produite par un circuit, c'est-à-dire l'ordre de grandeur de la puissance que ce circuit peut délivrer. Elle s'exprime généralement par le nombre de circuits analogiques qu'il peut piloter. (Voir **Entrance**).

Sous-chaîne. Série de caractères extraite d'une chaîne.

Sous-programme. Partie d'un programme contenant des instructions utilisables par différentes autres parties y accédant à l'aide d'un saut avec retour (Return).

Structure. Terme désignant une série d'éléments ayant des liens logiques.

Structure de recherche. En Pascal, il s'agit d'une chaîne qui devra être comparée à d'autres pour déterminer si elle est contenue dans celles-ci.

Syntaxe. Ensemble de règles indiquant comment doivent être utilisées les instructions d'un langage déterminé. Elles peuvent être représentées synthétiquement à l'aide de diagrammes syntaxiques.

Système d'exploitation. Logiciel contrôlant l'exécution des programmes d'application, y compris des éventuels interpréteurs ou compilateurs, et les programmes de gestion des périphériques (voir **DOS**).

T

Table de correspondances. Désigne la description des liens de dépendance entre les différents enregistrements de fichiers.

Tableau. Ensemble d'éléments, à une ou à plusieurs dimensions, désigné par un nom symbolique et par un ou plusieurs indices. Un tableau à deux dimensions, par exemple, est une matrice dont chaque élément est défini par sa ligne et sa colonne.

Tableau bidimensionnel. Tableau de données à deux dimensions.

Tambour. Dans le cas des traceurs, le tambour est le dispositif sur lequel défile le papier. Le tambour peut aussi être un support de mémoire de masse.

Tampon. Voir **Mémoire-Tampon**.

Taux de succès. Exprime le nombre d'enregistrements que l'on pense devoir être traités en un passage. C'est généralement un pourcentage de la dimension du fichier (en nombre d'enregistrements).

Temps d'exécution. Période pendant laquelle un programme est activé.

Temps partagé. Méthode d'utilisation d'une même mémoire par plusieurs programmes.

Temps réel. Une application en temps réel est une tâche dont les actions, après la phase d'entrée, sont immédiatement exécutées.

Terminal. Dispositif d'entrée/sortie dans un réseau de télécommunications.

Théorème de Morgan. Théorème utilisé dans l'analyse des circuits numériques pour convertir des schémas à logique négative et vice versa.

Time sharing. Voir **Temps partagé**.

Topographie. Schéma ou description du contenu de la mémoire décrivant l'utilisation de ses différentes zones.

Total de contrôle. Totalisation de certaines zones des enregistrements d'un fichier afin d'obtenir une valeur numérique (sans signification en tant que donnée) permettant de contrôler qu'aucune donnée n'a été perdue ou modifiée. Une technique analogue est utilisée en transmission ; dans ce dernier cas, on parle parfois de procédure d'« apurement ».

Tracé élastique. Technique de génération d'une ligne consistant à en fixer une extrémité et à tracer l'autre en fonction des déplacements d'un périphérique d'entrée, par exemple un manche à balai.

Traitement par lots. Mode particulier d'exécution

d'un programme selon lequel le traitement est différé par rapport au moment de l'introduction des données.

Transaction. Opération provoquant la création ou la suppression d'un enregistrement.

Transfert anticipé. Technique visant à accélérer l'accès aux mémoires de masse ou, en général, à des périphériques lents. Elle consiste à déplacer des blocs entiers de données vers un périphérique plus rapide, avant même que le programme ne les demande.

Transfert sur demande. Procédé consistant à transférer un bloc de données d'un périphérique lent dans un périphérique rapide quand le programme veut l'utiliser. C'est le contraire du transfert anticipé.

Tri « à bulle » (par permutation de paires). Technique consistant à exécuter un tri, dans un tableau, par sélection d'une partie des éléments du tableau et par échange de positions. On obtient ainsi le classement désiré. Ce processus est répété jusqu'à ce que tout le contenu du tableau soit trié. Le nom « à bulle » (de l'anglais Bubble Sort) provient de la technique utilisée, qui consiste à prélever au fur et à mesure les valeurs les plus petites, c'est-à-dire les plus « légères », qui « montent » vers le sommet de la liste. Une version particulière du Bubble Sort, plus rapide si un tri partiel a déjà été effectué, consiste à considérer simultanément une valeur située vers le sommet de la liste et une valeur vers la fin de la liste. Elle est appelée « Cocktail Shaker Sort ».

Trois états. Se dit d'un dispositif électronique numérique qui peut se permettre à l'état 0, 1, ou à l'état de haute impédance.

TTY. Abréviation de Teletype (télescripteur).

U

UAL. Unité arithmétique et logique (anglais ALU) utilisée dans les systèmes à micro-ordinateur et chargée d'exécuter des calculs.

UART (Universal Asynchronous Receiver Transmitter). Dispositif programmable de gestion des fonctions d'E/S.

UC. Unité centrale de traitement (CPU, Central Processing Unit). Composant matériel commandant le fonctionnement des autres unités. Il effectue les calculs et gère les périphériques.

V

Validation. Opération généralement effectuée dans les programmes d'application et consistant à vérifier les données.

Validé. Etat d'un processus répondant à certaines conditions et permettant de ce fait des interruptions. (Contraire d'**Invalidation**).

Valorisé à l'exécution. Dans certains systèmes d'exploitation ou dans certains langages, se dit d'un module particulier contenant les informations nécessaires à l'utilisation d'un programme (run-time module en anglais).

Vidage. Opération d'entrée sur imprimante ou écran, sans traitement, du contenu d'une zone de mémoire ou d'une partie de disque.

Vide. Qualifie un ensemble, par exemple une chaîne, ne contenant que des blancs.

Virgule fixe. Définit une représentation de valeurs numériques réelles dans laquelle la virgule a une position définie.

Virgule flottante. Définit une représentation de valeurs numériques réelles sous forme d'une mantisse et d'un exposant.

Viseur. Dispositif permettant d'afficher une grandeur.

Vitesse de transfert. Vitesse à laquelle les données sont transférées entre un dispositif et l'unité centrale. Elle s'exprime normalement en milliers de caractères par seconde.

W

Walking-one. Technique de contrôle des mémoires.

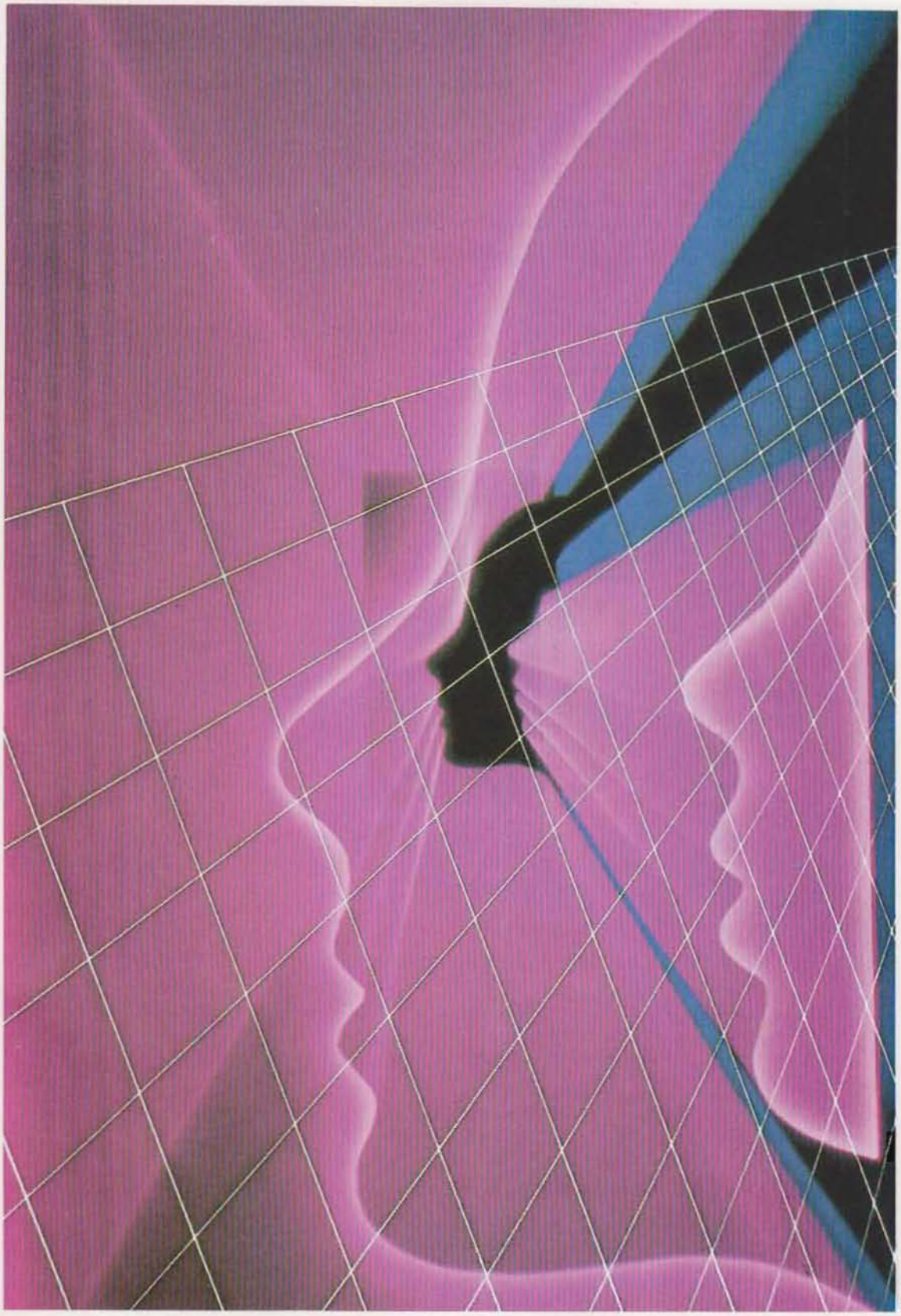
Window. Voir **Fenêtre**.

Z

Zone. Désigne généralement une partie de la mémoire, ou encore une position de mémoire de masse adressable dans une base de données. En multiprogrammation, une zone est allouée à chaque utilisateur pour ses traitements.

Zone de débordement. Désigne, dans le cas des fichiers, une zone particulière dans laquelle sont mémorisés les enregistrements qui ne peuvent pas être placés dans la zone logique à laquelle ils appartiennent.

Zoom (grossissement). Fonction de représentation grossie de parties d'un dessin.



Marks

1705

Table des matières

Volume 1

Page	**13**	L'ORDINATEUR, MACHINE INTELLIGENTE ?
»	15	**Les données**
»	22	**De la calculatrice à l'ordinateur : le programme**
»	22	**Passé et présent**
»	28	**Evolution des ordinateurs : applications**
»	**33**	LA LOGIQUE DE L'ORDINATEUR
»	33	**Les ordinateurs analogiques et les ordinateurs numériques**
»	37	**Où et comment utiliser un ordinateur**
»	38	**Classement des ordinateurs**
»	43	**Réseaux d'ordinateurs**
»	**45**	LES SYSTEMES DE NUMERATION
»	45	**Les impulsions**
»	46	**Commutation**
»	47	**Les bases de numération**
»	50	Notation binaire
»	52	Notation hexadécimale
»	55	Notation octale
»	56	**Opérations en notation binaire**
»	60	Addition binaire
»	61	Soustraction binaire
»	**72**	ELEMENTS DE LOGIQUE
»	72	**Les opérateurs**
»	73	**Les opérateurs logiques**
»	79	**Logique câblée**
»	80	Circuits logiques
»	80	Opérateur NON
»	81	Opérateur ET
»	89	Opérateur OU
»	89	Opérateur OUX
»	90	Autres types d'inverseurs
»	97	**Les circuits intégrés**
»	97	Les circuits composites
»	99	Les applications
»	100	Autres types de circuits intégrés
»	**107**	LES CODES DE TRANSMISSION DES DONNEES
»	107	**Bit et byte**
»	110	**Le code ASCII**
»	111	**Codes transparents**
»	114	**Symboles et nombres**
»	114	**Lettres**
»	115	**Applications**
»	117	**Transmission des codes ASCII**
»	121	**Contrôle de réception**
»	121	**Codes de sécurité**
»	**125**	MICROSYSTEMES
»	125	**Les principaux organes**
»	126	**Structure de l'unité centrale**
»	126	Fonctionnement de l'unité centrale
»	129	Phases d'une instruction de saut
»	132	Gestion des interruptions

Page 133 Lecture des données
» 139 **Les mémoires**
» 140 **Les périphériques**
» 140 Le moniteur
» 142 Le clavier
» 142 Les imprimantes
» 142 Unités à disques
» 146 **Les instructions dans l'UC**
» 147 Instructions de transfert des données
» 148 Opérations arithmétiques et logiques
» 148 Instructions de saut
» 148 Fonctions E/S (Entrée/Sortie)
» 148 Instructions spéciales

» **154** LA PROGRAMMATION

» 154 **Les langages de programmation**
» 154 Langages compilés et interprétés
» 156 Compilation
» 156 Basic
» 156 Fortran
» 156 Cobol

» **157** ANALYSE ET ORGANIGRAMMES

» 157 **Etablissement et écriture d'un programme**
» 158 Analyse
» 162 Synthèse
» 168 Etablissement des organigrammes
» 168 Symboles d'usage courant
» 170 Symboles graphiques moins utilisés
» 172 **Exemple d'application**
» 174 **Les boucles**
» 175 Boucle à limites explicites
» 175 Boucle à limites paramétrées
» 176 Boucle avec indicateur de fin de données
» 177 Boucle à rupture de code
» 186 Les boucles à pas différent de 1
» 190 Sortie forcée d'une boucle
» 193 **Utilisation des organigrammes**
» 200 Applications scientifiques
» 203 Applications de la méthode itérative
» 209 **Programmation structurée**

» **220** LE STOCKAGE DES DONNEES

» 220 **Les mémoires de masse**
» 220 Les bandes magnétiques
» 227 Les disques magnétiques
» 229 Les tambours magnétiques
» 229 Les disquettes ou disques souples
» 231 **Les fichiers**
» 236 Organisation des fichiers sur bande magnétique
» 238 Organisation des fichiers sur disque magnétique
» 238 Organisation séquentielle
» 238 Organisation séquentielle avec liens de drainage
» 240 Organisation séquentielle indexée
» 242 Organisation directe
» 246 L'organisation des fichiers sur les micro-ordinateurs
» 248 Les fichiers directs (ou aléatoires)
» 248 Les fichiers séquentiels
» 248 Les fichiers indexés
» 254 **Méthode ISAM de gestion d'un fichier indexé**
» 255 Les opérations d'accès aux données
» 256 La structure générale du fichier ISAM
» 256 La structure en arbre binaire équilibré
» 258 La structure en arbre ISAM

» 258 Le nombre de fichiers ouverts et les zones tampon
» 259 L'adressage indirect
» 259 Subdivision
» 260 Multiplication
» 260 Multiplication par 11
» 260 Puissance deux
» 261 **Exemple d'application : la gestion du budget familial**
» 261 Plan des comptes
» 262 Le module de calcul du compteur d'enregistrements
» 269 Le module de contrôle des données en entrée
» 271 **La recherche des données dans les fichiers**
» 273 La recherche séquentielle
» 275 La recherche dichotomique
» 275 Le classement des données
» 283 Le compactage des données
» 283 Les banques de données des gros systèmes
» 285 Les bases de données relationnelles

Volume 2

» **289** ETABLISSEMENT DES PROGRAMMES

» 289 **Définition des sorties nécessaires**
» 290 **Vérification des données en entrée et des algorithmes de calcul**
» 291 **Choix du matériel**
» 291 **Les systèmes d'exploitation**
» 292 Les fonctions du système d'exploitation
» 292 Le module de traitement des commandes
» 296 La gestion des fichiers sur disque souple
» 296 Le formatage
» 298 Les extensions de données
» 299 Les opérations de gestion des fichiers du système d'exploitation
» 311 Les opérations de passage sous Basic
» 314 Critères d'évaluation d'un système d'exploitation
» 314 **Subdivision des programmes**
» 316 Transfert des paramètres

» **317** LE LANGAGE DE PROGRAMMATION BASIC

» 320 **Généralités**
» 320 Les modes opératoires
» 322 Structure des instructions Basic
» 322 Les fonctions de la touche CTRL
» 323 Utilisation interactive du Basic (mode direct ou immédiat)
» 323 **Symbolique, signification et priorité des opérateurs**
» 323 Opérateurs arithmétiques
» 326 Opérateurs relationnels
» 326 Opérateurs logiques
» 332 Opérateurs fonctionnels
» 337 Opérateurs de division entière
» 337 Opérateurs modulo
» 338 **Constantes et variables**
» 338 Constantes numériques
» 339 Précisions
» 340 Variables numériques
» 343 Conversion des différents types de variables
» 345 Arrondis dans les affectations
» 345 Arrondis dans les calculs
» 347 Arrondis dans les opérations logiques
» 347 Les variables structurées
» 347 Tableaux
» 348 Les chaînes de caractères
» 352 Les portions de chaînes
» 353 **Les commandes**
» 353 Commandes d'écriture ou de sauvegarde des programmes

Page 363 — Commandes de la bande magnétique
» 363 — Affichage et listage des programmes
» 363 — Exécution, interruption et mise au point des programmes
» 364 — **Les instructions**
» 364 — Concaténation
» 366 — Début et fin des programmes
» 368 — Déclaration du type de variables
» 370 — **Instructions d'affectation**
» 370 — LET
» 372 — DATA, READ, RESTORE
» 382 — LSET et RSET
» 382 — SPACE$(N)
» 384 — SPC(N)
» 385 — SWAP
» 385 — Instructions de boucle
» 386 — FOR... NEXT...
» 393 — WHILE... WEND
» 396 — Instructions de saut
» 397 — GOTO...
» 397 — ON... GOTO...
» 401 — Instructions conditionnées
» 409 — Utilisation des variables réelles dans les instructions conditionnelles
» 411 — Appel d'un sous-programme et enchaînement de programmes
» 413 — **Exemple d'application : gazole ou essence ?**
» 417 — **Les diagnostics d'erreurs**
» 434 — **Les fonctions**
» 435 — Définition d'un programme par points
» 437 — Interpolation et extrapolation
» 439 — Représentation analytique des fonctions
» 440 — **Les fonctions du Basic**
» 440 — Les fonctions mathématiques
» 440 — ABS(R)
» 440 — CDBL(R)
» 440 — CINT(R)
» 441 — CSNG(R)
» 441 — FIX(R)
» 443 — INT(R)
» 444 — SNG(R)
» 444 — SQR(R)
» 444 — RND(R)
» 447 — ATN(Y)
» 447 — Les fonctions trigonométriques
» 447 — COS(R)
» 447 — SIN(R)
» 447 — TAN(R)
» 448 — Applications des fonctions trigronométriques
» 449 — EXP(R)
» 450 — LOG(R)
» 450 — Fonctions permettant le traitement des chaînes de caractères
» 450 — ASC(A$)
» 450 — CHR$(N)
» 452 — CVI(A$)
» 454 — CVS(A$)
» 454 — CVD(A$)
» 454 — HEX$(N)
» 455 — INKEY$
» 455 — INPUT$
» 455 — INSTR(N,A$,B$)
» 457 — LEFT$(A$,N)
» 458 — LEN(A$)
» 458 — MID$(A$,NS,NC)
» 460 — MKI$(N)
» 460 — MKS$(R)
» 460 — MKD$(D)
» 460 — OCT$(N)

Page 461 RIGHT$(A$,N)
» 465 STR$(R)
» 469 STRING$(N,K)
» 469 VAL(A$)
» 469 Fonctions spéciales
» 469 EOF(N)
» 470 FRE(A)
» 470 INP(N)
» 470 LOC(N)
» 471 LPOS(N)
» 472 OUT N,M
» 472 PEEK(N)
» 472 POKE N,M
» 473 POS(N)
» 473 Fonctions définies par l'utilisateur
» 481 Fonctions numériques : résolution d'équations du second degré
» 485 Fonctions sur chaînes : contrôle des minuscules et des majuscules
» 488 Domaines d'utilisation
» 491 **Exemple d'application : le chauffage des locaux**
» 496 Analyse générale du problème
» 501 Les sous-programmes
» 505 Le programme
» 520 **Les tableaux**
» 521 Organisation des tableaux
» 522 Tableau à une et à N dimensions
» 522 Instructions d'affectation sur des tableaux
» 529 Application des tableaux
» 531 Exécution de calculs avec un nombre quelconque de chiffres significatifs
» 537 Applications à la statistique
» 547 **Les fonctions d'entrées/sorties des données**
» 548 Fonctions de saisie des données
» 548 INPUT
» 552 INKEY$
» 554 INPUT$(N)
» 555 LINE INPUT
» 555 Fonctions de sortie des données
» 556 LPRINT
» 557 TAB(N)
» 557 LPRINT USING
» 560 Structuration des impressions
» 560 Impressions non paramétrées
» 566 Impressions paramétrées

Volume 3

» 577 **Gestion de l'imprimante**
» 577 Caractéristiques techniques des imprimantes à aiguilles
» 581 Exemple d'impression
» 588 Présentation des diagrammes à barre
» 594 **Gestion de la console vidéo**
» 595 Structure et caractéristiques d'une console vidéo
» 595 CONVERSATIONNEL
» 595 PAGE
» 595 SCROLL
» 596 PROTECT
» 596 Gestion du curseur
» 596 Les masques de saisie
» 605 Les touches de fonction
» 616 Les éditeurs de textes
» 617 Les fonctions d'édition et de recherche des chaînes de caractères
» 617 Les menus
» 630 Génération d'histogrammes
» 637 **Gestion des fichiers**
» 637 Fonctions d'accès aux fichiers séquentiels
» 638 INPUT

Page 638 — LINE INPUT
» 638 — WRITE
» 639 — PRINT et PRINT USING
» 639 — EOF(N)
» 639 — LOC(N)
» 639 — KILL
» 639 — NAME
» 639 — Insertion des données dans un fichier séquentiel
» 642 — Fonctions d'accès aux fichiers en accès direct
» 642 — OPEN
» 643 — FIELD
» 643 — CLOSE
» 643 — PUT
» 652 — GET
» 652 — LOC(N)
» 654 — Formats et codifications des données
» 659 — Sélection des données
» 661 — Commandes et fonctions spéciales
» 661 — Fichiers multivolumes
» 662 — Gestion des fichiers sur les grands systèmes
» 662 — La gestion d'une entreprise
» 665 — Réévaluation des stocks et inventaires
» 665 — Fichier Nomenclature
» 665 — Simulation
» 666 — Relations entre procédures
» 666 — **Exemple d'application : la gestion d'un répertoire**
» 666 — Ouverture du fichier (sous-programme 1000)
» 670 — Menu (sous-programme 2000)
» 670 — Saisie (sous-programme 3000)
» 670 — Mise à jour (sous-programme 4000)
» 672 — Recherche (sous-programme 5000)
» 673 — Impression (sous-programme 6000)
» 673 — Opérations d'impressions particulières
» 675 — **L'agencement des données**
» 676 — Tri en mémoire
» 680 — Tri d'un fichier sur disque
» 683 — Le sous-programme utilitaire OLISORT
» 689 — Les prestations
» 692 — Les modalités d'exécution
» 695 — **Conception d'un programme de gestion de données**
» 695 — L'acquisition des données
» 695 — Préparation du masque de saisie
» 699 — Affichage du masque de saisie
» 703 — Entrée des données
» 708 — La phase de mémorisation sur disque
» 709 — L'écriture (instructions 1242 à 1272)
» 712 — La lecture (lignes 1228 à 1238)
» 712 — Le programme principal
» 717 — **La compilation**
» 717 — La préparation
» 717 — BASCOM.COM
» 717 — BRUN.COM
» 724 — BASLIB.REL
» 724 — OBSLIB.REL
» 724 — BCLOAD
» 724 — L80.COM
» 724 — La compilation
» 725 — Objet
» 725 — Liste
» 725 — Source
» 727 — Les fonctions particulières du compilateur
» 730 — Spécification des conventions
» 730 — Fonctions de traitement des erreurs
» 731 — Codes spéciaux
» 731 — L'édition de liens

Page 733 **Instructions particulières complétant le Basic 80**
» 733 Instructions de portée générale
» 733 CALL
» 736 COMMON
» 736 USR
» 738 STOP
» 738 Instructions de tracés graphiques
» 740 Instructions particulières
» 740 Synthétiseur
» 740 Joystick et paddle
» 741 STICK(N)
» 741 STRING
» 741 ON STRING...
» 741 Le stylo optique
» 741 PEN ON
» 741 PEN OFF
» 741 Z=PEN(N)
» 742 **Précis du Basic 80**
» 742 Instructions et commandes
» 744 Fonctions
» 746 **Variantes du Basic**
» 746 Gestion des chaînes de caractères
» 747 Gestion des fichiers sur disque
» 753 Gestion des opérations d'E/S
» 755 Gestion des périphériques : l'interfaçage
» 760 Transmission et réception des données

» **769** LE LOGICIEL

» 770 **L'architecture du logiciel**
» 772 La modélisation
» 773 Le domaine macroscopique
» 774 Le modèle microscopique
» 776 **La structuration des programmes**
» 776 La méthode de développement descendante
» 784 **Les erreurs du logiciel**
» 784 **Test et contrôle du logiciel**
» 786 Les méthodes de test
» 787 La méthode ascendante
» 788 La méthode descendante
» 788 La méthode bing-bang
» 788 La méthode sandwich
» 788 Les diagrammes de cause-effet
» 789 La mise au point
» 791 **La sécurité du logiciel**
» 793 **Les logiciels d'application généralisée**
» 793 **Le tableur**
» 798 La saisie des données
» 800 Le mode commandes
» 800 FORMAT
» 801 GLOBAL
» 801 TITLES
» 813 **Les fonctions du tableur**
» 813 Fonctions mathématiques
» 814 Fonctions numériques spéciales
» 814 AVERAGE (n1,n2...)
» 817 COUNT(NS...NE)
» 817 SUM(NS...NE)
» 818 MAX(NS...NE),MIN(NS...NE)
» 818 NPV(S,NS...NE)
» 818 Fonctions de recherche
» 818 LOOKUP(M,NS...NE)
» 820 Fonctions logiques
» 820 AND (listes d'expressions)
» 820 IF (condition vrai/faux)

Page 822 NOT (condition)
» 822 ISNA (cellule)
» 822 ISERROR (cellule)
» 823 OR (conditions)
» 823 Fonctions sans arguments
» 823 PI
» 823 ERROR
» 823 FALSE/TRUE
» 823 NA
» 825 **Exemples d'utilisation du tableur**
» 825 Répartition des frais de copropriété
» 832 Gestion d'un magasin
» 835 Analyse des variations (WHAT IF)
» 843 Les fonctions par le calcul itératif
» 843 ITERCNT ( )
» 843 DELTA ( )
» 845 **Evolution du tableur**
» 845 Fonctions Multiplan et leurs équivalents Visicalc
» 847 **Les programmes de gestion des fichiers**
» 847 Création de fichiers et définition des zones
» 849 Introduction et mise à jour des données
» 849 Recherche (FIND)
» 851 Tri (SORT)
» 851 Traitement des données
» 858 Edition d'états
» 858 Traitement de textes

Volume 4

» **865** **LE LANGAGE ASSEMBLEUR**

» 866 **Structure et fonctionnement du microprocesseur**
» 870 Les registres
» 871 Fonctionnement de l'unité centrale de traitement
» 876 L'architecture interne du mircoprocesseur
» 877 L'unité arithmétique et logique (UAL)
» 877 Addition binaire
» 881 Soustraction binaire
» 881 Complémentation
» 882 Multiplication binaire
» 884 Division binaire
» 885 Opérations booléennes
» 888 Opérations de décalage
» 889 Le registre d'état
» 890 Retenue
» 890 Zéro
» 890 Dépassement
» 891 Signe
» 891 Demi-retenue
» 891 Parité
» 892 Indicateurs spécifiques
» 893 Le tampon de l'UAL
» 893 L'unité de contrôle
» 896 **L'exécution des instructions**
» 896 La synchronisation des signaux
» 903 La gestion de l'interruption
» 905 La pile
» 910 Structure des instructions Assembleur
» 911 Mode d'adressage
» 912 Adressage immédiat
» 912 Adressage absolu ou direct
» 913 Adressage indirect
» 915 Adressage indexé

Page 915 — Adressage relatif
» 917 — Adressage par registre
» 917 — Adressage par pile
» 918 — **Les instructions Assembleur**
» 919 — Instructions de test
» 920 — Instructions arithmétiques
» 922 — Instructions logiques
» 922 — Instructions de transfert des données
» 929 — Instructions de branchement (Branch)
» 930 — Instructions de saut (Skip)
» 931 — Instructions d'appel de sous-programme et de retour
» 932 — Instructions diverses
» 932 — NOP
» 933 — PUSH et POP
» 933 — HALT et WAIT
» 933 — Interruption (Break)
» 933 — Autorisation d'interruption (Enable Interrupt) et interdiction d'interruption (Disable Interrupt)
» 934 — Adjust et Translate
» 934 — Les directives d'assemblage
» 942 — **Exemple de programmation en Assembleur**
» 945 — **L'Assembleur du Rockwell 6502**
» 947 — **L'Assembleur du Zilog Z80**

» **951** — **LE LANGAGE COBOL**

» 952 — **Généralités**
» 953 — Le programme source
» 956 — Compilation et édition de liens
» 957 — La feuille de programmation Cobol et les règles d'écriture
» 960 — **Structure d'un programme Cobol**
» 964 — **IDENTIFICATION DIVISION**
» 965 — **ENVIRONMENT DIVISION**
» 965 — CONFIGURATION SECTION
» 965 — SPECIAL-NAMES
» 966 — FILE-CONTROL
» 967 — **DATA DIVISION**
» 967 — FILE SECTION
» 968 — Description des données : la clause BLOCK CONTAINS
» 968 — La clause LABEL RECORD
» 969 — La clause RECORDING MODE
» 970 — La clause RECORD CONTAINS
» 970 — Structure des enregistrements
» 973 — WORKING-STORAGE SECTION
» 973 — Le niveau 77
» 974 — Le niveau 88
» 974 — Le niveau 66 et la clause RENAMES
» 975 — La clause REDEFINES
» 976 — La clause PICTURE
» 978 — Description des champs numériques
» 979 — Description des champs alphabétiques
» 979 — Description des champs alphanumériques
» 981 — **PROCEDURE DIVISION**
» 988 — Exemple de PROCEDURE DIVISION
» 992 — **Les instructions du Cobol**
» 994 — Verbes d'E/S
» 995 — Ouverture et fermeture des fichiers : les verbes OPEN et CLOSE
» 996 — Utilisation en lecture : OPEN INPUT
» 997 — Utilisation en écriture : OPEN OUTPUT
» 997 — Utilisation en lecture-écriture : OPEN E/S
» 997 — OPEN EXTEND
» 1000 — Lecture et écriture sur les fichiers : les verbes READ et WRITE
» 1003 — Ecriture sur fichiers d'imprimante
» 1003 — Clause de saut de lignes
» 1004 — Clause de saut de pages

Page 1004 — La clause LINAGE
» 1006 — Les instructions ACCEPT et DISPLAY
» 1009 — **Verbes arithmétiques**
» 1009 — Représentation des données en mémoire : la clause USAGE
» 1011 — USAGE DISPLAY
» 1011 — USAGE COMPUTATIONAL
» 1012 — USAGE COMP-3
» 1013 — USAGE COMP-1 et COMP-2
» 1014 — Les expressions arithmétiques
» 1015 — L'instruction COMPUTE
» 1020 — Le verbe ADD
» 1022 — Le verbe SUBTRACT
» 1023 — Le verbe MULTIPLY
» 1023 — Le verbe DIVIDE
» 1025 — **Verbes de transfert et de manipulation des données**
» 1025 — Le verbe MOVE
» 1027 — MOVE alphanumérique
» 1030 — La clause ALL
» 1036 — MOVE numérique
» 1040 — L'instruction INSPECT
» 1040 — La fonction de comptage
» 1040 — La clause ALL
» 1041 — La clause LEADING
» 1041 — La clause CHARACTERS
» 1041 — La clause BEFORE
» 1043 — La clause AFTER
» 1043 — La fonction de substitution
» 1044 — Format général de l'instruction INSPECT
» 1046 — L'instruction STRING
» 1049 — L'instruction UNSTRING
» 1051 — **Verbes de contrôle**
» 1052 — La proposition IF
» 1053 — Logique d'exécution de l'instruction IF
» 1056 — L'option NEXT SENTENCE
» 1057 — Conditions liées à l'instruction IF
» 1058 — Analyse de relation
» 1064 — Analyse de signe
» 1064 — Analyse de condition
» 1067 — Analyse de classe
» 1068 — Emploi des opérateurs logiques
» 1073 — L'instruction GO TO
» 1075 — La clause DEPENDING ON
» 1076 — Le verbe PERFORM
» 1084 — L'instruction EXIT
» 1084 — Exécutions itératives de la même procédure
» 1085 — La clause UNTIL
» 1085 — La clause VARYING... FROM... BY
» 1085 — L'instruction STOP
» 1086 — **Gestion des tableaux en Cobol**
» 1089 — Tableaux à une dimension
» 1095 — Tableaux à deux dimensions
» 1098 — Tableaux à trois dimensions
» 1103 — Chargement d'un tableau
» 1111 — Les indices des tableaux
» 1111 — Le verbe SET pour la gestion d'un indice
» 1112 — Le verbe SEARCH pour la recherche dans les tableaux
» 1113 — L'instruction SEARCH
» 1115 — Emploi du SEARCH séquentiel
» 1119 — L'instruction SEARCH ALL (recherche binaire)
» 1121 — Classement d'un tableau
» 1124 — **Classement des données (SORT)**
» 1127 — L'INPUT PROCEDURE et le verbe RELEASE
» 1127 — L'OUTPUT PROCEDURE et le verbe RETURN
» 1129 — **Gestion des fichiers annexes**
» 1133 — Description des fichiers IS dans ENVIRONMENT DIVISION

Page 1133 Description des fichiers dans DATA DIVISION
» 1133 Les instructions dans PROCEDURE DIVISION
» 1133 Le verbe READ
» 1134 Le verbe WRITE
» 1134 Instructions spécifiques aux fichiers IS
» 1135 Le verbe REWRITE
» 1135 Le verbe DELETE
» 1136 Le verbe START
» 1138 Le verbe CLOSE
» 1138 **Communication entre programmes. Les routines**
» 1145 Le verbe CALL
» 1145 PROCEDURE DIVISION d'un sous-programme
» 1145 L'instruction EXIT PROGRAM
» 1147 Exemples d'application

Volume 5

» **1153** **LE LANGAGE FORTRAN**

» 1154 **Caractéristiques fondamentales du Fortran comparé au Basic**
» 1155 **Opérateurs arithmétiques, logiques et relationnels**
» 1156 **Variables et constantes en Fortran**
» 1158 Déclaration du type de variables
» 1160 Dimensionnement des variables structurées
» 1160 Les instructions COMMON et EQUIVALENCE
» 1162 Instructions d'assignation
» 1163 **Les instructions Fortran**
» 1163 L'instruction GOTO
» 1164 Les boucles
» 1164 Boucles implicites
» 1166 La forme DO... WHILE...
» 1166 Instructions conditionnelles
» 1168 L'emploi des sous-programmes (ou routines)
» 1169 Instructions de fin et d'arrêt de l'exécution
» 1170 **Les fonctions du Fortran**
» 1170 Fonctions de bibliothèque
» 1170 Conversion de type
» 1171 Fonctions de calcul
» 1172 Fonctions de chaîne
» 1172 Fonctions lexicales
» 1173 Fonctions mathématiques
» 1173 Fonctions programmables par l'utilisateur
» 1175 **Les instructions et les formats d'E/S**
» 1176 PRINT
» 1176 WRITE
» 1178 READ
» 1178 Format d'E/S
» 1178 Formats pour les entiers
» 1179 Formats pour les réels
» 1180 Formats pour les caractères
» 1180 Formats pour les constantes logiques
» 1180 Descripteurs de mise en page
» 1180 Mise en page en entrée
» 1180 Mise en page en sortie
» 1180 Descripteurs communs
» 1180 Facteurs d'échelle
» 1181 Répétitions de format
» 1181 Instructions d'E/S non formatées
» 1181 Instructions d'entrée
» 1182 Instructions de sortie
» 1182 **Exemples de programmation en Fortran**
» 1186 **Gestion de fichiers**
» 1190 Gestion des fichiers sur disque
» 1191 OPEN
» 1193 READ et WRITE

Page 1193 BACKSPACE
» 1193 REWIND
» 1193 ENDFILE
» 1193 INQUIRE
» 1194 CLOSE
» 1194 **Instructions particulières au Fortran**
» 1194 EXTERNAL
» 1196 ENTRY
» 1196 SAVE
» 1197 BLOCKDATA
» 1197 Améliorations spéciales
» 1198 **Format normal des instructions**

» **1202** **LE LANGAGE PASCAL**

» 1205 **La programmation structurée**
» 1205 L'organisation des programmes
» 1206 Le pseudocode
» 1207 Les organigrammes structurés
» 1212 Les structures de contrôle
» 1213 Séquence
» 1214 Sélection
» 1214 Répétition
» 1216 **Les données en Pascal**
» 1216 Types de données
» 1217 Le type entier
» 1220 Le type réel
» 1221 Le type caractère
» 1227 Le type booléen
» 1227 Les déclarations de type
» 1228 La déclaration de constante
» 1228 La déclaration de variable
» 1230 Types de données scalaires non standard
» 1231 Types de données définies par l'utilisateur
» 1232 Types de données sous-ensembles
» 1233 **Les premières instructions du Pascal**
» 1233 Expressions arithmétiques et booléennes
» 1234 Emploi des fonctions Pascal
» 1235 L'instruction d'assignation
» 1239 Les instructions d'entrée : READ et READLN
» 1240 Les instructions de sortie : WRITE et WRITELN
» 1243 **Structure d'un programme Pascal**
» 1249 **Les instructions de contrôle**
» 1249 Les instructions d'itération
» 1249 L'instruction FOR-TO-DO
» 1251 L'instruction WHILE-DO
» 1252 L'instruction REPEAT-UNTIL
» 1254 Les instructions conditionnelles
» 1254 L'instruction IF-THEN-ELSE
» 1257 L'instruction CASE-OF
» 1259 L'instruction de saut inconditionnel : GOTO
» 1260 **Aspects particuliers du Pascal**
» 1262 **Exemples de programmation en Pascal**

» **1266** **RESUME : DEVELOPPEMENT D'UNE FONCTION DE FACTURATION**

» 1266 **Analyse du problème**
» 1273 **Fichiers utilisés**
» 1273 **Menu principal**
» 1277 **Gestion des fichiers**
» 1286 **Emission de factures**
» 1303 **Impression de la facture**
» 1305 **Autres fonctions**

» **1316** **LA TRANSMISSION DES INFORMATIONS**

» 1317 **Modes de transmission**

Page 1320 — Canaux simplex, demi-duplex, duplex
» 1321 — **Transmission par les lignes téléphoniques**
» 1321 — Caractéristiques des signaux
» 1325 — Techniques de modulations. Les modems
» 1326 — Configuration des liaisons
» 1328 — Configuration point à point
» 1328 — Configuration multipoint
» 1329 — Configuration multiplex
» 1330 — Interfaces de communication
» 1334 — **Communications à faible distance**
» 1337 — **Protocoles de communication**
» 1339 — Le "handshake"
» 1340 — La méthode XON/XOFF
» 1341 — La méthode ENQ/ACK
» 1349 — Le protocole BSC
» 1351 — Le protocole HDLC
» 1353 — Le protocole IEEE-488
» 1355 — **Réseaux pour la communication de données**
» 1356 — Topologie et caractéristiques des réseaux
» 1356 — Réseau en chaîne
» 1356 — Réseau étoilé
» 1356 — Réseau en anneau
» 1356 — Réseau arborescent ou hiérarchisé
» 1360 — Réseaux publics
» 1360 — Réseaux locaux
» 1362 — Vers des architectures standard
» 1364 — **Instruction en Basic pour la communication des données**

» **1368** — **SYSTEMES D'EXPLOITATION**

» 1368 — **Mono-utilisateur et multi-utilisateur**
» 1369 — **Traitement par lots et interactivité**
» 1369 — **Multiprogrammation**
» 1371 — **Le temps partagé**
» 1372 — **Traitement en temps réel**
» 1373 — **Systèmes de gestion des fichiers**
» 1374 — Structure du gestionnaire fichiers Unix
» 1378 — **Processus simultanés**
» 1379 — Communication entre les processus

» **1383** — **L'INFORMATIQUE GRAPHIQUE EN BASIC**

» 1384 — **Applications de l'informatique graphique**
» 1384 — Le graphique de décision
» 1384 — La conception assistée par ordinateur (CAO)
» 1384 — La conception mécanique
» 1385 — La conception électronique
» 1385 — La conception architecturale
» 1386 — Le design industriel
» 1386 — Le traitement des images
» 1387 — L'animation
» 1387 — **Les terminaux graphiques**
» 1388 — Dispositifs d'entrée
» 1389 — La tablette graphique
» 1394 — Dispositifs spéciaux
» 1396 — Dispositifs de sortie
» 1396 — L'imprimante graphique
» 1398 — La table traçante
» 1405 — Le moniteur monochrome
» 1408 — Les moniteurs couleur
» 1411 — Matériel spécialisé
» 1414 — **Instructions Basic pour le graphique**
» 1414 — Tracé des segments
» 1418 — Rotation d'un segment
» 1427 — Tracé d'une circonférence
» 1432 — **Visualisation du graphique d'une fonction**
» 1433 — Tracé du graphique d'une fonction par segments
» 1436 — Tracé d'une droite

Volume 6

» 1441 Visualisation des axes cartésiens
» 1443 Tracé d'une droite à l'aide de deux points
» 1448 Régression linéaire des moindres carrés
» 1462 Programme du tracé graphique d'une fonction
» 1465 Calcul et arrondi du facteur d'échelle
» 1465 Recherche des valeurs extrêmes de Y
» 1465 Présentation du graphique
» 1465 Gestion des erreurs
» 1482 **Visualisation des caractères en mode graphique**
» 1502 **Histogrammes et diagrammes circulaires**
» 1530 **Gestion de la mémoire en mode graphique**
» 1530 La mémoire graphique
» 1532 Adressage de la mémoire vidéo
» 1533 Adressage le long de l'axe
» 1537 Adressage selon axe X
» 1539 **L'adressage direct de la mémoire**
» 1543 Stockages des images graphiques
» 1555 Calcul de zones de paramètres
» 1564 Instructions Basic pour l'archivage des figures
» 1564 POINT
» 1564 GET
» 1565 PUT
» 1565 **Les fenêtres**
» 1566 Programme de génération de fenêtres
» 1587 Instructions en Basic de gestion des fenêtres vidéo
» 1590 **Vecteurs graphiques et tables des figures**
» 1590 Codages des vecteurs de déplacement
» 1597 Programme de génération et de gestion de tables de figures
» 1605 Gestion des départements d'ensemble (sous-programme 4100)
» 1605 Déplacement de la figure (sous-programme 6000)
» 1610 Rotation de la figure (sous-programme 4500)
» 1611 Enregistrement sur le disque (sous-programme 5000)
» 1611 Lecture sur le disque (sous-programme 5200)
» 1614 Programmes utilitaires pour la création de formes graphiques
» 1614 L'éditeur de forme
» 1623 Le chargeur de formes
» 1623 Instructions Basic de gestion des figures graphiques
» 1623 DRAW
» 1631 XDRAW
» 1632 ROT
» 1632 SCALE
» 1633 **Graphisme tridimensionnel (3D)**
» 1633 Les fonctions à deux variables
» 1635 Représentation 3D
» 1637 Les problèmes du graphismes 3D
» 1658 **L'animation d'images par ordinateur**
» 1658 Les objets animés
» 1658 Pilotage des objets animés par logiciel
» 1658 L'animation d'images graphiques
» 1661 Visualisation d'un objet animé
» 1663 Problèmes d'animation
» 1669 Gestion matérielle des objets animés
» 1669 La mémorisation des objets animés
» 1673 La visualisation des objets animés
» 1674 Le déplacement de l'objet
» 1674 Utilisation de la mémoire pour la gestion des objets animés
» 1674 Exemple d'application
» 1679 Gestion évoluée des objets animés
» 1679 Définition des objets animés
» 1679 Visualisation
» 1685 Contrôle de collision
» **1688** **GLOSSAIRE**

Index des mots réservés

FONCTIONS DU SYSTEME D'EXPLOITATION

SEARCH, p. 304
OPEN, p. 304
CLOSE, p. 304
RENAME, p. 304
READ, p. 305
WRITE, p. 305
SELECT, p. 305
ERA, p. 305
DIR, p. 305
REN, p. 307
SAVE, p. 307
TYPE, p. 307
STAT, p. 307
ASM, p. 307
DDT, p. 307
LOAD, p. 307
PIP, p. 310
ED ou EDIT, p. 310
FORMAT, p. 310

BASIC

NOT, AND, OR, XOR (opérateurs logiques), pp. 326-331
DEF FNA (opérateurs fonctionnels), pp. 332-335
MBASIC, p. 354
NEW, p. 354
AUTO, pp. 353-354
SAVE, pp. 353-354
EDIT, pp. 356-358
CSAVE, CLOAD, p. 363
LIST, LLIST, p. 363
CONT, p. 363
TRON, TROFF, pp. 363-417
MERGE, p. 364
CLEAR, p. 364
OPTION BASE 1, p. 368
REM, p. 368
DEFINT, DEFSNG, DEFDBL, DEFSTR, p. 370
LET, p. 370
PRINT, p. 371
INPUT, pp. 371, 548-550
DATE, READ, RESTORE, pp. 372-382
LSET, RSET, p. 382
SPACE$ (N), p. 382
SPC (N), p. 384
SWAP, p. 385
FOR..., NEXT..., p. 386
WHILE... WEND..., p. 393
GOTO, p. 396
ON... GOTO..., p. 397
IF... THEN... ELSE, p. 408
GOSUB, p. 411
CHAIN, p. 411
ALL, p. 412
COMMON, p. 412
DELETE, p. 413
RETURN, p. 417
ON ERROR GOTO, p. 418
ABS (R), p. 440
CDBL (R), p. 440
CINT (R), p. 440
CSNG (R), p. 441
FIX (R), p. 441
INT (R), p. 443
SNG (R), p. 443
SQR (R), p. 443
RND (R), p. 444
RANDOMIZE, p. 446
ATN (Y), p. 447
COS (R), p. 447
SIN (R), p. 447
TAN (R), p. 447
EXP (R), p. 449
LOG (R), p. 450
ASC (A$), p. 450
CHR $ (N), p. 450
CVI (A$), p. 452
CVS (A$), p. 454
CVD (A$), p. 454
HEX $ (N), p. 454
INKEY $, pp. 455, 550-552
INPUT $ (N), pp. 455, 554
INSTR (N, A$, B$), p. 455
LEFT $ (A$, N), p. 457
LEN (A$), p. 458
MID $ (A$, NS, NC), p. 458
MKI $ (N), p. 460
MKS $ (N), p. 460
MKD $ (N), p. 460
OCT $ (N), p. 460
RIGHT $ (A$, N), p. 460
STR $ (R), p. 465
STRING $ (N, K), p. 469
VAL (A$), p. 469
EOF (N), p. 469
FRE (A), p. 470
INP (N), p. 470
LOC (N), p. 470
LPOS (N), p. 471
OUT N,M, p. 472
PEEK (N), p. 472
POKE N,M, p. 472
POS (N), p. 473
DIM NOME (N), pp. 520-522
LINE INPUT, p. 555
LPRINT, p. 556
TAB (N), p. 557
LPRINT USING, pp. 557-559
OPEN, pp. 637-642
INPUT #, p. 638
LINE INPUT #, p. 638
WRITE #, p. 638
PRINT #, p. 639
EOF (N), p. 639
LOC (N), pp. 639-652
KILL, p. 639
NAME, p. 639
FIELD, p. 643
CLOSE #, p. 643
PUT #, p. 643
GET #, p. 652

COMPILATION

BASCOM.COM, p. 717
BRUN.COM, p. 717
BASLIB.REL, p. 724
OBSLIB.REL, p. 724
BCLOAD, p. 724
L80.COM, p. 724

INSTRUCTIONS PARTICULIERES AU BASIC 80

CALL, p. 733
COMMON, p. 736
USR, p. 736
STOP, p. 738
STICK, p. 741
STRING, p. 741
ON STRING, p. 741
PEN ON, PEN OFF, p. 741
Z = PEN (N), p. 741
Précis du Basic 80, pp. 742-745
ALLOCATE, p. 746
DEALLOCATE, p. 746

COMMANDES DOS

CATALOG, p. 747
LOAD, p. 747
SAVE, p. 747
INIT, p. 747
DELETE, p. 748
LOCK, p. 748
RENAME, p. 748
VERIFY, p. 748
Mode commandes, p. 800
Fonctions Multiplan et leurs équivalents Visicalc, pp. 845-846

ASSEMBLEUR

ADD A, #, p. 912
ADD A,/, p. 912
ADDC, p. 919
BGT, p. 919
Z, p. 919
NZ, p. 919
Tests conditionnels, p. 920
Instructions arithmétiques, p. 921
Instructions logiques, pp. 923-924
Instructions de transfert des données, pp. 926-927
Instruction de saut, pp. 928-929
Instructions d'appel à sous-programme, p. 932
Instructions de retour, p. 932
Instructions diverses, p. 933
Directives d'assemblage, p. 934
Instructions Assembleur du Rockwell 6502, p. 947
Instructions Assembleur du Zilog Z80, pp. 949-950

COBOL

Constante figurative, p. 993
Verbes OPEN et CLOSE, p. 995
Verbes READ et WRITE, p. 1000
Instructions ACCEPT et DISPLAY, p. 1006
Clause USAGE, p. 1009
USAGE DISPLAY, p. 1011
USAGE COMPUTATIONAL, p. 1011
USAGE COMP-3, COMP-1, COMP-2, pp. 1012-1013
Expressions arithmétiques, p. 1014
Instruction COMPUTE, p. 1015
Verbe ADD, p. 1020
Verbe SUBTRACT, p. 1022
Verbe MULTIPLY, p. 1023
Verbe DIVIDE, p. 1023
Verbe MOVE, p. 1025
Instruction MOVE, p. 1039
Instruction INSPECT, p. 1040
Instruction STRING, p. 1046
Instruction UNSTRING, p. 1049
Proposition IF, p. 1052
Instruction GO TO, p. 1073
Verbe PERFORM, p. 1073
Verbe SEARCH, p. 1112
INPUT PROCEDURE et le verbe RELEASE, p. 1127
OUTPUT PROCEDURE et le verbe RETURN, p. 1127
Instructions dans PROCEDURE DIVISION, p. 1133
Instructions spécifiques aux fichiers IS, p. 1134
Résumé des opérations réalisées sur des fichiers IS, p. 1138
Verbe CALL, p. 1145
Instruction EXIT PROGRAM, p. 1145

FORTRAN

TRUE & FALSE, p. 1157
Opérations logiques et de relation, p. 1157
Déclarations de types de variables, p. 1158
COMPLEX, p. 1158
LOGICAL, p. 1158
CHARACTER, p. 1158
Types de variables résultant des calculs, p. 1159
IMPLICIT, IMPLICIT INTEGER, p. 1160
DIMENSION, p. 1160
COMMON, p. 1160
EQUIVALENCE, p. 1160
DATA, pp. 1162-1163
PARAMETER, p. 1163
GOTO, pp. 1163-1164
DO... WHILE..., p. 1166
IF, pp. 1166-1167
CALL, p. 1168
PAUSE, p. 1169
STOP, p. 1169
END, p. 1169
Fonctions de bibliothèque, pp. 1170-1173
FUNCTION, p. 1173
PRINT, p. 1176
WRITE, pp. 1176-1177
READ, p. 1178
Formats d'E/S, p. 1178
OPEN, pp. 1191-1192
READ et WRITE, p. 1193
BACKSPACE, p. 1193
REWIND, p. 1193
ENDFILE, p. 1193
INQUIRE, p. 1194
CLOSE, p. 1194
EXTERNAL, p. 1194
ENTRY, p. 1196
SAVE, p. 1196
BLOCKDATA, p. 1197
Améliorations spéciales, p. 1197

PASCAL

Les fonctions, p. 1221
CHR(I), ORD ('C'), PRED ('C'), SUCC ('C'), p. 1221
Les déclarations de type, p. 1227
Expressions arithmétiques et booléennes, pp. 1233-1234
L'instruction d'assignation, pp. 1235-1236
READ et WRITE, p. 1239
WRITE et WRITELN, p. 1240
FOR-TO-DO, p. 1249
WHILE-DO, p. 1251
REPEAT-UNTIL, p. 1252
IF-THEN-ELSE, p. 1254
CASE-OF, p. 1257

INSTRUCTIONS DU BASIC POUR LA COMMUNICATION DES DONNEES

TIME $, p. 1367

INSTRUCTIONS BASIC POUR LE GRAPHISME

HPLOT, p. 1414
LINE, p. 1414
LINE STEP, p. 1416
COLOR, p. 1423
CIRCLE, p. 1427
TEXT, p. 1522
HOME, p. 1522
VTAB n, p. 1522
HTAB m, p. 1522
HGR2, p. 1522
HCOLOR =n, p. 1522
HPLOT X1, Y1 TO X2, Y2, p. 1522
PEEK, pp. 1538, 1543
HIMEM LOMEM, pp. 1546-1547
Instructions Basic pour l'archivage des figures, p. 1564
POINT, p. 1564
GET, p. 1564
PUT, p. 1565
HGR, p. 1565
Instructions Basic pour la gestion de la fenêtre, p. 1587
WINDOW, pp. 1587-1589

Index des listings et des programmes d'exemples

Sous-programme de calcul de surfaces et de périmètres, p. 373
Exemple d'utilisation de l'instruction RESTORE, p. 380
Exemples d'utilisation des instructions READ et DATA, p. 381
Exemple de l'utilisation des instructions SPACE $, LSET et RSET, p. 383
Lecture des données d'état civil et impression sous forme de tableaux, p. 386
Exemple de boucle à valeurs non entières, p. 388
Exécution d'une boucle interprétée et compilée, p. 390
Exemple de trois boucles imbriquées, p. 391
Sous-programme 1000 de génération de chaînes, p. 392
Exemple d'utilisation du sous-programme 1000, p. 392
Programme employant les instructions WHILE... WEND, p. 395
Comparaison de consommation entre une voiture à gazole et une voiture à essence, pp. 415-416
Programme de gestion d'un agenda, pp. 429-430
Programme pour l'arrondi d'un nombre, p. 442
Programme de simulation d'un plan épargne-logement, p. 442
Programme de génération de nombres aléatoires, p. 444
Exemple d'utilisation des fonctions trigonométriques, p. 447
Calcul de la distance d'un navire à la côte, p. 449
Exemple d'utilisation de l'instruction ASC(A$), p. 450
Programme de l'utilisation de l'instruction CHR$, p. 452
Emploi de la fonction HEX$(N), p. 455
Détermination du nombre d'apparitions d'un caractère donné dans une chaîne, p. 457
Programme de sélection des données dans un fichier d'adresses, p. 460
Exemples de traitement de chaînes, p. 461
Programme de conversion, p. 462
Exemple d'utilisation du sous-programme 1000, p. 464
Exemple d'utilisation du sous-programme 2000, p. 466
Exemple d'application du sous-programme 3000, p. 467
Programme appelant un sous-programme, p. 476
Programme utilisant une fonction utilisateur, p. 476
Programme de traitement de chaînes, p. 482
Résolution d'équations du second degré, p. 484
Programme de contrôle et de transformation de caractères, p. 487
Programme de calcul de surfaces élémentaires, p. 491
Programme de calcul des déperditions thermiques, pp. 505-509
Remplissage d'un tableau bidimensionnel, p. 526
Remplissage d'un tableau tridimensionnel, pp. 526-527
Programme de simulation des erreurs, pp. 529-531
Exemple de calcul utilisant un tableau de chaînes, p. 534
Programme de calcul des moyennes, p. 539
Programme de calcul de l'écart type, p. 544
Exemple d'utilisation de la fonction INKEY$, p. 552
Exemple d'utilisation de la fonction INPUT$(N), p. 555
Exemple d'utilisation de l'instruction LPRINT, p. 557
Exemple d'utilisation de l'instruction LPRINT USING, p. 559
Programme d'édition sur 80 colonnes, pp. 564-566
Programme d'impression paramétrée, pp. 574-576
Exemple de type d'impression, p. 583

Programme de tracé de diagrammes à barres, pp. 592-593
Exemple de gestion de l'affichage, p. 597
Programme de saisie et de contrôle de la date, pp. 604-605
Exemple de programmation des touches de fonction, p. 608
Menu principal, pp. 625-626
Menu du stock, pp. 628-629
Exemple de préparation d'histogrammes, pp. 634-635
Exemple de lecture et d'écriture d'un fichier séquentiel, p. 641
Programme d'écriture sur un fichier séquentiel, p. 653
Programme d'écriture sur un fichier en accès direct, p. 655
Préparation, lecture et écriture d'un enregistrement, pp. 657-658
Création du fichier et initialisation du dernier enregistrement, p. 668
Routine d'ouverture du fichier, p. 669
Routine de saisie des données, p. 669
Routine de mise à jour, p. 670
Classement d'un tableau, pp. 678-679
Affectation des libellés et des longueurs de champs, p. 698
Affichage des masques de saisie, p. 702
Utilisation des masques de saisie, pp. 706-708
Sous-programme paramètre de gestion du disque, p. 712
Programme principal de test, pp. 715-716
Exemple de compilation, pp. 726-727
Exemple d'utilisation de PEEK et POKE, p. 739
Exemple de gestion des fichiers à accès séquentiel sous DOS, p. 748
Lecture et écriture de données numériques en accès séquentiel, p. 750
Extension et lecture d'un fichier à accès séquentiel, p. 751
Exemple d'utilisation de l'instruction ONERR (DOS), p. 752
Emploi de l'instruction GET, p. 753
Programme d'écriture et de lecture en accès direct, p. 754
Programme de contrôle des mots de passe, p. 792
Structure d'une procédure division, pp. 989-992
Exemple d'ouverture d'un fichier, p. 996
Exemple d'emploi de OPEN et CLOSE, p. 999
Exemple d'emploi des instructions ACCEPT et DISPLAY, pp. 1007-1008
Exemple d'application de COMPUTE, pp. 1018-1019
Exemple d'utilisation de la clause BEFORE, p. 1042
Conversion d'une date, p. 1047
Application de l'instruction IF, p. 1055
Lecture et traitement d'un fichier cartes, p. 1061
Calcul des racines d'une équation du second degré, p. 1066
Exemple d'application de AND, p. 1068
Exemple d'application de AND et de OR, p. 1072
Cycle de lecture d'un fichier séquentiel, p. 1075
Lecture et traitement d'un fichier (1), p. 1088
Lecture et traitement d'un fichier (2), pp. 1090-1091
Lecture et traitement d'un fichier (3), pp. 1093-1094
Utilisation d'un tableau à deux dimensions, pp. 1099-1100
Exemple d'impression des données d'un tableau à deux dimensions, p. 1102
Programme pour le chargement et la gestion d'un tableau, pp. 1108-1110
Description d'un tableau avec indice géré par le compilateur, p. 1111
Recherche dans un tableau au moyen d'un subscript, p. 1114
Recherche dans un tableau au moyen du SEARCH séquentiel, p. 1115
Recherche séquentielle dans un tableau, pp. 1118-1119
Recherche dichotomique dans un tableau, p. 1122
Exemple de classement et de sélection des données, p. 1126
Exemple d'utilisation de l'instruction RELEASE, p. 1128
Exemple d'utilisation de la commande SORT, pp. 1130-1131
Structure d'un programme appelant, p. 1146
Structure d'un programme appelé, p. 1146
Exemple d'application : contrôle du formatage d'une date, pp. 1148-1150
Exemple d'application : calcul du jour de la semaine, pp. 1151-1152
Exemple de programme en Fortran (comparaison Basic/Fortran), p. 1184
Sous-programme d'entrée des données (comparaison Basic/Fortran), p. 1186
Sous-programme de contrôle et de calcul (comparaison Basic/Fortran), p. 1189
Sous-programme d'impression (comparaison Basic/Fortran), p. 1191
Lecture d'un fichier et calcul de la moyenne des valeurs, p. 1196
Emploi des fonctions PRED et SUCC, p. 1263
Programme d'application des instructions READ et READLN, p. 1264
Programme d'application de l'instruction WHILE-DO, p. 1265
Procédure de facturation et de gestion des archives, pp. 1287-1291
Programme de classement (principal), p. 1308
Programme de classement (sous-programme), pp. 1313-1315
Tracé d'un segment, p. 1416
Tracé paramétré de carrés et de rectangles, pp. 1421-1422
Rotation d'un segment, pp. 1426-1427
Tracé d'un cercle, p. 1430-1431
Tracé d'une droite, pp. 1438-1439
Tracé d'une droite passant par deux points, pp. 1446-1448
Tracé d'une droite de régression, pp. 1456-1459
Tracé d'une fonction, pp. 1473-1478
Visualisation des chiffres en mode graphique, pp. 1489-1490
Visualisation des caractères en mode graphique, pp. 1496-1498
Représentation des histogrammes, pp. 1504-1510
Visualisation de diagrammes à secteurs, pp. 1527-1528
Adressage mémoire graphique, p. 1542
Lecture et visualisation du contenu de la mémoire, p. 1546
Mémorisation des images graphiques sur disque, pp. 1553-1554
Programme de calcul de surfaces, pp. 1561-1562
Création et gestion des fenêtres, pp. 1579-1581
Gestion d'une table de figures, p. 1596
Création et gestion des tables de figures, pp. 1612-1614
L'éditeur de formes, pp. 1618-1621
Le chargeur de figures, pp. 1626-1630
Graphique d'une fonction à deux variables, pp. 1640-1642
Graphique trimensionnel avec effacement des lignes cachées, pp. 1653-1657
Exemple de gestion des objets animés par le logiciel, pp. 1667-1669
Exemple d'utilisation des objets animés CBM 64, pp. 1681-1684
Gestion des objets animés sur MSX, pp. 1685-1687

Volume 1 :	pages 1 à 288
Volume 2 :	pages 289 à 576
Volume 3 :	pages 577 à 864
Volume 4 :	pages 865 à 1152
Volume 5 :	pages 1153 à 1440
Volume 6 :	pages 1441 à 1728

Index des tests

Test n° 1 (calculateurs, nombres binaires, décimaux, octaux, hexadécimaux), p. 57. Solutions, pp. 76-77

Test n° 2 (tables de vérités), p. 106. Solutions, pp. 122-123

Test n° 3 (code ASCII, méthodes de transmissions, imprimante), p. 124. Solutions, pp. 150-151

Test n° 4 (compteur de programme, UC, SE, ADM, topographie mémoire, interruptions, mémoires), pp. 152-153. Solutions, p. 183

Test n° 5 (organigrammes), pp. 206-207. Solutions, pp. 214-215

Test n° 6 (mémoires de masse), p. 240. Solutions, pp. 244-245

Test n° 7 (fichiers, tri, recherche sérielle et dichotomique), p. 279. Solutions, pp. 286-287

Test n° 8 (éléments principaux d'un SE, table des domaines, répertoire, formatage), p. 311. Solutions, p. 313

Test n° 9 (mode immédiat, champs), p. 335. Solutions, pp. 350-351

Test n° 10 (commande SAVE, lignes de programme), p. 365. Solutions, pp. 374-375

Test n° 11 (constantes, tableaux, MOD), p. 385. Solutions, pp. 398-399

Test n° 12 (définitions du type de variables, DATA), p. 410. Solutions, p. 413

Test n° 13 (bloc de programme, variable), p. 446. Solutions, p. 451

Test n° 14 (chaînes), p. 468. Solutions, p. 473

Test n° 15 (fonctions, calculs combinatoires), p. 535. Solutions, pp. 545-547

Test n° 16 (matrices), p. 560. Solutions, p. 567

Test n° 17 (imprimantes), p. 593. Solutions, p. 599

Test n° 18 (fonctions Basic), p. 637. Solutions, p. 644

Test n° 19 (fichiers de données), p. 663. Solutions, p. 671

Test n° 20 (enregistrements, compilateurs), p. 733. Solutions, p. 737

Test n° 21 (Pascal), p. 1237. Solutions, p. 1248

Test n° 22 (transmission des données), p. 1338. Solutions, p. 1342

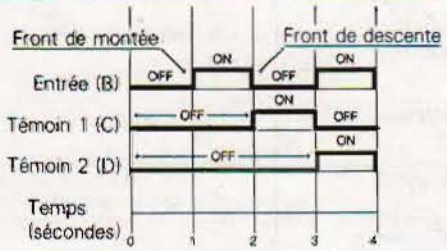
Test n° 23 (protocoles de transmission), p. 1363. Solutions, p. 1367

Index des articles

L'ordinateur peut-il penser ?, pp. 18-21
Le tisseur de nombres : histoire d'une invention, pp. 24-27
L'ordinateur à la maison, pp. 40-42
A l'école avec l'ordinateur, pp. 48-49
La révolution informatique, pp. 58-59
Un terminal dans un téléviseur, pp. 68-71
Comment consulter un ordinateur ?, pp. 84-87
La banque de l'avenir, pp. 92-95
Codes secrets contre les pirates de logiciels, pp. 102-105
Des mémoires qui n'oublient rien, pp. 134-137
Fonctionnement des mémoires mortes, pp. 144-145
Des machines qui savent lire, pp. 164-167
Une révolution dans l'imprimerie, pp. 180-182
Les circuits intégrés, pp. 194-195
Des machines qui parlent, pp. 216-219
L'homme qui inventa le jeu vidéo, pp. 234-235
L'informatique au bureau, pp. 252-253
Les banques de données spécialisées, pp. 280-282
Le dessin animé et l'ordinateur, pp. 300-303
La modélisation sur ordinateur, pp. 328-330
Wafer, chip and Co, pp. 359-362
Concerto pour clavier et microprocesseur, pp. 376-378
Quel avenir pour l'ordinateur, pp. 402-405
La "tourismatique" est née, pp. 420-423
L'homme face à la machine, pp. 431-433
Un ordinateur sous le capot, pp. 477-480
L'élève et la tortue, pp. 492-495
L'ordinateur à l'étable, pp. 516-519
La compréhension de la parole, pp. 540-543
L'intelligence artificielle, pp. 584-587
Informatique et vie privée, pp. 612-615
Quelques kilo-octets pour jouer, pp. 646-651
Communiquer à l'aide d'un micro-ordinateur, pp. 684-687
Sammie et l'automobile, pp. 719-723
Du silicium à l'ordinateur, pp. 765-768
Quel logiciel ?, pp. 777-779
La bureautique, (1), pp. 807-812, (2), pp. 827-831, (3), pp. 852-857
Le diagnostic assisté par ordinateur, (1), pp. 935-939, (2), pp. 983-986, (3), pp. 1031-1035
Le dialogue homme-machine, pp. 1078-1082
Les prévisions de l'ordinateur, pp. 1140-1144
Les jeux vidéo : pourquoi ?, pp. 1222-1226, 1269-1272, 1295-1296
Un wargame pour professionnels, pp. 1343-1348
L'ordinateur et les aiguilleurs du ciel, pp. 1511-1517
Radioscopie de l'ordinateur personnel, pp. 1548-1550
L'ordinateur et la voile, pp. 1583-1586

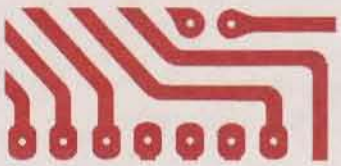
Errata

Dans la colonne de gauche "Références", sont successivement indiqués la page, la colonne (A pour gauche et B pour droite) ou bien s'il s'agit d'un tableau ou d'un encadré, enfin, en 3e position, le numéro de la ligne. Dans la colonne du milieu "Errata", on trouve les mauvais libellés ; dans la colonne de droite "Corrections" sont indiqués les bons libellés.

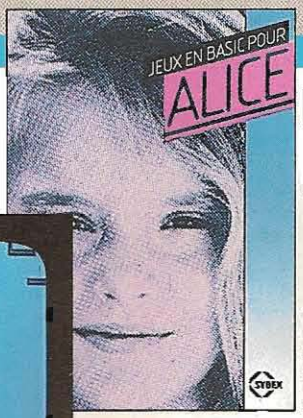
REFERENCES	ERRATA	CORRECTIONS
47/A/7	2101	2001
47/A/tableau		
57 7e question/b	En notation hexadécimale, on ne peut indiquer de nombre supérieur à 9.	En notation hexadécimale, on peut indiquer des nombres supérieurs à 9.
62/A/7	226-174 : 152 décimal =	226-174 : 52 décimal =
77/encadré/22	7/ a-vrai b-vrai c-faux d-vrai	7/ a-vrai b-vrai c-vrai d-vrai
85/B/8	Nom = Michel OU Frédéric	Prénom = Michel OU Frédéric
97/A/28	Ce boîtier comporte quatorze branches, trois	Ce boîtier contient quatorze broches, trois
108/tableau	Sortie = représentation négative inversée	Sortie = représentation négative inversée (H=Ø)
111/A/15	code ASCII (American Standard Computer	code ASCII (American Standard code for
121/B/27	masque = 1111111 1	masque = 1111111 O
185/tableau	code = A code = A code = A code = A	code = A code = A code = A code = B
208//B/tableau	BxT déperdition par rayonnent	BxT déperdition par conduction
264/tableau	Sortie du module (en direction du bloc 450 du schéma précédent).	Sortie du module (en direction du bloc 350 du schéma précédent).
287/encadré/5	250 Ko et d'1 Mo (1 Ko= 1000 octets ; 1 Mo = 1 méga-octet=1000 Ko).	250 Ko et d'1 Mo (1 Ko=1000 octets environ — en réalité 1024 ; 1 Mo : 1 méga-octet=1000 Ko).
347/tableau	Les valeurs sont : N(0)=30 N(1)=11 N(2)=15	Les valeurs sont : N(0)=5 N(1)=11 N(2)=30
350/encadré/44	1400/120=11 fois 1 minute, soit 110 minutes. C'est-à-dire qu'il faudrait presque 2 heu-	1400/120=11 fois 1 minute, soit 11 minutes. C'est-à-dire qu'il faudrait presque 2 heu-
385/encadré/24	Les solutions du test se trouvent pages 398 et 399.	Les solutions du test se trouvent pages 398 et 400.
391/listing		`160 FOR J=3 TO 7 STEP 2 'DEUXIEME BOUCLE (FERMEE LIGNE 220)`
395/listing		`80 LPRINT"VILLE=";VILLE$;`
396/tableau	WHILE F Si la valeur (VILLE) lue sur le disque n'est pas Paris, l'indicateur F est mis à	WHILE C Si la valeur (VILLE) lue sur le disque n'est pas Paris, l'indicateur C est mis à
428/tableau	MOIS=MF AND JOUR<JF? MOIS=MD AND JOUR>JD?	MOIS=MF AND JOUR<=JF? MOIS=MD AND JOUR>=JD?

429/listing		50 IF FLAG=1 GOTO 103:REM ON ABANDONNE LA LECTURE
429/listing		100 NEXT I
429/listing		103 END
429/listing		1070 REM
430/listing		5020 IF(MOIS>MF) OR (MOIS=MF AND JOUR<=JF) THEN FLAG=1:PRINT MD$:GOTO 5090
430/listing		5030 IF(MOIS<MD) OR (MOIS=MD AND JOUR>=JD)THEN PRINT MD$:GOTO 5090
432/B/22	Un manuel, une page vide ou un menu mal écrit peuvent nuire à l'esprit, davantage qu'une	Un manuel, une page vidéo ou un menu mal écrits peuvent nuire à l'esprit davantage qu'une
452/listing		225 IF 47<N AND N<58 THEN B$=" ":GOTO 240
454/A/1	CV$(A$). Convertit une chaîne de quatre	CVS(A$). Convertit une chaîne de quatre
455/A/9	Arrivé à cette instruction, le système indique, par un curseur clignotant, qu'il attend une entrée et suspend l'exécution du programme jusqu'à la frappe d'une touche.	Contrairement à l'instruction suivante INPUT$(N), cette commande n'indique pas par un curseur clignotant qu'elle attend une entrée et ne suspend pas l'exécution du programme en cours.
555/listing		220 B$=INPUT$(1) 'LECTURE D'UN SEUL CARACTERE A LA FOIS
556/listing		70 LPRINT"ESSAI N.2";A1;A2;A3
556/listing		80 LPRINT"ESSAI N.3";A1;A2;A3
559/listing		*Noter que la lettre "P" est mise pour "Paris" ou "Parisiens"* 370 END P 123.8
817/A/10	COUNT(NS..NE). Cette fonction restitue la	COUNT(NS...NE). Cette fonction restitue la
817/B/4	SUM(NS..NE). Dans la page suivante, on a	SUM(NS...NE). Dans la page suivante, on a
817/B/8	est immédiate : SUM(C2..C5) équivaut à	est immédiate : SUM(C2...C5) équivaut à
818/A/1	MAX(NS..NE), MIN(NS..NE). Elles extraient	MAX(NS...NE), MIN(NS...NE). Elles extraient
818/A/4	NPV(S, NS..NE). NPV signifie Net Present	NPV(S,NS...NE). NPV signifie Net Present
818/A/9	térêt, les valeurs NS..NE sont les chiffres que	térêt, les valeurs NS...NE sont les chiffres que
818/B/9	LOOKUP (M,NS..NE). Cette fonction recherche la valeur M dans le champ NS..NE. La	LOOKUP (M,NS...NE). Cette fonction recherche la valeur M dans le champ NS...NE. La
818/tableau	COUNT(A2..A5) SUM(C2..C5) SUM(D2..D5) SUM(E2..E5)	COUNT(A2...A5) SUM(C2...C5) SUM(D2...D5) SUM(E2...E5)
819/ 2e col/8 et 9	B14 apparait la moyenne des valeurs de la colonne B et en D14	B12 apparait la moyenne des valeurs de la colonne B et en D12
823/tableau	SUM(B2..B6) SUM(C2..C6	SUM(B2...B6) SUM(C2...C6)
824/1re col/10	fonction SUM(B2..B6), qui dé-	fonction SUM(B2...B6), qui dé-
826/tableau	SUM(E4..G4) SUM(E5..G5) SUM(E6..G6) SUM(E7..G7) SUM(E8..G8) SUM(E9..G9)	SUM(E4...G4) SUM(E5...G5) SUM(E6...G6) SUM(E7...G7) SUM(E8...G8) SUM(E9...G9)
838/ 2e col/5 et 6	lonne H) par le coût unitaire (identification, colonne D). La	lonne E) par le coût unitaire (identification, colonne C). La
878/2e schéma	S=[(NON A)ET(B)]OUX[(A)ET(NON B)] OU exclusif = OUX	S=[(NON A)ET(B)OU[(A)ET(NON B)] OU
965/A/16,43-B/2,3	ENVIRONNEMENT DIVISION	ENVIRONMENT DIVISION

968/B/11	BLOCK CONTAINS 3 RECORDS	BLOCK CONTAINS 3 RECORDS.
968/B/13 à 16	BLOCK CONTAINS 10 RECORDS BLOCK CONTAINS 1 TO 10 RECORDS BLOCK CONTAINS 100 CHARACTERS BLOCK CONTAINS 30 TO 900 CHARACTERS	BLOCK CONTAINS 10 RECORDS. BLOCK CONTAINS 1 TO 10 RECORDS. BLOCK CONTAINS 100 CHARACTERS. BLOCK CONTAINS 30 TO 900 CHARACTERS.
968/B/21	BLOCK CONTAINS 1 RECORDS	BLOCK CONTAINS 1 RECORDS.
973/B/26	01 CHAMP-1	01 CHAMP-1.
973/B/28	05 SOUS-CHAMP-2	05 SOUS-CHAMP-2.
973/B/38	77 DEPARTEMENT PIC X(2)	77 DEPARTEMENT PIC X(2).
974/A/8	IF DEPARTEMENT IS EQUAL TO''69'	IF DEPARTEMENT IS EQUAL TO '69'
974/A/13	88 MARSEILLE VALUE '13'	88 MARSEILLE VALUE '13'. .
974/A/14	88 PARIS VALUE '75'	88 PARIS VALUE '75'.
974/B/8	01 EMPLOYE	01 EMPLOYE.
974/B/10	05 PRENOM ET NOM	05 PRENOM ET NOM.
974/B/13	05 DATE DE NAISSANCE	05 DATE DE NAISSANCE.
975/tableau	Code-1 : 33 caractères	Code-1 : 23 caractères
975/A/4	longueur totale de 33 caractères. Elle peut	longueur totale de 23 caractères. Elle peut
976/A/1	01 DATA-TR	01 DATA-TR.
976/A/2	05 JOUR PIC 9(2)	05 JOUR PIC 9(2).
976/A/3	05 MOIS PIC 9(2)	05 MOIS PIC 9(2).
976/A/4	05 ANNEE PIC 9(2)	05 ANNEE PIC 9(2).
976/B/25	05 SIEGE PIC X(2) VALUE'LY'	05 SIEGE PIC X(2) VALUE'LY'.
978/A/1	A X 9 P X * $ B 0 + - . , / S V CR DB	A 9 P X * $ B 0 + - . , / S V CR DB
979/B/27	01 SERIE-ARTICLE PIC AA9(2)	01 SERIE-ARTICLE PIC AA9(2).
980/tableau/19	1234 9(4) $1234	1234 $9(4) $1234
993/A/17	OPEN OUTPUT FICHIER-SORTIE	OPEN OUTPUT FICHIER-SORTIE.
995/A/14	SELECT FICHIER-A ASSIGN TO DISC FICHIER A	SELECT FICHIER-A ASSIGN TO DISC.
996/A/7	OPEN INPUT-FICHIER-A	OPEN INPUT-FICHIER-A.
997/B/33	OPEN E-S nom-du-fichier	OPEN E-S nom-du-fichier.
1000/A/4	OPEN EXTEND nom-du-fichier	OPEN EXTEND nom-du-fichier.
1000/B/6	FICHIER-5	FICHIER-5.
1000/B/11	FICHIER-5	FICHIER-5.
1001/B/21	UPON PRINTER	UPON PRINTER.
1001/B/22	CLOSE FICHIER-A	CLOSE FICHIER-A.
1004/B/34	[LINE AT BOTTON marge inférieure]	[LINE AT BOTTOM marge inférieure]
1011/A/17-18	01 CHAMP PIC S9 (5)	01 CHAMP PIC 59 (5).
1025/A/11,14	DISPLAY «NOMBRE PAIR»	DISPLAY 'NOMBRE PAIR'
1027/A/6	01 FICHE PIC X(3).	01 FICHE
1027/A/33	MOVE SPACES TO TO BAC-A-FICHE	MOVE SPACES TO BAC-A-FICHE.
1027/A/41,44	MOVE SPACES TO BAC-A-FICHE	MOVE SPACES TO BAC-A-FICHE.
1027/B/1	MOVE BAC-A-NOM TO NOM	MOVE BAC-A-NOM TO NOM.
1027/B/4	MOVE 'ABC' TO TYPE-FICHE	MOVE 'ABC" TO TYPE-FICHE.
1027/B/7	MOVE 83 TO ANNEE	MOVE 83 TO ANNEE.
1028/A/1	01 DEPART PIC X (3) VALUE'ABC	01 DEPART PIC X (3) VALUE'ABC'.
1041/A/14	01 COULEUR PICX(20) VALUE	01 COULEUR PIC X(20) VALUE.
1042/B/15	SPACES	SPACES.
1046/A/23	01 CHAINE	01 CHAINE.
1047/B/10	INTO DATE-CONVERTIE	INTO DATE-CONVERTIE.
1054/B/18	REMAINDER RESTE	REMAINDER RESTE.
1117/tableau	MOVE 2 TO DIMENSIONE	MOVE 2 TO DIMENSION
1145/A/13	01 B	01 B.
1261/tableau	ELEVE RECORD	ELEVE=RECORD

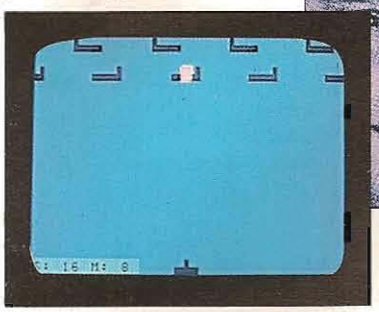


D.C.A.



Un jeu pour Alice

Tiré de «Jeux en BASIC pour ALICE» de Pierre Monsaut, Éditions SYBEX, Réf. 320, Ft 16 x 22, 96 p., 49 F.



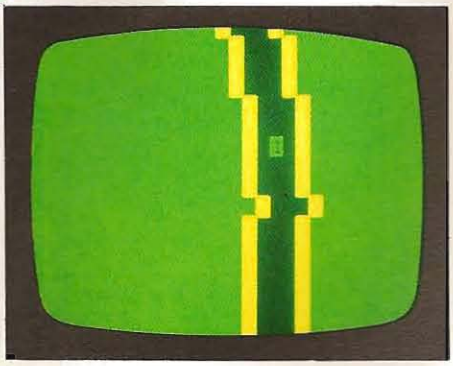
Les rôles sont maintenant inversés. Vous manœuvrez la D.C.A. et devez essayer d'abattre les avions qui passent au-dessus de vous. Pour tirer, utilisez n'importe quelle touche. Vous disposez au départ de dix obus. Si vous abattez huit avions, vous obtenez un bonus de huit points et huit obus supplémentaires.

```
10 REM D.C.A
20 CLEAR 250
30 GOTO 10000
40 A$=RIGHT$(A$,1)+LEFT$(A$,31)
50 B$=RIGHT$(B$,31)+LEFT$(B$,1)
60 PRINT@ 0,A$;:PRINT@ 64,B$;
80 R$=INKEY$
90 IF R$<>"" AND M=495 THEN M=46
3:NM=NM-1
100 IF M<>495 THEN M=M-64:PRINT@
 M,M$;:PRINT@ M+64,O$;
110 IF M<>143 THEN 200
120 IF POINT(32,5)<>0 THEN 200
130 PRINT@ 79,CHR$(191);
140 PRINT@ M,O$;
150 SOUND 1,1
160 S=S+1
170 B$=LEFT$(B$,14)+O1$+RIGHT$(B
$,13)
180 GOTO 300
200 IF M<>79 THEN 320
210 IF POINT(30,1)<>0 THEN 320
220 PRINT@ 15,CHR$(191);:S=S+1
240 PRINT@ M,O$;
250 SOUND 1,1
260 A$=LEFT$(A$,13)+O1$+RIGHT$(A
$,14)
300 IF S>1 AND INT(S/8)=S/8 THEN
 GOSUB 1000
310 M=495
320 PRINT@ 480,"S:";S;"M:";NM;
330 IF NM<1 AND M=495 THEN 400
340 IF M<32 THEN M=495
350 GOTO 40
400 IF S>R THEN R=S
410 PRINT@ 195,"SCORE :";S,"RECO
RD :";R;
420 PRINT@ 233,"UNE AUTRE ?";
430 R$=INKEY$
440 IF R$="" THEN 430
450 IF R$<>"N" THEN 30
460 CLS:END
1000 PRINT@ 173,"BONUS";
1010 A$=A1$:B$=B1$:NM=NM+8
1040 FOR I=1 TO 300:NEXT I
1060 PRINT@ 173,O1$+O1$;
1070 S=S+8
1080 RETURN
10000 S=0:A$="":B$=""
10030 FOR I=1 TO 32
10040 READ A,B
10050 A$=A$+CHR$(A)
10060 B$=B$+CHR$(B)
10070 IF I/8=INT(I/8) THEN RESTO
RE
10080 NEXT I
10090 A1$=A$:B1$=B$
10110 J$=CHR$(174)+CHR$(168)+CHR
$(172)
10120 M=495:M$=CHR$(171)
10140 NM=10:O$=CHR$(175):CLS 3
10170 PRINT@ 494,J$;:PRINT@ 0,A$
;
10190 PRINT@ 64,B$;:O1$=""
10210 FOR I=1 TO 5
10220 O1$=O1$+CHR$(175)
10230 NEXT I
10240 GOTO 40
15000 DATA 175,175,164,172,172,1
72,172,168,175,175,175,175,175,1
75,175,175
```

Dans la même collection : ATARI, ATMOS, Commodore 64, DRAGON, ORIC, Spectrum, TO 7, TRS-80, TRS-80 couleur, TRS-80 MC-10, VIC 20 et ZX 81.

SYBEX
6-8, impasse du Curé
75018 PARIS
Tél. 203-95-95

Dans le prochain numéro : **Apollo II,** un jeu pour Vic 20.



GRAND-PRIX

Un jeu pour ALICE



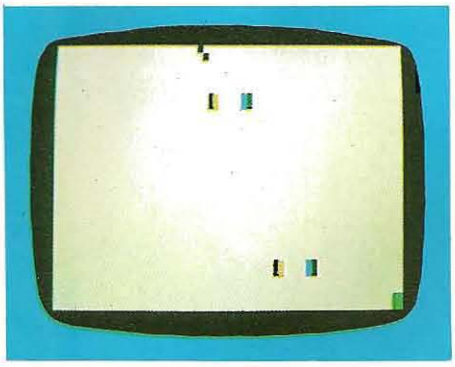
Au volant de votre formule 1, essayez de parcourir la plus grande distance possible. Votre voiture dispose de deux vitesses que vous pouvez sélectionner à l'aide des touche <1> et <2>. La direction est commandée par les touche <,> et <.>. En seconde vitesse, votre voiture roule deux fois plus vite. Mais gare à l'accident.



```
5 REM  ***************
10 REM * GRAND-PRIX *
15 REM ***************
20 CLS
30 N$=CHR$(128)
35 REM R$=ROUTE
40 R$=CHR$(159)+CHR$(128)+CHR$(1
28)+CHR$(128)+CHR$(159)
45 REM R=POSITION ROUTE
50 R=13
55 REM T=KM PARCOURUS/TEMPS
60 T=1
65 REM V$=CARACTERE VOITURE
70 V$="I"
75 REM V=POSITION VOITURE
80 V=207
90 V1=V
95 REM DEBUT ROUTE DROITE
100 FOR I=0 TO 15
110 PRINT@ R,R$;
120 R=R+32
130 NEXT I
140 R=R-32
145 REM AFFICHAGE VOITURE
150 PRINT@ V,V$;
154 REM
155 REM BOUCLE PRINCIPALE
156 REM
160 D$=INKEY$
165 REM CHANGEMENT DE VITESSE
170 IF D$="1" THEN T=1:GOTO 160
180 IF D$="2" THEN T=2:GOTO 160
185 REM DIRECTION
200 V=V+(D$=",")-(D$=".")
210 IF D$<>"" THEN 160
215 REM NOUVELLE POS. VOITURE
220 P=V+32
225 REM CALCUL COORDONNEES X,Y
230 Y=INT(P/32)*2
240 X=(P-16*Y)*2
245 REM ACCIDENT?
250 IF POINT(X,Y)<>0 THEN 400
255 REM AFFICHAGE ROUTE
260 R=R+(RND(2)=1)-(RND(2)=1)
270 IF R<480 THEN R=480
280 IF R>506 THEN R=506
290 PRINT@ 511," ";
295 REM AFFICHAGE VOITURE
300 PRINT@V1-32,N$;
310 PRINT@ R,R$;
320 PRINT@ V,V$;
345 REM COMPTE KM
350 K=K+T
355 REM DELAI
360 DL=(2-T)*50
370 FOR I=1 TO DL:NEXT I
380 V1=V
390 GOTO 160
394 REM
395 REM ACCIDENT
396 REM
400 PRINT@ P,CHR$(191);
410 PRINT@ V1,N$;
420 FOR I=1 TO 4
430 FOR J=1 TO 20
440 SOUND J*10,1
450 NEXT J
460 NEXT I
465 REM AFFICHAGE SCORE
470 PRINT@ 166,"KMS PARCOURUS :"
;K;
480 D$=INKEY$
490 PRINT@ 230,"UNE AUTRE ?";
500 D$=INKEY$
510 IF D$="" THEN 500
520 IF D$<>"N" THEN RUN
530 CLS
540 END
```

Tiré de «Jeux en BASIC sur ALICE»
de Pierre Monsaut, Editions SYBEX
Réf. 320 Ft 16 x 22, 96 p., 49 F.

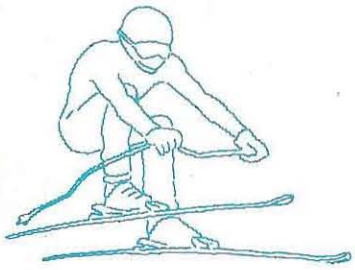
SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



SLALOM

Un jeu pour ALICE



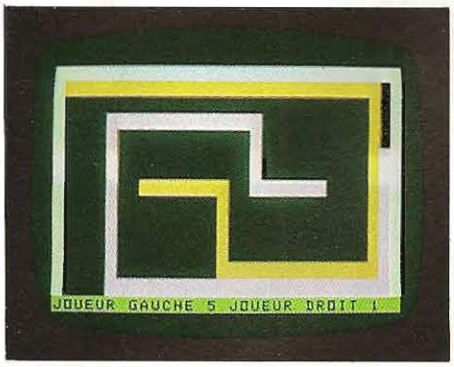


Partez aux sports d'hiver sans risque de vous casser une jambe ! Lancez-vous du haut de la piste et essayez de passer le plus grand nombre possible de portes sans heurter les piquets. Frappez n'importe quelle touche pour changer de direction.

```
10 REM SLALOM
15 REM TABLEAU DES POSITIONS
16 REM        DU SKIEUR
20 DIM S$(1)
30 FOR I=1 TO 32
40 E$=E$+CHR$(207)
50 NEXT I
55 REM SKIEUR ALLANT A GAUCHE:
60 S$(0)=CHR$(201)
65 REM SKIEUR ALLANT A DROITE:
70 S$(1)=CHR$(198)
75 REM ECRAN BLANC
80 CLS 5
85 REM DIRECTION INITIALE:
86 REM              GAUCHE
90 D=-1
95 REM POSITION INITIALE
96 REM           DU SKIEUR
100 J=16
105 REM DESSIN DES PORTES
110 P$=CHR$(181)+CHR$(207)+CHR$(
207)+CHR$(170)
115 REM
116 REM BOUCLE PRINCIPALE
117 REM
120 FOR K=1 TO 300
125 REM CALCUL DES COORDONNEES
126 REM     X ET Y DU SKIEUR
130 Y=INT(J/32)*2
140 X=(J-16*Y)*2
145 REM SKIEUR AU NIVEAU D'UNE
146 REM              PORTE?
150 IF K>=16 AND (K-5)/10=INT((K
-5)/10) THEN GOSUB 350
155 REM AFFICHAGE D'UNE PORTE?
160 IF K<284 AND K/10=INT(K/10)
THEN GOSUB 400
165 REM MOUVEMENT DU SKIEUR
170 IF INKEY$<>"" THEN D=-D
180 J=J+D
190 IF J<2 THEN J=2
200 IF J>29 THEN J=29
210 PRINT@ 511,E$;
220 PRINT@ J,S$(D/2+0.5);
230 NEXT K
235 REM FIN DE LA DESCENTE
240 PRINT@ 164,"PORTE(S) RATEE(S
) :";T;
250 PRINT@ 230,"UNE AUTRE DESCEN
TE ?";
260 D$=INKEY$
270 IF D$="" THEN 260
280 IF D$<>"N" THEN RUN
290 CLS
300 END
344 REM
345 REM PORTE RATEE?
346 REM
350 IF POINT(X-2,Y)<>0 OR POINT(
X+4,Y)<>3 THEN IF POINT(X-4,Y)<>
0 OR POINT(X+2,Y)<>3 THEN T=T+1:
SOUND 1,1
360 RETURN
394 REM
395 REM AFFICHAGE D'UNE PORTE
396 REM
400 P1=RND(3)-2
410 P=P-6*P1
420 IF P<482 THEN P=488
430 IF P>506 THEN P=500
440 PRINT@ P,P$;
450 RETURN
```

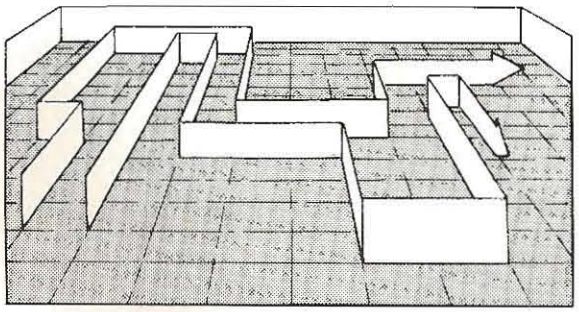
Tiré de «Jeux en BASIC sur ALICE» de Pierre Monsaut, Editions SYBEX, Réf. 320, Ft 16 x 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



TRACE

Un jeu pour ALICE



Deux joueurs s'affrontent pour se partager l'espace vital. Chacun doit s'efforcer, tout en se déplaçant, de ne jamais recouper sa trace ou celle de son adversaire, et de ne pas sortir du rectangle dessiné sur l'écran. Les commandes à utiliser sont :

Joueur de droite : <P>, <L>, <M> et <.>
Joueur de gauche : <Z>, <Q>, <S> et <W>

```
10 REM TRACE
20 CLS 0
30 C$=CHR$(207)
40 J1$=CHR$(239)
50 J2$=CHR$(255)
60 P1=248
70 P2=232
80 D1=-1
90 D2=1
120 FOR I=0 TO 31
130 PRINT@ I,C$;
140 PRINT@ I+448,C$;
150 NEXT I
160 FOR I=1 TO 13
170 PRINT@ I*32,C$;
180 PRINT@ I*32+31,C$;
190 NEXT I
200 PRINT "JOUEUR GAUCHE";J2,"JO
UEUR DROIT";J1
210 PRINT@ P1,J1$;
220 PRINT@ P2,J2$;
230 D$=INKEY$
240 C1=(D$="L")-(D$="M")+32*((D$
="P")-(D$="."))
250 C2=(D$="Q")-(D$="S")+32*((D$
="Z")-(D$="W"))
260 IF C1<>0 THEN D1=C1
270 IF C2<>0 THEN D2=C2
280 P1=P1+D1
290 Y=INT(P1/32)*2
300 X=(P1-16*Y)*2
310 IF POINT(X,Y)<>0 THEN 1000
320 PRINT@ P1,J1$;
330 P2=P2+D2
340 Y=INT(P2/32)*2
350 X=(P2-16*Y)*2
360 IF POINT(X,Y)<>0 THEN 2000
370 PRINT@ P2,J2$;
380 SOUND 1,1
390 GOTO 230
1000 J2=J2+1
1010 GOSUB 5000
1020 IF J2=10 THEN 3000
1030 D$=INKEY$
1040 GOTO 10
2000 J1=J1+1
2010 GOSUB 5000
2020 IF J1=10 THEN 4000
2030 D$=INKEY$
2040 GOTO 10
3000 CLS
3010 PRINT@ 164,J2$;"LE JOUEUR G
AUCHE GAGNE";J2$
3020 PRINT@ 202,J2;"A";J1
3030 GOTO 4500
4000 CLS
4010 PRINT@ 164,J1$;"LE JOUEUR D
ROIT GAGNE";J1$
4020 PRINT@ 202,J1;"A";J2
4500 R$=INKEY$
4510 PRINT@ 266,"UNE AUTRE ?"
4520 R$=INKEY$
4530 IF R$="" THEN 4510
4540 IF R$<>"N" THEN RUN
4550 END
5000 FOR I=5 TO 255 STEP 5
5010 SOUND I,1
5020 NEXT I
5030 RETURN
10000 SOUND 1,1
10010 RETURN
```

Tiré de «Jeux en BASIC pour ALICE» de Pierre Monsaut, Editions SYBEX, Réf. 320, Ft 16 × 22, 96 p., 49 F



SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



CHIFFRES

Un jeu pour ATARI

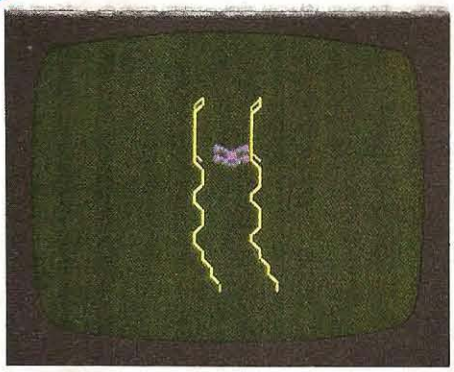
Utilisez les touches <Z> et <X> pour modifier votre croqueur de chiffres. A droite de l'écran, des nombres apparaissent les uns après les autres et si vous ne parvenez pas à les dévorer, vous aurez perdu. Pour dévorer un chiffre, utilisez la touche <.>, vous ne pouvez dévorer qu'un chiffre identique à celui de votre croqueur de chiffres. Si vous dévorez un signe «*», vous obtenez un bonus.

```
0 GOTO 135:REM ** CHIFFRES DE P.BUNN **
1 FOR I=50 TO 75 STEP 2:SOUND 0,I,10,15:
NEXT I:I=0:Y=M+48:IF M=10 THEN Y=10
2 Q$=CHR$(Y):IF A$(1,1)=Q$ THEN I=1:S=S+
10
3 IF A$(2,2)=Q$ AND I=0 THEN A$(2,2)=A$(
1,1):I=1:S=S+20
4 IF A$(3,3)=Q$ AND I=0 THEN A$(3,3)=A$(
2,2):A$(2,2)=A$(1,1):I=1:S=S+30
5 IF A$(4,4)=Q$ AND I=0 THEN A$(4,4)=A$(
3,3):A$(3,3)=A$(2,2):A$(2,2)=A$(1,1):I=1
:S=S+40
6 IF A$(5,5)=Q$ AND I=0 THEN A$(5,5)=A$(
4,4):A$(4,4)=A$(3,3):A$(3,3)=A$(2,2):I=1
:S=S+50
7 IF A$(6,6)=Q$ AND I=0 THEN A$(6,6)=A$(
5,5):A$(5,5)=A$(4,4):A$(4,4)=A$(3,3):A$(
3,3)=A$(2,2):I=1:S=S+60
8 IF I=1 THEN A$(1,1)=" ":N=N+1
10 IF I=1 AND Q$=CHR$(10) THEN FOR I=255
 TO 20 STEP -18:SOUND 0,I,10,15:NEXT I:S
=S+(INT(RND(0)*7+3)*100)
11 POSITION 1,0:? #6;SC$;S:? #6;" RECORD
:";REC
12 SOUND 0,0,0,0:RETURN
20 FOR I=14 TO 0 STEP -2:SOUND 0,M+15%30
,10,I:NEXT I:RETURN
135 DIM A$(6),Q$(1),SC$(6):GOSUB 2000
140 GRAPHICS 2:SETCOLOR 4,2,2:SETCOLOR 2
,2,4
150 A$="      ":A$(6,6)=CHR$(RND(0)*9+48)
160 S=0:N=5:M=0:U=18:GOSUB 11
165 Y=M+48:IF M=10 THEN Y=10
170 POSITION 5,5:? #6;CHR$(Y);":";A$
180 I=PEEK(764):I2=PEEK(53775)
190 IF I=22 AND I2=251 THEN M=M+1:GOSUB
20:IF M>10 THEN M=0
200 IF I=23 AND I2=251 THEN M=M-1:GOSUB
20:IF M<0 THEN M=10
210 IF I=34 AND I2=251 THEN GOSUB 1
215 I=0
220 TRAP 1000:O=O+1:IF O/U=INT(O/U) THEN
 A$(N,N)=CHR$(RND(0)*10+48):I=1
222 IF I=1 THEN IF A$(N,N)=CHR$(58) THEN
 A$(N,N)=CHR$(10)
224 IF I=1 THEN N=N-1:I=0
225 IF INT(RND(0)*20)=3 THEN U=U-1:IF U<
8 THEN U=8
230 TRAP 40000:GOTO 165
1000 ? #6:? #6;"..perdu..":FOR I=90 TO 2
20 STEP 11:FOR H=80 TO 90 STEP 2:SOUND 0
,H,10,15:SOUND 1,I,10,15
1010 SETCOLOR 4,H,4
1020 POKE 709,INT(RND(0)*15)*16+8
1030 NEXT H:NEXT I:SETCOLOR 4,2,4:POKE 7
64,255
1040 SOUND 0,0,0,0:SOUND 1,0,0,0:IF S>RE
C THEN REC=S:POSITION 0,2:? #6;" record!
!!"
1050 ? "UNE AUTRE ";:TRAP 1050:INPUT A$
1055 IF A$(1,1)="N" THEN GRAPHICS 0:? "A
U REVOIR.":? :? :END
1060 IF A$(1,1)="O" THEN 140
1070 GOTO 1050
2000 FOR P=1 TO 6
2010 READ N
2020 SC$(P,P)=CHR$(N)
2030 NEXT P
2040 DATA 243,227,239,242,229,186
2050 RETURN
```



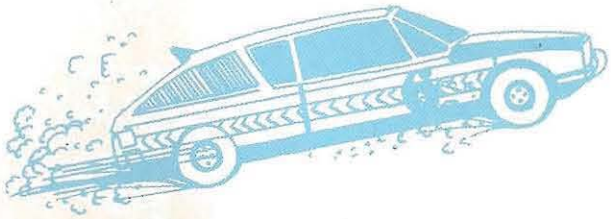
Tiré de «Jeux en BASIC sur ATARI» de Paul Bunn, Éditions SYBEX, Réf. 282, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé 75881 PARIS CEDEX 18 Tél. 203.95.95



GRAND PRIX 2

Un jeu pour ATARI





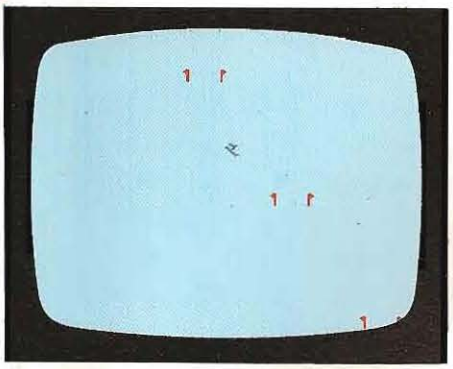


Pour conduire votre bolide, utilisez le manche à balai. Si vous réussissez à atteindre l'arrivée, vous aurez un bonus, et la valeur de la distance parcourue sera doublée.

```
10 REM ** GRAND PRIX 2 - P.BUNN **
15 PI=5:GOSUB 3000:ACCIDENT=1000
20 X2=14:X=116:S=PEEK(106)-8:Q=S*256:FOR
 N=Q+512 TO Q+640:POKE N,0:NEXT N:POKE 5
4279,S
30 POKE 559,46:POKE 53248,X:POKE 704,216
:POKE 704,70:POKE 53256,1
40 FOR N=Q+552 TO Q+581:READ A:POKE N,A:
NEXT N
50 DATA 129,195,165,24,24,153,219,165,36
,24
60 GRAPHICS 0:SETCOLOR 2,0,0:POKE 53277,
3:POKE 559,46
65 POKE 752,1:POKE 53278,A
66 FOR P=0 TO 23:POKE 201,X2:? ,S2$:NEXT
 P
70 S=STICK(0):X=X+(S=7)*2:X=X-(S=11)*2:P
OKE 53248,X
72 SOUND 0,40,10,15:SC=SC+PI:SOUND 0,0,0
,0
75 IF Q THEN Q=0:Z=2:GOTO 110
80 A=PEEK(53770)
90 IF A>85 AND A<170 THEN Z=2
100 IF A>170 AND X2<15 THEN Z=3:Q=1
105 IF A<85 AND X2>2 THEN X2=X2-1:Z=1:Q=
1
110 POKE 201,X2
115 IF A=192 AND  NOT I THEN ? ,F$:I=1:G
OTO 145
120 IF Z=1 THEN ? ,S1$
130 IF Z=2 THEN ? ,S2$
140 IF Z=3 THEN ? ,S3$:X2=X2+1
145 Y=PEEK(53252):IF Y<>0 AND I THEN GOT
O 2000
150 IF Y<>0 THEN GOTO ACCIDENT
160 GOTO 70
1000 REM ** ACCIDENT!!!! **
1010 N=INT(RND(0)*10):POKE Q+552+N,PEEK(
53770):SOUND 0,RND(0)*20+20,80,15:POKE 7
04,PEEK(53770)
1020 IF PEEK(53770)<240 THEN 1010
1030 POKE 53248,0:? :? "VOTRE SCORE:";SC
:? :? :? "TAPEZ UNE TOUCHE.":POKE 764,25
5
1035 SOUND 0,0,0,0
1040 IF PEEK(764)=255 THEN 1040
1050 RUN
2000 FOR Y=0 TO 255:POKE 710,Y:NEXT Y
2005 ? CHR$(125);"      SCORE :";SC:POKE
710,0
2010 ? "     BONUS  :1000":B=1000
2020 FOR G=1 TO 1000 STEP 10:B=B-10:SC=S
C+10:POSITION 14,0:? SC:POSITION 14,1:?
B;"  ":SOUND 0,G/4,10,10:NEXT G
2030 SOUND 0,0,0,0
2040 ? :PI=PI*2:? "1 KM VAUT ";PI;" POIN
TS"
2045 RESTORE :I=0
2050 GOTO 20
3000 DIM F$(8),S1$(8),S2$(8),S3$(8)
3010 RESTORE 3000:FOR P=1 TO 8
3020 READ A,B,C,D
3030 F$(P,P)=CHR$(A)
3040 S1$(P,P)=CHR$(B)
3050 S2$(P,P)=CHR$(C)
3060 S3$(P,P)=CHR$(D)
3070 NEXT P
3080 RESTORE :RETURN
3090 DATA 160,6,22,7,198,32,32,32
3100 DATA 201,32,32,32,206,32,32,32
3110 DATA 201,32,32,32,211,32,32,32
3120 DATA 200,32,2,32,160,6,32,7
```

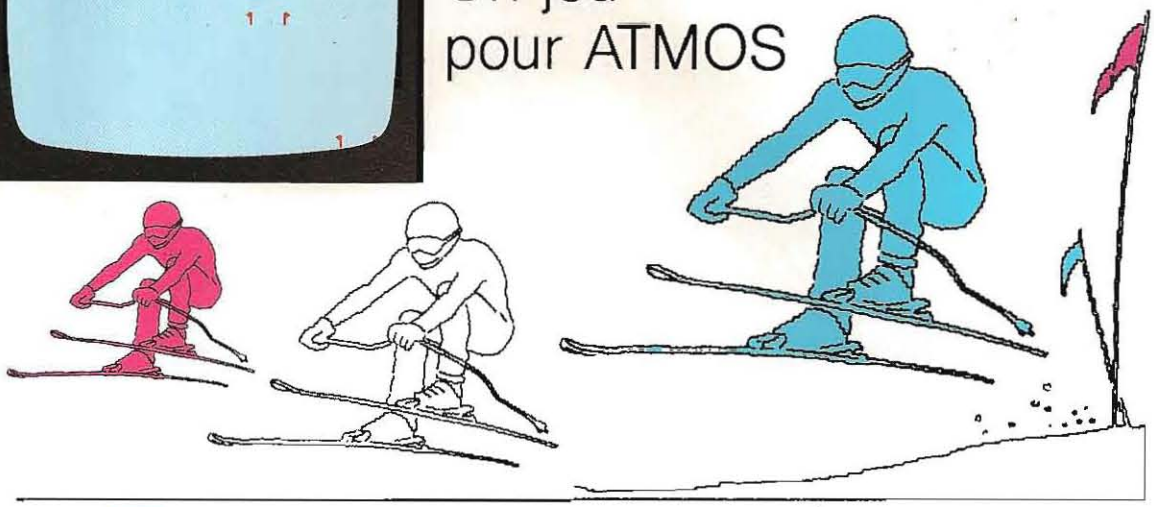
Tiré de «Jeux en BASIC sur ATARI» de Paul Bunn, Editions SYBEX, Réf. 282, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé 75881 PARIS CEDEX 18 Tél. : 203.95.95



SLALOM

Un jeu pour ATMOS

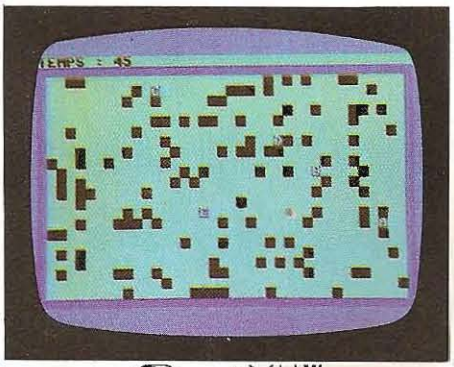


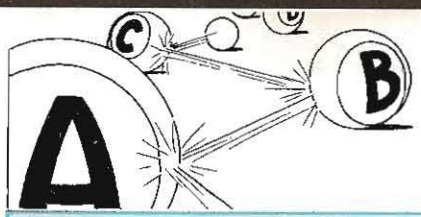
Partez aux sports d'hiver sans risque de vous casser une jambe ! Lancez-vous du haut de la piste et essayez de passer le plus grand nombre possible de portes sans heurter les piquets. Frappez n'importe quelle touche pour changer de direction.

```
 10 REM **********
 20 REM * SLALOM *
 30 REM **********
 50 GOSUB 460
 60 FOR K=1 TO 300
 90 IF K>=16 AND (K-5)/10=INT((K-5)/10)T
HEN GOSUB 360
 100 IF K<284 AND K/10=INT(K/10) THEN GO
SUB 380
 110 IF KEY$<>"" THEN DX=-DX
 120 S1=SX
 130 SX=SX+DX
 140 IF SX<2 THEN SX=2:DX=-DX
 150 IF SX>37 THEN SX=37:DX=-DX
 160 PLOT S1,SY,N$
 170 PRINT@ 38,26;N$
 180 PLOT SX,SY,S$(DX/2+0.5)
 190 NEXT K
 200 PLOT 8,15,"PORTE(S) RATEE(S) :"+STR
$(T)
 210 PLOT 8,18,"UNE AUTRE DESCENTE ?"
 220 REPEAT
 230 D$=KEY$
 240 UNTIL D$=""
 250 REPEAT
 260 D$=KEY$
 270 UNTIL D$<>""
 280 IF D$<>"N" THEN PRINT CHR$(17):RUN
 290 INK 0
 300 PRINT CHR$(17)
 310 CLS
 320 END
 360 IF SCRN(SX-1,SY)=93 AND SCRN(SX+2,S
Y)=94 THEN 375
 365 IF SCRN(SX-2,SY)=93 AND SCRN(SX+1,S
Y)=94 THEN 375
 370 T=T+1
 372 ZAP
 375 RETURN
 380 P=P+(INT(RND(1)*3)-1)*8
 390 IF P<2 THEN P=10
 400 IF P>34 THEN P=26
 410 PLOT P,26,P$
 420 PLOT 1,26,CHR$(1)
 430 RETURN
 460 CLS
 470 PAPER 7
 480 INK 4
 490 PRINT CHR$(17)
 500 FOR I=0 TO 31
 510 READ A
 520 POKE 46808+I,A
 530 NEXT I
 540 S$(0)=CHR$(91)
 550 S$(1)=CHR$(92)
 560 N$=CHR$(32)
 570 P$=CHR$(93)+N$+N$+CHR$(94)
 580 DX=-1
 590 SY=11
 600 SX=19
 610 P=18
 620 RETURN
1000 DATA 4,56,17,26,4,8,16,32
1010 DATA 8,7,34,22,8,4,2,1
1020 DATA 12,28,12,4,4,4,4,4
1030 DATA 6,7,6,4,4,4,4,4
```

Tiré de «Jeux en BASIC sur ATMOS» de Pierre Monsaut, Editions SYBEX, Réf. 346, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95





ALPHABET

Un jeu pour COMMODORE 64

Voici un jeu d'adresse assez difficile. Vous tenterez de marquer le plus grand nombre de points possible en effaçant les lettres affichées par l'ordinateur. Les touches à utiliser sont <W> (haut), <Z> (bas) <A> (gauche) et <S> (droite). La barre d'espacement vous permet de vous arrêter. *Attention :* vous devez effacer les lettres dans l'ordre alphabétique en évitant les obstacles placés au hasard sur l'écran, et ceci en un temps limité. (Le temps qui vous reste est affiché sur la première ligne.) Lorsque toutes les lettres sont effacées, le jeu reprend avec une lettre supplémentaire.

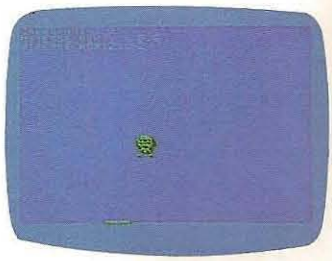
```
5 REM  ************
10 REM * ALPHABET *
15 REM ************
20 GOSUB 1000
100 FOR I=1 TO X
110 GET X$
120 D=(X$="A")-(X$="S")+40*((X$="W")-(X$
="Z"))
130 IF D<>0 THEN D0=D
140 IF X$=" " THEN D0=0
150 T=T-0.1
160 T$=STR$(INT(T+1))
170 PRINT H$;
180 PRINT "TEMPS :";T$;" ";
190 IF T<0 THEN 3000
200 J=J+D0
210 C=PEEK(J)
220 IF C=I+128 THEN S=S+I:GOTO 290
230 IF C<>32 THEN J=J1
240 POKE J1,CR
250 POKE J,CJ
260 POKE J+M,JC
270 J1=J
280 GOTO 110
290 POKE J1,CR
300 POKE J,CJ
310 POKE J+M,JC
320 J1=J
330 NEXT I
340 GOTO 20
1000 X=X+1:CJ=81
1020 JC=10:X$=""
1040 D=0:D0=D
1060 T=50:T$=""
1080 H$=CHR$(19)
1090 J=0:J1=J
1110 C=0:CR=32
1130 M=54272
1140 CB=160
2000 PRINT CHR$(147);
2010 POKE 53280,14
2020 POKE 53281,1
2030 FOR I=0 TO 39
2040 POKE 1064+I,CB
2050 POKE 1064+I+M,6
2060 POKE 1984+I,CB
2070 POKE 1984+I+M,6
2080 NEXT I
2090 FOR I=1 TO 22
2100 POKE 1064+I*40,CB
2110 POKE 1064+I*40+M,6
2120 POKE 1103+I*40,CB
2130 POKE 1103+I*40+M,6
2140 NEXT I
2150 FOR I=1 TO 120
2160 GOSUB 5000
2170 POKE P,CB
2180 POKE P+M,0
2190 NEXT I
2200 FOR I=1 TO X
2210 GOSUB 5000
2220 POKE P,I+128
2230 POKE P+M,6
2240 NEXT I
2250 GOSUB 5000
2260 J=P
2270 POKE J,CJ
2280 POKE J+M,JC
2290 J1=J
2300 PRINT CHR$(144);
2310 RETURN
3000 IF S>RE THEN RE=S
3010 PRINT CHR$(147):PRINT:PRINT
3040 PRINT TAB(13)"TEMPS ECOULE"
3050 PRINT:PRINT:PRINT
3080 PRINT TAB(13)"SCORE :";S
3090 S=0:PRINT:PRINT:PRINT
3130 PRINT TAB(13)"RECORD :";RE
3140 PRINT:PRINT:PRINT:PRINT:GET X$
3180 PRINT TAB(13)"UNE AUTRE ?"
3190 GET X$
3200 IF X$="" THEN 3190
3210 IF X$<>"N" THEN 20
3220 END
5000 P=INT(RND(TI)*960)+1064
5010 IF PEEK(P)<>32 THEN 5000
5020 RETURN
```

Tiré de «Jeux en BASIC sur COMMODORE 64» de Pierre Monsaut, Editions SYBEX Réf. 317, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé 75881 PARIS CEDEX 18 Tél. : 203.95.95

ATTERRISSAGE

Un jeu pour Commodore 64





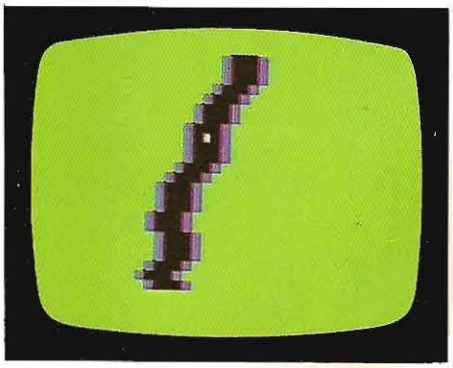
Tiré de « Jeux en BASIC sur Commodore 64 » de Pierre Monsaut, Éditions SYBEX, Réf. 317, Ft 16 x 22, 96 p., 49 F.

Après un long voyage en apesanteur, poser une navette spatiale en douceur n'est pas chose aisée ; mais grâce à votre ordinateur, vous allez être en mesure de vous entraîner sans danger. Vous devez poser votre navette sur l'aire prévue à cette effet. Vous pouvez vous diriger vers la droite et vers la gauche à l'aide des touches de contrôle du curseur ou freiner votre descente avec la barre d'espacement. Pour réussir un atterrissage, les vitesses verticales et horizontales doivent être inférieures ou égales à 1.

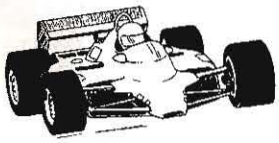
```
5 REM  ******************
10 REM * ATTERRISSAGE *
15 REM ******************
20 GOSUB 1000
100 GET X$:IF FU=0 THEN 150
110 IF X$<>"" THEN FU=FU-1
120 IF X$=" " THEN UU=UU-1
130 IF X$=G$ THEN UH=UH-1
140 IF X$=D$ THEN UH=UH+1
150 UU=UU+0.1:U=U+UU:H=H+UH
190 IF H>255 THEN H=0:MS=M1
200 IF H>90 AND MS=M1 THEN 4540
210 IF H<0 AND MS=M1 THEN H=255:MS=M0
220 IF H<0 AND MS=M0 THEN 4540
240 PRINT HO$;
250 PRINT "CARBURANT :";STR$(FU);"     "
255 IF V<0 THEN 300
260 POKE D+16,MS:POKE D,H
270 POKE D+1,INT(V)
275 IF V>V1-3 THEN 3000
300 PRINT "VITESSE VERT. :";STR$(INT(10*
UU)/10);"     "
310 PRINT "VITESSE HORIZ.:";STR$(UH);"
320 GOTO 100
1000 D=53248:RESTORE
1010 FOR I=0 TO 190
1020 READ A:POKE 832+I,A
1035 NEXT I
1040 M1=5:M0=0:POKE D+39,7:POKE D+40,7
1090 POKE D+41,0:POKE 2040,13
1110 POKE 2041,14:POKE 2042,15
1130 V1=230:V0=0:FU=90:G$=CHR$(29)
1170 D$=CHR$(17):HO$=CHR$(19):M=54272
2000 PRINT CHR$(147);:POKE D+21,0
2020 H=INT(RND(TI)*191)+65
2030 AH=INT(RND(TI)*191)+65
2050 FU=FU+10:MS=M0:UH=0:UU=0:U=0
2100 POKE D+2,AH:POKE D+3,V1
2120 POKE D,H:POKE D+1,V:POKE D+21,3
2150 RETURN
3000 IF ABS(H-AH)>4 THEN 4000
3005 IF UU>1 THEN 4000
3010 IF ABS(UH)>1 THEN 4000
3020 FOR I=1 TO 4000:NEXT I
3030 SC=SC+1:GOSUB 2000
3050 GOTO 100
4000 POKE D+5,V+5:POKE D+4,H
4020 POKE D+21,6
4500 PRINT:PRINT:PRINT:PRINT:PRINT
4530 PRINT TAB(6)"VOTRE NAVETTE S'EST EC
RASEE"
4540 IF SC>RE THEN RE=SC
4545 FOR I=1 TO 2000:NEXT I
4560 PRINT:PRINT:PRINT
4580 PRINT TAB(13)"SCORE :";SC
4590 SC=0
5000 PRINT:PRINT:PRINT
5030 PRINT TAB(13)"RECORD :";RE
5040 IF RE<SC THEN RE=SC
5050 SC=0:PRINT:PRINT:PRINT:GET X$
5100 PRINT TAB(13)"UNE AUTRE ?"
5110 GET X$
5120 IF X$="" THEN 5110
5130 IF X$<>"N" THEN POKE D+21,0:GOTO 20
5140 END
10000 DATA 0,255,0,1,255,128,3,255,192
10010 DATA 7,255,224,12,195,48,12,195,48
10020 DATA 15,255,240,12,102,48,12,102,4
8
10030 DATA 15,255,240,7,255,224,3,129,19
2
10040 DATA 1,129,128,0,255,0,1,255,128
10050 DATA 1,24,128,2,60,64,2,36,64
10060 DATA 4,0,32,4,0,32,14,0,112,0
10100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10120 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10130 DATA 0,0,0,0,0,0,0,0,0,255,255,255
10140 DATA 255,255,255,255,255,255,0
10200 DATA 0,0,0,0,0,0,45,0,27,3,0,0,8,0,1
27,0
10210 DATA 0,5,0,0,236,0,0,67,0,2,64,240
10220 DATA 7,32,112,7,32,56,15,145,56
10230 DATA 31,23,120,31,25,124,95,97,254
10240 DATA 127,255,254,255,255,255,255
10250 DATA 255,255,0,0,0,0,0,0,0,0,0
```

Dans la même collection : ATARI, ALICE, ATMOS, DRAGON, ORIC, Spectrum, TO 7, TRS-80, TRS-80 couleur, TRS-80 MC-10, VIC 20 et ZX 81.

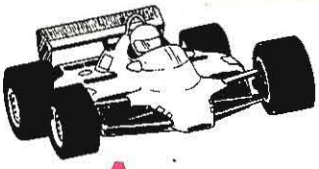
SYBEX
6-8, impasse du Curé
75018 PARIS
Tél. 203-95-95

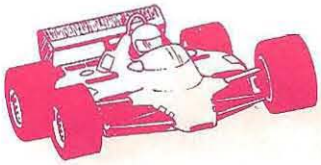


GRAND PRIX



Un jeu pour COMMODORE 64



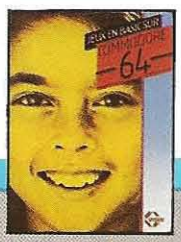




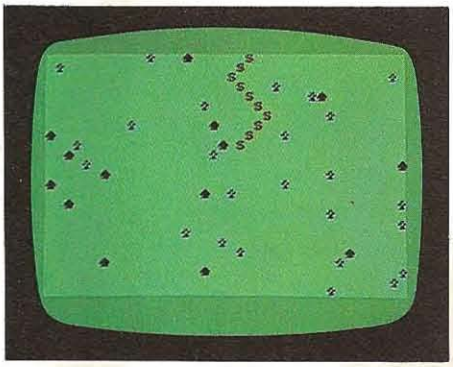
Au volant de votre formule 1, essayez de parcourir la plus grande distance possible. Votre voiture dispose de deux vitesses que vous pouvez sélectionner à l'aide des touches <1> et <2>. La direction est commandée par les touches de contrôle du curseur. En seconde vitesse, votre voiture roule deux fois plus vite. Mais gare à l'accident !

```
5 REM  xxxxxxxxxxxxxx
10 REM x GRAND-PRIX x
15 REM xxxxxxxxxxxxxx
20 GOSUB 1000
100 GET X$
110 IF X$=U1$ THEN T=1
120 IF X$=U2$ THEN T=2
130 U=U+(X$=G$)-(X$=D$)
140 P=PEEK(U+M+40)
150 IF INT(P/16)<>P/16 THEN 3000
160 R=R+(RND(TI)<0.5)-(RND(TI)<0.5)
170 IF R<UN THEN R=UN
180 IF R>UM THEN R=UM
190 PRINT TAB(R);R$
200 POKE U1-40,CR
210 POKE U1-40+M,0
220 POKE U,CU
230 POKE U+M,UC
240 K=K+T
250 DL=(2-T)*50
260 FOR I=1 TO DL
270 NEXT I
280 U1=U
290 GOTO 100
1000 PRINT CHR$(147)
1010 POKE 53280,5
1020 POKE 53281,5
1030 R=18
1040 U=1364
1050 U1=U
1060 M=54272
1070 T=1
1080 S=0
1090 U1$="1"
1100 U2$="2"
1110 D$=CHR$(29)
1120 G$=CHR$(17)
1130 CR=160
1140 CU=90
1150 UC=1
1160 R$=CHR$(18)+CHR$(31)+" "+CHR$(144),
1170 R$=R$+"    "+CHR$(31)+" "
1180 UN=1
1190 UM=34
2000 FOR I=1 TO 25
2010 PRINT TAB(R);R$
2020 NEXT I
2030 POKE U,CU
2040 POKE U+M,UC
2050 RETURN
3000 POKE U1,CR
3010 POKE U1+M,0
3020 POKE U+40,160
3030 POKE U+M+40,UC
3040 FOR I=1 TO 500
3050 GET X$
3060 NEXT I
3070 PRINT CHR$(147)
3080 PRINT
3090 PRINT
3100 PRINT
3110 PRINT TAB(15)"ACCIDENT !!"
3120 PRINT
3130 PRINT
3140 PRINT
3150 PRINT TAB(12)"KMS PARCOURUS :";K
3160 PRINT
3170 PRINT
3180 PRINT
3190 PRINT TAB(15)"UNE AUTRE ?"
3200 GET X$
3210 IF X$="" THEN 3200
3220 IF X$<>"N" THEN RUN
3230 END
```

Tiré de «Jeux en BASIC sur Commodore 64» de Pierre Monsaut, Editions SYBEX, Réf. 317, Ft 16 × 22, 96 p., 49 F.



SYBEX 6-8, impasse du Curé
75881 PARIS-CEDEX 18
Tél. : 203.95.95



SERPENT

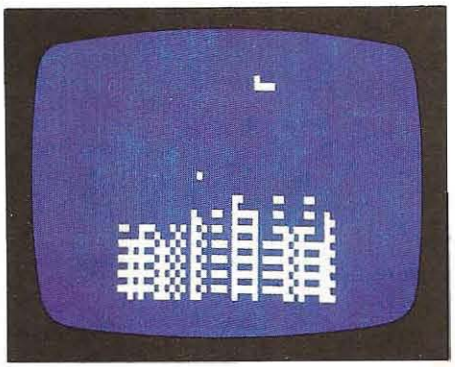
Un jeu pour COMMODORE 64

Dans ce jeu, vous êtes un serpent qui se déplace en ondulant sur l'écran. Le changement de direction s'effectue en tapant n'importe quelle touche. Pour pouvoir vous déplacer, vous devez vous nourrir. Heureusement, vous êtes entouré par un grand nombre de champignons. Mais attention ! Si les bleus sont excellents, vous devez absolument éviter les noirs qui, eux, sont vénéneux. Chaque champignon bleu vous apporte suffisamment de calories pour avancer de dix lignes. Essayez de ne pas mourir de faim sans pour autant finir empoisonné !

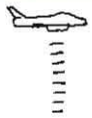
```
5 REM  ***********
10 REM * SERPENT *
15 REM ***********
20 GOSUB 1000
100 GET X$
110 IF X$<>"" THEN D=-D
120 T=T+D
130 IF T<L0 THEN T=L0
140 IF T>L1 THEN T=L1
150 T1=T+40
160 IF PEEK(T1)=65 THEN 500
170 IF PEEK(T1)=88 THEN S=S+10:H=H+10
180 PRINT B$;
190 P=INT(RND(TI)*40)
200 POKE 1984+P,88
210 POKE 1984+P+M,BC
220 P=INT(RND(TI)*40)
230 IF RND(TI)<0.5 THEN 260
240 POKE 1984+P,65
250 POKE 1984+P+M,RC
260 POKE T,CT
270 POKE T+M,TC
280 S=S-1
290 IF S=0 THEN 500
300 H=H+1
310 GOTO 100
500 PRINT B$;
510 POKE T,CT
520 POKE T+M,TC
530 FOR I=1 TO 500
540 NEXT I
550 IF H>RE THEN RE=H
560 PRINT CHR$(19);
580 FOR I=1 TO 10
590 PRINT B$;
600 NEXT I
610 PRINT TAB(13)"SCORE :";H;CHR$(19);
620 FOR I=1 TO 15
630 PRINT B$;
640 NEXT I
650 PRINT TAB(13)"RECORD :";RE;CHR$(19);
660 FOR I=1 TO 20
670 PRINT B$;
690 GET X$
700 NEXT I
710 PRINT TAB(13)"UNE AUTRE ?";
720 GET X$
730 IF X$="" THEN 720
740 IF X$<>"N" THEN 20
750 END
1000 PRINT CHR$(147)
1010 POKE 53280,5
1020 POKE 53281,13
1030 CT=19
1040 TC=9
1050 D=-1
1060 T=1404
1070 T1=T
1080 BC=6
1090 RC=0
1100 S=100
1110 M=54272
1120 B$=CHR$(17)
1130 L1=1423
1140 L0=1384
1150 H=0
1200 FOR I=0 TO 24
1210 PRINT B$;
1220 NEXT I
1230 PRINT CHR$(144);
1240 RETURN
```

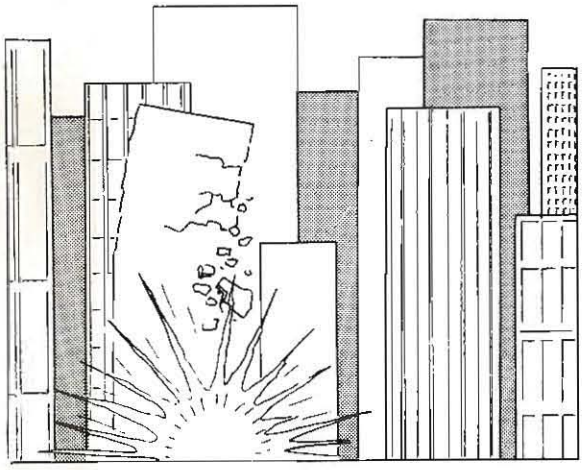
Tiré de «Jeux en BASIC sur Commodore 64» de Pierre Monsaut, Editions SYBEX, Réf. 317, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé 75881 PARIS-CEDEX 18 Tél. : 203.95.95



BLITZ





Un jeu pour DRAGON

Votre mission est de détruire la ville que vous survolez afin de pouvoir atterrir. A chaque passage votre avion vole un peu plus bas. Vous ne pouvez larguer une bombe (en appuyant sur une touche quelconque) que lorsque la bombe précédente a atteint son objectif ou le sol. Lorsque votre avion a atterri (ou lorsqu'il s'est écrasé contre un immeuble), le score est affiché ainsi que le record du jour. Si ce jeu vous paraît trop difficile, vous pouvez changer les limites de la ville (6 et 26, ligne 100).

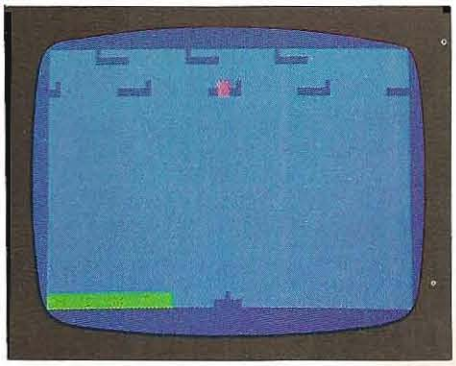
```
10 REM *********
20 REM * BLITZ *
30 REM *********
34 REM
35 REM INITIALISATION
36 REM
39 REM A$=AVION
40 A$=CHR$(128)+CHR$(155)+CHR$(1
47)
49 REM B$=BOMBE
50 B$=CHR$(145)
58 REM H=POSITION DE L'AVION ET
59 REM SCORE
60 H=0
69 REM B=POSITION DE LA BOMBE
70 B=0
80 B1=B
90 CLS 0
94 REM
95 REM AFFICHAGE DE LA VILLE
96 REM
99 REM 6 ET 26 : LIMITES VILLE
100 FOR I=6 TO 26
110 C=RND(7)+151
114 REM LIGNE 120: REMPLACER 8
115 REM PAR VALEUR SUPERIEURE
116 REM POUR DIMINUER LA HAUTEUR
117 REM DES IMMEUBLES
120 FOR J=15 TO RND(4)+8 STEP-1
130 PRINT@ J*32+I,CHR$(C);
140 NEXT J
150 NEXT I
154 REM
155 REM BOUCLE PRINCIPALE
156 REM
159 REM AFFICHAGE AVION
160 PRINT@ H,A$;
169 REM AVION ECRASE?
170 IF PEEK(1027+H)<>128 THEN 24
0
179 REM TIR
180 IF INKEY$<>"" AND B=0 THEN B
=H+33
189 REM BOMBE ATTEINT LE SOL
190 IF B<>0 THEN GOSUB 360
199 REM PAS DE BOMBE LACHEE?
200 IF B=0 THEN GOSUB 400
209 REM AVANCE AVION
210 H=H+1
219 REM AVION POSE?
220 IF H=507 THEN 240
230 GOTO 160
234 REM
235 REM AVION POSE OU ECRASE
236 REM
239 REM RECORD BATTU?
240 IF H>R THEN R=H
250 PRINT @ 3,"SCORE :";H,
260 PRINT "RECORD :";R;
270 FOR I=1 TO 100
280 NEXT I
290 R$=INKEY$
300 PRINT@ 73,"UNE AUTRE ?";
310 R$=INKEY$
320 IF R$="" THEN 310
330 IF R$<>"N" THEN 40
340 CLS
350 END
354 REM
355 REM BOMBE LACHEE
356 REM
359 REM SOL ATTEINT?
360 IF B>=510 THEN B=0
365 REM AFFICHAGE BOMBE
370 PRINT@ B1,CHR$(128);
380 IF B<>0 THEN PRINT@ B,B$;:B1
=B:B=B+32
390 RETURN
394 REM
395 REM DELAI POUR RALENTIR
396 REM L'AVION SI AUCUNE BOMBE
397 REM N'EST LACHEE
398 REM
400 FOR I=1 TO 20
410 NEXT I
420 RETURN
```

Tiré de «Jeux en BASIC sur ORAGON» de Pierre Monsaut, Éditions SYBEX, Réf. 324, Ft 16 × 22, 96 p., 49 F.



SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. 203.95.95

Dans le prochain numéro : **Chiffres**, un programme-jeu pour Atari.



D.C.A.

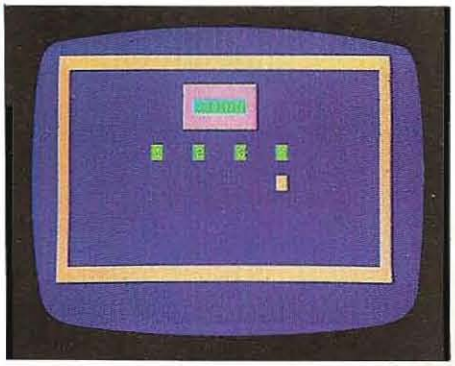
Un jeu pour DRAGON

Les rôles sont maintenant inversés. Vous manœuvrez la D.C.A. et devez essayer d'abattre les avions qui passent au-dessus de vous. Pous tirer, utilisez n'importe quelle touche. Vous disposez au départ de dix missiles. Si vous abattez huit avions, vous obtenez un bonus de huit points et huit missiles supplémentaires.

```
10 REM **********
20 REM * D.C.A. *
30 REM **********
40 CLEAR 250
50 GOTO 490
60 A$=RIGHT$(A$,1)+LEFT$(A$,31)
70 B$=RIGHT$(B$,31)+LEFT$(B$,1)
80 PRINT@ 0,A$;
90 PRINT@ 64,B$;
100 R$=INKEY$
110 IF R$<>"" AND M=495 THEN M=4
63:NM=NM-1
120 IF M<>495 THEN M=M-64:PRINT@
 M,M$;:PRINT@ M+64,O$;
130 IF M<>143 THEN 210
140 IF POINT (32,5)<>0 THEN 210
150 PRINT@ 79,CHR$(191);
160 PRINT@ M,O$;
170 SOUND 1,1
180 S=S+1
190 B$=LEFT$(B$,14)+O1$+RIGHT$(B
$,13)
200 GOTO 280
210 IF M<>79 THEN 300
220 IF POINT(30,1)<>0 THEN 300
230 PRINT@ 15,CHR$(191);
240 S=S+1
250 PRINT@ M,O$;
260 SOUND 1,1
270 A$=LEFT$(A$,13)+O1$+RIGHT$(A
$,14)
280 IF S>1 AND INT(S/8)=S/8 THEN
 GOSUB 420
290 M=495
300 PRINT@ 480,"S:";S;"M:";NM;
310 IF NM<1 AND M=495 THEN 340
320 IF M<32 THEN M=495
330 GOTO 60
340 IF S>R THEN R=S
350 PRINT@ 195,"SCORE :";S,"RECO
RD :";R;
360 PRINT@ 233,"UNE AUTRE ?";
370 R$=INKEY$
380 IF R$="" THEN 370
390 IF R$<>"N" THEN 50
400 CLS
410 END
420 A$=A1$
430 B$=B1$
440 NM=NM+8
450 FOR I=1 TO 300
460 NEXT I
470 S=S+8
480 RETURN
490 S=0
500 A$=""
510 B$=""
520 FOR I=1 TO 32
530 READ A,B
540 A$=A$+CHR$(A)
550 B$=B$+CHR$(B)
560 IF I/8=INT(I/8) THEN RESTORE
570 NEXT I
580 A1$=A$
590 B1$=B$
600 J$=CHR$(174)+CHR$(168)+CHR$(
172)
610 M=495
620 M$=CHR$(171)
630 NM=10
640 O$=CHR$(175)
650 CLS 3
660 PRINT@ 494,J$;
670 PRINT@ 0,A$;
680 PRINT@ 64,B$;
690 O1$=""
700 FOR I=1 TO 5
710 O1$=O1$+CHR$(175)
720 NEXT I
730 GOTO 60
740 DATA 175,175,164,172,172,172
,172,168.175,175,175,175,175,175
,175,175
```

Tiré de «Jeux en BASIC sur DRAGON» de Pierre Monsaut, Editions SYBEX, Réf. 324, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



SIMON

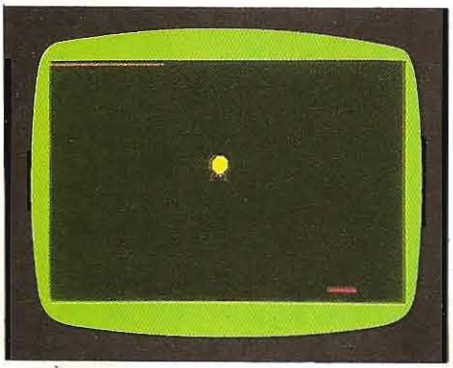
Un jeu pour DRAGON

Dans cette version du jeu bien connu, vous utiliserez les touches <1>, <2>, <3>, et <4> pour répéter la séquence produite de façon aléatoire par l'ordinateur. A chaque touche correspondent une couleur et un son :
(du plus grave au plus aigu)
1 = blanc ; 2 = bleu ; 3 = violet ; 4 = orange.
La limite de la longueur de la séquence à répéter est 100. Lorsque vous faites une erreur, le jeu s'arrête et votre score, correspondant à la longueur de la plus longue séquence que vous avez réussi à reproduire, est affiché ainsi que le record du jour.

```
10 REM *********
20 REM * SIMON *
30 REM *********
40 DIM T(100)
50 X=0
60 CLS 0
70 GOSUB 370
79 REM X=LONGUEUR DE LA SEQUENCE
80 X=X+1
90 T(X)=RND(4)
100 FOR I=1 TO X
110 GOSUB 330
120 NEXT I
130 FOR I=1 TO X
139 REM ATTENTE
140 FOR J=1 TO 1000
150 D$=INKEY$
160 IF D$<>"" THEN 190
170 NEXT J
179 REM TROP LONG, PERDU
180 GOTO 250
189 REM ERREUR?
190 IF ASC(D$)-48<>T(I) THEN 250
200 GOSUB 330
210 NEXT I
220 FOR I=1 TO 500
230 NEXT I
240 GOTO 80
244 REM
245 REM PERDU
246 REM
250 X=X-1
260 IF X>R THEN R=X
270 PRINT@ 324,"SCORE :";X,"RECO
RD :";R;
280 PRINT@ 394,"UNE AUTRE ?";
290 D$=INKEY$
300 IF D$="" THEN 290
310 IF D$<>"N" THEN 50
320 END
324 REM
325 REM AFFICHAGE DE LA SEQUENCE
326 REM
330 POKE 1285+4*T(I),191+T(I)*16
340 SOUND 50+T(I)*45,2
350 POKE 1285+4*T(I),128
360 RETURN
364 REM
365 REM INITIALISATION
366 REM
369 REM TRACE DU CADRE
370 FOR I=1024 TO 1055
380 POKE I,255
390 POKE I+448,255
400 NEXT I
410 FOR I=1 TO 13
420 POKE 1024+I*32,255
430 POKE 1055+I*32,255
440 NEXT I
449 REM AFFICHAGE DES CHIFFRES
450 FOR I=1 TO 4
460 POKE 1221+4*I,I+48
470 NEXT I
480 FOR I=1 TO 7
490 POKE 1099+I,239
500 POKE 1163+I,239
510 NEXT I
520 POKE 1132,239
530 POKE 1138,239
540 PRINT@ 109,"SIMON";
550 RETURN
```

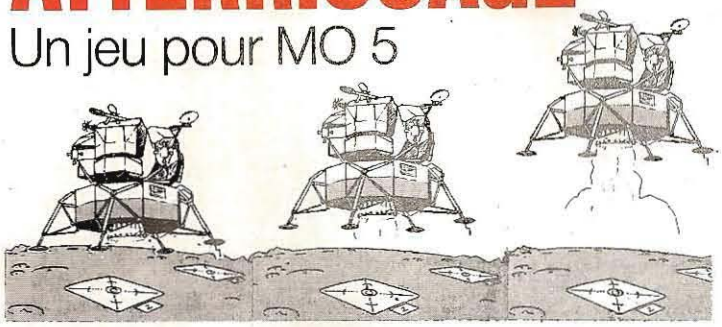
Tiré de «Jeux en BASIC sur DRAGON»
de Pierre Monsaut, Editions SYBEX
Réf. 324, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



ATTERRISSAGE

Un jeu pour MO 5



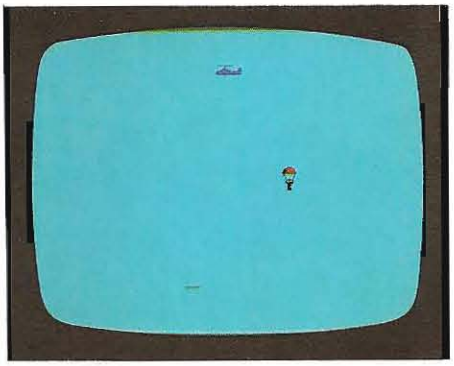
Après un long voyage en apesanteur, poser une navette spatiale en douceur n'est pas chose aisée; mais grâce à votre ordinateur, vous allez être en mesure de vous entraîner sans danger. Vous devez poser votre navette sur l'aire prévue à cet effet. Vous pouvez vous diriger vers la droite et vers la gauche à l'aide des touches < , > et <. > ou freiner votre descente avec la barre d'espacement. (Vous pouvez également utiliser le joystick.) La quantité de carburant disponible est indiquée par la longueur du trait horizontal, en haut de l'écran.

Si vous n'avez pas de joystick, supprimez les lignes 140 à 160.

```
10 REM ****************
20 REM * ATTERRISSAGE *
30 REM ****************
40 DEFINT A-Z
50 DIM N(15,20)
60 DIM R(15,20)
70 DIM C(15,10)
80 FU=100
90 GOTO 620
100 GOSUB 750
110 DV=DV+1
120 IF FU<1 THEN D$="":GOTO 170
130 D$=INKEY$
140 Q=STICK(0)
150 DV=DV+3*(Q=5)
160 DH=DH+(Q=3)-(Q=7)
170 DV=DV+3*(D$=" ")
180 DH=DH+(D$=".")-(D$=",")
190 IF D$<>"" OR Q<>0 THEN FU=FU-1:FF=1
200 DRAW "COL"+STR$(FF)
210 FF=0
220 IF MH<0 OR NH>319 THEN 250
230 IF MV<0 THEN 250
240 PUT (MH,MV)-(NH,NV),R
250 NH=NH+DH
260 NV=NV+DV
270 IF NV>187 THEN 340
280 MV=NV-20
290 MH=NH-15
300 IF MH<0 OR NH>319 THEN 110
310 IF MV<0 THEN 110
320 PUT (MH,MV)-(NH,NV),N
330 GOTO 110
340 IF DV>5 THEN 420
350 IF ABS(NH-A)>4 THEN 420
360 IF ABS(DH)>1 THEN 420
370 PUT (MH,167)-(NH,187),N
380 FOR I=0 TO 1000
390 NEXT I
400 S=S+10
410 GOTO 100
420 FOR I=1 TO 10
470 NEXT I
480 CLS
490 SCREEN 4,6,6
500 LOCATE 5,10
510 PRINT "VOTRE NAVETTE S'EST ECRASEE"
520 LOCATE 10,13
530 PRINT "SCORE :";S;
540 LOCATE 10,16
550 PRINT "UNE AUTRE ?";
560 IF INKEY$<>"" THEN 560
570 D$=INKEY$
580 IF D$="" THEN 570
590 IF D$<>"N" THEN RUN
600 CLS
610 END
620 CLS
630 SCREEN 3,0,2
640 LOCATE 0,0,0
650 FOR I=1 TO 15
660 FOR J=0 TO 10
670 C(I,J)=RND*1000
680 NEXT J
690 NEXT I
700 DRAW "C3BM50,50R2L1U3E3H4U6E4R7F4D6G
4F3D3L1R2L1U3H3L3D2R1D1U1L3D1U1R1U2L3"
710 PAINT (55,40),3
720 GET (50,30)-(65,50),N
730 GET (0,0)-(15,20),R
740 GOTO 100
750 CLS
760 A=INT(RND*260)+20
770 DRAW "C1BM"+STR$(A)+",188R4L23D1R23D
1L23"
780 NH=160
790 NV=20
800 DH=0
810 DV=0
820 MV=0
830 MH=NH-15
840 FU=FU+10
850 DRAW "C1BM0,3R"+STR$(FU)
860 RETURN
```

Tiré de « MO 5 JEUX D'ACTION» de Pierre Monsaut, Editions SYBEX, Réf. 367, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



PARACHUTE

Un jeu pour MO5

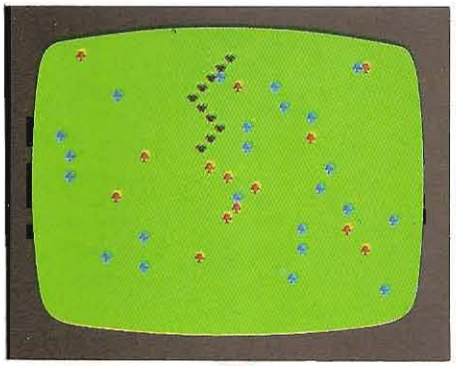


Essayez, en sautant d'un hélicoptère en vol, d'atteindre la cible qui se trouve au sol. Une première pression sur une touche vous permet de descendre verticalement en chute libre. Une deuxième pression entraîne l'ouverture du parachute. La descente continue alors plus lentement et avec un angle de 45 degrés car le vent vous pousse. Plus vous attendrez pour ouvrir votre parachute et moins vous serez déporté. Mais n'attendez pas trop longtemps car, au-dessous de cent mètres, le parachute ne s'ouvre plus.

```
10 REM *************
20 REM * PARACHUTE *
30 REM *************
40 GOSUB 470
50 HH=HH-4
60 IF HH=0 THEN PUT (0,1)-(27,9),R
70 IF HH=0 THEN HH=288
80 PUT (HH,1)-(HH+27,9),H
90 D$=INKEY$
100 IF D$="" THEN 140
110 IF PV>100 THEN 140
120 IF SP=1 THEN OP=1 ELSE SP=1
130 IF OP=0 THEN PV=10:PH=HH
140 IF SP=0 THEN 230
150 IF OP=0 THEN PV=PV+8
160 IF OP=1 THEN PV=PV+1:PH=PH-1
170 IF PV>167 OR PH<1 THEN 260
180 IF OP=1 THEN 210
190 PUT (PH,PV)-(PH+14,PV+23),PF
200 GOTO 50
210 PUT (PH,PV)-(PH+14,PV+23),PO
220 GOTO 50
230 FOR I=1 TO 50
240 NEXT I
250 GOTO 50
260 IF ABS(PH-A)>4 THEN 320
270 FOR I=1 TO 1000
280 NEXT I
290 S=S+10
300 GOSUB 670
310 GOTO 50
320 CLS
330 SCREEN 1,2,2
340 LOCATE 10,10
350 ATTRB 1,1
360 PRINT "SCORE :";S;
370 LOCATE 10,16
380 PRINT "UNE AUTRE ?";
390 ATTRB 0,0
400 IF INKEY$<>"" THEN 400
410 D$=INKEY$
420 IF D$="" THEN 410
430 IF D$<>"N" THEN RUN
440 CLS
450 SCREEN 4,6,6
460 END
470 LOCATE 0,0,0
480 SCREEN 4,6,6
490 DEFINT A-Z
500 DIM H(27,8)
510 DIM PF(14,23)
520 DIM PO(14,23)
530 DIM R(27,8)
540 CLS
550 DRAW "C4BM51,50R14L6G4L2G1L1G1D1F1R2
1E1U5L2D1G2L5H1L1H1L1H1L1"
560 PAINT (58,56),4
570 GET (50,50)-(77,58),H
580 GET (100,50)-(127,58),R
590 CLS
600 DRAW "C1BM54,50G1L1G1D1G1D1R12U1H1U1
H1L1H1L4"
610 PAINT (56,54),1
620 DRAW "C1BM51,56D1F1D1F1D3R2D6L1R1U3R
2D3R1L1U4L1U4L1U1R2D1L1D4R1U2R2U3E1U1E1U
1"
630 GET (50,46)-(64,69),PO
640 CLS
650 DRAW "C1BM54,61R1D6L2D2U2R3D2L1D1R2U
1L1U2R3D2U2L3U3D3R1U6R1"
660 GET (50,46)-(64,69),PF
670 CLS
680 HH=288
690 HV=1
700 A=INT(RND*191)+10
710 DRAW "C0BM"+STR$(A)+",191R12"
720 SP=0
730 OP=0
740 PV=0
750 PH=0
760 RETURN
```

Tiré de «MO5 JEUX D'ACTION»
de Pierre Monsaut, Editions SYBEX
Réf. 367, Ft 16 x 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



SERPENT

Un jeu pour MO 5

Dans ce jeu, vous êtes un serpent qui se déplace en ondulant sur l'écran. Le changement de direction s'effectue en tapant n'importe quelle touche. Pour pouvoir vous déplacer, vous devez vous nourrir. Heureusement, vous êtes entouré par un grand nombre de champignons. Mais attention ! Si les bleus sont excellents, vous devez absolument éviter les rouges qui, eux, sont vénéneux. Chaque champignon bleu vous apporte suffisamment de calories pour avancer de dix lignes. Essayez de ne pas mourir de faim sans pour autant finir empoisonné !

```
10 REM ***********
20 REM * SERPENT *
30 REM ***********
40 CLEAR ,,2
50 GOSUB 540
60 D$=INKEY$
70 IF D$<>"" THEN D=-D
80 X=X+D
90 IF X<0 THEN X=1
100 IF X>39 THEN X=38
110 IF POINT(X*8+4,Y*8+4)=1 THEN 260
120 IF POINT(X*8+4,Y*8+4)=4 THEN S=S+10:
H=H+10:BEEP
130 LOCATE X,Y
140 COLOR 0
150 PRINT S$;
160 LOCATE 0,24,0
170 PRINT
180 LOCATE INT(RND*38)+1,24
190 COLOR 4
200 PRINT C$;
210 IF RND>0.5 THEN LOCATE INT(RND*38)+1
,24:COLOR 1:PRINT C$;
220 S=S-1
230 IF S=0 THEN 260
240 H=H+1
250 GOTO 60
260 BEEP
270 LOCATE 0,24
280 PRINT
290 LOCATE X,Y-1
300 COLOR 5
310 PRINT S$;
320 FOR I=1 TO 5
330 BEEP
340 FOR J=1 TO 50
350 NEXT J
360 NEXT I
370 IF INKEY$<>"" THEN 370
380 IF H>R THEN R=H
390 LOCATE 4,20
400 COLOR 0
410 PRINT "SCORE :";H;
420 LOCATE 23,20
430 PRINT "RECORD :";R;
440 LOCATE 15,23
450 PRINT "UNE AUTRE ?"
460 D$=INKEY$
470 IF D$="" THEN 460
480 IF D$<>"N" THEN 520
490 SCREEN 4,6,6
500 CLS
510 END
520 GOSUB 610
530 GOTO 60
540 CLS
550 SCREEN 2,2,2
560 DEFINT A-Z
570 DEFGR$(0)=0,0,102,255,219,126,60,24
580 DEFGR$(1)=60,126,126,255,255,24,24,6
0
590 S$=GR$(0)
600 C$=GR$(1)
610 CLS
620 S=100
630 H=0
640 D=1
650 X=19
660 Y=10
670 RETURN
```

Tiré de «MO 5 JEUX D'ACTION» de Pierre Monsaut, Editions SYBEX, Réf. 367, Ft 16 × 22, 96 p., 49 F.

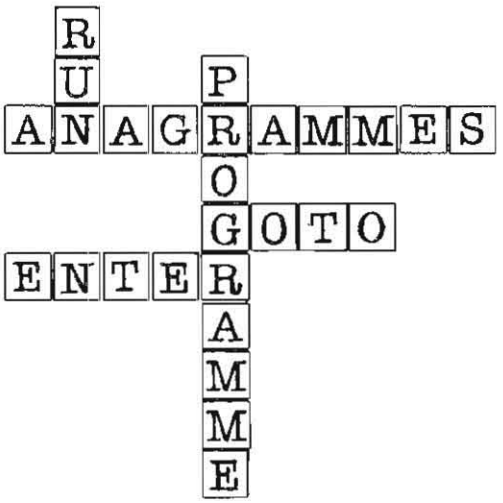
SYBEX 6-8, impasse du Curé 75881 PARIS-CEDEX 18 Tél. : 203.95.95



ANAGRAMME

Un jeu pour ORIC



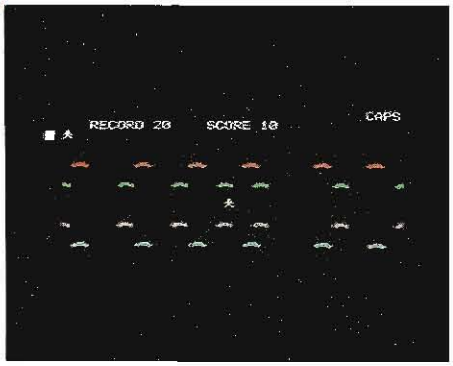


Dans ce jeu, vous disposez de dix essais pour trouver l'anagramme. Ce programme utiiise un dictionnaire de quarante mots. Il peut être étendu en modifiant le nombre de mots (42) dans la ligne 9000.

```
10 REM ANAGRAMME
20 REM PETER SHAW 1983
30 GOSUB 9000'CHOIX MOT
35 ZAP
40 GOSUB 8000
45 EXPLODE
50 FOR A=1 TO 10
60 GOSUB 7000
70 GOSUB 6000
75 PING:GOSUB 8000
80 NEXT A
90 REM ********************************

100 REM PERDU
110 CLS:PLOT 7,2,"LE MOT EST "+W$
120 A=1
130 REPEAT
140 PLOT 12,10,CHR$(A)+"PERDU!!"
150 A=A+1:IF A=8 THEN A=1
155 WAIT 4:PLAY 1,0,1,100:SOUND 1,100,A
160 UNTIL KEY$<>""
170 PLAY 0,0,0,0:RUN
5000 REM GAGNE
5010 CLS
5020 A=1
5030 REPEAT
5040 PLOT 12,10,CHR$(A)+"BRAVO!!"
5050 A=A+1:IF A=9  THEN A=1
5055 WAIT 4:PLAY 1,0,1,100:SOUND 1,100,A

5060 UNTIL KEY$<>""
5070 PLAY 0,0,0,0:RUN
6000 IF G$=W$ THEN 5000
6010 RETURN
7000 CLS
7010 PRINT"ESSAI ";A
7020 PRINT
7030 PRINT"VOTRE ANAGRAMME EST ";S$
7040 PRINT
7050 INPUT"TAPEZ LE MOT ";G$
7060 RETURN
8000 L=LEN(W$)
8010 S$=LEFT$("                    ",L)
8020 FOR I=1 TO L
8030 R=INT(RND(1)*L)+1
8040 IF MID$(S$,R,1)>" " THEN GOTO 8030
8050 S$=LEFT$(S$,R-1)+MID$(W$,I,1)+MID$(
S$,R+1)
8060 NEXT I
8070 RETURN
9000 FOR A=1 TO INT(RND(1)*42)+1
9010 READ W$
9020 NEXT A
9030 RETURN
9040 DATA PROGRAMME,SOFTWARE,BASIC,HARDW
ARE,AFFICHAGE
9050 DATA ORDINATEUR,GRAPHIQUE,CLAVIER,I
MPRIMANTE
9060 DATA JEUX,SILICONE,ARCADE,MEMOIRE,P
UCE
9070 DATA ASSEMBLEUR,INTERFACE,VIDEO
9080 DATA DIGITAL,PARALLELE,DISQUE,MONIT
EUR
9090 DATA PROCESSEUR,SERIE,TRANSFERT,MAC
HINE
9100 DATA LANGUAGE,CASSETTE,PRISME,ACCES
,ALEATOIRE
9110 DATA CODEUR,COMPILER,ADAPTER,CIRCUI
T
9120 DATA PASCAL,COBOL , ECRAN,FORTRAN
9130 DATA LISTING,ANAGRAMME,EDITEUR,CONN
ECTER
```

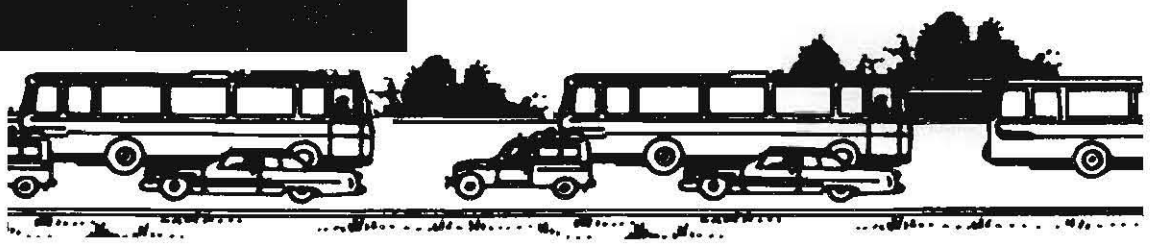
Tiré de «Jeux en BASIC sur ORIC»
de Peter Shaw, Editions SYBEX
Réf. 278, Ft 16 × 22, 96 p., 49 F

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95



AUTOROUTE

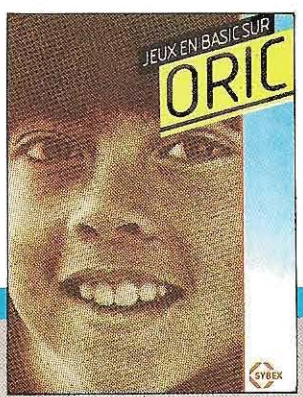
Un jeu pour ORIC



Traverser l'autoroute à l'heure de pointe n'est une partie de plaisir pour personne, mais c'est pourtant le but de ce jeu. Vous devez éviter deux files de voitures roulant en sens inverse ; si vous êtes écrasé par un véhicule, vous perdez l'une de vos six vies. Pour chaque piéton arrivé sain et sauf, vous gagnez dix points. Utilisez la barre d'espacement pour faire traverser votre piéton. Il y a un record à battre !

```
10 REM AUTOROUTE
20 REM PETER SHAW 1983
30 RE=0:PRINT CHR$(6)
40 GOSUB 9000
50 GOSUB 8000
60 V=14
70 H=19
80 GOSUB 7000
90 PLOT H,V," "
100 IF KEY$=" " THEN V=V-1
110 PLAY 0,1,1,10
140 PLOT 1,4,A$:PLOT 1,6,B$
150 PLOT 1,10,B$:PLOT 1,12,A$
160 IF SCRN(H,V)<>32 THEN 1000
170 PLOT H,V,"#"
180 L1$=LEFT$(A$,1):R1$=RIGHT$(A$,37)
190 L2$=LEFT$(B$,37):R2$=RIGHT$(B$,1)
200 A$=R1$+L1$:B$=R2$+L2$
210 IF V=2 THEN 2000
220 GOTO 90
1000 EXPLODE:WAIT 50
1010 IFML>0 THEN ML=ML-1:GOTO 60
1020 IF SC>RE THEN RE=SC
1030 PLOT 6,20,"TAPEZ UNE TOUCHE POUR JO
UER"
1040 GET A$
1050 GOTO 50
2000 PING:WAIT 50
2010 SC=SC+10
2020 GOTO 60
```

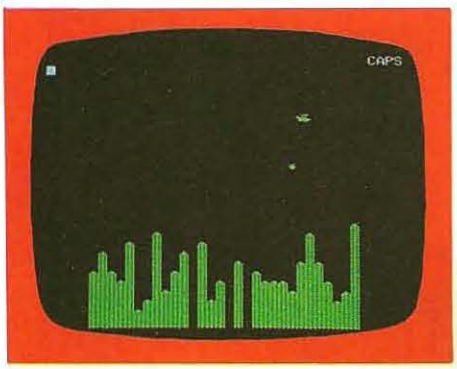
```
7000 CLS:PAPER 0:INK 7
7010 PRINT,"RECORD ";RE,"SCORE ";SC
7020 PLOT 0,4,CHR$(1)+A$
7030 PLOT 0,6,CHR$(2)+B$
7040 PLOT 0.10,CHR$(5)+B$
7050 PLOT 0,12,CHR$(6)+A$
7070 PLOT H,V,"#"
7080 FOR A=1 TO ML
7090 PLOT A,1,"#"
7100 NEXT A
7110 RETURN
7120 REM
8000 PAPER 0:INK 7
8010 SC=0
8020 B$="$%      $%      $%      $%    $%  $%
     "
8030 A$="'&      '&      '&      '&      '&
'&      "
8040 ML=5
8050 RETURN
8060 REM
9000 FOR A=(46080+(ASC("#")*8)) TO (4608
0+(ASC("'")*8)+7)
9010 READ US:POKE A,US:NEXT:RETURN
9020 DATA 12,12,30,45,12,18,33,0
9030 DATA 0,0,0,7,15,63,12,0
9040 DATA 0,0,0,48,62,62,6,0
9050 DATA 0,0,0,56,60,63,6,0
9060 DATA 0,0,0,3,31,63,24,0
```



Tiré de «Jeux en BASIC sur ORIC» de Peter Shaw, Éditions SYBEX, Réf. 278, Ft 16 × 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé 75881 PARIS CEDEX 18 Tél. 203.95.95

Dans le prochain numéro : **Blitz**, un programme-jeu pour Dragon.



BLITZ

Un jeu pour ORIC

Larguez vos bombes sur les immeubles pendant que votre avion traverse l'écran. Utilisez la barre d'espacement pour larguer vos bombes. Si vous désirez augmenter la difficulté, changez le nombre 10 ligne 8020 pour un nombre plus petit et le nombre 16 pour un plus grand.

```
10 REM BLITZ
20 REM PETER SHAW 1983
25 GOSUB 9000
26 GOSUB 8000
60 FOR A=0 TO 26
70 FOR B=1 TO 34
80 PLOT B,A," #$"
90 IF SCRN(B+3,A)=39 OR SCRN(B+4,A)=40 T
HEN GOTO 1000
95 PLOT B,A," #%"
100 X$=KEY$
110 IF X$=" " THEN GOSUB 250
120 PLAY 0,1,1,15
130 NEXT B
140 PLOT B,A,"    "
150 NEXT A
160 PLOT 1,26,"#$"
170 PRINT"                BRAVO!!"
180 PRINT
190 PRINT"ATTERRISSAGE REUSSI!!"
200 PRINT"VOUS AVEZ TUE ";A*B;" PERSONNE
S POUR"
210 PRINT"          SAUVER VOTRE VIE"
220 PRINT
230 PRINT" J'ESPERE QUE CELA NE PERTURBE
RA PAS  VOTRE SOMMEIL"
240 END
250 REM LANCEMENT BOMBE
260 B1=B
270 FOR DR=A TO 25
280 PLOT B,A," #$"
290 IF SCRN(B+3,A)=39 OR SCRN(B+4,A)=40
THEN GOTO 1000
300 PLOT B,A," #%"
305 B=B+1:IF B>34 THEN B=1:A=A+1:PLOT 34
,A-1,"      "
307 PLAY 0,1,1,10
320 PLOT B1,DR," ":PLOT B1,DR+1,"&"
340 NEXT DR
360 PLOT B,A,"    ":PLOT B1,26," "
370 RETURN
999 STOP
1000 EXPLODE
1010 FOR L=A TO 26
1020 PLOT B,L," #$"
1030 EXPLODE:WAIT L
1040 PLOT B,L,"    "
1050 NEXT L
1060 PRINT" SCORE ";A*10+B*10
1070 FOR A=0 TO 6
1080 INK A:WAIT 10
1090 NEXT A
1100 IF KEY$<>"" THEN 1100
1110 GET A$:RUN
8000 CLS:PAPER 0:INK INT(RND(1)*7)+1
8010 FOR S=33 TO 4 STEP -1
8020 HG=INT(RND(1)*10)+16
8030 PLOT S,HG,"'"
8040 FOR K=HG+1 TO 26
8050 PLOT S,K,"("
8060 NEXT K
8070 NEXT S
8080 RETURN
9000 FOR U=46359 TO 46407
9010 READ US:POKE U,US
9020 NEXT U:RETURN
9030 DATA 0,15,34,53,63,31,31,1
9040 DATA 0,48,32,4,52,60,52,4
9050 DATA 0,48,32,0,48,60,48,0
9060 DATA 0,18,12,30,30,12,0,0
9070 DATA 0,12,30,63,45,63,45,63
9080 DATA 45,63,45,63,45,63,45,63,45
```

Tiré de «Jeux en BASIC sur ORIC»
de Peter Shaw, Editions SYBEX
Réf. 278, Ft 16 x 22, 96 p., 49 F.

SYBEX 6-8, impasse du Curé
75881 PARIS CEDEX 18
Tél. : 203.95.95