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.

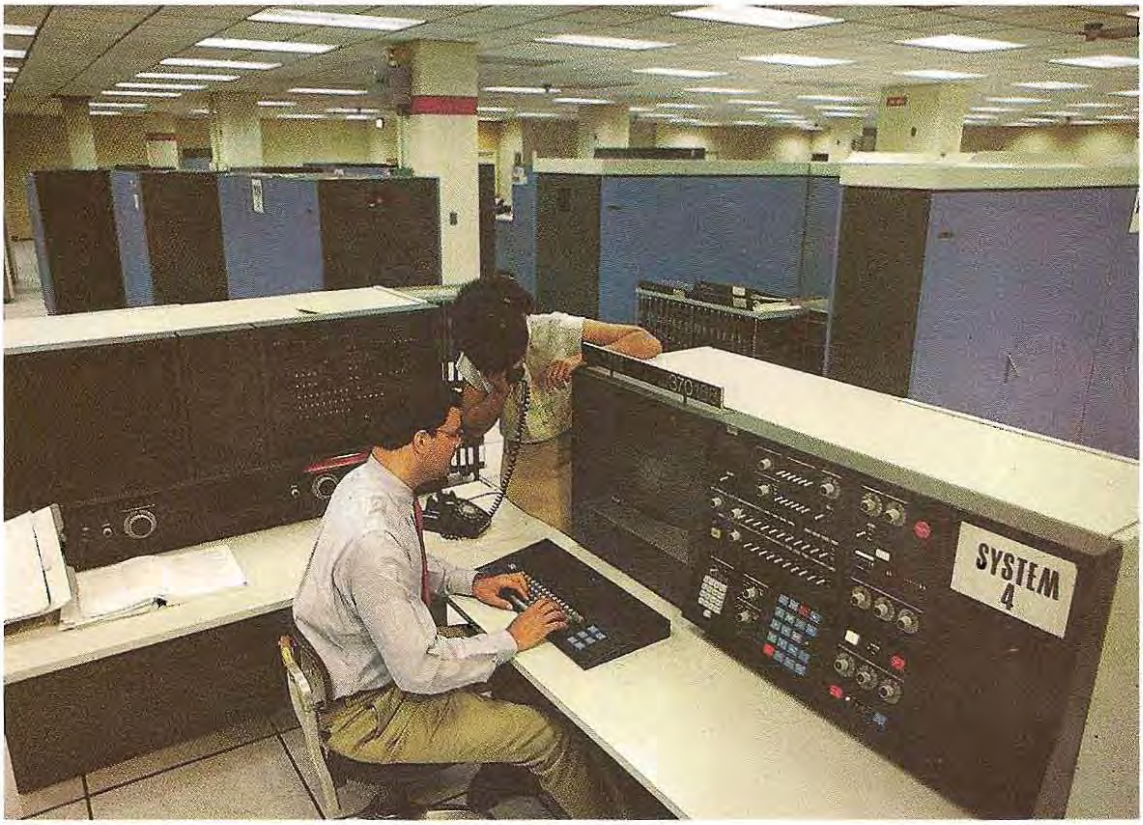
Véritable synthèse du projet, le modèle macroscopique sert de référence pendant la conception du logiciel, de guide pour la recherche des erreurs, et constitue un élément de comparaison pour évaluer le coût et la complexité des logiciels achevés ou en cours d'écriture.

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.

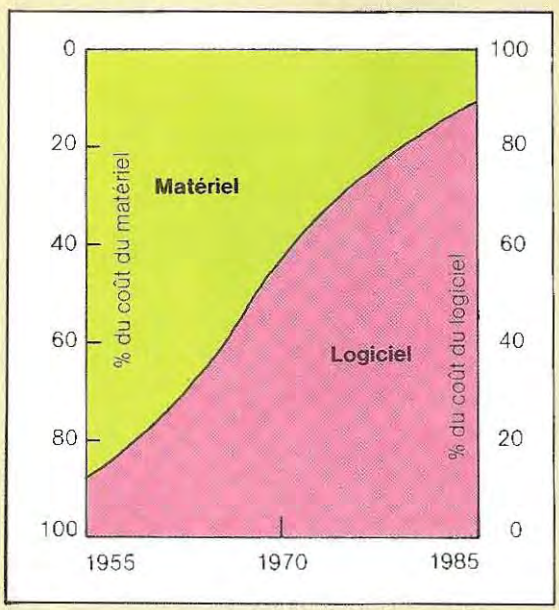
L'utilisation des abstractions, telle qu'elle est pratiquée en ***génie logiciel****, tend à mettre l'accent sur les propriétés fonctionnelles et les résultats voulus, au détriment de la manière de les obtenir.*

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.*

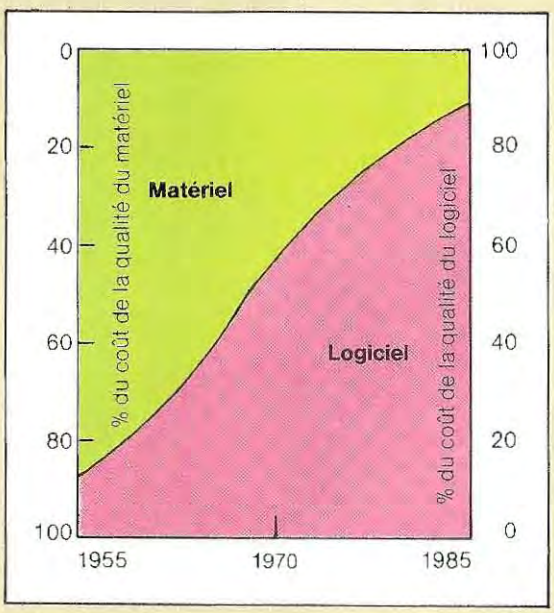
La recherche et le développement de nouvelles méthodologies et de nouveaux langages ont été menés en réponse aux besoins nés des limites étroites des langages (Basic, Fortran, Algol, etc.) et des techniques de programmation classiques.

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 »*

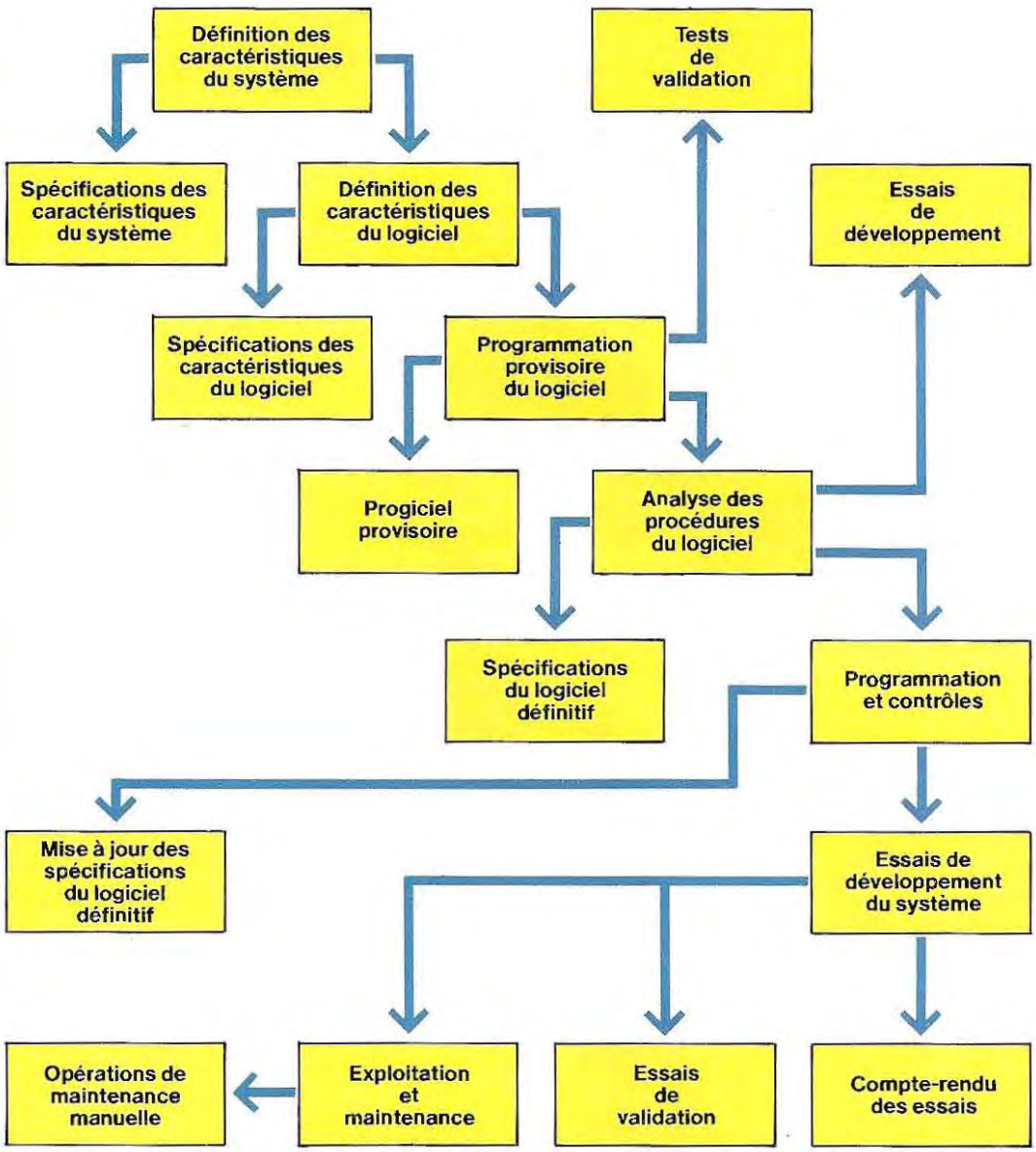
Répartition relative du coût d'un système informatique



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.

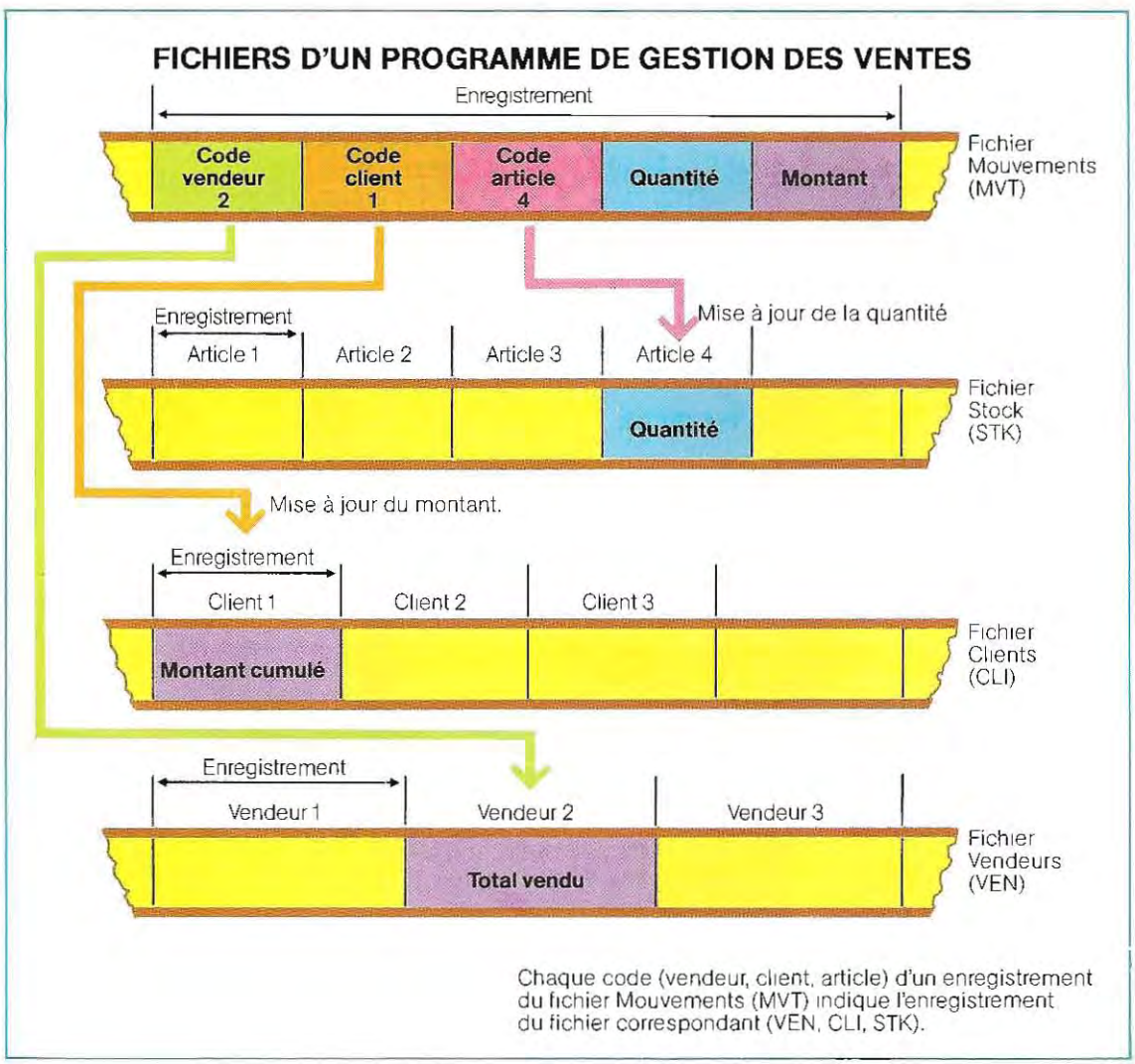
Code structuré. Codage des méthodes logicielles dans lequel l'accent est mis sur les structures logiques directes.

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.

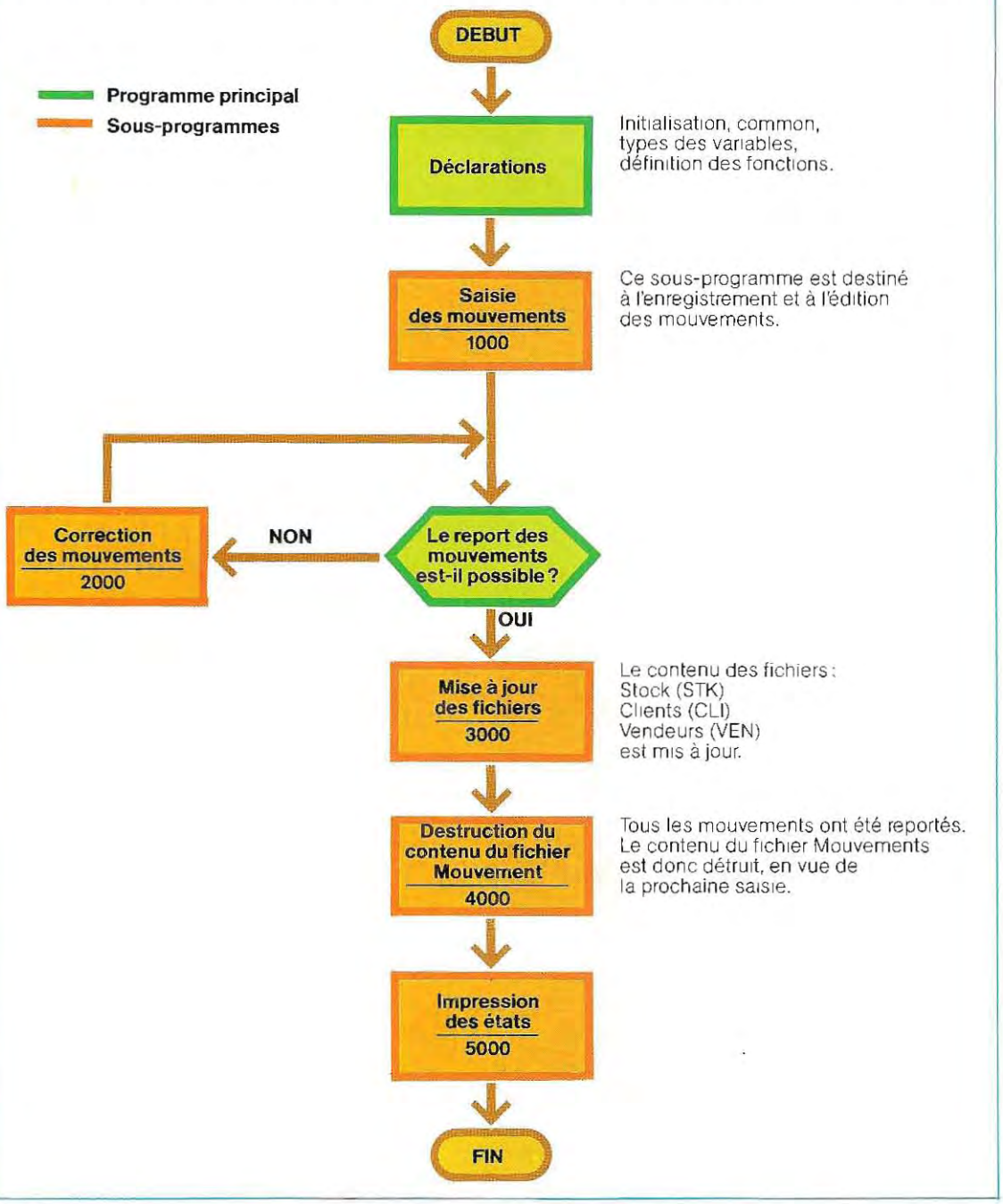
remplacer le module ou le sous-programme qui l'exécute. En outre, la validité du logiciel dans son ensemble est plus simple à établir.
L'exemple qui suit va nous permettre d'illustrer la technique de l'analyse descendante.
Une société emploie plusieurs vendeurs. On veut enregistrer les ventes de chacun (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.

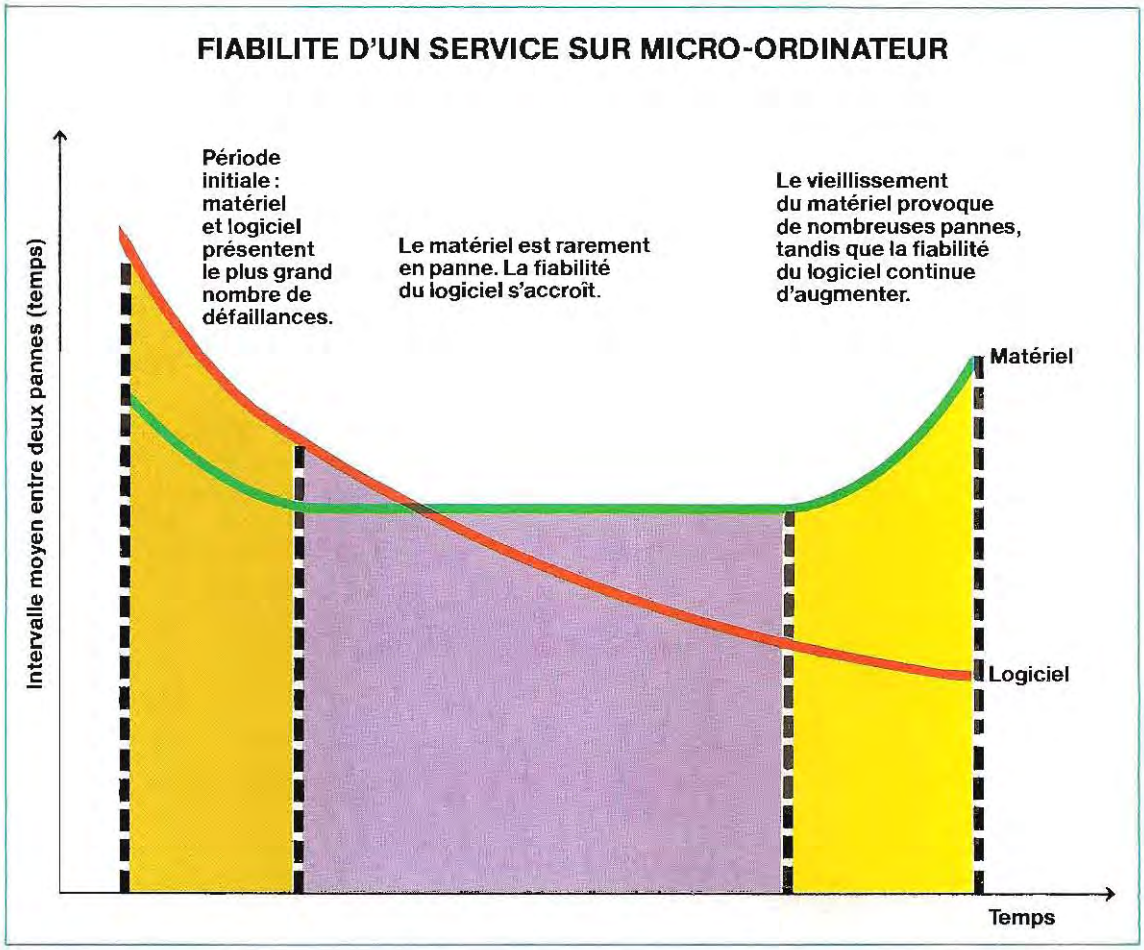
De toute façon, une erreur peut toujours se produire à la suite d'une mauvaise utilisation du logiciel ; il est impensable de prévoir toutes les actions incorrectes et de mettre en place un contrôle pour chacune d'elles. On ne peut protéger un logiciel que contre les erreurs les plus graves ou les plus 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

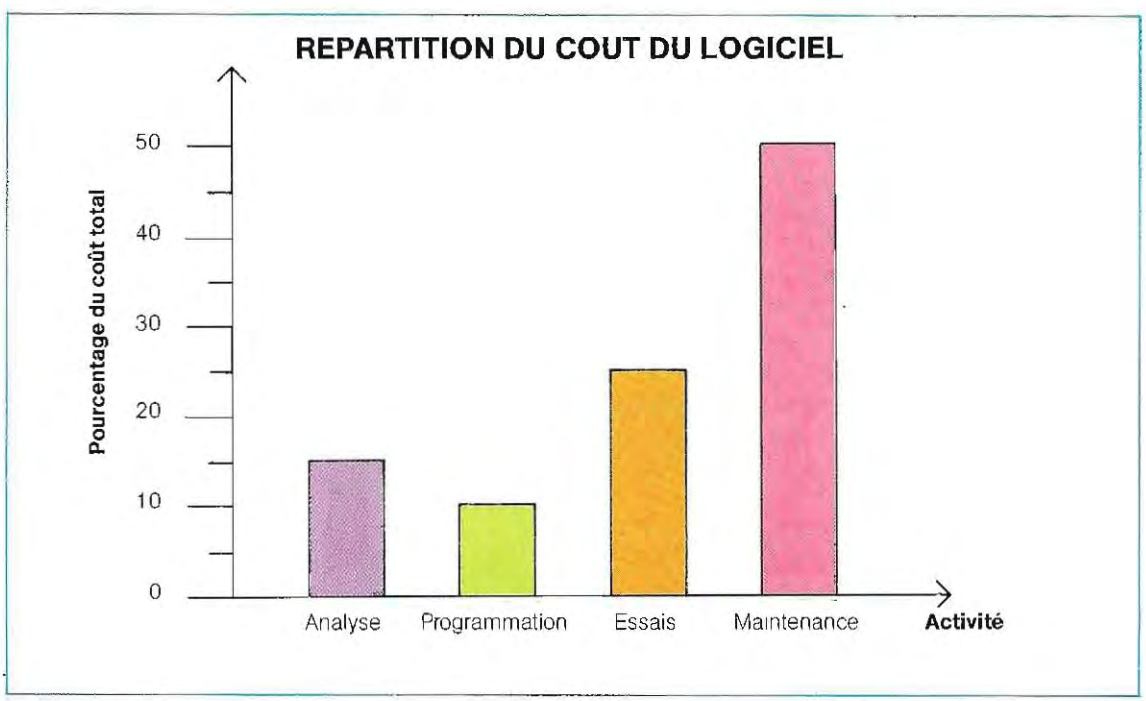
Cette étape de test est la plus délicate et la plus complexe lors de la production d'un logiciel, et cela pour deux raisons essentielles. En premier lieu, c'est une activité mal connue et généralement mal abordée. D'autre part, on la considère très souvent comme une simple démonstration de l'absence 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

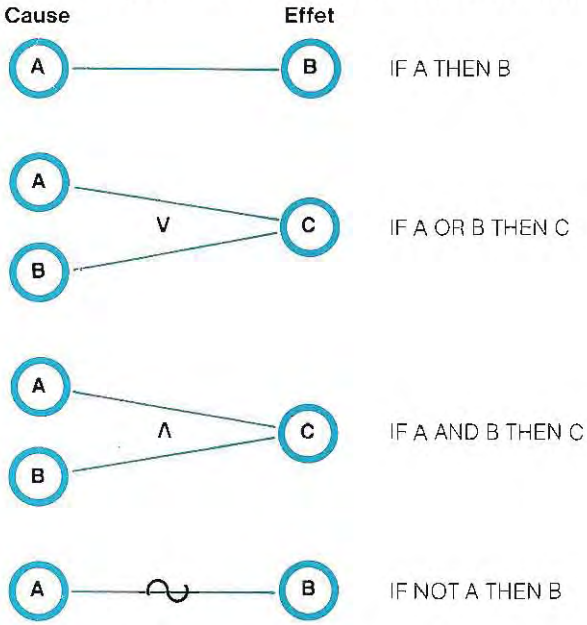
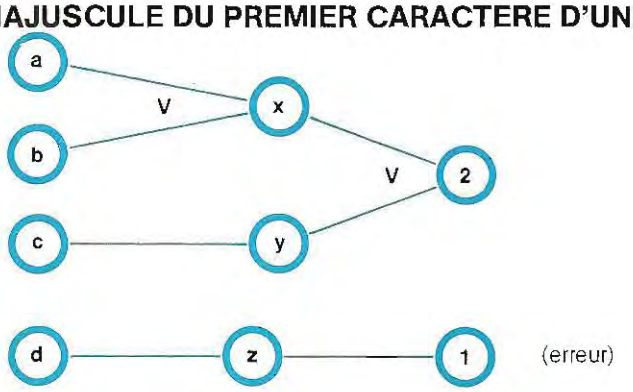


DIAGRAMME DE DECISION DU PROGRAMME DE CONVERSION EN MAJUSCULE DU PREMIER CARACTERE D'UNE CHAINE



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

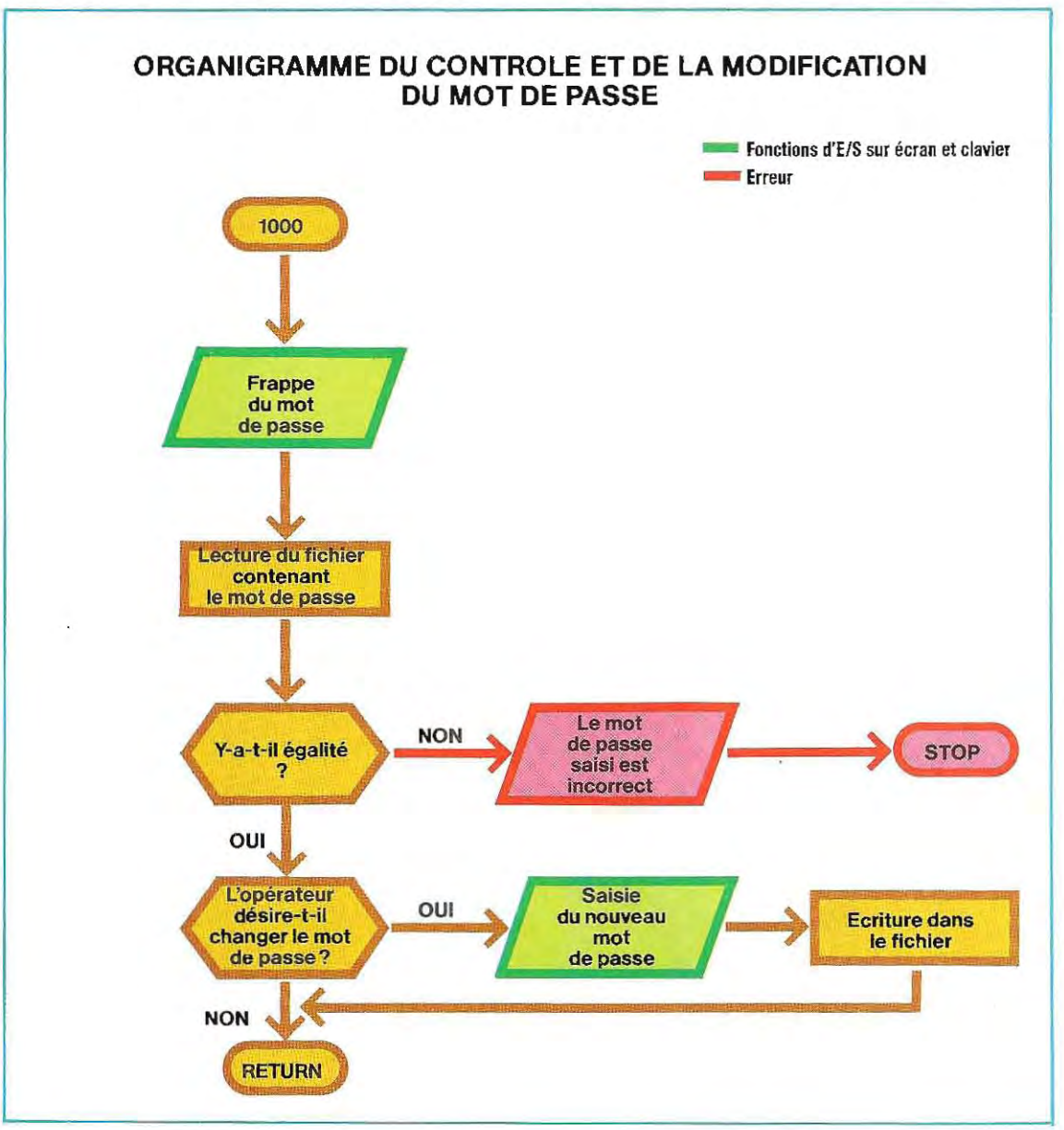
- What (quoi) : symptôme de l'erreur.
- Where (où) : endroit du programme où l'erreur se produit.
- When (quand) : conditions dans lesquelles l'erreur se produit.

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.

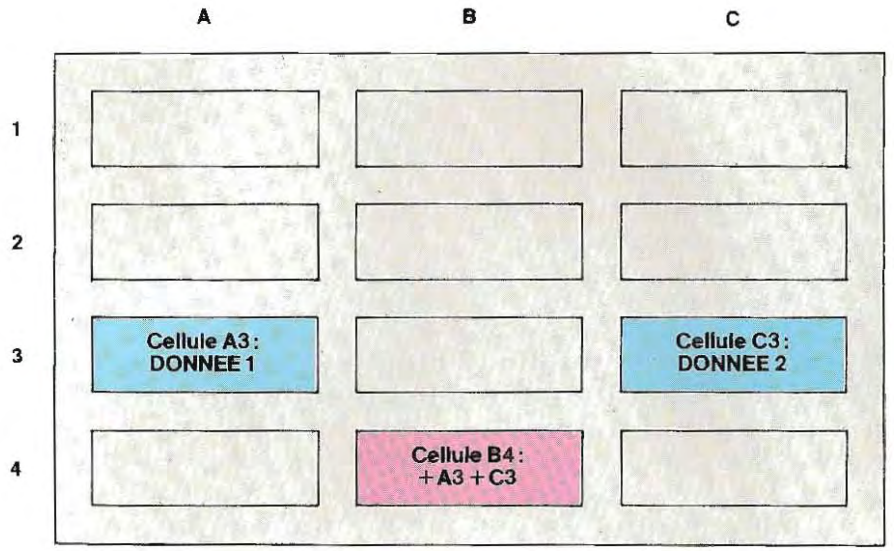
Le tableur

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.

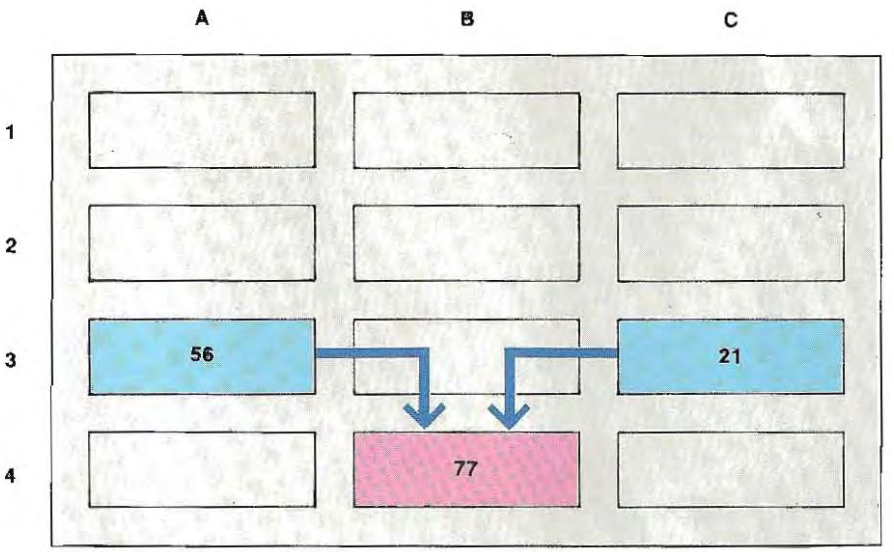
Utilisation de l'ordinateur aux Olympiades de Moscou de 1980.

M. Vucela/Marka

CALCUL D'UNE SOMME PAR TABLEUR

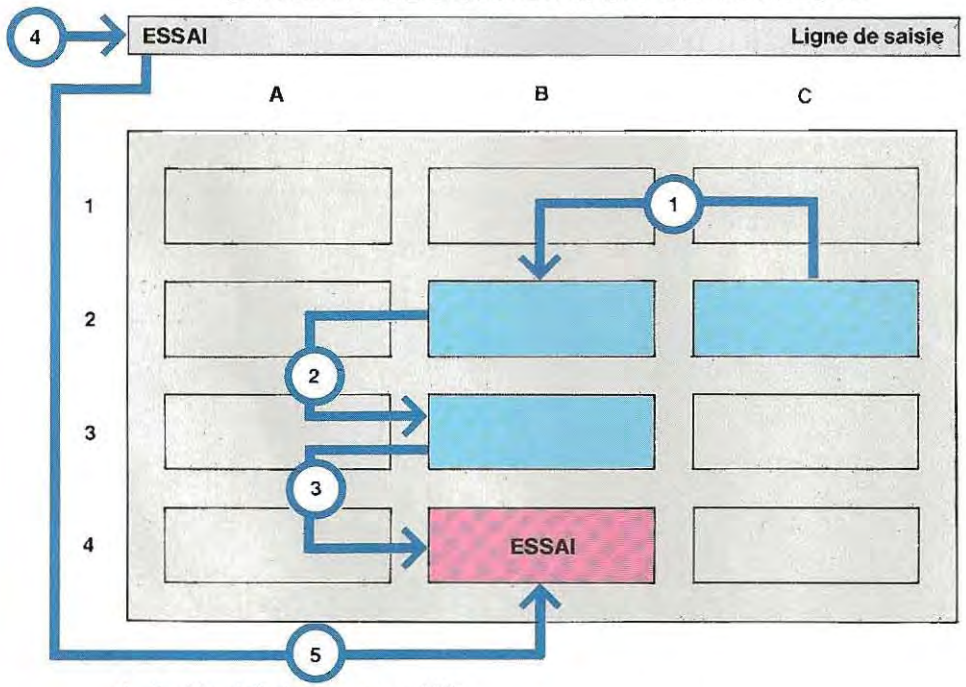


La cellule B4 contient l'indication du calcul à exécuter :
+A3 + C3 signifie « afficher la somme des données contenues dans les cellules A3 et C3 ». Après la saisie de deux valeurs numériques dans les cellules A3 et C3, on obtient le calcul automatique et l'affichage de la somme demandée dans la cellule 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

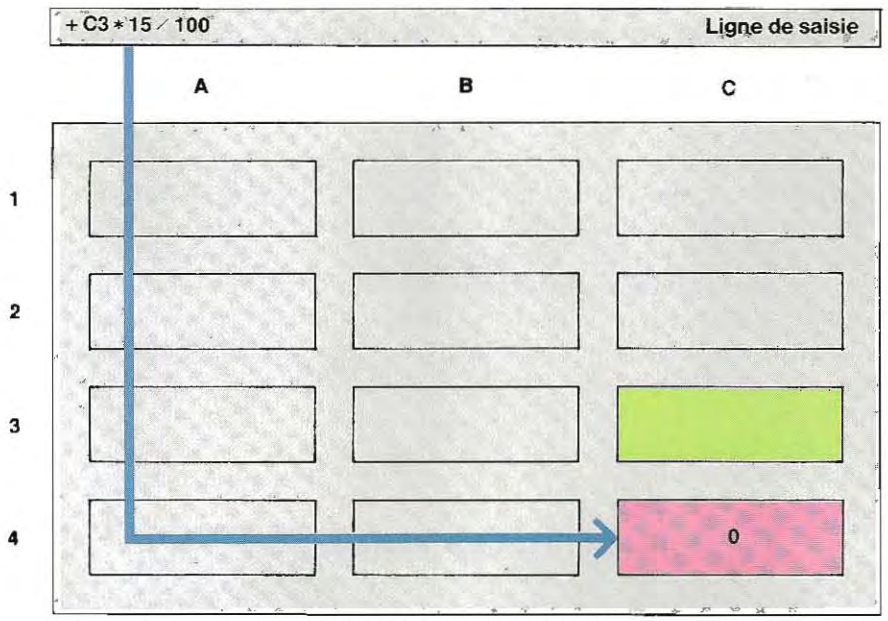


1 Position initiale du curseur (C2).
Premier déplacement : positionnement horizontal (touche ←).
2-3 Après le positionnement horizontal, deux déplacements verticaux consécutifs (espace + ←, espace + ←) amènent le curseur à la position voulue (B4)
4 Saisie de la donnée.
5 Toute donnée saisie est automatiquement transférée dans la cellule pointée par le curseur.

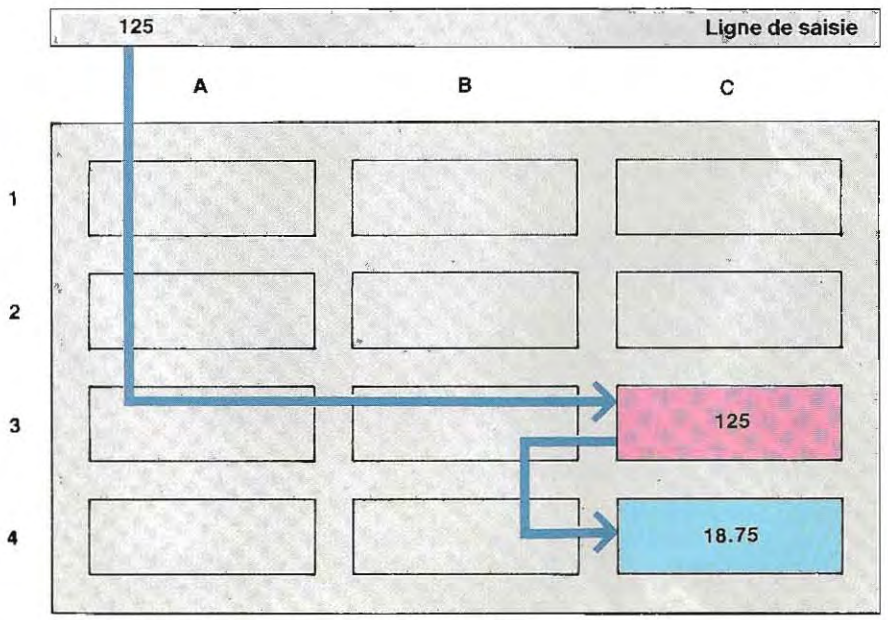
SAISIE D'UNE EXPRESSION

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.

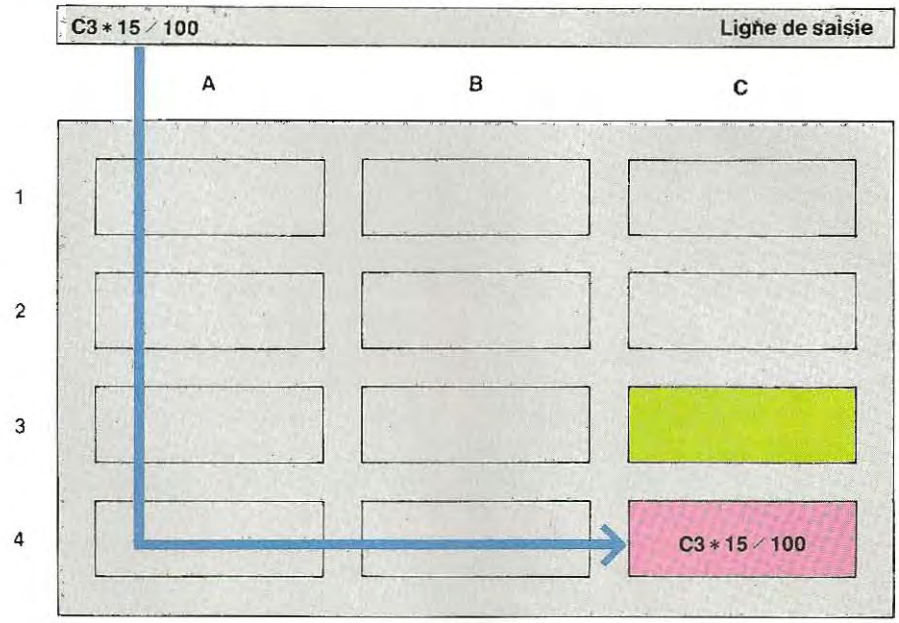


Après la saisie d'une valeur dans la cellule C3, le calcul s'exécute automatiquement et le résultat s'affiche en C4.

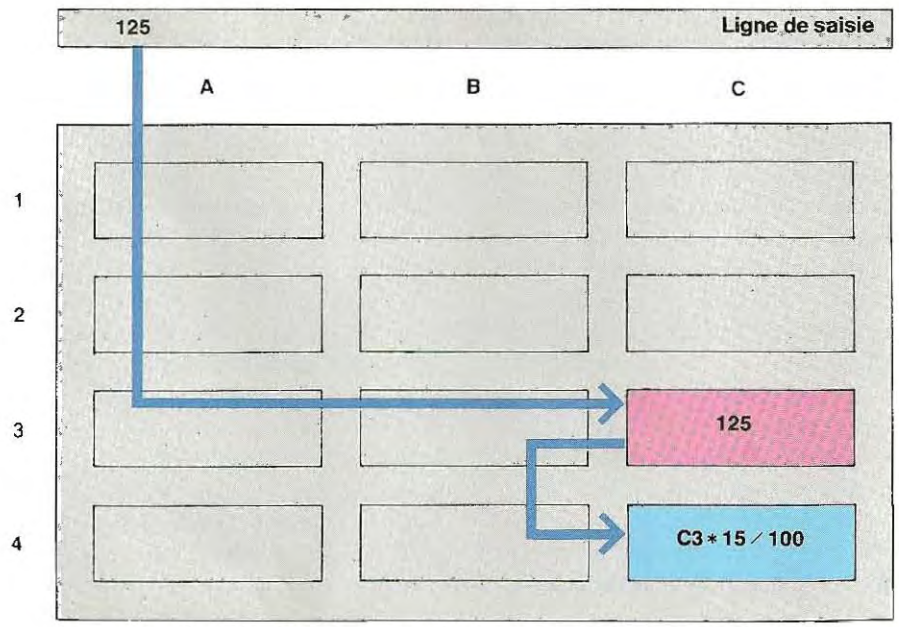
SAISIE D'UNE FORMULE

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.

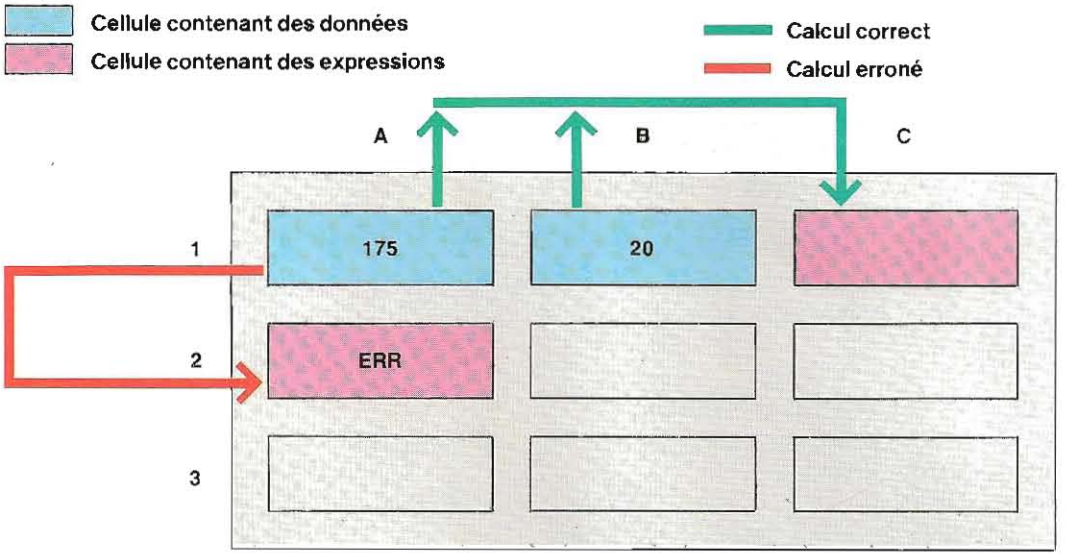
La saisie des données

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.

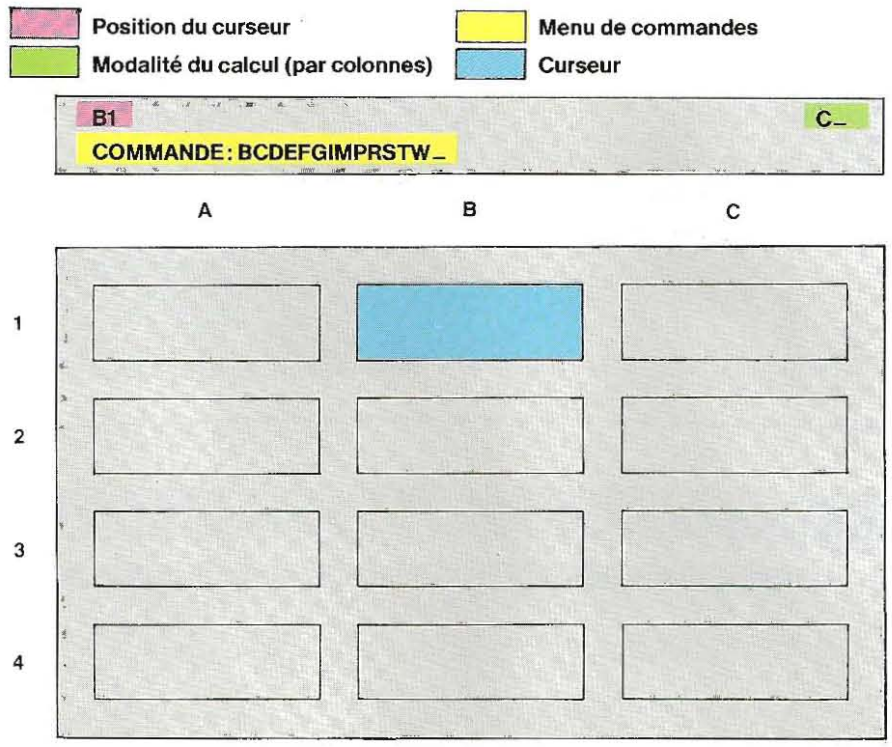
_ REPEAT — Répète le caractère alphanumérique spécifié, sur toute la longueur de la cellule.

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.

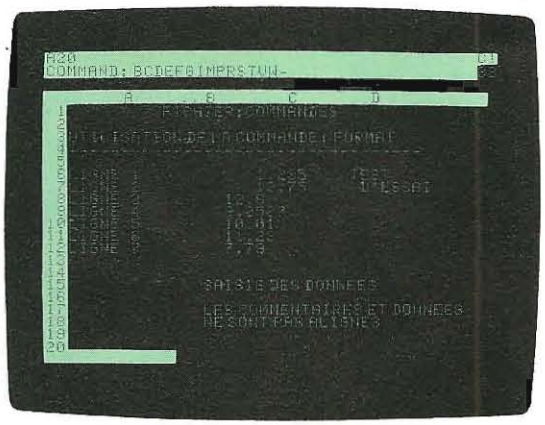
R Permet de choisir le mode de calcul : manuel ou automatique. Dans ce dernier cas, la feuille est recalculée à chaque donnée saisie. En mode manuel, le système ne procède au recalcul qu'après la reconnaissance d'un accord (par exemple, le symbole !). Cette option sert principalement lors d'une saisie 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.

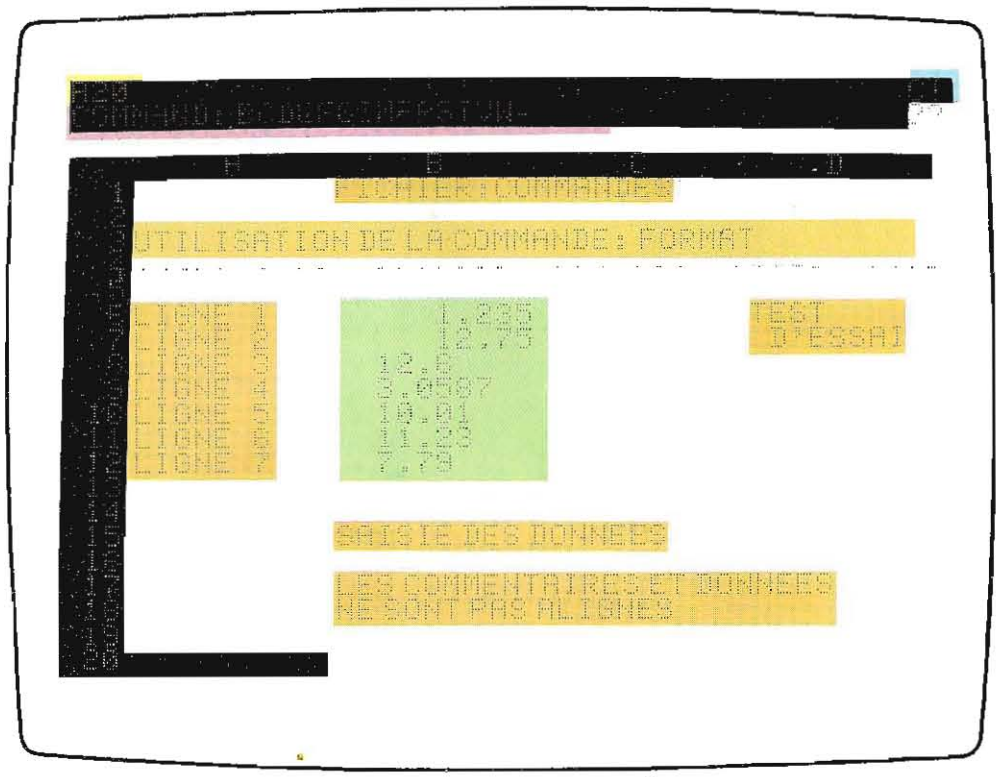
O Ordre du calcul (ligne, colonne). Avec cette option, on définit la modalité selon laquelle se déroulera le calcul.

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).

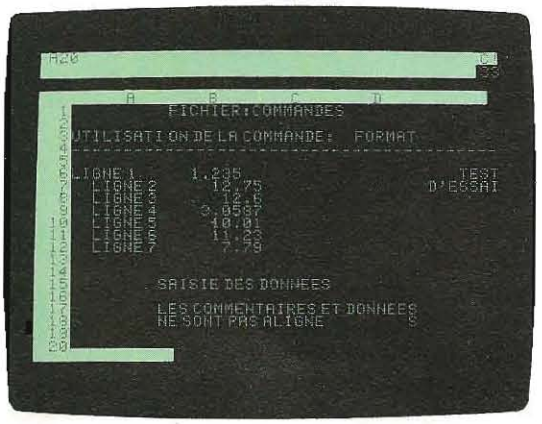
Sur la deuxième ligne, on peut voir le menu des commandes à disposition La commande FORMAT est symbolisée par la lettre F.
Cette ligne n'est affichée que sur demande de l'utilisateur. Sur la machine particulière considérée, la présentation est déclenchée par la touche ←┘
En mode saisie, la même ligne 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.

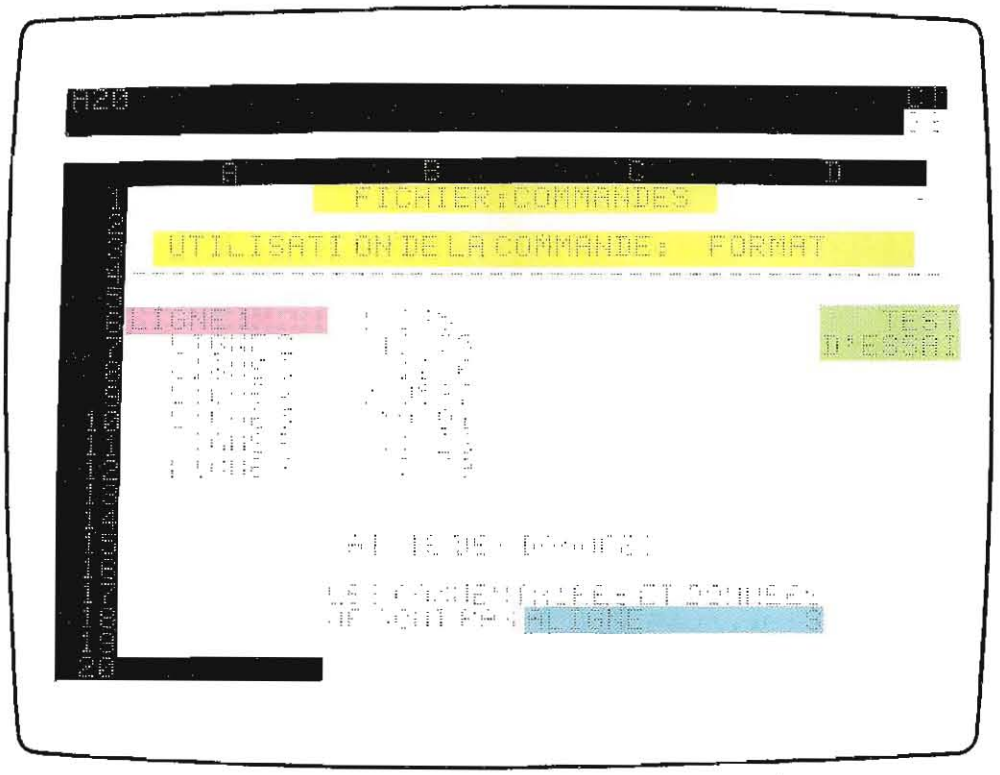
En particulier, les commentaires (titles) ont été placés n'importe où dans la feuille.

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.

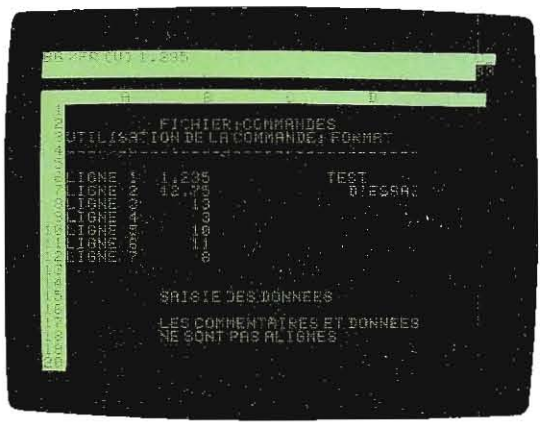
Le même problème se produit pour les champs FICHIER : COMMANDES et UTILISATION.

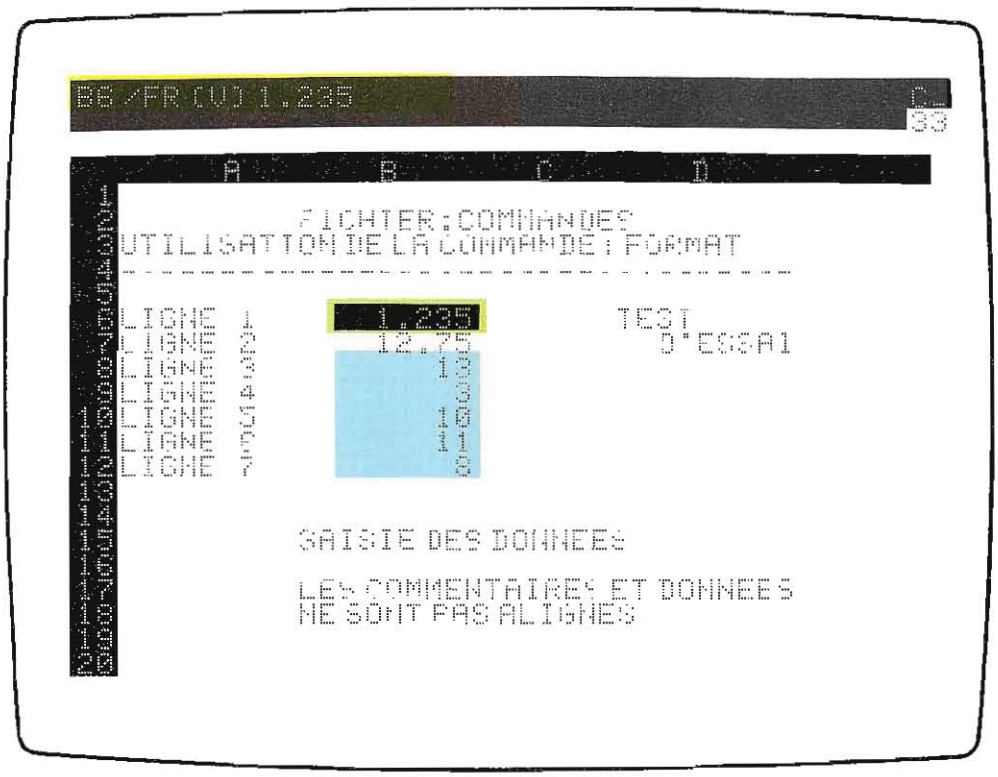
Le champ TEST d'ESSAI, au contraire, formé du contenu de deux cellules différentes mais qui appartiennent à la même colonne ne présente pas ce défaut.

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.

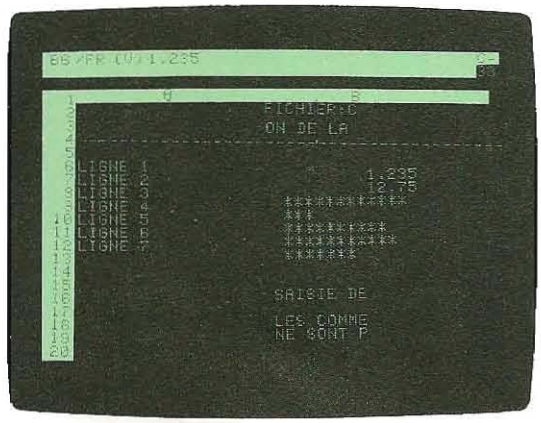
Sur la première ligne de l'écran apparaissent le numéro de la cellule sélectionnée (B6), son contenu (1.235) et les spécifications imposées [/FR et (V)]. Ici, la cellule B6 possède un format de cadrage à droite (/pour commande, F comme format, R comme right = droite) et contient une donnée [(V) = value = 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.

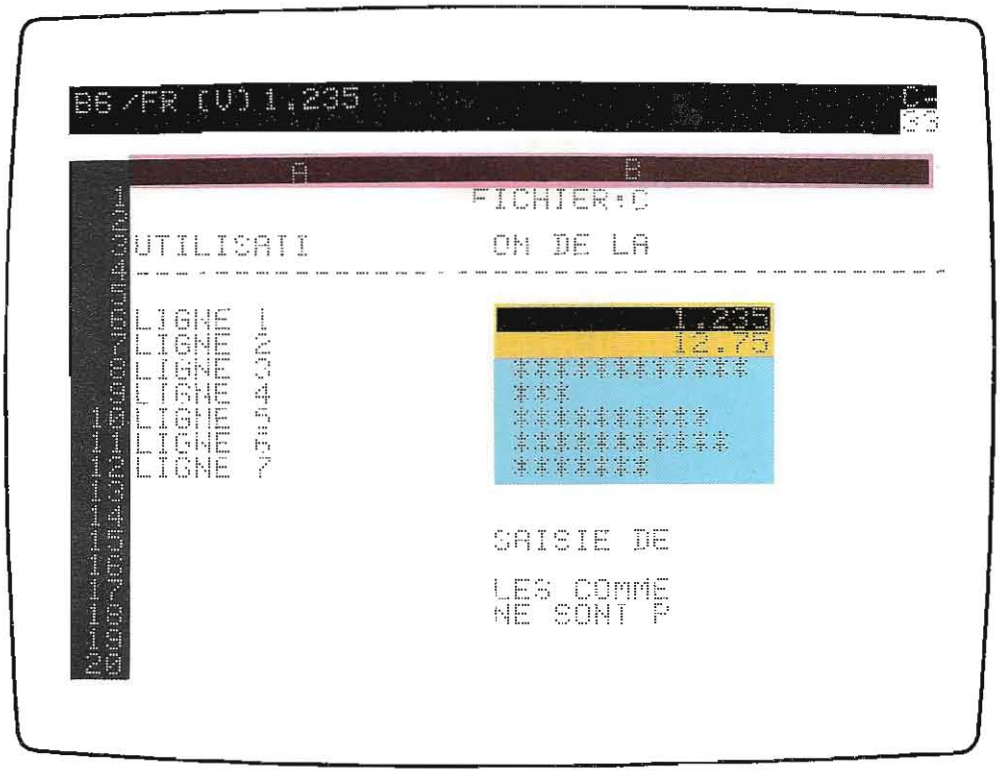
Les données sont celles de la phase de saisie (voir schéma précédent), mais après la commande G/I (GLOBAL/INTEGER), par laquelle on impose à toutes les cellules le format entier. Les valeurs ont été automatiquement arrondies à la valeur la plus proche et présentées sans décimales. Font exception les 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.

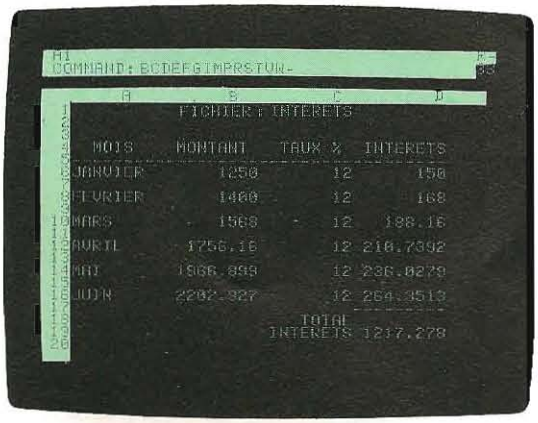
Dans chaque cellule apparaît une suite d'astérisques, proportionnelle à la valeur numérique qui s'y trouve (arrondie à la valeur inférieure).

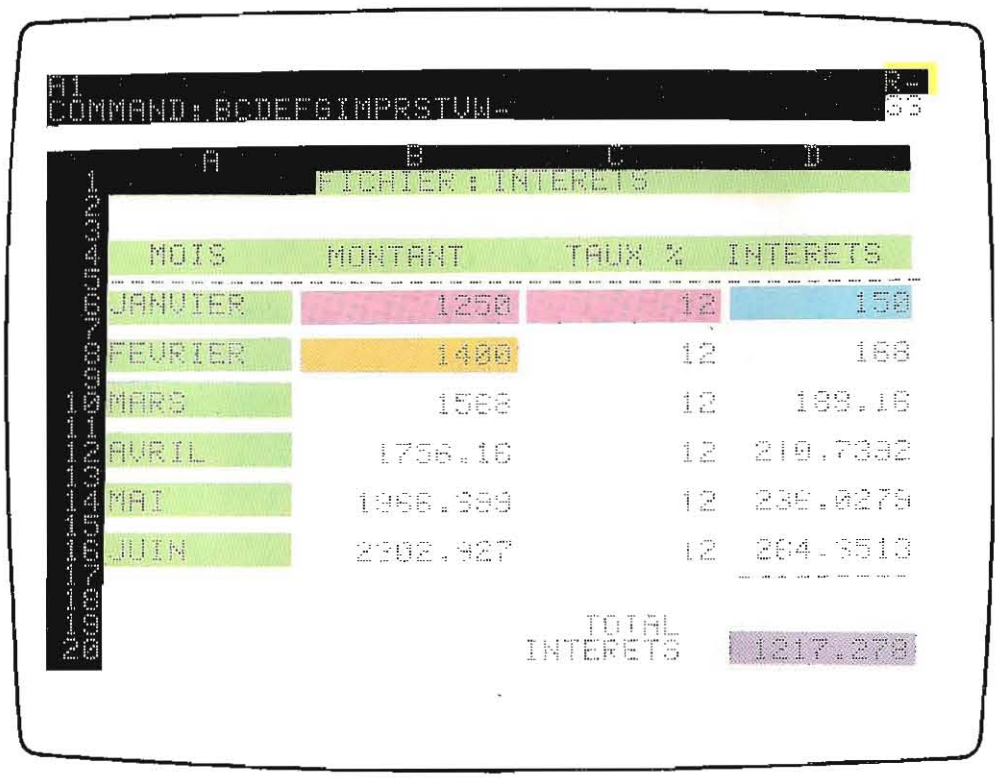
Les cellules B6 et B7 sont exclues du fait des spécifications précédentes de format « local »

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).

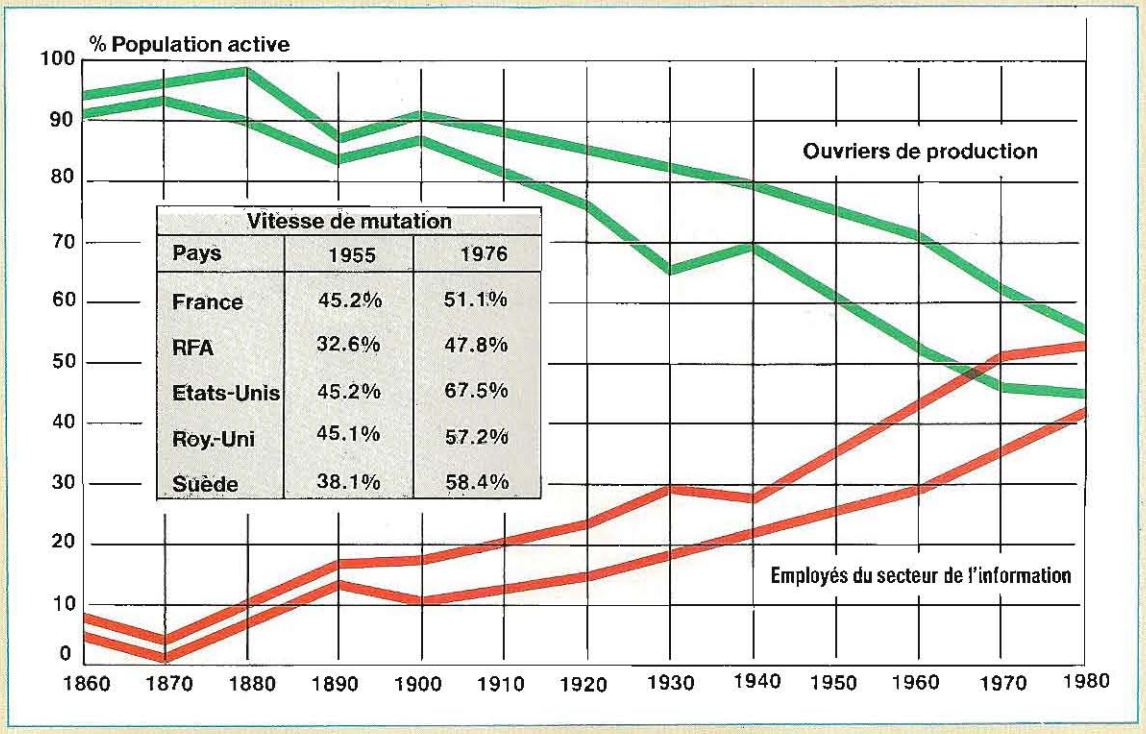
Nous avons déjà évoqué l'importance dominante que revêt l'**information** dans le monde industriel avancé.

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.

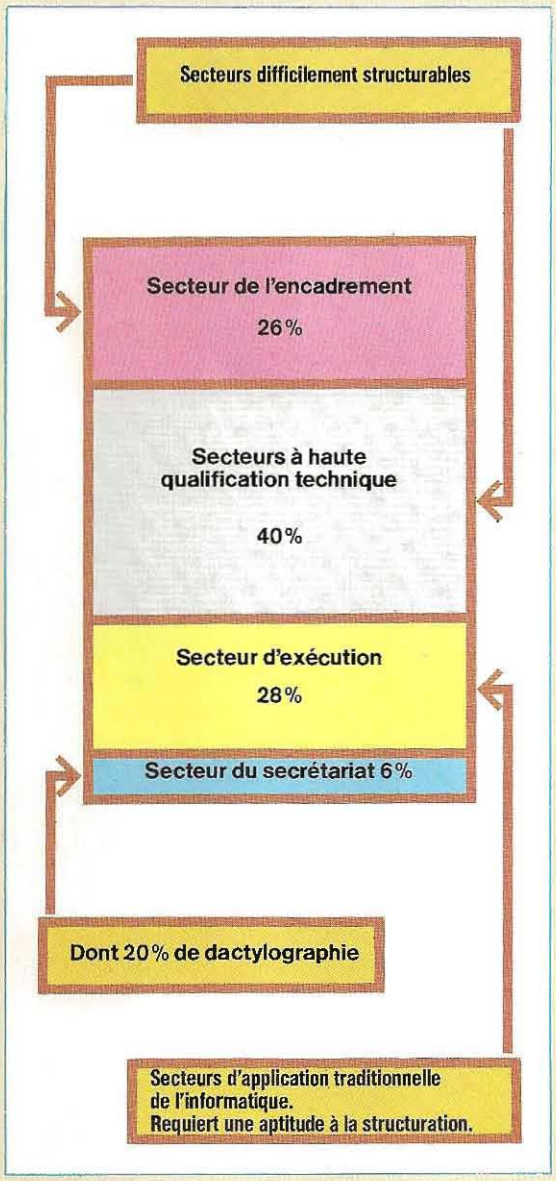
Une telle étude ne peut prendre en compte que des performances atteintes, c'est-à-dire le

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

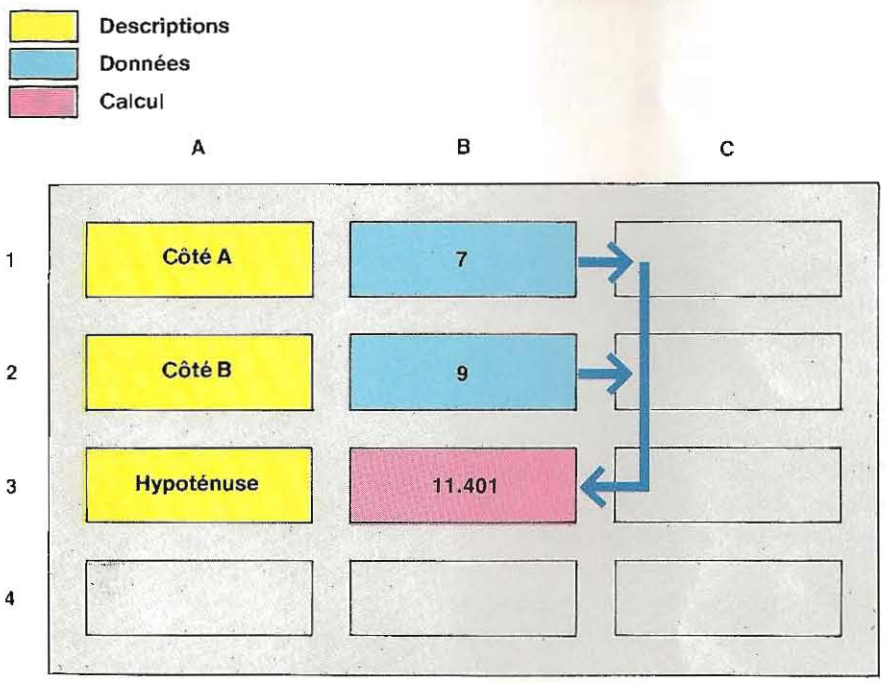
L'application majeure du tableur concerne les calculs économiques ou financiers (prévisions budgétaires, calculs d'intérêts, plans d'investissement, modélisation).
Pour répondre à ces problèmes particuliers, on a introduit une série de fonctions qui n'existent pas en Basic (à moins de les définir comme fonctions-utilisateur).
Les principales sont répertoriées ci-dessous :

AVERAGE (n1, n2,...). Calcule la moyenne arithmétique des contenus des cellules spécifiées comme paramètres. Par exemple :

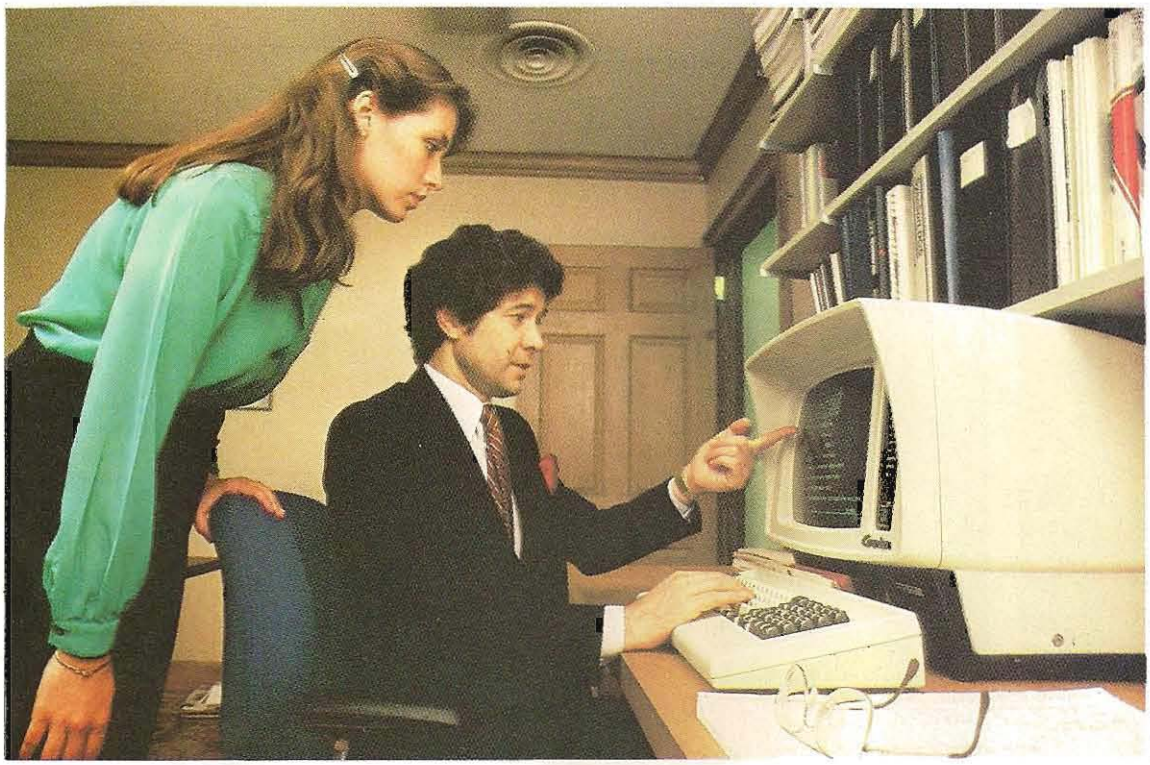
AVERAGE (+B1, +B3, +C6)

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).

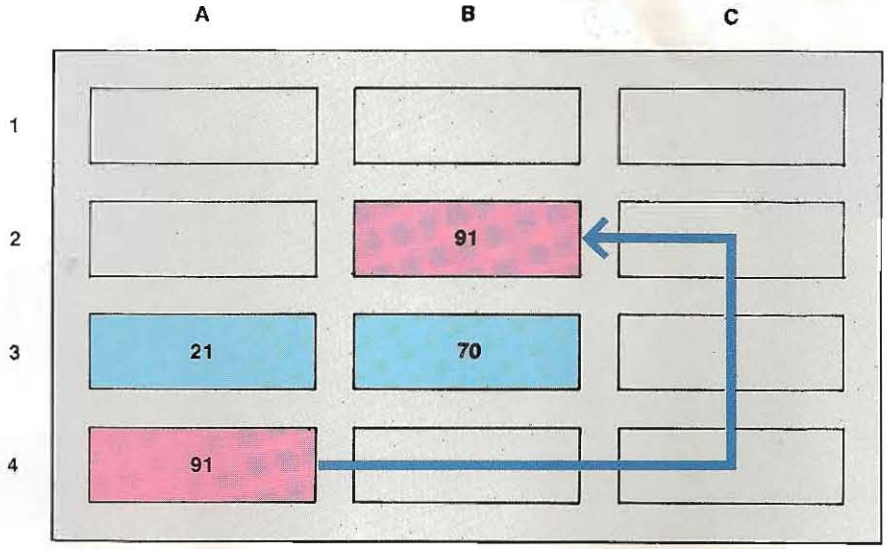
ADRESSAGE ABSOLU ET RELATIF

Cellule contenant l'expression

Cellule contenant des données non valides

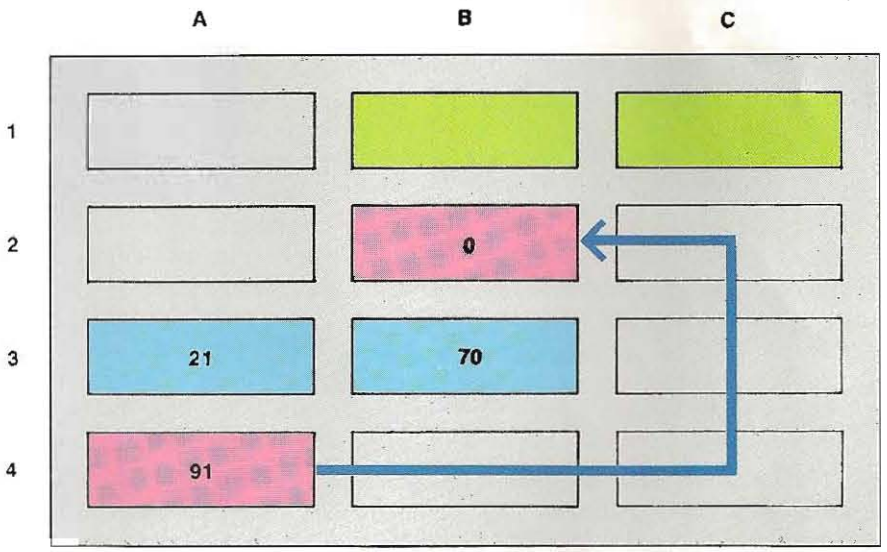
Cellule contenant des données valides

Adressage absolu



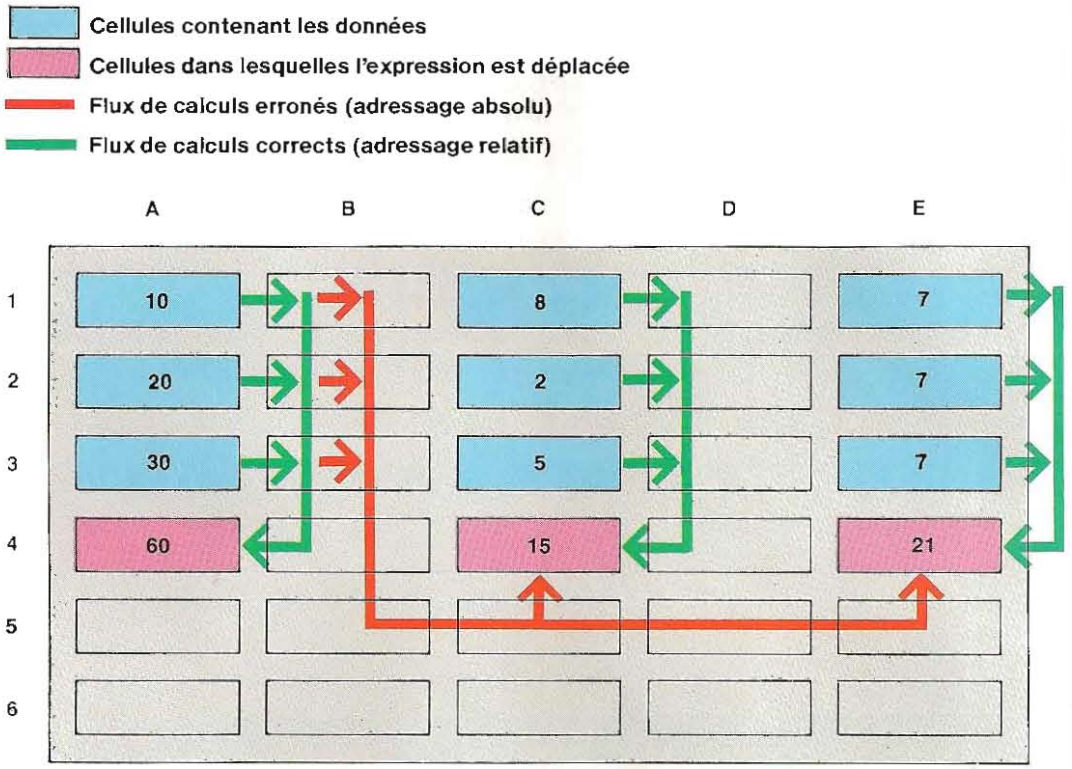
La cellule A4 contient l'expression + A3 + B3 et donne comme résultat la somme des contenus des cellules A3 et B3.
Si l'on déplace l'expression dans toute autre cellule, le résultat reste inchangé, étant donné que les coordonnées des cellules auxquelles le calcul se réfère restent les mêmes.

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.

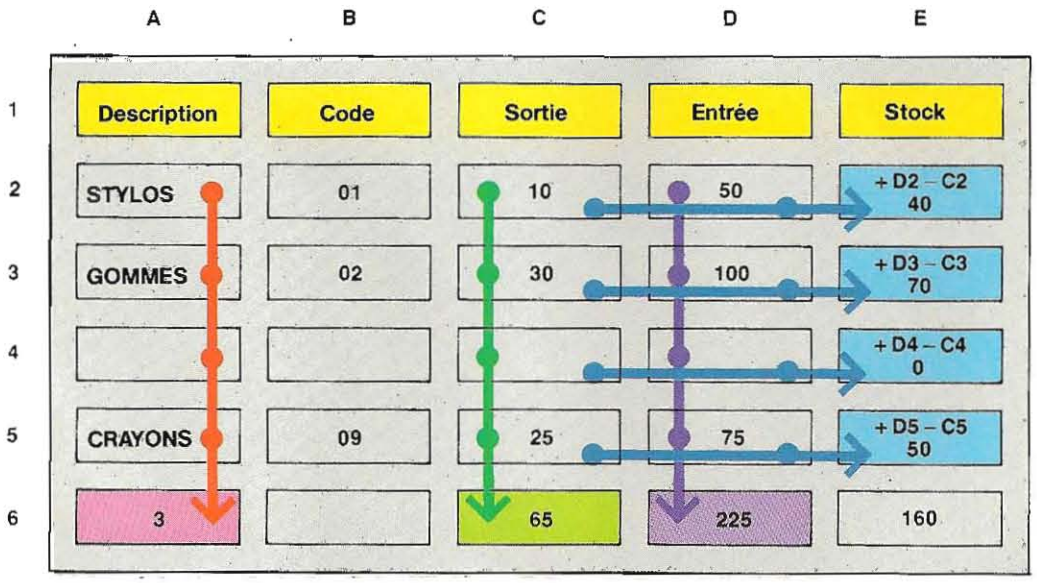
NPV (S, NS..NE). NPV signifie Net Present Value. On l'utilise pour calculer (à partir d'un résultat final et d'un taux d'intérêt fixé à l'avance) le montant du capital initial exigé par la transaction. Le paramètre S est le taux d'intérêt, les valeurs NS..NE sont les chiffres que l'on veut obtenir à 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.

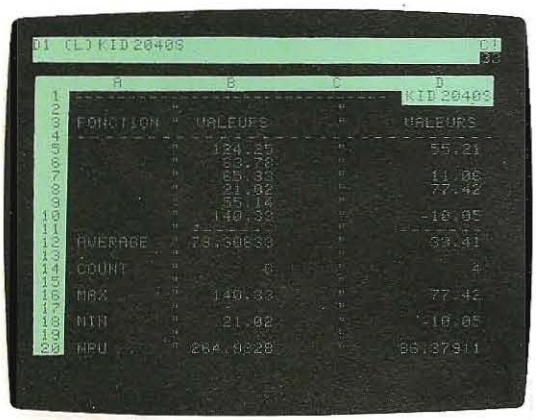
LES FONCTIONS COUNT ET SUM DANS LA GESTION D'UN MAGASIN

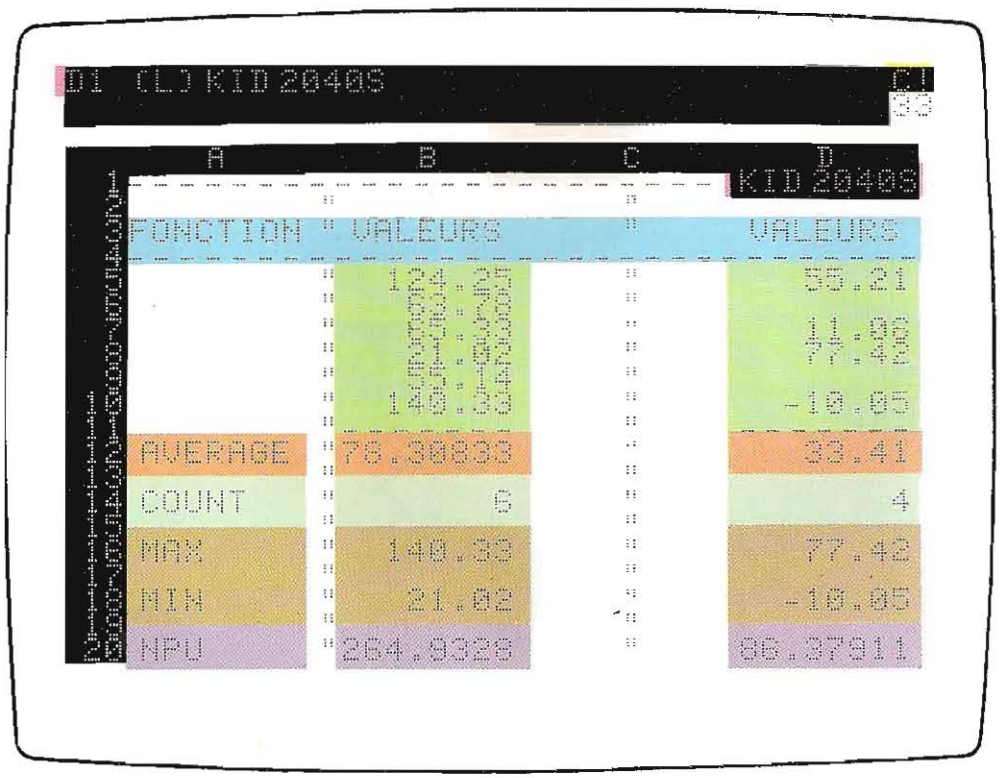


En mettant dans la cellule...	la fonction...	il apparaît...
A6	@ COUNT (A2..A5)	le numéro des différents articles présents dans le magasin
C6	@ SUM (C2. C5)	le total sorti
D6	@ SUM (D2..D5)	le total entré
E6	@ SUM (E2..E5)	le total en stock

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)

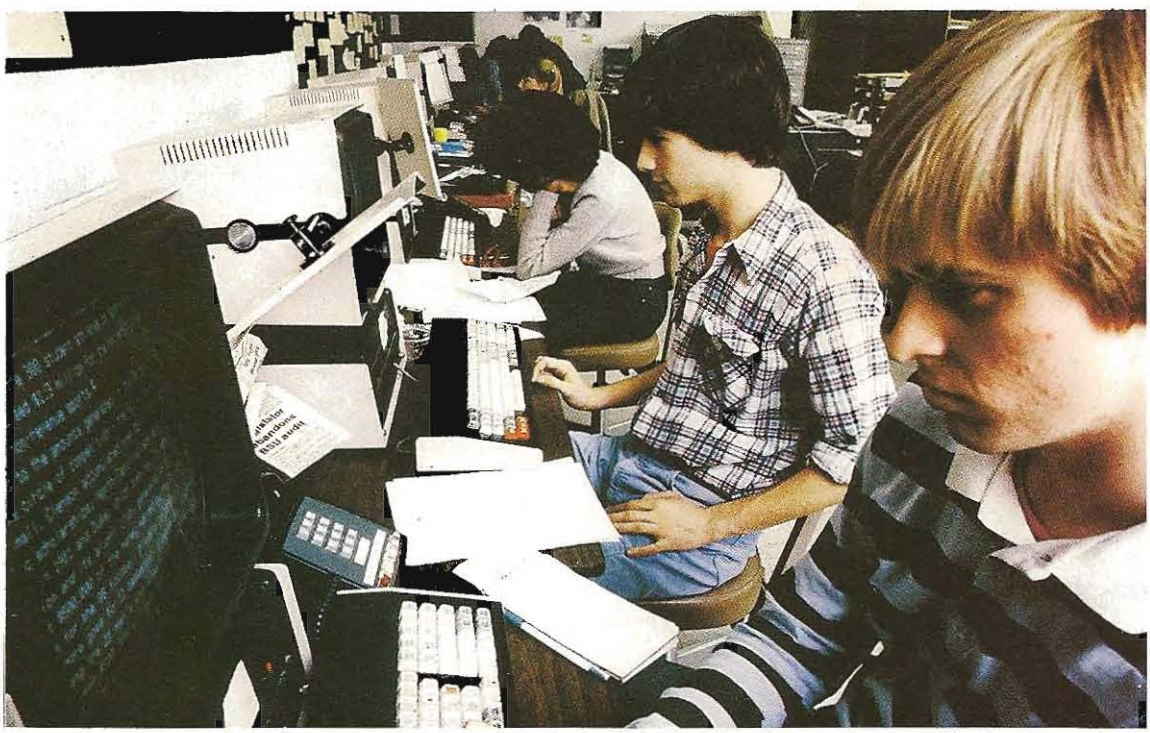
on obtient le contrôle simultané des conditions A7 = TRUE et A9 = FALSE qui, à leur tour, peuvent dépendre d'autres conditions mises en places respectivement en A7 et A9. Ci-contre, un exemple d'application de la fonction AND.

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.

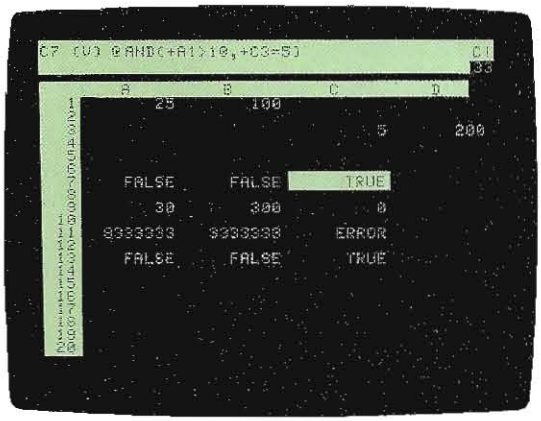
Un centre de traitement de l'université de Maryland aux Etats-Unis.

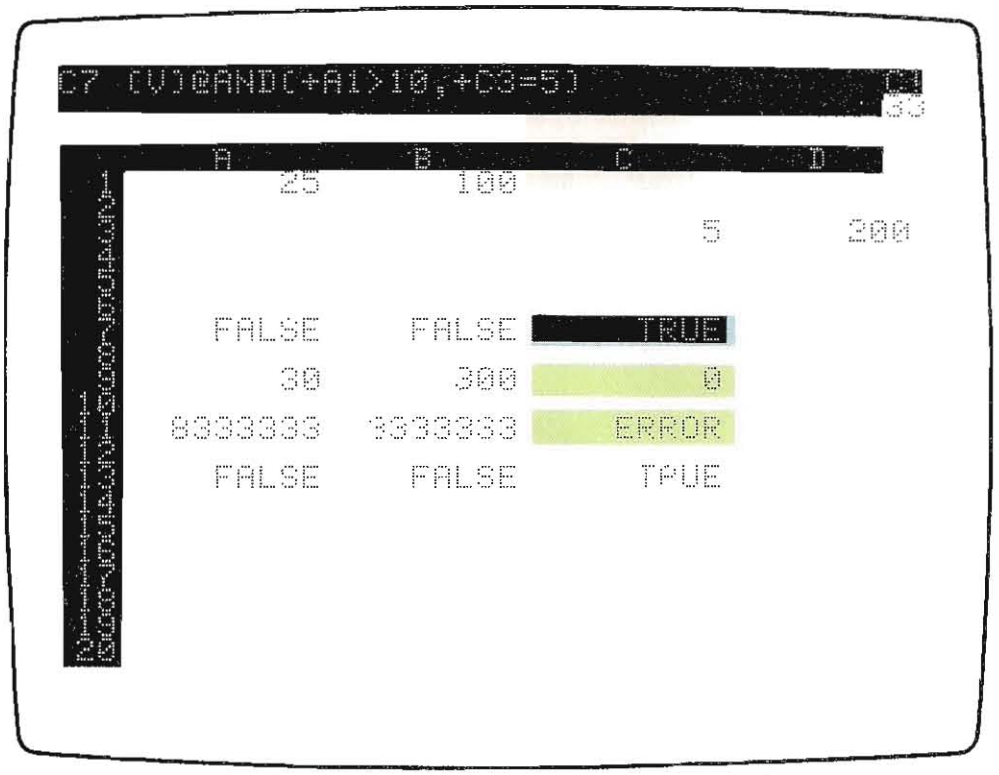


J. Pickerell/Marka

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...)

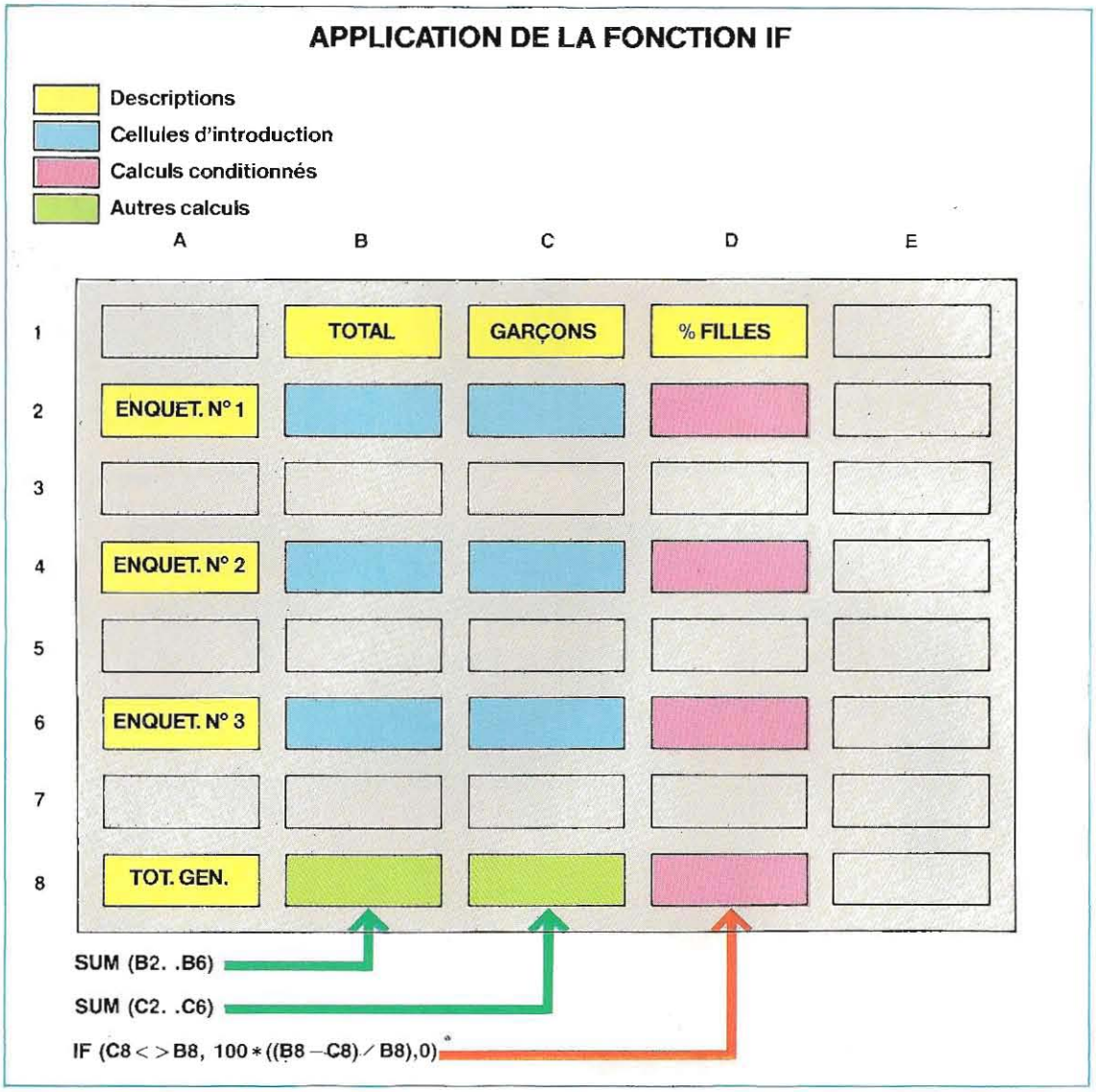
le contenu de B1 est TRUE jusqu'à ce que NA apparaisse dans la case A1. En introduisant une donnée quelconque en A1, FALSE est transféré en B1, et l'on effectue les calculs dans toutes les cellules comportant cet indicateur.

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 $\pi = 3{,}14159265336$.

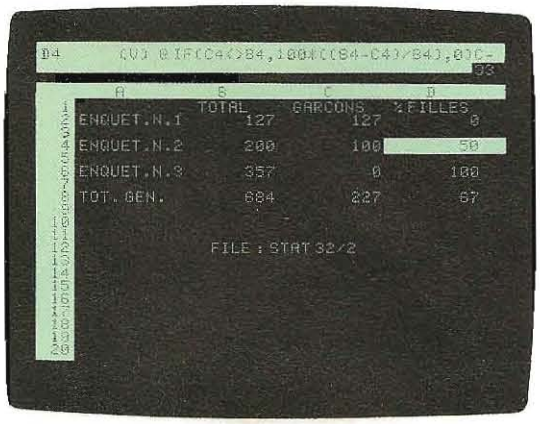
ERROR. Apparaît comme un message, dans toute cellule se rapportant à cette fonction.

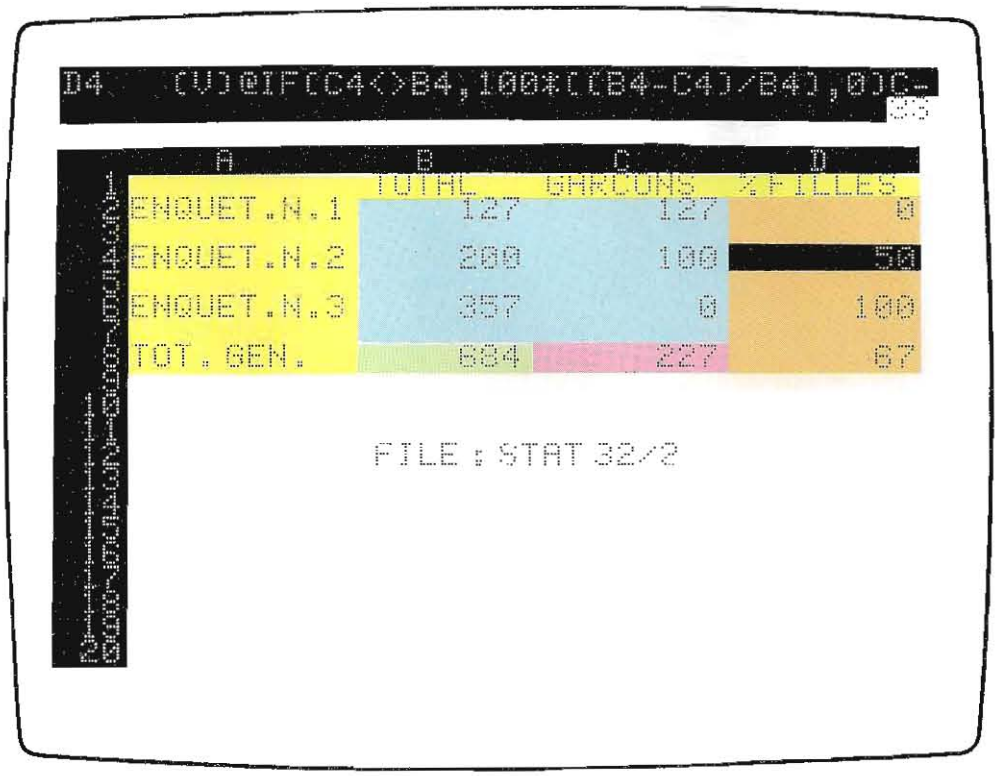
FALSE/TRUE. Fonctions qui affichent les indicateurs correspondants.

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)

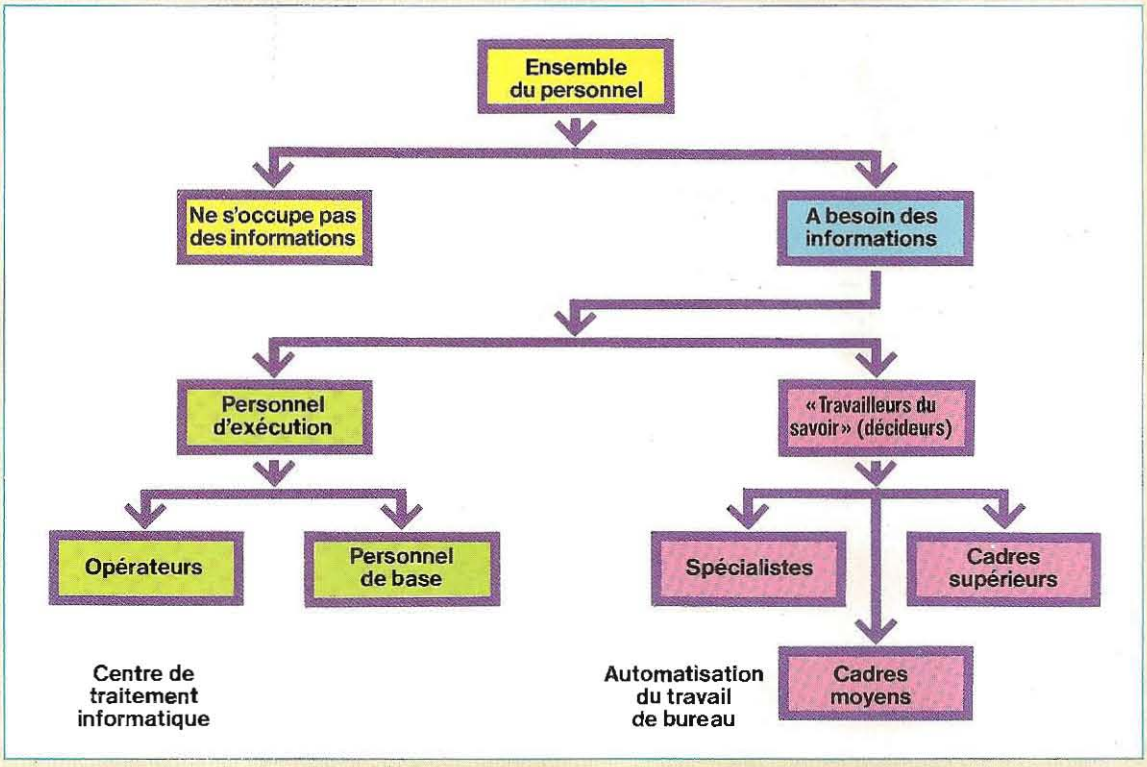
A la lumière de l'analyse menée jusqu'ici, il apparaît clairement que, par automatisation du travail de bureau, on entend :

- 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.

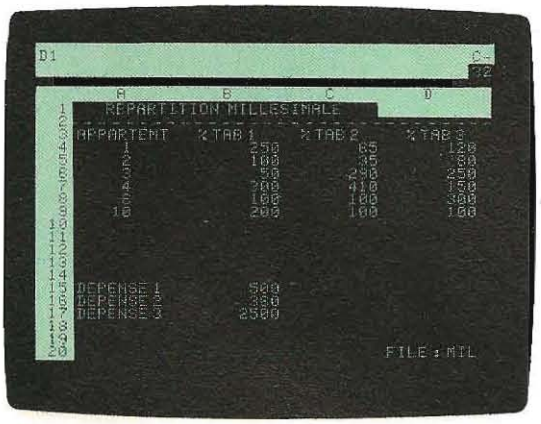
Deux cas peuvent se rencontrer : soit le programme ne transfère rien, dans la mesure où il reconnaît la présence d'expressions, soit il en effectue le transfert. En entrant cette colonne dans un nouveau tableau, des valeurs erronées seront affichées puisque les expressions seront appliquées d'une manière 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.

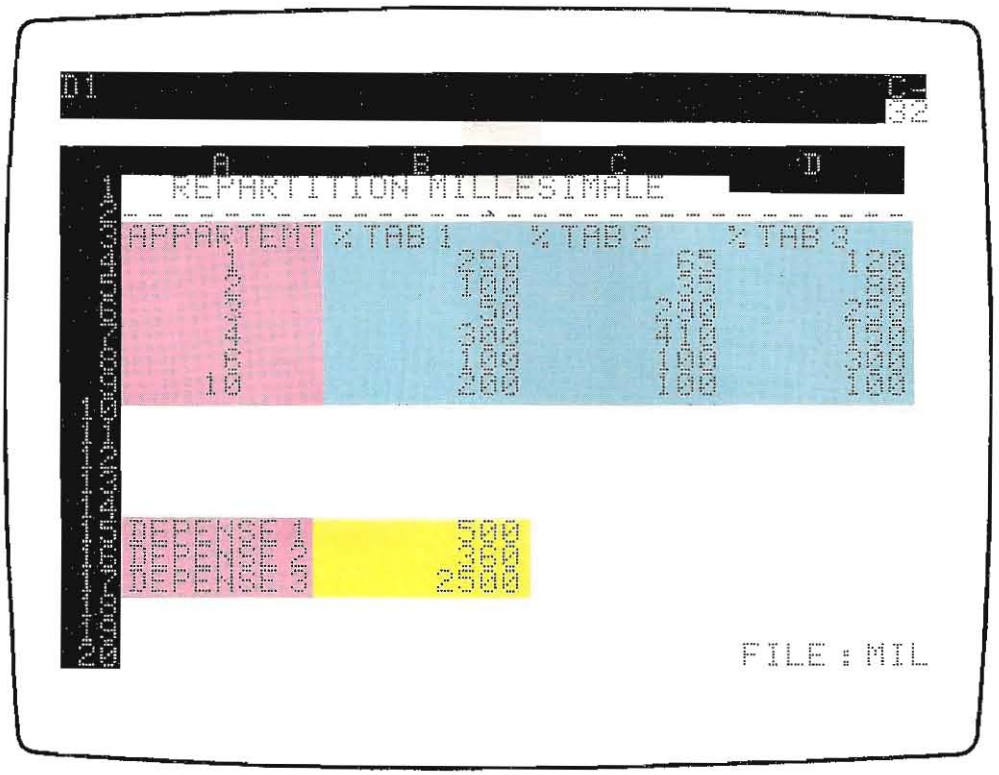
Gestion d'un magasin

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.

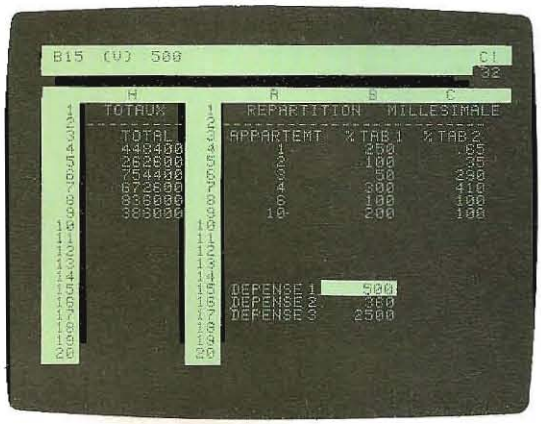
Les colonnes B, C, D renferment les millièmes exprimés en pour mille. Dans l'illustration, on a adopté le symbole % (pour cent), car le symbole exact (‰) ne figure pas sur le clavier ASCII du standard américain.
Chacune des 3 colonnes renferme une répartition différente, liée à un indice de dépense particulier. Les millièmes ne varient pas et doivent être entrés une fois pour toutes.

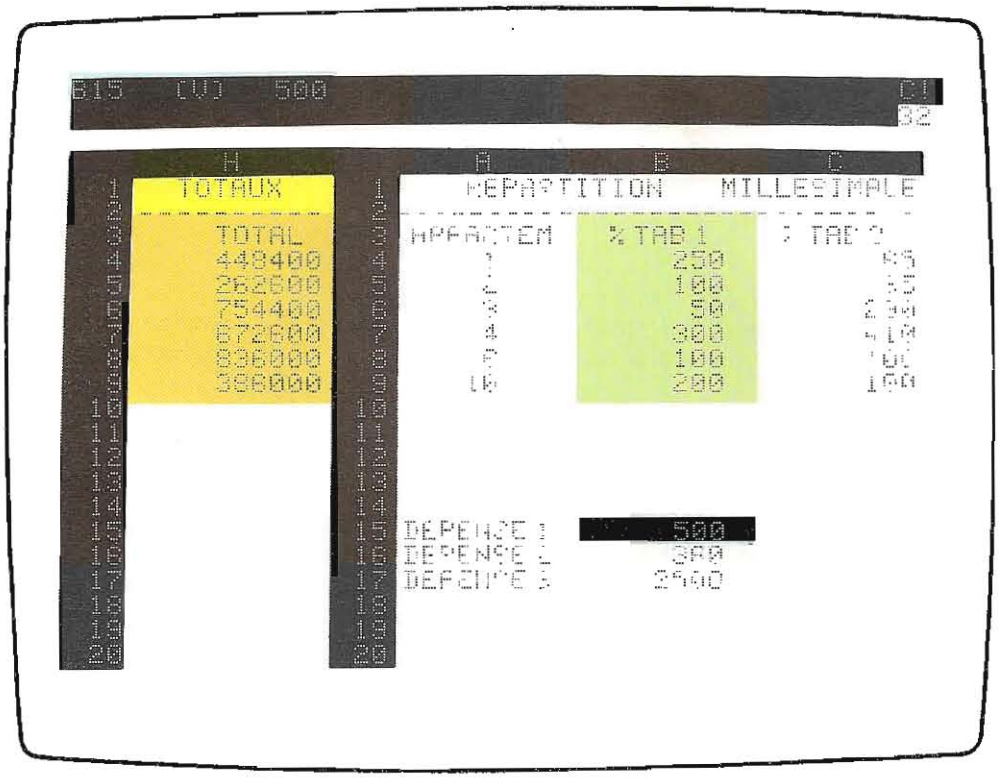
Les cellules B15, B16, B17 sont réservées à l'entrée des sommes à répartir. Après l'entrée d'une valeur, les répartitions sont calculées d'après les millièmes correspondants.

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.

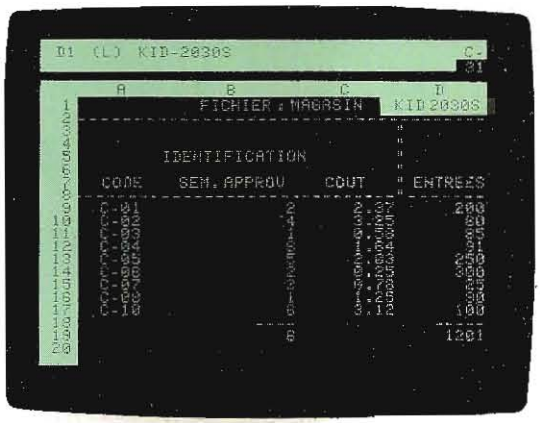
On trouvera un exemple d'application pages 836 à 840.
Pour garder un contact permanent avec l'écran, on a préféré illustrer l'exemple (identification, mouvements, coûts de production...) avec une séquence d'écrans. Mais cette sortie n'est pas la seule possible : on peut faire aussi des copies papier 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.

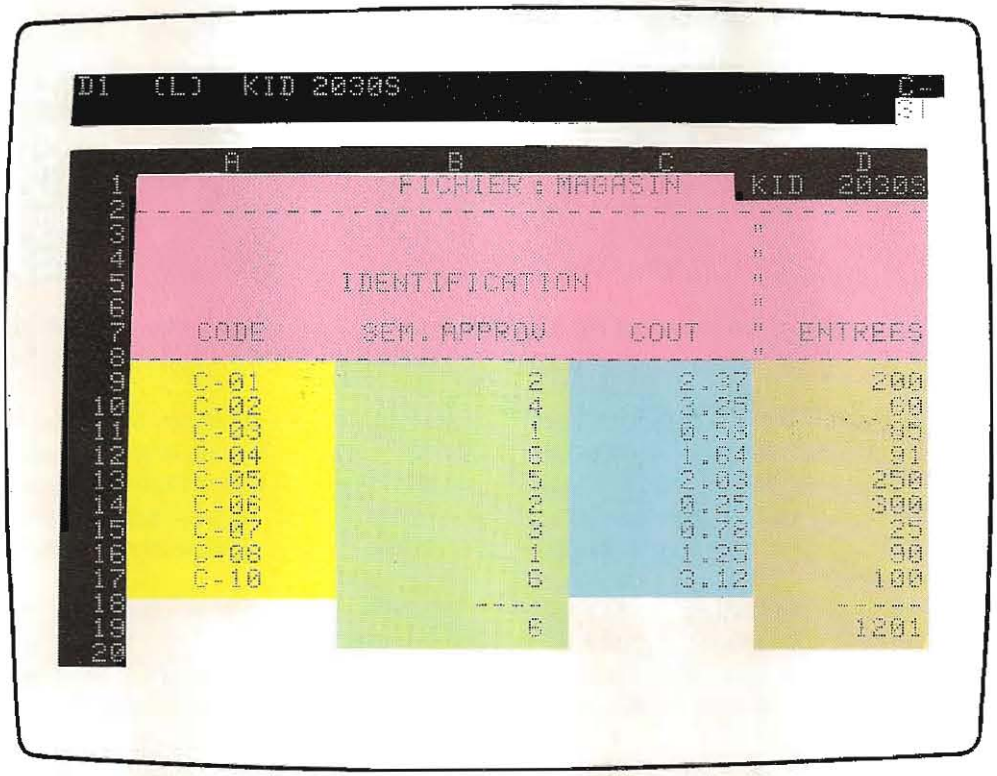
Analyse des variations (WHAT IF)

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.

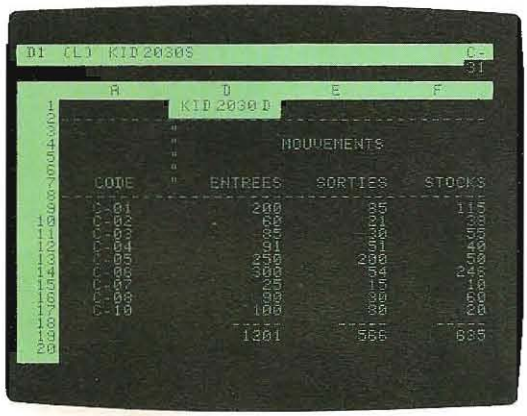
Dans la colonne B sont affichées les périodes d'approvisionnement (en semaine) de chaque article. La cellule B19 contient la fonction @ MAX (B9...B17) qui calcule le maximum des valeurs contenues dans la colonne B. La période d'approvisionnement de 6 semaines est commune aux deux articles C-04 et C-10.

Dans la colonne C, on a entré les coûts des articles.

La colonne D fait partie de la zone suivante du tableau (MOUVEMENTS).

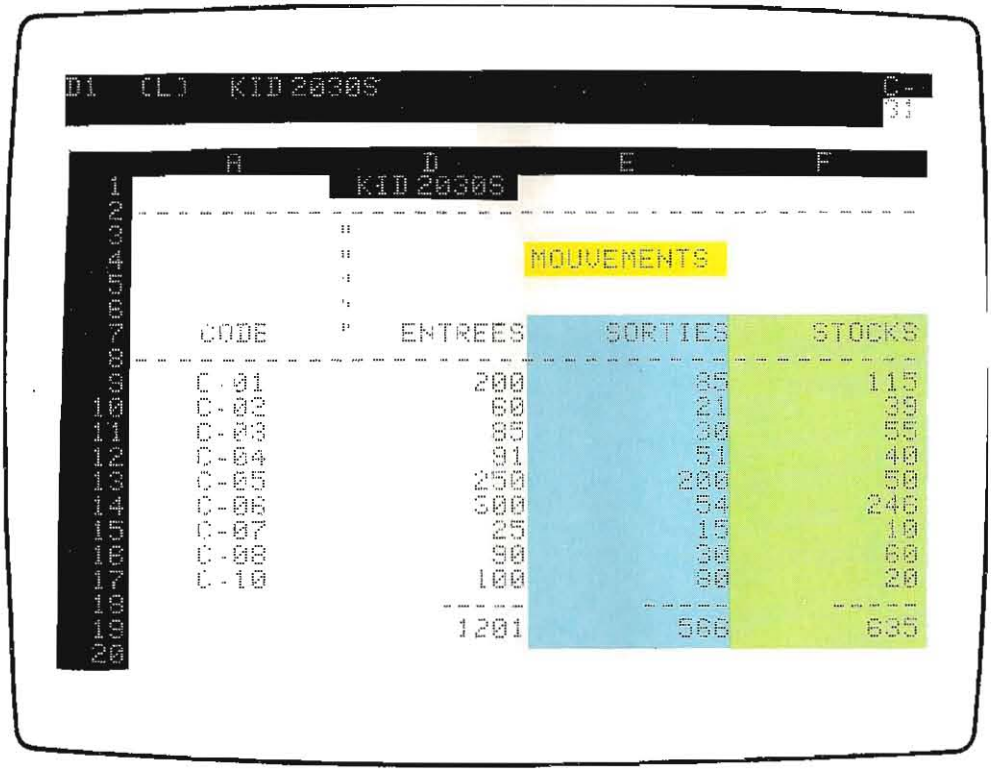
GESTION D'UN MAGASIN : MOUVEMENTS

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

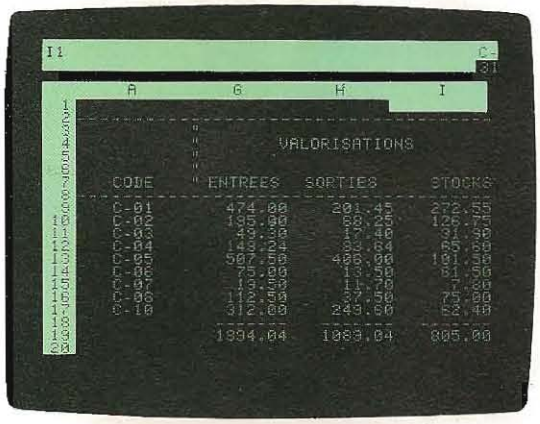
La partie mouvements occupe une zone de tableau placée loin de la colonne A, qui contient les codes articles. Pour afficher les mouvements, il est nécessaire de définir la colonne A (TITRE/TITLE) verticale.

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.

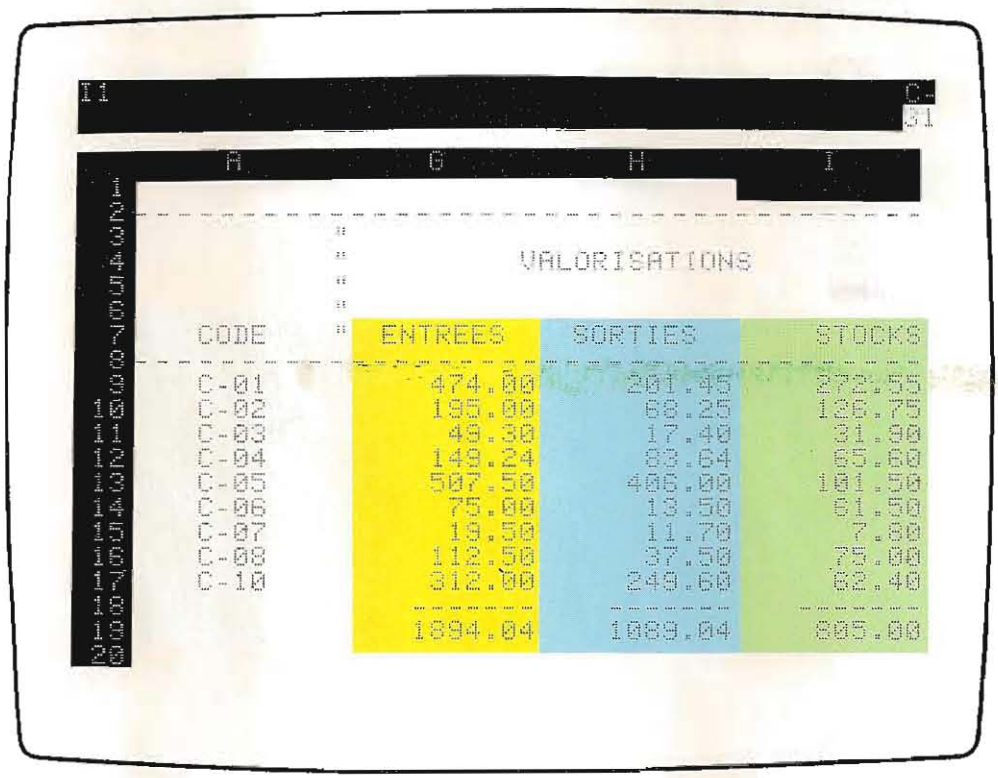
GESTION D'UN MAGASIN : VALORISATIONS

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.



I1 C-
31

	A	G	H	I
1				
2				
3				
4		VALORISATIONS		
5				
6	CODE	ENTREES	SORTIES	STOCKS
7	C-01	474.00	201.45	272.55
8	C-02	195.00	68.25	126.75
9	C-03	49.30	17.40	31.90
10	C-04	149.24	83.64	65.60
11	C-05	507.50	406.00	101.50
12	C-06	75.00	13.50	61.50
13	C-07	19.50	11.70	7.80
14	C-08	112.50	37.50	75.00
15	C-10	312.00	249.60	62.40
16				
17		1894.04	1089.04	805.00
18				
19				
20				



I1 C-
31

	A	G	H	I
1				
2				
3				
4		VALORISATIONS		
5				
6	CODE	ENTREES	SORTIES	STOCKS
7	C-01	474.00	201.45	272.55
8	C-02	195.00	68.25	126.75
9	C-03	49.30	17.40	31.90
10	C-04	149.24	83.64	65.60
11	C-05	507.50	406.00	101.50
12	C-06	75.00	13.50	61.50
13	C-07	19.50	11.70	7.80
14	C-08	112.50	37.50	75.00
15	C-10	312.00	249.60	62.40
16				
17		1894.04	1089.04	805.00
18				
19				
20				

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.

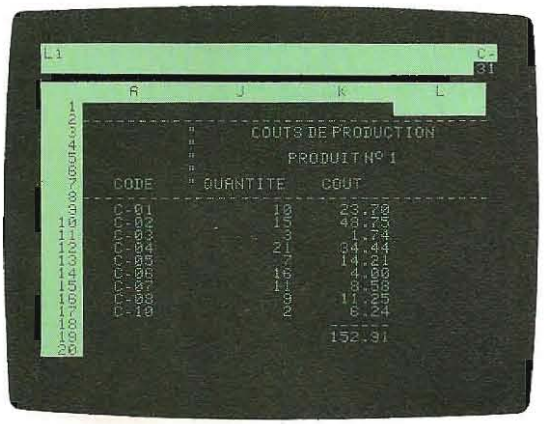
Les valeurs proportionnelles des sorties (colonne H) sont le produit de la quantité en sortie d'un article (mouvements, colonne H) par le coût unitaire (identification, colonne D). La donnée 201.45 en H9 est obtenue par l'expression : +C9*E9.

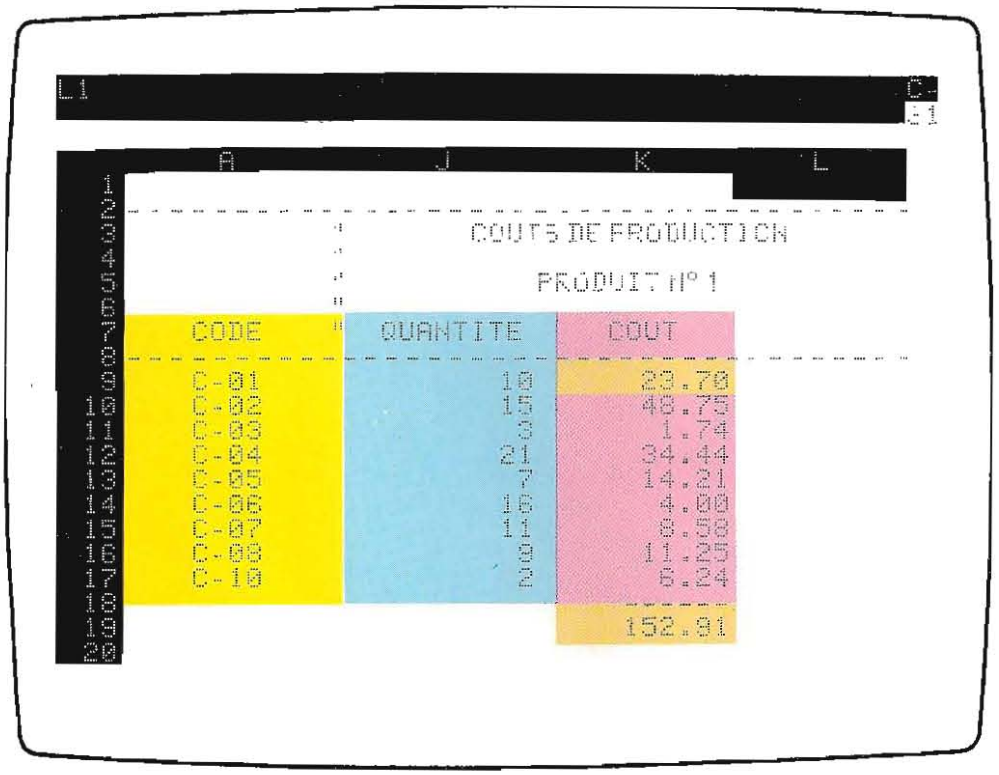
La valorisation du stock s'obtient de façon analogue à l'aide de l'expression : +C9*F9.

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

```
A2:A1                FICHIER PROJECTIONS          A10
-------------------------------------------------------

                 "
                 "  JANV   FEV    MAR    AVR    TOT
-------------------------------------------------------
QUANT. PROD      "   104     86     75    160    425
-------------------------------------------------------
COUTS            "
-------------------------------------------------------
PRODUCTION       "    71     63     65     96    135
DISTRIBUTION     "    10     10      5     16     43
AGIOS            "    15     14     12     17     58
-------------------------------------------------------
COUT TOTAL       "    96     87     82    112    336
-------------------------------------------------------

CH. AFF.         "   198     86     77    152    424
-------------------------------------------------------
RESULTAT         "    19      5     -5     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

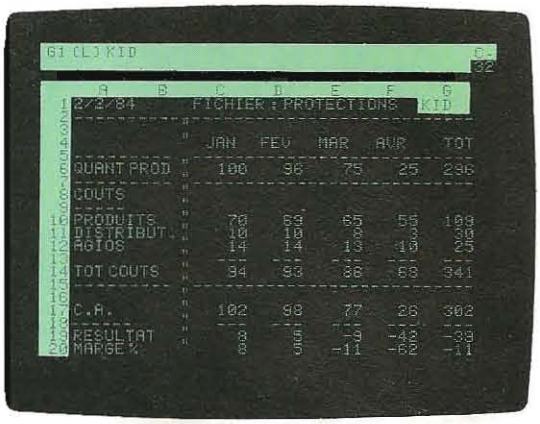
Assemblage de composants électroniques dans une unité de production.

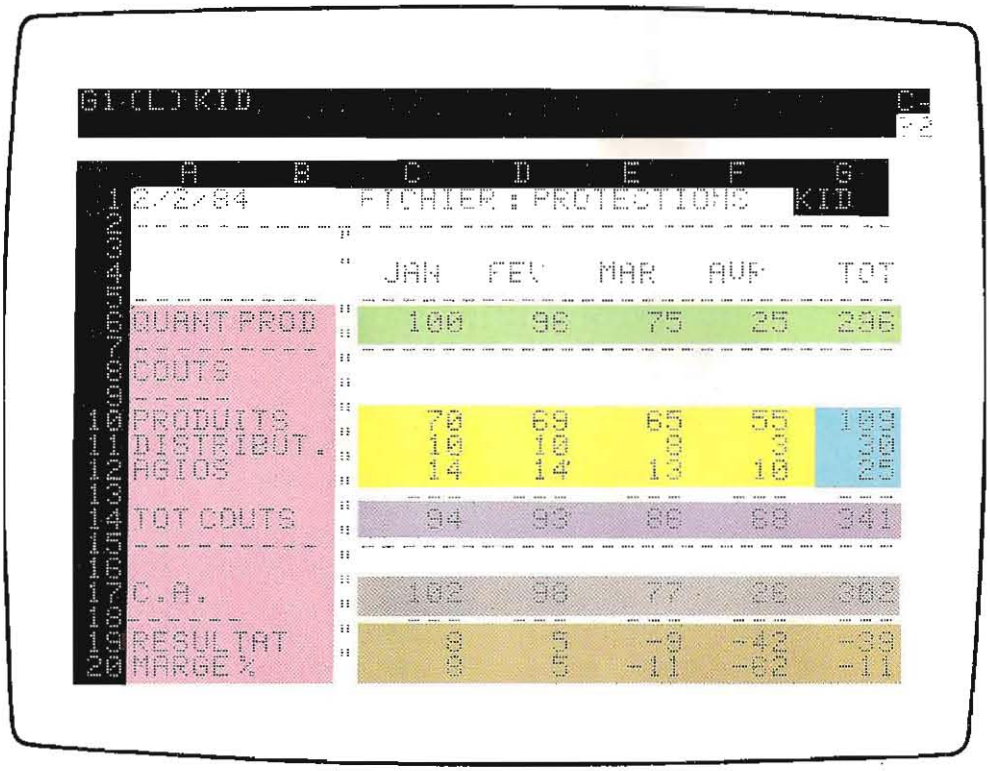
DELTA (). Cette fonction fournit la valeur absolue maximale enregistrée lors des changements de valeurs entre deux itérations successives. On peut donc l'utiliser comme critère de convergence. Ainsi, DELTA () < 0,1 arrête le calcul quand le maximum de variations est inférieur à 0,1.

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

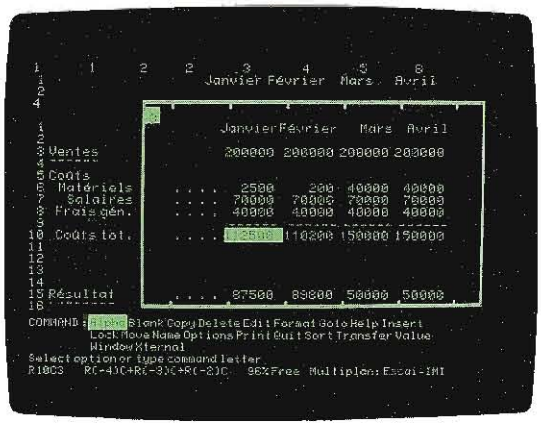
où X est l'unité disque sélectionnée (elle peut être omise, auquel cas le système l'affecte automatiquement par défaut) et NOM le nom du fichier à créer. Après avoir vérifié l'espace disponible sur le disque, le programme sauvegarde le nom dans un dictionnaire et passe à l'introduction des attributs 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 :

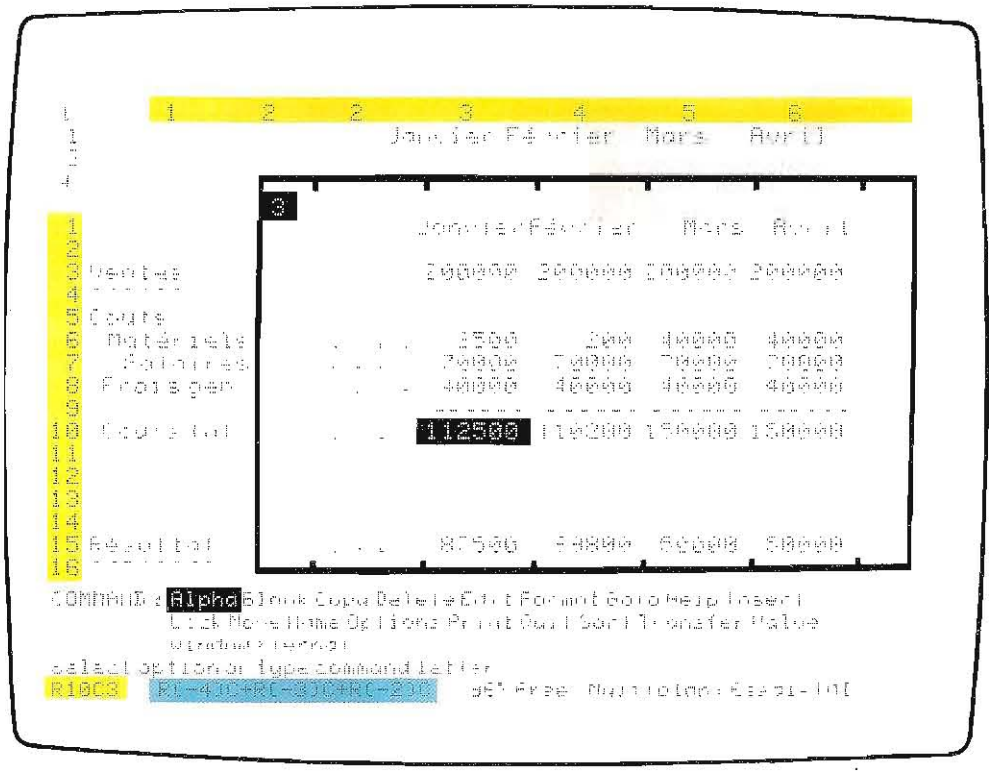
- nom : il s'agit du nom symbolique par lequel l'utilisateur et le programme identifient le contenu de chaque zone ;
- type : définition du type de zone (alphanumérique ou numérique) ;
- longueur : c'est le nombre maximum de caractères prévus dans la zone.

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.

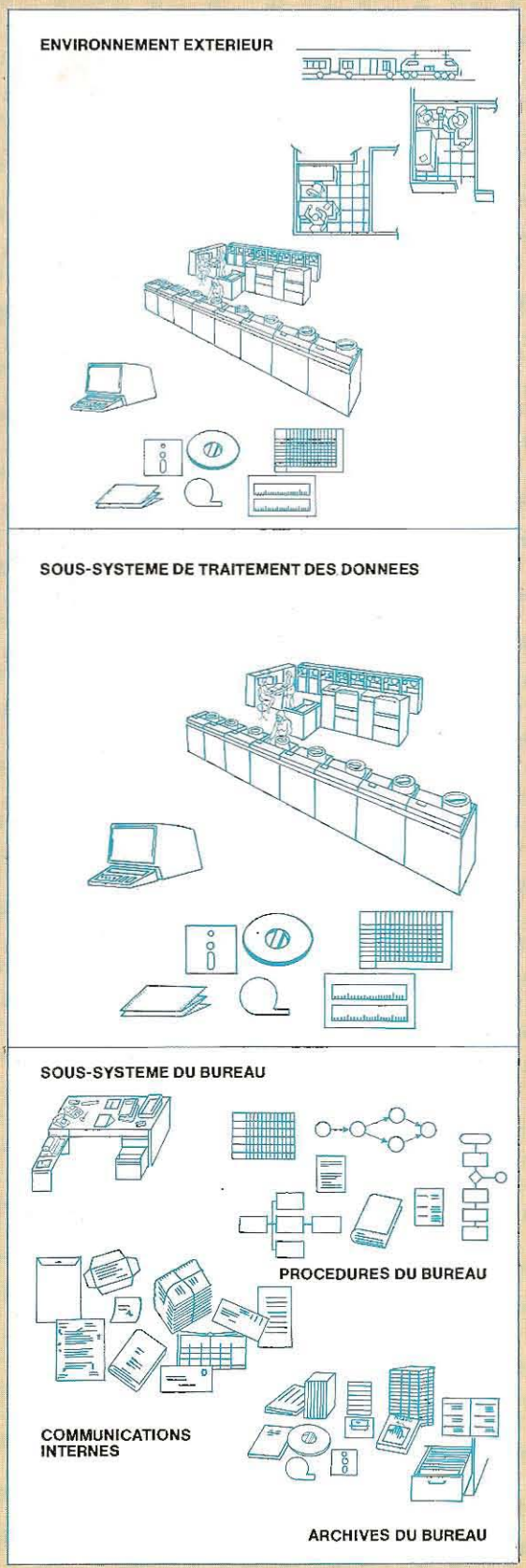
On remarquera que ces lettres sont des objets « normalisés » se rattachant à la correspondance formelle d'un bureau traditionnel. Elles se différencient, en cela, des « messages ».

■ 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) ;*

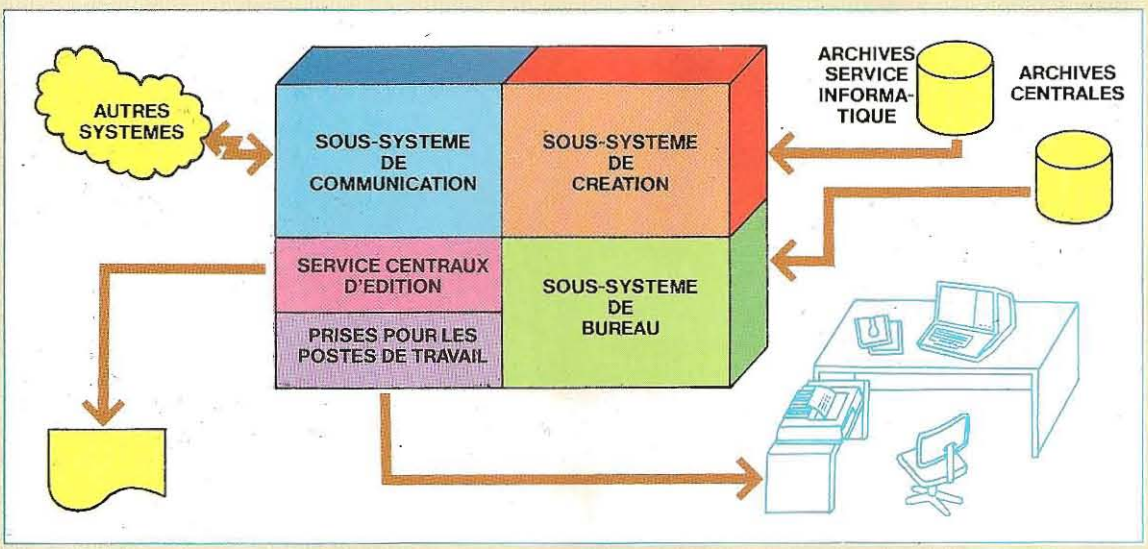
– acquisition/transfert de données à partir ou vers des archives.

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.

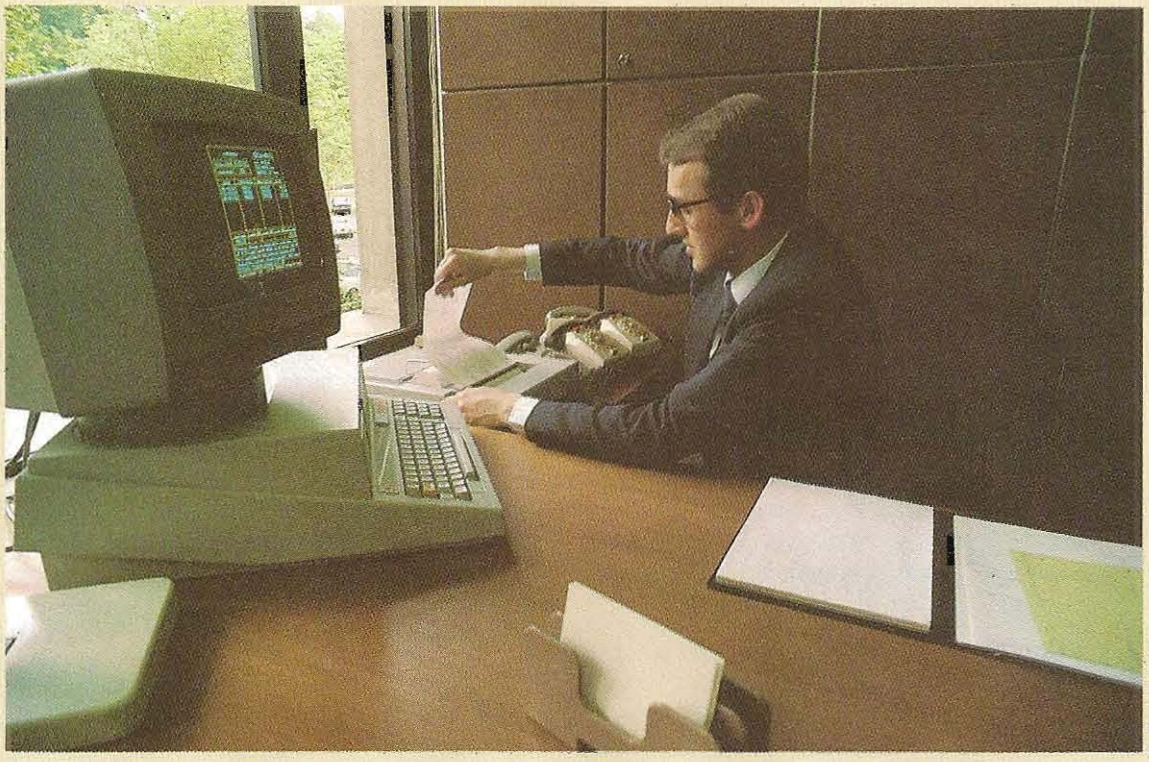
9. ADMINISTRATION DU SYSTEME

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

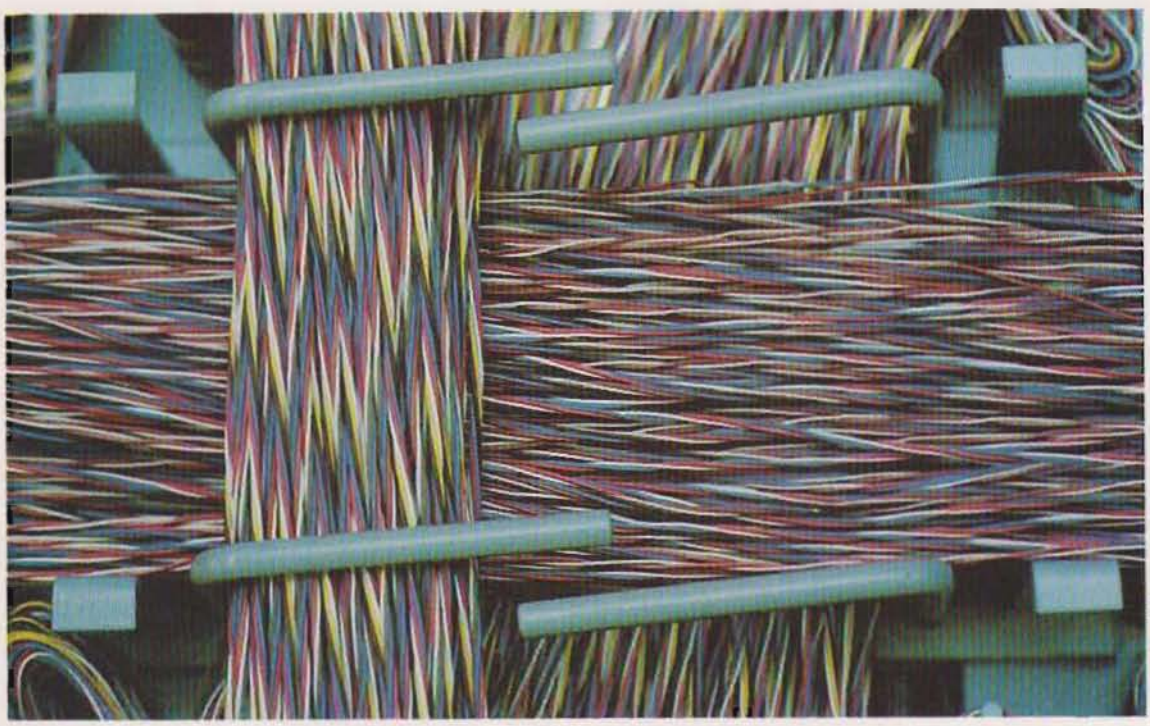
Au centre de chaque système, qu'il s'agisse d'un mini-ordinateur ou d'un micro à usage professionnel ou domestique, se trouve le microprocesseur : c'est l'unité centrale (UC). Il est constitué d'un ensemble de circuits intégrés, transistors, diodes, résistances et condensateurs occupant un espace très 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) :

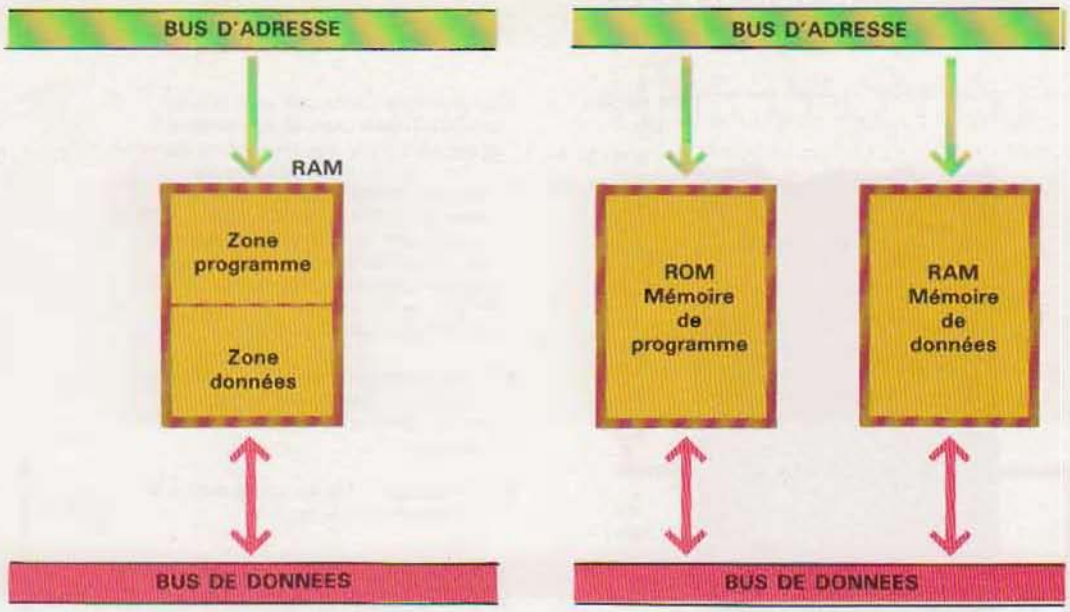
```
10 LET V=10
20 LET S=V+20
```

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.

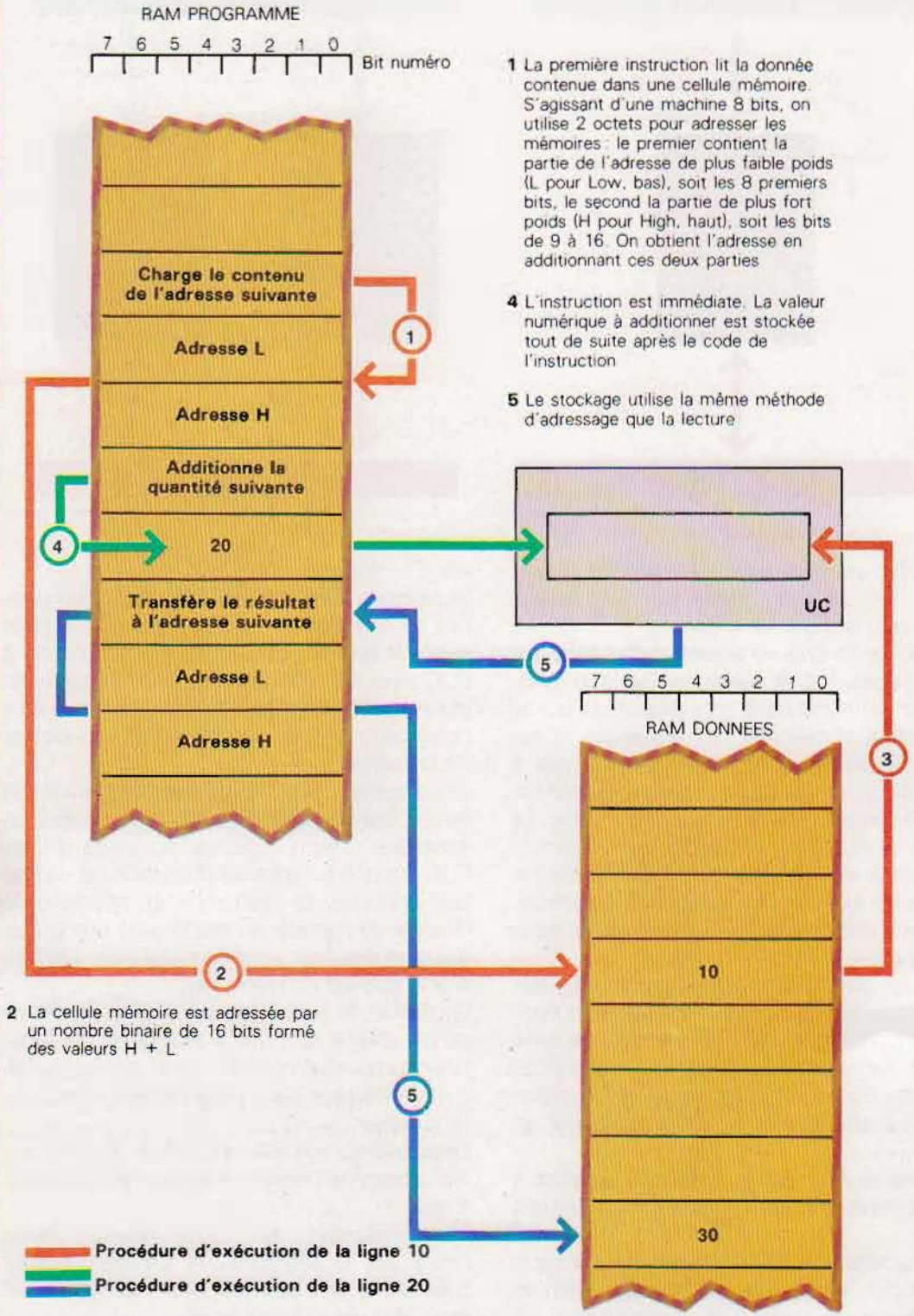
On appelle **cycle machine** l'intervalle de temps (qui correspond, pour la machine, à un ensemble d'états logiques) au cours duquel l'UC transfère l'adresse d'un mot sur le bus des adresses et recherche en mémoire la donnée demandée qu'elle charge sur le bus des données (voir schéma page 869 : Lecture d'une donnée en mémoire).

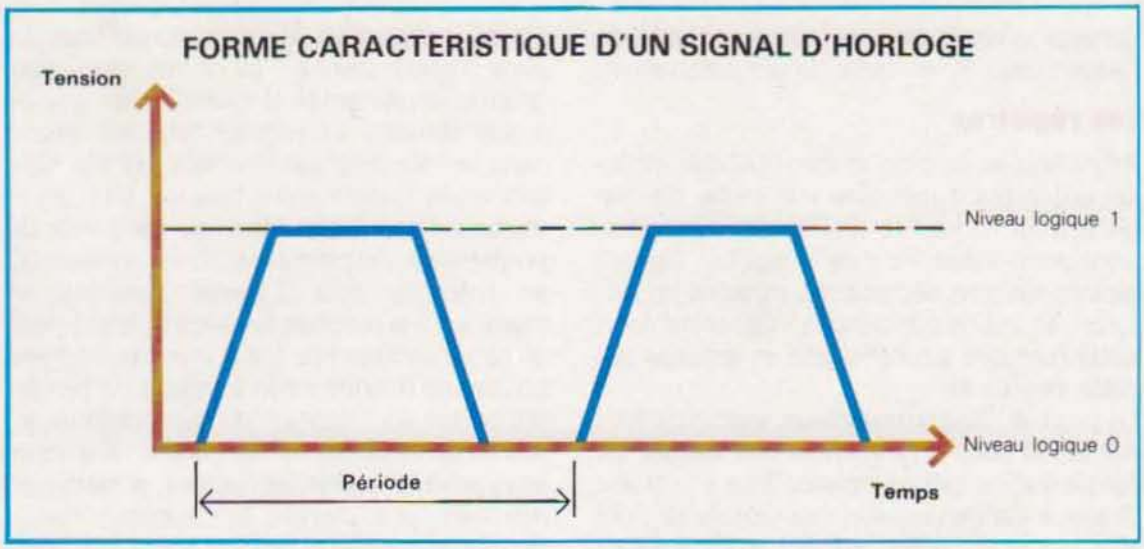
En réalité, la procédure décrite ici ne donne qu'un aperçu simplifié à l'extrême du fonctionnement réel de l'UC, dont la complexité est étroitement liée à l'architecture du microprocesseur.

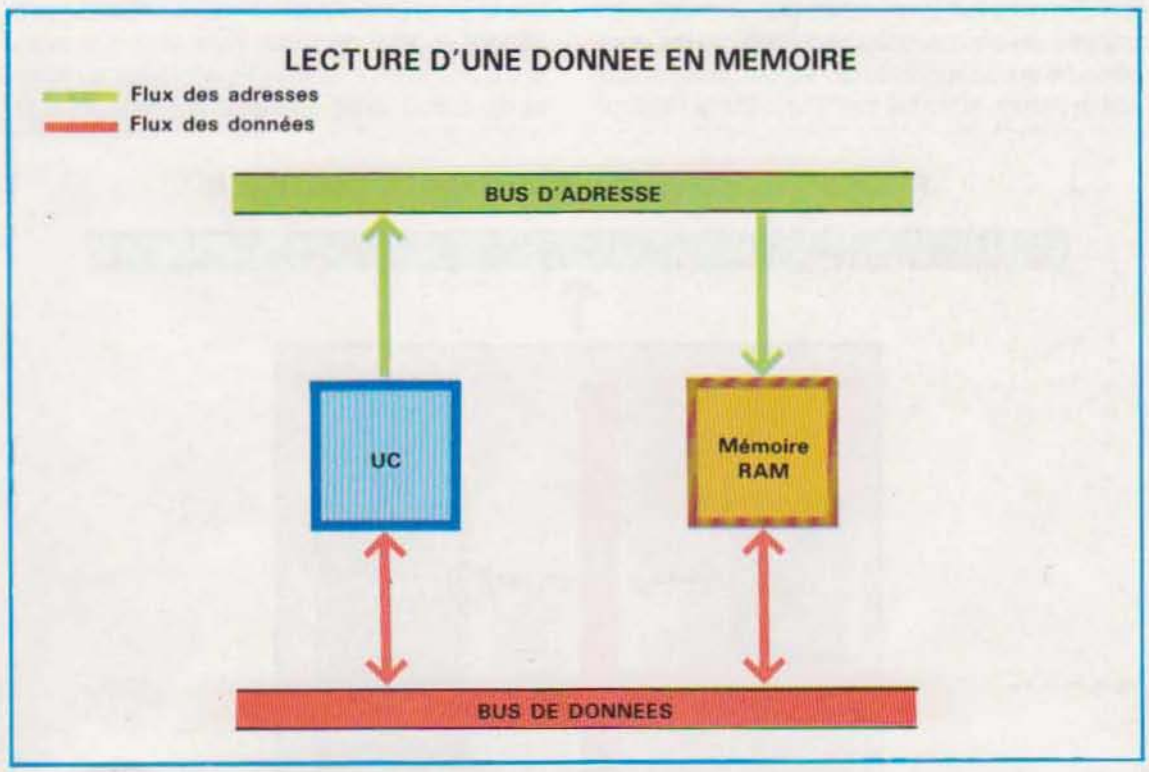
Le décodage suivi de l'exécution de l'instruction constitue l'essentiel d'un cycle d'instructions.

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.

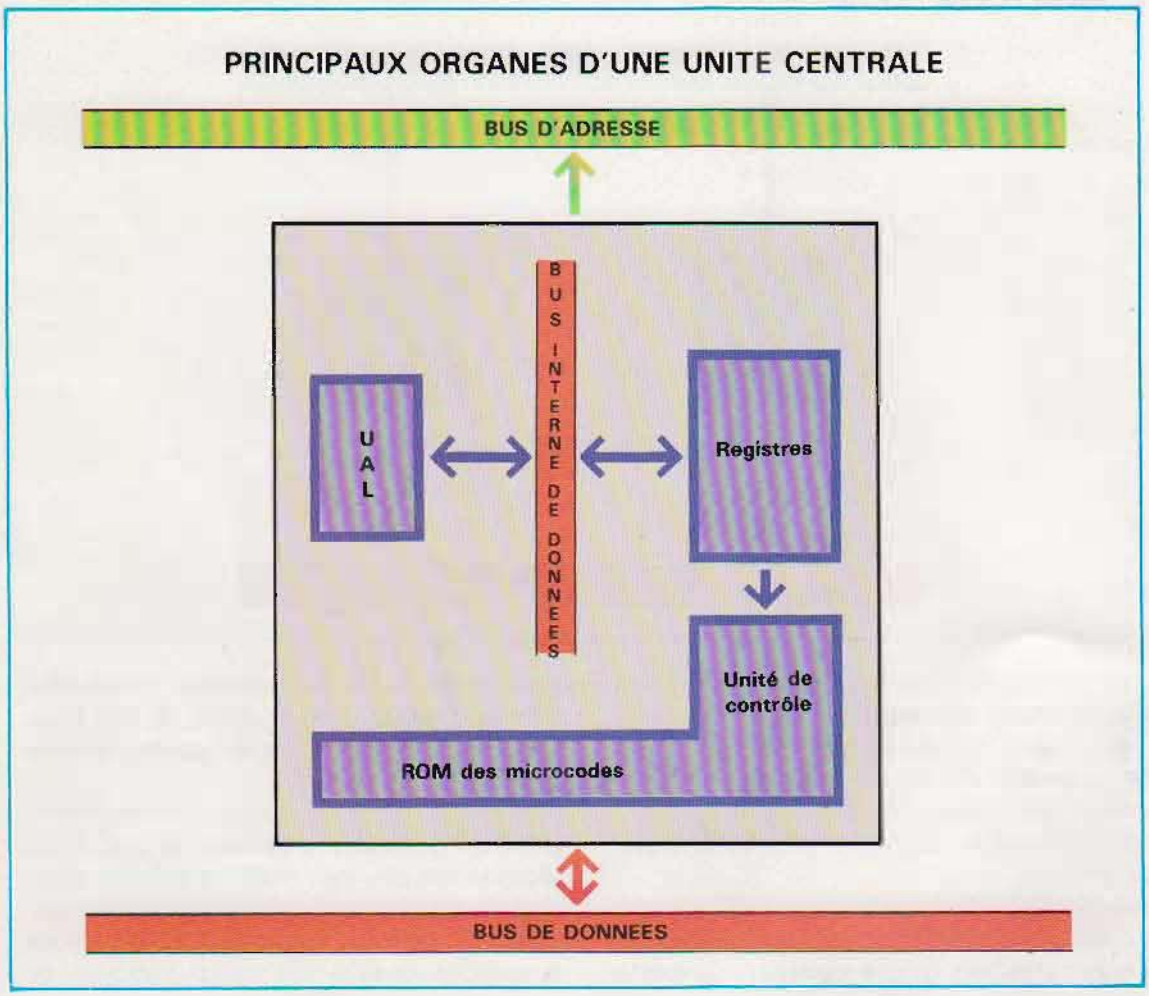
D'une façon générale, il faut avant tout comprendre le mécanisme de base de l'UC. Nous allons le voir plus loin, mais il a déjà été exposé dans les chapitres précédents sur les systèmes à base de microprocesseurs. Nous illustrerons ensuite les caractéristiques de

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.

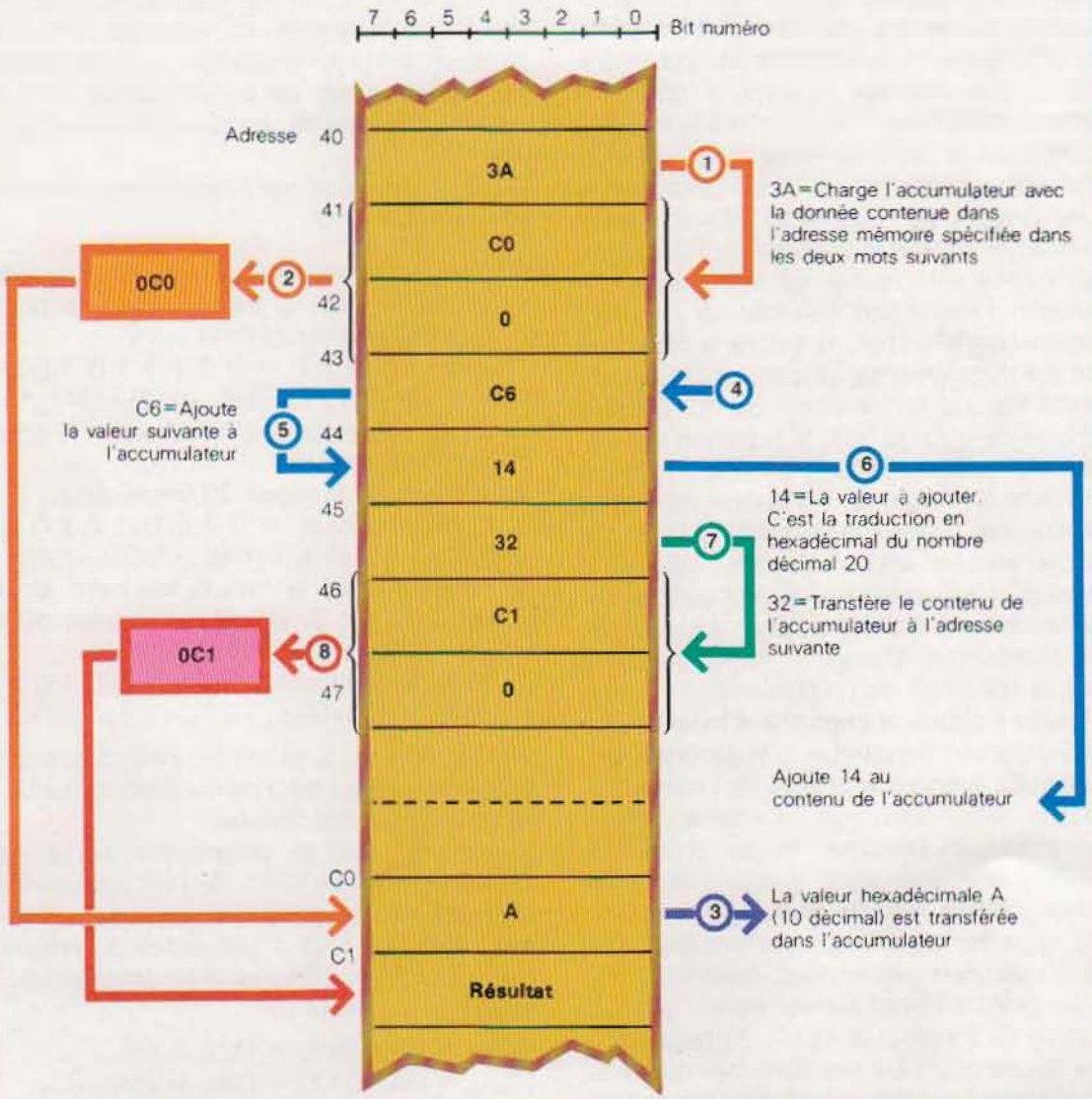
Ces valeurs devront être mises en mémoire dans les positions de mémoires adéquates pour pouvoir ensuite être utilisées par le programme. Ci-dessous, on a reporté le schéma complet de tout le programme et illustré les étapes de son exécution.
Pages 873 à 875, le déroulement du programme est suivi pas 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 :

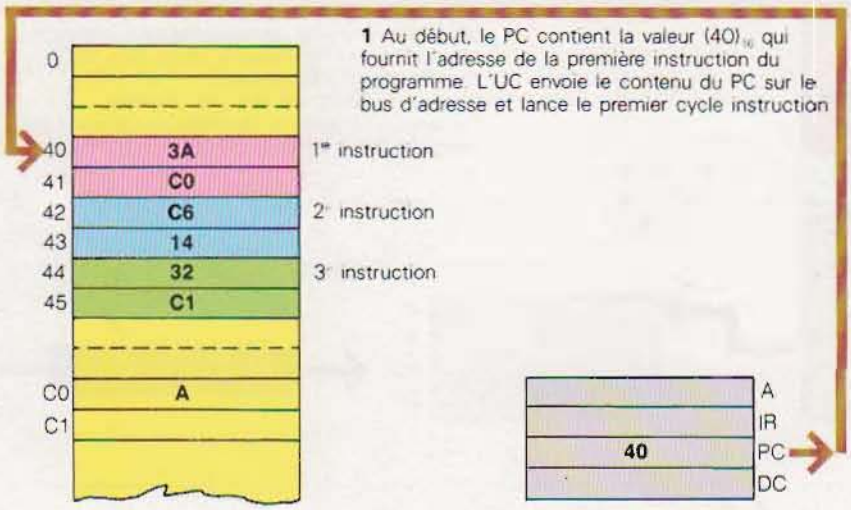
— les instructions (traduites en codes hexadécimaux) sont difficiles à identifier. La vérification du programme devient une

EXECUTION D'UN PROGRAMME ECRIT EN LANGAGE-MACHINE



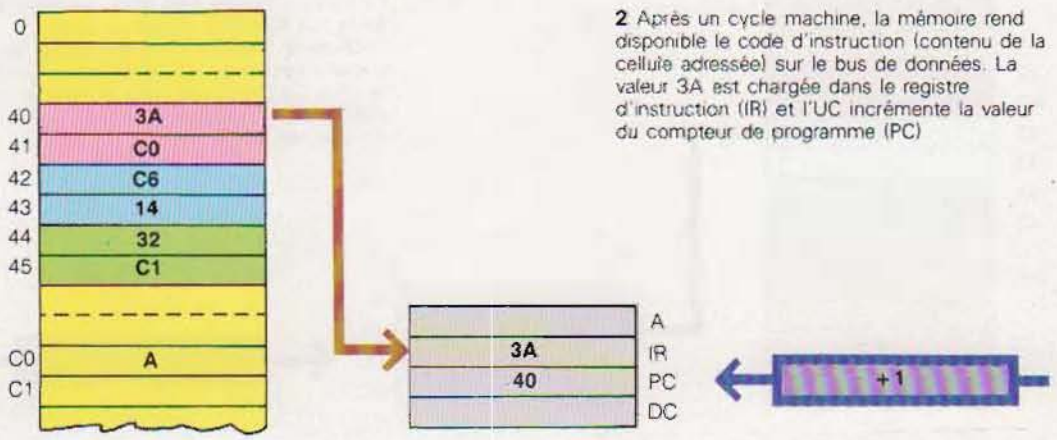
Ici, on a utilisé deux types d'instructions de chargement. Dans le premier, le code 3A (Charger l'accumulateur...), les données sont prélevées par l'intermédiaire d'un adressage vers n'importe quelle cellule mémoire (ici C0), même loin de celle contenant l'instruction. Le second (immédiat, code C6) utilise 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

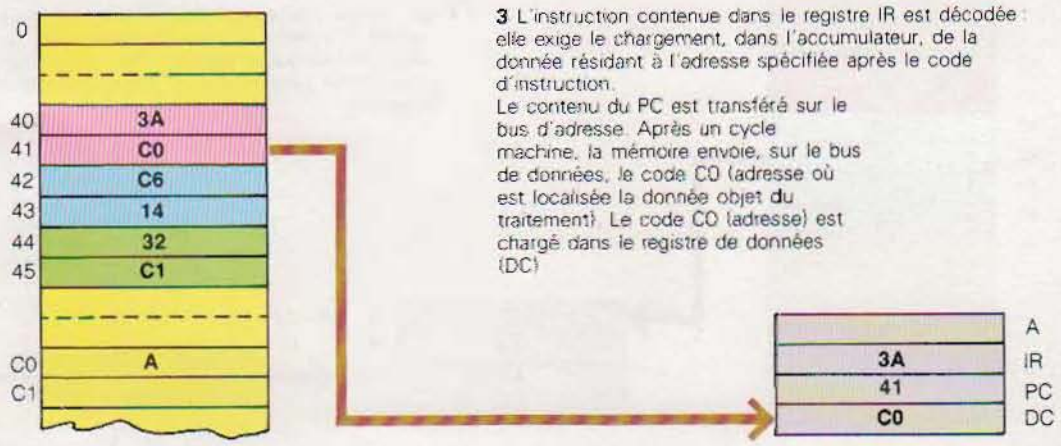


1 Au début, le PC contient la valeur $(40)_{16}$ qui fournit l'adresse de la première instruction du programme. L'UC envoie le contenu du PC sur le bus d'adresse et lance le premier cycle instruction

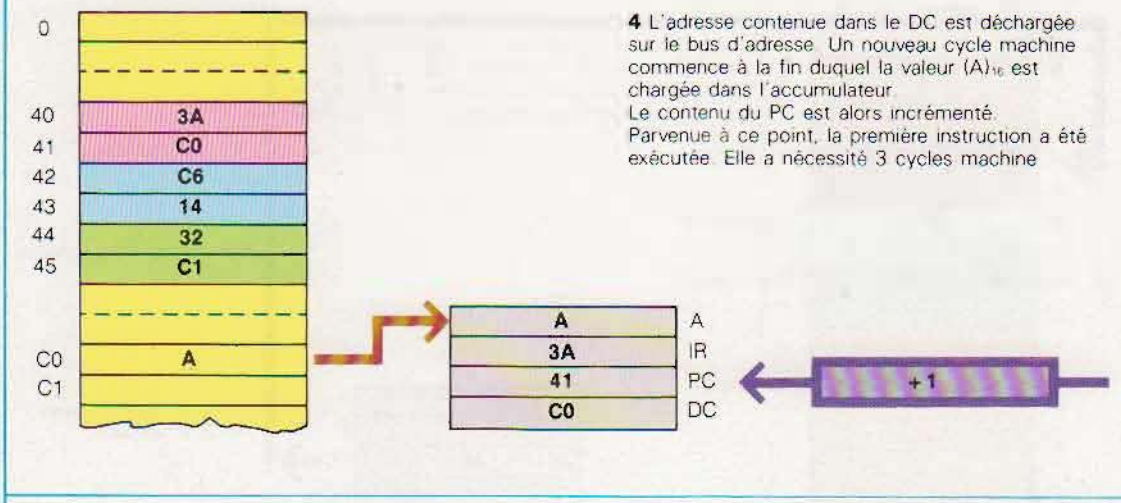
A = Accumulateur
IR = Registre d'instruction
PC = Compteur de programme
DC = Registre de données



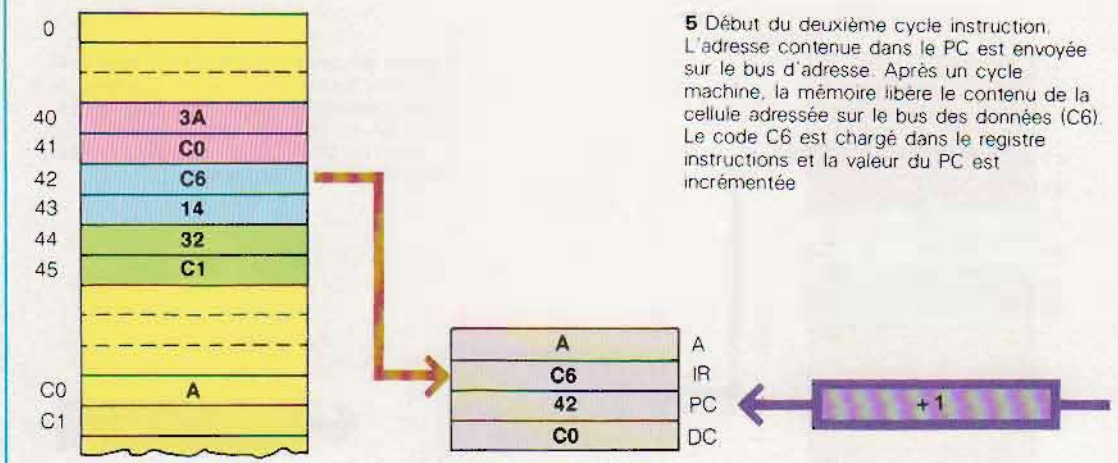
2 Après un cycle machine, la mémoire rend disponible le code d'instruction (contenu de la cellule adressée) sur le bus de données. La valeur 3A est chargée dans le registre d'instruction (IR) et l'UC incrémente la valeur du compteur de programme (PC)



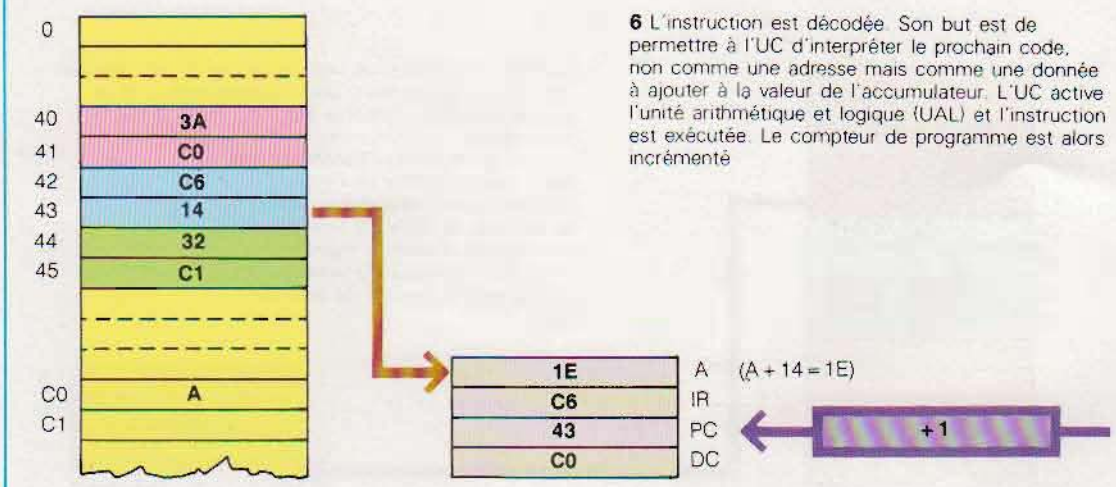
3 L'instruction contenue dans le registre IR est décodée : elle exige le chargement, dans l'accumulateur, de la donnée résidant à l'adresse spécifiée après le code d'instruction.
Le contenu du PC est transféré sur le bus d'adresse. Après un cycle machine, la mémoire envoie, sur le bus de données, le code C0 (adresse où est localisée la donnée objet du traitement). Le code C0 (adresse) est chargé dans le registre de données (DC)



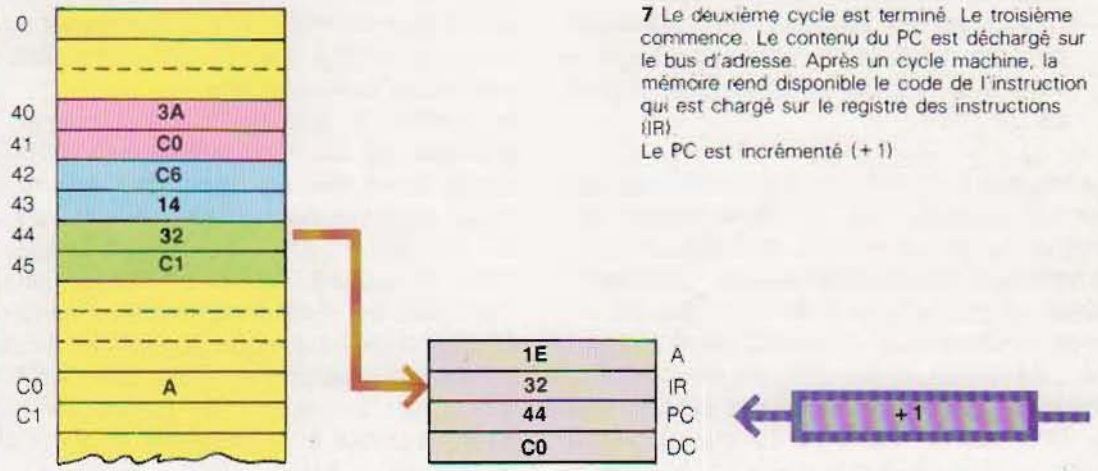
4 L'adresse contenue dans le DC est déchargée sur le bus d'adresse. Un nouveau cycle machine commence à la fin duquel la valeur $(A)_{16}$ est chargée dans l'accumulateur.
Le contenu du PC est alors incrémenté.
Parvenue à ce point, la première instruction a été exécutée. Elle a nécessité 3 cycles machine



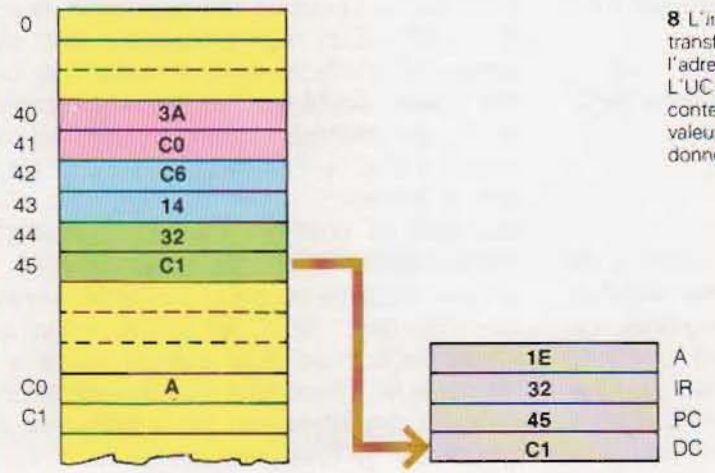
5 Début du deuxième cycle instruction.
L'adresse contenue dans le PC est envoyée sur le bus d'adresse. Après un cycle machine, la mémoire libère le contenu de la cellule adressée sur le bus des données (C6). Le code C6 est chargé dans le registre instructions et la valeur du PC est incrémentée



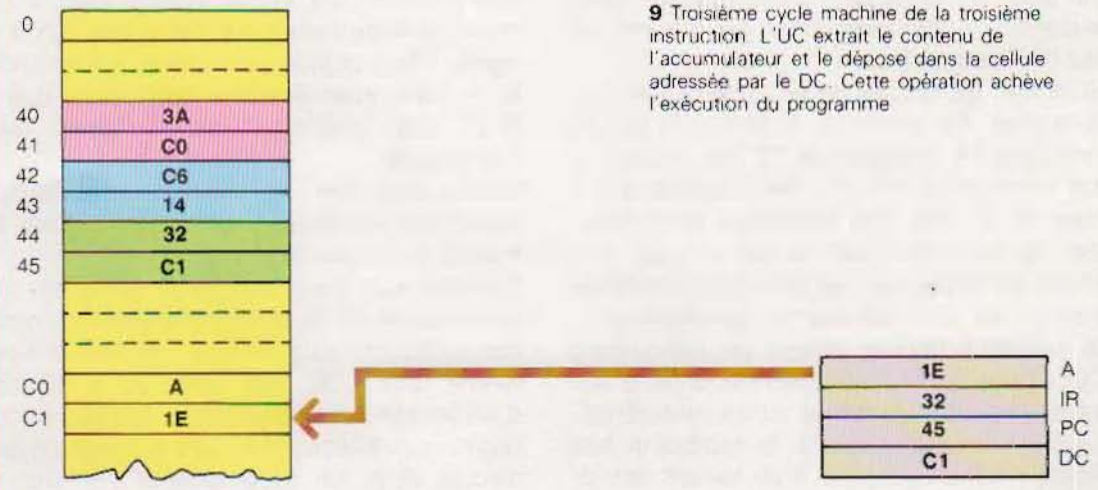
6 L'instruction est décodée. Son but est de permettre à l'UC d'interpréter le prochain code, non comme une adresse mais comme une donnée à ajouter à la valeur de l'accumulateur. L'UC active l'unité arithmétique et logique (UAL) et l'instruction est exécutée. Le compteur de programme est alors incrémenté



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.

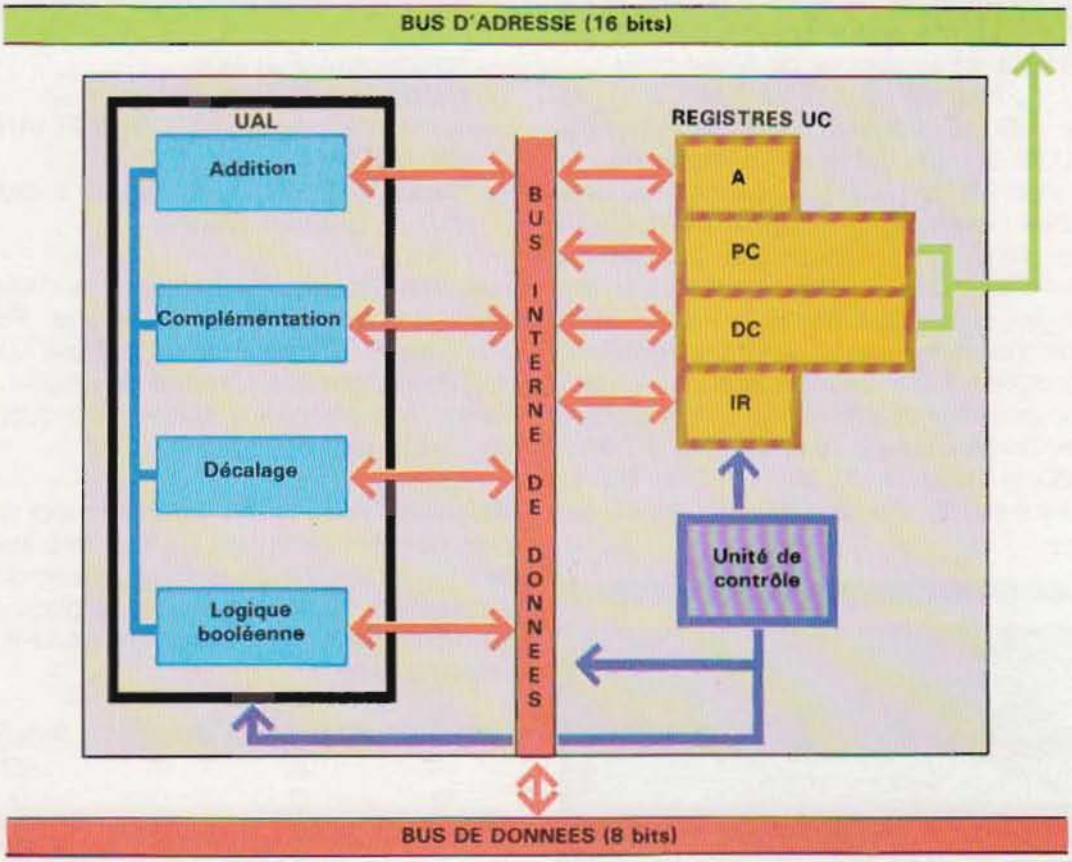
Bien évidemment, ces opérations ne s'effectuent que sur des nombres binaires. Pour comprendre le fonctionnement d'une UAL, une description plus détaillée des règles de l'arithmétique binaire, ainsi que des circuits et dispositifs, est nécessaire.

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

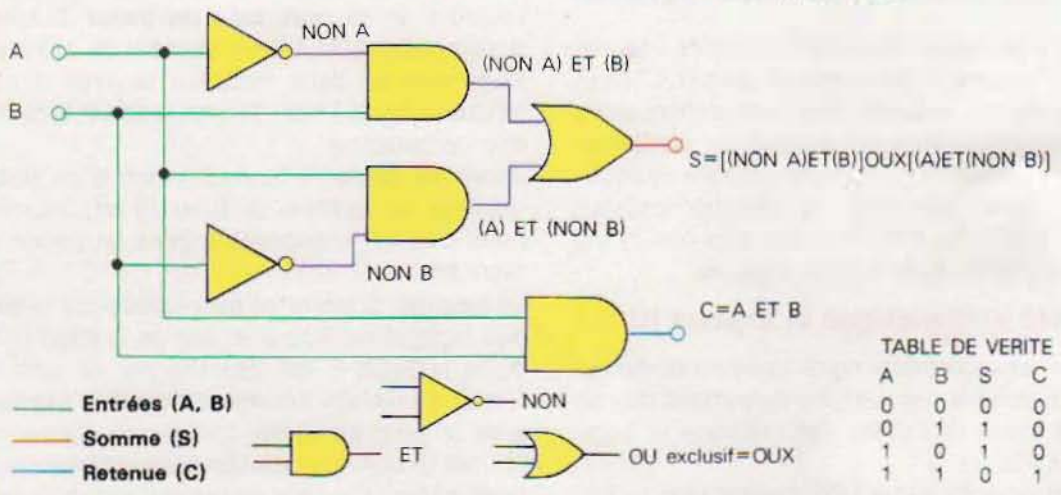


TABLE DE VERITE

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)

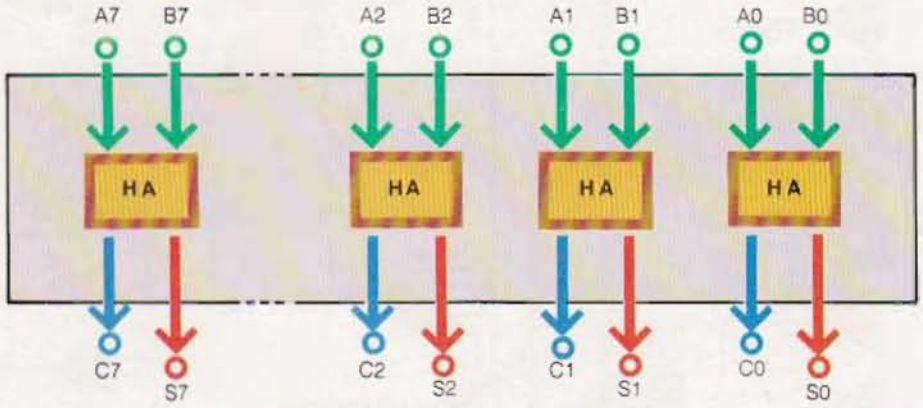
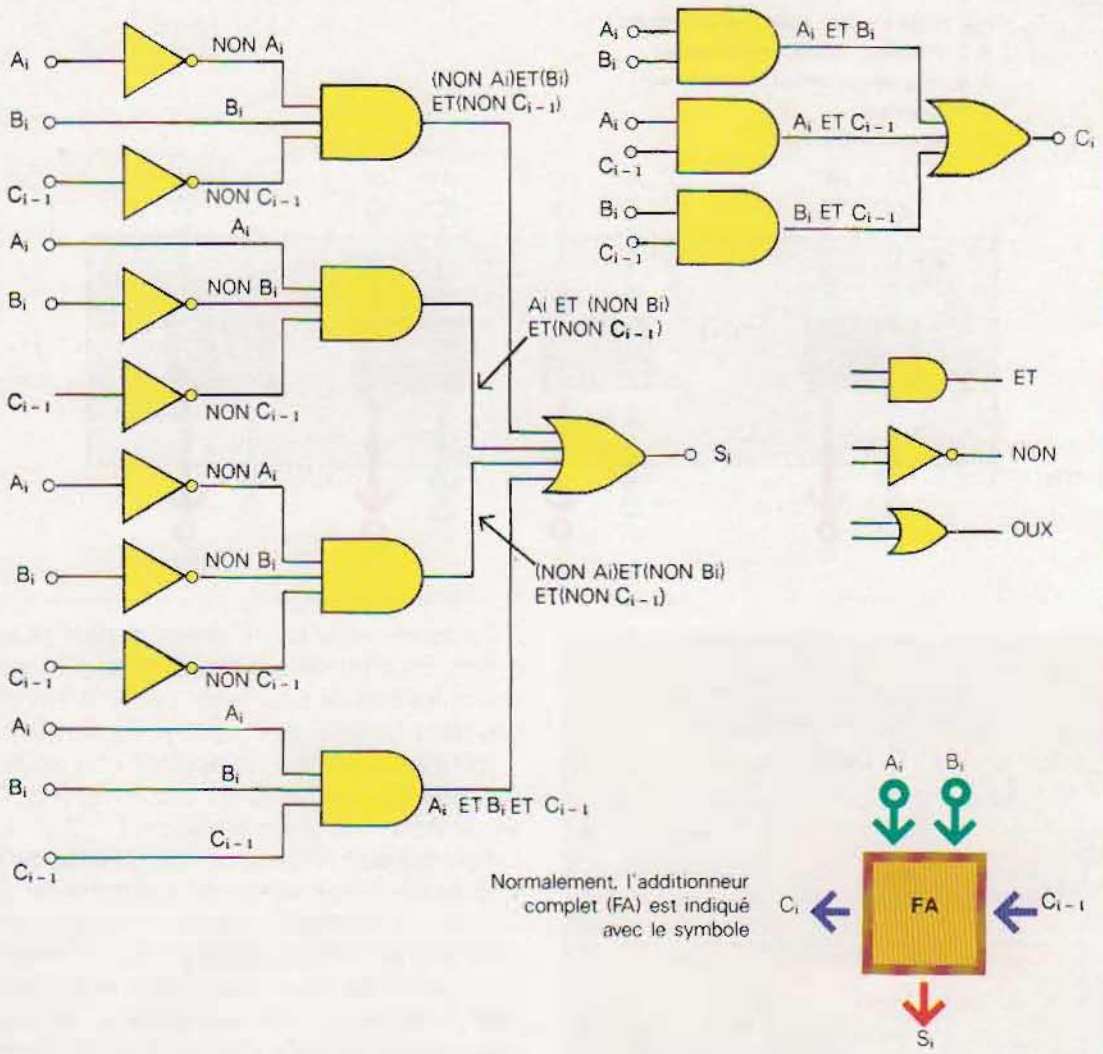


TABLE DE VERITE D'UN ADDITIONNEUR COMPLET

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

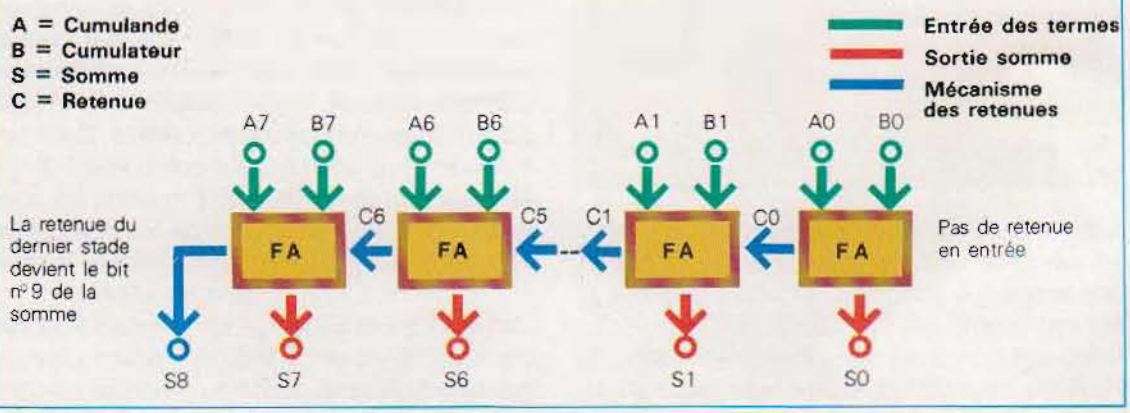
à 8 entrées avec 8 demi-additionneurs placés en parallèle comme ci-dessus, où le circuit additionneur à 2 entrées (exemple précédent) est représenté par les lettres HA.
Cette solution n'est cependant pas viable car l'additionneur à 2 entrées ne tient pas compte d'éventuels reports provenant d'états plus 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

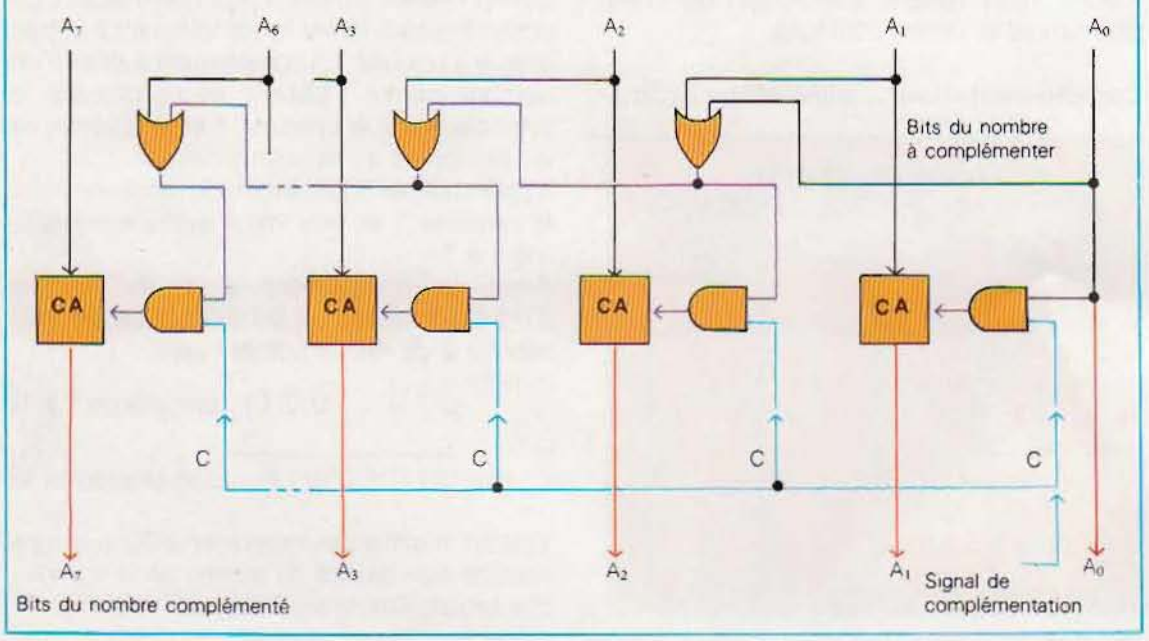
	Binaire	Décimal
Plus grand nombre	1 1 1 0 1 0 1 0 —	234 —
Plus petit nombre	1 0 1 0 1 1 1 0 =	174 =
Résultat	0 0 1 1 1 1 0 0	60

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.

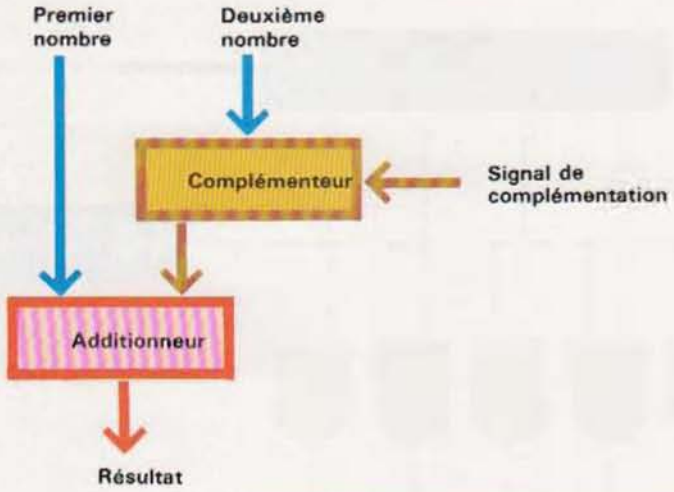
Sur cette page, on a représenté un circuit calculant le complément à 2 d'un nombre de 8 bits. Le symbole CA indique le circuit qui se charge de l'inversion du bit du nombre binaire à complémenter. Cette opération est subordonnée à la présence du signal C. S'il est égal à 1, le nombre binaire est complémenté, 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-

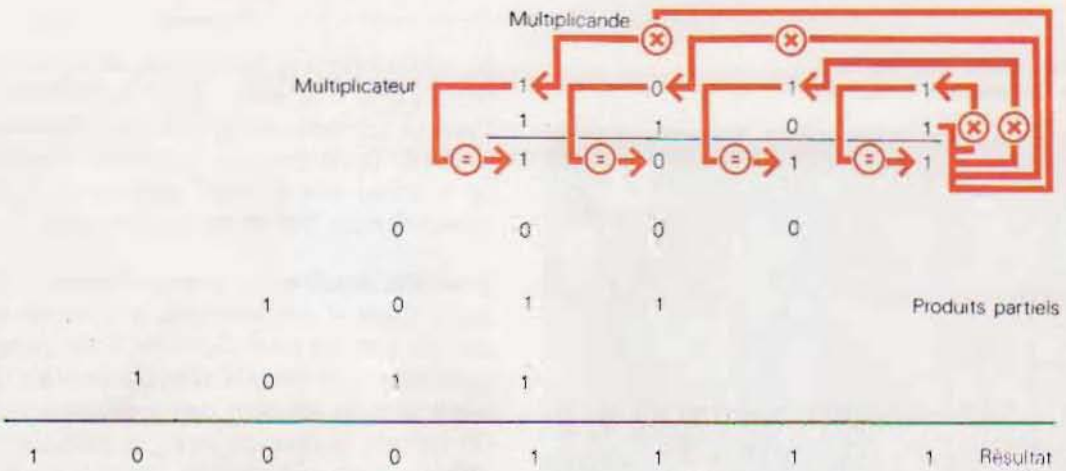
CIRCUIT PARALLELE DE COMPLEMENTATION A 2



FONCTIONNEMENT D'UN ADDITIONNEUR-SOUSTRACTEUR

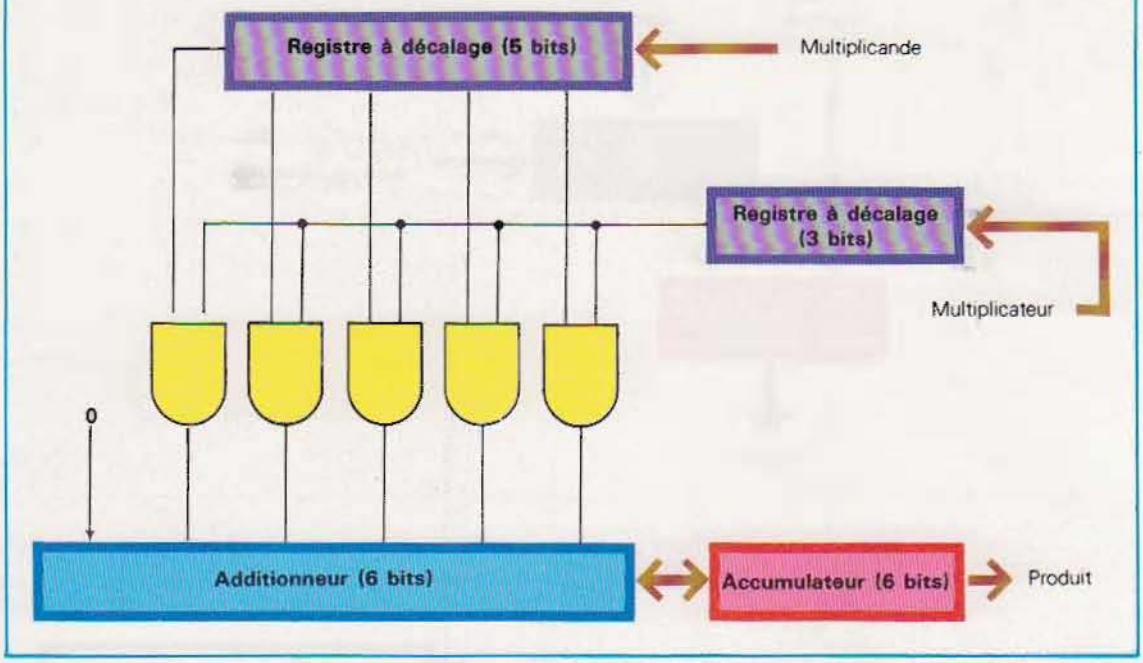


MULTIPLICATION BINAIRE



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.

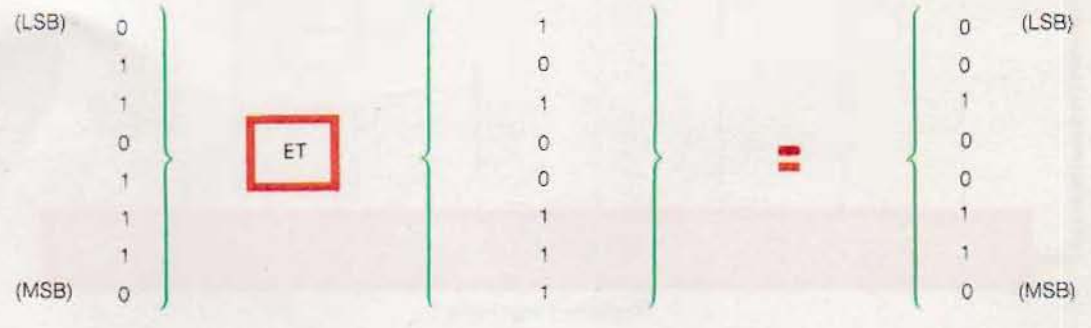
Opérations booléennes. ET, OU, NON et OUX (OU exclusif) sont des opérations logiques couramment effectuées par l'UAL.
Elles ont déjà été décrites (page 72) et on a vu comment il est possible de les représenter d'une manière synthétique à l'aide de la table de vérité des fonctions, ET, OU, OUX, NON.
Supposons 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.

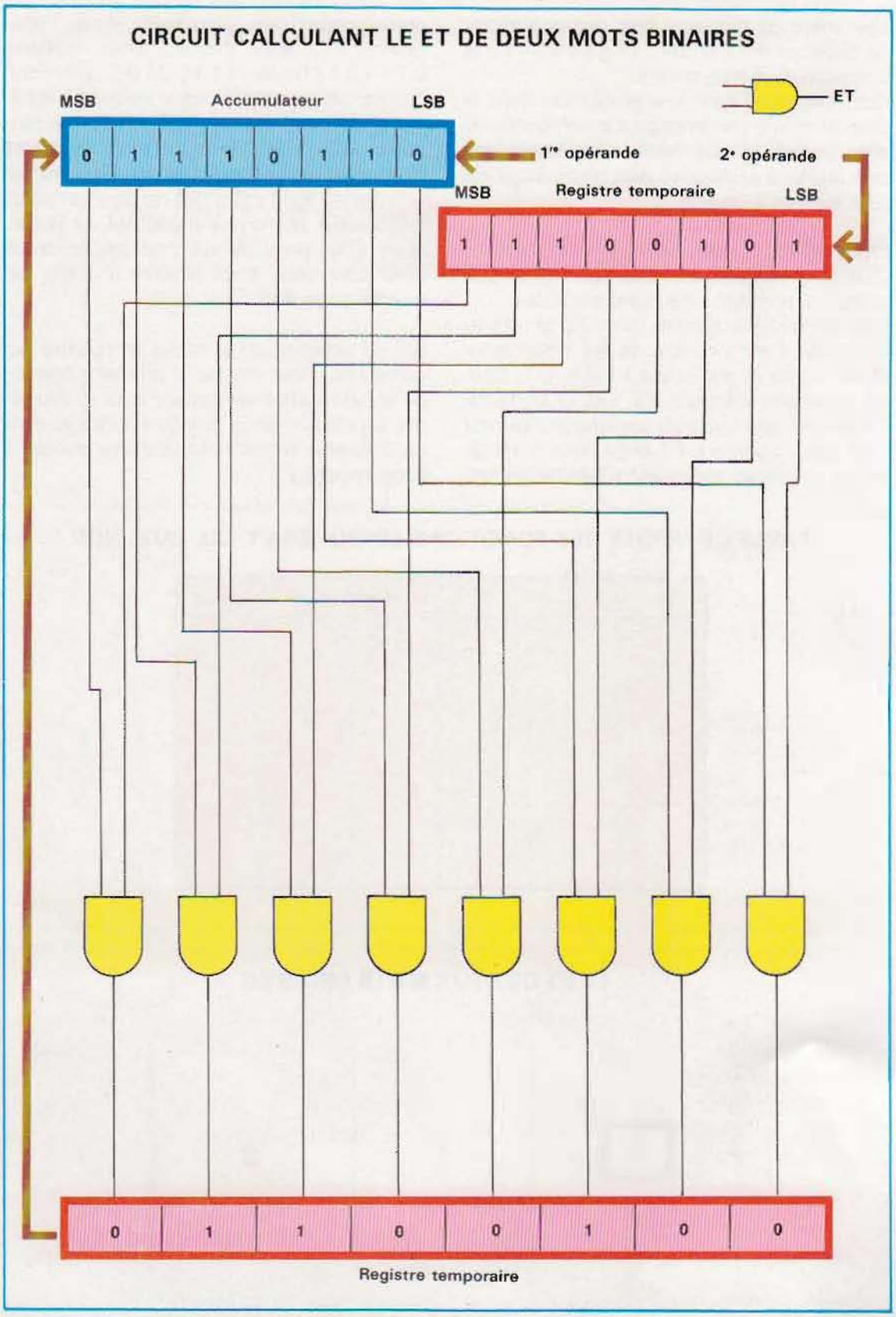
Sur ce schéma, on a utilisé un registre accumulateur pour stocker la première opérande et un registre temporaire pour la deuxième. Le résultat de ET (AND) est chargé dans un troisième registre avant d'être envoyé à l'accumulateur.

TABLE DE VERITE DES FONCTIONS LOGIQUES ET, OU, OUX, NON

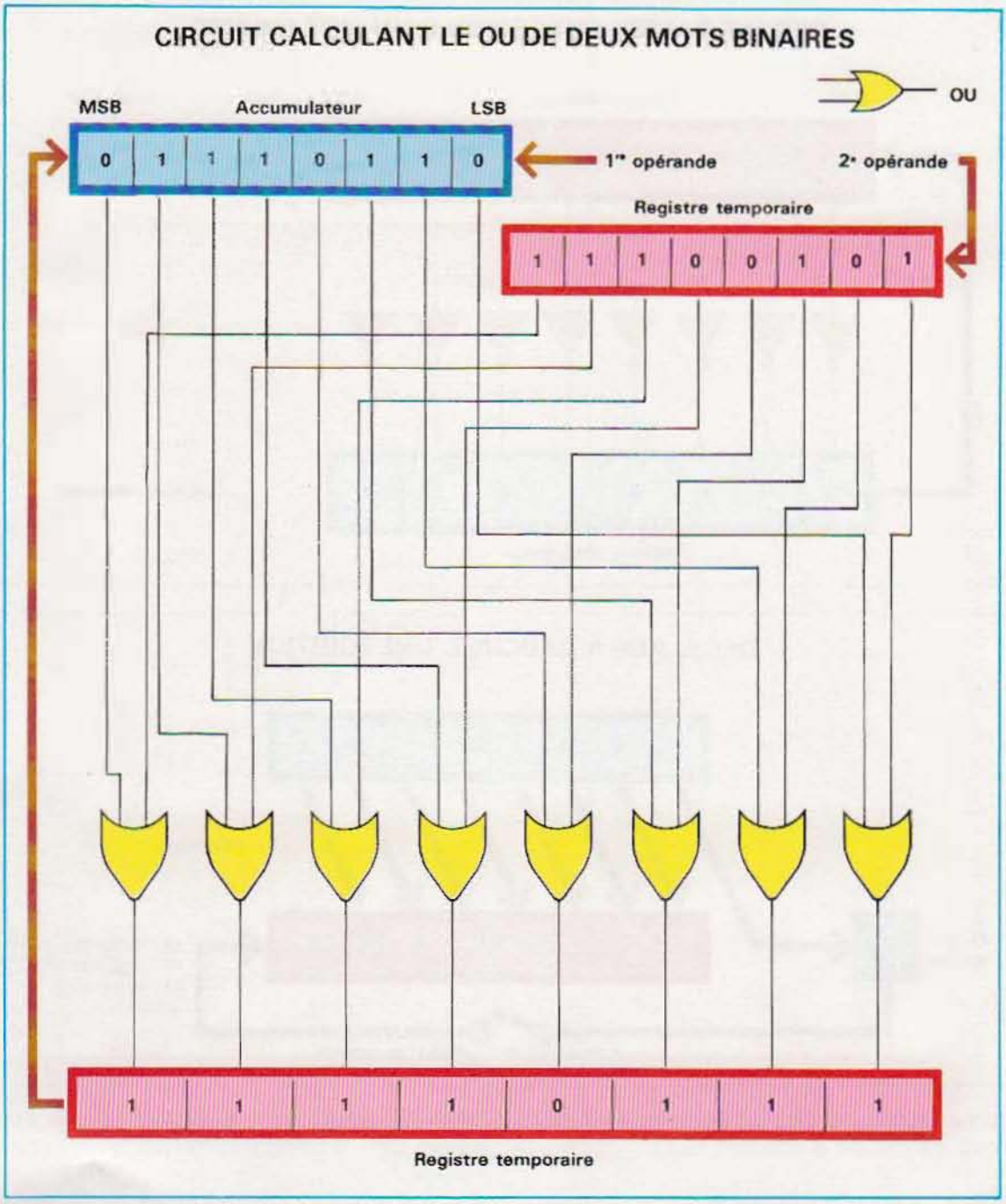
A	B	ET	OU	OUX	NON A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

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

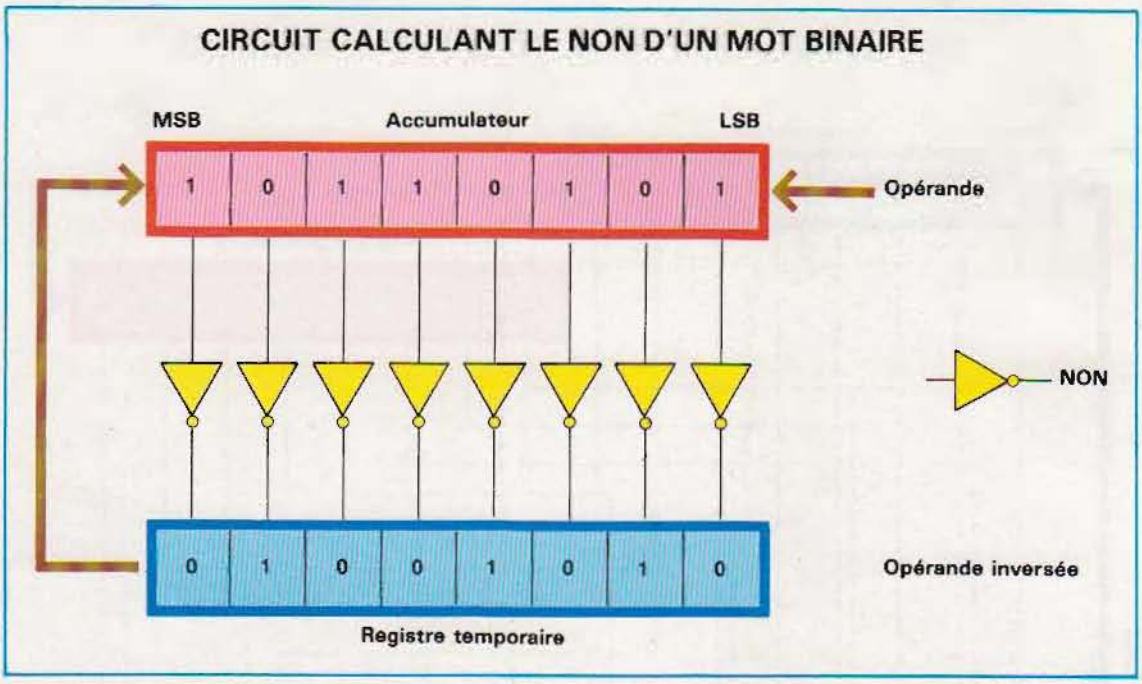
CIRCUIT CALCULANT LE OU DE DEUX MOTS BINAIRES

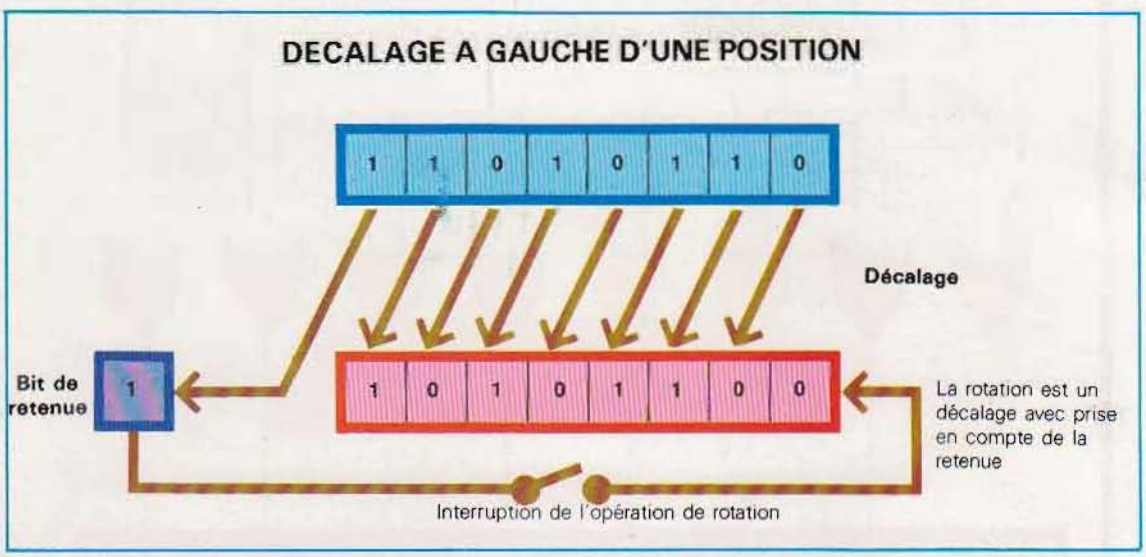


L'opération OU peut être réalisée comme ci-dessus, où les opérateurs ET ont été remplacés par des opérateurs OU.
Il n'est pas nécessaire de disposer de tous les circuits correspondant aux fonctions logiques pour les calculer. En principe, n'ont été prévus que deux sortes de circuits : soit le couple 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.



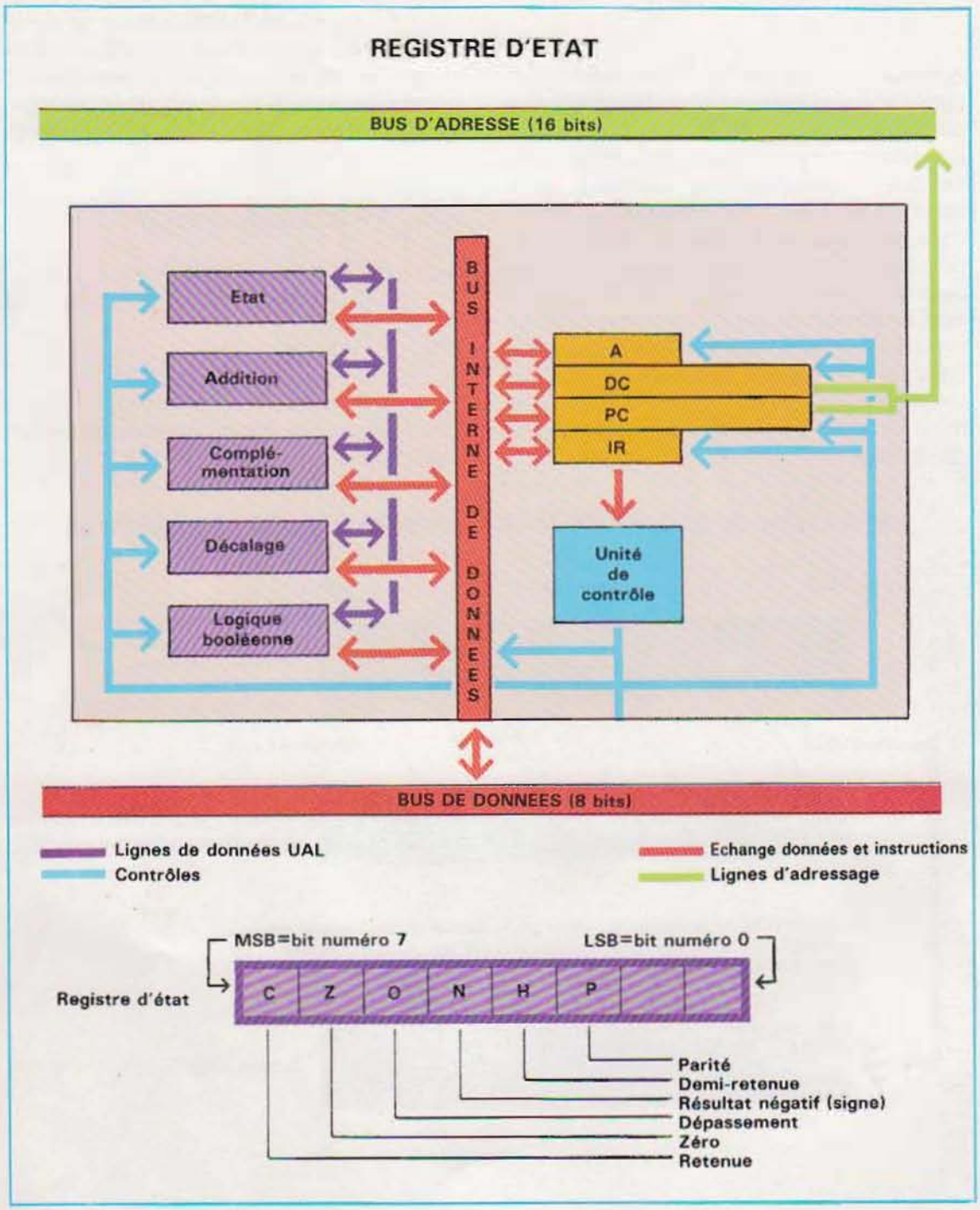


On a reporté, ci-dessus, le schéma d'un circuit qui calcule la fonction NON.

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.

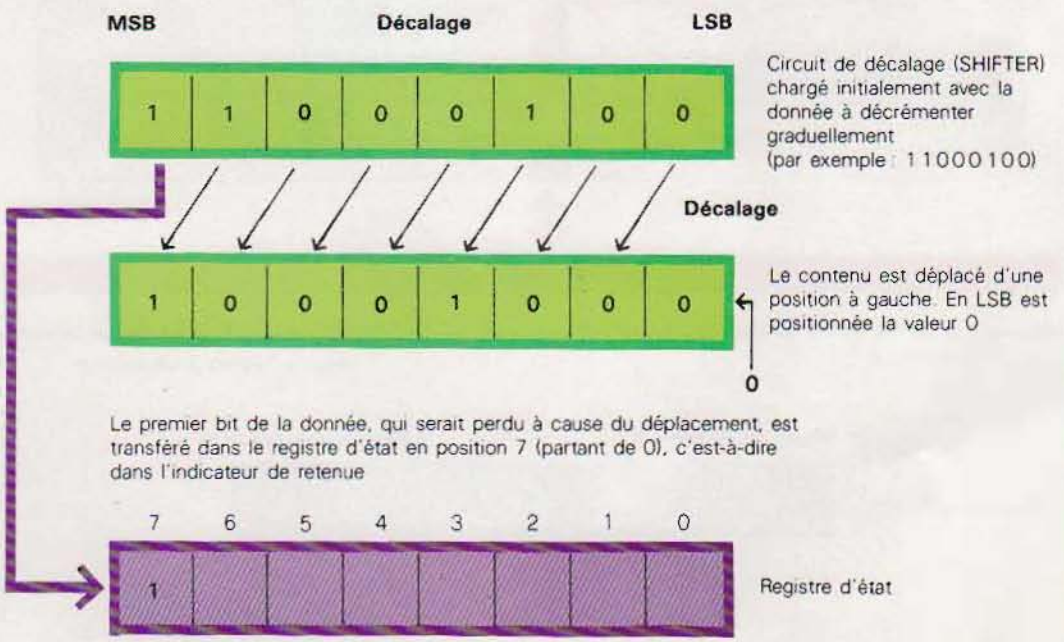
Dépassement. L'indicateur de dépassement prend la valeur 1 quand le résultat de la dernière opération arithmétique n'est pas correct, à cause de la présence d'une retenue après le chiffre le plus significatif. Normalement, le dernier bit (MSB) indique le signe. Ceci signifie qu'il y a eu une retenue 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 :

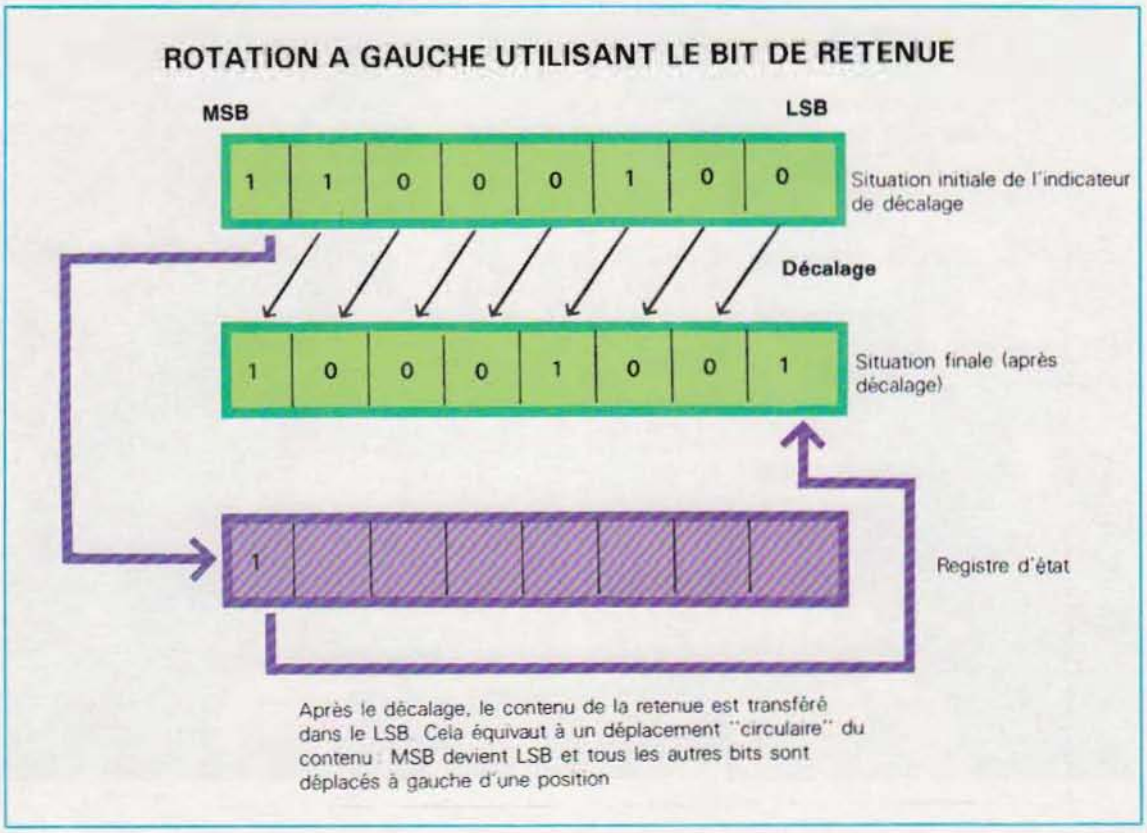
Bit du signe — Bit le plus significatif

```
  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.

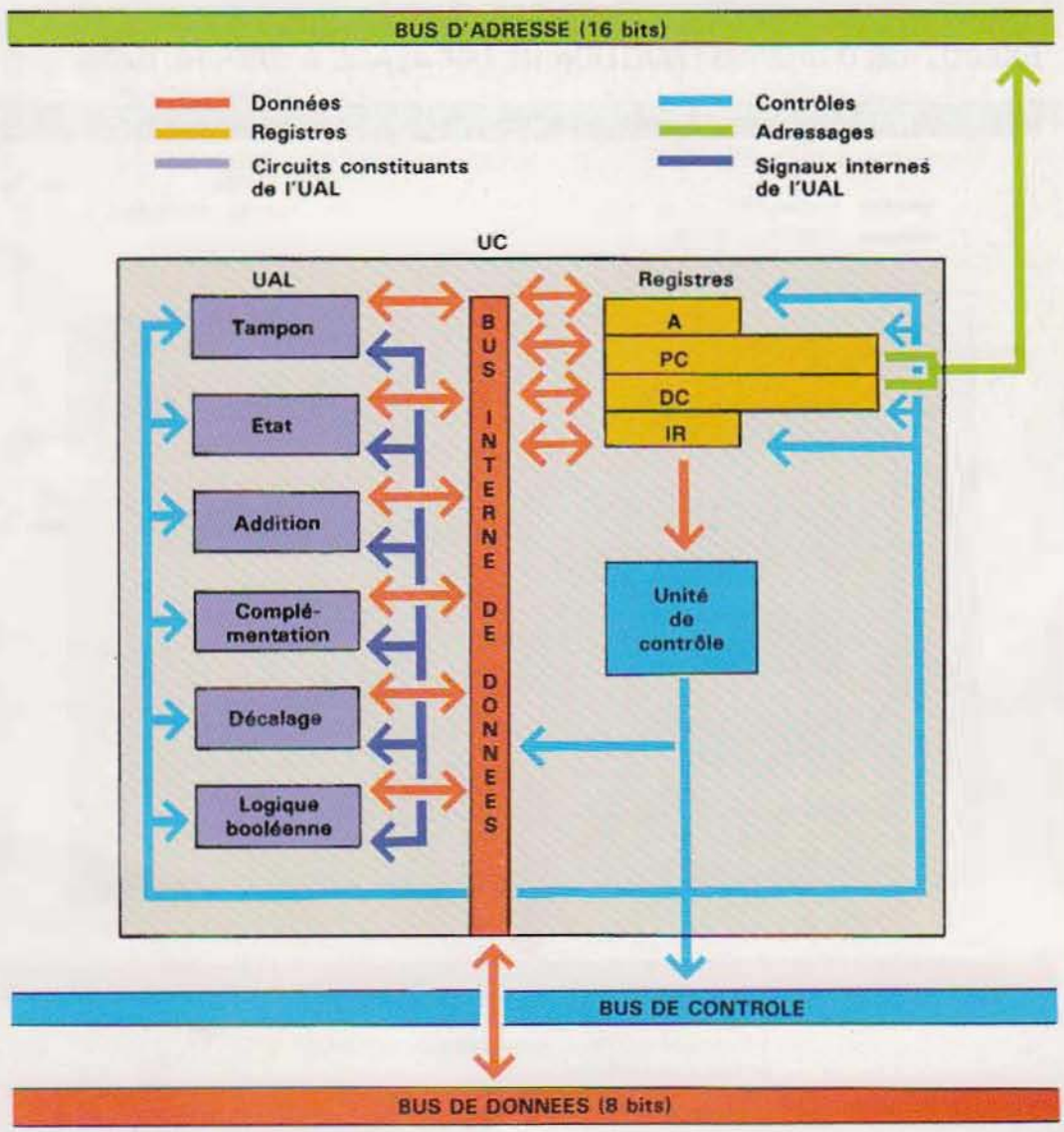
Le tampon de l'UAL

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).

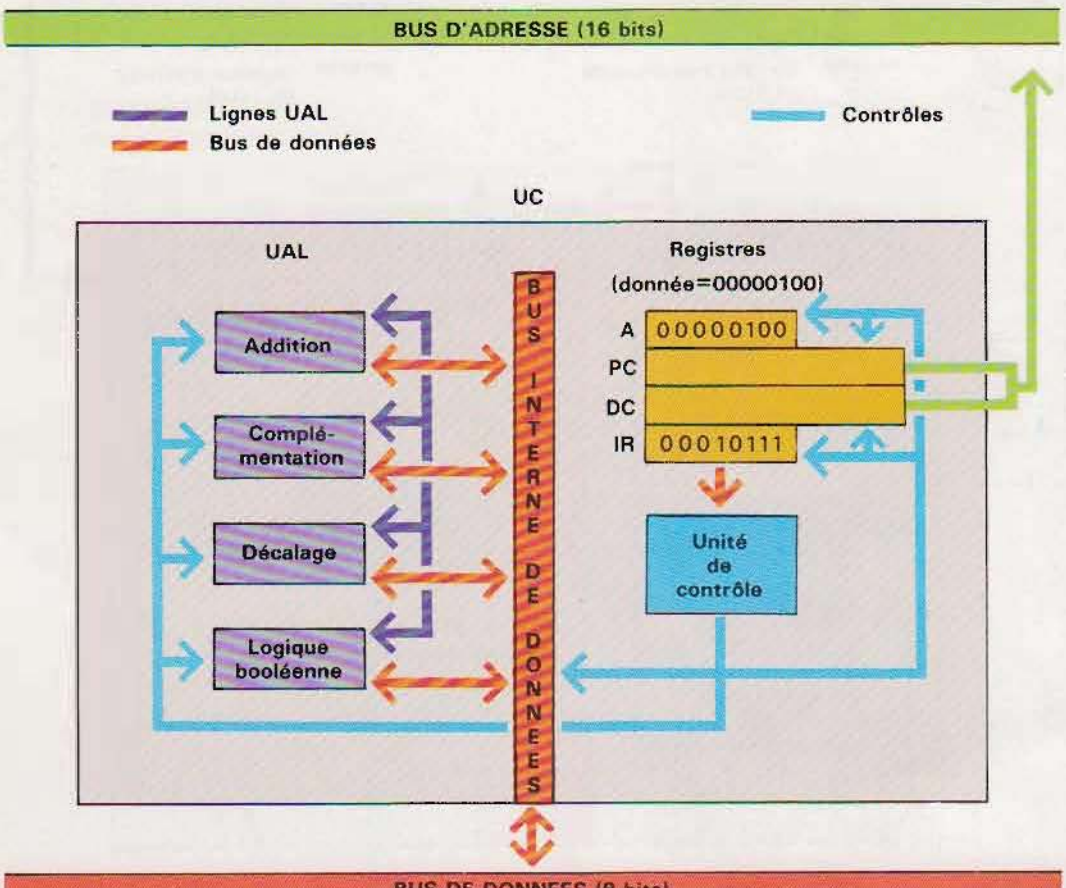
SCHEMA COMPLET D'UNE UC



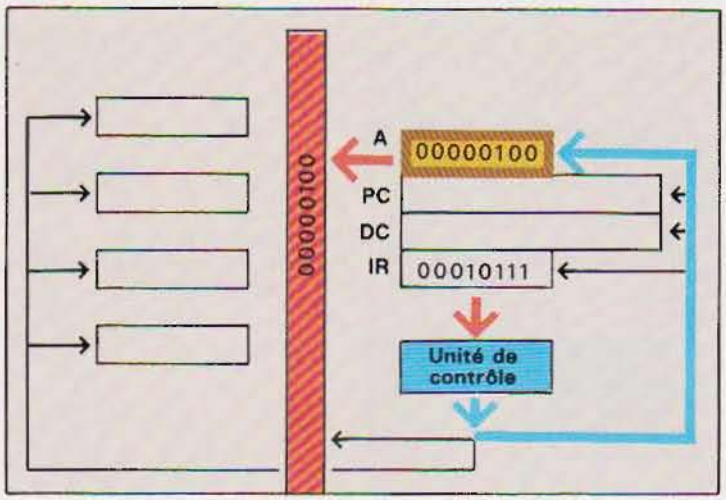
En fonction du code binaire contenu dans l'IR, l'unité de contrôle émet les signaux d'autorisation qui permettent aux éléments de l'UAL de recevoir les données. L'exécution d'une instruction de décalage à gauche du contenu de l'accumulateur est illustrée pages 894 à 896. Les signaux de contrôle jouent 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).

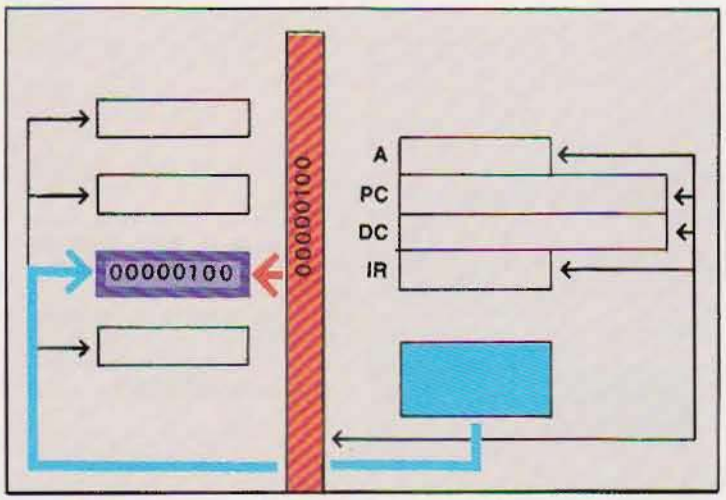
EXECUTION D'UNE INSTRUCTION DE DECALAGE A GAUCHE DANS L'UC



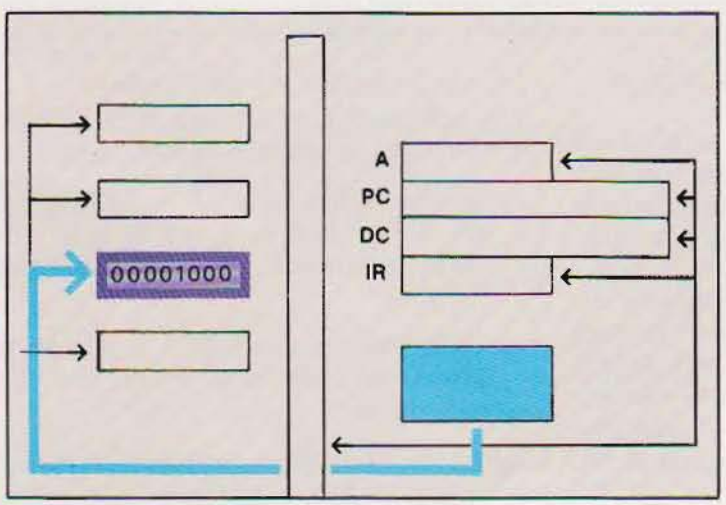
1 Etat initial de l'UC. L'accumulateur A contient la donnée à décrémenter.
Le registre d'instructions IR contient le code de l'instruction à exécuter (décalage à gauche)



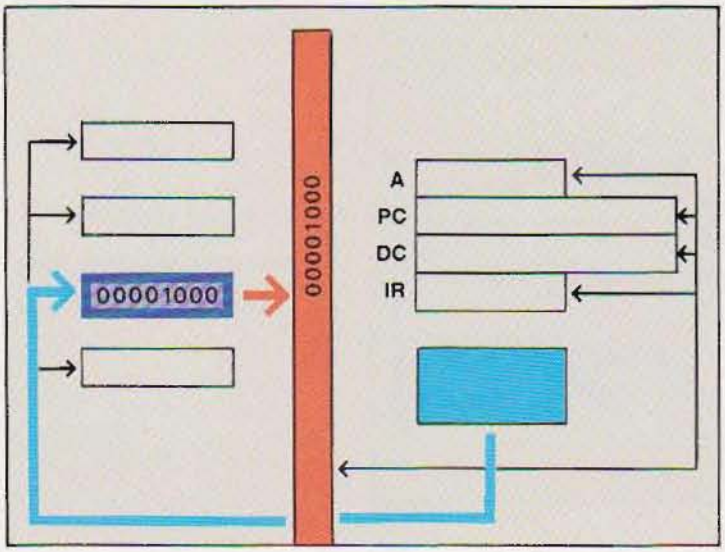
2 L'unité de contrôle décode et reconnaît l'instruction.
Le premier signal de contrôle active le transfert du contenu de l'accumulateur sur le bus interne de données



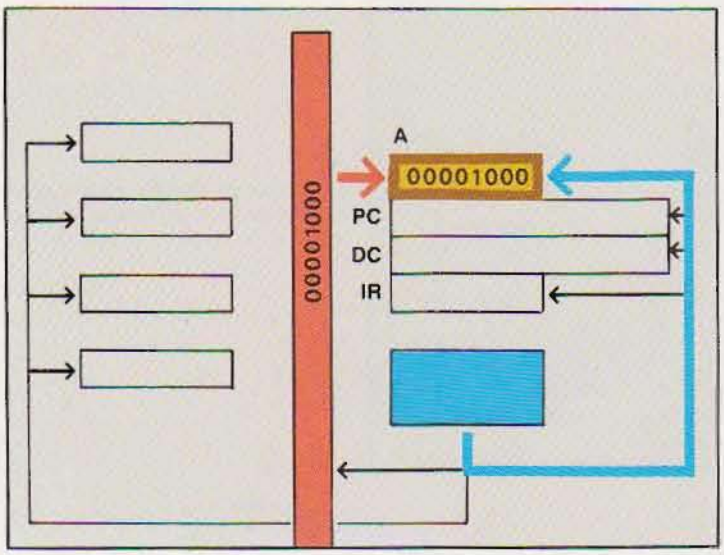
3 Le signal de contrôle suivant active le transfert de la donnée du bus vers le circuit de décalage (interne à l'UAL)



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)

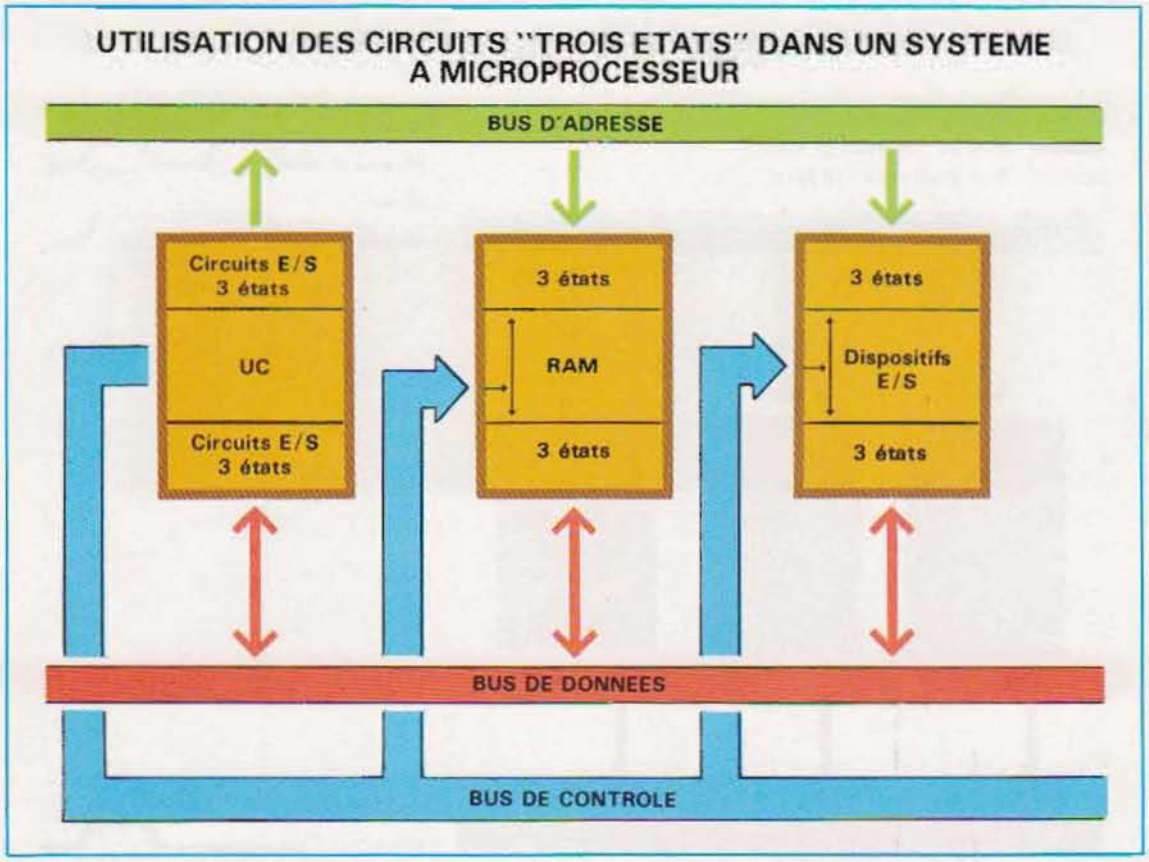
L'exécution des instructions

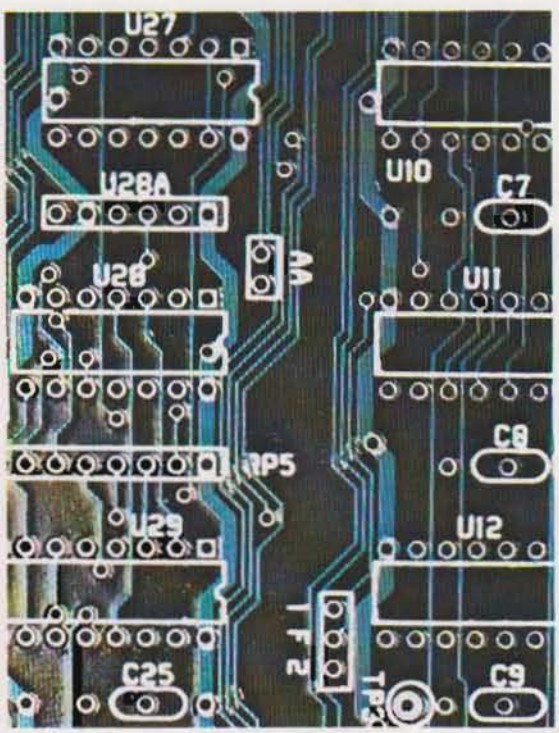
La phase d'exécution des instructions est peut-être celle qui utilise la plupart des ressources internes d'un microprocesseur. Elle est aussi celle qui échappe le plus à une logique simple de déroulement.
La lecture des paragraphes qui suivent demandera une attention particulière : la compréhension des 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-





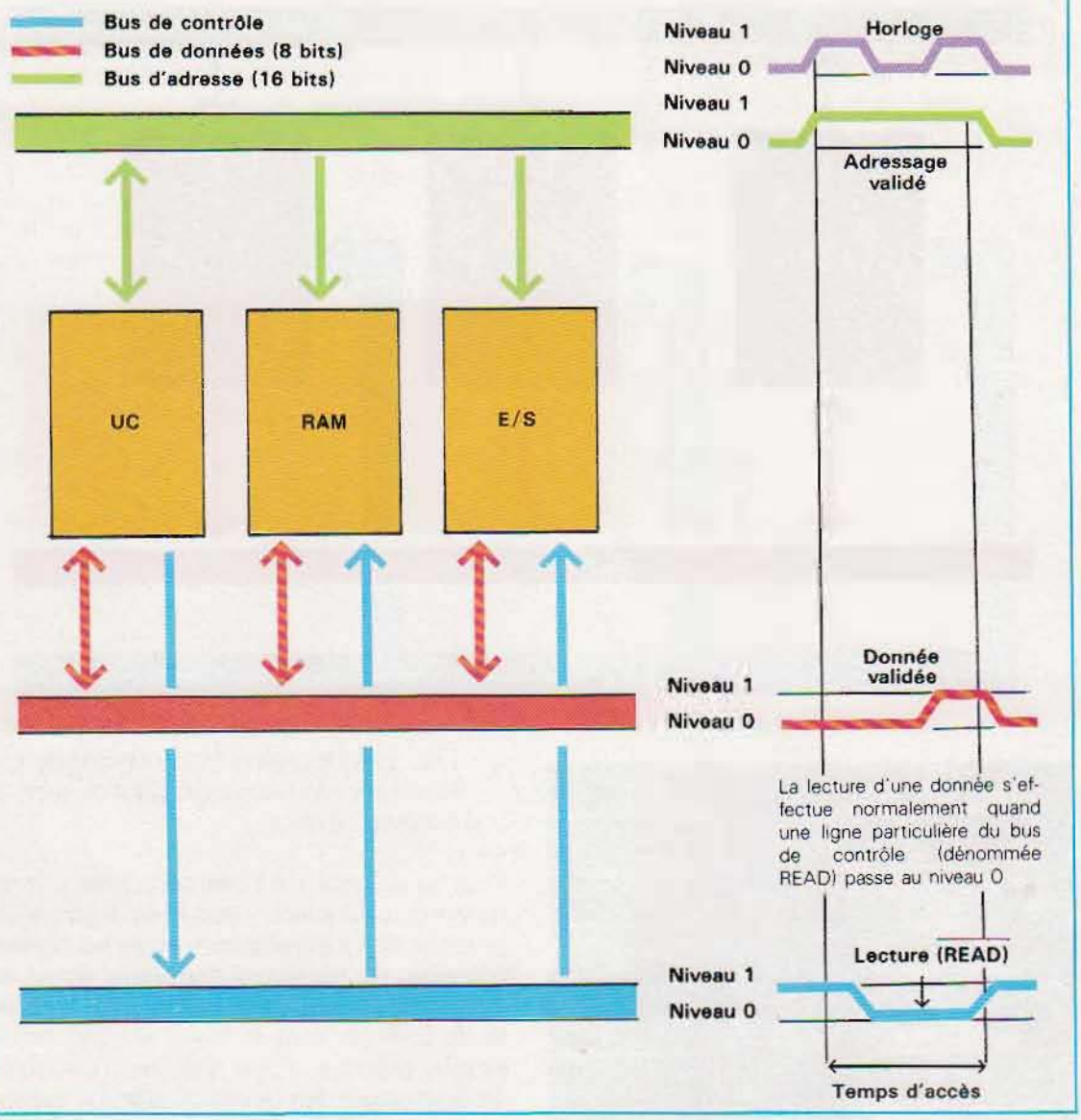
Gros plan d'un circuit imprimé.

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.

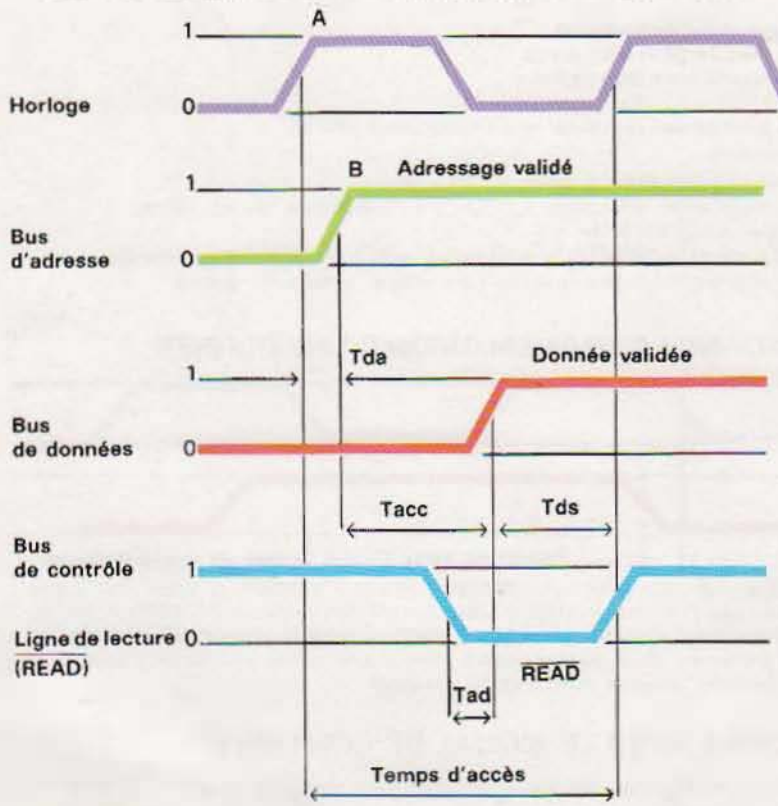
En réalité, le déroulement de ces opérations est beaucoup plus complexe. La synchronisation des signaux est un des paramètres les plus critiques dans la conception d'un système à base de microprocesseur. Les temps d'accès (voir schéma à droite) varient, par exemple, en fonction de la température. Les 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

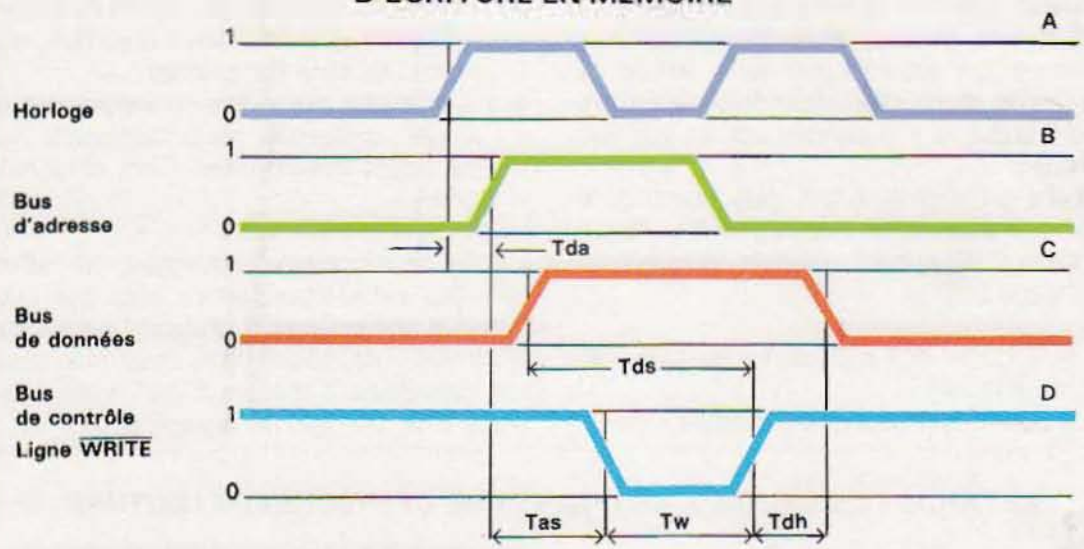
La mémoire ainsi adressée place son contenu sur le bus de données. Cette opération a aussi besoin d'un délai (Tacc) ; la donnée est donc validée en retard par rapport à l'apparition de l'adresse

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

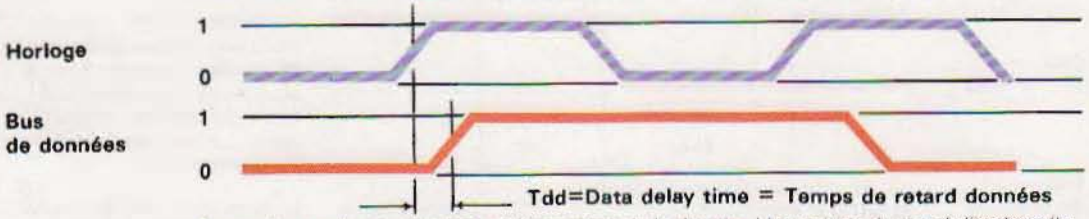
A L'horloge active l'opération d'écriture

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

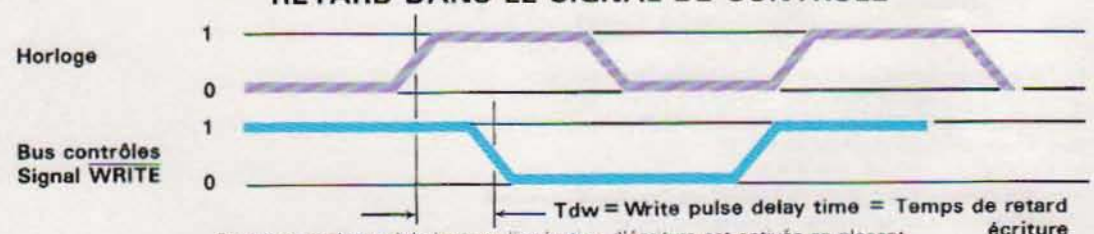
D En dernier lieu, arrive l'impulsion d'écriture (WRITE) ; la mémoire adressée au point B reçoit la donnée. A la fin du WRITE, la donnée reste disponible pour le temps Tdh ; ensuite, le cycle est terminé

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)

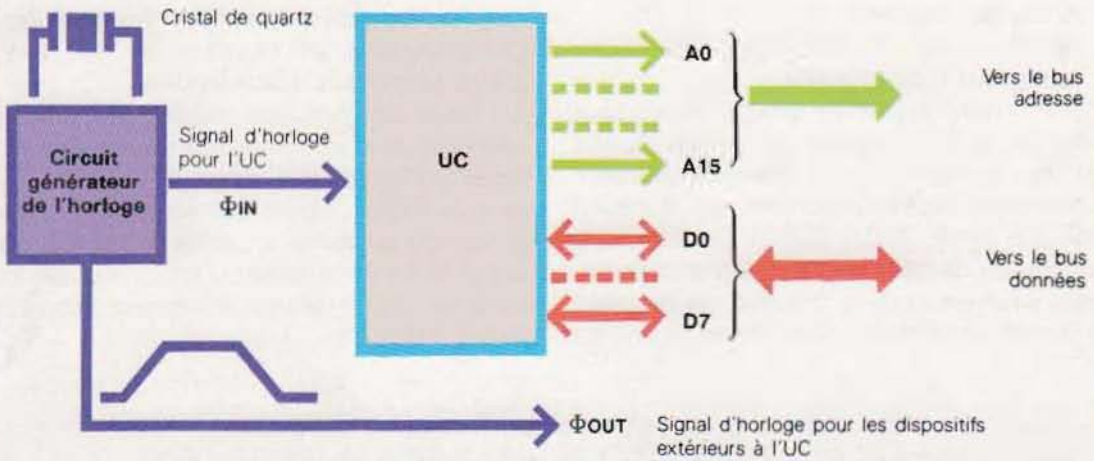
RETARD DANS LE SIGNAL DE CONTROLE



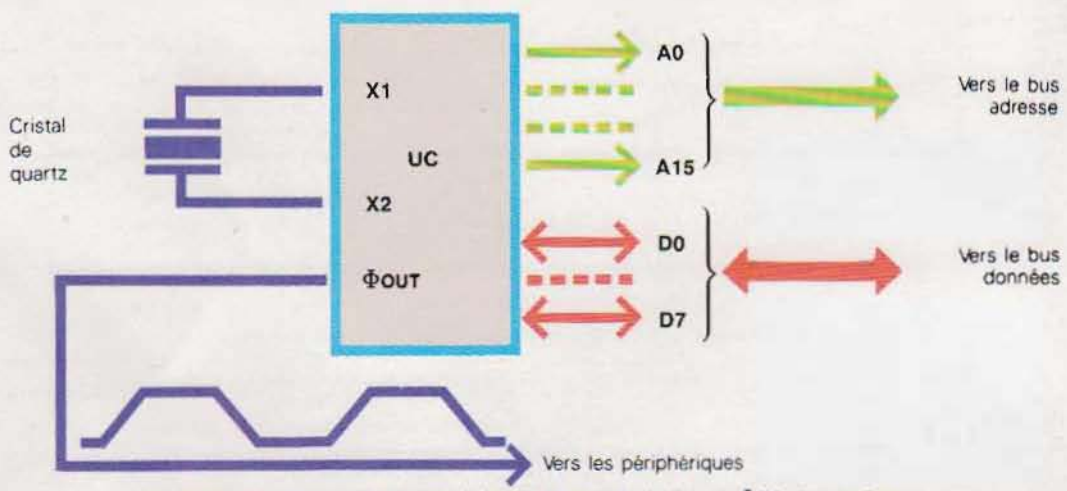
De façon analogue à la lecture, l'opération d'écriture est activée en plaçant à zéro le niveau d'une ligne particulière appelée WRITE. Le temps de retard de cette impulsion relative à l'horloge est indiqué par Tdw

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



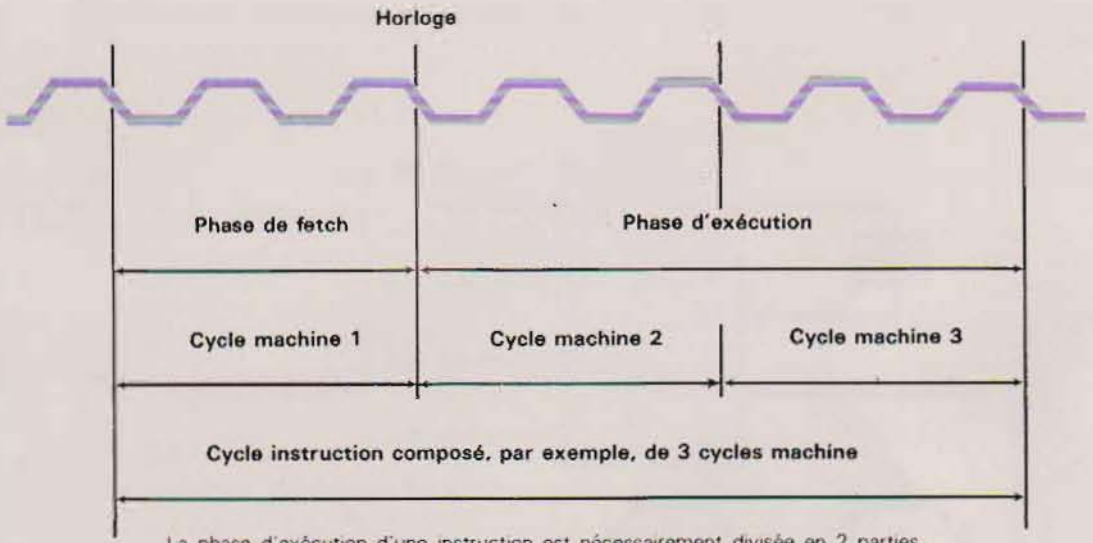
Oscillateur d'horloge intégré dans l'UC. Dans ce cas, la liaison ΦIN n'a pas lieu puisque le signal correspondant est interne à l'UC. De plus, les deux liaisons X1 et X2 du stabilisateur à quartz arrivent sur l'UC

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

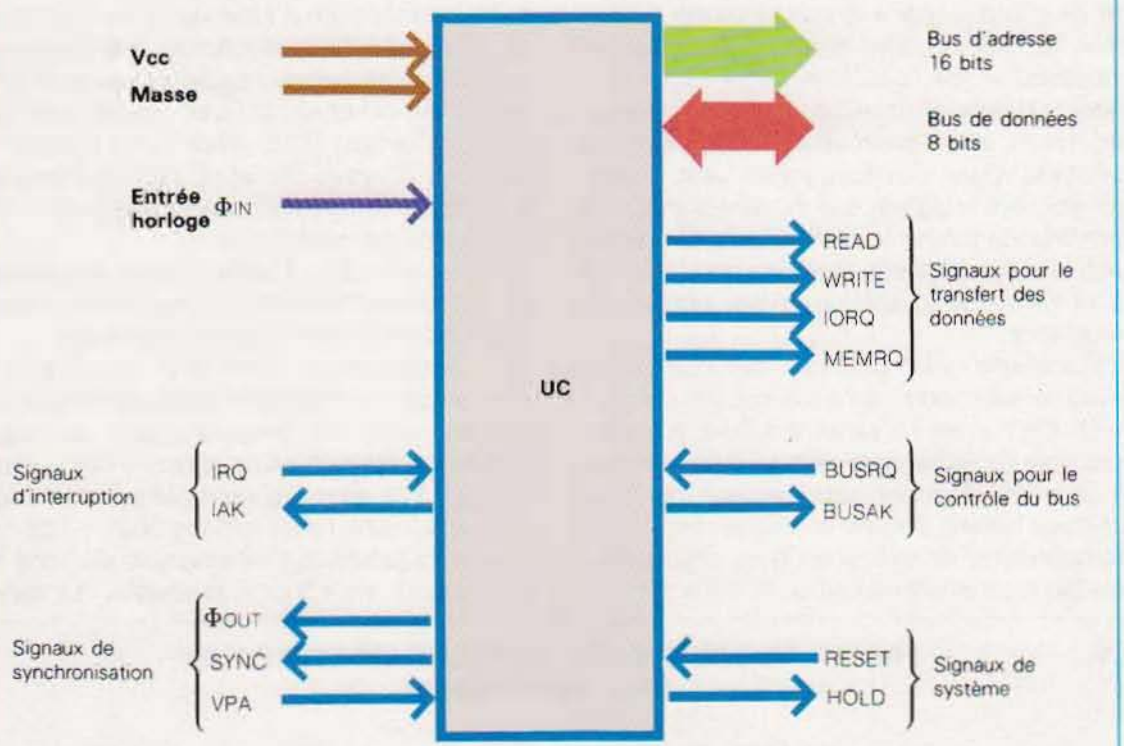
Appartiennent au premier groupe, les signaux spécifiant que l'opération de lecture (READ) ou d'écriture (WRITE) doit être effectuée vers la mémoire (MEMRQ) ou vers un dispositif général d'entrée/sortie (IORQ). On a recours aux signaux de contrôle de bus quand un dispositif externe exige le transfert 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

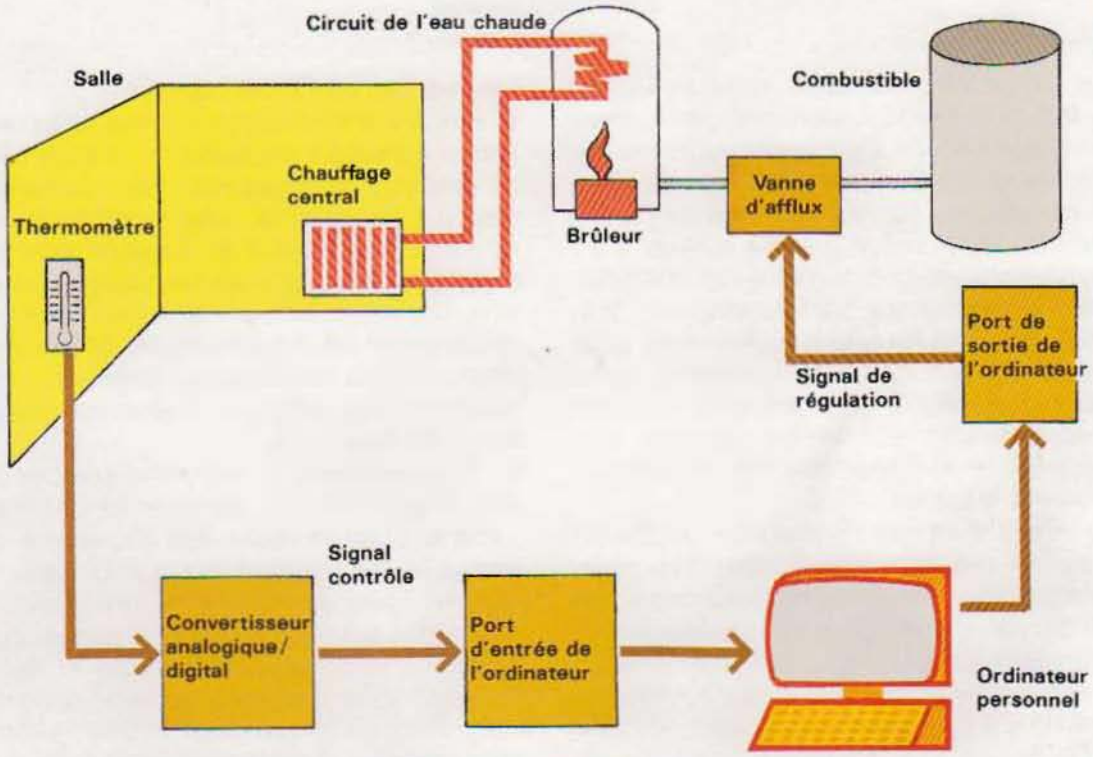
comparera donc cette valeur avec la température souhaitée. En fonction du résultat, il enverra les données nécessaires à un dispositif de contrôle (par exemple la vanne d'émission du combustible dans la chaudière de l'installation de chauffage) pour réguler ou ajuster la température. Le schéma ci-dessous 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).

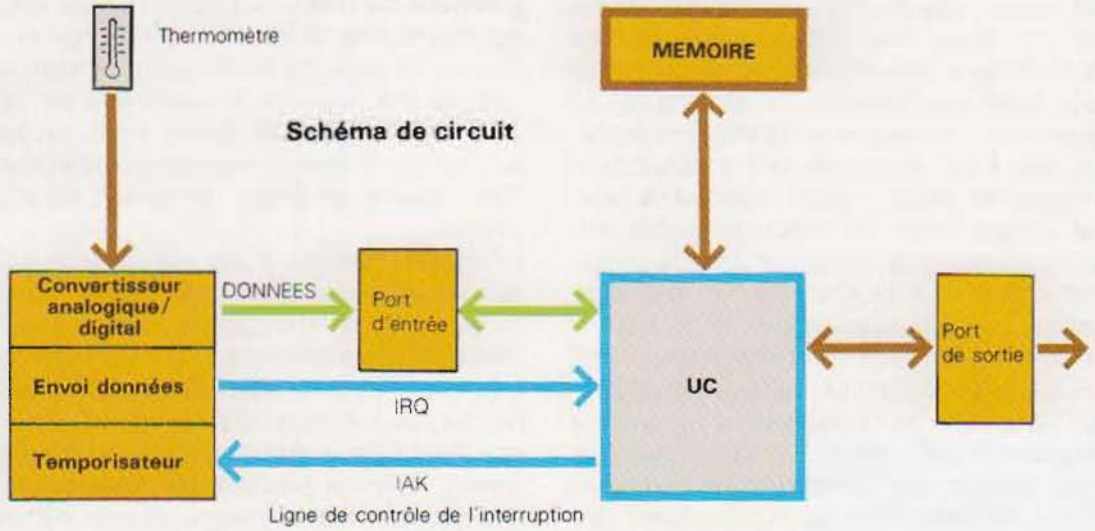
Un temporisateur (« timer ») associé au convertisseur analogique-digital engendre, à des intervalles de temps imposés, une demande d'interruption plaçant à la valeur logique 1 la ligne de contrôle appelée IRQ (voir aussi schéma de la page 903). L'UC reconnaît la demande d'interruption, suspend le programme en cours d'exécution, se rend

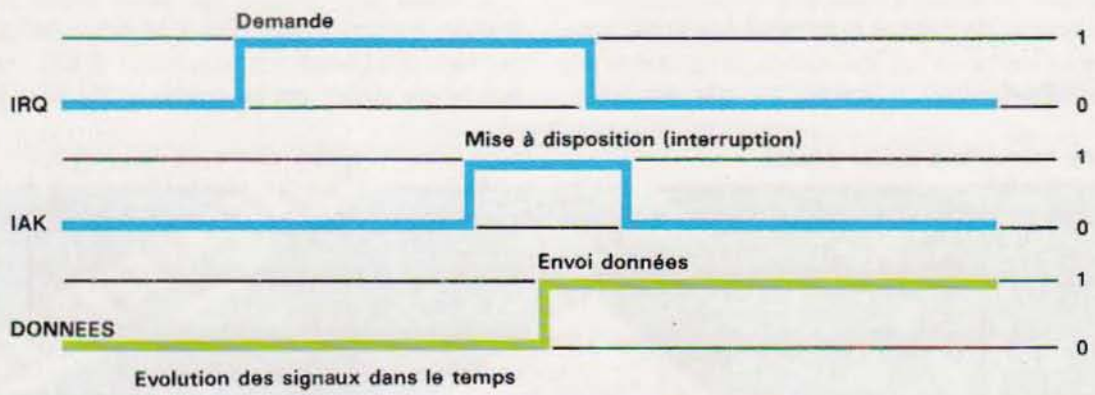
REGULARISATION DE TEMPERATURE EFFECTUEE PAR ORDINATEUR



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





Evolution des signaux dans le temps

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.

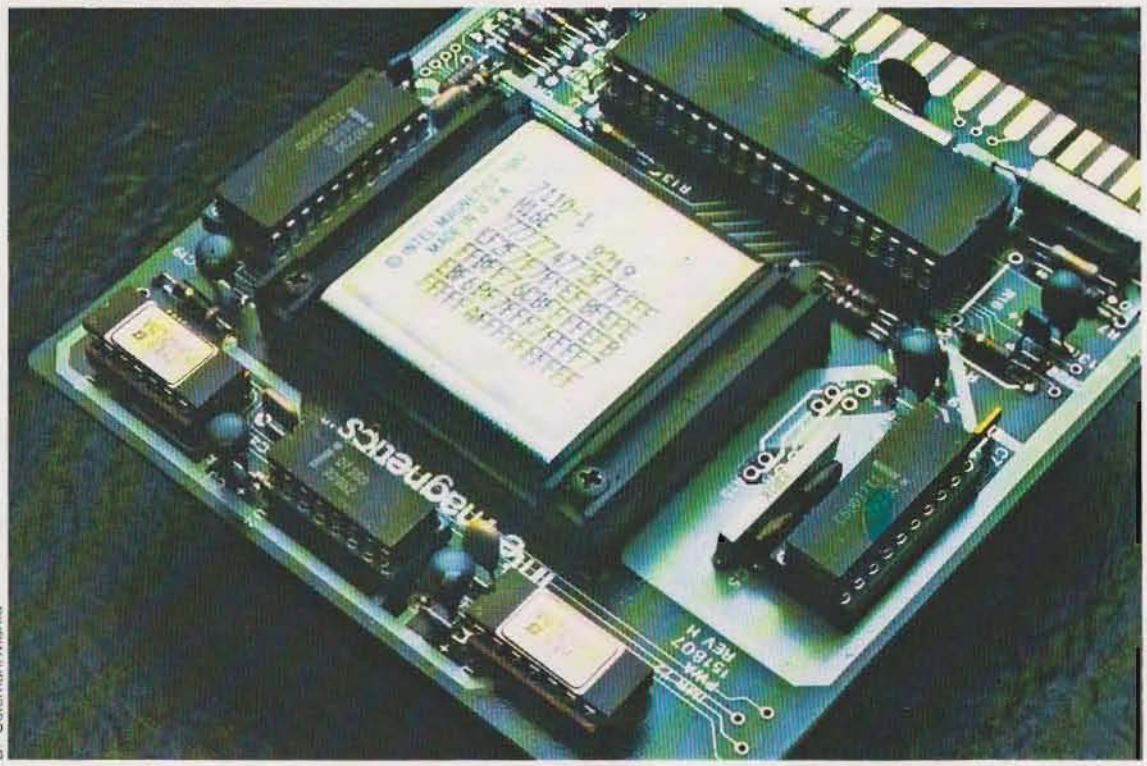
Le contenu du registre d'instructions n'est pas sauvegardé car l'UC achève l'exécution de l'instruction en cours avant d'accepter la demande d'interruption. Par conséquent, le code contenu dans l'IR correspond à une exécution déjà effectuée, qu'il n'est donc pas nécessaire de sauvegarder.

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.

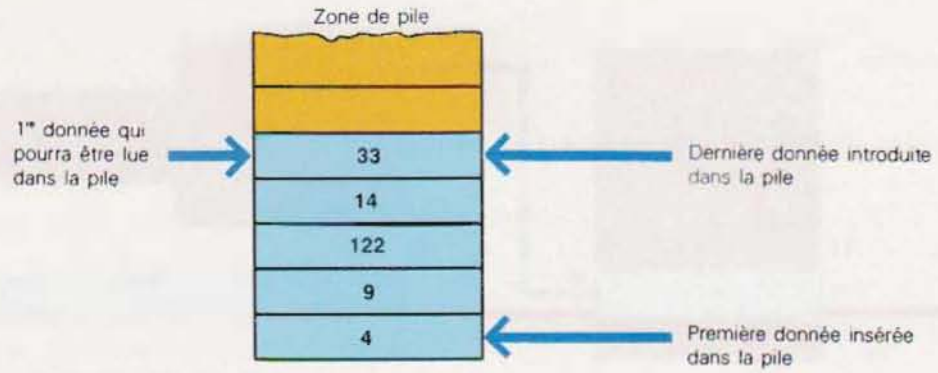
Sur la page ci-contre (au centre), est montré le schéma de l'UC à la réception d'une demande d'interruption. Dans le pointeur est contenu l'adresse hexadécimale 0100, qui pointe au début de la mémoire de la pile.

Une mémoire à bulles d'Intel.

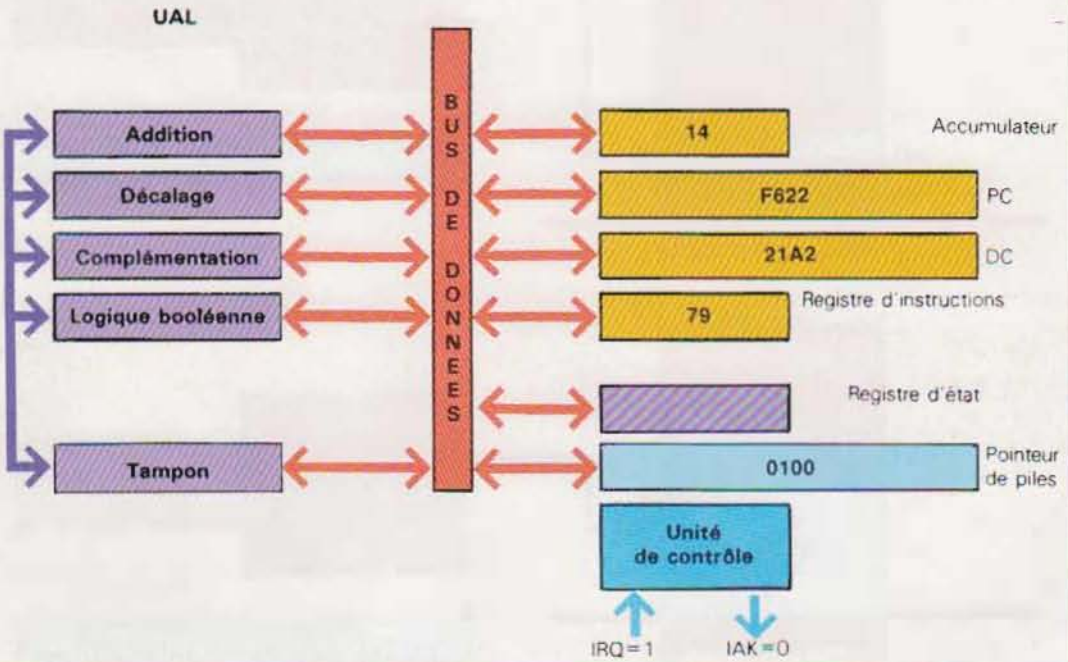


B. Coleman/Marka

STRUCTURE D'UNE PILE TYPE LIFO



ETAT DE L'UC A LA RECEPTION D'UNE DEMANDE D'INTERRUPTION

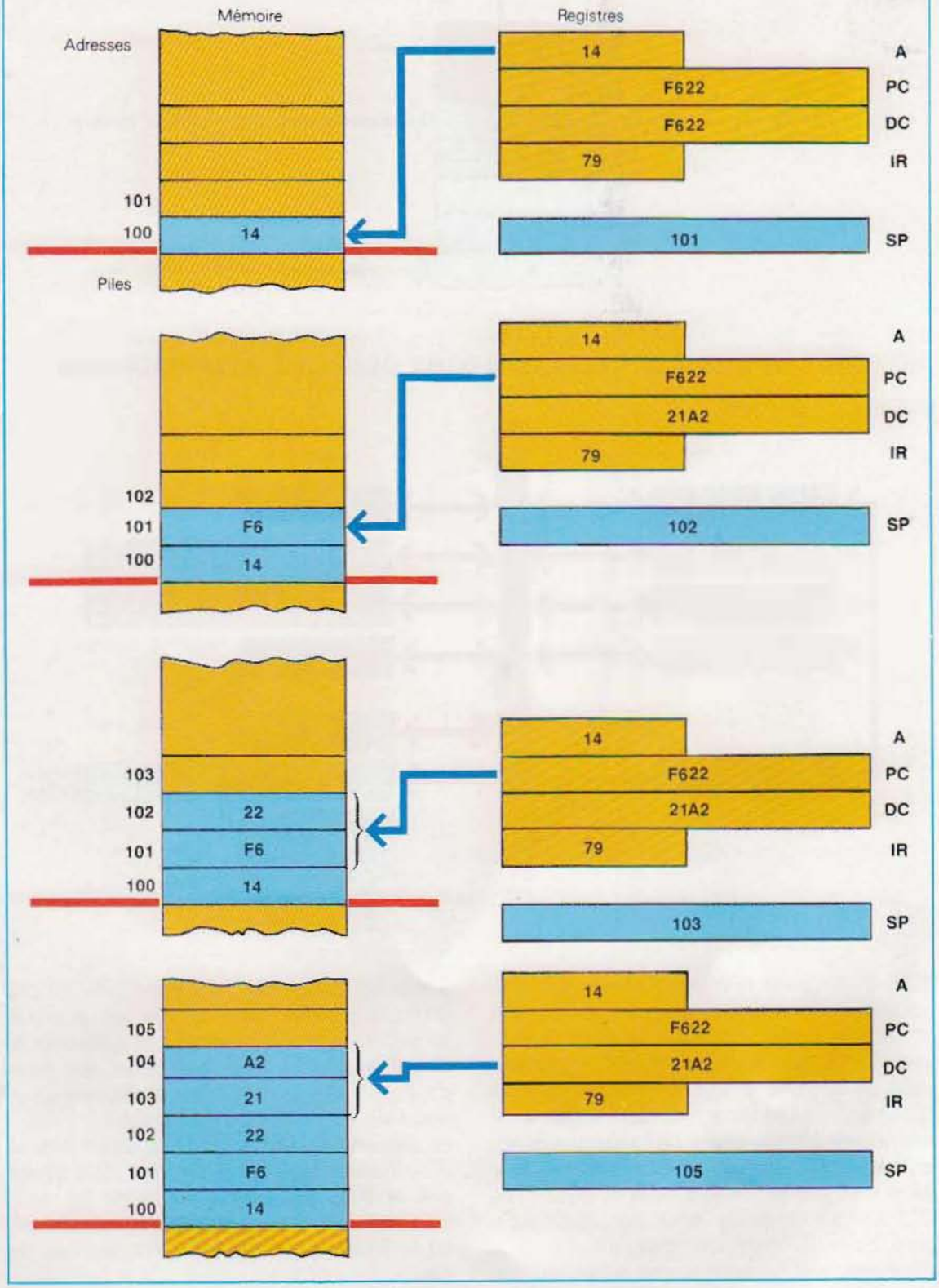


L'UC sauvegarde le contenu des registres à partir de cette adresse, place à 1 la ligne IAK et accepte l'interruption. L'opération de sauvegarde est détaillée en page 908. L'UC sauvegarde le contenu du premier registre dans la pile en écrivant la donnée dans la cellule de mémoire adressée par le pointeur 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.

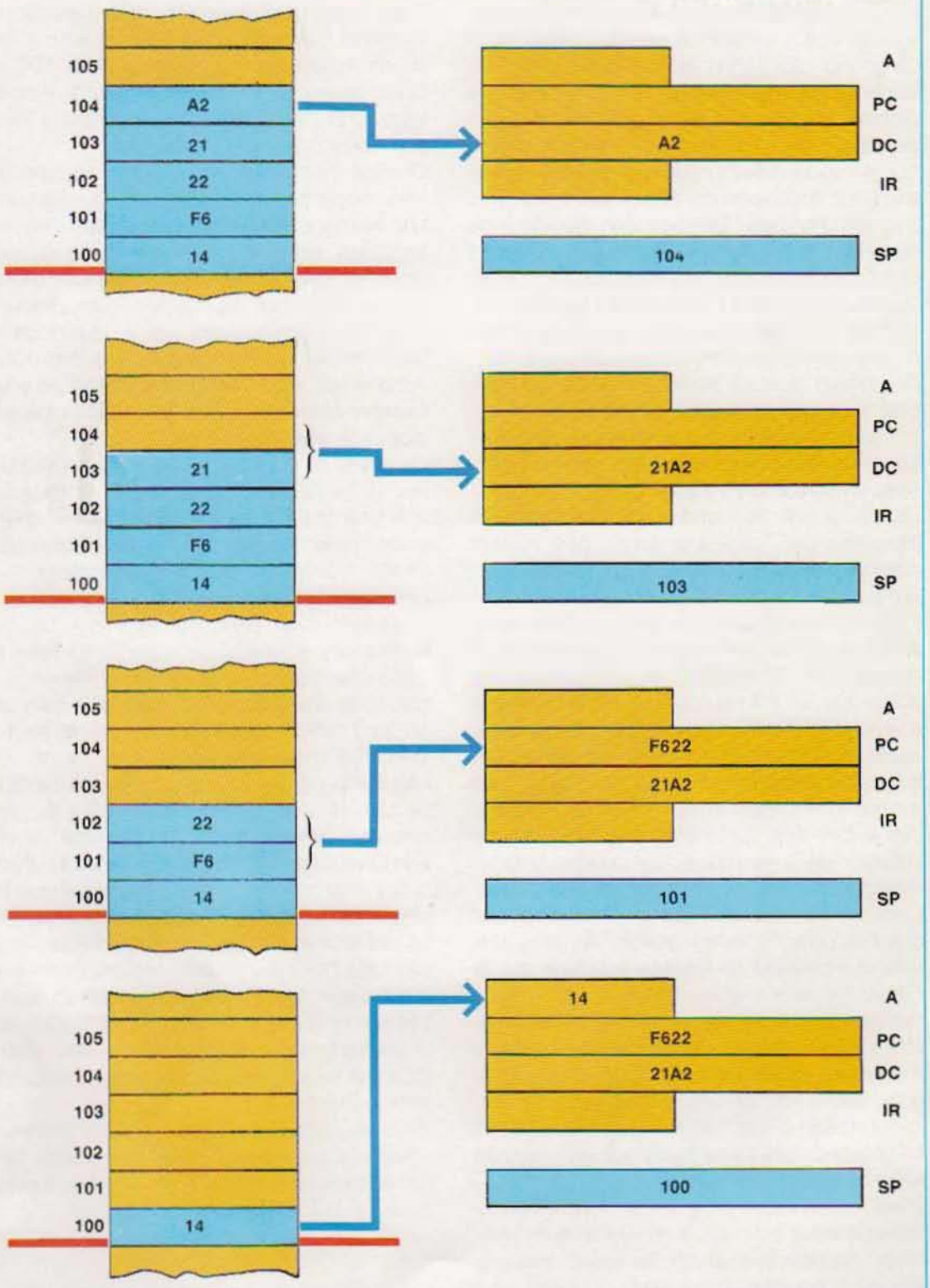
Pour réactiver le programme suspendu, on suit la logique inverse (voir page 909). L'opération d'écriture dans la pile se poursuit jusqu'à saturation. Dans certains systèmes, la pile a des dimensions fixes alors que dans d'autres systèmes elle est extensible et varie selon les exigences du programme.

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.

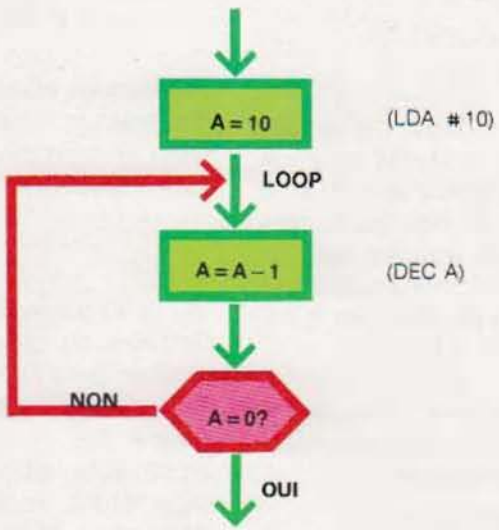
Le champ commentaire est utilisé à seule fin de documenter le programme. Sur la page ci-contre, une partie de programme en Assembleur illustre la syntaxe employée. La première instruction « charge » (mnémonique LD pour load) la valeur 10 dans l'accumulateur. La deuxième instruction décrémente (mnémonique 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.

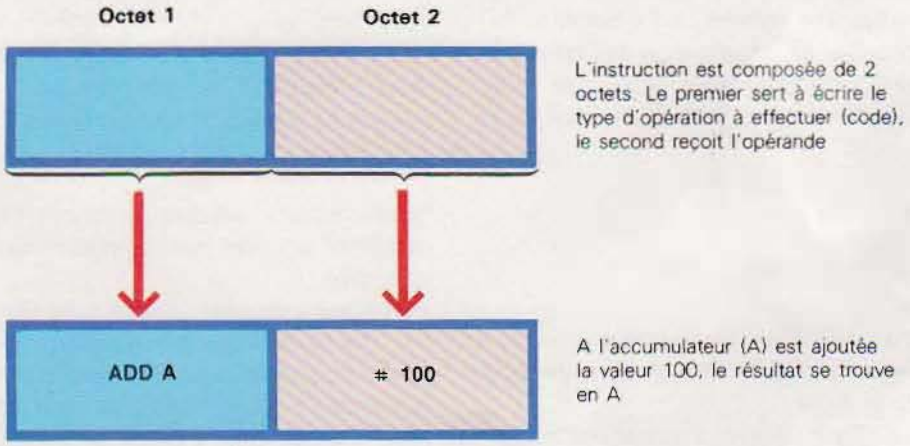
Adressage absolu ou direct. Avec ce type d'adressage, l'instruction contient l'emplacement en mémoire de l'opérande. Par exemple, l'instruction

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.

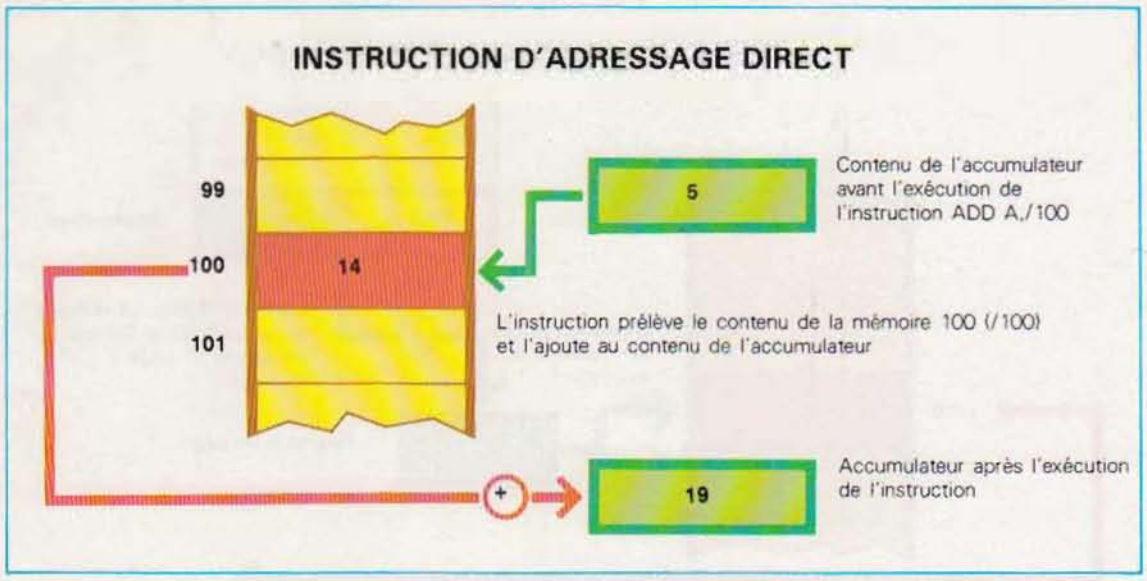
*Les codes mnémoniques qui sont utilisés dans les exemples ont un caractère indicatif. Pour s'en servir réellement, il faut les réécrire dans la forme prévue pour le type particulier de microprocesseur utilisé.

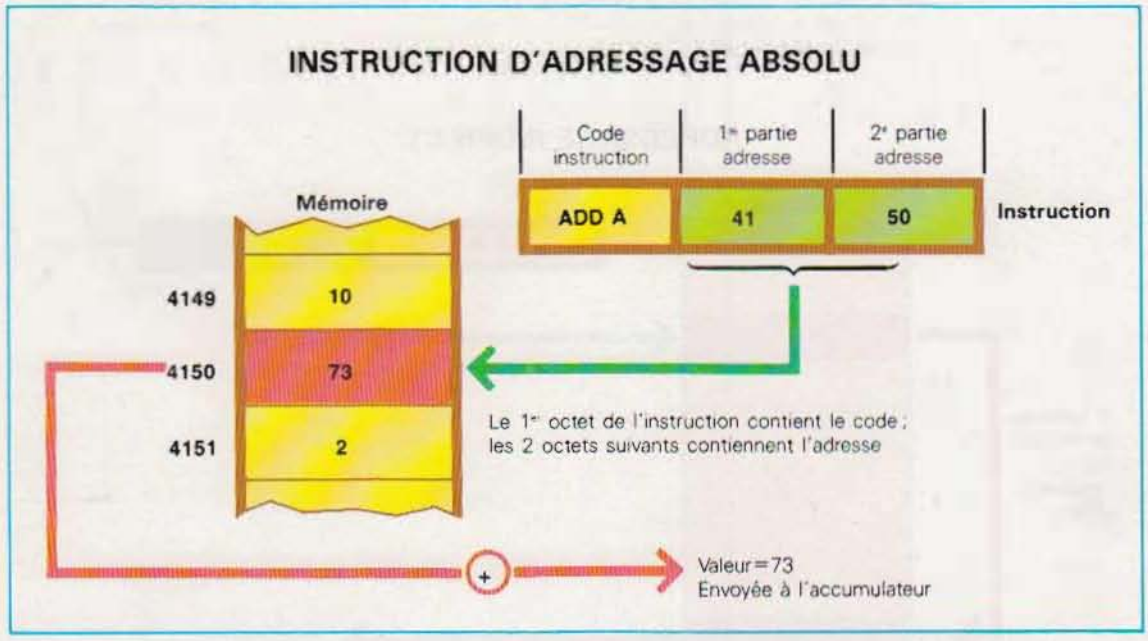
INSTRUCTION AVEC ADRESSAGE IMMEDIAT



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

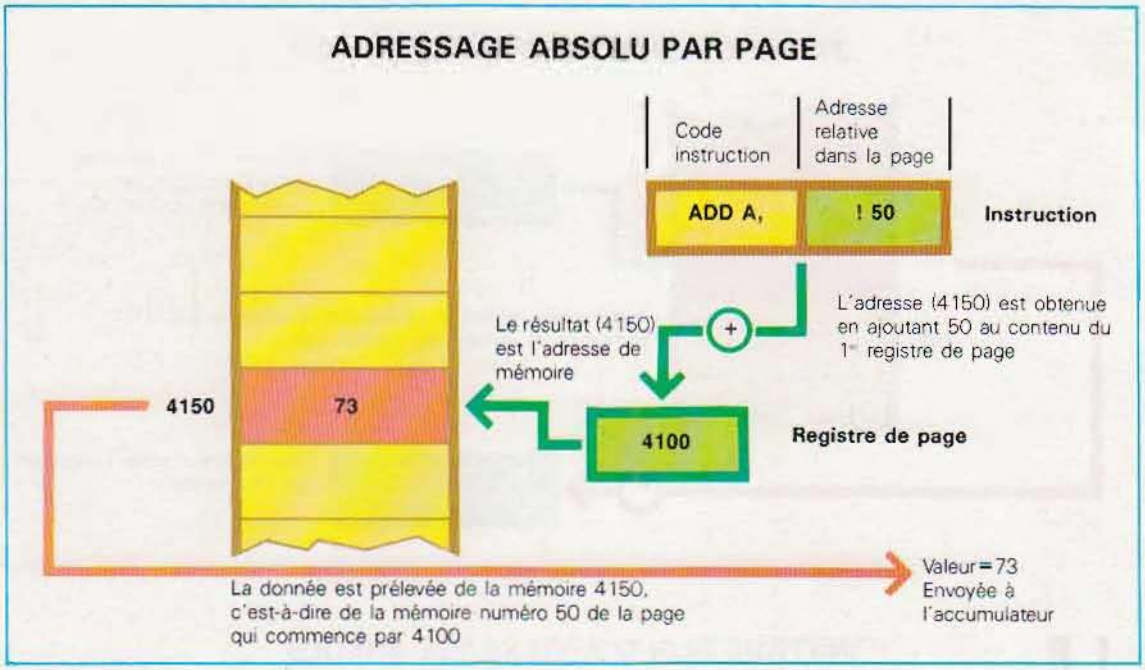


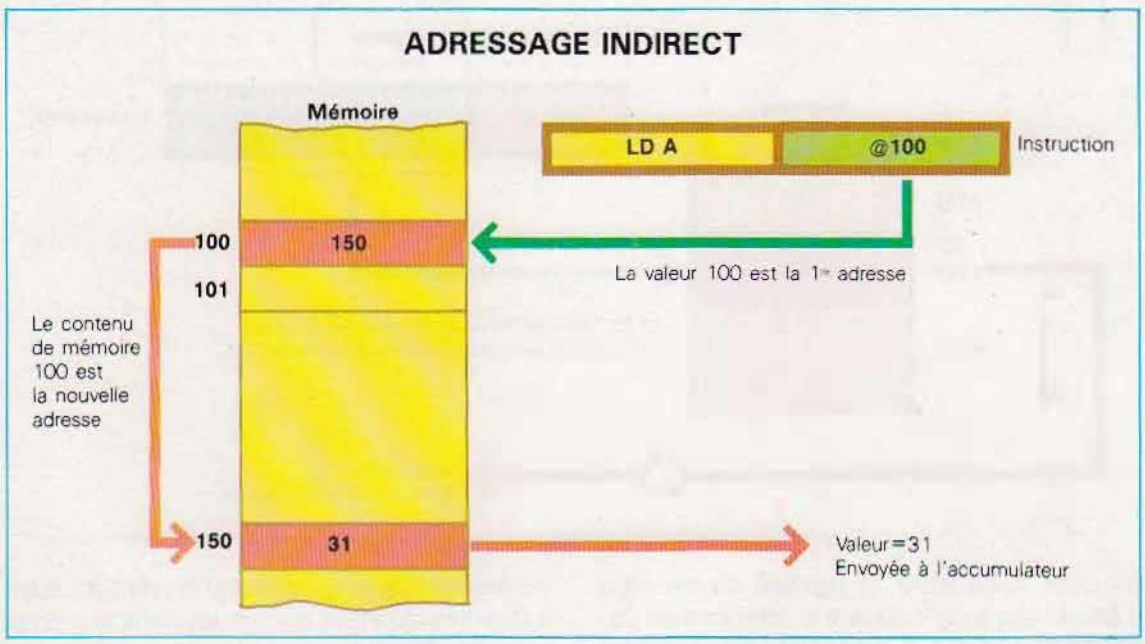


On peut augmenter la rapidité de ce type d'adressage en découpant la mémoire en pages de 256 octets. L'instruction adresse alors un des 256 octets d'une **page**. A cet effet, il existe un registre de page (**Page Register**). Il contient l'adresse de départ de la page qui, ajoutée à l'adresse (relative) 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.

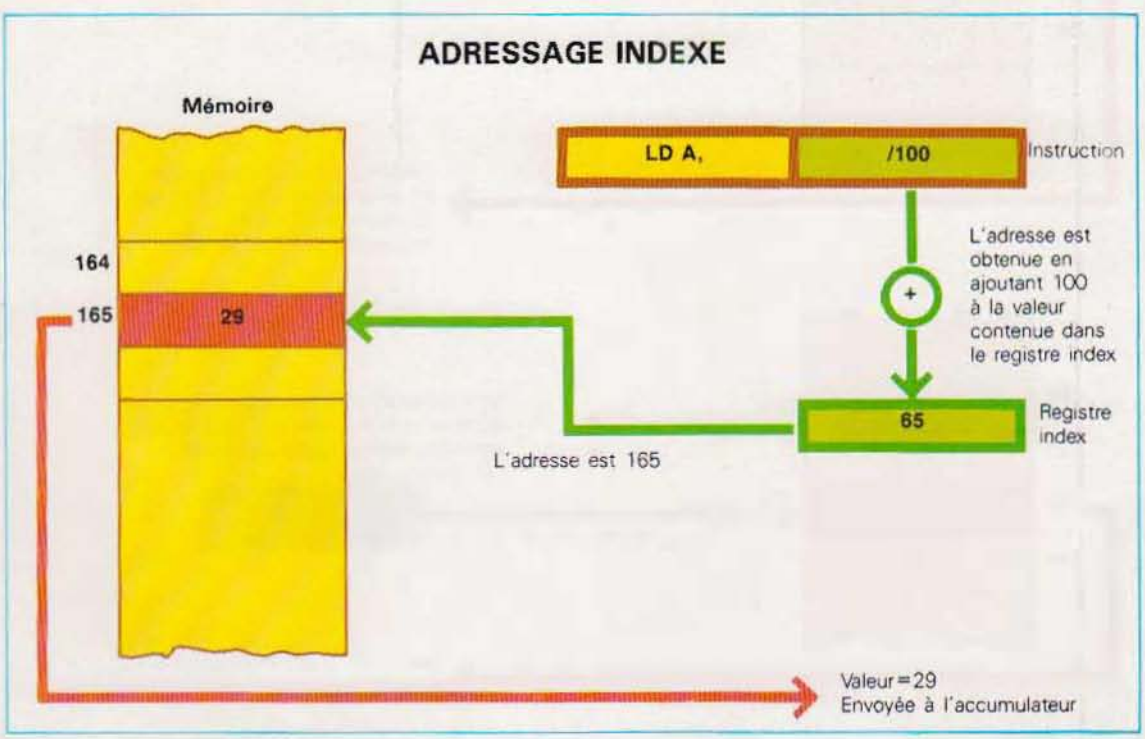
Adressage indexé. Dans ce mode, l'UC additionne l'adresse de base de l'instruction avec le contenu d'un registre particulier, appelé **registre index**, pour calculer l'adresse effective de l'opérande. Le programmeur charge au préalable le registre index (que nous appellerons X) avec la valeur qu'il désire utiliser pour l'indexation avec une instruction du type LD X=3. Lorsque l'instruction

LD A,100(X)

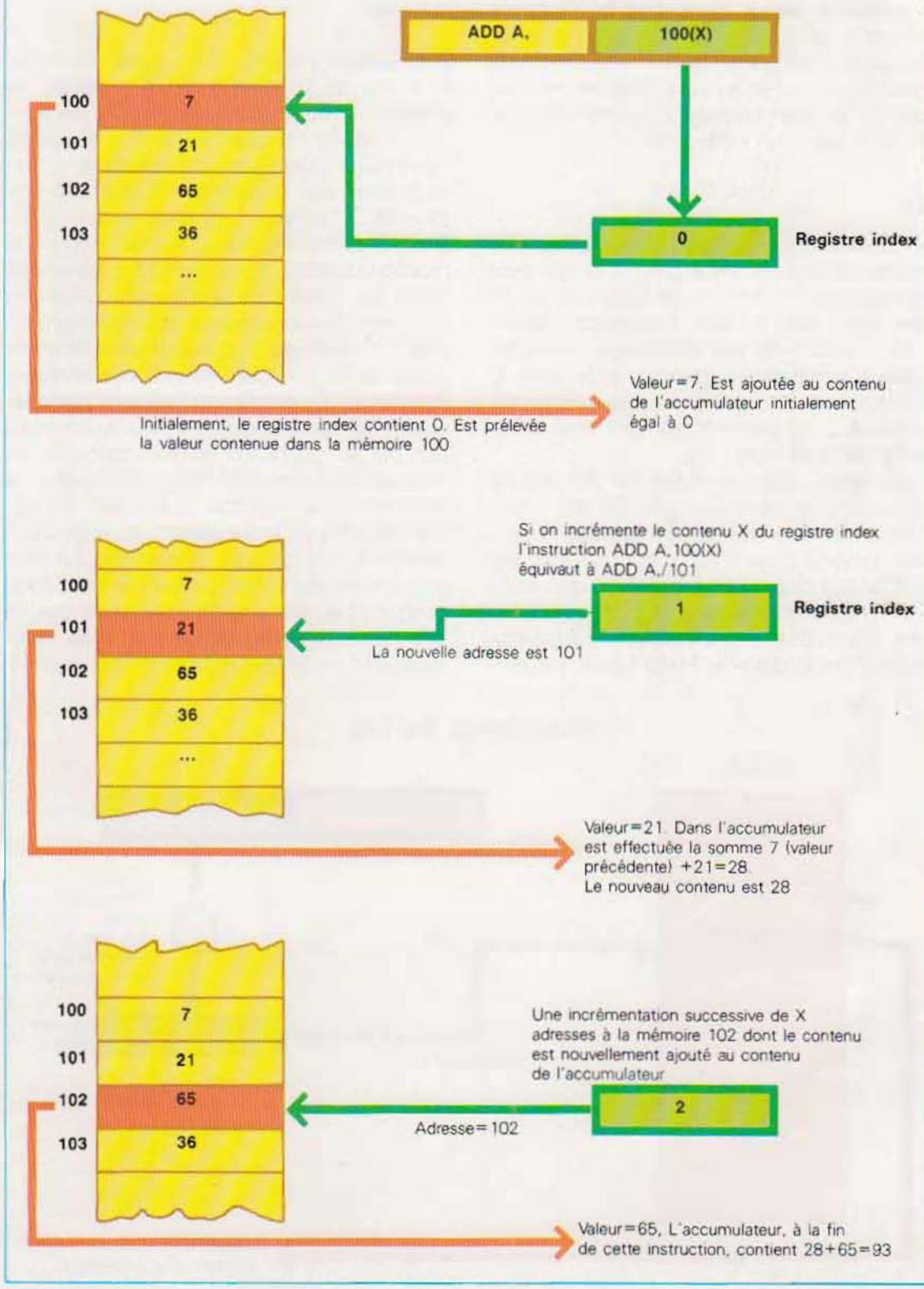
est exécutée, l'UC charge dans l'accumulateur la donnée contenue dans la cellule dont l'adresse est 100+X (voir schéma). Les parenthèses indiquent le mode d'adressage indexé. Il est très proche de l'adressage direct par page. A la différence du registre de page, le registre index ne contient pas nécessairement l'adresse d'une page de mémoire, mais n'importe quelle adresse.

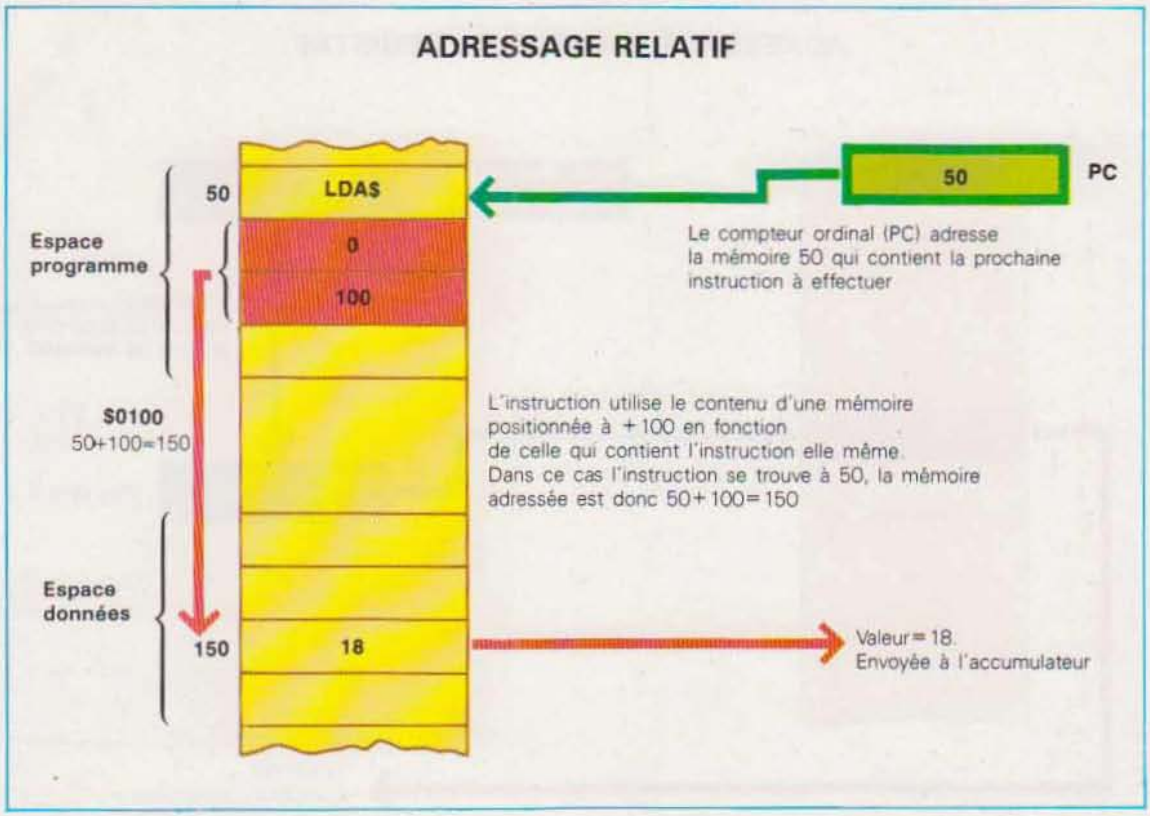
L'adressage indexé se révèle très utile pour le traitement des données organisées en mémoire sous forme de tableaux de matrices (array) - (voir schéma page 916). Il est plus lent que l'adressage direct parce que l'UC doit additionner le contenu du registre index et l'adresse de l'instruction avant d'accéder 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





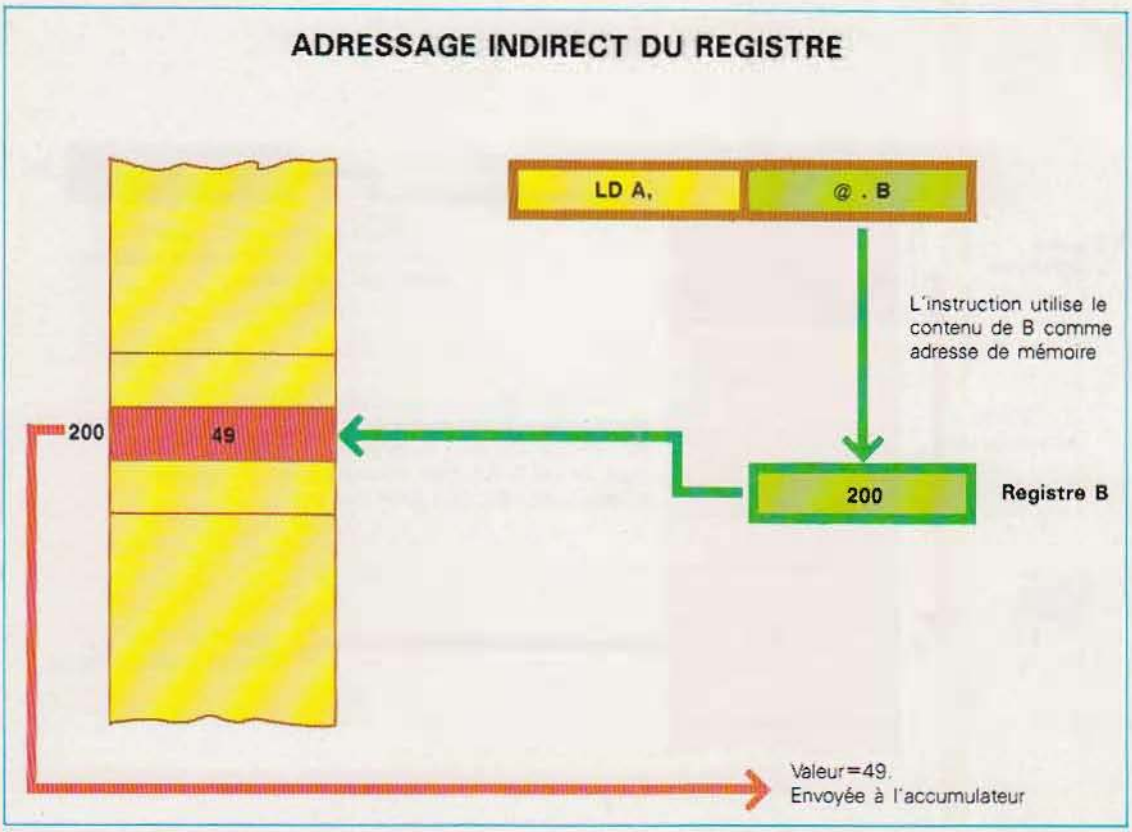
Adressage par registre. On a un adressage par registre quand l'instruction spécifie un registre. Les registres sont contenus dans l'UC et sont en nombre limité.
Cette méthode d'adressage permet d'accroître la vitesse d'exécution des instructions et de réduire la longueur du code. Par exemple, l'instruction

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.

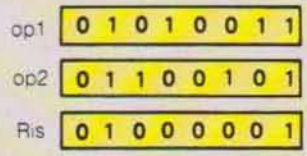
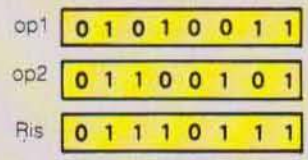
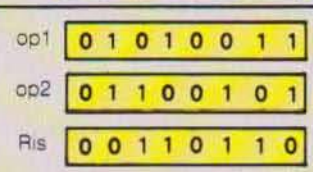
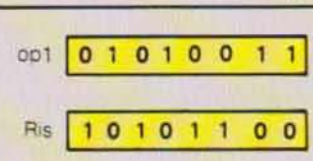
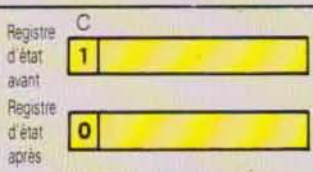
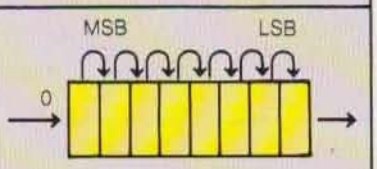
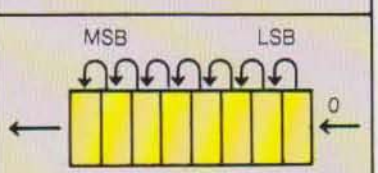
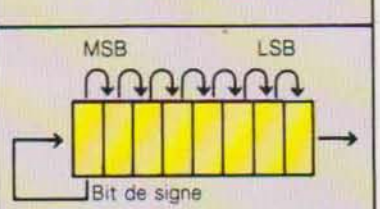
Instructions de transfert des données

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) :

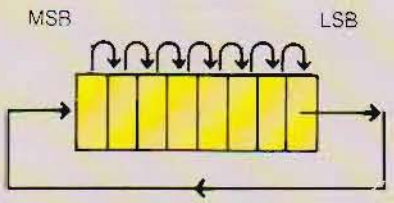
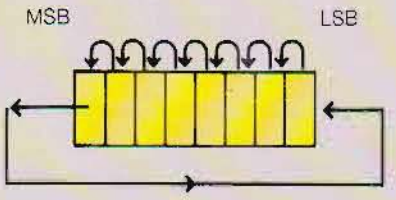
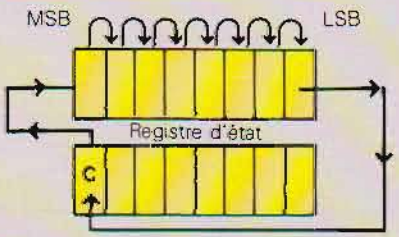
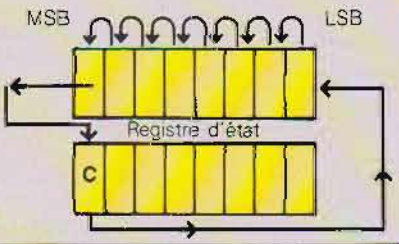
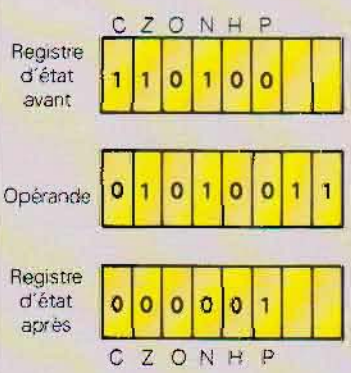
Code instruction	Première opérande	Seconde opérande
LD	A,	0

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).

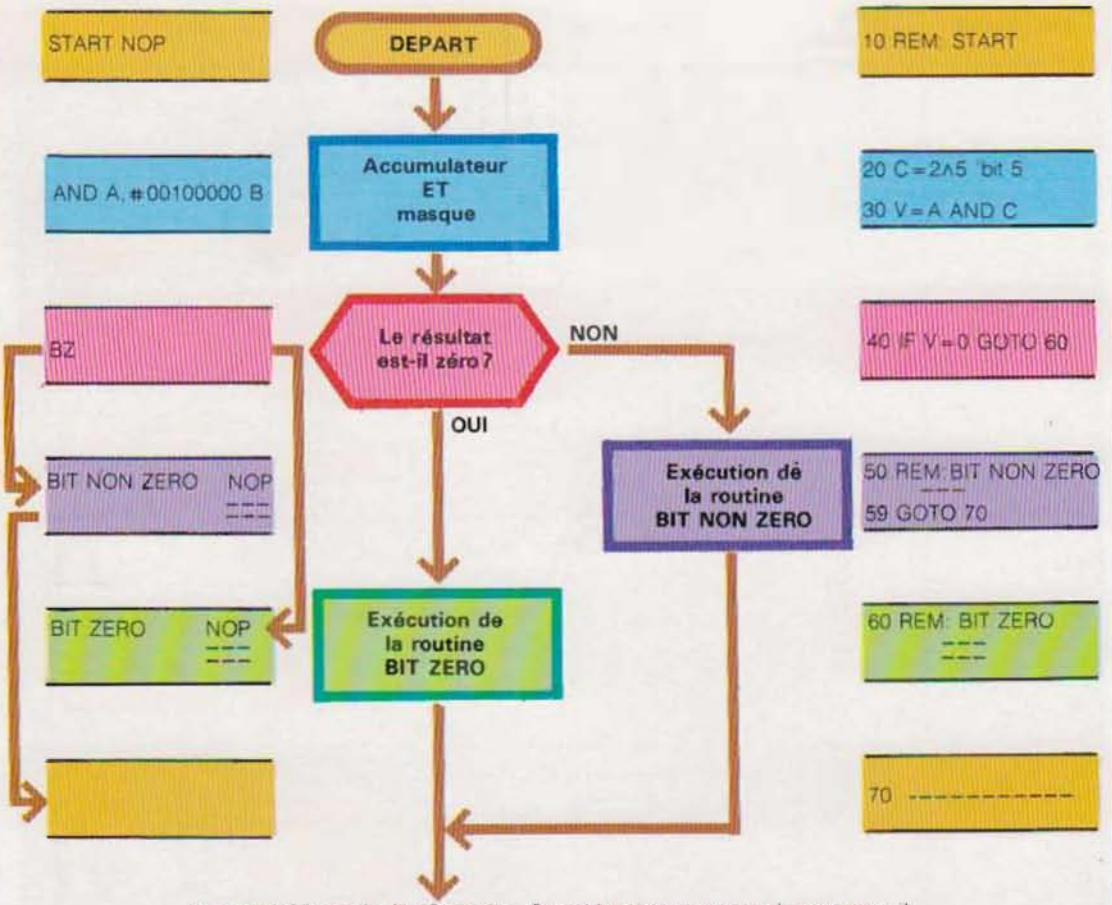
INSTRUCTIONS LOGIQUES

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)

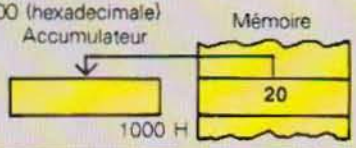
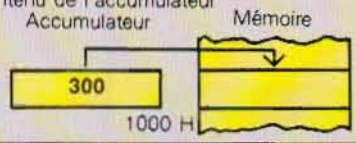
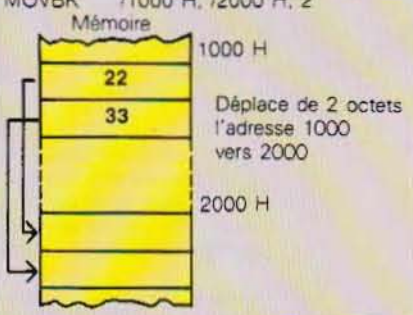
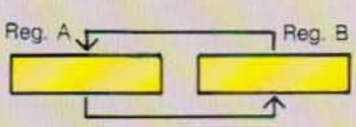
transfère le contenu de l'accumulateur dans la case mémoire d'adresse hexadécimale*

* Le nombre hexadécimal est signalé par la lettre H.

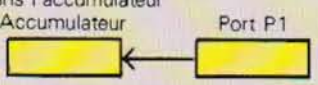
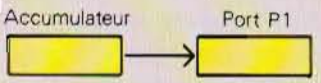
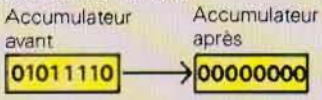
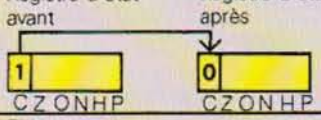
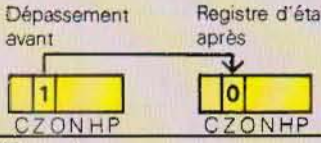
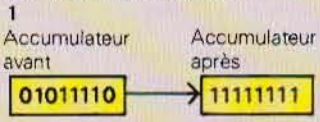
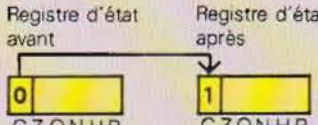
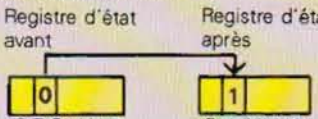
100 (adressage absolu). Les principales instructions de transfert sont résumées en pages 926 et 927.

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.

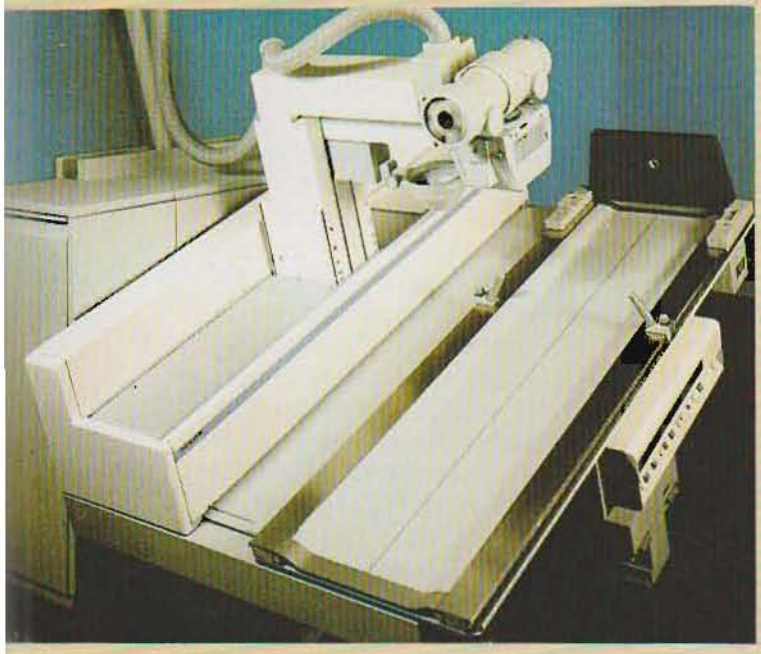
Les possibilités de mouvement du système lors de l'examen sont considérables : examens radiologiques de l'appareil gastro-intestinal et tomographie plane-parallèle (c'est-à-dire effectuée sur un plan parallèle à celui de la table).

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**.*

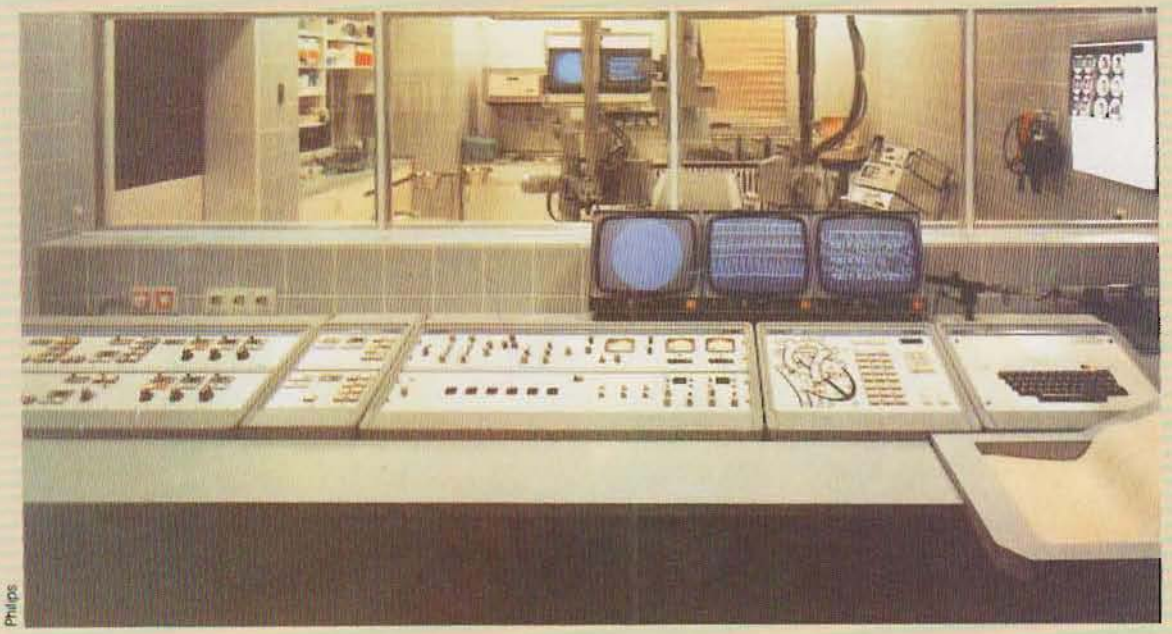
Les deux systèmes angiographiques de la dernière génération sont représentés par le Poly-Diagnost UPI pour l'angiographie générale et par le Poly-Diagnost C, spécialement conçu pour l'angiocardiographie.

*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.

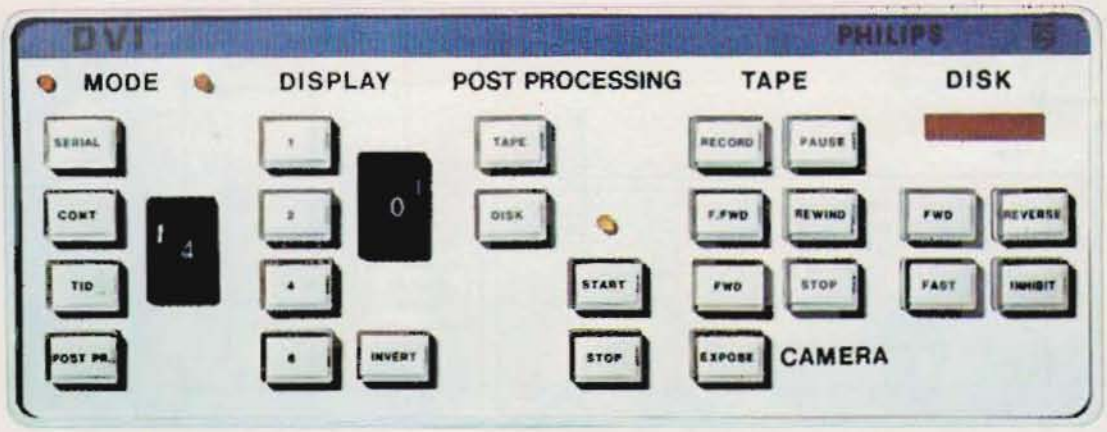
Le "Time interval difference mode" (TID), dernier des trois algorithmes, consiste à traiter les images à des intervalles constants par le Continuous imaging mode (voir schéma Ⓒ, page 949).

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.*

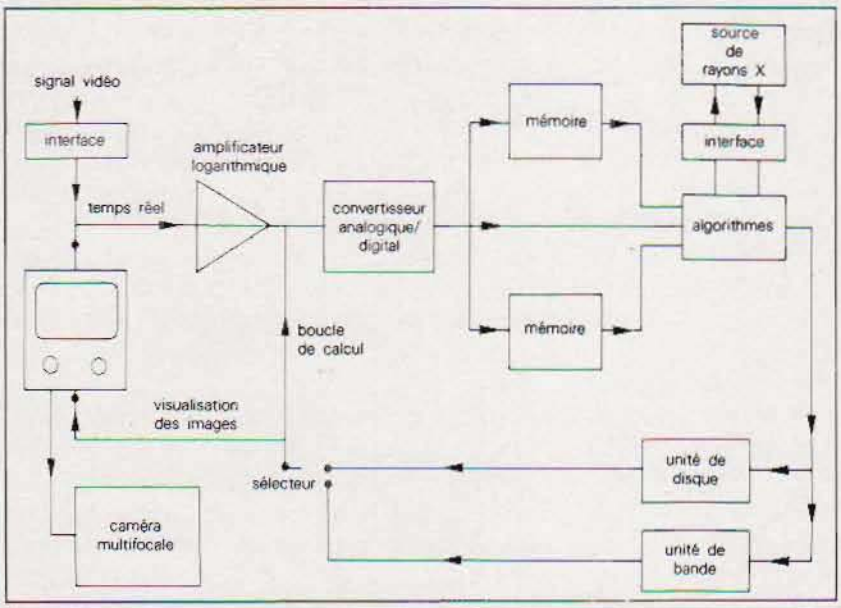
*Notons enfin que le système digital accepte, en entrée, les images produites par d'****autres équipements radiologiques.*** *Ces propriétés en font un outil de très large diffusion scientifique dans un proche avenir.*

Tableau de commandes et de contrôle du système angiographique.

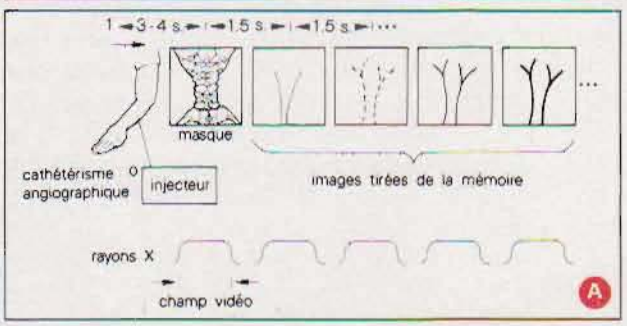


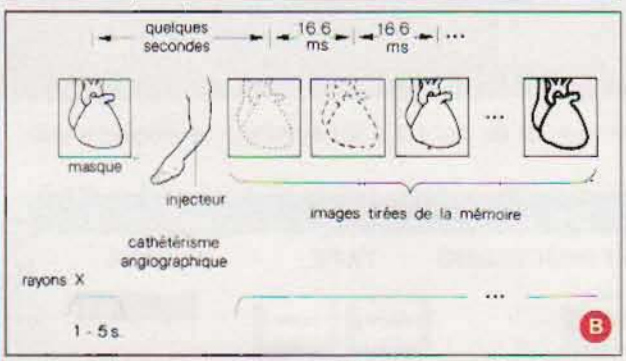
Philips

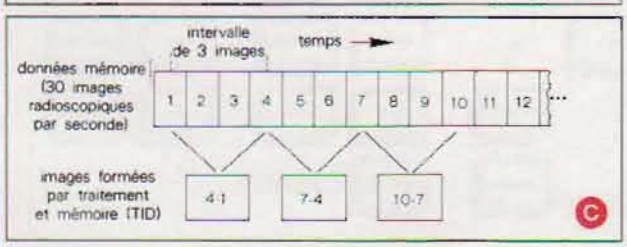
SCHEMA ET FONCTIONNEMENT D'UN SYSTEME DIGITAL (OU NUMERIQUE) ANGIOGRAPHIQUE

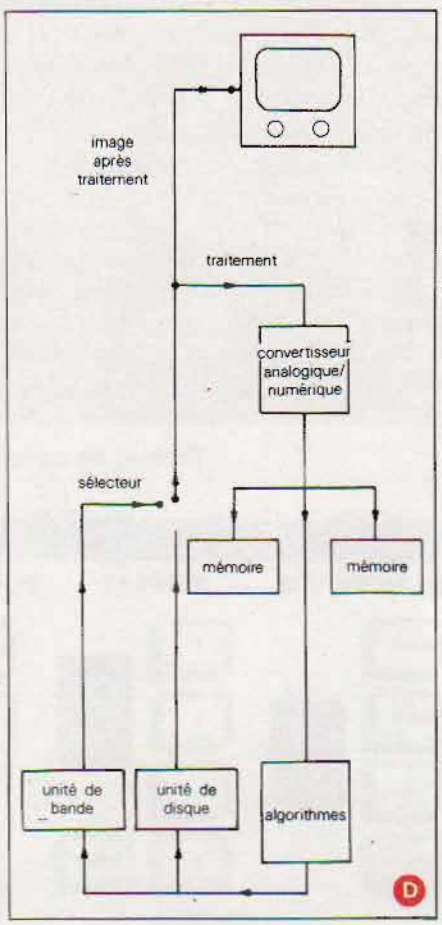


A gauche, on trouve le schéma de fonctionnement d'un appareil d'analyse du système vasculaire. Quatre modes d'examen sont utilisés :
En Ⓐ : "Serial imaging mode", en Ⓑ : "Continuous imaging mode" en Ⓒ, "Interval difference mode (TID), en Ⓓ : "Post processing mode" qui peut intervenir sur les images traitées 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

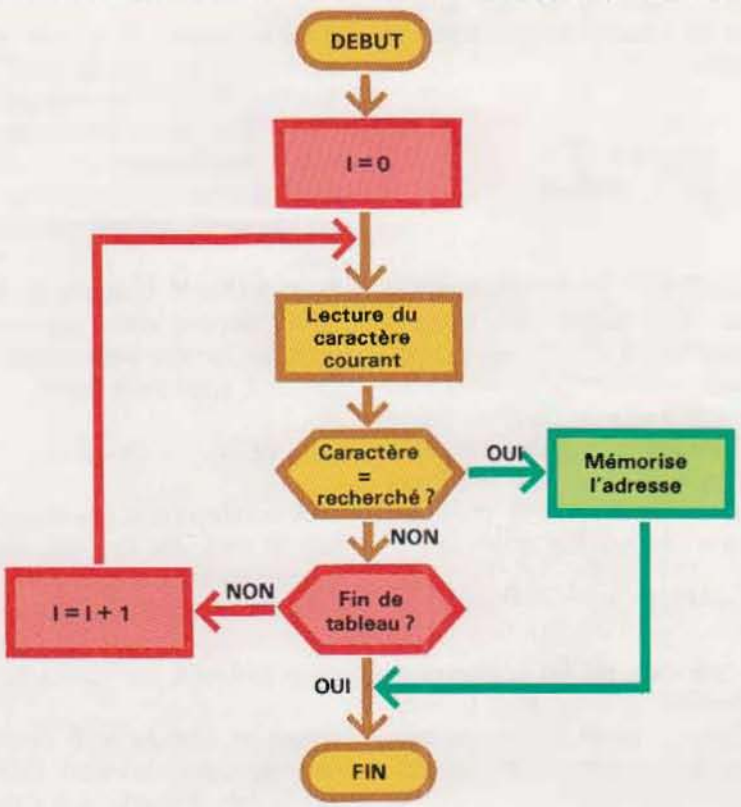
Comme on vient de le voir, le langage Assembleur est évidemment plus complexe que les langages symboliques comme le Basic. Il demeure encore aujourd'hui indispensable dans la résolution d'applications. Son rapport étroit avec le fonctionnement réel du microprocesseur exige, de la part du programmeur, 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.

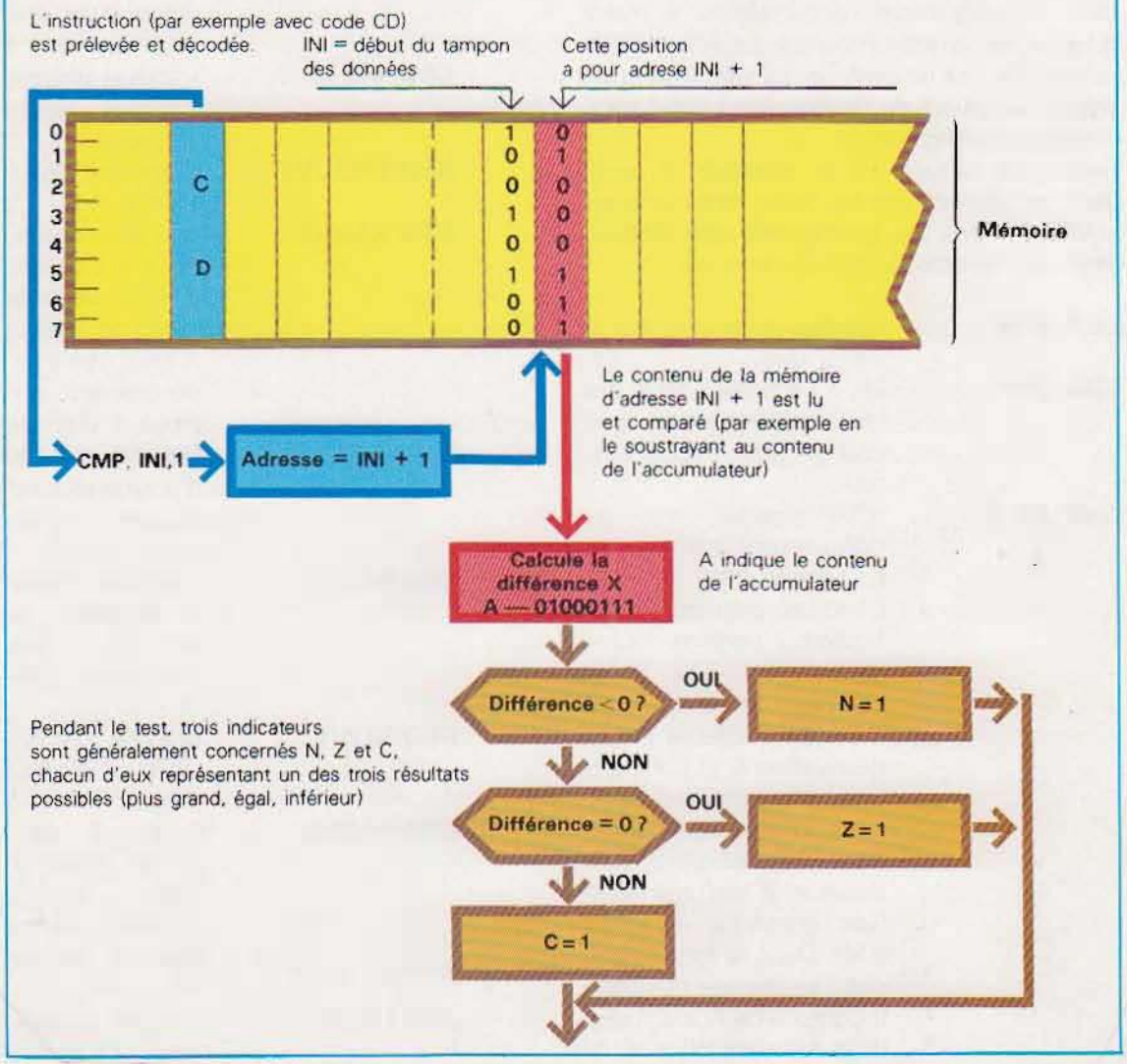
BEQ NONT Branche à NONT (NON TROUVE) si égal.

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)

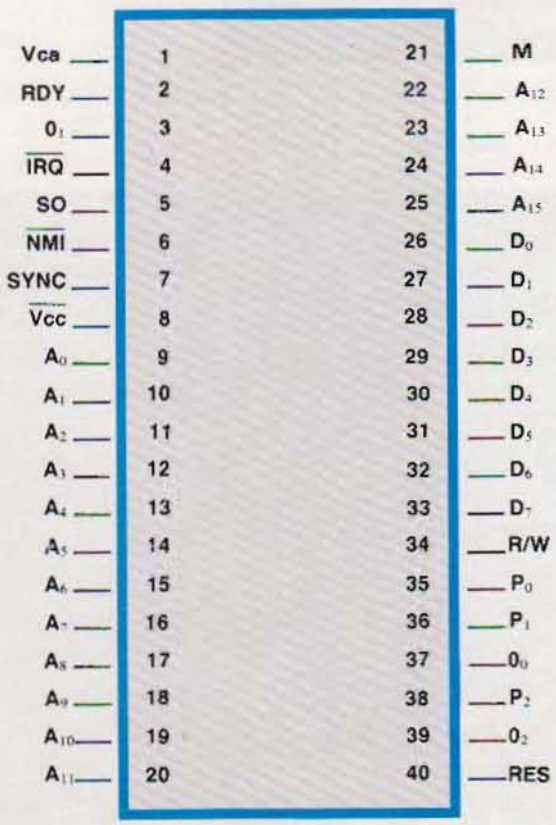
I Bit d'interruption matériel (positionné à 1 quand les interruptions sont impossibles)

Z Bit zéro (1 si la dernière opération a eu pour résultat zéro)

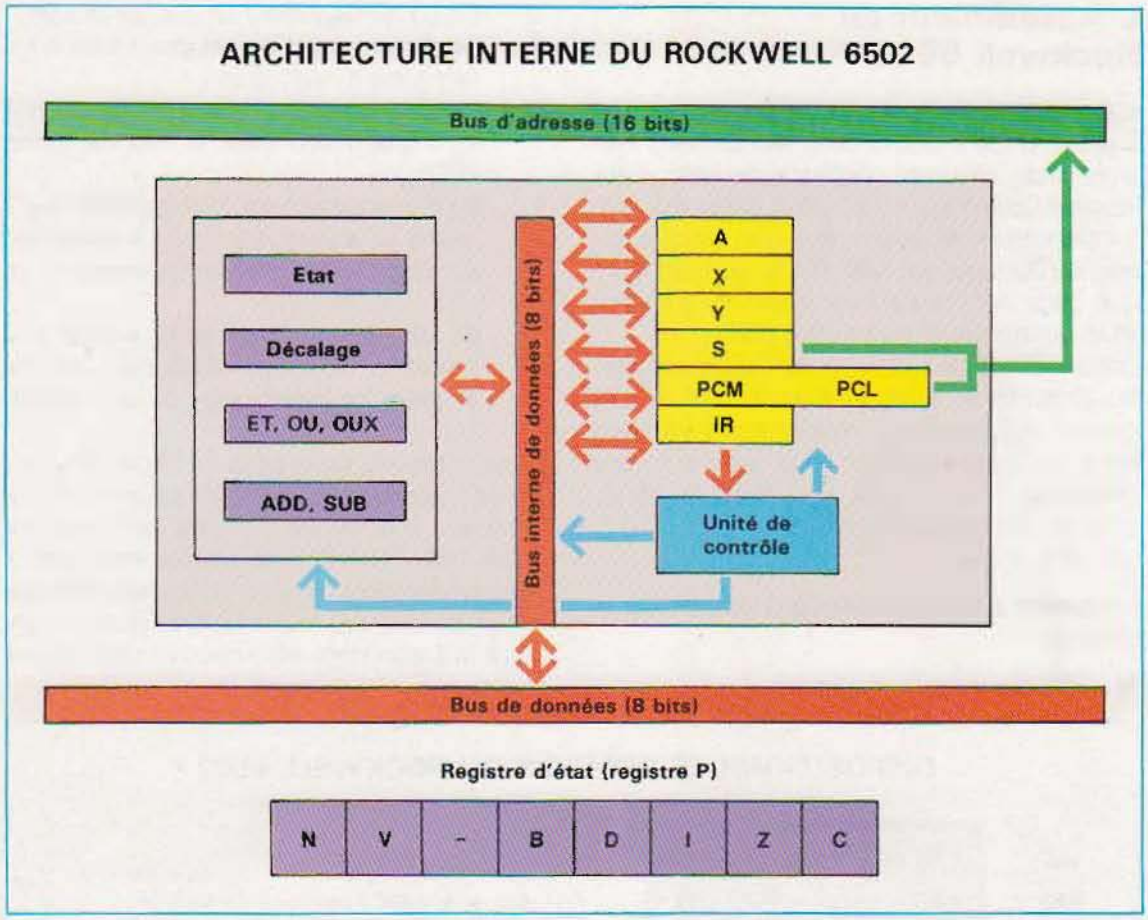
C Bit de retenue (1 s'il y en a une). Cet indicateur est aussi utilisé pour les instructions de permutation ou de rotation.

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).

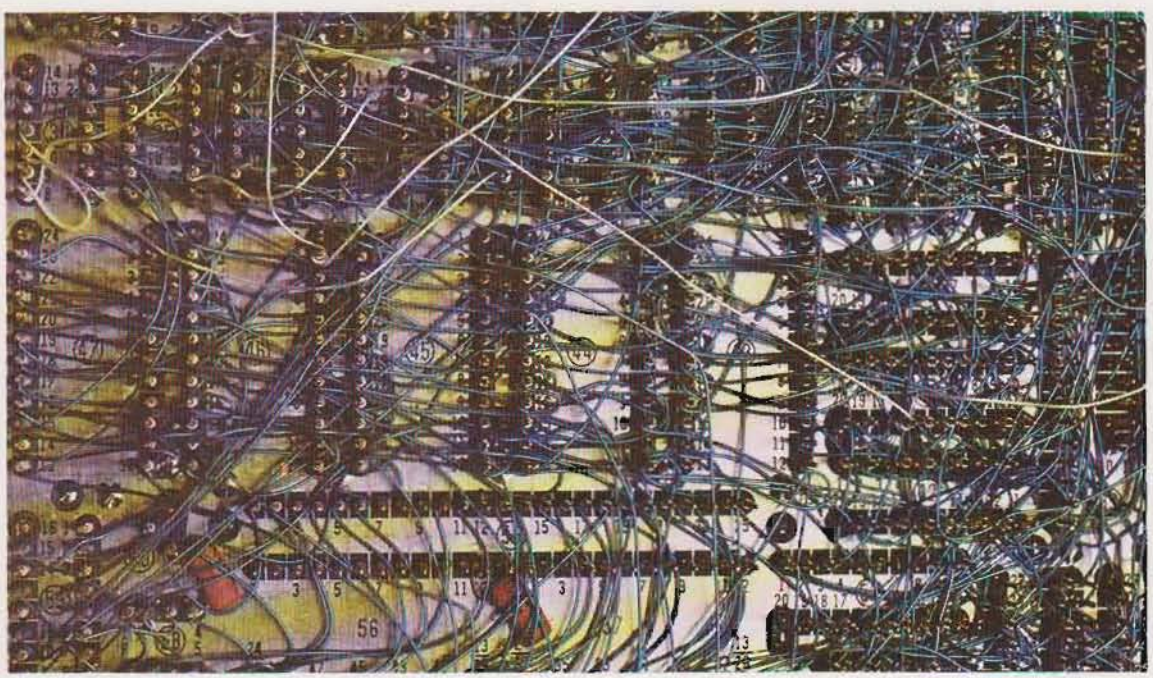
DISPOSITION DES BROCHES DU ROCKWELL 6502



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.

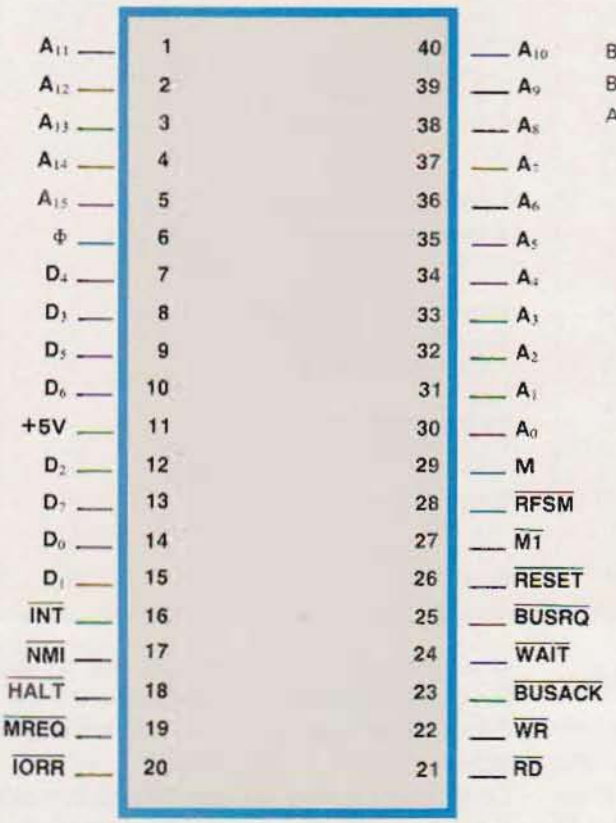
La structure interne du Z80 est très complexe. Il est composé de deux accumulateurs (A, A') et de deux couples de 6 registres (BC, DE, HL, B'C', D'E', H'L') avec, en plus, deux registres index (IX, IY), un pointeur de pile (SP), un compteur ordinal (PC), le registre d'instruction (RI) et deux registres 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

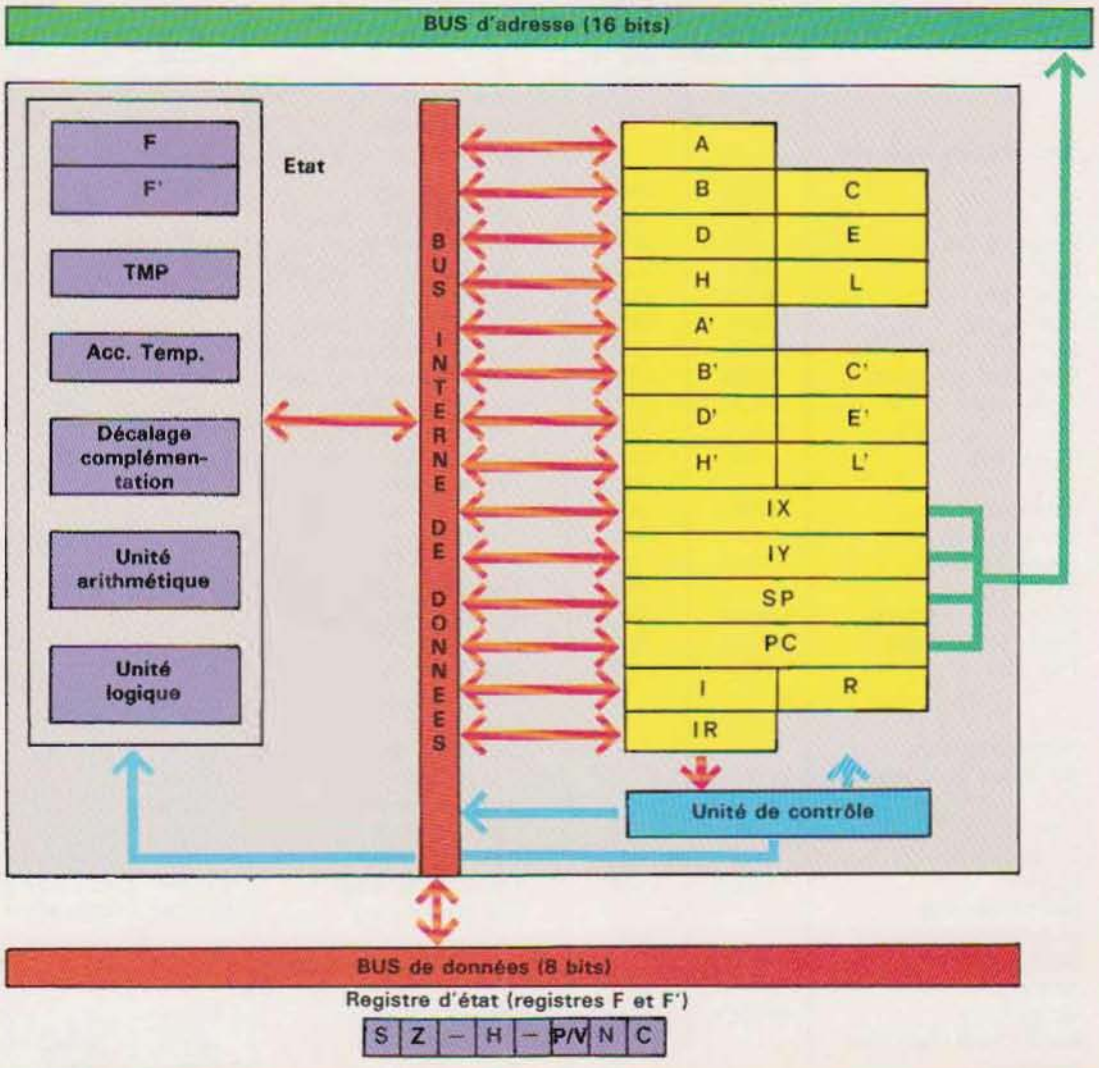
Le tableau des pages 949-950 répertorie les codes mnémoniques des instructions du Zilog Z80 comparés aux codes standard IEEE. Comme on peut le voir, l'ensemble des instructions disponibles est très large, ce qui donne une grand souplesse d'exécution au microprocesseur.

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).

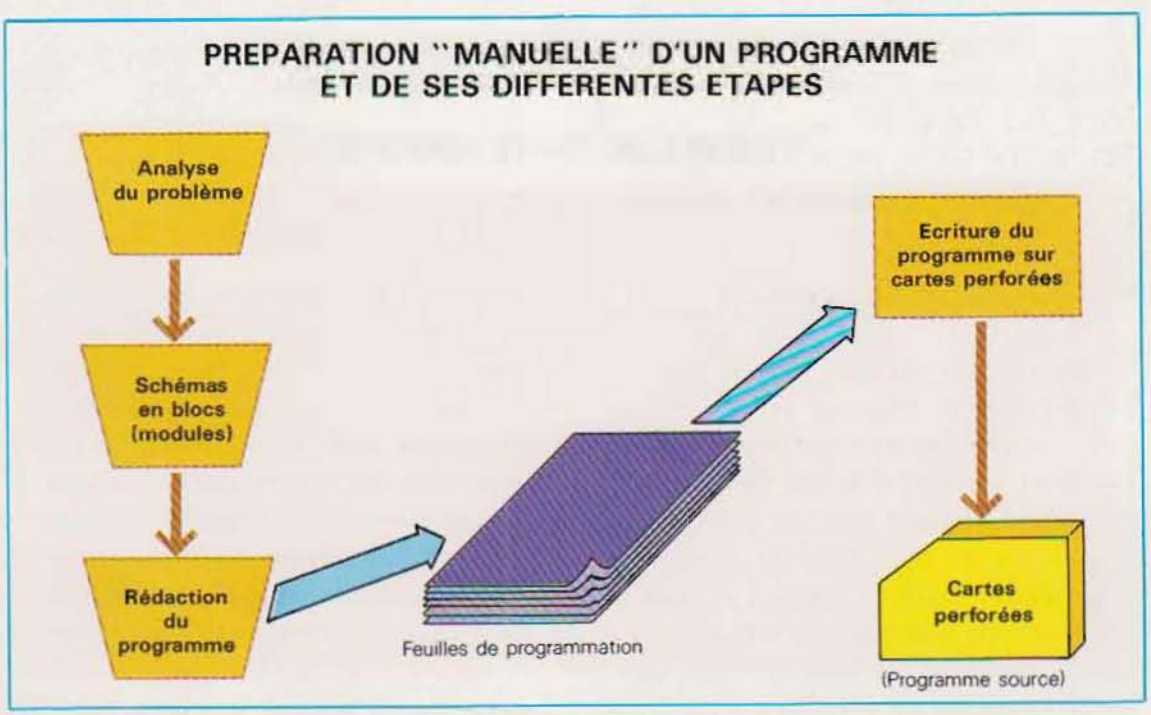
2/ Préparation des matériaux.
Pour pouvoir fournir les résultats attendus, la colle doit agir sur les matériaux à rassembler après qu'ils ont été soigneusement préparés.

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.

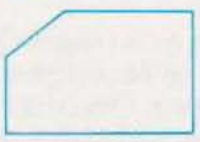
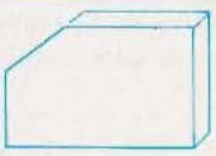
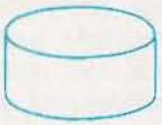
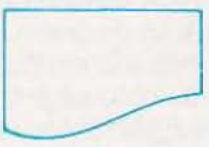
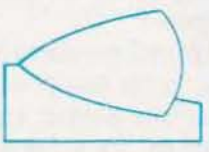
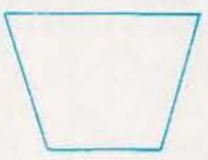
De même que pour la colle utilisée pour assembler deux axes, l'ordinateur n'intervient qu'en fin de parcours. Ce n'est qu'une des étapes que le programmeur doit accomplir avant d'aboutir au résultat. Ci-dessous sont représentées toutes les étapes nécessaires pour obtenir une première ébauche du programme 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

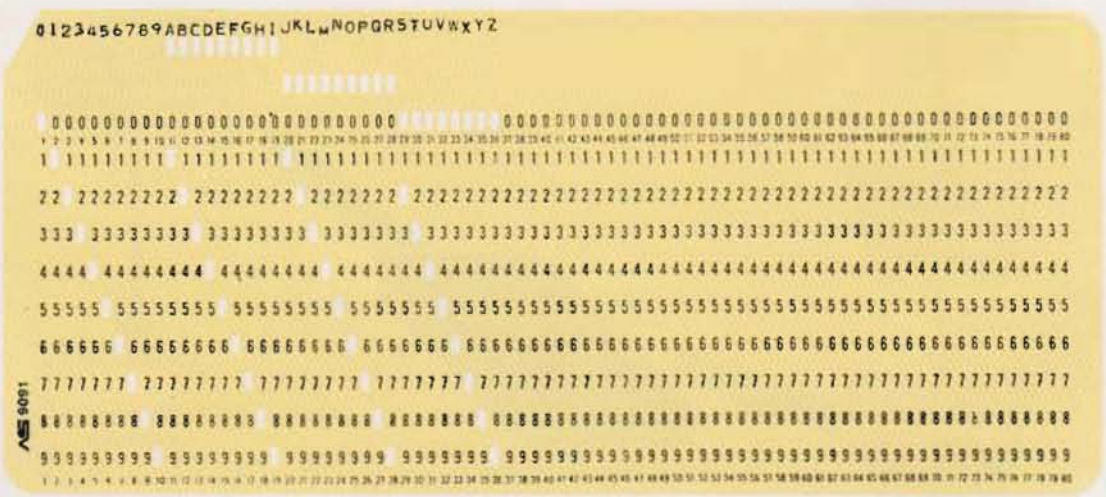
Une fois le découpage en blocs terminé, le programmeur commence la rédaction du programme, à savoir la traduction des différentes étapes de l'organigramme en série d'instructions (qui respectent la forme prévue par le langage choisi) et écrites sur des formulaires spéciaux appelés **feuilles de programmation.**
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.

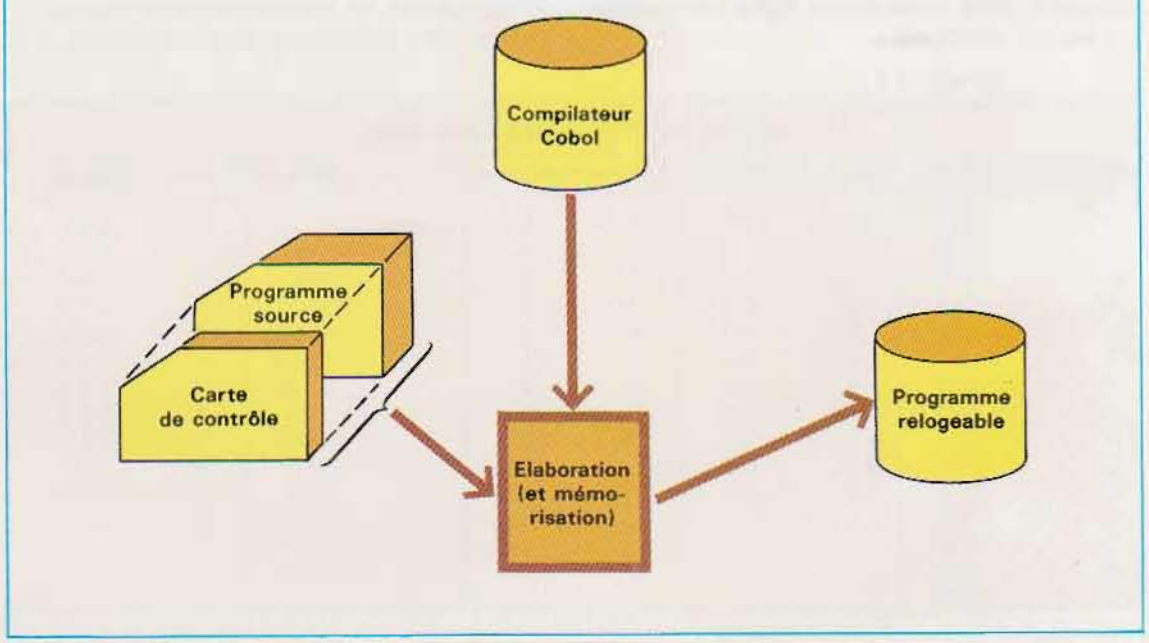
- Communiquer au système d'exploitation le nom du compilateur qui doit être chargé en mémoire (Cobol, Fortran, Basic...).
- Lancer l'opération de chargement du compilateur en faisant intervenir un programme appelé **chargeur** (loader).
- Fournir au compilateur le nom du programme source et le nom du 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.

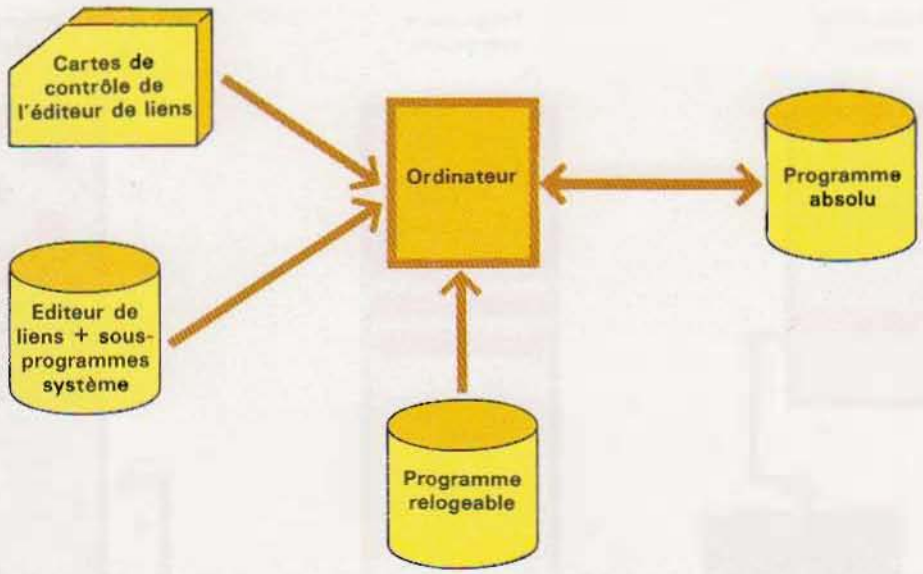
Prenons comme exemple les fonctions de lecture-écriture sur fichier : des routines d'entrée-sortie sont destinées à la réalisation de ces fonctions. Ainsi, le compilateur crée-t-il les points d'entrée nécessaires dans tous les sous-programmes appelés par les instructions du programme principal en enregistrant 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

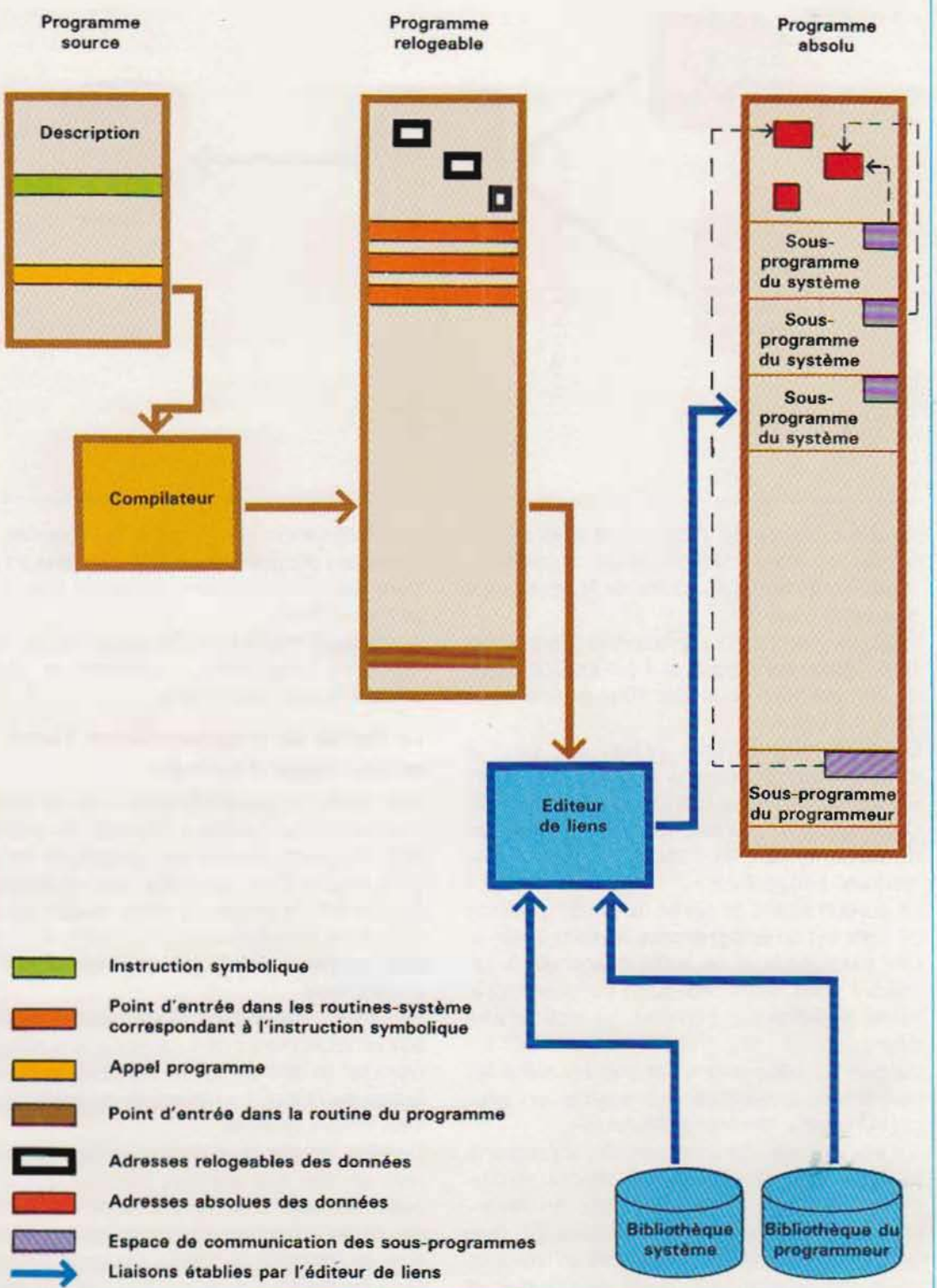


tions écrites par le programmeur et faisant partie des autres bibliothèques, à condition toutefois qu'on lui en fournisse le nom grâce à la carte contrôle.
En fin de phase de compilation, le programme objet réside sur disque et il est indispensable de lui associer tous les sous-programmes dont il a besoin.
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.

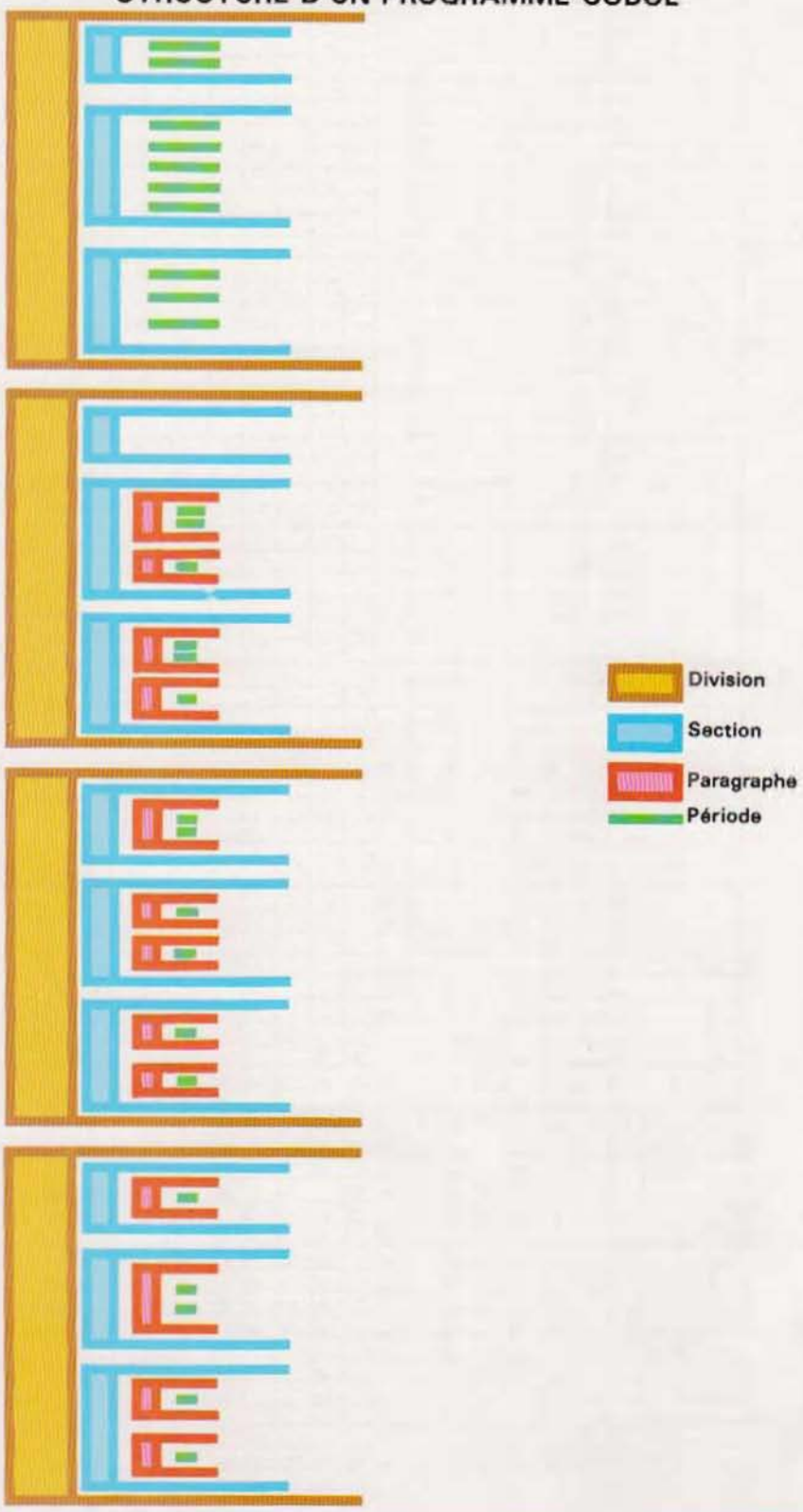
PROGRAMMEUR _____ DONNEES _____ COBOL

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-

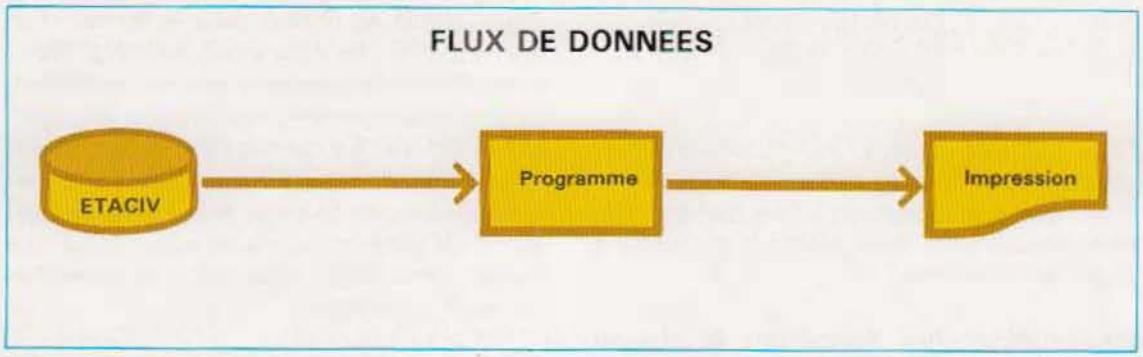
FORMAT DE LA CLAUSE SELECT

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).

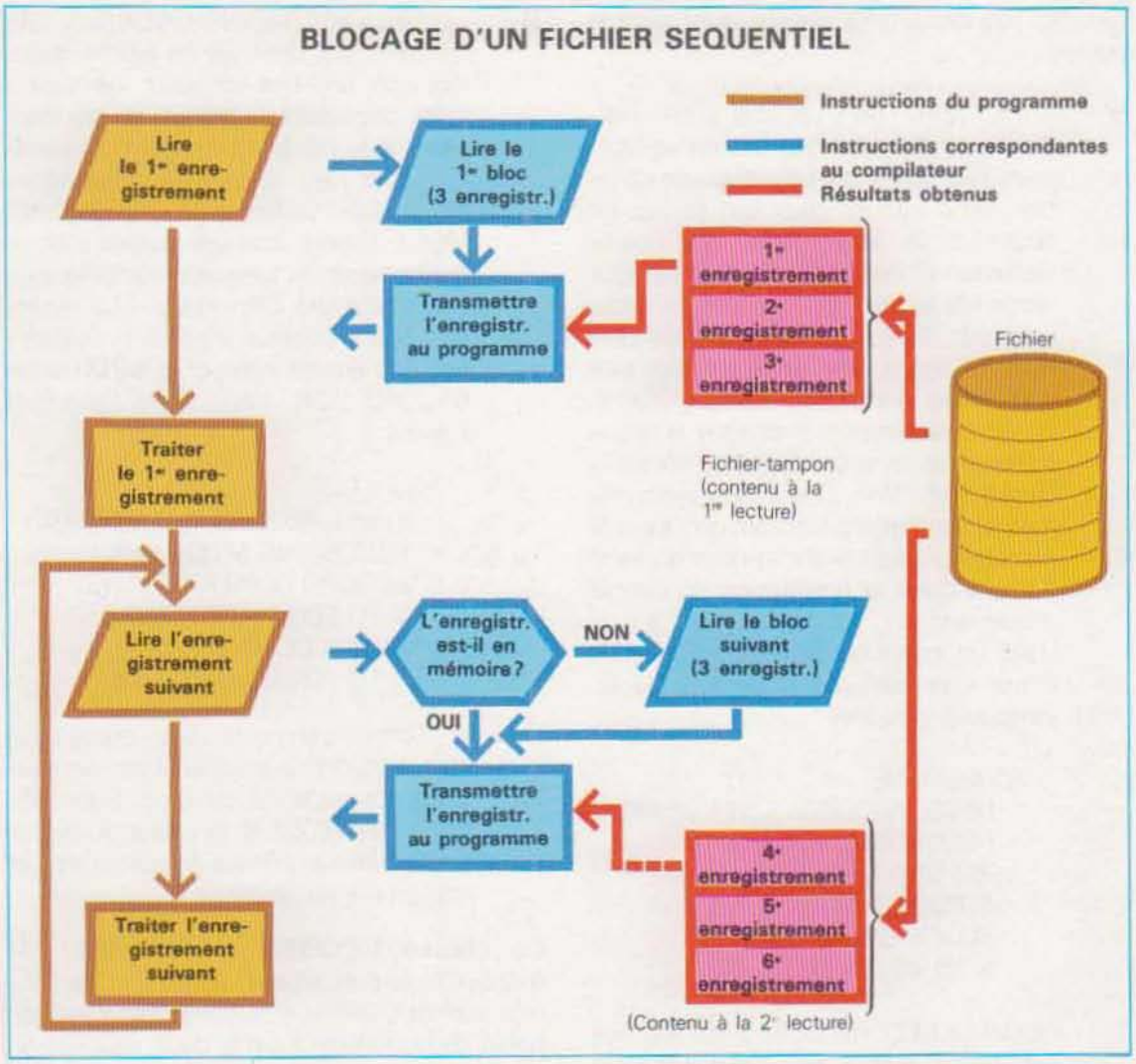
FORMAT GENERAL DE LA CLAUSE LABEL RECORD

```
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

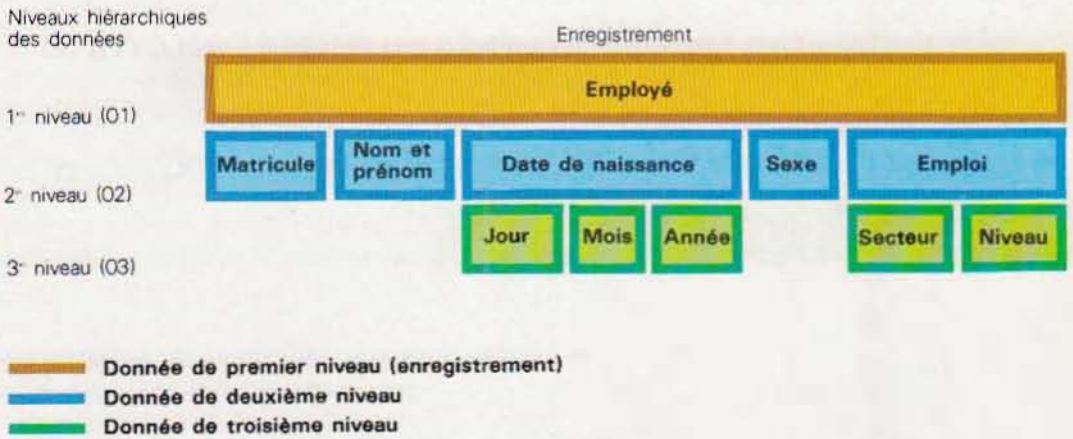
FORMAT DE LA CLAUSE RECORD CONTAINS

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.
```

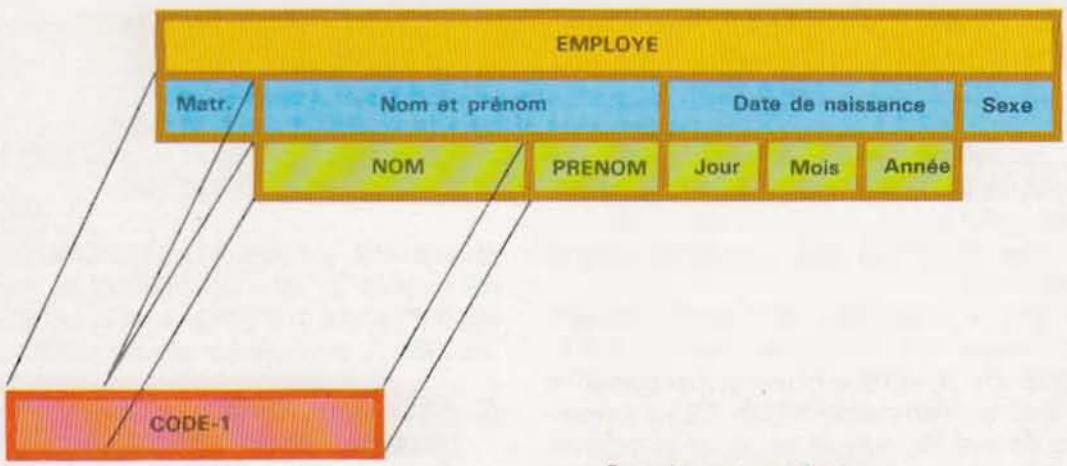
Supposons qu'il faille maintenant grouper en un seul champ (dénommé CODE-1) à la fois le numéro de matricule (MATRICULE) et le nom de l'employé (NOM) sans toutefois faire de transfert. L'opération est réalisée au moyen de la clause RENAMES, en écrivant :

```
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

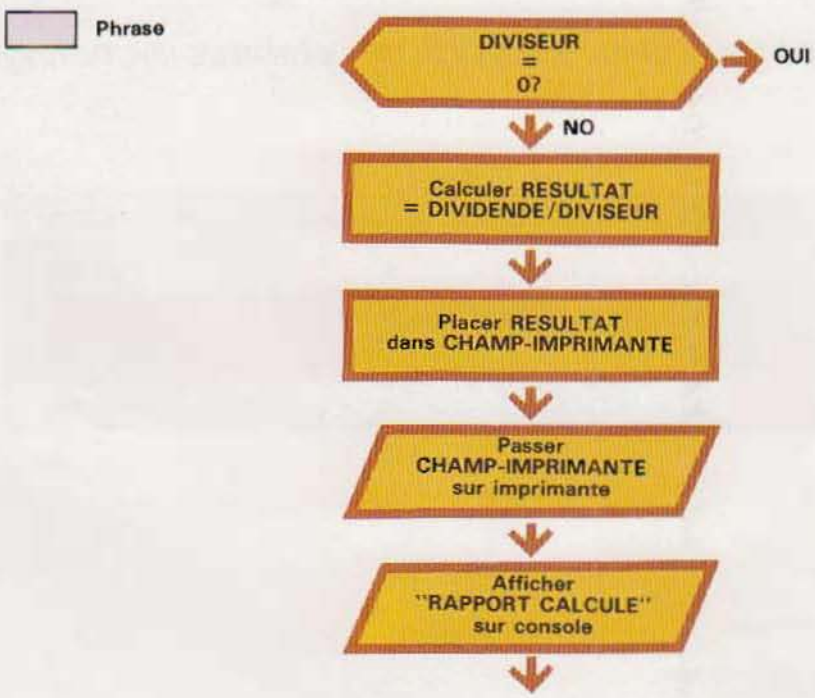
* Pour définir une donnée alphanumérique, il faut au moins un autre symbole de définition de classe (A, X, 9)

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)

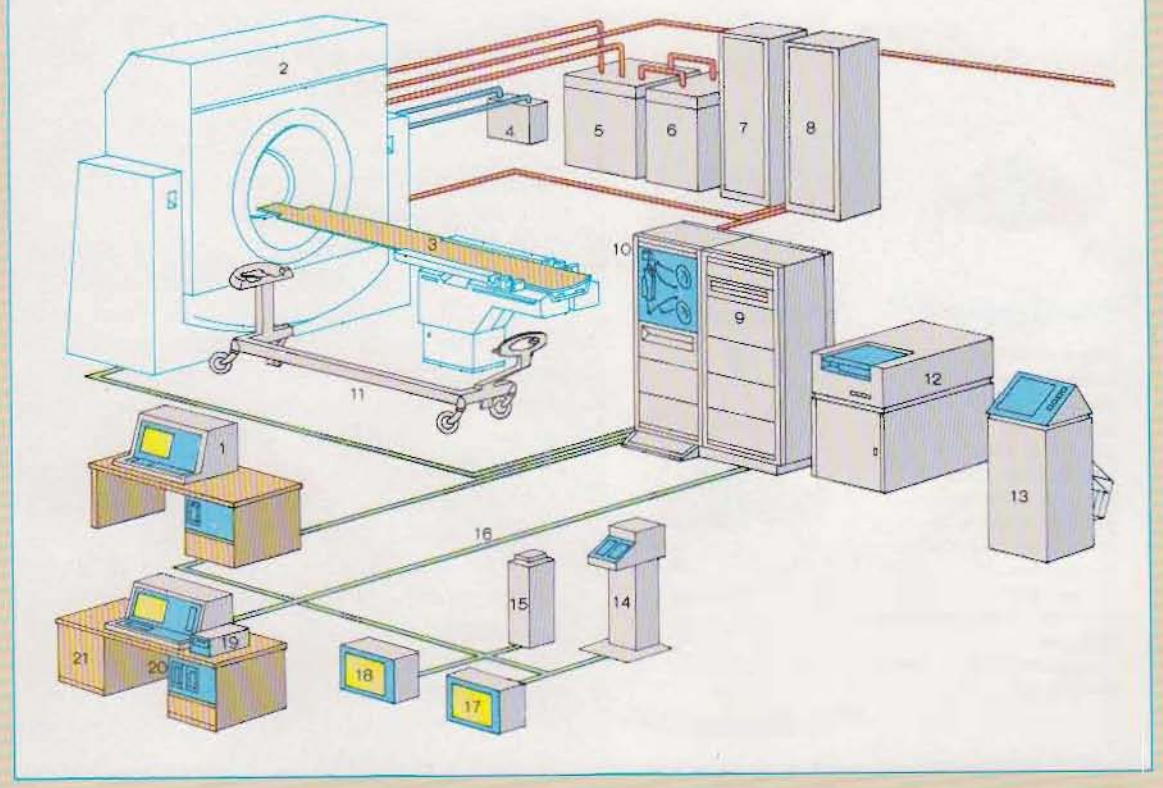
C'est vers 1970 que l'informatique a commencé à s'introduire dans le domaine biomédical et la tomographie est peut-être la technique qui a le plus bénéficié, jusqu'ici, de ses progrès.

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.*

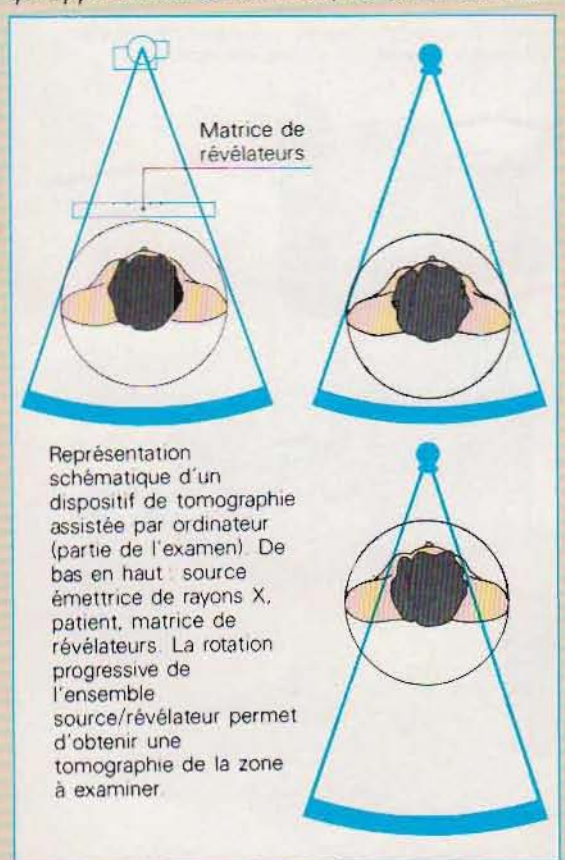
La TAO repose sur la reconstruction d'une ***image globale*** *issue des différentes images radiographiques réalisées sous plusieurs incidences. Une source émettrice, permettant une haute résolution, est positionnée le long d'une circonférence et orientée vers le centre. Au point de la circonférence 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.

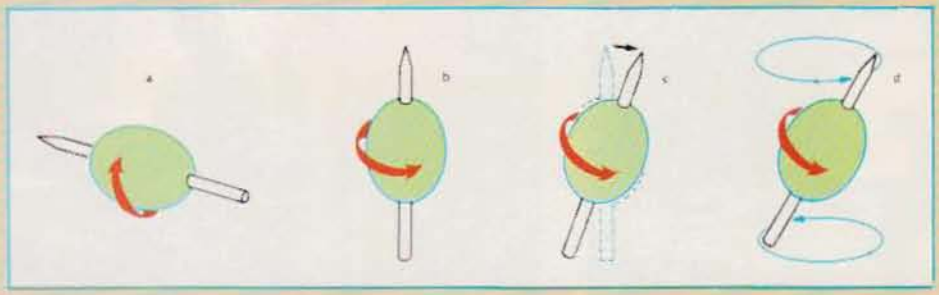
Tomographie axiale assistée par ordinateur (TAO). Photo du haut à gauche : vue d'ensemble du Tomoscan. Ci-dessus : positionnement automatique du sujet à examiner. Ci-contre : examen à distance de la coupe sériée sur une console.

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)
```

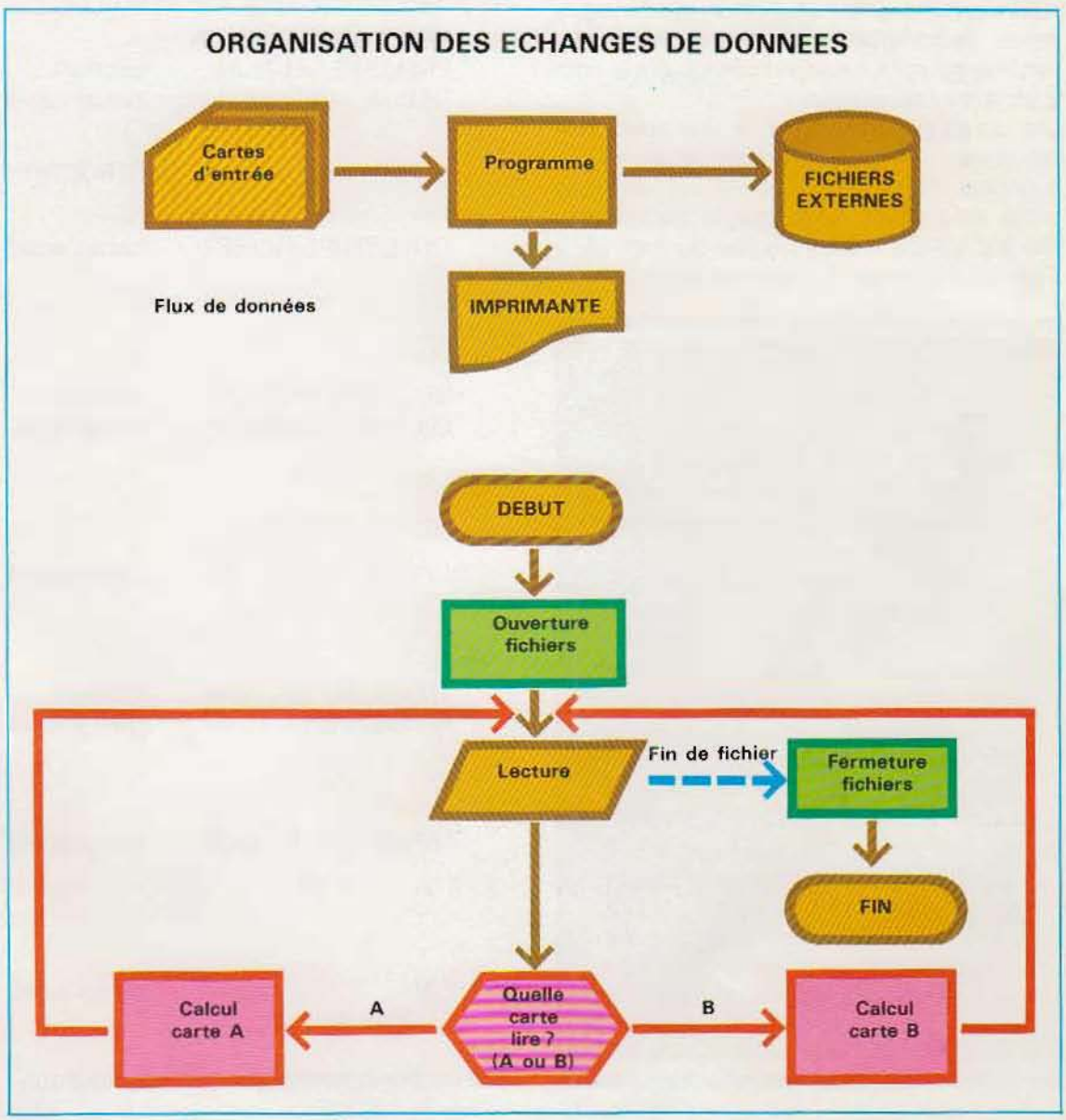
au lieu de :

```
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