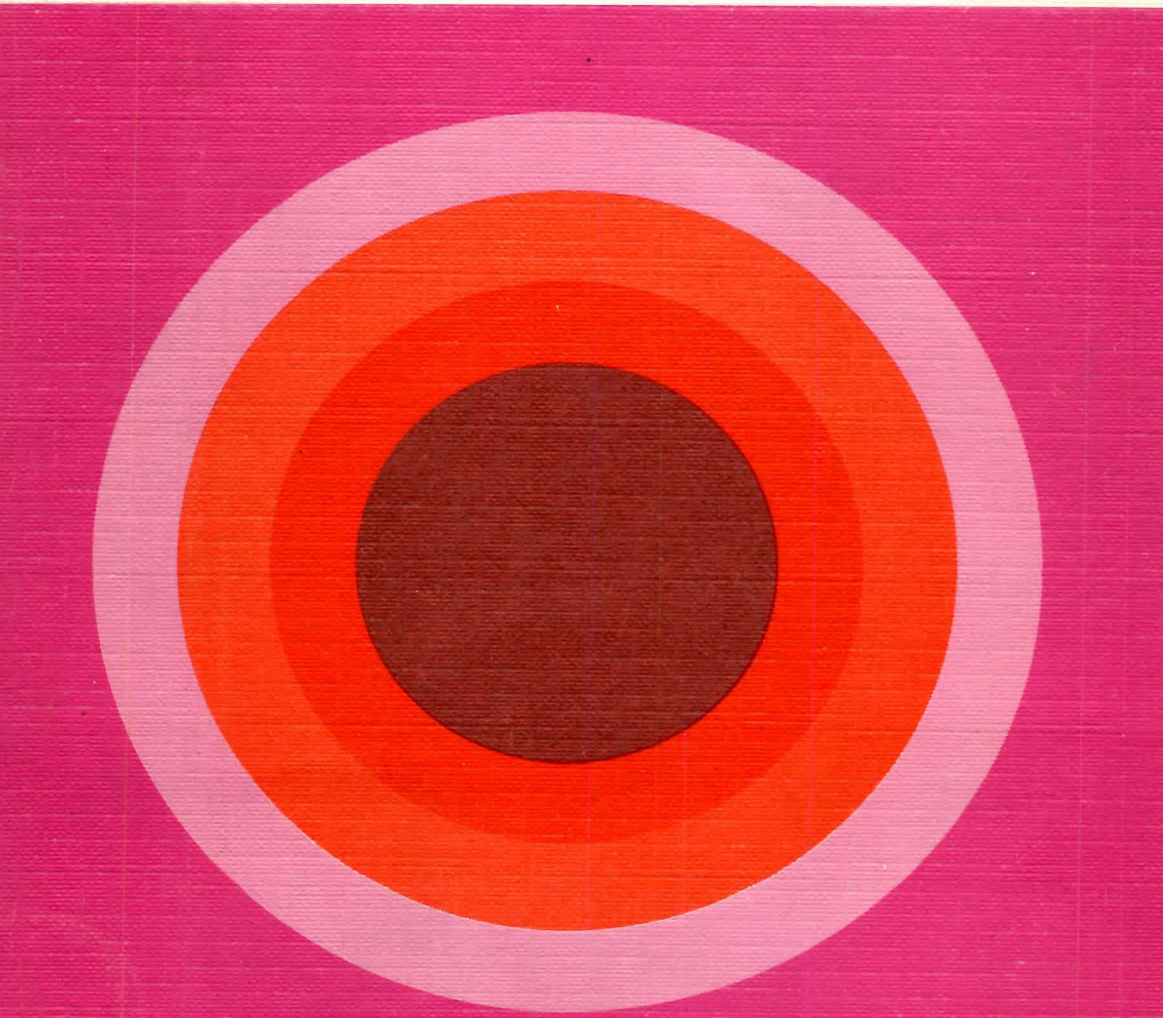
cours élémentaire d'informatique

JEAN-PIERRE BOYER

RÉSERVÉ AUX FUTURS UTILISATEURS DE L'ORDINATEUR



ÉDITIONS RADIO

6 grands de l'électronique

Spécimen gratuit sur demande.

Toute l'Electronique

Revue mensuelle de technique expliquée et appliquée fondée en 1934 (11 numéros par an). Traitant de tous les aspects de l'électronique, elle est lue par tous les techniciens spécialisés de l'agent technique à l'ingénieur de recherches.

électronique pour vous HI-FI INTERNATIONAL

Revue mensuelle fondée en 1972 et consacrée à l'électronique « Grand Public » (11 numéros par an). En plus de la Hi-Fi et de ses bancs d'essais authentiques et rigoureux, des articles de synthèse sur les grands thèmes modernes d'actualité technique, des réalisations pratiques dans les domaines de la photo, la sonorisation, l'auto, la maison, le bateau, la télécommande, les modèles réduits, etc., et bien sûr des critiques de disques, des classiques aux plus pop.

Electronique & microélectronique industrielles

Revue fondée en 1955 et s'adressant aux promoteurs et utilisateurs des méthodes et appareils électroniques appliqués à tous les domaines de l'industrie (16 numéros par an).

électronique actualités

Hebdomadaire fondé en 1965, destiné aux cadres supérieurs de l'industrie et contenant toutes les nouvelles techniques, commerciales, financières et syndicales.

automatique & informatique industrielles

Revue mensuelle fondée en 1972 (11 numéros par an). Seule revue en France spécialisée dans les techniques et applications industrielles de l'automatisation.

REVUE DU SON
LES ARTS SONORES ET LES TECHNIQUES AUDIOVISUELLES

Première revue d'électroacoustique et de haute fidélité en langue française, fondée en 1953. Contenu des rubriques : acoustique, banc d'essai, restitution sonore, circuits, sonorisation, enregistrement, panorama audio, Hi-Fi télex, documents techniques, mesures, techniques audiovisuelles, musique électronique, arts sonores.

ÉDITIONS RADIO

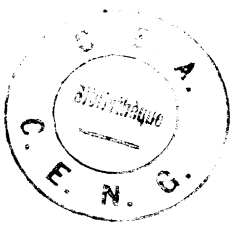
9, RUE JACOB, 75006 PARIS - TÉL. : 033 . 13 . 65

Jean-Pierre BOYER

Ingénieur E. S. E.
Ingénieur à la société Philips
(division ordinateur)

COURS ÉLÉMENTAIRE D'INFORMATIQUE

Réservé aux futurs utilisateurs
de l'ordinateur



ÉDITIONS RADIO

9, rue Jacob, 75006-Paris

Imprimerie Berger-Levrault
Dépôt légal 4e trimestre 1972
no éditeur : 562 - no imprimeur : 778635

PRÉFACE

Il y a quelques mois M. J.P. BOYER m'a communiqué pour avis le manuscrit de son livre COURS ÉLÉMENTAIRE d'INFORMATIQUE. Je l'ai d'abord *parcouru rapidement, puis j'ai été frappé par la clarté de ses explications et des exemples indiqués. La disposition de parties délicates réalisée d'une manière extrêmement intelligente m'a beaucoup frappée.*

C'est alors que j'ai lu l'ouvrage intégralement et immédiatement j'ai pensé combien il serait utile à des milliers de jeunes désireux d'apprendre ce qu'est réellement l'informatique. J'ai aussi songé à de nombreux techniciens spécialisés et d'agents techniques avides d'évoluer dans leurs fonctions.

Sur ma recommandation M. J.-P. BOYER s'est adressé à la Société des ÉDITIONS RADIO et son ouvrage — à juste titre d'ailleurs — a été accepté.

Des essais de textes et des avant-projets de livre ou d'articles me sont souvent proposés mais hélas ils ne correspondent pas souvent à des besoins nettement définis et comportent trop de reprises d'éléments déjà connus.

L'éditeur de M. BOYER a vu juste et je suis persuadé du succès de cette édition qui a le mérite d'être essentiellement pratique et d'avoir été réalisée par un jeune technicien pour les autres jeunes techniciens.

Dans de tels cas, il ne suffit pas de frapper à la bonne porte, car les propositions sont nombreuses et le sommaire d'un ouvrage ne correspond souvent pas à un but précis.

M. J.-P. BOYER a parfaitement réussi une œuvre délicate, les Éditions RADIO ont su apprécier à sa juste valeur un manuscrit de qualité; le succès certain récompensera les efforts conjugués.

Un tel livre vient à son heure; de récentes manifestations, salons ou expositions ont montré combien les éléments fondamentaux de l'informatique manquaient à tous ceux qui, de près ou de loin, œuvrent dans cette noble spécialité, ou font appel à ses services, pour la gestion des entreprises auxquelles ils sont liés.

C'est un grand plaisir pour moi, que de présenter l'ouvrage de mon ami, J.-P. BOYER car je suis persuadé que les services qu'il rendra, n'auront d'égal que l'extraordinaire qualité de l'enseignement qu'il apporte.

Maurice LORACH

Docteur-Ingénieur de la
Faculté des Sciences de Paris

Avertissement de l'auteur

Certaines applications de l'informatique sont maintenant bien connues du grand public. Il suffit de citer, par exemple, les sondages d'opinion, le tiercé, la réservation des places d'avion, l'utilisation des étiquettes sur les marchandises des grands magasins, l'adressage des journaux ou tout simplement l'édition du bulletin de paye. L'informatique, technique du traitement automatique de l'information, a en effet, influencé tous les milieux. Pourtant, sa compréhension, son assimilation, restent difficile à la fois à cause de son origine et de son vocabulaire.

Avant les années 60, il existait d'une part en « mécanographie » le « matériel classique », c'est-à-dire, des « machines électro-comptables » capables de faire subir à des fichiers sur cartes perforées des traitements élémentaires. Il y avait d'autre part, des « calculateurs », appareils scientifiques capables à partir de quelques données d'effectuer un grand nombre d'opérations arithmétiques. L'avènement des supports magnétiques et aussi la miniaturisation des éléments électroniques a permis le rassemblement de toutes ces machines en une seule : l'ordinateur.

L'informatique ensuite est née mais elle souffre toujours de cette double hérédité : Venant de la « mécanographie », le service informatique est le plus souvent considéré comme un sous-traitant des autres services. Il n'est pas intégré dans l'entreprise. Par suite les applications ne sont pas forcément utiles, rentables. L'informatique ne pénètre pas dans l'entreprise et une bonne gestion reste impossible. Pour changer certaines habitudes il faudra une génération d'hommes.

Héritière des « calculateurs », l'informatique est placée à tort, à côté des mathématiques. Ainsi, il n'est pas rare de commencer son initiation par « l'algèbre de Boole », « la théorie des ensembles », et « la théorie des graphes ». En fait, ces théories intéressantes ne sont utilisées que dans des domaines étroits et très particuliers. L'informatique demande surtout un esprit logique, à la fois analytique et synthétique, proche du grammairien ou du spécialiste du Droit.

La barrière du vocabulaire est également difficile à franchir. Les mots utilisés en informatique changent de signification lorsqu'on change de gamme d'ordinateurs. Qui dira par exemple la différence entre un sous-programme, un bloc de programme, un module de programme, une routine, une procédure? Encore, restons-nous là dans la langue de Racine sans pénétrer dans l'anglais. Cet ouvrage essaye donc d'utiliser les mots les plus courants et les plus clairs.

Le cheminement de ce petit livre est simple. Il part de l'ordinateur pour arriver aux applications. Le chapitre 1 présente le matériel et donne le principe général de l'ordinateur. Il permet au chapitre 2 de montrer plusieurs solutions pour mécaniser un travail simple. Les avantages supplémentaires apportés y sont évidents. Ensuite, les chapitres 3, 4 et 5 expliquent la programmation et l'analyse, le mode d'emploi de l'ordinateur. Les chapitres 6, 7 et 8 présentent le software, c'est-à-dire, l'ensemble de tous les programmes standards placés entre l'homme et la machine, ceux proches de l'ordinateur (software d'exploitation) comme ceux proches des applications (software d'applications). Successivement, le lecteur apprend ce que sont : le chaînage des travaux, la multiprogrammation avec sa gestion des tâches, sa gestion des ressources et sa gestion des travaux, le traitement en temps réel, le temps partagé, les banques de données, les tris, les programmes utilitaires. Enfin, le chapitre 10 place l'ordinateur et le service informatique dans l'entreprise. Il donne l'éventail de ses possibilités et de ses limites.

En conclusion, les applications sont plus importantes que l'ordinateur lui-même. L'informaticien lorsqu'il étudie un nouveau système pense à la saisie des informations, à l'organisation des fichiers, aux résultats à transmettre. L'ordinateur ne vient qu'ensuite. Ce dernier ne pourra que répéter un très grand nombre de fois, mais très rapidement, que ce que nous lui avons montré. Alors, l'homme débarrassé des tâches routinières et inintéressantes pourra réfléchir et donner libre cours à son imagination.

J.P. BOYER

CHAPITRE PREMIER

LE HARDWARE

1. — Définition

Dans l'expression « Traitement de l'Information » (qu'on appelle aussi « Informatique »), il y a deux mots :

Le mot « information » implique une notion de mouvement par rapport à des fonctions statiques, mouvement ou transmission de quelque chose contenant une nouvelle, un renseignement.

Le mot « traitement » implique une réception, une collecte de ces informations, un filtrage pour les envoyer dans différentes directions, une modification pour les rendre plus compréhensibles.

Le traitement de l'information groupe plusieurs phases. Il y a d'abord la phase *organisation :* c'est l'étude des structures et des circuits dans le domaine de la gestion des entreprises, l'examen d'un processus dans le domaine industriel, la recherche et les études dans le domaine scientifique. Cette phase groupe donc des disciplines absolument diverses dans des domaines différents. Elle est amenée pour certains secteurs — certaines parties de circuit, certains problèmes — à utiliser les moyens techniques de l'automatisme.

La phase *analyse* reprend chacune de ces parties de circuit, chacun de ces problèmes et les « analyse » en détail afin de définir comment ils vont être traités automatiquement. Elle détermine les moyens mécaniques pour passer d'un « nœud d'entrée » à un « nœud de sortie », d'un ensemble de données à un ensemble de résultats.

La phase *programmation* étudie la succession des différents ordres à donner à la « machine ».

La phase *exécution* constitue le traitement automatique de l'information proprement dit. Elle calcule, donne les résultats à partir des données.

Ces quatre phases : organisation, analyse, programmation, exécution résument la mise en place d'un problème de traitement de l'information. Il faut ajouter la construction de la « machine », condition préalable évidemment nécessaire à son fonctionnement.

Elle a d'abord été un ensemble de plusieurs machines électro-mécaniques ayant chacune une fonction élémentaire. On appelle cet ensemble : « matériel mécanographique » ou aussi « matériel classique ». On a réuni ensuite toutes ces fonctions élémentaires dans une seule machine : « l'ordinateur ».

Dans la construction d'un ordinateur on distingue deux parties : une partie physique, câblée, c'est le *hardware*. Une partie plus difficilement palpable mais nécessaire, qui relie l'utilisateur au hardware, c'est le *software*.

Avant de donner un exemple simple de traitement de l'information, d'entrer dans les domaines de la programmation, de l'analyse, du software, de l'organisation il est nécessaire de posséder une vue synthétique sur l'évolution de la technologie des machines (sur le hardware). Moins de dix années de progrès technique, nous a conduit de la carte perforée aux unités de traitement en temps réel, du matériel « classique » à l'ordinateur possédant un système d'exploitation à priorité.

2. — Les supports

Afin de rendre automatique (mécanisable) le traitement de l'information, il est nécessaire de placer l'information sur un support pouvant être lu par une machine. Passons en revue les différents supports utilisés :

2.1. — LA CARTE PERFORÉE

Le premier support auquel on a pensé s'appelle la carte perforée. Il s'agit d'un morceau de carton de forme rectangulaire aux dimensions standardisées. On divise ce rectangle en colonnes (généralement 80 colonnes).

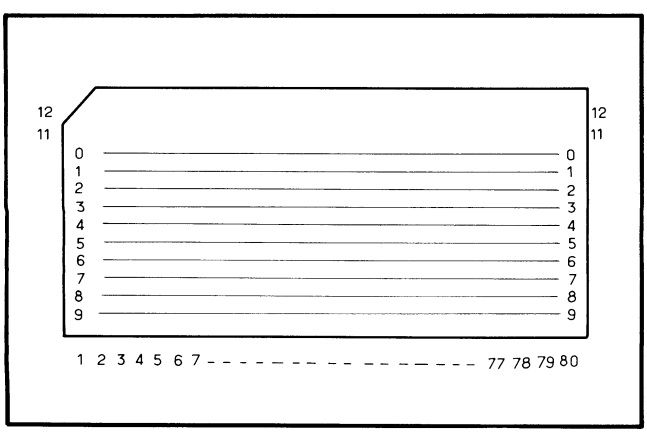


Fig. 1-1. — Carte perforée.

A chaque colonne correspond un caractère (numérique, alphabétique ou spécial) de l'Information (en principe donc 80 caractères au maximum sur une carte) (fig. 1.1).

On divise également le rectangle en lignes afin d'établir une codification : par exemple il y a les lignes 0 à 9 pour représenter les chiffres et deux lignes supplémentaires 11 et 12 pour pouvoir représenter les autres caractères. On place un caractère dans une colonne en perforant des trous (rectangulaires) dans cette colonne à la hauteur des lignes suivant une codification qui dépend du constructeur. Généralement, un angle de la carte est biseauté afin de pouvoir vérifier facilement que, dans un paquet, toutes les cartes sont dans le même sens. Souvent, pour faciliter la lecture de la carte, on imprime sur celle-ci en trait plus fort ce qu'on appelle un *dessin de carte*. Par exemple, si on décide de placer les numéros d'article dans les colonnes 4 à 10 et les prix unitaires correspondants dans les colonnes 11 à 16 pour constituer un « fichier » stock, pour les cartes de ce fichier on place un trait fort entre les colonnes 3 et 4, un autre entre les colonnes 10 et 11 et un autre après la colonne 16.

2.2. — LE RUBAN PERFORÉ

Il s'agit d'un ruban de papier divisé en canaux longitudinaux; (le nombre de canaux varie avec les constructeurs). On représente un caractère par un certain nombre de perforations placées sur une ligne perpendiculaire au sens du ruban. Généralement, on s'arrange pour que le nombre de perforations représentant un caractère soit toujours impair (ou pair) en réservant un canal de la bande pour respecter cette imparité (fig. 1.2).

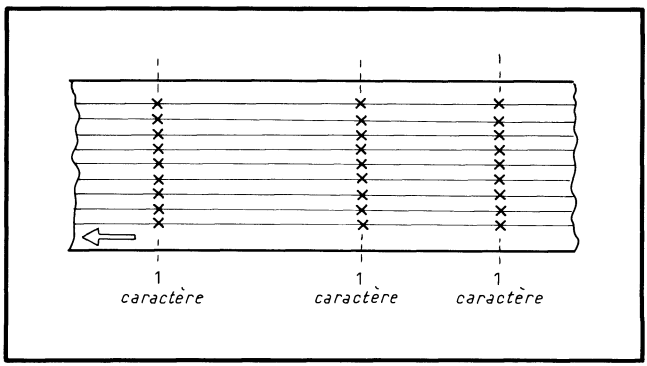


Fig. 1-2. — Ruban perforé.

2.3. — LA BANDE MAGNÉTIQUE

Le principe est le même que sur le support précédent sauf que le papier est remplacé par une matière plastique et les trous par des zones magnétisées. On peut réécrire sur une bande magnétique en effaçant les informations qui s'y trouvent. Un canal spécial d'imparité (ou de parité) permet de vérifier la validité des caractères.

Deux petites plaquettes métalliques (appelées sticker) placées au début et à la fin de la bande la délimitent physiquement.

A titre indicatif donnons quelques ordres de grandeur : la longueur d'une bande magnétique varie de 700 à 1200 mètres; les enregistrements contiennent habituellement 300 à 3000 caractères; ils sont séparés par des entre-enregistrements de 1 à 2 cm; la densité est de 80 à 640 caractères par centimètre.

2.4. — LES MÉMOIRES A ACCÈS SÉLECTIF

Nous verrons, lorsque nous parlerons des organisations de fichier, que les supports précédents sont du type séquentiel. Pour lire une information sur une bande, il faut positionner la bande sur son sticker de début et faire défiler la bande (en séquence) jusqu'à ce que l'on trouve cette information.

Les mémoires à accès sélectif permettent d'avoir accès directement à un enregistrement.

On rencontre dans cette catégorie de mémoire les disques, les tambours et les feuillets magnétiques.

Les disques magnétiques

Il s'agit (fig. 1.3) d'une pile de disques magnétiques tournant en permanence autour de leur axe principal. Sur chaque face de ces disques se trouvent des pistes circulaires. La pile de disques peut être généralement montée ou démontée sur une machine appelée : unité de lecture-écriture disque. Lorsque la pile est en place, à chaque face correspond une tête de lecture-écriture. Ces têtes de lecture-écriture sont solidaires entre elles par un bras de commande. Si ce bras est fixe, les têtes de lecture-écriture découpent dans la pile de

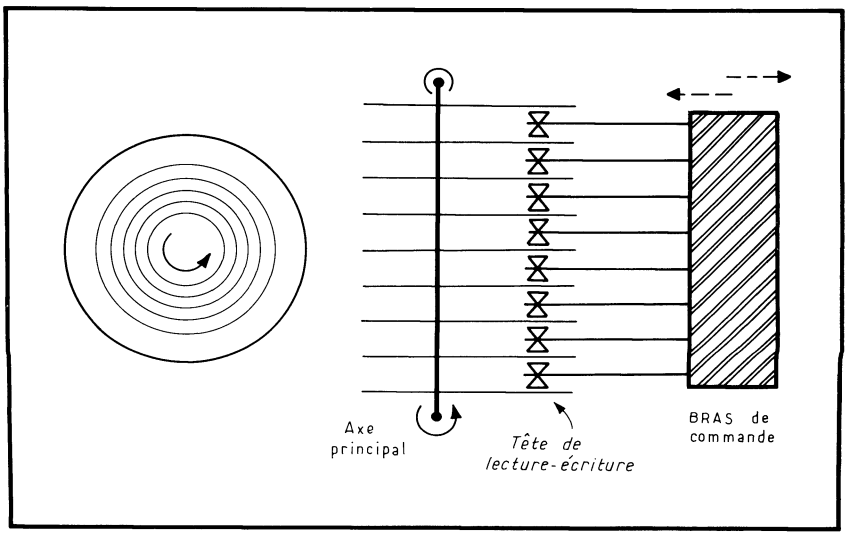


Fig. 1-3. — Disques magnétiques.

disques, qui tourne sans cesse, un *cylindre*. Pour se positionner sur une piste donnée, il suffit de placer le bras de commande sur le bon cylindre (temps mécanique) et ensuite de sélectionner la bonne tête de lecture-écriture (temps électronique). Pour lire sur cette piste, il faut alors attendre le temps nécessaire à la rotation pour que l'information défile sous les têtes de lecture-écriture.

Un signal spécial permet de déterminer physiquement un début de piste.

A noter aussi que sur une piste d'un disque nous n'avons, en principe, qu'un seul canal. Les différentes particules magnétiques définissant un caractère sont placées les unes à la suite des autres. On dit qu'un caractère a la représentation série par opposition à la bande où ces différentes particules apparaissent en même temps (représentation parallèle).

Les disques magnétiques mettent quelques dizaines de millisecondes à faire un tour. Le nombre de faces varie de 10 à 100, le nombre de cylindres de 100 à 500. On peut mettre de 1000 à 10 000 caractères par piste.

Une autre technique est utilisée dans « les disques magnétiques à têtes fixes » : à chaque piste correspond une tête de lecture-écriture. Il n'y a alors plus de bras à placer mécaniquement sur le bon cylindre. Le choix de la bonne piste demande seulement un temps électronique. On a ainsi accès aux informations plus rapidement mais avec une technologie plus compliquée.

Les tambours magnétiques

Les informations sont placées sur la surface d'un cylindre selon des pistes circulaires faisant le tour de ce cylindre. Ce cylindre tourne sans cesse, à vitesse constante autour de son axe principal. Des têtes de lecture-écriture, placées en face de chaque piste, permettent de lire et d'écrire. On peut sur un tambour magnétique enregistrer les signaux codifiant un caractère soit en série (comme sur les disques) soit en parallèle (comme sur les bandes) (fig. 1.4).

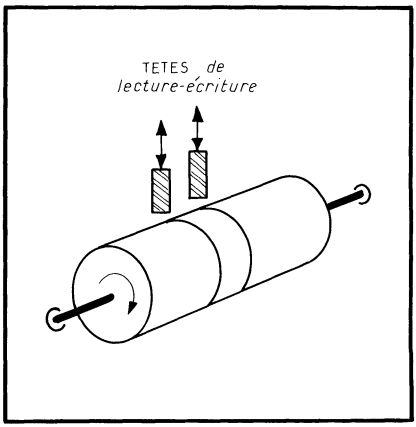


Fig. 1-4. — Tambours magnétiques.

Du point de vue volume d'information un tambour correspond à un cylindre de disque par contre le temps d'accès est plus rapide car il n'y a pas de bras à déplacer.

Les feuillets magnétiques

Les informations sont placées sur des cartes magnétiques lesquelles sont rangées en cellule. Il est possible à un bras d'une unité de lecture-écriture d'aller chercher dans un numéro de cellule donné une carte donnée et de la placer sous une tête de lecture-écriture. On crée ainsi des mémoires de très grande capacité avec toutefois un temps d'accès plus long qu'avec les disques.

2.5. — L'ÉCRITURE HUMAINE

Il est difficile pour une machine de lire notre écriture. Cela est dû au fait que nous ne lisons pas, n'écrivons pas, caractère par caractère, ni même mot par mot, mais ensemble de mots par ensemble de mots. Les différentes tentatives, qui ont été faites jusqu'à présent pour rendre notre écriture lisible par une machine, se sont bornées à une normalisation des caractères et à une lecture caractère par caractère. Citons :

Le Mark-Sensing

Il consiste à donner sur des cartes non perforées un coup de crayon à carbone sur le chiffre que l'on veut écrire. Pour rendre ce coup de crayon lisible, on l'étend sur plusieurs colonnes de la carte. Deux balais de lecture sur une machine permettent de déterminer, en trouvant contact ou non-contact, la position de ce coup de crayon.

Les lecteurs de caractères magnétiques

Ces lecteurs peuvent lire des caractères écrits sur du papier avec une encre spéciale contenant des particules magnétiques très fines.

Les lecteurs optiques

Ces lecteurs peuvent lire des caractères écrits sur du papier avec une encre ordinaire.

Ces deux types de lecteurs sont capables de lire des informations préimprimées sur un document. Les caractères doivent se présenter sous des formes normalisées. (Citons par exemple le numéro de compte sur un chèque. Une machine pourra le ranger lorsqu'il aura été complété.)

Le balayage des caractères peut s'effectuer de différentes manières :

— par comparaison optique point par point avec des caractères enregistrés dans la mémoire du lecteur;

— par analyse matricielle, par comparaison également mais avec des matrices (division de la zone occupée par le caractère en petits carrés sombres ou clairs);

— par analyse des traits. On compte le nombre d'intersections avec le caractère, le nombre de zones sombres rencontrées, par des balayages horizontaux ou verticaux ou diagonaux;

— par le suivi du contour. Cette méthode, plus souple mais plus difficile à mettre au point que les précédentes, permet de lire des caractères écrits par l'homme.

On impose à celui-ci d'écrire chacun de ces caractères dans des cases bien déterminées d'un document. Les caractères doivent être écrits suivant des normes faciles à comprendre.

Le lecteur optique peut placer ces documents sous son unité de lecture, vérifier que ce document est bien en place, positionner son rayon lumineux sur le caractère (par programmation, on donne au lecteur les coordonnées X1 Y1 du début de ligne, les coordonnées X1 Y2 de la fin de ligne). Un rayon lumineux (fig. 1.5), très fin, (par réflexion ou par réfraction) détermine les contours par des balayages successifs autour du caractère. Il y trouve des directions privilégiées, des points de rebroussement, des intersections. La machine peut alors chercher dans une table, le caractère correspondant. (Par exemple le chiffre 2 est le seul à avoir une base horizontale, il a au-dessus successivement une tangente oblique, une tangente verticale puis une tangente horizontale...)

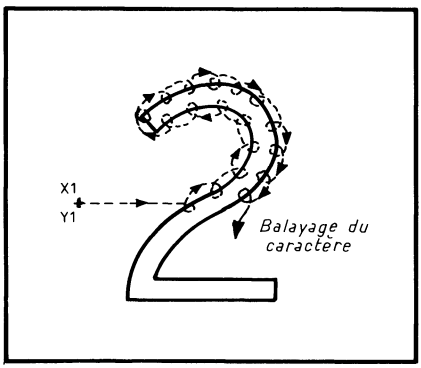


Fig. 1-5. — Lecture d'un caractère par la méthode du suivi du contour.

On peut dire à l'heure où nous écrivons ces lignes que très bientôt la lecture directe de l'écriture humaine sera plus un problème d'éducation de l'homme qu'un problème technique. On remplacera le pénible travail de perforation des cartes par un problème de normalisation de l'écriture.

2.6. — LES SUPPORTS DIVERS

Au fur et à mesure que l'on avance, dans le temps, on découvre sans cesse de nouveaux supports. Citons encore :

Les tables traçantes : une plume dessine des courbes en suivant des coordonnées x et y.

Les écrans des tubes à rayons cathodiques.

Les organes à réponse vocale.

Les différents claviers permettant d'entrer directement des informations (à distance ou non). L'enregistrement des informations sur bande magnétique sans passer par l'intermédiaire sur cartes perforées.

Les microfilms permettant d'emmagasiner un gros volume d'informations qu'il n'est pas nécessaire d'imprimer, l'impression sur papier étant beaucoup plus lente.

3. — Le matériel « classique »

On désigne sous ce vocable un ensemble de machines électromécaniques capables de lire des cartes perforées et à partir de ces cartes d'effectuer un traitement élémentaire. Certes, le matériel classique est de nos jours complètement désuet, mais l'étude de ce matériel, procédant par fonction simple, permet ensuite de comprendre plus facilement la logique de l'ordinateur, réalisant en même temps toutes ces fonctions.

On distingue :

3.1. — *LA TRIEUSE* capable de trier (classer) un paquet de cartes suivant une certaine séquence. On verra, lorsqu'on expliquera le fonctionnement des tris que, si le numéro (argument) sur lequel on veut que les cartes soient triées fait « *n* » colonnes, il faut exécuter « *n* » passages. (Nous verrons cette machine au chapitre IX.1.)

3.2. — *L'INTERCLASSEUSE* capable de fusionner en un seul paquet (fichier) deux paquets (fichiers) de cartes triées dans la même séquence (fig. 1.6). Il est possible de faire tomber dans d'autres cases de réception certains cas particuliers.

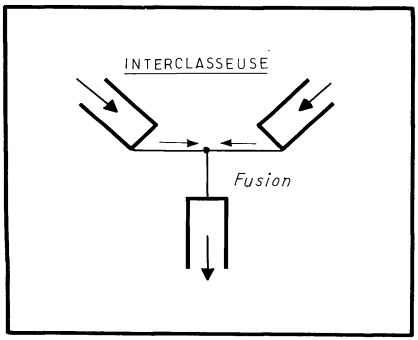


Fig. 1-6. — Fusion de deux paquets de cartes sur une interclasseuse.

3.3. — LA REPRODUCTRICE (fig. 1.7)

Capable de reproduire des cartes — certaines colonnes de ces cartes — ou de déplacer le contenu de certaines colonnes dans d'autres colonnes.

Capable aussi de reproduire le contenu de la première carte dans toutes les cartes suivantes. C'est ce qu'on appelle de la perforation série. Dans ce cas, un seul chemin de carte suffit.

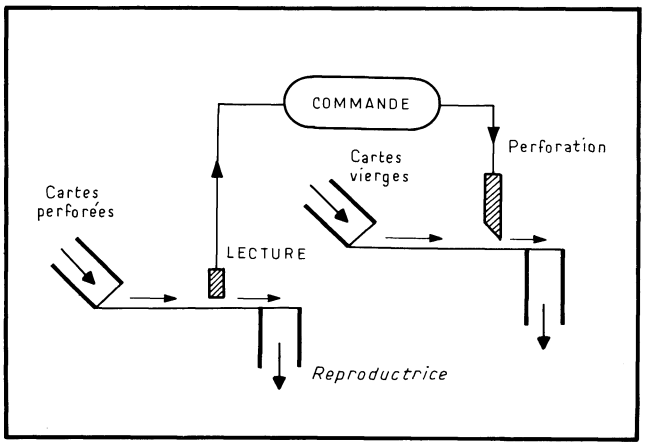


Fig. 1-7. — La reproductrice.

3.4. — LA CALCULATRICE capable de multiplier, diviser, additionner, soustraire des informations contenues dans des cartes et de perforer dans ces cartes le résultat des opérations effectuées.

3.5. — LA TRADUCTRICE DE CARACTÈRES capable d'imprimer en clair sur la carte, en haut de chaque colonne le caractère correspondant à la perforation, ceci afin de rendre le contenu de la carte rapidement lisible.

3.6. — LA TABULATRICE capable, à partir des cartes perforées, d'imprimer un état, tout en faisant des tableaux récapitulatifs.

Il est souvent possible de connecter entre elles plusieurs de ces machines, par exemple une tabulatrice peut commander une reproductrice. A côté de ce matériel, on peut ranger le matériel nécessaire à la mise de l'information sur cartes perforées.

On distingue la perforatrice, capable de perforer des cartes à l'aide d'un clavier de machine à écrire, et la vérificatrice, semblable à la perforatrice, mais où les poinçons sont remplacés par des balais permettant de vérifier que les trous déjà perforés correspondent à ce que l'on frappe sur le clavier.

4. — L'ordinateur digital

4.1. — COMPOSITION

Il se compose :

D'une *unité centrale* comprenant elle-même :

— Des générateurs d'impulsions — ces impulsions sont émises à intervalles réguliers (comme les battements d'un cœur). On appelle temps de base de l'ordinateur le temps qui sépare deux impulsions successives. Il s'exprime en microsecondes, voire en nanosecondes (milliardième de seconde) : 10^{-9} s, (la lumière ne parcourt que 30 cm pendant une nanoseconde). Ce temps de base peut varier d'un point à l'autre de l'ordinateur.

— Des circuits logiques aiguillant et contrôlant ces impulsions. Citons les circuits de base ET, OU à deux entrées et une sortie. Dans les tableaux de la figure 1.8 le symbole 0 signifie : pas d'impulsion, le symbole 1 : une impulsion. A un état d'entrée correspond un état de sortie. Ces tableaux sont aussi appelés des *Tables de décision.*

CIRCUIT ET

Entrée 1	Entrée 2	Sortie
0	0	0
0	1	0
1	0	0
1	1	1

CIRCUIT OU inclusif

Entrée 1	Entrée 2	Sortie
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 1-8. — Table de décision du circuit ET et du circuit OU.

(par exemple avec le circuit ET pour avoir une impulsion en sortie il faut avoir une impulsion à la fois sur l'entrée 1 et sur l'entrée 2) :

— des organes de commande pouvant mettre en ligne tel ou tel circuit;

— des registres spécialisés; ce sont des mémoires modifiables contenant à chaque instant l'état d'une partie donnée de l'ordinateur;

— des indicateurs définissant un état particulier de l'ordinateur.

D'une *Mémoire Centrale*, généralement à ferrites.

— Rappelons brièvement le fonctionnement de la ferrite.

Ce sont des noyaux de fer à cycle d'hystérésis rectangulaires disposés dans des plans comme dans la figure 1.9.

Les noyaux de fer sont traversés par des fils horizontaux constituant la sélection horizontale, par des fils verticaux constituant la sélection verticale.

En l'absence de courant, chaque ferrite possède deux états stable : les positions 0 et 1 représentées sur le cycle d'hystérésis (fig. 1.10). Chaque ferrite

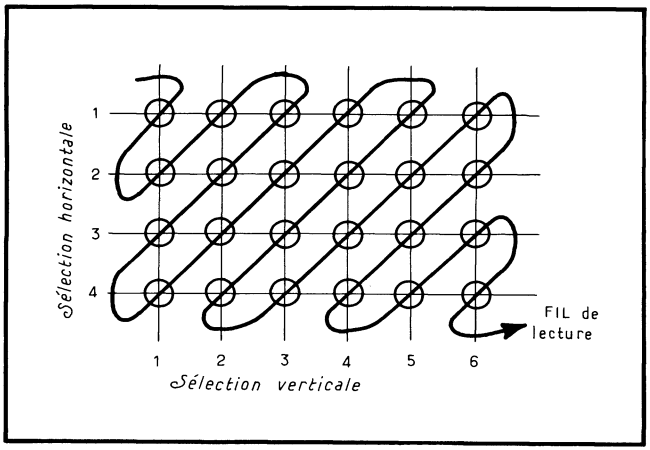


Fig. 1-9. — Disposition des ferrites dans un plan de mémoire.

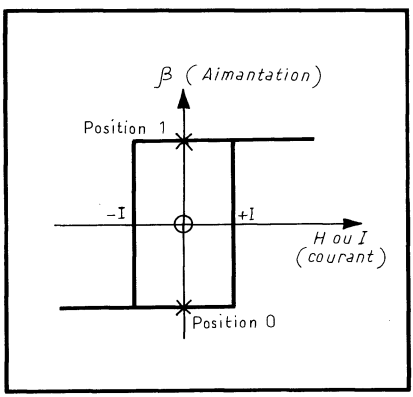


Fig. 1-10. — Cycle d'hystérésis rectangulaire.

se trouve au croisement d'un fil vertical et d'un fil horizontal. Supposons que l'on veuille positionner la ferrite située au croisement du fil horizontal 2 et du fil vertical 3 sur l'état stable 1. On va envoyer sur ces deux fils une impulsion très légèrement, supérieure à $+I/2$. Seule la ferrite située au croisement de ces deux fils recevra une impulsion supérieure à $+I$. Si elle est sur la position stable 0 elle décrira la moitié du cycle d'hystérésis et viendra sur la position stable 1. Si elle est sur la position 1 elle restera dans cet état. Autrement dit, notre ferrite sera mise dans l'état 1. On dit qu'on a écrit sur cette ferrite, qu'on lui a mis un *top*, un *bit*. Elle restera dans cet état tant qu'elle ne recevra pas d'impulsion supérieure (en valeur absolue) à $-I$.

Supposons maintenant que l'on veuille lire le contenu de cette ferrite. Pour cela on fait traverser les ferrites par un 3e fil appelé fil de lecture, disposé comme il est indiqué sur la figure. On va envoyer une impulsion légèrement supérieure à $|-I/2|$ en même temps sur les deux fils (horizontal 2 et vertical 3). Seule la ferrite située au croisement de ces deux fils recevra une impulsion supérieure à $-I$. Si elle est sur la position 0 elle ne changera pas de position.

Si, par contre, elle est sur la position 1 elle va décrire la moitié du cycle d'hystérésis et basculer sur la position 0. Il y aura variation de flux et on enregistrera une impulsion sur le fil de lecture.

Il faut noter que cette méthode de lecture détruit le contenu de la ferrite. Il faut donc prévoir de régénérer ce contenu.

Plusieurs ferrites sont groupées ensemble afin de pouvoir définir un caractère. Par exemple, pour définir un chiffre compris entre 0 et 9 il faut au moins 4 ferrites,

0	0	0	0	numérotés 1, 2, 4 et 8.
1	2	4	8	

Pour représenter le chiffre 5, on met « un top » sur les ferrites 1 et 4. Pour représenter une lettre, il faut en plus d'autres ferrites. On appelle ce groupe de ferrites nécessaires pour mettre un caractère dans la mémoire une « position de mémoire ». Généralement, pour les contrôles, on y adjoint une ferrite spéciale dite d'imparité pour que le nombre de tops en place dans une position de mémoire soit toujours pair ou impair.

Chacune de ces positions porte un numéro de 00 000 à *n; n* dépendant de la taille de mémoire. Ce numéro s'appelle l'*adresse* de la position, et la mémoire contient $n+1$ positions.

Parfois, en plus de la mémoire principale, l'ordinateur peut comporter d'autres mémoires à ferrites plus spécialisées.

Remarque : En dehors des mémoires à ferrites nous connaissons :

— *les mémoires à aiguilles :* où les ferrites sont remplacées par des aiguilles sur lesquelles se trouvent un dépôt ferromagnétique de 4/100 de millimètre. Les conducteurs sont enroulés autour de ces aiguilles;

— *les mémoires à films minces :* une très faible couche de dépôt ferromagnétique de 1/1 000 à 1/10 000 de millimètre est déposé sur un support non magnétique. En l'absence de tout courant il n'y a aucun magnétisme rémanent dans l'axe perpendiculaire à la couche. Le seul magnétisme rémanent possible se trouve dans le sens de cette couche.

L'envoi d'un courant électrique provoquant un champ magnétique perpendiculaire à la couche génère une variation d'induction que l'on peut mesurer. Lorsque ce courant électrique cesse on n'a rien changé au magnétisme rémanent initial. On peut donc lire le contenu de ces mémoires sans le détruire.

— *les mémoires à semi-conducteurs :* Sur une surface de quelques mm^2 d'un circuit intégré peuvent se trouver, en plusieurs couches, plusieurs milliers de résistances, de diodes et de transistors.

— Citons aussi les *mémoires mortes*, au contenu indestructible, fixé une fois pour toutes (voir à la fin de ce chapitre la microprogrammation). *Elles* sont constituées de selfs ou de capacités.

Des organes de calcul et de logique permettent, à partir du contenu de certains registres, d'effectuer des opérations arithmétiques, des opérations logiques. Par exemple, on peut comparer deux registres et positionner des indicateurs suivant le résultat de la comparaison.

Des unités d'entrée et de sortie permettent d'entrer et de sortir les informations.

Citons :

— lecteur de cartes perforées;

— perforateur de cartes;

— imprimante;

— lecteur de rubans perforés;

— perforateur de rubans;

— dérouleur-lecteur de bandes magnétiques;

— unité de disques magnétiques;

— unité de tambours magnétiques;

— unité de lecture optique de documents;

— unité de feuillets magnétiques;

— centraux pouvant relier l'ordinateur à des unités d'entrée-sortie placées à distance;

— tables traçantes;

— unité d'écran de télévision (display);

— unité d'impression sur des microfilms;

— unité permettant de connecter un ordinateur à un autre.

— unité construite spécialement pour un traitement bien déterminé (par exemple unité de commande ou de contrôle d'un processus industriel).

Cette liste n'est pas close.

— Enfin un *pupitre* de commande permettant à l'opérateur de donner ou de modifier des ordres à l'ordinateur. Généralement, ce pupitre permet, au moins, de faire trois opérations élémentaires :

— aller voir le contenu d'une position de mémoire;

— modifier ce contenu;

— modifier le contenu d'un registre que l'on appelle compteur ordinal ou registre instruction.

La plupart du temps ces opérations se font à l'aide d'un télétype (machine à écrire).

Des unités de synchronisation, de contrôle et de décodage-recodage permettent entre les unités d'entrée-sortie et l'ordinateur le ou la :

a) *Synchronisation :* le temps dans un ordinateur se mesure en microseconde voire en nanoseconde (unité de temps des opérations électroniques) — le temps dans une unité d'entrée-sortie se mesure en milliseconde (unité de temps des opérations mécaniques) — par exemple un lecteur de cartes fonctionnant à 1200 cartes par minute demande 50 millisecondes pour lire une carte). Il faut synchroniser ces différences de temps.

b) *Contrôle :* il faut vérifier la validité de ce que l'on rentre et de ce que l'on sort. Par exemple : vérifier la parité des caractères sur une bande magnétique, vérifier que le caractère lu sur une carte correspond à un caractère connu de l'ordinateur, relire une carte que l'on vient de perforer, faire un contrôle par écho du fonctionnement des marteaux de l'imprimante.

c) *Codification :* La codification des caractères sur les supports des entrées-sorties n'est pas la même que dans la mémoire principale. Il faut décoder et recoder.

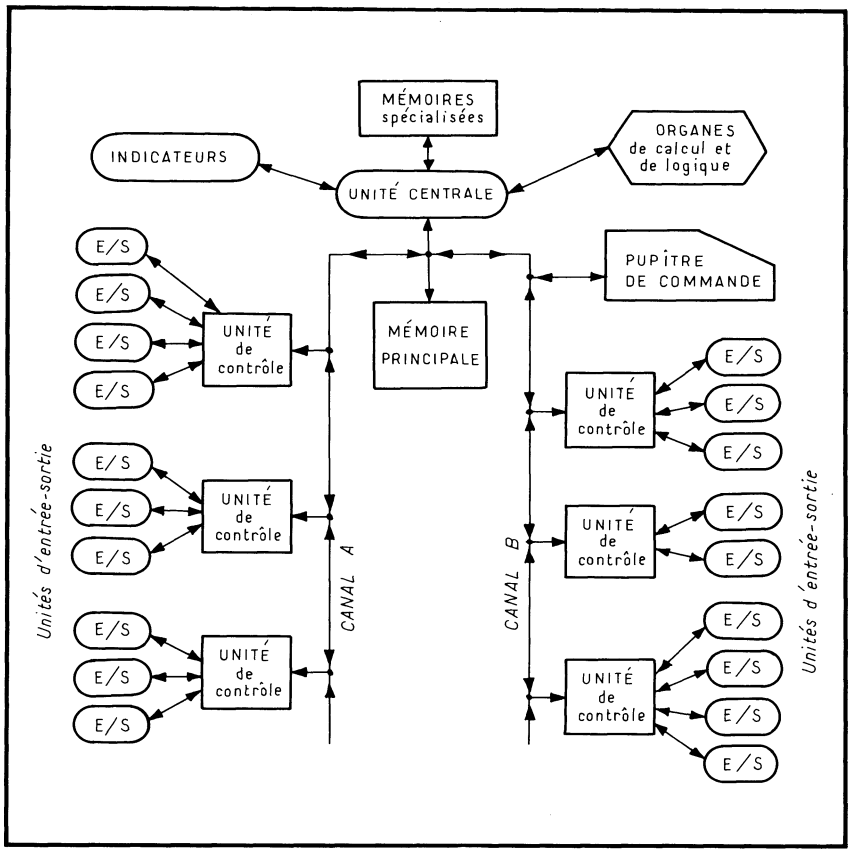


Fig. 1-11. — Schéma général d'un ordinateur.

Des Canaux permettant de véhiculer (transmettre) l'information entre les unités précédentes et la mémoire principale sous le contrôle de l'unité centrale.

En résumé, on peut schématiser un ordinateur comme dans la figure 1.11.

4.2. — PRINCIPE DU FONCTIONNEMENT D'UN ORDINATEUR

Dans la mémoire principale, on place des *données constantes*, par exemple le chiffre 3 si on a des multiplications par 3 à effectuer. On place aussi des *données variables.* Ce sont des zones dont le contenu est modifié sans cesse.

On a des compteurs et des positions permettant de recueillir le résultat des opérations arithmétiques, des zones d'entrée-sortie permettant de recueillir ou d'envoyer les données des unités de lecture ou d'écriture.

Le programme : constitué par une suite d'instructions, une instruction étant un ordre élémentaire que l'on donne à l'ordinateur. Souvent le programme est contenu dans la mémoire.

— On a en principe 2 types de mémoire principale :

La mémoire à mots :

La mémoire principale est divisée en mots — chaque mot étant composé d'un nombre fixe de positions. Par exemple, si chaque mot prend 8 positions de mémoire, pour mettre en mémoire un nombre de 3 chiffres, on utilise 1 mot; pour mettre en mémoire un nombre de 9 chiffres (comme 9 est supérieur à 8) on utilise 2 mots consécutifs : (fig. 1.12).

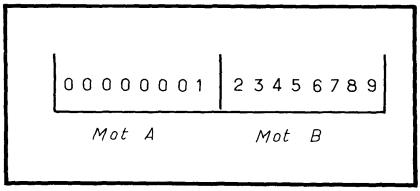


Fig. 1-12. — Représentation d'un nombre de 9 chiffres dans une mémoire à mots, avec 1 mot = 8 chiffres.

La mémoire à caractère :

Dans ce cas, pour désigner un nombre dans la mémoire, on dit qu'il se trouve dans la position de mémoire contenant son chiffre de gauche (ou de droite suivant le type d'ordinateur) :

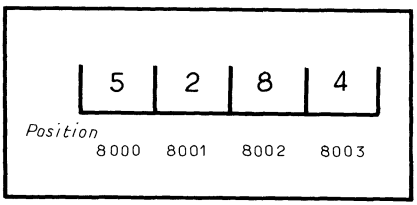


Fig. 1-13. — Le nombre 5 284 se trouve à l'adresse 8 000.

Dans la figure 1.13, on dit que le nombre 5284 placé dans les positions de mémoire 8000 à 8003 se trouve à *l'adresse 8000*. On dit aussi que 5284 est le *contenu* et 8000 le *contenant*.

Mais pour déterminer complètement le nombre, il faut aussi le limiter à droite, car la longueur de ce nombre peut être absolument variable. Pour cela, trois solutions :

1re manière : appeler le nombre en lui donnant deux adresses : les adresses 8000 et 8003. Cette solution est rarement utilisée car elle est lourde.

2e manière : ajouter à chaque position de mémoire une ferrite supplémentaire appelée : marque de mot. On mettra un top sur cette ferrite dans la dernière position du nombre.

Par exemple : (fig. 1.14).

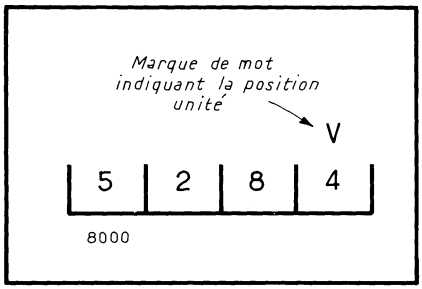


Fig. 1-14. — La marque de mot.

3e *manière :* indiquer la longueur du nombre directement dans l'instruction chargée de faire un traitement sur ce nombre.

Nous allons maintenant donner le principe de fonctionnement d'un ordinateur en prenant comme exemple un ordinateur à caractères utilisant le concept de la marque de mot.

Nous avons mis dans la mémoire principale, à partir de la position 800, les caractères suivants : (fig. 1.15)

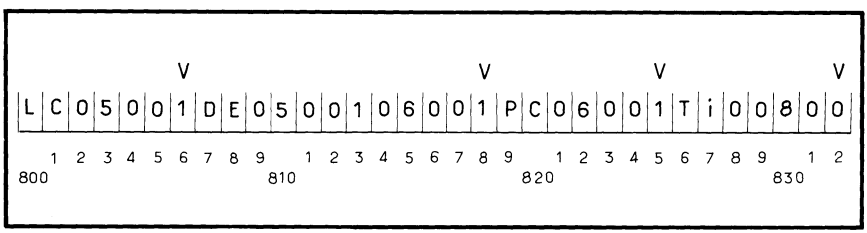


Fig. 1-15. — Caractères à partir de la position 800.

Les signes « V » indiquent les marques de mot. Tout ceci semble de l'hébreu, mais il s'agit d'une codification que nous allons expliquer pas à pas. C'est le langage de l'ordinateur que l'on appelle *langage absolu.* De 00 800 à 806, on dit que l'on a une instruction, il en est de même pour 807 à 818, pour 819 à 825, et pour 826 à 832.

Nous avons aussi placé des marques de mot dans les positions de mémoire 5080 et 6080. Nous verrons plus loin comment on place ces différentes informations dans l'ordinateur.

L'ordinateur possède des registres. Il y en a un que l'on appelle *compteur ordinal* ou registre instruction, qui dirige le programme. D'autres registres s'appellent : registre code opération, registre adresse A, registre adresse B, registres intermédiaires, etc.

Pour notre exemple, on considère que nous avons :

— un compteur ordinal (ou registre instruction) à 5 positions;

— un registre code opération à 2 positions;

— un registre adresse A à 5 positions;

— un registre adresse B à 5 positions.

Avec le pupitre nous avons mis dans le registre instruction la valeur 00 800 (fig. 1.16).

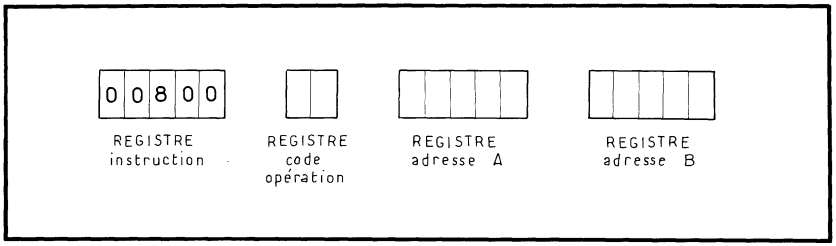


Fig. 1-16. — 00800 dans le registre instruction.

On appuie sur le bouton départ de l'ordinateur. L'unité centrale va :

1 — aller voir le contenu du registre instruction. Elle lit 00800;

2 — elle va lire le contenu de la position de la mémoire principale 00800 : « L » et le placer dans la première position du registre code opération (si la mémoire principale est à ferrite le contenu de la position de mémoire 00800 est régénéré);

3 — elle se pose la question : « Y a-t-il une marque de mot sur la position que je viens de lire? » la réponse est non, « je continue »;

4 — elle fait +1 sur le registre instruction. Celui-ci contient maintenant 00801;

5 — elle va lire le contenu de la position de mémoire 00801. Elle place ce contenu dans la 2e position du code opération (tout en régénérant la position 00801 de la mémoire);

6 — y a-t-il une marque de mot sur la position 00801? — « Non, je continue »;

7 — elle fait +1 sur le compteur ordinal, ce qui donne 00802;

8 — elle va chercher le contenu de la position de mémoire 00802 et le placer dans la 1re position du registre adresse A (tout en régénérant la position 00802 de la mémoire);

9 — « y a-t-il une marque de mot sur la position 00802? « Non, je continue. »

et ainsi de suite, en effectuant +1, +1 sur le registre instruction jusqu'à ce que l'on ait dans les registres la situation de la figure 1.17.

Arrivée à ce point, l'unité centrale va une nouvelle fois se poser la question

« Y a-t-il une marque de mot sur la position 00806? » La réponse cette fois est oui.

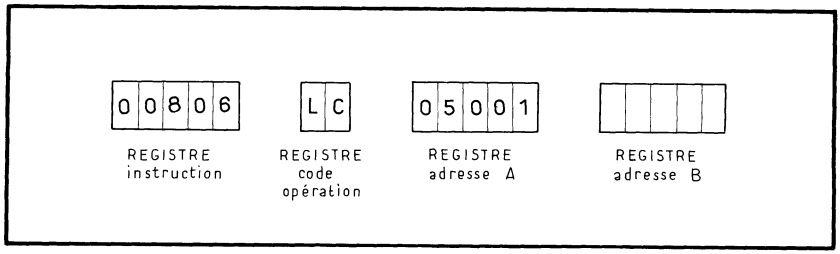


Fig. 1-17. — Chargement d'une instruction dans les registres.

Alors, on continue par les opérations :

1 — l'unité centrale fait encore +1 sur son registre instruction afin de se positionner sur l'instruction suivante en 00807;

2 — l'unité centrale vérifie la validité de cette instruction, par exemple : la longueur de l'instruction est-elle correcte? (on vérifie que l'on termine bien à la fin d'un registre).

— le registre adresse A ne contient-il bien seulement que des chiffres?

— le code opération (contenu du registre code opération) est-il connu de l'ordinateur?

Si tout est bon, on continue, sinon l'ordinateur signale : erreur dans le programme;

3 — l'unité centrale décodifie le contenu du registre code opération et examine ce qu'il veut dire, par exemple ici « LC » veut dire « Lecture carte ».

A ce moment, on dit que l'ordinateur a terminé la *phase analyse* de l'instruction, il peut passer à la *phase exécution* de cette instruction. Avant d'exécuter un ordre, il faut le comprendre, c'est ce que fait l'ordinateur.

Il va maintenant exécuter (phase exécution) l'instruction : lecture d'une carte avec embrayage de la lecture sur l'unité de lecture carte. Le contenu de la carte lue est envoyé dans la mémoire principale à partir de la position indiquée par le registre adresse A, à raison d'une colonne de la carte par position de mémoire. C'est-à-dire 1re colonne dans la position 05001, 2e colonne dans la position 05002 jusqu'à 80e colonne dans la position 05080. En même temps l'unité centrale contrôle cette exécution.

On peut schématiser ce travail de l'ordinateur par le diagramme de la figure 1.18.

On passe donc à l'instruction suivante indiquée par le contenu du registre instruction 00807. Et de la même manière que précédemment on va placer dans les registres toujours en faisant + 1, + 1 sur le registre instruction, les informations comme dans la figure 1.19.

L'instruction est complètement chargée dans les registres car on a rencontré une marque de mot sur la position 00818.

On vérifie que l'instruction est valide, qu'on a bien terminé à la fin d'un registre.

On décodifie le code opération « DE » — il indique ici que c'est un « déplacement » du contenu d'une partie de la mémoire principale dans une autre.

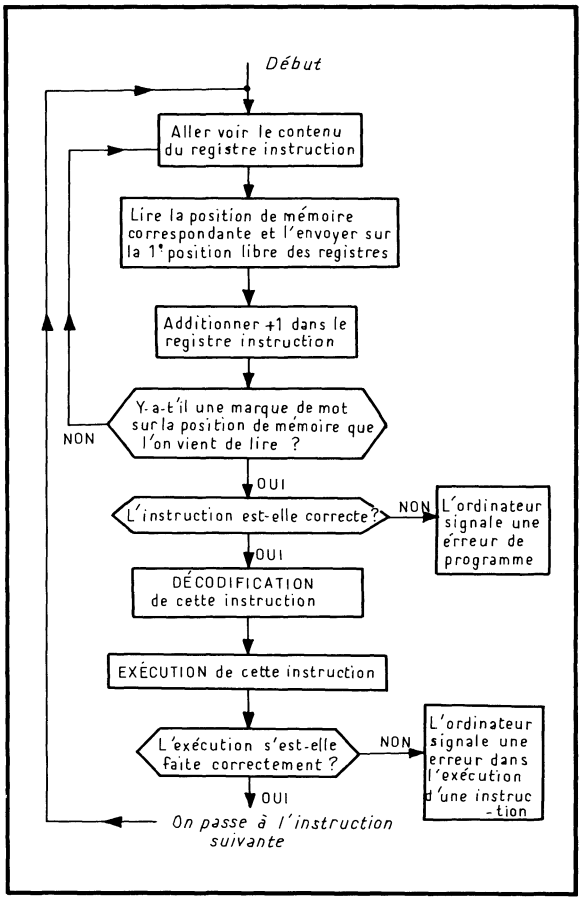


Fig. 1-18. — Travail de l'ordinateur pour exécuter une instruction. La phase d'analyse précède la phase exécution.

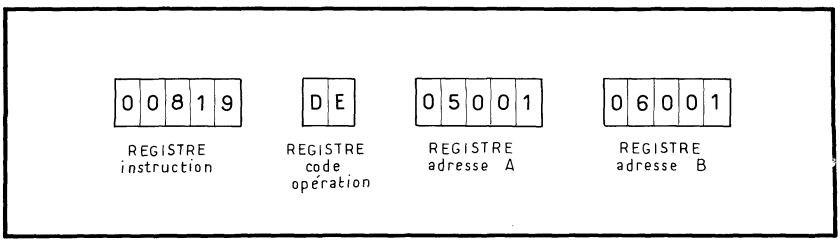


Fig. 1-19. — Une nouvelle instruction se trouve dans les registres.

L'exécution de cette instruction peut par exemple s'effectuer de la manière suivante :

1° lecture de la position de mémoire 05001 (avec régénération de cette position) et écriture de ce contenu sur la position 06001;

2° faire + 1 sur le registre A et + 1 sur le registre B, ceux-ci contiennent maintenant 05002 et 06002;

3° y a-t-il une marque de mot dans la position de mémoire que l'on vient de transporter? Si la réponse est oui, l'exécution de l'instruction est terminée — si la réponse est non, on continue;

4° envoi de la position de mémoire 05002 dans la position de mémoire 06002;

5° faire + 1 sur le registre A et + 1 sur le registre B;

6° y a-t-il une marque de mot dans la position transportée? et ainsi de suite jusqu'à ce que la réponse à la question précédente soit affirmative.

Comme nous avons placé une marque de mot sur la position de mémoire 05080, à la fin de l'exécution de cette instruction on aura dans les registres la situation de la figure 1.20.

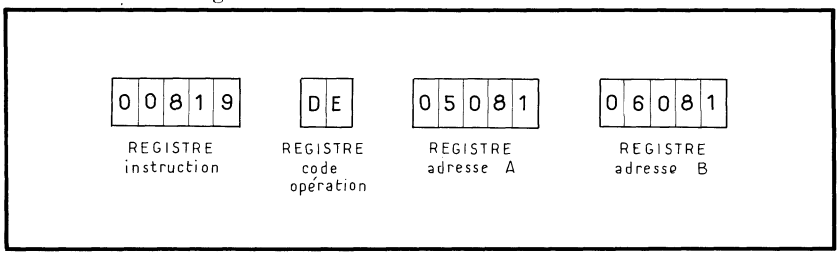


Fig. 1-20. — Contenu des registres à la fin de l'exécution de l'instruction.

et on aura déplacé le contenu des 80 positions de mémoire 05001 à 05080 respectivement dans les positions de mémoire 06001 à 06080.

On peut représenter la phase exécution de cette instruction par le diagramme de la figure 1.21.

L'instruction suivante est indiquée par le contenu du registre instruction : 00819. (A remarquer aussi que les registres A et B contiennent des renseignements : 05081 et 06081 qui peuvent être utilisés, dans certains cas, dans certains ordinateurs, par les instructions suivantes, cela s'appelle du chaînage d'instruction.) On va analyser l'instruction commençant en 00819, toujours de la même manière, en effectuant + 1, + 1 sur le contenu du registre instruction.

A la fin de cette phase les registres sont comme la figure 1.22 (on ne s'occupe pas ici du contenu du registre B : 06081) :

L'unité centrale contrôle l'instruction, puis décode le contenu du registre code opération : PC veut dire : « perforation d'une carte ».

On va perforer une carte en prenant le contenu de la mémoire à partir de la position indiquée par le registre A. C'est-à-dire que l'on va prendre les contenus des positions 06001 à 06080 et les perforer, une position pour une colonne dans une carte.

Comme on a précédemment mis dans cette zone de mémoire le contenu de la carte lue, on va perforer une carte identique à celle que l'on a entrée. On fait de la reproduction.

Une fois cette carte perforée, on passe à l'analyse de l'instruction suivante, toujours de la même manière jusqu'à ce que les registres soient dans la position de la figure 1.23.

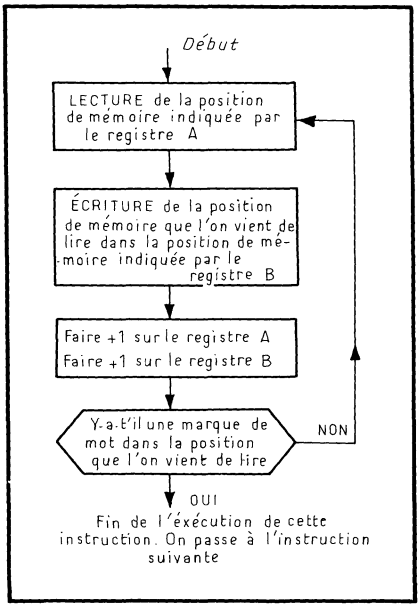


Fig. 1-21. — Exécution de l'instruction déplacement du contenu d'une partie de la mémoire principale dans une autre. Le déplacement a lieu position par position.

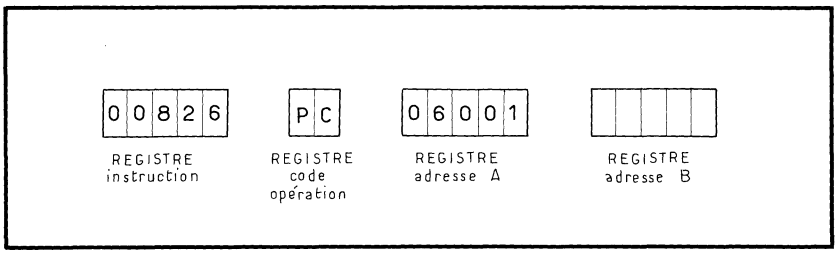


Fig. 1-22. — L'instruction PC 06001 se trouve dans les registres.

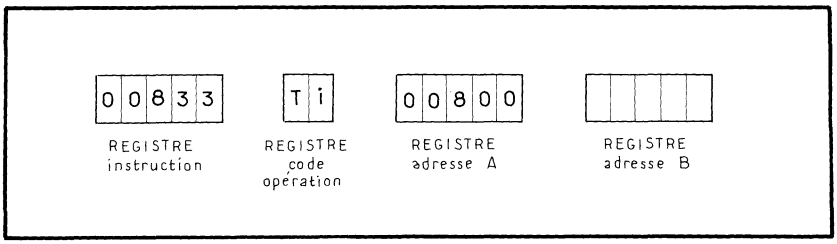


Fig. 1-23. — L'instruction TI 00800 se trouve dans les registres.

Contrôle et décodification de l'instruction TI : elle correspond ici à « transfert inconditionnel ». L'exécution consiste à prendre le contenu du registre adresse A et à le placer dans le registre instruction (fig. 1.24).

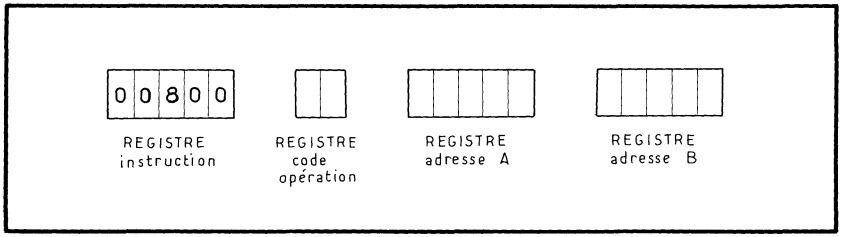


Fig. 1-24. — Heureux qui comme Ulysse a fait un beau voyage.

Et l'on continue en considérant le contenu du registre instruction : 00800. On est revenu au point de départ! On tourne dans une boucle.

On représente le schéma de ce « programme » comme dans la figure 1.25.

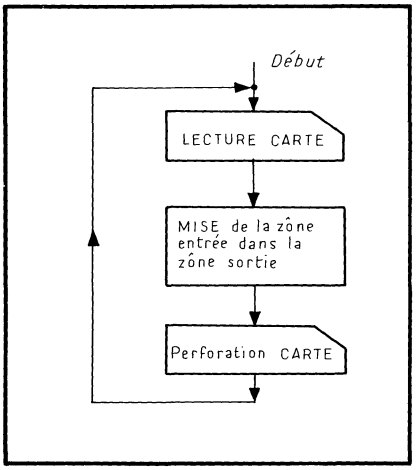


Fig. 1-25. — Ordinogramme : reproduction d'un paquet de cartes perforées.

Ce schéma s'appelle un *ordinogramme.*

Nous répétons : Les instructions placées dans les positions 00800 à 00832 respectent un codage connu de l'ordinateur. C'est ce qu'on appelle *langage* machine ou *langage absolu.* Nous verrons que le programmeur peut écrire d'une manière plus proche du langage humain.

Par exemple, il écrit quelque chose du genre :

DÉBUT :	LECTURE D'UNE CARTE. DÉPLACEMENT DE LA ZØNE LECT—CARTE DANS LA ZØNE PERF—CARTE. PERFØ D'UNE CARTE. ALLER A DÉBUT.

et ce sera un programme spécial appelé *compilateur* qui transformera le langage du programmeur (appelé *langage symbolique*) en langage absolu. Cette dernière opération s'appelle une *compilation;* on dit aussi qu'on *compile* un programme. Nous attirons l'attention sur deux points particuliers :

1° nous représentons la lettre O par le caractère Ø barré, et le chiffre « zéro » par le caractère O non barré;

2° dans l'organigramme précédent, il manque un point important. On ne peut pas, bien sûr, tourner indéfiniment dans une boucle. Il faut aussi poser la question « Y a-t-il encore des cartes à lire? » avant de lancer la lecture d'une carte. (Transfert si la condition est réalisée : Transfert conditionnel.)

4.3. — CLASSIFICATION DES ORDINATEURS

Sur un ordinateur à mots le principe de fonctionnement est le même, sauf que le registre instruction, au lieu de progresser + 1 par + 1, progresse mot par mot. D'autre part, au lieu de transporter les informations dans la mémoire, position de mémoire par position de mémoire, le déplacement se fait mot par mot. C'est-à-dire que si un mot fait par exemple 8 positions, on transporte les 8 positions en même temps (en parallèle) donc avec gain de temps.

Par contre, la place prise par les informations en mémoire dans un ordinateur à mots est plus coûteuse. Un nombre de 3 chiffres dans un ordinateur à mots où chaque mot fait 8 positions prend 8 positions.

On peut résumer de la manière suivante :

— ordinateur à caractères : gain de place;

— ordinateur à mots : gain de temps.

On divise les applications des ordinateurs en deux catégories : scientifique et gestion. Dans une application scientifique, il y a beaucoup de calcul mais très peu d'entrées-sorties, dans une application de gestion il y a peu de calcul mais beaucoup d'entrées-sorties. Dans une application scientifique, c'est le temps de la mémoire centrale qui compte. Dans les applications de gestion ce sont les entrées-sorties et dans ce cas il n'est pas nécessaire d'avoir une mémoire coûteuse. C'est pourquoi en général on prend un *ordinateur à mots pour le scientifique — un ordinateur à caractères pour la gestion.*

Le plus souvent *l'ordinateur travaille en binaire* (la ferrite — cellule de base a 2 positions stables, mais il n'est pas interdit de concevoir un ordinateur avec cellule de base à 3 positions stables et plus).

On peut ainsi avoir le langage absolu des instructions en binaire avec un adressage de la mémoire en binaire. On gagne du temps dans la phase analyse de l'instruction et de la place en mémoire. (Lorsque le programmeur utilise un langage proche du langage absolu, on perd par contre un peu de temps à la compilation car il faut convertir ses adresses décimales en adresses binaires.)

L'organe de calcul peut aussi effectuer les opérations arithmétiques soit en décimal, soit en binaire. Lorsqu'il effectue les calculs en binaire, il faut convertir du décimal en binaire les nombres lus par la machine et inversement convertir du binaire en décimal les nombres à sortir sur imprimante. (L'homme ne connaît que le décimal.) Ces conversions prennent du temps et le calcul en binaire est surtout intéressant lorsqu'il y a beaucoup d'opérations à faire.

C'est pourquoi, généralement, il y a intérêt à effectuer :
— *dans les applications scientifiques, les calculs en binaire;*
— *dans les applications de gestion, les calculs en décimal.*

4.4. — ÉVOLUTION DANS LE HARDWARE

Un livre entier ne suffirait pas pour expliquer toutes les évolutions qui ont eu lieu dans le monde, plein de logique, de philosophie et d'astuces, du hardware. La chose est encore loin d'être figée, nous nous contentons de donner une synthèse des tendances actuelles.

La miniaturisation des circuits

En diminuant la longueur des circuits, on diminue leur temps de réponse et par suite on peut diminuer aussi le temps de base. En moins de 10 ans nous avons vu l'avènement des ordinateurs à lampes, des ordinateurs à transistors et maintenant des ordinateurs à micromodules et circuits intégrés. (Ces catégories d'ordinateurs ont été respectivement appelées première génération, deuxième génération et troisième génération.)

La microprogrammation

Parallèlement l'organisation de l'ordinateur est modifiée sans cesse. Au début on avait des circuits logiques séquentiels. Après décodification du code opération, il y avait mise en ligne des circuits logiques correspondant à ce code. C'est-à-dire qu'il y avait une séquence de circuits logiques par code opération.

Plus récemment, on a divisé l'instruction en plusieurs opérations simples, comme nous l'avons fait précédemment en décomposant la phase analyse et la phase exécution. Chacune de ces opérations simples s'appelle une *micro-instruction.* Par exemple : mise du contenu d'une position de mémoire dans un registre, régénération de cette position de mémoire, mise du contenu d'un registre dans un autre registre, addition du contenu d'un registre sur le contenu d'un autre, etc. Ce sont des opérations élémentaires que l'on peut trouver dans des instructions complètement différentes.

On codifie ces opérations élémentaires, et on place les codifications qui en résultent dans une petite *mémoire inaltérable* (mémoire que l'on peut lire mais sur laquelle on ne peut pas écrire) connectée à l'unité centrale. Cette mémoire peut, par exemple, être constituée par des capacités, ou par un système de circuits imprimés s'enroulant ou ne s'enroulant pas autour d'électro-aimants. Elle possède, elle aussi, un registre adresse et à chaque adresse correspond une opération élémentaire codifiée dans cette mémoire inaltérable. Lorsque ce registre est positionné sur une micro-instruction, cette micro-instruction déclenche l'opération élémentaire correspondante. En fonction du résultat de cette opération élémentaire, de la position de certains indicateurs, et du contenu du registre micro-instruction, il y a création dans ce registre d'une nouvelle adresse de micro-instruction. On exécute cette nouvelle micro-instruction et ainsi de suite. Ainsi donc, comme pour

les instructions du langage absolu (correspondant au programme de l'utilisateur) il y a, à un étage en dessous, un autre véritable programme appelé microprogramme. Le temps de base de ce microprogramme est bien sûr inférieur au temps de base du programme.

Les avantages du microprogramme appelés aussi parfois firmware sont :

— La diminution considérable des circuits dans l'ordinateur. Toutes les fonctions de commande élémentaires sont rassemblées dans la mémoire inaltérable. La même micro-instruction peut être utilisée pour des instructions différentes.

— Il est possible avec le microprogramme de commencer la phase exécution d'une instruction pendant que l'on termine sa phase analyse; de commencer la phase analyse de l'instruction suivante pendant que l'on termine la phase exécution de l'instruction en cours. On gagne ainsi du temps.

— La possibilité d'exécuter des microprogrammes excessivement compliqués permet d'avoir un jeu d'instructions très riche. Il n'est pas interdit de penser qu'un jour on fabriquera des instructions exécutant directement les verbes du langage symbolique. Ce jour-là le langage du programmeur sera presque identique au langage absolu. Juste retour des choses : ce ne sera plus l'homme qui devra faire convertir son langage, mais la machine qui s'élèvera jusqu'au langage humain!

La modularité et le dépannage

Une autre amélioration du hardware consiste à diviser l'ordinateur en plusieurs blocs, chacun de ces blocs en sous blocs, etc. ceci afin de faciliter le dépannage. En cas de panne il est facile, alors, de remplacer le bloc défaillant par un autre équivalent. Cette technique de division en bloc s'appelle la *modularité.*

Parfois, il est même possible pour l'ordinateur de corriger lui-même ses pannes. On divise les pannes en deux catégories : les pannes permanentes et les pannes intermittentes. Pour les pannes intermittentes, l'ordinateur a la possibilité de recommencer l'instruction qui s'est exécutée anormalement tout en conservant une image de l'incident. Pour les pannes permanentes, il est possible à l'ordinateur de mettre « hors-ligne » le circuit défaillant et de mettre « en ligne » un autre circuit absolument équivalent.

La fiabilité

Dans la construction d'un ordinateur intervient un facteur important : c'est la probabilité pour un circuit, un ensemble de circuits, de faire une erreur en fonctionnant n fois (le nombre n étant très grand). L'inverse de cette probabilité s'appelle la fiabilité. Elle intervient à tous les étages de l'ordinateur. Il y a d'abord la fiabilité des éléments de base; du micromodule, du circuit ET, de la ferrite. Il y a ensuite la fiabilité des blocs groupant plusieurs éléments de base, puis à l'étage supérieur la fiabilité de l'ordinateur pris dans son ensemble.

Plus la probabilité d'erreur est faible et plus la qualité de l'ordinateur est grande. On augmente la fiabilité par des opérations de contrôle : c'est la

ferrite supplémentaire pour vérifier la parité des caractères dans la mémoire, c'est la relecture des cartes que l'on vient de perforer, etc. Pratiquement la fiabilité des ordinateurs modernes est telle que l'ordinateur signale les erreurs qu'il fait.

A l'autre bout, il est nécessaire que la fiabilité des constituants simples corresponde à la fiabilité de l'ensemble. Un ordinateur ayant des contrôles à tous les étages mais ayant des constituants de mauvaise fiabilité, est pratiquement inutilisable car il signale sans cesse des erreurs.

La simultanéité

Une autre possibilité du hardware est ce qu'on appelle la *simultanéité :* la possibilité pour l'ordinateur d'effectuer plusieurs opérations en même temps. Mais ne fait pas de simultanéité qui veut?

C'est l'histoire humoristique du garde-barrière chargé de surveiller deux voies de chemin de fer se coupant à angle droit, l'une nord-sud, l'autre ouest-est. Un train passe tous les jours de l'ouest vers l'est à 10 heures, un autre passe tous les jours du nord vers le sud à 9 heures. Le travail du garde-barrière consiste à faire pivoter la plaque tournante de 90° après le passage de chaque train. La vie de ce garde-barrière se passe monotone, à travers la campagne verdoyante de notre France, avec la maison au sud-est de la plaque tournante, le tas de fumier au nord-est, la brouette dans le jardin du sud-ouest, et le grand-père sur sa chaise longue au nord-ouest. Lorsqu'un jour de période électorale, la Direction de la SNCF communique, à 8 h 50, par téléphone au garde-barrière l'ordre suivant : « Dorénavant, le train venant de l'ouest à 10 h passera à 9 h ». Le problème est : Qu'auriez-vous fait à la place du garde-barrière devant cette simultanéité? et la réponse est

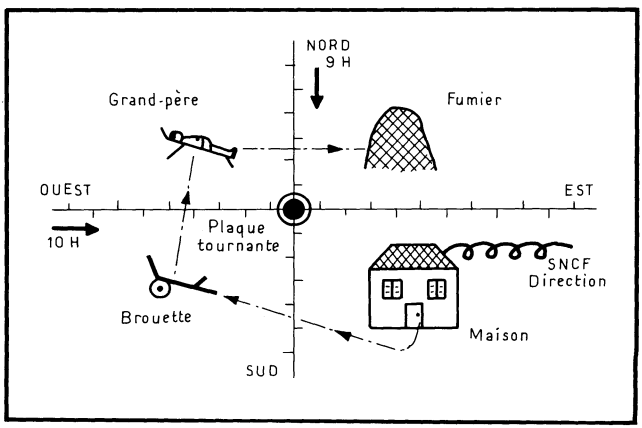


Fig. 1-26. — Le problème du garde-barrière.

simple. Il est sorti rapidement de sa maison, il est parti vers l'ouest chercher sa brouette, et, avec elle, s'en est allé vers le nord afin de porter son grand-père. Il a conduit le tout vers l'est, il a posé son grand-père sur le tas de fumier puis il a prononcé cette phrase : « Et pépé, regarde, tu vas assister à l'accident du siècle! » (fig. 1.26.)

On ne peut pas faire de la simultanéité avec une plaque tournante.

La solution consiste à diviser l'ordinateur en plusieurs parties, par exemple on relie l'unité centrale à deux mémoires principales.

On peut alors travailler, en même temps sur 2 mémoires pendant que l'on fait des calculs sur les registres de l'unité centrale. Sur notre figure 1.27, les chemins 1, 2 et 3 peuvent se faire en même temps car ils ne se croisent pas.

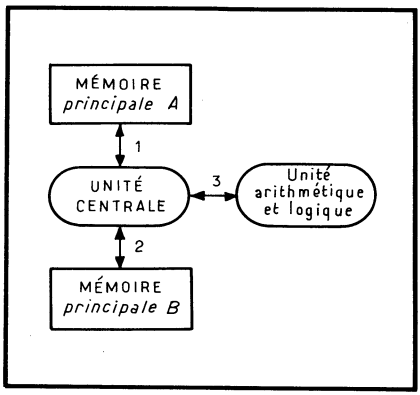


Fig. 1-27. — Suppression de la plaque tournante et mise en place d'un pont.

La simultanéité avec les entrées-sorties

Sur les entrées-sorties le temps (mécanique) s'exprime en millisecondes alors que dans la mémoire principale le temps (électronique) s'exprime en microsecondes, voire en nanosecondes. Nous revenons souvent sur cette notion, elle est capitale. Lorsque nous donnons l'ordre d'exécuter une entrée ou une sortie, si nous n'avons pas prévu de dispositifs spéciaux on va monopoliser la mémoire centrale pendant un temps assez long, pendant tout le temps nécessaire à cette entrée-sortie. La solution consiste à placer dans le canal des registres spécialisés et une mémoire tampon. Lorsqu'on lance une instruction d'entrée-sortie, on place dans les registres spécialisés du canal tous les renseignements nécessaires à l'exécution de cette entrée-sortie. Le canal peut alors commander lui-même cette entrée-sortie en se servant de la mémoire tampon, tout ceci en complète indépendance par rapport au reste de l'ordinateur.

Pendant ce temps la mémoire centrale peut être utilisée pour autre chose. Lorsque la mémoire tampon est remplie (entrée) ou vidée (sortie) il y a arrêt de l'opération en cours dans l'unité centrale (pendant un temps électronique) afin de permettre un déplacement d'information de la mémoire tampon du canal vers la mémoire principale (ou de la mémoire principale vers la mémoire tampon), puis reprise de la simultanéité et ainsi de suite jusqu'à ce que l'instruction d'entrée-sortie soit complètement exécutée.

Interruption

Dans certaines conditions, l'ordinateur a la possibilité d'*interrompre* un programme en cours, de stocker l'état de l'ordinateur et de se placer à

un autre point de la mémoire principale. Nous verrons ce point en détail lorsque nous étudierons le software. (Chapitre VII.2.2.)

Au fur et à mesure que l'on s'achemine dans le temps, la technologie de l'ordinateur devient de plus en plus complexe, les possibilités d'analyse et de programmation deviennent de plus en plus grandes. L'écart entre le hardware et l'utilisateur s'accroît sans cesse, par suite le software chargé de combler cet écart devient de plus en plus nécessaire.

Les ordinateurs spécialisés

Le domaine industriel emploie lui aussi les ordinateurs. N'utilise-t-on pas des ordinateurs pour fabriquer d'autres ordinateurs?

Il s'agit le plus souvent d'ordinateurs construits pour résoudre un problème bien déterminé : commande de machines diverses, contrôle de ces machines, ...

Signalons aussi *l'ordinateur analogique* construit pour résoudre un problème en le simulant. Par exemple, la réalisation de circuits électriques pour résoudre des problèmes mécaniques, acoustiques, etc.

CHAPITRE II

UN EXEMPLE DE TRAITEMENT DE L'INFORMATION EN GESTION

1. — Présentation du problème

Prenons un exemple d'école. Imaginons un service chargé de compléter les factures venant des magasins de vente. Ces factures contiennent, après le numéro du client et le numéro de facture, les différents numéros article avec les quantités vendues.

Il s'agit de placer sur ces factures le nom et l'adresse du client, la désignation et le prix unitaire de chaque article, de calculer les montants puis le montant total de la facture (fig. 2.1).

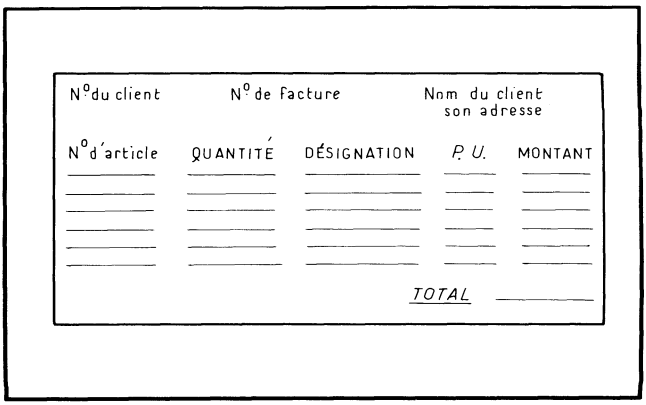


Fig. 2-1. — La facture.

Pour effectuer ce travail, ce service dispose de deux catalogues.

Un catalogue client où les différents numéros de client sont rangés en séquence. En face de chacun de ces numéros on trouve un nom et une adresse avec aussi le numéro de l'agence et le numéro de la sous-agence dont dépend ce client.

Un catalogue article où les différents numéros d'articles sont rangés en séquence. En face de chacun de ces numéros on trouve la désignation et le prix unitaire de l'article.

Le traitement manuel consiste à :

1° lire sur la facture le numéro du client;

2° rechercher dans le catalogue client ce numéro;

3° écrire sur la facture le nom et l'adresse trouvés dans le catalogue;

4° lire le numéro article de la ligne qui suit;

5° rechercher ce numéro dans le catalogue article;

6° écrire sur la facture la désignation et le prix unitaire trouvés dans le catalogue;

7° effectuer le produit quantité par prix unitaire;

8° écrire le résultat dans la colonne montant;

9° on se pose alors la question : « Y a-t-il d'autres numéros article à traiter sur cette facture? Si oui, on revient à l'opération 4 (lire le numéro article de la ligne qui suit). Sinon, on passe à l'opération suivante — (opération 10). On dit que l'on continue en séquence;

10° calculer la somme des montants de chacun des articles;

11° écrire cette somme au bas de la facture;

12° on se pose alors la question : « Y a-t-il d'autres factures à traiter? » Si oui, on prend la facture suivante et on retourne à l'opération 1. Sinon on continue en séquence;

13° fin du travail.

Mais ce n'est pas tout. Dans le traitement de l'information, il faut tout prévoir.

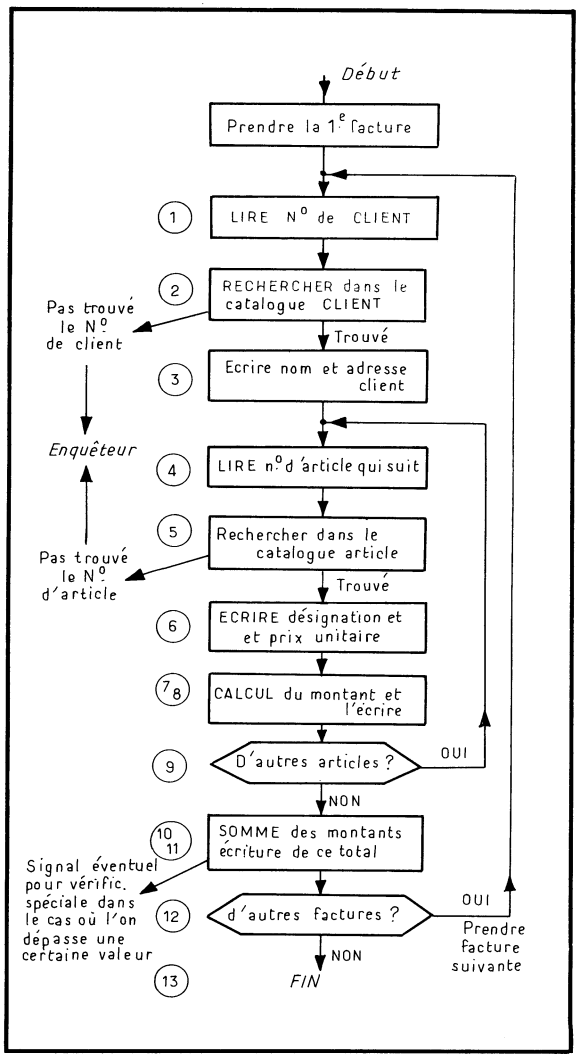
Que se passe-t-il dans le cas où le numéro de client recherché ne se trouve pas dans le catalogue? (Le même problème se pose pour le numéro d'article.) Il est alors nécessaire de faire une enquête. L'erreur peut provenir soit du magasinier, soit de la mise à jour du catalogue.

On peut aussi dans le cas où le total de la facture dépasse une certaine valeur effectuer une vérification spéciale.

Nous pouvons résumer ces différentes opérations à l'aide du diagramme de la figure 2.2. Ce diagramme s'appelle un *organigramme.*

En principe, on peut avoir dans un tel organigramme plusieurs « DÉBUT » et plusieurs « FIN » mais généralement on a, au moins, un « DÉBUT » et une « FIN ».

Fig. 2-2. — Organigramme du travail à automatiser.



Que faut-il remarquer?

— Ce travail est périodique (période : la journée);

— Il s'effectue toujours de la même façon.

C'est par définition un travail que l'on peut donner à une machine.

2. — 1re solution : mécanisation sur matériel classique

Nous disposons de 3 documents :

— le catalogue client;

— le catalogue article;

— les factures.

Il est nécessaire, avant de commencer tout traitement mécanique, de transformer ces données en quelque chose pouvant être lu par la machine. Ce quelque chose est ici la carte perforée.

2.1. — TRANSFORMATION DU CATALOGUE ARTICLE EN FICHIER CARTES

Nous allons transformer chaque ligne du catalogue en une carte. Il y aura donc autant de cartes qu'il y a d'articles. Chaque carte contiendra : un numéro d'article, une désignation et un prix unitaire. Il est nécessaire d'analyser dans quelle colonne de la carte nous allons placer ces différents renseignements.

Le code carte : Nous plaçons les perforations FO1 dans les colonnes 1 à 3 de toutes les cartes. Ces perforations sont nécessaires pour distinguer les différents paquets de cartes les uns des autres. Si, par mégarde, une carte s'égare, il faut pouvoir la replacer.

Le numéro d'article : une étude est nécessaire pour savoir combien de caractères il faut prendre. Sur le catalogue, les différents numéros pouvaient avoir des longueurs différentes. Sur la carte, ils doivent prendre le même nombre de colonnes, O non significatif compris. Pour notre exemple, nous prenons 8 chiffres et perforons le numéro d'article dans les colonnes 4 à 11

La désignation : Le nombre de positions pour cette désignation est très variable sur le catalogue. « vis » ne prend que 3 caractères alors que « tarauds à filet hélicoïdal » en prend beaucoup plus. Il faut prendre sur la carte un nombre de positions suffisant pour que même les plus longues désignations un peu abrégées ou rognées restent compréhensibles.

Pour notre exemple, nous prenons 15 caractères et perforons la désignation dans les colonnes 12 à 26 de la carte.

Le prix unitaire : Aux difficultés précédentes, s'ajoutent les décimales.

Pour notre exemple, nous prenons 4 chiffres à gauche de la virgule et 2 à droite $\times\times\times\times,\times\times$. Nous perforons ce prix unitaire dans les colonnes 27 à 32 de la carte. Il faut remarquer que nous ne perforons pas la virgule. Les machines sauront que l'unité se trouve dans la colonne 30.

A la fin de ces opérations, nous avons déterminé ce que l'on appelle un dessin de carte (fig. 2.3).

Nous avons placé sur la carte ces différents renseignements à la suite des autres mais ce n'est pas obligatoire.

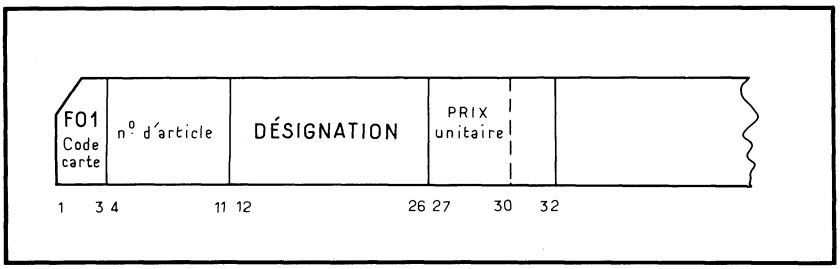


Fig. 2-3. — Dessin de carte pour le fichier article.

L'opération suivante consiste à construire un document pouvant être utilisé par l'atelier de perforation. Ce document c'est le *bordereau de perforation.* C'est une feuille quadrillée où chaque ligne représente une carte, et chaque colonne une colonne de la carte. En trait plein apparaît le dessin de carte (fig. 2.4).

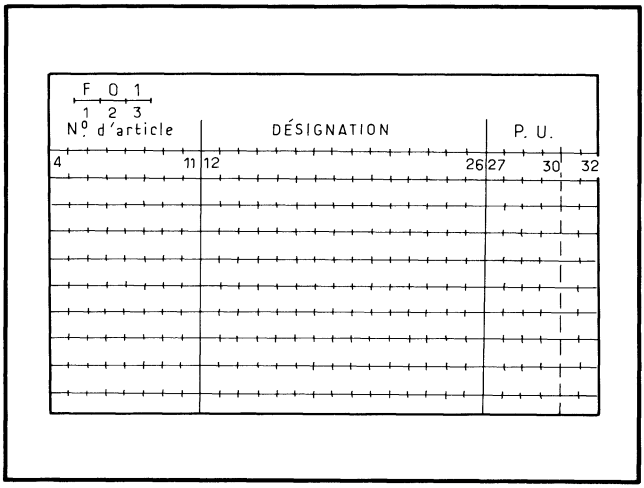


Fig. 2-4. — Bordereau de perforation pour le fichier article.

Nous avons placé le code carte en haut de page car il est à perforer pour toutes les cartes. Il n'est pas nécessaire de le répéter à chaque ligne. Il faut ensuite recopier, ligne par ligne, ou article par article, le catalogue sur les bordereaux de perforation en respectant rigoureusement le quadrillage : un caractère par colonne. Cette opération est nécessaire. Le catalogue, généralement, ne comporte aucune règle mécanographique, ces renseignements ne sont pas rigoureusement cadrés. Il comporte des ratures dues aux mises à jour. Sur le bordereau de perforation, le chiffre unité du prix unitaire doit être sur la colonne 30, le dernier chiffre du numéro d'article sur la colonne 11 (le cadrage à droite est nécessaire si on veut respecter la séquence des numéros d'articles).

Une fois le catalogue recopié, les bordereaux de perforation sont portés à l'atelier de perforation.

Il y a d'abord *perforation des cartes.* Le code carte est généralement perforé seulement sur la première carte. La perforatrice (machine) pouvant ensuite recopier ce numéro sur les cartes suivantes. C'est ce qu'on appelle de la perforation série.

Il y a ensuite *vérification de ces cartes.* La vérificatrice (machine) est une machine identique à la perforatrice à ceci près qu'elle vérifie l'existence d'un trou dans la carte au lieu de perforer ce trou. La vérifieuse (femme) peut ainsi contrôler les erreurs de la perforeuse (femme).

Si donc au départ, on a dans le catalogue 40 000 articles, on a recopié 40 000 lignes sur les bordereaux, perforé 40 000 cartes et vérifié 40 000 cartes. Ce sont des opérations longues, mais elles ne sont à faire qu'une fois.

Après ces opérations, il est nécessaire de placer ces cartes en séquence par numéro d'article. En effet, lorsque nous aurons plusieurs numéros d'articles à rechercher, il suffira de classer ces numéros, puis de lire nos cartes dans l'ordre. Lorsque nous aurons trouvé un numéro d'article, il suffira de lire les cartes suivantes pour trouver le numéro d'article suivant et ainsi de suite. Nos cartes ne seront lues qu'une fois.

La mise de ces cartes en séquence se fait sur une machine que l'on appelle la *trieuse.*

Nous obtenons ainsi un paquet de cartes triées. C'est ce qu'on appelle un *fichier séquentiel* — en séquence sur le numéro d'article. Pour ce qui suit on l'appellera fichier article.

2.2. — *TRANSFORMATION DU CATALOGUE CLIENT EN FICHIER CARTES*

Ce que nous avons fait pour les renseignements articles, nous devons également le faire pour les renseignements clients.

— Détermination du dessin de cartes qui peut être, celui de la figure 2.5.

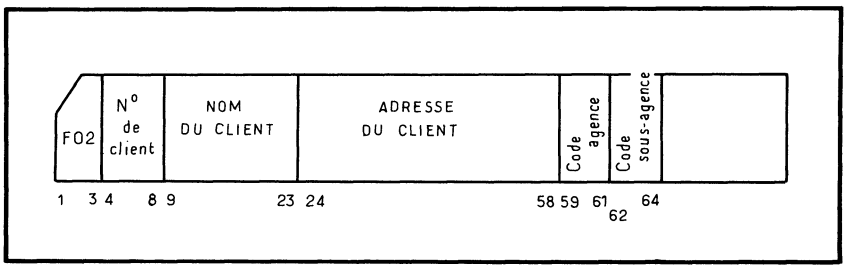


Fig. 2-5. — Dessin de cartes pour le fichier client.

Nous conservons les codes agences et sous-agences bien que ceux-ci ne servent pas dans l'impression des factures, on verra plus loin pourquoi.

— Construction des bordereaux de perforation.

— Recopiage du catalogue sur ces bordereaux.

— Perforation des cartes à partir de ces bordereaux.

— Vérification de ces cartes.

— Tri des cartes par numéro de client.

On obtient ainsi le *fichier client* en séquence par numéro de client.

Nous répétons que ces opérations sont longues mais ce sont des opérations faites une fois pour toutes. Il n'y aura plus ensuite que des mises à jour de fichiers qui consisteront à ajouter ou à retrancher des cartes lorsque nous aurons de nouveaux articles ou clients et des articles périmés.

2.3. — *TRANSFORMATION DES DONNÉES VENANT DU MAGASIN EN CARTES*

Ici aussi on commence par déterminer un dessin de carte, par exemple celui de la figure 2.6.

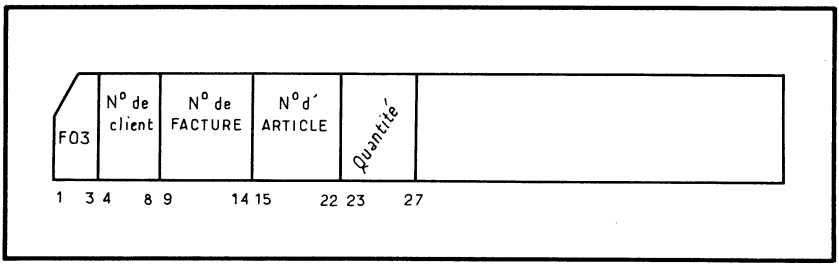


Fig. 2-6. — Dessin de cartes pour les données venant du magasin.

Il nous faut maintenant un bordereau de perforation mais il faut noter une différence importante par rapport à la création des deux fichiers précédents. Les opérations de remplissage des bordereaux, de perforation et de vérification ne sont pas à faire une fois et une fois pour toutes. Elles seront à faire tous les jours.

Il y a donc intérêt dans la mesure où cela est possible (compréhension des magasiniers vis-à-vis du traitement de l'information) à modifier directement les documents fournis par les magasins. Autrement dit les magasiniers écriront directement sur les bordereaux, ceci afin d'éviter un travail de recopiage. Ces bordereaux seront perforés et vérifiés chaque jour par l'atelier de perforation. Nous obtenons ainsi les *cartes données*.

2.4. — *LE TRAITEMENT MÉCANOGRAPHIQUE*

Les différents documents d'entrée sont pris pour être lus par nos machines. Il s'agit d'obtenir comme documents de sortie les factures complètes.

On appelle traitement mécanographique ou chaîne mécanographique tous les passages machines nécessaires pour passer de documents d'entrée *(nœud d'entrée)* à des documents de sortie *(nœud de sortie)*.

Nous allons montrer un exemple de traitement mécanographique, mais sans entrer dans les détails, ces détails pouvant changer avec le matériel dont on dispose. Notre but n'est pas d'expliquer les détails techniques de tel matériel mais d'expliquer à quoi sert le traitement automatique de l'information et finalement comment on peut s'en servir.

Il s'agit ici d'ajouter aux renseignements des cartes données les renseignements correspondants contenus dans les fichiers articles et clients. Pour cela, il sera nécessaire, pour ne lire qu'une fois ces fichiers, de trier les cartes données par client et aussi par numéro d'article. Mais il faut remarquer que les factures, à la sortie, devront être imprimées par numéro de client, de façon que toutes les factures d'un même client sortent ensemble. A l'intérieur de chacun de ces clients il faudra que les factures soient classées par numéro de facture. Ensuite sur une facture on pourra s'arranger pour que les numéros d'articles apparaissent en séquence. On dit dans ce cas que les factures doivent sortir par numéro de client, par numéro de facture, par numéro d'article; pour dire par numéro de client, à l'intérieur de chaque client par numéro de facture, à l'intérieur de chaque numéro de facture par numéro d'article. On dit aussi que :

— le numéro de client est *l'argument majeur;*

— le numéro de facture, *l'argument inter;*

— le numéro d'article, *l'argument mineur.*

On constate qu'il faut sortir par numéro de client. Il est donc préférable de commencer par rechercher les renseignements du fichier article et de finir par les renseignements du fichier client afin d'avoir le minimum de tri.

1re opération : Tri des cartes données par numéro d'article. Cette opération se fait sur une *trieuse.*

2e opération : Interclassement du fichier article avec les cartes données. Il s'agit de lire les cartes données et les cartes du fichier article en même temps et de placer derrière chaque carte du fichier article les cartes données correspondantes. L'opération se fait sur une *interclasseuse* (fig. 2.7).

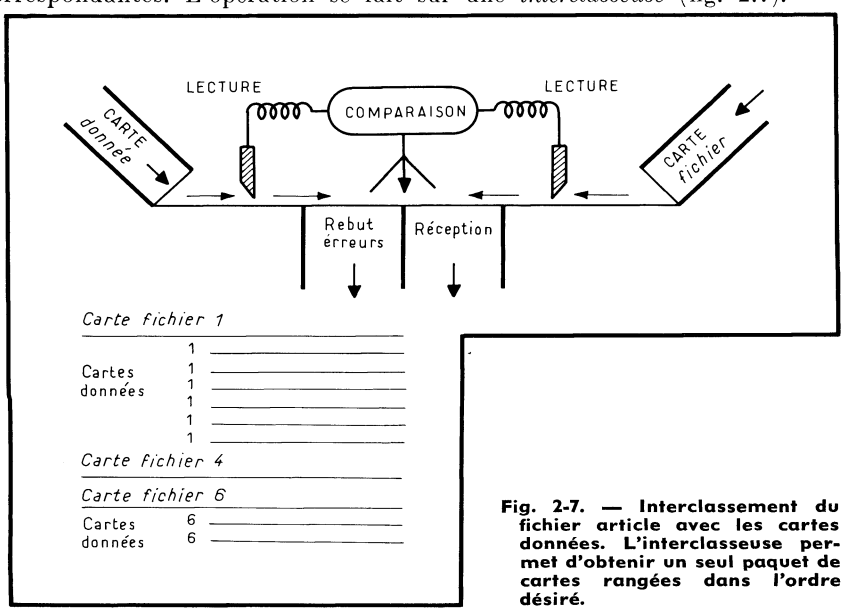


Fig. 2-7. — Interclassement du fichier article avec les cartes données. L'interclasseuse permet d'obtenir un seul paquet de cartes rangées dans l'ordre désiré.

Nous avons à l'entrée deux rampes de lecture. A la sortie les cartes ne forment plus qu'un seul paquet. Dans ce paquet les cartes venant du fichier article s'appellent « *cartes maîtresses* », les cartes venant des données s'appellent « *cartes détails* ». Nous avons une autre case de réception dans laquelle tomberont les quelques cartes données sans correspondance avec le fichier article. Ces cartes, après enquête, vérification et corrections, seront placées à l'entrée du passage suivant (le lendemain).

L'interclasseuse est commandée par un tableau modifiable suivant les différents travaux à faire. Signalons aussi qu'un des principes mécanographiques est de toujours vérifier les opérations précédentes. « La mécanographie a le droit de faire des erreurs mais ces erreurs ne doivent pas sortir de la mécanographie. » Il est possible, en général avec une interclasseuse de vérifier en même temps la séquence des deux paquets de cartes à l'entrée.

3e opération : Mise des renseignements (désignation et prix unitaire) de la carte « maîtresse » dans les cartes « détails » suivantes (s'il y en a), une carte « maîtresse » pouvant très bien ne pas avoir de carte « détails ». Cette opération se fait sur une *reproductrice.*

Cette machine prend dans les cartes de code carte F 01 (cartes du fichier articles) le contenu des colonnes 12 à 32 (désignation et prix unitaire) pour les perforer dans les colonnes 28 à 48 des cartes F 03 (cartes données suivantes) et arrête cette perforation à la rencontre d'une nouvelle carte F 01.

A la fin de cette opération, les cartes données contiennent donc la désignation et le prix unitaire de l'article (fig. 2.8).

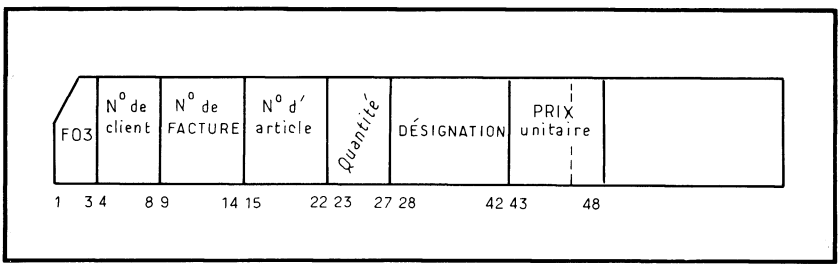


Fig. 2-8. — Contenu des cartes données après la troisième opération.

4e opération : Maintenant que nous avons utilisé le fichier articles, nous n'en avons plus besoin. Il suffit de faire sur trieuse un tri sur la colonne 3 (on voit ainsi une des utilités du code carte) pour séparer les cartes maîtresses des cartes détails. Après cette opération, le fichier article est reconstitué, tel qu'il était au départ, ce qui est nécessaire pour le travail du lendemain.

5e opération : Il faut maintenant aller chercher les renseignements du client. Pour cela, un tri par client va être nécessaire. Mais n'oublions pas que nous voulons à l'intérieur de chaque client les factures en séquence par numéro de facture. C'est pourquoi nous effectuons ici le tri par client — par facture. Nous verrons plus loin, en expliquant le fonctionnement des différents tris, que le tri sur matériel classique conserve le tri précédent. C'est-à-dire que comme nos cartes sont déjà triées par numéro d'article, à l'intérieur de chaque facture elles seront toujours triées par numéro d'article (fig. 2.9).

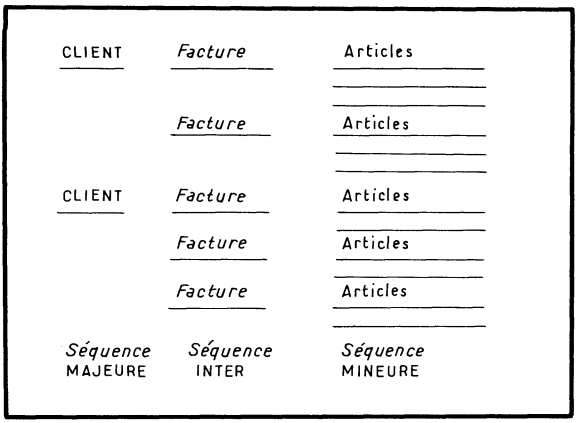


Fig. 2-9. — Les cartes doivent être triées par client, par facture, par article.

6e *opération :* Il va falloir ajouter aux cartes données : le nom du client : 15 positions, l'adresse : 25 positions, le code agence : 3 positions, le code sous-agence : 3 positions. Or nous avons déjà dans la carte 48 colonnes de prises; Tout ne tient pas dans une carte qui en principe fait 80 colonnes. C'est pourquoi, nous allons reproduire partiellement les cartes données F 03 pour obtenir les cartes F 04. Ces dernières ont le dessin de carte de la figure 2.10.

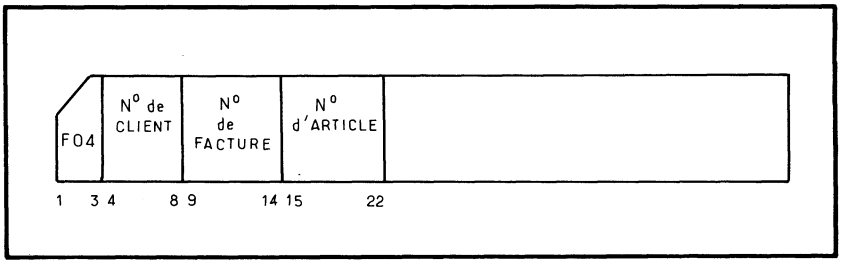


Fig. 2-10. — Dessin de carte des cartes données complémentaires FO 4.

Ce travail se fait également sur une reproductrice, à l'aide de 2 chemins de carte entièrement indépendants. Dans l'un, chemin de lecture, on place les cartes F 03, dans l'autre, chemin de perforation, on place les cartes vierges. A la sortie du premier chemin, on récupère les cartes F 03, à la sortie du deuxième on a les cartes F 04.

7e *opération :* Interclassement des cartes F 04 avec le fichier client F 02 comme nous l'avons fait avec le fichier articles.

Les cartes sans correspondance avec le fichier client tombent dans une case à part, elles sont remises au bureau de vérification, mais il ne faut pas oublier de retirer les cartes F 03 correspondantes. Le bureau de vérification après enquête et correction remettra les cartes corrigées dans le traitement mécanographique du lendemain.

8e opération : Reproduction du nom, de l'adresse, des codes agences et sous-agences dans les cartes détail F 04.

A la fin de ce passage, les cartes F 04 ont le contenu indiqué dans la figure 2.11.

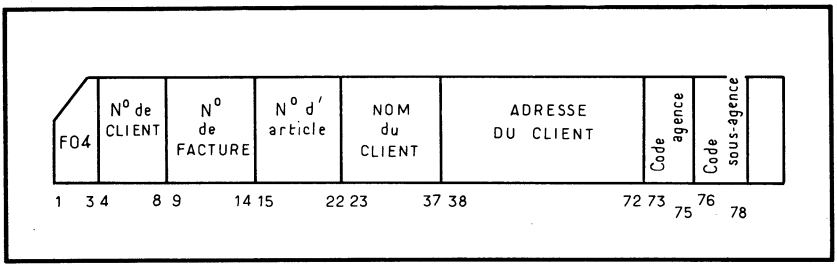


Fig. 2-11. — Contenu des cartes FO 4 après la huitième opération.

9e opération : Un tri sur la colonne 3 (code carte) pour séparer les cartes F 04 des cartes F 02.

A la fin de ces opérations, le fichier client est reconstitué, tel qu'il était au début du travail, ce qui est nécessaire pour le travail du lendemain.

10e opération : Il nous manque encore le montant. On passe les cartes F 03 dans une calculatrice qui exécute le produit : quantité par prix unitaire, et perfore le résultat dans les colonnes 49 à 55 (fig. 2.12).

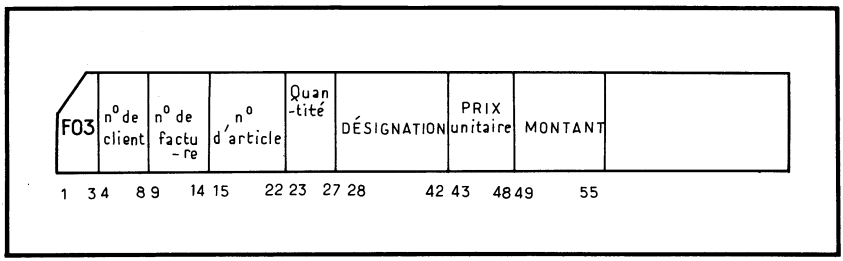


Fig. 2-12. — Contenu des cartes FO 3 après la dixième opération.

11e opération : Interclassement des cartes F 04 et F 03. A chaque carte F 03 correspond une carte F 04.

12e opération : Nous avons maintenant tout ce qu'il faut pour imprimer les factures (ouf enfin !). Nous sommes dans la bonne séquence par numéro de client, par facture, par article. Un passage sur une machine appelée tabulatrice nous imprime les factures (selon un dessin que nous devons lui aussi analyser). Cette tabulatrice possède des compteurs relevant à chaque fois les montants. Elle peut à la fin de chaque facture imprimer le total. La figure 2.13 reprend ces douzes opérations et présente ainsi l'organigramme de la *solution* sur matériel classique.

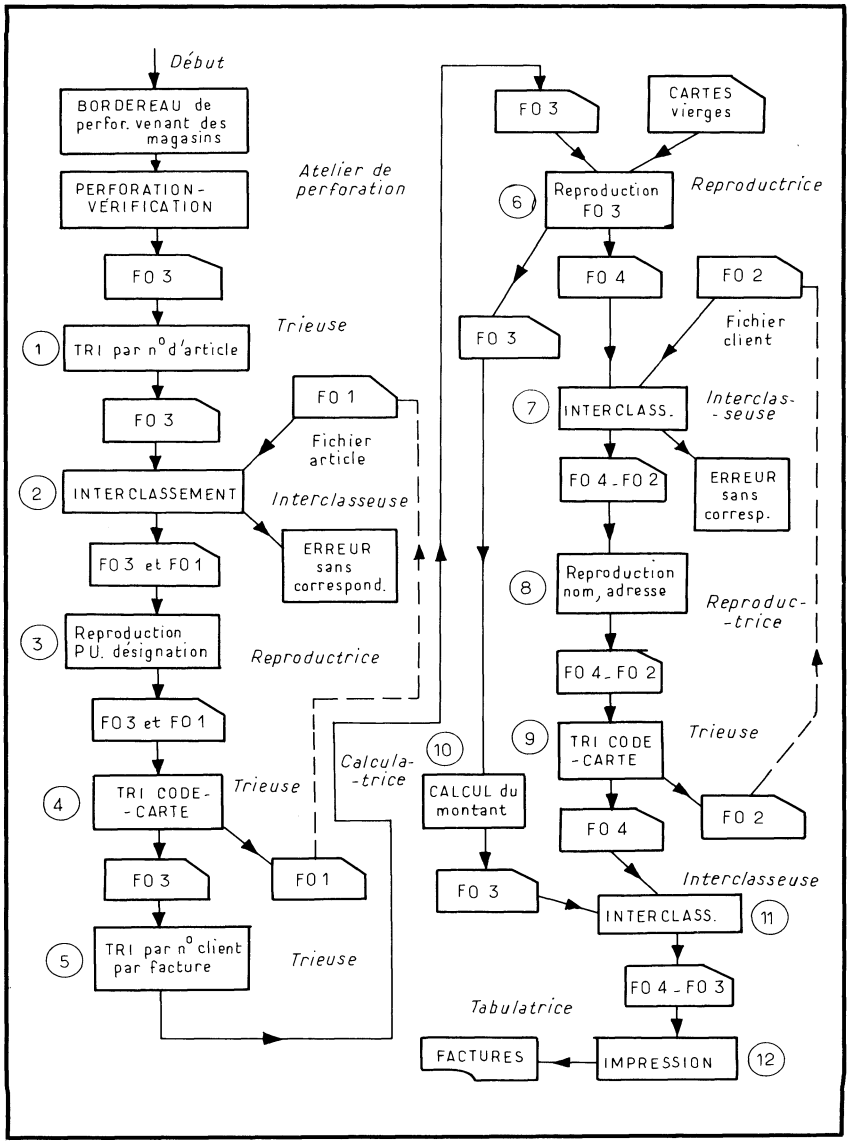


Fig. 2-13. — Organigramme d'une solution sur matériel classique.

2.5. — *AMÉLIORATIONS A L'ORGANIGRAMME PRÉCÉDENT*

Dans le domaine du traitement de l'information, on trouve toujours une solution qui marche, mais, plus que dans n'importe quel autre domaine, on peut aussi toujours améliorer cette solution, soit en la modifiant soit en y trouvant une autre méthode. Le tout est d'arriver à un point où les modifications possibles apportent un avantage infime par rapport au coût de ces modifications. Arrivé à ce point, on peut s'estimer satisfait jusqu'au jour où un changement de structure, de contexte ou une amélioration de technique viennent à nouveau tout modifier.

Peut-on améliorer l'organigramme précédent? En fermant les yeux, on peut répondre *oui*.

— D'abord en modifiant les dessins de cartes, il est possible de simplifier le travail des reproductrices. Par exemple, dans les cartes F 03 et F 04, le numéro d'article figure dans les colonnes 15 à 22. N'est-il pas préférable de placer ce numéro d'article dans les mêmes colonnes pour les cartes F 01?

— Il est possible de travailler avec le fichier article d'une autre manière.

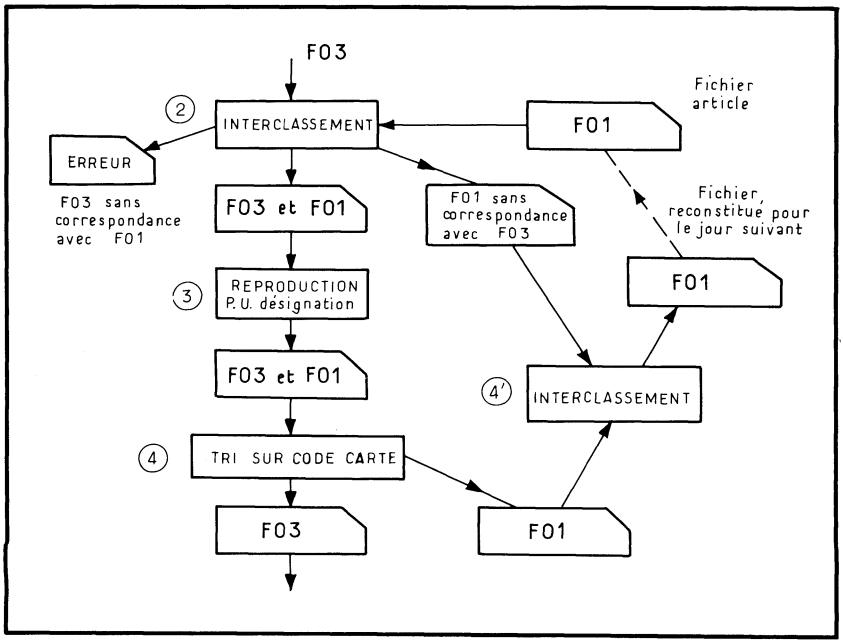


Fig. 2-14. — Modification de l'organigramme de la solution sur matériel classique.

Au cours de l'interclassement (2e opération) faire tomber dans une case de réception spéciale les cartes maîtresses sans cartes détails. Cela exige (après la 4e opération) un passage en interclasseuse supplémentaire pour remettre le fichier article F 01 en place. Mais nous serons probablement

gagnants au point de vue temps dans le cas où les cartes données sont peu nombreuses par rapport à la dimension du fichier articles — la reproductrice (3e opération) étant une machine lente par rapport à l'interclasseuse. Enfin cette question est à étudier soigneusement par l'analyste (fig. 2.14).

— La même question est à poser avec le fichier client.

— Une carte F 04 seulement est nécessaire par client, donc on peut supprimer carrément ce type de cartes et les remplacer par les cartes du fichier client F 02, les cartes F 04 sont parfaitement inutiles! etc.

2.6. — LES NOUVEAUX BESOINS

A chaque fois que l'homme a mis en place un automatisme nouveau, il a toujours pensé : « Que de monde cela va me supprimer. » En fait, à chaque fois, il a découvert qu'il pouvait aussi avec sa machine faire autre chose. Il s'est trouvé des *besoins nouveaux.* On peut dire que parmi tous les automatismes, le traitement (électromécanique ou électronique) de l'information est celui qui apporte le plus grand nombre de besoins nouveaux à l'homme.

Ici que constate-t-on? A la fin de chaque journée, nous avons des cartes données F 03 et F 04. Nous pouvons conserver ces cartes et, (à la fin du mois ou de l'année), obtenir des statistiques mensuelles ou annuelles.

Les statistiques sur le numéro article

Nous pouvons par exemple calculer pour chaque numéro d'article la quantité vendue dans le mois et le montant qui en résulte.
et aussi imprimer un total final donnant le chiffre d'affaires du mois (fig. 2.15).

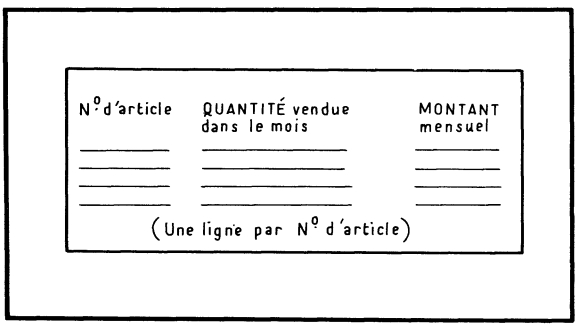


Fig. 2-15. — Statistique mensuelle par article.

Pour cela il suffit de prendre les différentes cartes F 03 du mois et de les trier par numéro d'article.

Un passage en tabulatrice avec une ligne d'impression à la dernière carte de chacun des numéros d'article donne la statistique demandée.

Mais nous pouvons encore faire mieux. Nous pouvons connecter la tabulatrice à une perforatrice, cette perforatrice perforant une carte à chaque fois qu'elle imprime une ligne au cours de la statistique précédente. On obtient ainsi les cartes F 05 dont le dessin de carte est celui de la figure 2.16.

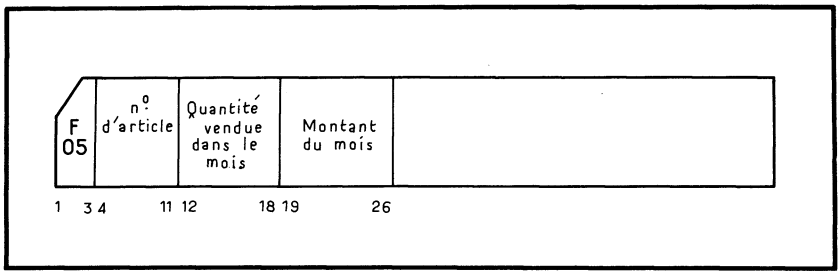


Fig. 2-16. — Dessin de carte des cartes récapitulatives FO 5. Une carte FO 5 par article.

Nous pouvons trier ces cartes par montant et sortir la statistique par montant d'article (en séquence par montant) (fig. 2.17).

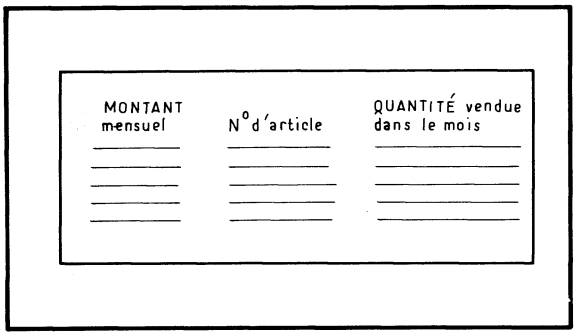


Fig. 2-17. — Statistique mensuelle par montant d'article.

Mais qu'elle est l'utilité d'une telle statistique?

Dans la statistique précédente, nous pouvons nous amuser à numéroter les lignes et aussi à imprimer à chaque ligne un montant progressif (montant progressif de la 1re ligne égal au 1er montant article, montant progressif de la 2e ligne égal à la somme des deux premiers montants article, montant progressif de la n^e ligne égal à la somme des n premiers montants). On peut alors tracer une courbe : montant progressif fonction du numéro de la ligne (ou des n articles ayant les montants les moins importants). Cette courbe a dans la plupart des cas l'allure de celle de la figure 2.18.

Une modification sur les articles de la région A n'influe que très légèrement sur le chiffre d'affaires global. Il n'en est pas de même pour les articles de la région C. Une légère modification sur la quantité vendue pour ces articles entraîne une modification importante sur le résultat global. Tous les efforts de l'entreprise doivent porter sur les articles C.

Les statistiques sur le numéro de client

Nous pouvons également sortir une statistique (fig. 2.19) par agence, par sous-agence, par numéro de client (voilà l'utilité des codes agences et sous-agences).

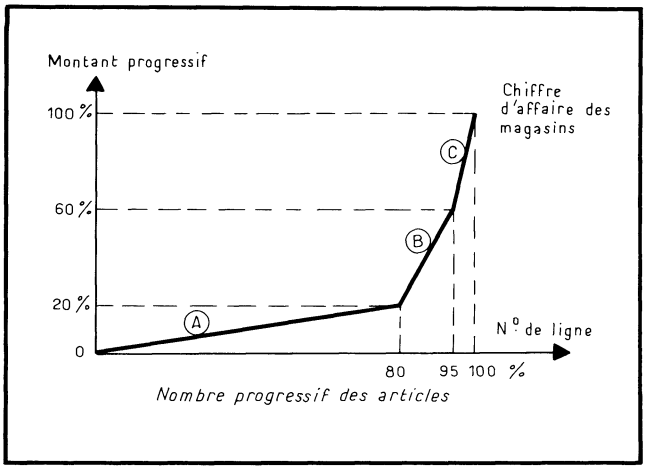


Fig. 2-18. — Courbe du montant progressif en fonction du nombre d'articles rangés par montant croissant.

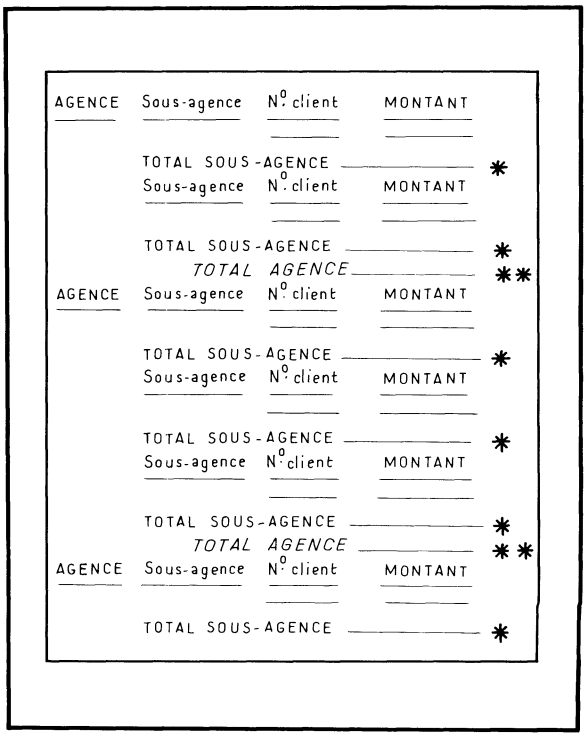


Fig. 2-19. — Statistique mensuelle par agence, par sous agence, par numéro de client.

(sans oublier d'imprimer un total final toujours nécessaire pour être sûr de ne pas avoir oublié une carte).

Pour obtenir cette statistique, il suffit de perforer une carte, pour chaque client, au moment de l'impression des factures (12e opération). Chacune de ces cartes F 06 contient le numéro de client, ses codes agence et sous-agence, et le total des montants des factures pour ce client. A la fin du mois on fusionne (interclassement) entre eux les différents paquets F 06 et on les trie par numéro agence, par numéro de sous-agence (puisqu'ils sont déjà triés par numéro de client). Cette statistique nous permet de connaître l'importance de chaque agence, à chaque agence de connaître le débit de leur sous-agence, enfin à ces dernières de connaître leurs bons clients. (Une agence pouvant être un pays, une sous-agence : une région de ce pays.)

Certes ces manipulations cartes sont lourdes (le nombre de cartes F 03 et F 06 de l'ensemble du mois pouvant être important). Nous allons voir qu'avec les techniques bandes et plus encore disques, ces manipulations sont plus simples.

3. — 2e solution avec un ordinateur à bandes magnétiques

3.1. — MISE DES FICHIERS ET DES DONNÉES D'ENTRÉE SUR BANDE MAGNÉTIQUE

Puisque nous adoptons la solution bande magnétique, il est logique de placer les fichiers et le « nœud d'entrée » sur bandes magnétiques (ou rubans magnétiques).

Pour aboutir à ce résultat on commence comme pour le matériel classique :

— étude de dessin de carte;

— bordereau de perforation;

— perforation des cartes;

— vérification des cartes.

Remarque : Dans l'état actuel de la technique il est également possible d'obtenir une machine à clavier enregistrant directement les informations sur bande magnétique, sans passer par un support intermédiaire (cartes perforées ou ruban perforé), mais cela ne change pas la logique des choses. Le bordereau de perforation, la frappe sur le clavier et la vérification restent nécessaire. (En principe on peut supprimer la vérification lorsque la prise d'information est « directe ». C'est-à-dire lorsque c'est celui qui détient les informations qui les rentre.) Il nous faut maintenant voir comment nous allons placer ces renseignements sur bande (dessin de bande).

Plusieurs remarques :

a) Nous ne sommes pas limités sur une bande aux 80 colonnes de la carte. Nous devons décomposer la bande en *enregistrement physique* (ou bloc de caractères). En effet, pour lire ou écrire sur une bande magnétique, il est nécessaire que cette bande défile à une certaine vitesse devant les têtes de lecture-écriture du dérouleur (lecteur de bande). Pour atteindre cette vitesse il faut un certain temps d'accélération. De même pour s'arrêter il

faut aussi un certain temps de décélération. Ces accélérations et décélérations représentent sur la bande un certain espace entre chaque enregistrement physique. On appelle cet espace : *entre-enregistrement.* Afin de perdre moins de temps, on a intérêt à avoir le moins d'enregistrement possible. C'est pourquoi on choisit, comme dans la figure 2.20, d'avoir des enregistrements physiques très longs en groupant dans un même enregistrement plusieurs renseignements venant de plusieurs cartes (on est limité par la capacité mémoire de l'ordinateur).

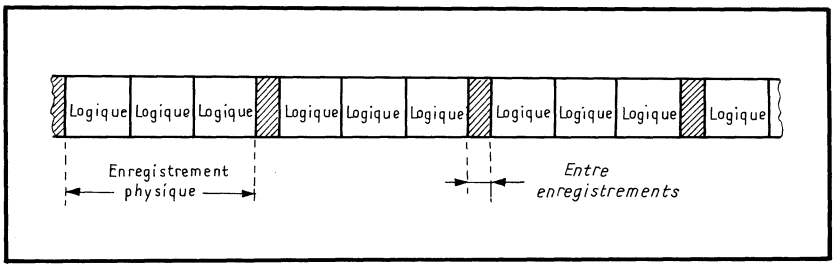


Fig. 2-20. — Un enregistrement physique contient plusieurs enregistrements logiques.

On dit qu'à chaque carte correspond un enregistrement logique (ou fiche). Le nombre d'enregistrements logiques par enregistrement physique s'appelle facteur de groupage.

b) Lorsque nous prenons une bande magnétique, nous ne savons pas ce qu'elle contient. Bien sûr, elle porte sur une de ses faces un papillon collé par un opérateur, mais ce n'est pas en mécanographie une précaution suffisante. Nous verrons au chapitre « Les différentes organisations de fichier » toutes les précautions qu'il faut prendre.

Revenons à notre exemple. Nous pouvons prendre comme dessin de bande pour le *fichier article :* 29 caractères, 8 pour le numéro d'article, 15 pour la désignation et 6 pour le prix unitaire, (comme dans la figure 2.21),

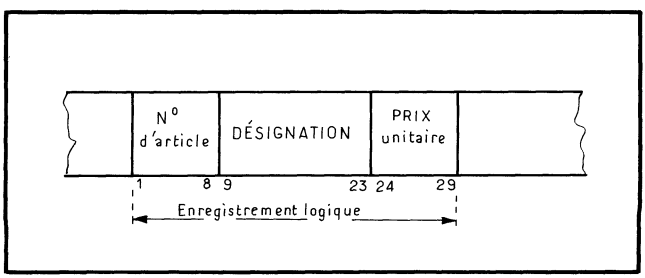


Fig. 2-21. — Dessin de bande pour le fichier article. La longueur de l'enregistrement logique est de 29 caractères.

et choisir un facteur de groupage de 50 (50 enregistrements logiques par enregistrement physique).

Longueur d'un enregistrement physique : $50 \times 29 = 1450$ caractères.

Pour le *fichier client* nous pouvons prendre 61 caractères, (comme dans la figure 2.22), avec un facteur de groupage de 30.

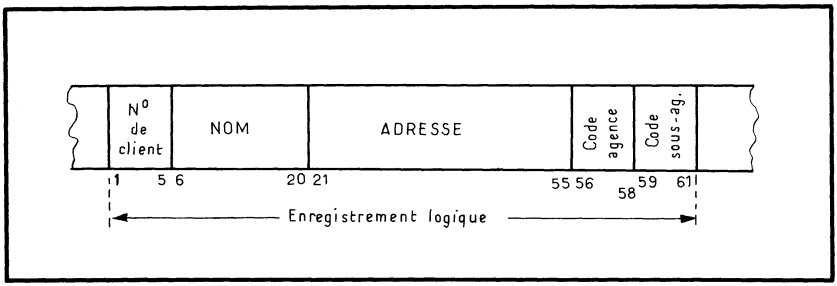
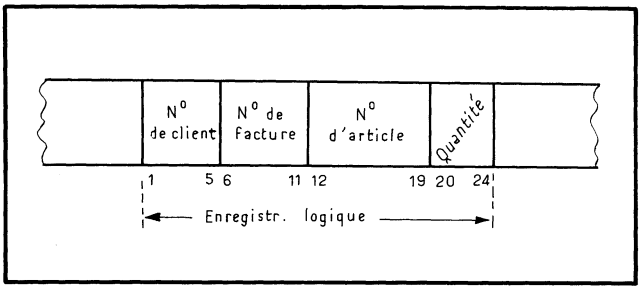


Fig. 2-22. — Dessin de bande pour le fichier client.

Longueur du bloc physique $30 \times 61 = 1830$ caractères.

Pour les données 24 caractères, (comme dans la figure 2.23), avec un facteur de groupage de 50.

Fig. 2-23 — Dessin de bande pour les données venant du magasin.



Longueur du bloc physique $50 \times 24 = 1200$ caractères.

Après ce choix nous mettons les cartes F 01 et F 02 sur bande à l'aide d'un programme que l'on peut appeler « cartes à bande ». Il n'est pas nécessaire de trier les cartes au préalable. Nous verrons au chapitre « Les tris » *que le tri* sur bande est bien plus rentable que le tri sur cartes.

Après ces deux cartes à bandes nous ferons donc deux tris bandes. Un tri par numéro d'article pour le fichier article, un tri par numéro de client pour le fichier client. Nous obtenons ainsi les deux fichiers sur bandes magnétiques, ils vont nous servir pour le traitement journalier des factures. Ces fichiers sont ici aussi créés une fois pour toutes. Il suffira d'avoir des programmes de mise à jour pour les modifications d'article et de client.

3.2. — LE TRAITEMENT JOURNALIER SUR ORDINATEUR

Les cartes données arrivent, comme pour la solution avec matériel classique, chaque jour, avec le code carte F 03, de l'atelier de perforation.

1re opération : mise de ces cartes sur bande avec un programme de « cartes à bande » — les données sont placées en respectant le dessin de bande que nous avons précédemment défini.

2e opération : tri de cette bande par numéro d'article, pour la même raison que pour la solution « matériel classique » (la sortie doit se faire par numéro de client). On commence par trier notre bande « données » par numéro d'article à l'aide d'un programme de tri sur bande.

3e opération : recherche des renseignements article sur le fichier article. On écrit un programme lisant en entrée la bande « donnée » et la bande fichier article (comme le fait l'interclasseuse). Lorsque ce programme trouve dans le fichier article le numéro article de la bande « donnée » il écrit sur une bande sortie un enregistrement (logique) contenant : le numéro de client, le numéro de facture, le numéro d'article, la quantité, la désignation et le prix unitaire. Là aussi l'analyste doit choisir un dessin de bande.

Lorsqu'un numéro d'article n'a pas de correspondance avec le fichier article, on imprime sur l'imprimante de l'ordinateur la totalité de l'enregistrement logique « donnée » correspondant.

4e opération : tri par numéro de client, par facture, par numéro d'article, avant d'aller chercher les renseignements du fichier client. (A noter qu'en tri sur bande le tri précédent est détruit. Il est nécessaire de spécifier au programme de tri la séquence mineure.)

5e opération : recherche des renseignements client sur le fichier client et calcul du montant. On a un programme lisant en entrée la bande « donnée » que l'on vient de trier et la bande fichier client. Lorsque le programme trouve dans le fichier client le numéro client de la bande, il fait le produit du prix unitaire par la quantité, et il écrit sur une bande de sortie un enregistrement (logique) contenant : le numéro client, le numéro de facture, le numéro d'article, la quantité, la désignation, le prix unitaire, le montant, le nom et l'adresse du client, ses codes agences et sous-agences.

6e opération : impression des factures à partir de la bande précédente.

— La figure 2.24 reprend ces 6 opérations et contient ainsi l'organigramme de la solution sur bande magnétique.

3.3. — ORDINATEUR PÉRIPHÉRIQUE ET ORDINATEUR DE TRAITEMENT

On aurait pu au cours du traitement précédent effectuer la 6e opération en même temps que la 5e mais généralement on divise en gestion les entrées-sorties en trois catégories :

Les unités lentes : lecteur et perforateur de cartes, de rubans perforés, imprimantes.

Les unités rapides : les bandes et les disques.

On classe dans une 3e catégorie *les unités de traitement en temps réel.*

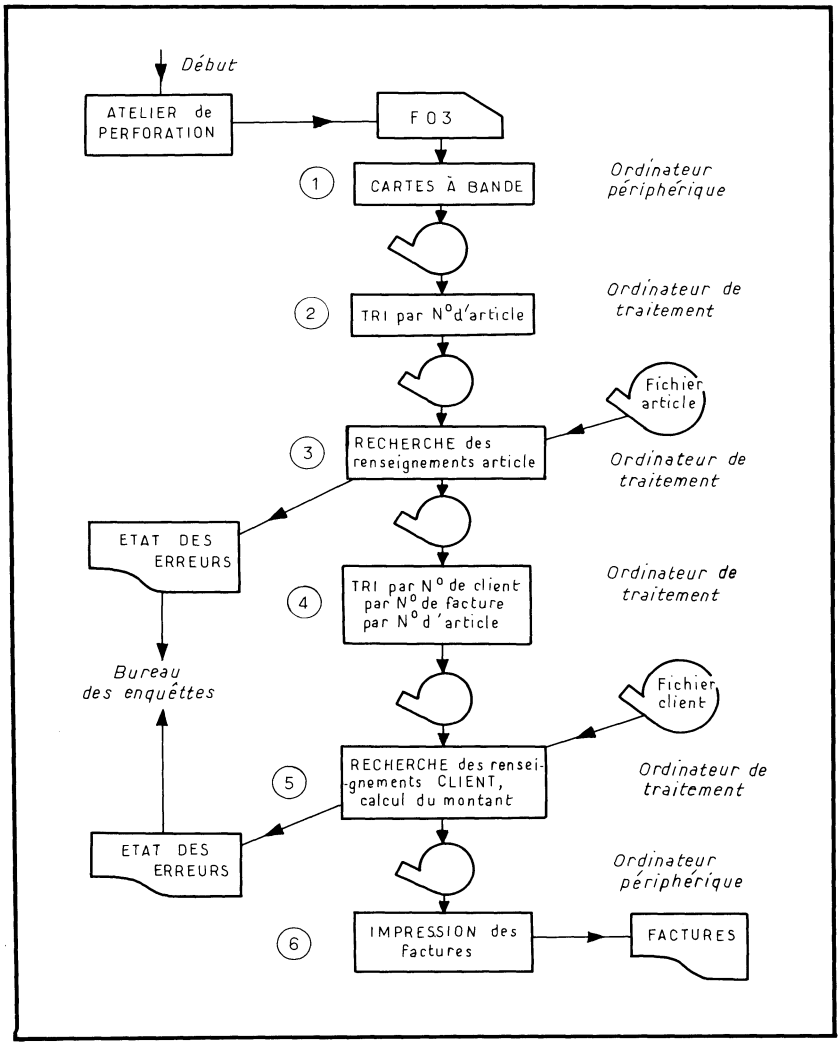


Fig. 2-24. — Organigramme d'une solution sur ordinateur à bandes magnétiques.

C'est pourquoi on peut rencontrer dans les ateliers de traitement de l'information deux types d'ordinateurs.

Des ordinateurs de traitement : ordinateur puissant, et par conséquent coûteux, orienté unités rapides.

Des ordinateurs périphériques : moins coûteux, spécialisés dans les traitements de cartes à bande, bande à imprimante. Ces opérations lentes n'ont pas besoin de monopoliser un ordinateur puissant.

Dans le traitement précédent, les opérations 1 et 6 se font sur un ordinateur périphérique, les opérations 2 à 5 sur un ordinateur de traitement.

Remarque : Faut-il construire des ordinateurs spécialisés ou des ordinateurs capables de tout faire? La 3e génération a carrément choisi la 2e solution, mais corrige peu à peu le tir. Citons par exemple des unités spécialisées bande magnétique à imprimante.

3.4. — LES STATISTIQUES

Avec la technique bande on peut obtenir les mêmes statistiques qu'avec le matériel classique mais les manipulations sont moins lourdes.

Les statistiques article

Nous pouvons modifier le programme de la 3e opération « recherche des renseignements article » de la manière suivante : Pendant le 1er traitement du mois (1er jour du mois) sortir en plus une bande récapitulative comportant un enregistrement logique pour chacun des articles traités avec en face du numéro d'article le montant total de la journée pour cet article.

Il est à remarquer qu'il est alors préférable dans l'organigramme précédent de calculer le montant dans la 3e opération plutôt que dans la 5e opération (amélioration de l'organigramme).

La 3e opération le 1er jour du mois devient celle de la figure 2.25.

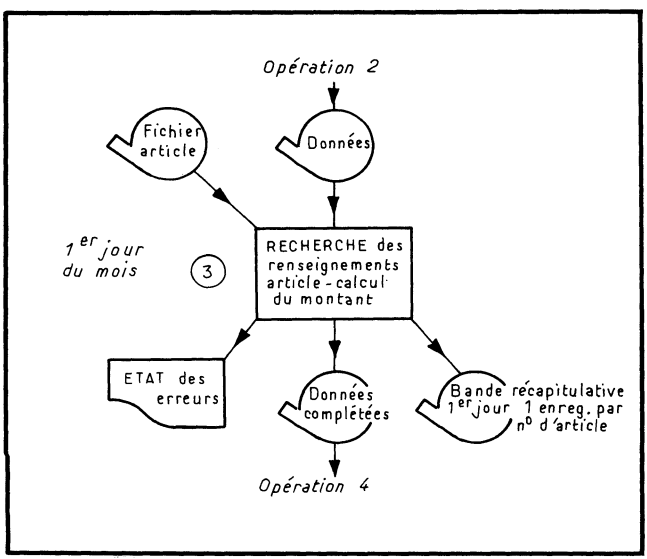


Fig. 2-25 — Le premier jour du mois la troisième opération crée une bande récapitulative des montants par article.

Dans le traitement du 2e jour du mois, le programme lit la bande récapitulative du 1er jour et sort une bande récapitulative des deux premiers jours contenant pour chacun des articles la somme des montants des deux

premiers jours (montant de la bande récapitulative placée en entrée plus la somme des montants de l'article pendant la journée).

Et ainsi de suite au cours de la n^e journée du mois (fig. 2.26) on recueillera la bande récapitulative de n premiers jours et à la fin du mois on aura directement de cette manière (bande tournante) la statistique par article donnant

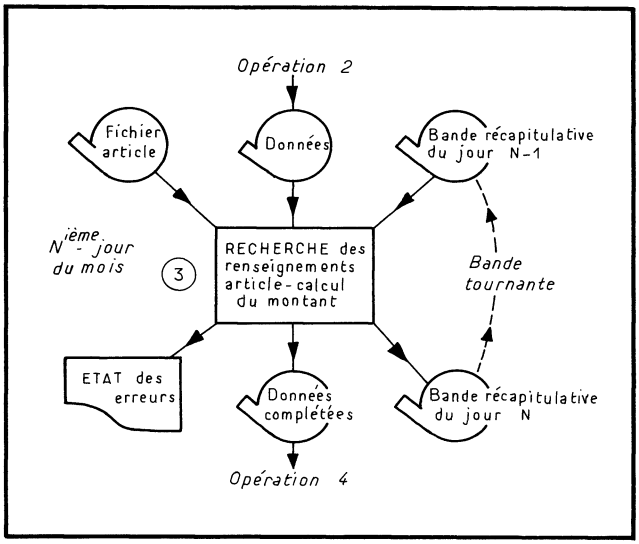


Fig. 2-26 — Le N^{ième} jour du mois la troisième opération crée la bande récapitulative des *n* premiers jours du mois.

le montant total du mois pour chacun des articles. Pour obtenir la statistique article par montant croissant on trie la bande récapitulative du dernier jour du mois par montant.

La statistique client

Pour obtenir la statistique par agence, par sous-agence, par numéro de client, on adopte la même méthode (une bande tournante) avec le programme de la 5^e opération. A la fin du mois, on prend la bande récapitulative du dernier jour et on la trie par agence, par sous-agence, par numéro d'article.

4. — 3^e solution : avec un ordinateur à mémoires à accès sélectif

Avec cette solution, l'organigramme va être encore plus simple : cela ne veut pas dire que l'ensemble de l'analyse l'est. En fait les mémoires à accès sélectif permettent d'avoir accès directement à un enregistrement logique donné. Encore faut-il que l'analyste ait défini une organisation, une méthode, une formule, pour que, à partir du numéro de client ou du numéro d'article,

le programme puisse chercher ou calculer l'endroit géographique où se trouve l'enregistrement correspondant. Il y a là une étude assez longue que nous verrons au cours du chapitre : « Les différentes organisations de fichier ».

4.1. — MISE DES FICHIERS SUR MÉMOIRES A ACCÈS SÉLECTIF

L'analyste a donc commencé par étudier et définir les méthodes nous permettant :

— à partir d'un numéro d'article donné, de trouver les renseignements correspondants de ce numéro à un endroit bien déterminé de la mémoire à accès sélectif;

— à partir d'un numéro de client donné, de trouver le nom et l'adresse correspondant à ce numéro à un endroit bien déterminé.

Les opérations initiales seront les mêmes qu'avec les deux premières solutions. C'est-à-dire : étude des deux dessins de cartes F 01 et F 02, bordereau de perforation, perforation et vérification à l'atelier de perforation. On obtient ainsi les deux fichiers sur carte.

Sur un ordinateur périphérique on met successivement ces deux fichiers sur bande.

Ces bandes sont données à l'ordinateur de traitement afin de mettre ces fichiers sur mémoires à accès sélectif. On a un programme qui lit l'enregistrement logique à l'entrée, à partir du numéro de client ou du numéro d'article lu, détermine (à l'aide de la méthode analysée) l'endroit géographique sur la mémoire à accès sélectif où il faut placer l'enregistrement, positionne la tête de lecture-écriture de la mémoire à accès sélectif sur cet endroit géographique et y écrit l'enregistrement.

Parfois, avant de mettre nos fichiers sur mémoire à accès sélectif, il est nécessaire de *préformater* cette mémoire. En effet, les enregistrements ne sont pas obligatoirement placés en séquence et il est nécessaire de bien diviser la mémoire en case afin qu'un enregistrement ne vienne pas s'intercaler sur un autre (c'est le problème du parking de voiture où il est nécessaire que les premières voitures soient bien garées).

4.2. — TRAITEMENT

Avec la technique « mémoire à accès sélectif », il n'est plus nécessaire de trier les enregistrements donnés avant d'aller chercher les renseignements dans les fichiers correspondants. Le seul tri nécessaire est le tri pour la sortie, les factures doivent être imprimées par numéro de client, par numéro de facture, par article.

Le traitement ne comprend donc plus que 4 opérations.

1re opération : (Comme pour la solution sur bande magnétique) mise des données F 03 venant de l'atelier de perforation sur bande magnétique. Ceci se fait sur un ordinateur périphérique.

2e opération : Tri des données par numéro de client, par numéro de facture, par numéro d'article sur un ordinateur de traitement (ce tri peut se faire sur bandes ou sur mémoires à accès sélectif).

3e opération : Le traitement proprement dit. On écrit un programme ayant en entrée la bande « donnée » que l'on vient de trier, et les deux fichiers article et client. Lorsque ce programme a lu « une donnée », il calcule, à l'aide du numéro de client, par la méthode analysée, l'endroit géographique où se trouve l'enregistrement correspondant sur le fichier client. Il lit cet enregistrement client, et cet enregistrement reste en mémoire principale de l'ordinateur, tant que l'on lit les données ayant le même numéro de client. On va chercher un autre enregistrement client lorsque l'on change, à l'entrée, de client.

Il faut noter qu'avec ce traitement on peut avoir le fichier client organisé en séquence sur le numéro de client (comme sur une bande magnétique). Cela est dû au fait que, dans notre cas particulier, les enregistrements « donnée » sont triés par numéro de client.

Lorsque notre programme lit « une donnée » il calcule aussi à l'aide du numéro d'article l'endroit géographique sur la mémoire à accès sélectif où se trouve l'enregistrement article correspondant. Il lit cet article, et il calcule le montant. Il écrit directement sur une bande magnétique tous les renseignements nécessaires pour l'impression des factures : numéro, nom et adresse du client : numéro, quantité, désignation, prix unitaire et montant du numéro d'article. Il est même possible « d'éditer sur la bande magnétique » (les données sont placées sur la bande magnétique avec le même dessin que sur l'état de l'imprimante, c'est-à-dire des espaces entre chaque renseignement, des points et des virgules pour les chiffres...).

4e opération : Impression des factures sur un ordinateur périphérique à partir de la bande précédente. L'organigramme se résume donc à celui de la figure 2.27.

4.3. — LES STATISTIQUES

Elles sont avec cette solution très facile à obtenir. Il suffit dans les enregistrements des fichiers clients et articles d'ajouter un compteur. Par exemple, les enregistrements logiques des fichiers client ont le dessin suivant : un *numéro* de client, un nom, une adresse, un code agence, un code sous-agence et un compteur.

Au début du mois, ce compteur est mis à zéro. Dans le programme de la 3e opération de l'organigramme précédent on lit le contenu de ce compteur. Tous les montants calculés pour ce client sont ajoutés à ce contenu. Ensuite avant de changer de client (ou à la fin du travail) on *réécrit* sur le fichier client l'enregistrement client avec le compteur *mis à jour* par ces nouveaux montants.

A la fin du mois, il suffit de prendre le contenu des compteurs pour (après un tri par agence, par sous-agence) avoir la statistique client.

On procède de la même manière avec le fichier article pour avoir les statistiques par article puis par montant d'article.

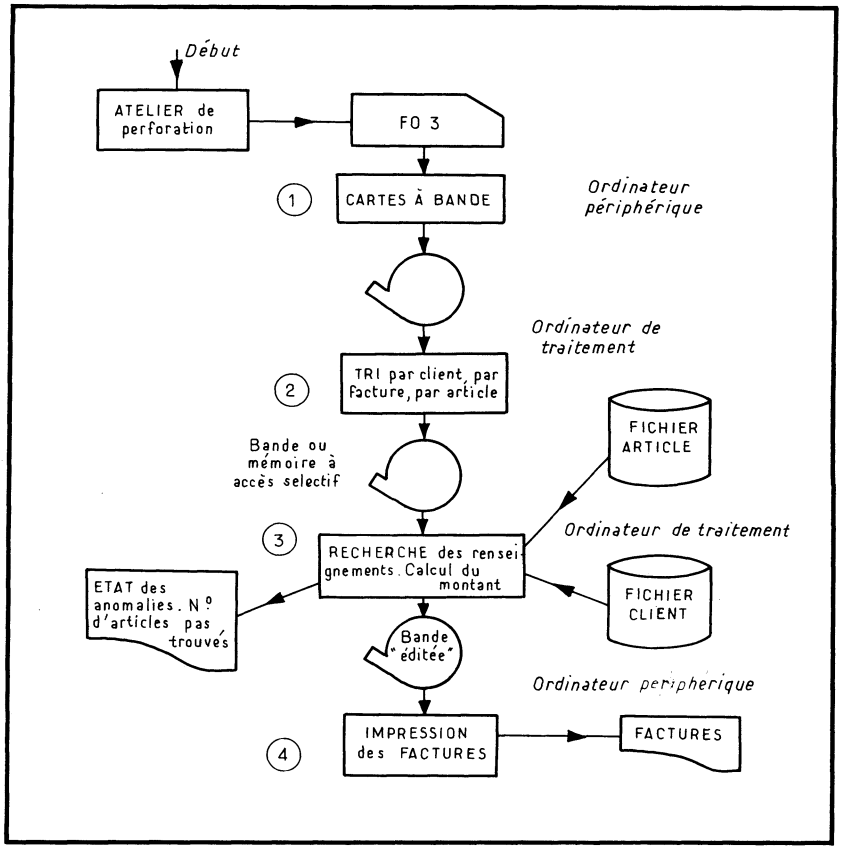


Fig. 2-27. — Organigramme d'une solution sur ordinateur à disques magnétiques.

5. — Comparaison entre les trois solutions

Nous allons comparer du point de vue temps chacune de ces *trois solutions*. Nous ferons ce calcul indépendemment de tout matériel, nous nous contenterons seulement de donner des ordres de grandeur, ceci pour fixer les idées, dans un but pédagogique.

Choisissons les hypothèses suivantes :

40 000 articles
10 000 clients
500 factures par jour en moyenne avec également une moyenne de 5 articles par facture.

5.1. — *TRAITEMENT SUR MATÉRIEL CLASSIQUE*

On a alors : 40 000 cartes F 01
10 000 cartes F 02
2500 cartes F 03

encore une fois les temps que nous donnons sont des ordres de grandeur, tenant compte à la fois de la machine et de l'opérateur.

1re opération : tri sur les colonnes 4 à 11 des cartes F 01. On prend 500 cartes à la minute et il faut un passage par colonne :

$$\frac{8 \times 2500}{500} = 40 \text{ mn}$$

2e opération : interclassement des cartes F 01 et F 03. On prend 400 cartes à la minute :

$$\frac{40\,000 + 2500}{400} = 1 \text{ h } 45 \text{ mn}$$

3e opération : reproduction. On prend 200 cartes à la minute :

$$\frac{40\,000 + 2500}{200} = 3 \text{ h } 30 \text{ mn}$$

4e opération : tri sur le code carte 500 cartes/mn :

$$\frac{42\,500 \times 1}{500} = 45 \text{ mn}$$

5e opération : tri par client, par facture — 11 colonnes 500 c/mn :

$$\frac{2500 \times 11}{500} = 55 \text{ mn}$$

6e opération : reproduction 200 c/mn mais deux magasins :

$$\frac{2500 \times 2}{200} = 25 \text{ mn}$$

7e opération : interclassement F 02 et F 04 400 c/mn :

$$\frac{10\,000 + 2500}{400} = 35 \text{ mn}$$

8e opération : reproduction F 02 F 04 200 c/mn :

$$\frac{10\,000 + 2500}{200} = 1 \text{ h } 05 \text{ mn}$$

9e opération : tri code carte 500/mn :

$$\frac{10\,000 + 2500}{500} = 25 \text{ mn}$$

10e opération : calcul du montant 150 c/mn :

$$\frac{2000}{150} = 20 \text{ mn}$$

11e opération : interclassement F 03 F 04 400 c/mn :

$$\frac{2500 + 2500}{400} = 15 \text{ mn}$$

12e opération : tabulatrice 200 c/mn :

$$\frac{2500 + 2500}{200} = 25 \text{ mn}$$

Ce qui donne un total de : 11 h 05 mn.

5.2. — *TRAITEMENT SUR UN ORDINATEUR A BANDE MAGNÉTIQUE*

1re opération : cartes à bande sur un ordinateur périphérique 2500 cartes. Le travail proprement dit demande 5 minutes mais il faut compter le double à cause de la préparation et la libération du travail (montage, démontage de la bande magnétique, mise en place des cartes) soit 10 minutes.

2e opération : tri de 2500 enregistrements sur bande : 5 minutes.

3e opération : 40 000 + 2500 enregistrements en entrée, 2500 enregistrements en sortie, peu de calcul : 10 minutes.

4e opération : tri de 2500 enregistrements : 5 minutes.

5e opération : 10 000 + 2500 enregistrements en entrée, 2500 enregistrements en sortie, peu de calcul : 8 minutes.

6e opération : impression de 2500 lignes avec saut de page — pose du papier de l'opérateur : 15 minutes.

Ce qui donne : 25 minutes d'ordinateur périphérique;
28 minutes d'ordinateur de traitement.

5.3. — *TRAITEMENT SUR UN ORDINATEUR A MÉMOIRES A ACCÈS SÉLECTIF*

1re opération : comme pour le traitement précédent : 10 minutes.

2e opération : comme pour le traitement précédent : 5 minutes. (Les opérations humaines, les réactions de l'opérateur interdisent de prévoir un chiffre plus faible pour une opération.)

3e opération : 2500 enregistrements en entrée. Recherche sur deux fichiers disques, 2500 enregistrements en sortie : 10 minutes.

4e opération : impression des factures : 2500 lignes : 15 minutes.

Ce qui donne : 25 minutes d'ordinateur périphérique;
15 minutes d'ordinateur de traitement.

La 1re solution est saturée, avec les deux autres nous pouvons considérablement augmenter notre nombre de commandes. Il s'agit ici avant tout d'un exemple pédagogique, de la mécanisation d'un seul service, d'une seule fonction, sans tenir compte des cas particuliers qui viennent compliquer l'analyse. L'informatique s'est maintenant débarrassée de son hérédité « mécanographique ». Elle ne procède plus fonction par fonction, elle est devenue la technique de base de l'organisateur. Elle commence par la création d'un système « dit de gestion intégrée ». Avec le concept « banque de données » ou « base commune de données » on verra tous les avantages de la solution disque, des mémoires à accès sélectif, avec la possibilité de lier et d'organiser tous les fichiers dans un ensemble.

En particulier le problème de facturation est intimement lié à la *tenue des stocks*, c'est-à-dire à l'existence d'une image sur disque magnétique de l'état des magasins, avec en face de chacun des numéros d'articles les quantités disponibles. A partir de cette tenue des stocks et de la conservation (de l'historique) des mouvements magasins il est possible de tracer des courbes de variation des stocks en fonction du temps. En prolongeant ces courbes vers le futur on effectue des prévisions et par suite on peut optimiser les prix et les approvisionnements. Cela s'appelle *la gestion des stocks.*

Parallèlement l'enregistrement des factures par numéro de client permet de tenir à jour des comptes-clients, ce que doivent chacun des clients. Par suite on peut vérifier les paiements, effectuer des relances, et aussi créer des historiques par client.

6. — Additif

Il est également très intéressant, dans le même cadre d'idées d'évaluer la solution manuelle, telle que nous l'avons présentée au début de ce chapitre :

1) Prendre la facture et lire le numéro du client . . . . . 5 s
2) Rechercher dans le catalogue client ce numéro . . . . . 25 s
3) Écrire sur la facture le nom et l'adresse trouvés dans le catalogue . . . . . 5 s
4) Lire le numéro article de la ligne qui suit . . . . . 5 s
5) Rechercher ce numéro dans le catalogue article . . . . . 30 s
6) Écrire sur la facture la désignation et le prix unitaire trouvés dans le catalogue . . . . . 5 s
7) Effectuer le produit quantité par prix unitaire . . . . . 25 s
8) Écrire le résultat dans la colonne montant . . . . . 5 s
9) « Y a-t-il d'autres articles à traiter sur cette facture? » Dans notre exemple il y en a moyenne 5 articles par facture. Par facture on exécutera donc 5 fois en moyenne les opérations 4-5-6-7 et 8.
10) Calculer la somme des montants de chacun des articles . . . 25 s
11) Écrire cette somme au bas de la facture . . . . . 5 s

On trouve ainsi une moyenne de 415 s par facture. 500 factures cela donne 415 s × 500 = 57 h 30 mn par jour, soit 7 personnes. Bien entendu tous les temps que nous donnons sont des ordres de grandeur. Il faut les vérifier et les préciser, sur le tas, par des expériences.

Il est possible, en l'absence de toute informatique, de réduire cette charge en spécialisant les différents postes de travail :

1) Un poste responsable du catalogue client et prenant en charge les opérations 1-2 et 3. Ce poste mettra ainsi moins de temps dans la recherche de chacun des numéros de client. De plus en groupant les factures d'un même client il n'aura qu'une seule recherche à faire pour cet ensemble. L'opération 2 ne demandera plus que par exemple : 15 s. Soit pour ce poste (5 + 15 + 5) s × 500 = 3 h 30 mn par jour.

2) Un poste responsable du catalogue article et prenant en charge les opérations 4-5 et 6. De la même manière l'opération 5 ne demandera plus que 15 s. Soit pour ce poste (5 + 15 + 5) s × 5 × 500 = 17 h 30 mn par jour.

On peut donc diviser ce poste en deux en divisant le catalogue article en deux parties. Par exemple l'une avec les numéros d'article commençant par un chiffre de 0 à 4, et l'autre par un chiffre de 5 à 9. On gagne ainsi encore un peu de temps dans les recherches.

3) Un poste équipé d'une petite calculatrice et spécialisé dans les opérations 7 et 8. L'opération 7 de multiplication ne demande plus que par exemple 7 s (frappe de 2 nombres). Soit pour ce poste (7 + 5) s × 5 × 500 = 8 h 30 mn par jour.

4) Un poste équipé d'une petite calculatrice et spécialisé dans les opérations 10 et 11. L'opération 10 d'addition ne demande plus par exemple que 15 s (frappe en moyenne de 5 nombres). Soit pour ce poste (15 + 5) s × 500 = 3 h par jour.

En confiant la responsabilité des postes 1) et 4) à une même personne et en veillant à la synchronisation des différents postes (files d'attente entre ces postes) on arrive ainsi à effectuer la fonction de cet exercice avec 4 personnes — mais sans obtenir des renseignements pour les statistiques.

Une autre solution consiste à reprendre la première avec des postes non spécialisés mais en les équipant de « facturières ». C'est-à-dire des machines à clavier possédant des dispositifs de tabulation et des organes de calcul pour les multiplications et les additions. De cette manière les personnes au lieu d'écrire manuellement les données les frappent au clavier. Les « facturières » effectuent les opérations arithmétiques 7 et 10 et aussi les opérations 8 et 11 sans perte de temps.

Le temps par facture n'est plus que de :

$$5 + 25 + 5 + (5 + 30 + 5) \times 5 = 235 \text{ s}$$

soit 235 × 500 = 32 h 40 mn par jour, soit 4 postes de travail.

On peut rapprocher cette solution de l'informatique en équipant les « facturières » de perforateur de ruban perforé. Ce ruban perforé permettra d'obtenir les statistiques sur un ordinateur. Cette solution est seulement valable lorsque les factures doivent être faites immédiatement et que l'ordinateur est loin.

Le travail de l'informaticien n'est pas différent de celui de l'organisateur. C'est rigoureusement le même.

CHAPITRE III

LA PROGRAMMATION

1. — L'ordinogramme

1.1. — ÉNONCÉ D'UN EXEMPLE SIMPLE

Avant d'entrer dans les détails de la programmation, il est nécessaire de voir comment on peut construire, réaliser, « l'ordinogramme » d'un problème simple.

On dispose d'un ordinateur périphérique possédant un lecteur de carte et une imprimante. A l'entrée, les cartes (fig. 3.1) contiennent dans les colonnes

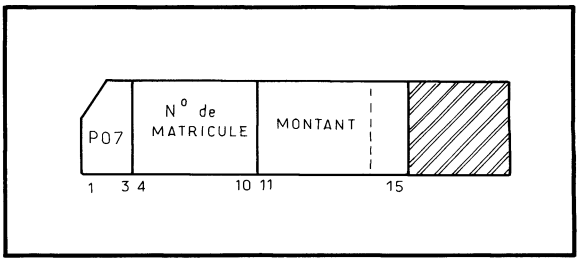


Fig. 3-1. — Dessin de carte *des cartes* à lire dans notre exemple.

1 à 3 le code carte P 07, dans les colonnes 4 à 10 un numéro de matricule, dans les colonnes 11 à 15 un montant avec 2 décimales. Ces cartes sont en séquence par numéro de matricule, mais il peut y avoir plusieurs cartes pour un même matricule.

En sortie, on désire obtenir un état (à l'imprimante); pour chaque carte lue, il faut imprimer une ligne ayant le numéro de matricule et le montant.

A la fin de chacun de ces numéros de matricule, on veut en plus imprimer une ligne donnant le total des montants pour ce matricule, comme dans la figure 3.2.

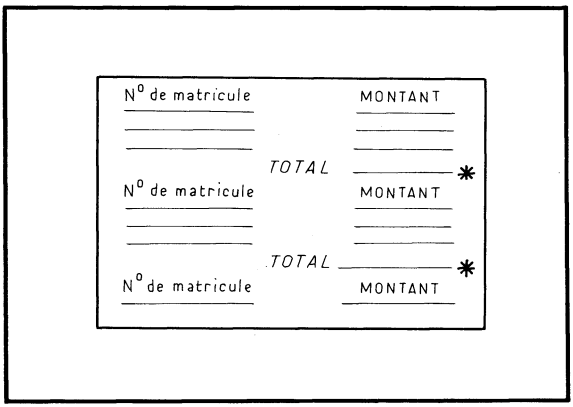


Fig. 3-2. — Dessin de l'état à imprimer dans notre exemple.

A partir de ces renseignements, comment peut-on arriver à programmer la chose? Il faut commencer par faire un ordinogramme, pour cela un crayon et une gomme sont nécessaires.

1.2. — L'ORDINOGRAMME DE CET EXEMPLE

Commençons par le « DÉBUT ».

On se dit, il faut lire une carte, mouvementer le numéro de matricule et le montant de cette carte dans la zone d'impression de la mémoire avec une édition (mise des « . » et des « , » pour les nombres, effacer les zéros non significatifs...).

Maintenant puisqu'il y a un total à imprimer on additionne le montant de la carte dans un compteur « CTR 1 » que l'on met à zéro avant de lire la 1re carte.

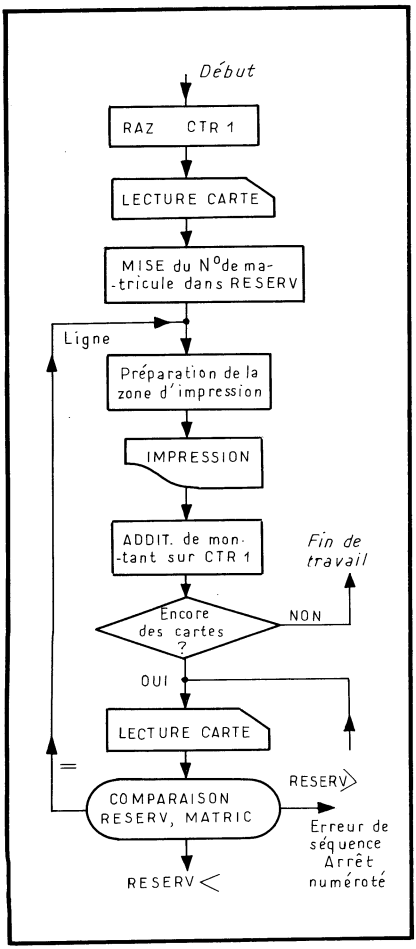
Ensuite, on lit la 2e carte et on arrive au squelette d'ordinogramme de la figure 3.3.

Après la lecture de cette 2e carte on tombe sur une impasse. En effet, suivant que le numéro de matricule de cette 2e carte est égal à celui de la 1re carte ou non, le traitement qui suit n'est pas le même. Il faut pouvoir faire une comparaison avec le numéro de matricule de la carte précédente. Or ce numéro, nous ne l'avons plus, car la lecture de la 2e carte l'a effacé dans la zone d'entrée, d'autre part dans la zone de sortie il est édité. Il aurait fallu mettre ce numéro dans une « réserve ».

Avant de lire la 2e carte il faut aussi se demander : « A-t-on des cartes à lire? » L'ordinateur possède un indicateur qui se met en fonction lorsqu'il n'y a plus de cartes à lire. Cet indicateur peut-être « testé » (interrogé) par le programmeur.

Nous prenons notre gomme et modifions l'ordinogramme pour arriver à la figure 3.4.

Fig. 3-3. — Squelette d'ordinogramme après la lecture de la deuxième carte.



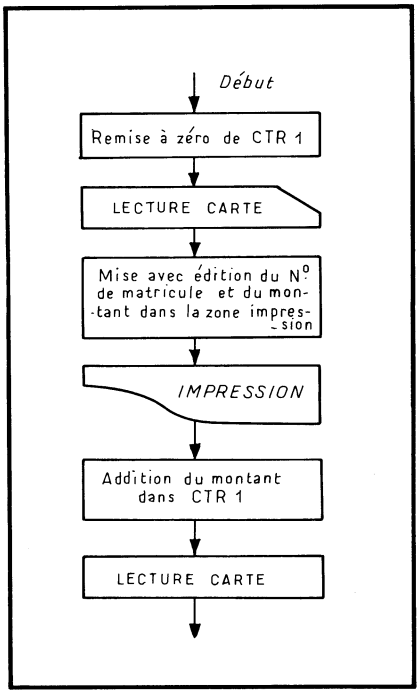


Fig. 3-4. — L'ordinogramme compare le numéro de la carte qui arrive avec celui de la première carte. *Il* regarde si il y a encore des cartes à lire.

Après la lecture de la 2e carte nous effectuons la comparaison entre RESERV et le numéro de matricule que l'on vient de lire. Voyons les 3 cas possibles :

— Si la réponse est « égale » il faut opérer comme nous l'avons fait pour la 1re carte, il est donc inutile de tout réécrire, on revient sur LIGNE où l'on va faire l'impression, addition du montant dans CTR 1 et ainsi de suite.

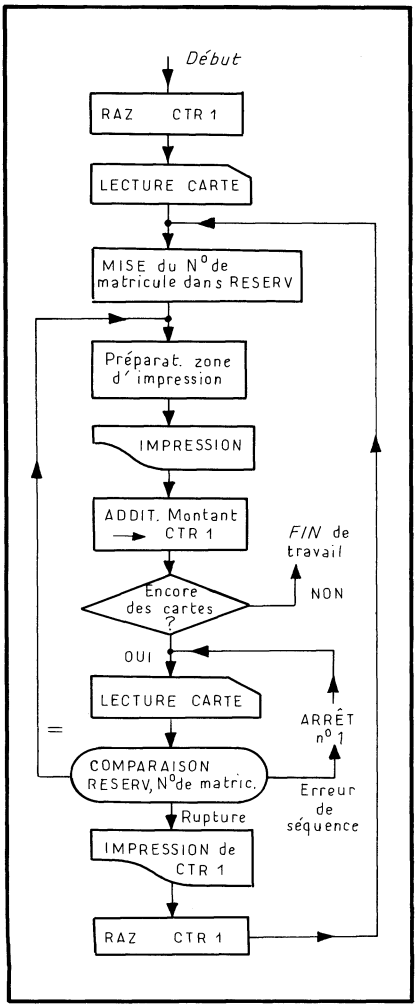


Fig. 3-5. — La comparaison donne l'un des trois résultats suivants : l'égalité (impression d'une ligne), la rupture ou l'erreur de séquence.

— Si la réponse est « numéro de matricule plus petit », cela veut dire que celui que l'on vient de lire est plus petit que celui de la carte précédente. C'est une erreur de séquence, il y a eu une erreur dans le tri précédent. Dans ce cas nous pouvons programmer un arrêt numéroté. (Numéroté : cela veut dire que le numéro sortira au pupitre, ceci pour distinguer les différents

arrêts du programme.) En appuyant sur départ on peut décider d'ignorer la carte qui a provoqué l'erreur en allant lire directement la carte suivante.

— Si la réponse est « numéro de matricule plus grand », cela veut dire que la carte que l'on vient de lire appartient au numéro de matricule suivant (on appelle cela une *rupture*) autrement dit le numéro de matricule précédent est terminé. Il faut avant de traiter la carte que l'on vient de lire, imprimer le Total qui se trouve dans CTR 1, remettre à zéro ce compteur CTR 1, et réinitialiser le contenu de RESERV avec le nouveau numéro de matricule. Notre ordinogramme devient celui de la figure 3.5.

Avant de poursuivre, introduisons une notion importante dans l'ordinogramme : « *l'aiguillage* ».

Un aiguillage dans un ordinogramme est un losange possédant une entrée et deux sorties. La première s'appelle normale, la deuxième transférée (fig. 3.6).

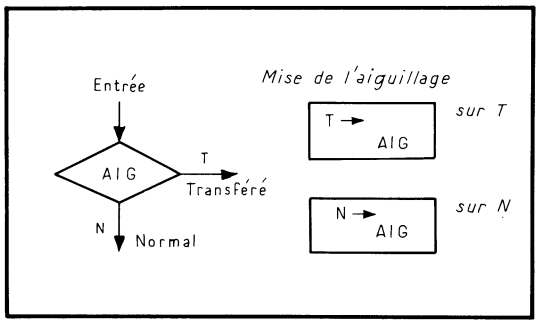


Fig. 3-6. — L'aiguillage.

En l'absence de toute opération sur cet aiguillage lorsqu'on y rentre, on sort par « normal ».

Mais il est possible dans le programme de modifier la réaction de cet aiguillage. En écrivant $\boxed{\overset{T}{\rightarrow} AIG}$ après, lorsque nous entrerons dans AIG on sortira par « transféré ». Si ensuite on écrit $\boxed{\overset{N}{\rightarrow} AIG}$ à nouveau on sortira par « normal ».

On peut à partir de cette notion modifier l'ordinogramme précédent de manière à n'avoir qu'un seul ordre de lecture carte, comme dans la figure 3.7. L'utilisation des aiguillages est bien commode, mais il ne faut pas en exagérer l'emploi car cela rend l'ordinogramme très difficilement lisible.

AIG 1 est appelé souvent aiguillage de 1re carte — car il est là seulement pour éviter que l'on passe par la comparaison lors de la lecture de cette 1re carte. Notre ordinogramme commence à prendre tournure.

La *fin* de travail? Devons-nous nous arrêter directement lorsque nous n'avons plus de carte à lire? Non, car cela signifie aussi que nous avons terminé le dernier numéro de matricule! — avant de terminer, il faut imprimer le contenu de CTR 1. Autrement dit, il faut exécuter le traitement correspondant à « Rupture » avant de s'arrêter. Ceci peut se faire à l'aide d'un aiguillage supplémentaire : AIGF, comme le montre la figure 3.8.

Il nous reste encore à effectuer le changement de page lorsque nous arrivons à la fin d'une page sur l'imprimante. Cette fin de page peut se tester de deux manières : soit compter le nombre de lignes par page, soit utiliser un dispositif de l'imprimante qui contrôle le déroulement du papier et excite un indicateur lorsque l'on rencontre la fin de page.

On peut s'arranger aussi à n'imprimer un numéro de matricule que lorsque l'on rencontre la 1re carte de ce matricule, il est inutile de le répéter sur les lignes suivantes. Enfin généralement, on imprime un total final, à la fin du travail, somme de tous les totaux partiels. Autant de modifications qu'il faut faire à notre ordinogramme.

1.3. — LES QUALITÉS QUE DOIT AVOIR UN ORDINOGRAMME

Nous venons de montrer comment on peut bâtir l'ordinogramme d'un problème simple. Il est juste de remarquer, une fois de plus, qu'il n'y a pas « une » solution, mais des solutions.

Mais il est un point important sur lequel nous voulons attirer l'attention, et cela surprend toujours, cette manière de procéder est mauvaise pour un cas réel. En effet, avec les ordinateurs modernes, on arrive en condensant tous les problèmes dans un seul programme à avoir des ordinogrammes tenant sur des feuilles de papier de 14 m, 20 m et plus de long. Un tel ordinogramme devient illisible, ou du moins un vrai casse-tête à lire.

Tout d'abord quel est le but de l'ordinogramme?

Écriture des instructions, mais ce n'est pas son rôle principal, un très bon programmeur, manipulateur d'instructions, peut à la rigueur s'en passer car il est bien imprégné de son problème.

Mise au point du programme. Un programme ne peut être remis à l'exécution que lorsqu'il marche à 100 %, lorsque la réussite est de 99,99 % ce n'est pas bon. Il va falloir le tester, nous verrons comment. L'ordinogramme permet de suivre les différentes boucles à la trace et de voir ce qui ne marche pas. (Faire marcher un programme du premier coup est une réussite exceptionnelle qui relève du miracle.)

Mise à jour du programme. La gestion évolue sans cesse. La paye et la comptabilité doivent respecter les lois, une gestion de stock doit suivre les besoins du moment, et les évolutions du marché. Les programmes sont alors modifiés, il faut pouvoir les mettre à jour aisément. Ces modifications peuvent se faire par d'autres programmeurs (ou par le même programmeur, mais celui-ci, entre temps, a été absorbé par d'autres problèmes).

Par conséquent — 1re règle — *un ordinogramme doit pouvoir être lu par n'importe quel programmeur.* L'ordinogramme doit également pouvoir être lu par les analystes, voire par les ingénieurs en organisation, ceux-ci ayant malgré tout un droit de contrôle et pouvant être amenés éventuellement à modifier les ordinogrammes.

Dans un but de généralisation — 2e règle — *Un ordinogramme doit être écrit sans tenir compte du langage de programmation qui sera utilisé.*

Enfin, 3e règle — Un ordinogramme doit utiliser la logique des blocs : Cette notion importante mérite un chapitre spécial. Elle entraîne la modularité du programme.

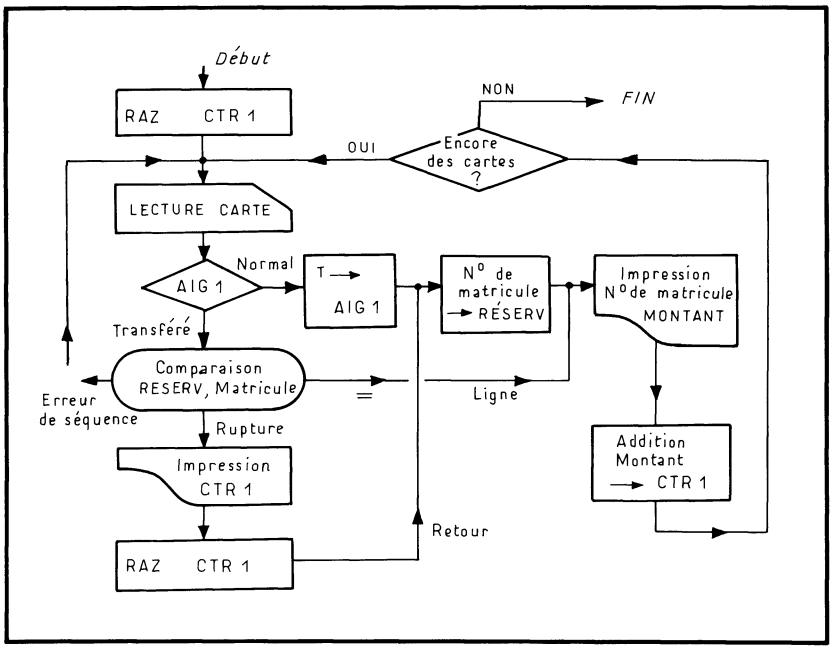


Fig. 3-7. — L'aiguillage AIG 1 permet l'utilisation d'un seul ordre de lecture.

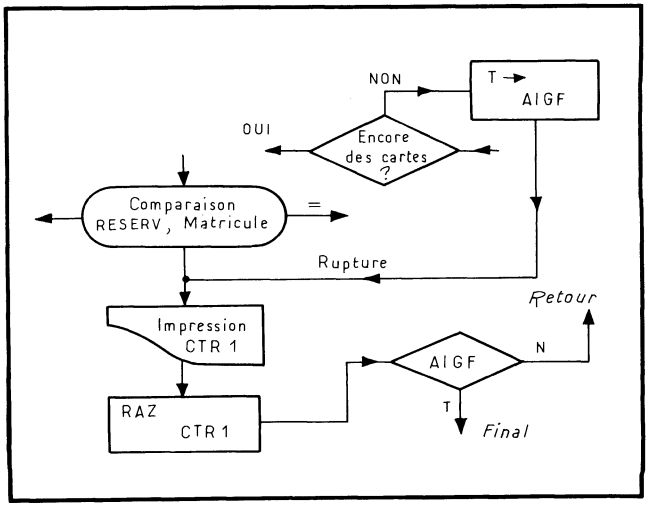


Fig. 3-8. — Si la réponse à la question : « encore des cartes? » est négative, nous positionnons l'aiguillage AIGF sur T et nous allons à RUPTURE.

1.4. — LA LOGIQUE DES BLOCS

C'est ce qui permet d'éviter les ordinogrammes de 20 pages. C'est la méthode cartésienne qui consiste à réduire la difficulté en divisant la question en autant de parties possibles.

La division d'un programme en plusieurs morceaux est souvent méconnue par les programmeurs car elle fait appel à des qualités de synthèse alors que le reste de la programmation touche plutôt aux qualités d'analyse. Pourtant, dans les problèmes complexes, la solution consiste à partir du général pour aller au particulier. Avant de s'enfouir dans les détails, il est nécessaire d'avoir une vue générale. Un problème doit être divisée en blocs, ces blocs en sous blocs, éventuellement ces sous-blocs, en sous-sous blocs et ainsi de suite.

Reprenons le programme précédent : que constatons-nous? C'est la lecture carte qui dirige l'ensemble. Lorsque nous lisons une carte il faut faire, pour chaque carte, un certain traitement : le *traitement ligne.* Lorsque nous arrivons à la fin d'un numéro de matricule il faut exécuter un traitement supplémentaire : le *Traitement rupture.* Lorsque nous arrivons à la fin du programme nous avons le *Traitement final.* Il n'est pas interdit de penser à un traitement spécial pour la *1re ligne du numéro de matricule.* A partir de ces remarques simples, notre ordinogramme devient celui de la figure 3.9.

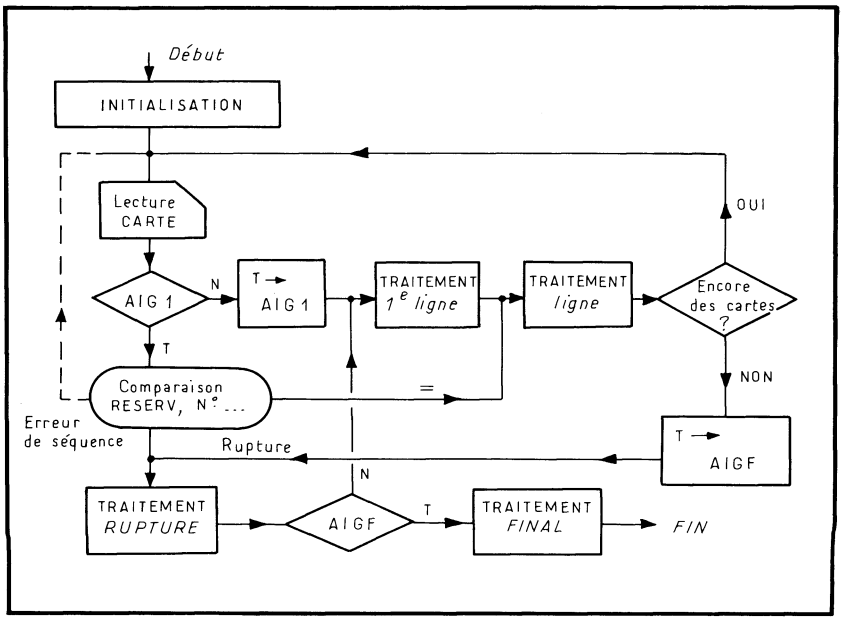


Fig. 3-9. — Ordinogramme : traitement d'un fichier séquentiel trié sur un seul argument.

Mais alors — et la constatation que nous allons faire est capitale — cet ordinogramme est valable pour n'importe quel programme ayant comme

entrée un fichier trié en séquence sur un seul argument de tri. C'est là un des avantages de la programmation par bloc : un même bloc peut se rencontrer dans des programmes différents. Il suffit de l'écrire une seule fois, on appelle cela la *modularité* du programme. En particulier l'ordinogramme général peut servir à plusieurs programmes.

Dans notre exemple, dans chacun des blocs que ferons nous? Il est inutile ici de diviser ces blocs en sous-blocs car le problème est très simple. Nous ne ferons un sous-bloc que pour le traitement de changement de page.

Bloc initialisation :

— Remise à zéro des compteurs CTR 1 et CTR 2;
— Positionnement du papier en haut de page;
— Mise de l'en-tête dans la zone d'impression;
— Impression, avancement supplémentaire du papier;
— Remise à zéro de la zone d'impression.

Bloc traitement 1re ligne :

— Mise dans RESERV du numéro de matricule;
— Mise de ce numéro de matricule dans la zone d'impression.

Bloc traitement ligne :

— Mise du montant dans la zone d'impression;
— Impression;
— Remise à blanc de la zone d'impression;
— Addition du montant dans les compteurs CTR 1 et CTR 2;
— On se pose la question : Y a-t-il fin de page? si la réponse est *non*, on sort du bloc, si la réponse est *oui*, on va vers le sous-bloc Traitement de changement de page, on revient ensuite pour sortir au même endroit du bloc.

Bloc traitement rupture :

— Mise de CTR 1 dans la zone d'impression;
— Impression, avancement supplémentaire de papier;
— Remise à blanc de la zone d'impression;
— Remise à zéro de CTR 1;
— On se pose la question : Y a-t-il fin de page? si la réponse est *non*, on sort du bloc, si la réponse est *oui*, on va vers le sous-bloc traitement de changement de page, on revient ensuite pour sortir au même endroit du bloc.

Bloc traitement final :

— Mise de CTR 2 dans la zone d'impression;
— Impression;
— Arrêt final.

En procédant de cette manière, on arrive à avoir des ordinogrammes tenant sur des feuilles de dimensions normales. La première contient l'embranchement logique des blocs entre eux — chacun de ces blocs comporte un nom et un numéro. Les pages suivantes contiennent l'ordinogramme de chacun de ces blocs. Si un bloc est trop complexe il comprend lui-même plusieurs pages : la première indiquant l'embranchement des sous-blocs, et les pages suivantes les ordinogrammes de chacun de ces sous-blocs. Et ainsi de suite s'il faut encore diviser les sous-blocs.

De cette manière la lecture d'un ordinogramme est facile. En partant de l'ordinogramme général, on a la synthèse du programme. On peut ensuite se pencher plus aisément sur un détail bien déterminé du programme. En cas de modification, on voit tout de suite quel bloc il faut remanier, et ce que cette modification apporte éventuellement dans les autres blocs.

1.5. — LES SOUS-PROGRAMMES

Une notion importante qui apporte son grain de sel à la logique des blocs est la notion de sous-programme. Il s'agit d'une série d'instructions formant un tout : un bloc. Ce bloc est utilisé plusieurs fois, à des endroits différents du programme. Bien sûr on ne veut l'écrire qu'une fois.

Deux manières :

— *Le sous-programme ouvert.*

La méthode consiste à reproduire les instructions symboliques du bloc autant de fois que nécessaire, et à placer les paquets de cartes qui en résultent à l'endroit même où on doit les utiliser dans le programme (fig. 3.10)

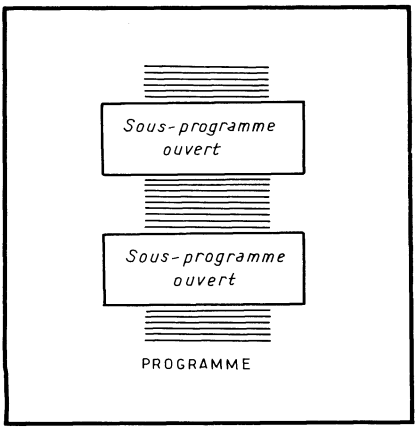


Fig. 3-10. — Sous-programme ouvert utilisé deux fois dans un programme.

— *Le sous-programme fermé.*

Le bloc n'est placé qu'une fois dans le programme. Lorsqu'on a besoin de l'utiliser, il est possible (dans les langages symboliques du programmeur, comme dans le langage absolu de la machine) de mettre en réserve le contenu

du registre instruction et de le transférer vers le sous-programme. A la fin du sous-programme on revient d'où l'on était parti (comme un boomerang) en utilisant le contenu de la réserve où nous avons stocké le registre instruction (fig. 3.11).

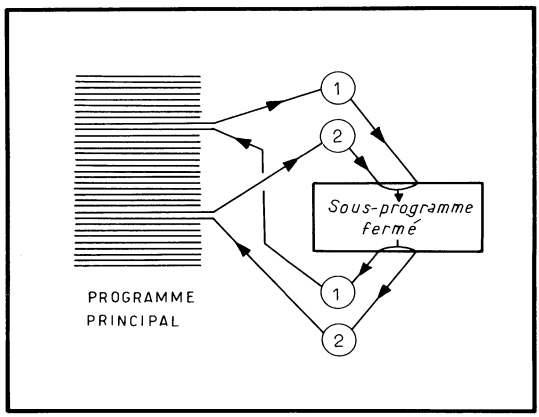


Fig. 3-11. — Sous-programme fermé appelé deux fois par un programme.

Dans l'exemple précédent, nous avons le cas qui se présente pour le sous-bloc changement de page. On a besoin de ce sous-bloc à la fois dans le traitement ligne et dans le traitement rupture. On ne l'écrira qu'une fois.

Ce sous-bloc contiendra :

— un changement de page;

— mise de l'en-tête dans la zone d'impression;

— impression;

— remise à zéro de la zone d'impression;

— un transfert à une adresse contenue dans une zone spéciale.

Dans le bloc traitement ligne, comme dans le bloc traitement rupture on placera, avant de se transférer à ce sous-programme, l'adresse de retour dans cette zone spéciale.

A noter évidemment que ce sous-programme peut paraître bien simple, il n'en est pas de même dans un cas réel. On peut du reste ici aussi utiliser ce sous-programme dans le traitement initialisation.

Remarques : Pour ceux qui voudraient chercher les modifications d'ordinogramme, nous leur signalons que l'on peut encore améliorer l'ordinogramme de notre exemple de la manière suivante :

— ne pas faire de changement de page lorsque nous venons d'imprimer la dernière ligne d'un numéro de matricule afin de ne pas imprimer le total de ce numéro de matricule, tout seul, sur la page suivante. Effectuer seulement ce changement de page après l'impression de ce total;

— ne pas imprimer le total d'un matricule, lorsque le matricule ne comporte qu'une ligne (dans ce cas en effet ce total est complètement inutile).

1.6. — UN EXEMPLE DE SOUS-PROGRAMME

Traitons un exemple de sous-programme très souvent utilisé en gestion : *la consultation de table.*

On appelle « table » une série d'arguments (par exemple des numéros de département), à chaque argument correspond une fonction (par exemple les noms de ces départements).

Il est toujours possible de ranger en séquence ces différents arguments et de s'arranger pour avoir des arguments et des fonctions de longueurs fixes (nombre de positions). Dans le cas où les longueurs des fonctions sont variables on peut en effet prendre comme fonction dans la table, les adresses où se trouvent ces fonctions proprement dites.

Pour ce qui suit on suppose donc que la table est déjà chargée en mémoire dans l'ordre : Argument 1, Fonction 1, (ou bien l'adresse où se trouve cette fonction), Argument 2, Fonction 2, ... Argument n, Fonction n, comme dans la figure 3.12. Les arguments sont en séquence :

$$\text{Argument } 1 < \text{Argument } 2 < \ldots < \text{Argument } n.$$

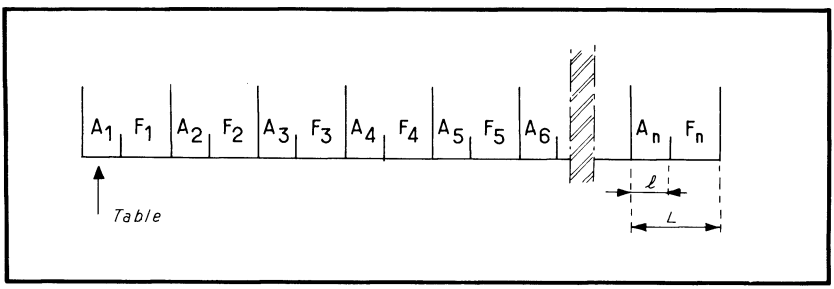


Fig. 3-12. — Table chargée en mémoire dans l'ordre : argument 1, fonction 1, argument 2, fonction 2, ...

Soit : N le nombre d'arguments;
L la longueur argument + fonction;
l la longueur argument

Cette table peut avoir été chargée en mémoire à l'aide d'un sous-programme de chargement de table. Ceci nous amène à introduire une nouvelle possibilité de la programmation : *l'indexation.* Une adresse peut-être indexée, un index étant une position bien déterminée de la mémoire, position connue de l'unité centrale. Une adresse indexée est égale à l'adresse indiquée plus le contenu de cet index. Cette possibilité existe aussi bien dans les langages symboliques que dans le langage absolu.

L'ordinogramme du bloc « Chargement de la table » peut être celui de la figure 3.13 en supposant que cette table est sur cartes et que dans chaque carte il y a un argument et une fonction.

On suppose aussi que notre table comporte en dernier une carte ayant un code carte spécial indiquant que la table est finie (c'est ce que l'on appelle une carte de fermeture). A la fin du bloc, le compteur CTR contient le nombre d'arguments de la table.

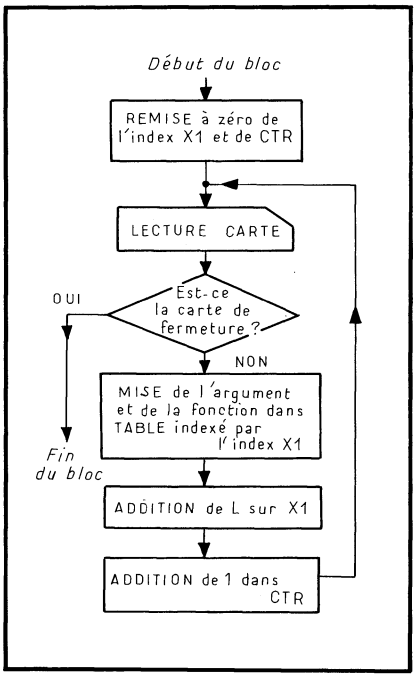


Fig. 3-13. — Ordinogramme du bloc de chargement de table.

Remarques : Il faut bien sûr, ajouter dans cet ordinogramme une vérification de séquence sur l'argument pour la sécurité. Dans le cas où la table ne contient que des constantes, il est préférable, afin d'avoir moins de manipulations, de placer ces constantes directement dans le programme. Par contre dans le cas où la table est susceptible d'être modifiée il est préférable de la traiter comme nous l'avons fait ici afin de ne pas être obligé de compiler le programme (transformer le langage du programmeur en langage machine) à chaque modification de cette table.

Notre table est maintenant chargée, nous allons pouvoir nous en servir. Les différents enregistrements que va traiter notre programme comportent des numéros, on appelle ces numéros des arguments de recherche. La consultation de table consiste à aller chercher dans la table la fonction correspondant à un argument de recherche.

Pour effectuer cette recherche il existe plusieurs méthodes : la première (fig. 3.14) consiste à comparer le premier argument avec l'argument de recherche, puis le 2e argument avec cet argument et ainsi de suite jusqu'à ce que l'on trouve cet argument dans la table, auquel cas on a trouvé la fonction correspondante, ou bien jusqu'à ce que l'on ait trouvé : argument de recherche $<$ auquel cas l'argument de recherche ne se trouve pas dans la table. Il en est de même si on arrive à la fin de la table sans avoir eu $=$ dans la comparaison.

Cette méthode de pas à pas peut-être utilisée lorsque le nombre d'arguments est très faible (< 15) mais est très fortement déconseillée lorsque le nombre d'arguments devient important.

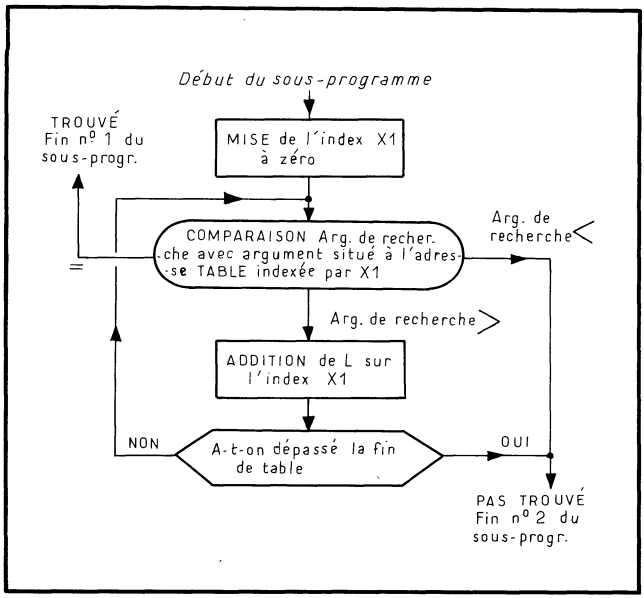


Fig. 3-14. — Consultation de table, argument par argument.

La meilleure méthode consiste à diviser la table, toujours par milieux successifs (par dichotomie), comme dans l'ordinogramme de la figure 3.15.

On compare l'argument de recherche avec l'argument milieu. Si l'argument de recherche est trop grand on compare avec l'argument situé au 1/4, s'il est trop petit avec celui situé au 3/4 et ainsi de suite jusqu'à ce que la fourchette soit réduite à un argument. Cette méthode exige très peu de comparaison et est très rapide par rapport à la précédente.

A titre d'expérience, sur un ordinateur donné, un programme, n'effectuant comme travail que de la consultation de table, demandait une heure avec la première méthode. Corrigé par la deuxième méthode il ne dure plus que 15 minutes.

1.7. — EXEMPLES D'ORDINOGRAMMES DIVISÉS EN BLOCS

Afin de montrer que la division d'un ordinogramme conduit fatalement à une généralisation des ordinogrammes nous montrons ici quelques exemples classiques :

Ordinogramme d'un programme ayant comme entrée des cartes (ou un fichier organisé séquentiellement) triées sur *deux arguments :* un majeur et un mineur (exemple : par pays — par ville), c'est la figure 3.16.

Dans le cas d'une rupture majeur il est nécessaire d'effectuer le traitement mineur avant le traitement rupture majeur (si on change de pays, on change aussi de ville). De même, avant d'exécuter le traitement final il faut passer par le traitement rupture mineur, puis par le traitement rupture majeur.

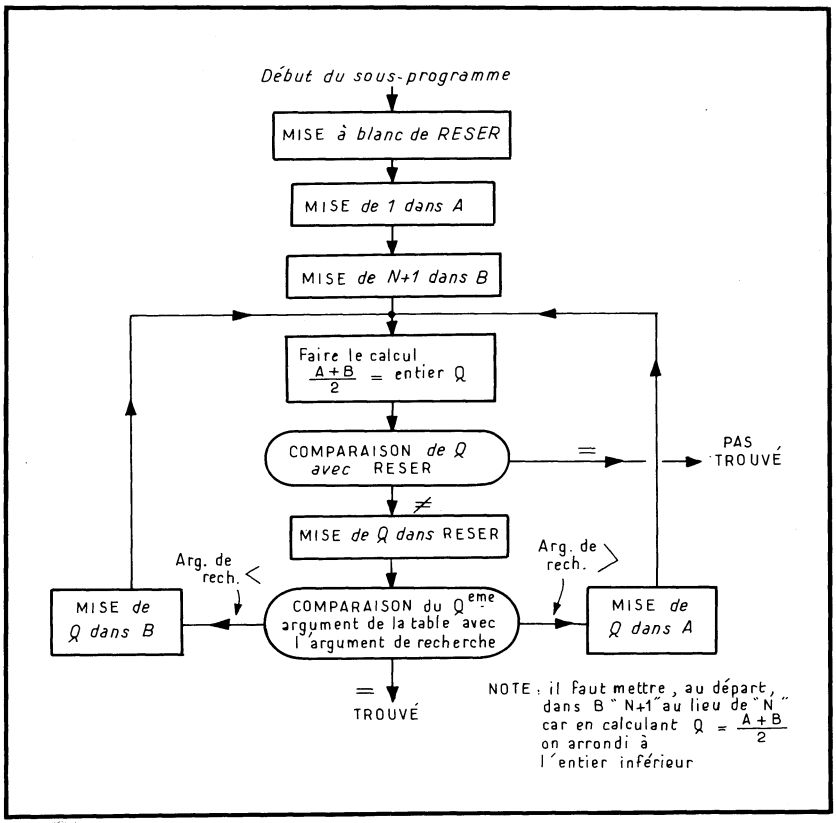


Fig. 3-15. — Consultation de table, par milieux successifs.

Nous laissons au lecteur le soin de voir comment il faut généraliser cet ordinogramme dans le cas où le nombre d'arguments dans la séquence d'entrée est supérieur à 2.

Fusion de 2 bandes magnétiques

Nous verrons lorsque nous étudierons le software que celui-ci donne directement au programmeur, l'accès à un enregistrement logique (à un article, à une fiche). Par conséquent, dans l'ordinogramme, lorsque nous écrivons lecture bande, il s'agit de la lecture d'un enregistrement logique.

Cela nous permet de raisonner, avec les bandes, comme avec les cartes.

Il en est de même pour l'écriture bande. Il s'agit ici de faire avec des bandes magnétiques le travail de l'interclasseuse. Nous avons 2 bandes possédant des enregistrements logiques de même format et triées sur le ou les mêmes arguments.

On désire obtenir en sortie une bande ayant tous les enregistrements des 2 bandes entrées dans le même ordre. L'ordinogramme est celui de la figure 3.17. Pour lire cet ordinogramme il suffit de comprendre qu'au départ les aiguillages AIG 1 et AIG 2 sont, par convention, positionnés sur « normal ». On commence par lire un enregistrement sur la bande 1 et un enregistrement sur la bande 2. On compare les arguments de ces 2 enregistrements. On écrit l'enregistrement possédant l'argument le plus faible sur la bande de sortie, puis on va lire l'enregistrement suivant sur la bande correspondante, on revient sur la comparaison et ainsi de suite. On peut remarquer, bien sûr, que ce que l'on écrit est égal à ce que l'on a lu.

Le plus délicat est de comprendre la fin de travail. Il ne faut s'arrêter que lorsque les 2 bandes d'entrée sont terminées. Lorsqu'une des 2 bandes a été complètement lue il faut poursuivre le travail avec l'autre bande jusqu'à ce que cette bande soit elle-même terminée. Une des astuces, le plus couramment utilisée, est de réserver dans le programme une constante ayant un nombre de positions égal au nombre de positions des arguments de séquence

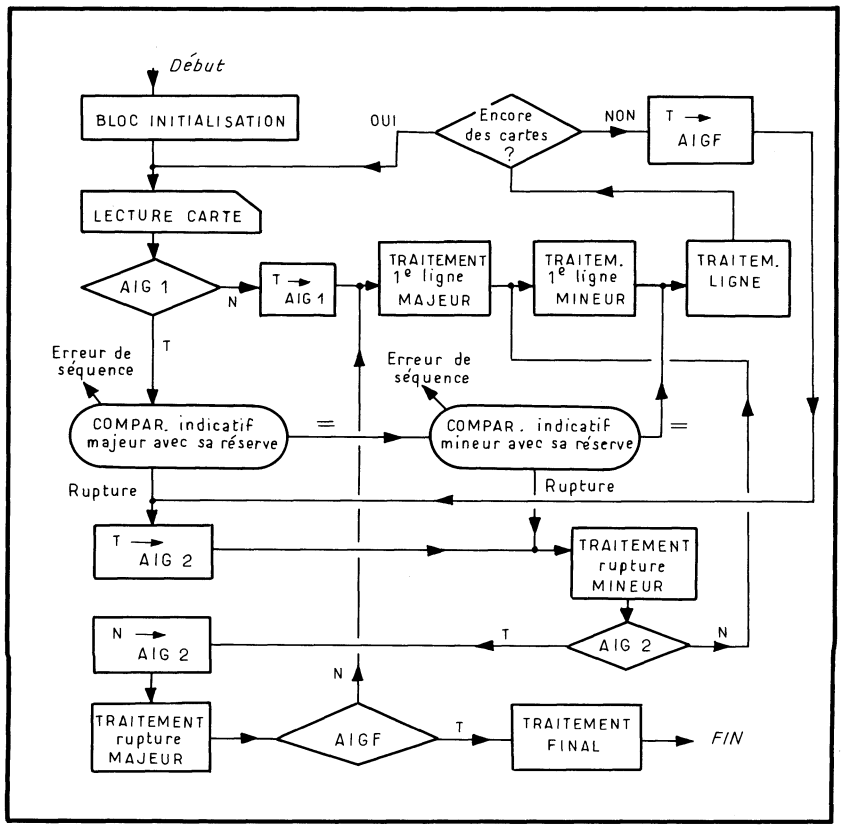


Fig. 3-16. — Ordinogramme : traitement d'un fichier séquentiel trié sur deux arguments.

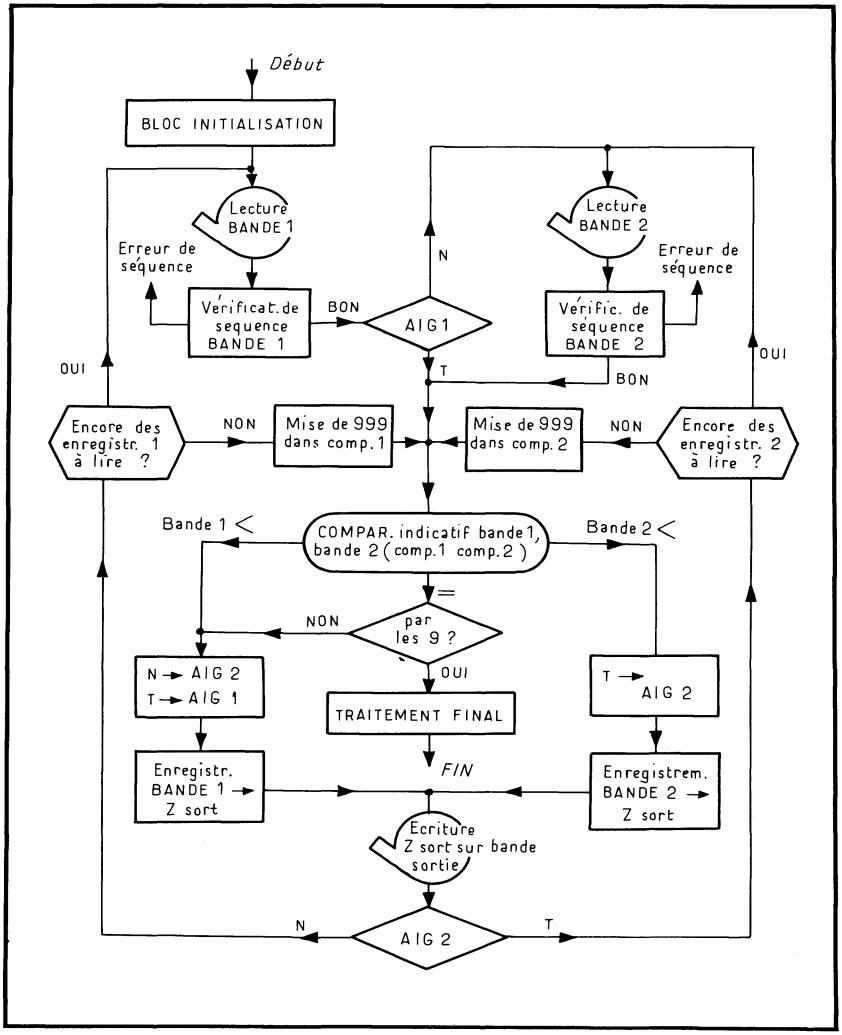


Fig. 3-17. — Ordinogramme : fusion de deux fichiers séquentiels.

des bandes. Cette constante ne contient que des 9 (ou un autre caractère s'il y a pour l'ordinateur un caractère supérieur à 9). 999999 est un numéro réservé au traitement de l'information.

Lorsque avant de lire un enregistrement, par exemple, sur la bande 1, la réponse à la question « Y a-t-il encore des enregistrements à lire? » est négative, on remplace dans l'instruction de comparaison l'adresse de l'indicatif de la bande 1 par l'adresse de cette constante ne contenant que des 9.

Puis on retourne vers cette comparaison. Cette comparaison va comparer l'indicatif de la bande 2 avec 99999. La résultat sera toujours : « indicatif de la bande 2 plus petit ». On va donc reproduire les enregistrements restants de la bande 2 sur la bande de sortie. Et cela jusqu'à ce qu'à la question : « Y a-t-il encore des enregistrements 2 à lire? » la réponse soit « non ». On remplace alors, dans la comparaison, de la même manière, l'adresse de l'indicatif de la bande 2 par l'adresse de la constante 9999.. Puis on va à la comparaison. Le résultat de celle-ci sera « égal ». On se pose alors la question : « l'égalité a-t-elle eu lieu par les 9? ». La réponse est « oui », on est sûr que les 2 bandes d'entrée sont complètement traitées. On peut exécuter le traitement final.

A remarquer, sur notre ordinogramme, lorsqu'il y a des enregistrements bande 1 et bande 2 ayant même indicatif, on place sur la bande de sortie les enregistrements de la bande 1 avant ceux de la bande 2.

A remarquer aussi que nous avons placé des vérifications de séquence bien que les bandes soient triées. C'est un principe, on doit toujours vérifier l'opération précédente. S'il y a eu une erreur de manipulation on doit s'en apercevoir tout de suite. En cas d'erreur de séquence il faut retrier la bande correspondante.

La figure 3.18 contient l'ordinogramme classique du bloc « vérification de séquence ».

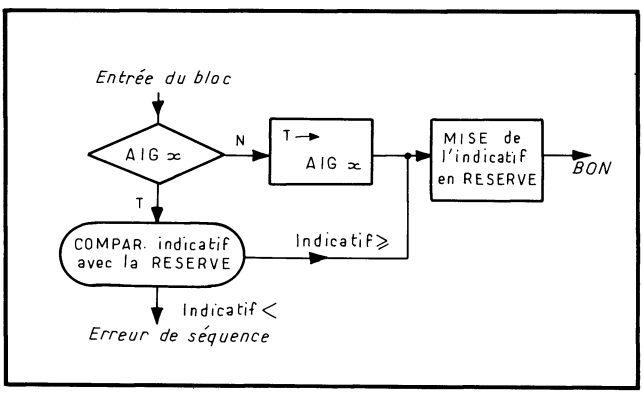


Fig. 3-18. — Ordinogramme du bloc « vérification de séquence ».

Ici aussi nous laissons au lecteur le soin de généraliser l'ordinogramme de fusion dans le cas où il y a plus de 2 bandes magnétiques.

Mise à jour d'un fichier organisé séquentiellement

La mise à jour d'un fichier est un problème de gestion qui revient souvent (fig. 3.19). On appelle « bande ancienne situation » la bande d'entrée du programme. La bande de sortie s'appelle « bande nouvelle situation ». Là aussi nous avons un ordinogramme général qui peut résoudre tous les problèmes de mise à jour d'un fichier. Il suffit d'avoir des cartes de mise à jour (on les appelle également « cartes mouvements »). Dans chacune de ces cartes, un code spécial indique si le numéro qui figure dans ces cartes est :

— un numéro à supprimer (ancien article périmé);
— un numéro à ajouter (article nouveau);
— une modification (par exemple un changement de prix unitaire).

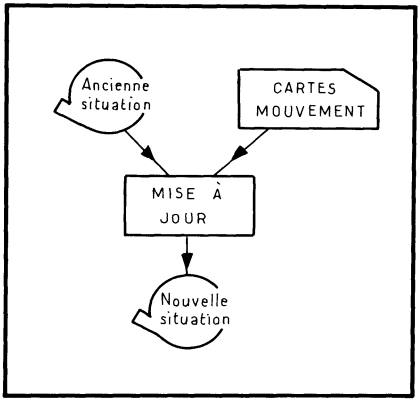


Fig. 3-19. — Mise à jour d'un fichier séquentiel.

Dans le cas d'une addition ou d'une modification, la carte contient tous les renseignements à ajouter ou à modifier. Les cartes sont en séquence par numéro comme sur la bande à mettre à jour. L'ordinogramme est celui de la figure 3.20.

Cet ordinogramme est une généralisation de l'ordinogramme de fusion. Lorsque toutes les cartes de modification sont des cartes d'addition il effectue un interclassement entre ces cartes et les enregistrements de la bande.

Dans le cas où la comparaison donne « égal », cela signifie que le numéro indiqué sur la carte se trouve sur la bande. La carte doit être une carte de suppression ou de modification sinon c'est une erreur.

Dans le cas où la comparaison donne « bande $<$ » cela veut dire que l'enregistrement bande que l'on vient de lire ne se trouve pas dans les cartes, il n'est pas à modifier, on le recopie donc sur la bande « nouvelle situation ».

Dans le cas où la comparaison donne « carte $<$ » cela signifie que le numéro de la carte ne se trouve pas sur la bande. Il doit s'agir d'une carte d'addition sinon c'est une erreur.

Pour la fin de travail, elle se détecte comme pour la fusion. Elle ne se déroulera que lorsqu'il n'y aura plus de cartes et plus d'enregistrement ancienne situation à lire.

Remarques

La logique des blocs est un principe important qui prend toute sa valeur dans les cas réels. Un autre avantage de cette division en bloc est la possibilité de tester (essayer) les blocs les uns après les autres dans l'ordinateur.

Nous avons examiné ici des programmes traitant des fichiers organisés séquentiellement. La construction et la généralisation des ordinogrammes a lieu de la même façon lorsque ces fichiers sont organisés autrement. Certains spécialistes ont présenté des normalisations dans l'écriture des ordinogrammes.

Nous pensons que ces normalisations n'apportent rien quant à la clarté. Nous sommes pour une normalisation assez souple et assez succincte (un carré pour une opération courante, un losange pour un aiguillage, un ovale pour une comparaison). L'essentiel est de faire comprendre avec le moins de choses possibles.

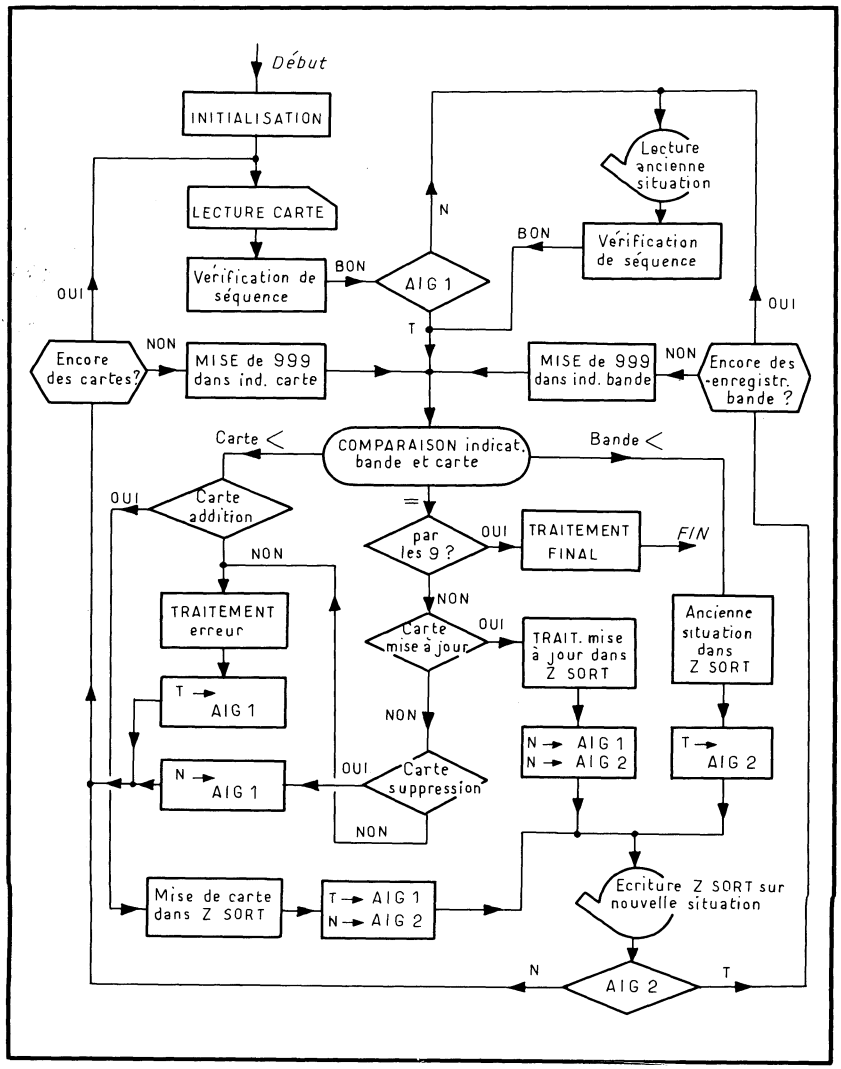


Fig. 3-20. — Ordinogramme : mise à jour d'un fichier séquentiel.

A partir de la logique des blocs il est possible de concevoir aussi des *ordinogrammes filiformes*, comme les instructions d'un programme forment une suite séquentielle dans la mémoire de l'ordinateur. Cela revient à ranger les blocs de nos ordinogrammes précédents les uns à la suite des autres et à placer entre chacun d'eux un certain nombre de conditions. Ces conditions sont reliées par des relations logiques « ET » et « OU » (de l'algèbre de BOOLE). Si l'ensemble est « VRAI » on exécute le bloc qui suit. Si l'ensemble des conditions est « FAUX » on ne doit pas exécuter ce bloc, on le saute. Après le dernier bloc de la chaîne on revient sur le premier (boucle). Bien qu'il ne soit pas raisonnable d'introduire les mathématiques là où elles ne sont pas nécessaires on peut dire qu'au fond le principal c'est d'avoir une méthode. Il n'y a pas en informatique une bonne méthode, il y a plusieurs méthodes toutes aussi valables les unes que les autres.

2. — Écriture des instructions

L'ordinogramme est maintenant construit. Il faut transformer cet ordinogramme en instructions symboliques. Cela va nous amener à parler des langages de programmation et de leur évolution.

2.1. — LES AUTOCODEURS DE BASE (ou programmation symbolique élémentaire)

La définition de ce type de langage est simple. A une instruction symbolique (instruction du programmeur) correspond une instruction absolue (instruction machine).

Ces instructions sont écrites sur des bordereaux de perforation à raison d'une ligne par instruction. Le dessin de ces bordereaux correspond sensiblement aux instructions de « l'absolu », à ceci près que l'on place des colonnes supplémentaires pour le contenu du registre instruction qu'on appelle ici « référence ».

Ainsi, par exemple, pour la machine qui nous a servi à expliquer le fonctionnement d'un ordinateur, on peut avoir un dessin de perforation comportant successivement :

— référence, code opération, adresse A, adresse B, comme dans la figure 3.21.

L'écriture du code opération est généralement obligatoire pour chaque instruction. Les codes opérations du langage absolu sont difficiles à retenir, car ils sont peu parlants. C'est pourquoi le langage symbolique utilise comme codes opérations des noms différents. Ce sont des abréviations mnémoniques de ce que font les instructions.

Les adresses A et B ne sont pas toujours obligatoires. Ces adresses peuvent être des *adresses absolues*, c'est-à-dire des adresses réelles de la mémoire principale. Il est évident que l'utilisation de ces adresses absolues n'est pas pratique, car elle oblige le programmeur à compter sans cesse les positions

de la mémoire. C'est pourquoi on préfère utiliser des *adresses symboliques* Ces adresses symboliques sont définies dans la colonne référence. Elles sont égales aux adresses où vont se trouver les codes opérations des instructions qui sont sur la même ligne. Par exemple, le premier programme de ce livre : « Reproduction de cartes » peut s'écrire à l'aide de quelque chose de semblable au contenu de la figure 3.22.

On peut utiliser le symbole RETØUR dans l'adresse A, car il est également défini dans la colonne référence. L'instruction « TRINC RETØUR » est un transfert inconditionnel à l'adresse RETØUR, cette adresse étant l'adresse du début de l'instruction « LECART 05001 ». Cette notion de référence symbolique est très souple et le programmeur peut choisir les noms qu'il désire. Il faut seulement que l'orthographe des noms soit la même, en colonne adresse comme en colonne référence.

Lorsque le programmeur donne un nom à une adresse de transfert (par exemple ici RETØUR) il doit placer le nom à l'endroit correspondant sur son ordinogramme. Ceci parce que lorsqu'il dépannera, modifiera, mettra au point son programme, il devra voir clairement la liaison entre son ordinogramme et ses instructions. Ce sont deux choses qui vont en parallèle. A une boucle de l'ordinogramme correspond une séquence d'instructions.

Sur la plupart des autocodeurs de base il est possible d'ajuster les adresses symboliques. L'écriture « RETØUR + 3 » signifie l'adresse de la 3e position de mémoire après celle de RETØUR.

Il est possible, aussi, si l'ordinateur possède des index (ou un dispositif d'adresses variables) d'écrire « RETØUR + X 1 ». Cela veut dire, l'adresse calculée : en faisant la somme de l'adresse RETØUR + le contenu de l'index X 1.

Nous arrêtons là les explications sur cet adressage symbolique, mais nous signalons que la plus grosse difficulté à laquelle se heurte le programmeur débutant est la différenciation entre le « *contenant* » et le « *contenu* ». (Dans notre exemple précédent RETØUR est l'adresse d'un contenant. Ce contenant contient le contenu LECART.)

En plus des instructions, les autocodeurs de base ont la possibilité de réserver dans la mémoire des constantes. Des instructions *déclaratives* permettent de réserver des zones d'entrée-sortie, des zones de travail, des zones pour les compteurs, enfin des constantes.

2.2. — L'AUTOCODEUR COMPLET AVEC SES MACRO-INSTRUCTIONS

Une 2e étape dans la progression des langages a été l'introduction des « *macro-instructions* » dans l'autocodeur. Une macro-instruction est une instruction symbolique à laquelle correspond plusieurs instructions absolues. Des macro-instructions sont utilisées pour relier le programme au software pour les commandes d'entrée-sortie. Par exemple : la vérification des labels de bande, les routines d'erreur, enfin tout ce que l'on appelle « IOCS » *(Input Output Control System)* et que l'on retrouvera en détail au chapitre « La préhistoire du software ». Des macro-instructions peuvent être utilisées pour relier un bloc de programme à un autre, pour des problèmes spéciaux de l'utilisateur, etc.

Une macro-instruction est un sous-programme *ouvert* qui s'écrit en symbolique avec une seule instruction. Ce sous-programme ouvert peut utiliser (et utilise généralement) les sous-programmes *fermés* de la partie du software en place dans la mémoire (moniteur).

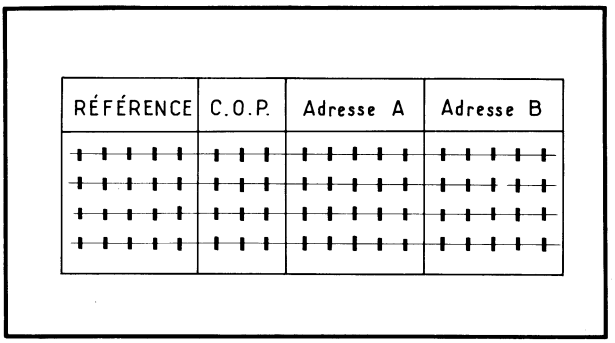


Fig. 3-21. — Bordereau de perforation d'un autocodeur de base.

RÉFÉRENCE	C.Ø.P.	Adresse A	Adresse B	N° de LIGNE
	PØSEMMØT	05080		00010
RETØUR	LECART	05001		00020
	DEPLACE	05001	06001	00030
	PERFCART	06001		00040
	TRINC	RETØUR		00050
				00060
				00070
				00080

Lignes numérotées de 10 en 10 cela permet d'intercaler les instructions oubliées

Fig. 3-22. — Écriture du programme « reproduction de cartes » en autocodeur de base.

Sur les autocodeurs de base plus récents, l'écriture des adresses est devenue plus souple. On remplace les colonnes adresse A et adresse B par une seule zone facteur. Les adresses (symboliques ou absolues) seront écrites dans cette zone facteur séparées entre elle par des virgules.

référence	COP	*facteurs*
	DEPLACE	ZENT, ZSORT

2.3. — LES LANGAGES ÉVOLUÉS

On appelle ces langages ainsi parce qu'ils sont plus proches des langages des hommes, ce qui semble sous entendre que dans l'esprit des auteurs de l'adjectif « évolué » les langages humains sont supérieurs à ceux des machines. Contentons-nous de remarquer qu'au 5e siècle avant notre ère Démocrite soutenait déjà que le langage de l'homme est d'institution arbitraire, qu'il y a actuellement de par le monde près de 1500 langues vivantes, qu'enfin les langages absolus des ordinateurs possèdent eux aussi une syntaxe et une sémantique.

On appelle « langage évolué » un langage symbolique dont la structure rappelle le langage de l'homme. Il est divisé en phrases, chaque phrase en mots parmi lesquels il y a au moins un verbe. Parmi ces mots il y a des mots clés connus du compilateur et connus de l'homme, car ce sont des mots de notre langage. Il y a aussi des mots symboliques choisis par le programmeur. Ces mots symboliques sont définis (comme pour les langages autocodeurs) soit par des phrases spéciales appelées « déclaratives », soit par des colonnes spéciales, appelées « référence », placées au début des phrases. Car les programmes écrits en langages évolués doivent eux aussi être transformés en cartes perforées, il faut donc eux aussi les écrire sur des bordereaux de perforation en respectant le dessin de ces bordereaux (un caractère par colonne).

Éducation

Un langage évolué peut s'apprendre très rapidement et ceci en ignorant tout de l'ordinateur. L'étude d'un langage évolué, même pour une personne ne connaissant pas le traitement de l'information, demande au maximum une semaine. Ceci est très avantageux pour un chercheur ou ingénieur désireux de résoudre ses problèmes sur calculateur. Cela peut amener, aussi, des frictions entre ces chercheurs et les spécialistes du traitement de l'information, car la connaissance parfaite d'un langage évolué donne l'illusion de tout savoir sur les ordinateurs. En fait, sur un ordinateur récent possédant un software moderne, il est nécessaire d'avoir, entre le chercheur (ou l'ingénieur qui vient d'écrire sa formule en langage évolué) et l'ordinateur, un spécialiste ayant une bonne connaissance des problèmes de software et capable de comprendre ce qu'a voulu écrire le chercheur.

En gestion, nous déconseillons absolument l'emploi d'un langage évolué sans des connaissances élémentaires de traitement de l'information. Le programmeur est souvent placé devant des choix où son bon sens doit intervenir.

Universalité

La propriété principale (nous pourrions dire primordiale) d'un programme écrit en langage évolué est de pouvoir passer sur n'importe quelle machine. Ceci bien sûr à la condition que cet ordinateur possède une configuration au-dessus d'un certain minimum (dimension de la mémoire principale, nombre d'entrées-sorties). Cette propriété est en général vraie pour les problèmes

scientifiques. En gestion, il n'en est pas toujours ainsi, car les programmes de gestion sont souvent trop dépendants du software, nous pensons en particulier à la gestion et à l'organisation des entrées-sorties, aux labels des bandes magnétiques.

De toute façon, le passage d'un ordinateur à un autre demande un travail infime par rapport à la réécriture complète de tous les programmes.

Ceci est un avantage énorme, car il permet à l'utilisateur de remplacer son ordinateur par un autre sans avoir à réécrire toute sa bibliothèque de programme.

L'écriture

L'écriture d'un programme en langage évolué se passe de la même manière que pour l'écriture d'un programme en autocodeur. Nous suivons l'ordinogramme et lorsque nous devons faire un transfert (GO TO) nous plaçons sur l'ordinogramme le nom symbolique que l'on se choisit dans le programme.

Certains prétendent que l'écriture d'un programme en langage évolué va plus vite que l'écriture en autocodeur. Les vieux spécialistes de l'autocodeur, héros de l'âge d'or de la programmation, prétendent le contraire. D'après les expériences que nous avons faites, nous pensons qu'en matière scientifique bien sûr, on va infiniment plus vite en langage évolué; par contre, en gestion, les deux vitesses d'écriture semblent équivalentes.

Les différents langages

Dans le domaine scientifique, citons le MAGE (méthode d'assemblage de grande efficacité), le FORTRAN (Formula Translation) avec différents niveaux, l'ALGOL (algorithmic langage).

Ces langages sont utilisables uniquement dans le domaine scientifique, car ils ne sont pas adaptés pour les nombreuses et diverses entrées-sorties de la gestion, ils sont trop rudimentaires pour certaines opérations de logique. Citons en particulier la difficulté pour mettre une constante (ou une table de constantes) directement en mémoire avec ces langages.

Le MAGE et le FORTRAN ne possèdent pas la possibilité de décomposer un programme en blocs, seul l'ALGOL la possède. Ces langages sont, par contre, facilement utilisables pour la résolution de formules mathématiques, pour la recherche d'une limite pas à pas (algorithme) pour le calcul matriciel, etc. Ils sont surtout efficaces lorsqu'ils possèdent une bibliothèque de fonctions : sinus, logarithme, tangente, etc.

Signalons en particulier l'emploi du signe « = ». Dans la phrase : TOC = TAC + TIC, le signal égal à un rôle d'exécution. On effectue les calculs de l'ensemble de l'expression placée à droite du signe (ici la somme TAC + TIC) et on place le résultat dans le contenant indiqué à gauche (ici TOC).

Le COBOL (Common Business Oriented Langage) est un langage réservé au problème de gestion. Il n'est pas utilisé en scientifique car il possède des ordres d'entrée-sortie trop lourds, et surtout une division déclarative bien trop verbeuse.

Tous ces langages possèdent comme verbe le plus utilisé (le plus puissant) le même verbe. C'est le « POUR » du MAGE et de l'ALGOL, le « DO » du

FORTRAN, et le « PERFORM » du COBOL. Il s'agit, à quelques détails près, d'exécuter n fois la même boucle. Pour cela, on donne au verbe, un symbole que l'on appelle indice. Cet indice ne peut contenir que des nombres entiers. On donne aussi une valeur initiale, une valeur limite, et un pas de progression. Au départ (fig. 3.23), le verbe place dans l'indice la valeur initiale, exécute

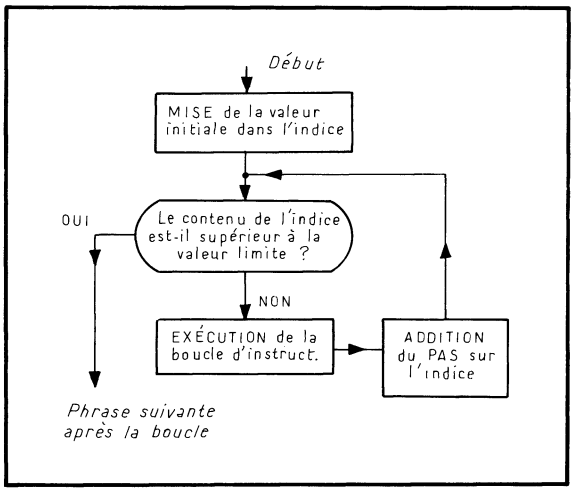


Fig. 3-23. — Exécuter n fois la même boucle d'instructions.

la boucle d'instructions, puis additionne dans l'indice le pas de progression, réexécute la boucle et ainsi de suite jusqu'à ce que l'on ait atteint, dans l'indice, la valeur limite. De plus, il est possible dans la boucle d'utiliser l'indice en considérant par exemple le « n^e nombre » d'une table, « n » étant la valeur de l'indice. Il est possible, également, de mettre dans la boucle un autre POUR, un autre DO ou un autre PERFORM en utilisant un autre indice. On dit alors que l'on a deux boucles imbriquées. On peut ainsi travailler sur des tableaux (2 indices, calcul matriciel), sur des matrices à p dimensions en imbriquant p boucles. Notons qu'il est possible de microprogrammer ce verbe, il devient alors une instruction du langage machine.

Les utilisations de langages différents pour la gestion et le scientifique ne simplifient pas les problèmes des informaticiens. Nous rencontrons, de plus en plus des problèmes touchant à la fois les 2 domaines d'application. En principe, tous les langages évolués ou non doivent permettre un lien avec les autres langages. C'est-à-dire la possibilité d'écrire un bloc de programme dans un langage, un autre bloc de même programme dans un langage différent. Néanmoins, ces liens ne peuvent être qu'une lourdeur et une perte de temps.

A l'heure où nous écrivons ces lignes, un seul langage évolué, le PL 1 (programming langage 1) permet de résoudre à la fois les problèmes scientifiques et les problèmes de gestion. Malheureusement, il semble bien, dans l'esprit de ses auteurs, que ce langage soit réservé pour un type bien déterminé d'ordinateurs. Il perdrait donc la qualité primordiale d'un langage évolué : l'universalité permettant à l'utilisateur de changer d'ordinateur, lorsqu'il le désire.

2.4. — LES LANGAGES SYNTHÉTIQUES

Avec ce type de langages, nous allons retrouver une nouvelle conséquence de la logique des blocs. Le peu d'ordinogrammes que nous avons vus démontre que l'on peut intégrer tous les ordinogrammes de gestion classique dans un seul et vaste ordinogramme général. Le compilateur contient tout cet ordinogramme. Le programmeur n'a par conséquent pas d'ordinogramme à faire puisque le compilateur le possède. Il se contente d'écrire sur des bordereaux des spécifications. Sur un type de bordereau il décrit le dessin des enregistrements, sur un autre les entrées, sur un troisième les calculs, puis sur un quatrième les sorties. Dans chacun de ces 3 derniers bordereaux (entrées, calculs, sorties) il indique ce qu'il faut faire et quand il faut le faire.

Le compilateur agit alors comme un sécateur en coupant dans son ordinogramme généralisé toutes les branches inutiles au programme. Il complète les branches qui restent à l'aide des spécifications apportées par le programmeur.

Les résultats des programmes écrits avec ces langages sont très lourds, évidemment l'utilisation d'un ordinogramme trop généralisé ne peut pas donner des performances optimales. Par contre, la mise au point et l'écriture du programme sont excessivement rapides. Le plus souvent, un programme écrit en langage synthétique marche du premier coup, alors que cela ne se produit jamais (rarement) avec les autres langages.

Il y a là une notion puissante qui sera très utile lorsque les mémoires deviendront encore plus rapides. Ce que nous pouvons dire c'est qu'actuellement un petit ordinateur périphérique orienté cartes, possédant un compilateur de langage synthétique permet aisément de résoudre les problèmes à la demande, au jour le jour, et enlève par conséquent le dernier point d'ancrage du matériel classique.

3. — La compilation

Notre programme a été écrit sur des bordereaux de perforation, il a été perforé, puis vérifié. Nous avons maintenant un paquet de cartes que l'on appelle jeu symbolique. Il faut transformer ce jeu symbolique en jeu absolu : c'est *la* compilation.

Certains parlent d'assemblage lorsqu'il s'agit d'un programme écrit en autocodeur, et de compilation lorsqu'il s'agit d'un autre langage. Nous refusons cette différenciation.

La compilation d'un programme autocodeur se passe d'une manière simple. Dans un premier temps les instructions symboliques sont lues par le compilateur, une par une. Il compte le nombre de positions prises par chacune des instructions absolues correspondantes. En même temps, il fait progresser, d'autant, un compteur (ce compteur contient l'adresse de la mémoire où l'on mettra l'instruction avant d'exécuter le programme, autrement dit ce sera le contenu du registre instruction au début de la phase analyse de l'instruction). Il se constitue en mémoire une « table des références » :

Lorsqu'il rencontre une instruction possédant une référence il la place dans la table et il met en face le contenu du compteur. Autrement dit en face de chaque référence (de chaque nom symbolique donné par le programmeur) il met l'adresse (machine ou adresse absolue) qu'elle aura lorsqu'on exécutera le programme.

Lorsqu'il rencontre une macro-instruction il doit la remplacer par plusieurs instructions (c'est-à-dire un sous-programme ouvert). Il utilise pour cette opération une bibliothèque de macro-instructions (sur bande ou sur mémoire à accès sélectif), dans laquelle pour chaque macro-instruction se trouve la liste des instructions correspondantes.

Parallèlement, au fur et à mesure qu'il traite de cette manière séquentielle les instructions et les macro-instructions du programmeur, il réécrit ces instructions et les sous-programmes ouverts remplaçant ces macro-instructions, dans l'ordre, sur une bande magnétique (ou sur une mémoire à accès sélectif).

Dans un deuxième temps cette bande (ou cette mémoire à accès sélectif) est relue et les instructions symboliques sont cette fois transformées en instructions absolues :

— décodage des codes opérations en allant chercher leur correspondant dans une table;

— décodage des adresses symboliques en adresses absolues en allant consulter la table des références que le compilateur vient de créer.

En même temps, le compilateur recopie ses instructions sur cartes, bandes ou disques, en les accompagnant d'instructions spéciales pour qu'elles puissent se mettre à l'endroit voulu dans la mémoire lorsqu'on voudra exécuter ce programme. Parallèlement, un listing de compilation sort sur l'imprimante avec par ligne : l'instruction symbolique, l'instruction absolue, l'adresse où cette instruction sera mise en mémoire au moment de l'exécution, puis éventuellement les erreurs que contient cette instruction. Car le compilateur détecte certaines erreurs dans le programme symbolique : code opération inconnu, adresse symbolique non définie, erreur de cadrage dans la perforation des cartes, etc.

Le programmeur doit alors corriger les erreurs correspondantes dans son jeu symbolique et redonner son programme à compiler et ainsi de suite jusqu'à ce que la compilation se fasse sans erreur.

La compilation d'un programme écrit en langage évolué est un peu plus complexe mais suit le même principe, à l'aide de table donnant les correspondances des verbes, des symboles et des formes des phrases, des séries convergentes pour les fonctions... (Une autre technique consiste par exemple à transformer le programme symbolique en une chaîne d'informations où chaque phrase du langage évolué est codifiée de manière à prendre le minimum de place. On peut alors laisser cette chaîne dans la mémoire de l'ordinateur. Le compilateur est divisé en plusieurs programmes, plusieurs phases. Chacune de ces phases vient tour à tour modifier la chaîne pour finalement la transformer en langage absolu. Curieuse conception où c'est le compilateur qui se déplace devant le programme. On limite ainsi les déplacements d'information mais on augmente le prix du compilateur.) On divise les compilateurs en deux classes :

— les compilateurs pour lesquels la compilation est très rapide, mais dont le résultat est à l'exécution assez loin des possibilités de la machine;

— les compilateurs pour lesquels la compilation est très lente, mais dont le résultat est à l'exécution presque à la limite des possibilités de la machine.

Car, au fond, le problème est là, si on veut un programme absolu de haute performance il faut avoir une compilation très compliquée utilisant toutes les astuces de l'ordinateur, ceci ne peut se faire qu'au dépens du temps de compilation.

Disons aussi que les langages évolués permettent une certaine souplesse et qu'il y a plusieurs manières d'indiquer la même chose. Lorsque l'on écrit un programme pour un ordinateur bien déterminé, il est possible de choisir sa construction de phrase, ses verbes, ses mots, de manière à aider le travail du compilateur et de rendre le programme résultant plus rentable. Mais évidemment cela est en contradiction avec l'universalité que doit avoir un langage évolué.

Selon la manière dont un programme est écrit, on sent les tendances, l'état d'esprit du programmeur; comme l'examen d'une écriture permet d'avoir un aperçu sur une personne. On reconnaît chez chacun ce fond de personnalité qui fait que les yeux de Modigliani, les sourires de Léonard de Vinci, (les changements de tons félins de Chopin et les sacrifices en douceur des fous d'Alekhine), sont toujours les mêmes. Tel programmeur utilise à outrance les propriétés originales de certaines instructions parce qu'il aime les astuces gratuites, telle programmeuse donne à ses références des noms de villes sur les bords de toutes les mers du globe car elle pense sans cesse à un tour du Monde en yatch, tel autre, joyeux drille, prend les prénoms de ses meilleurs copains : Toto, Jules, Amédée et Brigitte..., tel autre utilise les symboles A1, A2, A3, ..., B1, B2, ..., parce qu'il a le goût des choses rangées ou la sécheresse du mathématicien qui ne sait pas sourire. (Nous connaissons bien sûr des mathématiciens qui savent sourire.)

En ce qui nous concerne, disons cette règle d'or : En programmation, comme dans d'autres domaines, *la simplicité est payante.* Combien de fois avons-nous vu un programmeur être rayonnant de joie parce qu'il avait trouvé une astuce lui permettant de gagner quelques microsecondes dans une séquence d'instructions, et ce programmeur perdre ensuite des heures, voire des jours de mise au point? Dans l'écriture des programmes en langage évolué généralement il y a intérêt à écrire des phrases courtes. Lorsqu'on désire se faire comprendre de quelqu'un ne connaissant pas bien notre langue, il y a en effet intérêt à procéder de cette manière, ces phrases ayant un lien logique entre elles. Il en est de même pour les compilateurs.

4. — Le jeu d'essai

Nous attaquons là la phase la plus délicate de la programmation. Maintenant que notre programme est compilé et que cette compilation n'a signalé aucune erreur, il va falloir l'essayer (le tester). Pour cela, il est néces-

saire d'avoir un jeu d'essai. Qu'est-ce qu'un jeu d'essai? : c'est un ensemble de données (en général en quantité moins nombreuses que pour le travail réel) traitant *tous les cas* généraux et tous les différents cas particuliers du programme; pour lesquelles on *connaît le résultat* que doit donner le programme. Ce résultat a été obtenu d'une autre manière (manuellement, par un autre programme...), on le connaît déjà. L'essai du programme sera déclaré bon lorsque le résultat donné par le programme sera rigoureusement égal au résultat que le programmeur possède.

Cette définition du jeu d'essai peut paraître simple, et la création d'un jeu d'essai dans le domaine scientifique est en effet très simple. Il n'en est pas de même dans le domaine de la gestion qui est trop souvent, hélas, le domaine des cas particuliers. Ces cas particuliers s'imbriquent les uns dans les autres et il est très difficile d'avoir un jeu d'essai complet les testant tous. De toute manière, nous sommes contre l'emploi d'un fichier réel, ou d'une partie d'un fichier réel pour tester un programme. Ce fichier réel contient, le plus souvent, un très grand nombre de fois les mêmes cas particuliers tout en en ignorant beaucoup d'autres, qui se rencontrent une fois de temps en temps. D'autre part, en cas de modification de programme, l'existence d'un jeu d'essai permet (après une modification éventuelle dans ce jeu d'essai) de mettre plus facilement au point cette mise à jour. On verra aussi que le jeu d'essai est nécessaire pour le test automatique (essais chaînés) des programmes.

En fait, le jeu d'essai est une question d'analyse. Le jeu d'essai peut figurer déjà dans le dossier d'analyse, document que l'analyste remet au programmeur. Un bon exemple est toujours préférable à un « long baratin » — ce bon exemple peut être le jeu d'essai.

Remarque : Il faut bien sûr autant de jeux d'essai qu'il y a d'unités d'entrée dans le programme.

5. — Mise au point - Démarrage et exécution

Nous avons maintenant un jeu absolu et un jeu d'essai, nous pouvons essayer notre programme. Nous verrons au chapitre VI.3 « Les essais chaînés » comment, automatiquement, l'atelier peut :

— placer sur les différentes unités d'entrée les parties correspondantes du jeu d'essai;

— charger (mettre) le jeu absolu (les instructions absolues) dans la mémoire principale;

— lancer l'exécution, puis exécuter ce programme.

Aux premiers essais du programme, il n'est pas rare que celui-ci rate un virage et parte dans les décors, soit parce qu'il y a un dépassement de capacité, soit parce qu'il a mal positionné un aiguillage, soit parce qu'une instruction a démoli une autre instruction... etc, enfin parce que le pro-

grammeur n'a pas respecté à 100 % toutes les règles de l'art. En programmation, une note de 19,5 sur 20 est mauvaise, seul le 20 sur 20 est bon. Il faut corriger sans cesse jusqu'à ce que cette note soit obtenue.

Le programmeur doit donc pouvoir dépanner son programme. Pour cela, il doit avoir des renseignements sur cette panne.

— La première possibilité à laquelle on a pensé s'appelle *l'analyse mémoire*. Cela consiste à « vider » le contenu de la mémoire sur l'imprimante, au moment de la panne, ou bien à un moment bien déterminé de l'exécution du programme. (A noter que pour les programmeurs « analyse mémoire » est souvent impropre, ils préfèrent le terme « vidage mémoire »; l'analyse c'est eux qui la font). Dans le cas où la logique de l'ordinateur est le décimal par position de mémoire, et le programme écrit en autocodeur cette analyse mémoire permet d'aller voir par des recherches parfois assez longues la ou les causes de la ou des pannes.

— La deuxième possibilité est la *trace*. Certains compilateurs possèdent ce dispositif. Il permet, au moment de l'exécution, d'imprimer toutes les références symboliques par où le programme passe, de tel moment à tel moment. Le programmeur peut alors, à partir de cet état, suivre « à la trace » son programme et examiner sur son ordinogramme s'il est bien passé là où il fallait.

— Plus intéressante est la 3e possibilité. C'est la possibilité pour le programme de se tester lui-même.

Le programmeur ajoute dans son programme des blocs permettant de traiter des types déterminés d'erreurs. Il peut décider : en cas de dépassement de capacité d'imprimer les contenus de tels compteurs; en cas d'erreur dans la phase analyse d'une instruction d'imprimer les contenus des registres de l'unité centrale, de certaines positions de la mémoire principale, etc. Lorsque l'ordinateur rencontre un des types d'erreurs prévus par le programmeur, il passe la main (interruption du hardware traitée par le software) au bloc correspondant. A la fin de ces blocs le programmeur a le choix de terminer aussitôt son programme, soit de revenir à un endroit déterminé de celui-ci. Généralement, la compilation de ces blocs « *d'auto-tests* », peut se faire en dehors du programme, ce qui évite d'avoir à faire une compilation supplémentaire lorsque l'on désire modifier ou éliminer ces blocs.

Le programmeur a aussi la possibilité d'examiner ce qu'il a obtenu sur ses unités de sortie, de demander des analyses bandes ou des analyses disques.

Lorsque après plusieurs corrections et essais le programme a réussi « à tourner jusqu'au bout », le programmeur doit alors vérifier l'égalité entre ce que l'ordinateur a trouvé et qu'il aurait dû trouver. Et les corrections continuent jusqu'à ce que, enfin, cette égalité soit obtenue. Alors l'essai est déclaré « bon ».

Cela ne veut pas dire que le programme est parfait. Le travail réel par la quantité d'informations qu'il manipule, est une extrapolation de l'essai. Le tout bien sûr est de veiller à ce que le jeu d'essai par sa composition variée, réduise le plus possible cette extrapolation. Notre programme va être inclus avec d'autres programmes dans un ensemble : une chaîne automatique de traitement de l'information.

C'est le démarrage, premier passage de ce travail. L'atelier va l'exécuter mais les résultats devront être surveillés attentivement par les analystes, et les programmeurs. En fonction de ces résultats il pourra y avoir de légères retouches finales, après quoi la chaîne de programmes pourra vivre sa vie de chaîne...

(Nous connaissons ainsi une chaîne, fonctionnant chaque jour, dont les résultats ont été corrects pendant 365 jours, lorsque l'on s'est aperçu que le changement d'année ne marchait pas. Le jeu d'essai ne l'avait pas prévu, ce cas particulier coûte du temps lorsqu'on le rencontre.)

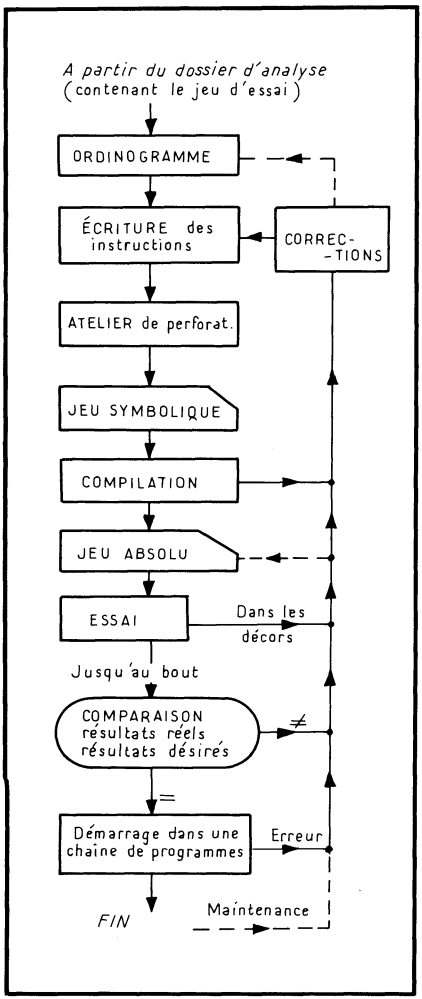


Fig. 3-24. — Organigramme de la programmation.

6. — Conclusion

Nous avons tourné la boucle. Noblesse oblige, nous présentons, avec la figure 3.24, l'organigramme du programmeur pour un programme. Nous pouvons voir que le métier de programmeur exige des qualités de logique (ordinogramme), de grammairien (instructions), de dépannage et de « pinaillage » (corrections). Il fut un temps où la difficulté d'un programme était compté à son nombre d'instructions. Il en est résulté certains chefs-d'œuvre où les boucles inutiles pleuvaient. La difficulté d'un programme n'est pas facile à estimer. Seul un analyste bien au courant des problèmes qu'il énonce, ou un spécialiste au vu du dossier d'analyse peuvent le faire. Elle s'exprime par un nombre de programmeurs multipliés par un nombre de jours, elle est fonction des cas particuliers que contient le programme et n'est pas fonction du temps machine que prendra ce programme lors de son exécution. Une paye de 40 000 employés n'est pas plus difficile à mettre en place qu'une paye de 2000. Il faut aussi dire que le produit : nombre de programmeurs par nombre de jours est une constante dans la mesure seulement où le nombre de programmeurs restent dans une certaine fourchette. (Il faut au moins autant de blocs de programmes que de programmeurs.)

Effectuons une comparaison entre les langages autocodeurs et les langages évolués. Au moment de la naissance des premiers ordinateurs on pouvait écrire le tableau de la figure 3.25.

	Autocodeur	langage évolué
éducation du programmeur	longue	rapide
vitesse d'écriture	égale	égale
compilation	égale	parfois plus longue
mise au point	longue	rapide
exécution	rapide	longue, parfois très longue, mais dépend de la manière dont le programmeur manipule son langage

Fig. 3-25. — Comparaison des autocodeurs et des langages évolués sur les premiers ordinateurs.

Sur ces ordinateurs, les langages évolués étaient utilisés pour des programmes passant sur machine une fois de temps en temps (fréquences mensuelles, trimestrielles et au-dessus). Pour les programmes journaliers et hebdomadaires on utilisait plutôt l'autocodeur. Pour les programmes passant une fois ou très rarement, il n'est pas nécessaire de perdre un temps de mise au point pour gagner un temps d'exécution. Il n'en est pas de même lorsque l'utilisation du programme est très fréquente.

Avec les ordinateurs actuels où les temps de la mémoire principale deviennent de plus en plus négligeables devant les temps des entrées-sorties, nous pensons que l'autocodeur ne doit être utilisé que pour des sous-programmes bien spéciaux et pour des domaines où il n'y a pas encore de langages évolués, comme les domaines industriels et certains traitements en temps réel.

Dans tous les autres cas, nous préconisons l'emploi d'un langage évolué, universel (permettant de passer aisément d'un type d'ordinateur à un autre). N'oublions pas non plus les langages synthétiques pour les travaux de gestion à la demande.

CHAPITRE IV

LA PROGRAMMATION SCIENTIFIQUE

1. — Avant-propos

La qualité principale d'un ordinateur, c'est la possibilité d'effectuer toujours la même chose un très grand nombre de fois pendant un temps très court. En calcul scientifique, cette faculté est utilisée de deux manières :

— Le même calcul est répété n fois en faisant varier certains indices. Le nombre n connu à l'avance peut par exemple être une dimension d'une matrice.

— Le même calcul est répété n fois, mais ce nombre n est inconnu. En reprenant toujours les résultats du calcul précédent on fait varier ces résultats vers une limite. On s'arrêtera lorsque la précision (de l'ordinateur ou une précision voulue) sera atteinte. C'est par exemple le cas pour calculer la somme d'une série convergente.

Ces méthodes de pas à pas s'appellent des *algorithmes*.

Comme il s'agit seulement de montrer ce qu'est la programmation, nous prendrons ici un exemple très simple de chacun des deux cas.

2. — Calcul du nombre π par la méthode des périmètres

Le principe est simple. Le nombre π c'est la longueur d'une demi-circonférence de diamètre 1. A l'intérieur de cette demi-circonférence on trace 3 triangles équilatéraux OAC, OCD et ODB et La Palisse peut dire que, dans la figure 4.1, la somme des 3 longueurs AC + CD + DB est inférieure

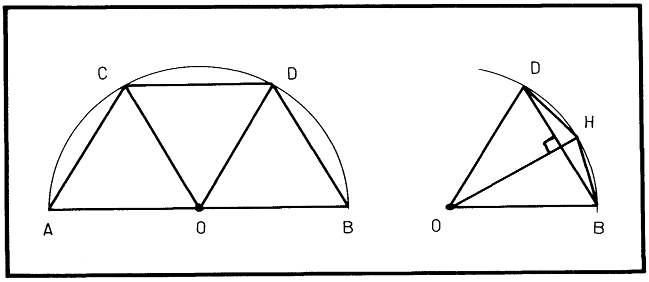


Fig. 4-1. — Calcul du nombre π.

à la demi-circonférence, comme AC + CD + DB = 3 on peut écrire $\pi > 3$. 3 côtés de longueur 1 appelons cela $C_1 = 3$ $L_1 = 1$ $\pi > C_1 L_1$.

Nous allons nous approcher un peu plus du nombre π en divisant par 2 les angles O. C'est-à-dire en multipliant par 2 le nombre de côtés sur la circonférence.

Par exemple, soit H le milieu de l'arc DB, on arrive aisément à trouver, par les règles élémentaires de la géométrie classique, la relation entre la longueur de la ligne droite DH et DB. En appelant $DH = L_2$ on a $L_2 = f(L_1)$, d'autre part le nombre de côté est $C_2 = 2\ C_1$. On peut continuer ainsi à diviser les arcs par 2, on a toujours $L_n = f(L_{n-1})$ et $c_n = 2\ c_{n-1}$ et aussi $C_{n-1}L_{n-1} < C_n L_n < \pi$. Par construction on arrive à $\pi - C_n L_n < \varepsilon$, ε étant la précision demandée.

On dit que le produit $P_n = C_n L_n$ tend vers π et il ne reste plus qu'à écrire le programme. La figure 4.2 contient l'ordinogramme général de tout algorithme, à ceci près que l'on peut remplacer la comparaison par une précision $Q - P < \varepsilon$ (c'est surtout vrai lorsqu'on atteint cette limite de part et d'autre).

Ici dans le bloc mise de P_1 dans P on écrit : C = 3 et L = 1. Dans le bloc calcul $Q = g(P)$: $CN = 2 \times C$ et $LN = f(L)$. Dans le bloc $Q \to P$: C = CN et L = LN.

Évidemment, il existe des solutions bien meilleures que cette méthode d'Archimède pour calculer le nombre π. Citons

$$\text{Arc sin } \frac{1}{2} = \frac{\pi}{6}$$

$$\frac{\pi}{3} = 1 + \frac{1}{2} \times \frac{1}{3 \times 4} + \frac{1 \times 3}{2 \times 4} \times \frac{1}{5 \times 4^2} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{1}{7 \times 4^3} + \dots + \frac{1 \times 3 \times \dots \times (2n-1)}{2 \times 4 \times \dots \times 2n} \times \frac{1}{(2n+1)\, 4^n} + \dots$$

$\frac{\pi}{4}$ = ATN(1)

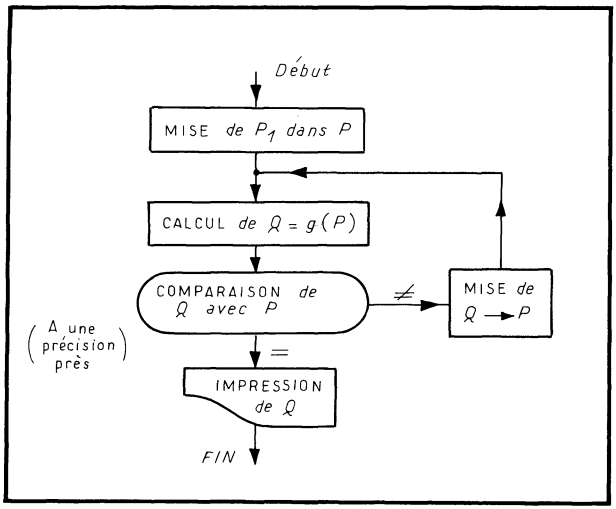


Fig. 4-2. — Ordinogramme général de tout algorithme.

$$\frac{\pi}{4} = 2 \text{ Arc tg } \frac{1}{3} + \text{Arc tg } \frac{1}{7}$$

$$\frac{\pi}{4} = 4 \text{ Arc tg } \frac{1}{5} - \text{Arc tg } \frac{1}{239} \text{ (Méchain)}$$

$$\frac{\pi}{4} = 5 \text{ Arc tg } \frac{1}{7} + 2 \text{ Arc tg } \frac{3}{79} \text{ (Euler)}$$

$$\text{Arc tg} = x\left[1 + \frac{1}{2 \times 3}x^2 + \frac{1 \times 3}{2 \times 4} \times \frac{1}{5}x^4 + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{1}{7}x^6 + \ldots + \frac{1 \times 3 \times \ldots \times (2n-1)}{2 \times 4 \times \ldots \times 2n} \times \frac{1}{2n+1}x^{2n} + \ldots\right]$$

avec $x^2 = \dfrac{t^2}{1+t^2}$ x étant le sinus de Arc tgt ou la formule de Ramanujan (1927) curieuse mais efficace :

$$\frac{4}{\pi} = \frac{1123}{882} - \frac{1123 + 21\,460}{882^3} \times \frac{1}{2} \times \frac{3}{4^2} + \frac{1123 \times 2(21460)}{882^5} \times \times \frac{1 \times 3}{2 \times 4} \times \frac{1 \times 3 \times 5 \times 7}{4^2 \times 8^2} + \ldots + (-1)^n \frac{1123 + n\,(21460)}{882^{(2n+1)}} \times \frac{1 \times 3 \times \ldots \times (2n-1)}{2 \times 4 \times \ldots \times 2n} \times \frac{1 \times 3 \ldots \times (4n-1)}{4^2 \times 8^2 \times \ldots (4n)^2} + \ldots$$

Ces différentes formules définissent chacune un algorithme. Le véritable problème c'est de choisir le meilleur. L'ordinogramme précédent subsiste, seule la formule permettant de calculer un nouveau pas à partir du pas précédent change.

Du point de vue historique, il est intéressant de noter qu'en 1949 l'ordinateur Eniac a pris 70 heures pour calculer 2037 décimales du nombre π. En 1962, un ordinateur IBM 7090 a calculé 100 000 décimales en 8 h 43 mn. (Au XIX^{e} siècle, Schanks a mis plusieurs années de sa vie pour calculer 700 décimales, il s'est trompé à la 529^{e}, c'est pourtant un exploit.)

3. — Le problème des 8 dames

Nous donnons maintenant l'exemple d'un algorithme où le nombre de pas est fixé à l'avance. Il s'agit de placer 8 dames sur un échiquier sans que l'une quelconque de ces dames puissent en prendre une autre. Un échiquier, (fig. 4.3),

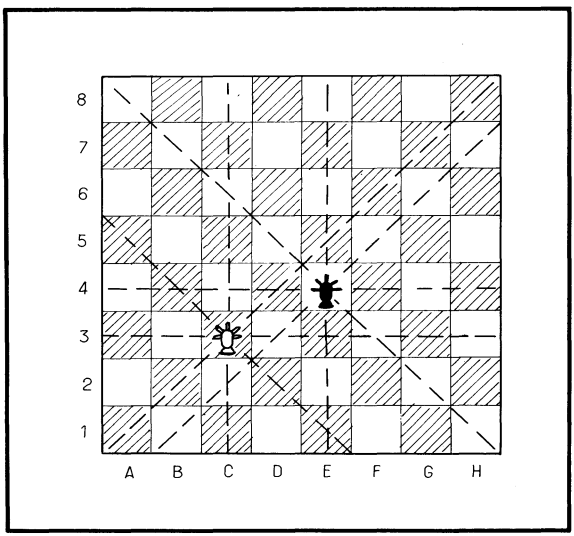


Fig. 4-3. — Deux dames sur un échiquier.

est un carré divisé en 8 lignes et 8 colonnes. La dame est une pièce marchant dans les 8 directions : nord, sud, est, ouest, nord-est, sud-est, sud-ouest et nord-ouest. Il s'agit de placer 8 dames sur l'échiquier avec les règles suivantes :

— il ne faut pas 2 dames sur la même colonne;

— il ne faut pas 2 dames sur la même ligne;

— il ne faut pas 2 dames sur une des diagonales ou sur *une des lignes* parallèles aux diagonales.

Analysons ce problème : Il nous faut donner un nom à chacune des cases. On peut baptiser les lignes de bas en haut 1, 2, 3,... 8, les colonnes de gauche à droite a, b, c,... h. Une case quelconque est au croisement d'une ligne et d'une colonne, et par exemple on l'appellera e4 pour dire qu'elle est sur la colonne 4 et la ligne e. C'est ce qu'en échecs on appelle la notation *algébrique.*

Lorsque nous aurons mis une dame sur la case e4 on ne pourra plus en mettre une autre sur la colonne e, sur la ligne 4, et sur les lignes obliques b1-h7

et a8-h1. Il existe plusieurs manières de résoudre ce problème. La solution la plus simple consiste à prendre en mémoire 8 compteurs N1, N2, N3,... N8 correspondant aux 8 colonnes de l'échiquier, comme dans la figure 4.4. En terme de programmation on écrit cela :

N (i) i variant de 1 à 8 : N (1), N (2) ...N (8)

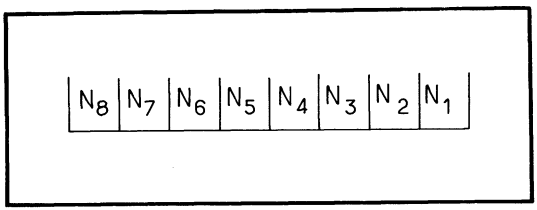


Fig. 4-4. — Représentation de l'échiquier dans la mémoire de l'ordinateur.

Pour avoir une position de ces 8 dames, il suffit de mettre un chiffre compris entre 1 et 8 dans chacun de ces compteurs. Ce chiffre donne un numéro de ligne dans la colonne. La position sera une solution si : quels que soient les nombres I et J compris entre 1 et 8, mais avec $I \neq J$ on a :

$N(I) \neq N(J)$ contrôle des lignes;

et $N(I) \neq N(J) \pm (J - I)$ contrôle des diagonales et des parallèles aux diagonales.

Le contrôle des colonnes est inutile car la schématisation des compteurs impose une seule dame par colonne. Dès lors le problème est simple, il suffit de mettre au départ : « 1 » dans chacun des compteurs et en les faisant progresser 1 par 1 jusqu'à 8, d'examiner à chaque fois si la position est une solution ou non. Écrivons cet ordinogramme en respectant la logique des blocs. Nous obtenons la figure 4.5.

Arrivé à ce stade le programmeur peut-il s'estimer satisfait? Certes son ordinogramme tourne, mais n'oublions pas qu'il est responsable de son temps machine. Que constatons-nous? Ses compteurs varient de 11111111 à 88888888, la différence entre ces 2 nombres est 77 777 777. Cette différence a beau être en base 8, c'est malgré tout un nombre énorme. Comme les boucles DØ ou POUR ne génèrent pas toujours des séquences d'instructions rapides, si nous disposons d'un ordinateur peu puissant il y a intérêt à regarder si on ne peut pas améliorer la performance de ce programme.

Au départ, nous mettons la valeur 11111111, mais nous sommes certains *que cette position* n'est pas bonne, car on ne doit pas avoir 2 chiffres égaux. En fait, nous sommes certains qu'il n'y a pas de position bonne en dessous de 12345678, nous pouvons donc commencer à cette valeur.

— Dans les comparaisons $N(I) = N(J)$? et $N(I) = N(J) \pm K$? il est préférable de commencer par les chiffres les plus significatifs, les centaines de millions et les dizaines de millions, plutôt que par les unités et les dizaines; par $N(8) = N(7)$? etc., plutôt que par $N(1) = N(2)$. En effet, si par exemple on trouve $N(8) = N(6)$ cela signifie que toutes les positions ayant cette valeur dans N (8) et N (6) ne sont pas des solutions. Par conséquent, il n'est pas utile, dans le bloc de progression, de faire + 1 sur N (1). Il est préférable de faire + 1 directement sur N (6) et de mettre respectivement 1, 2, 3, 4, 5 dans les positions N (5) à N (1).

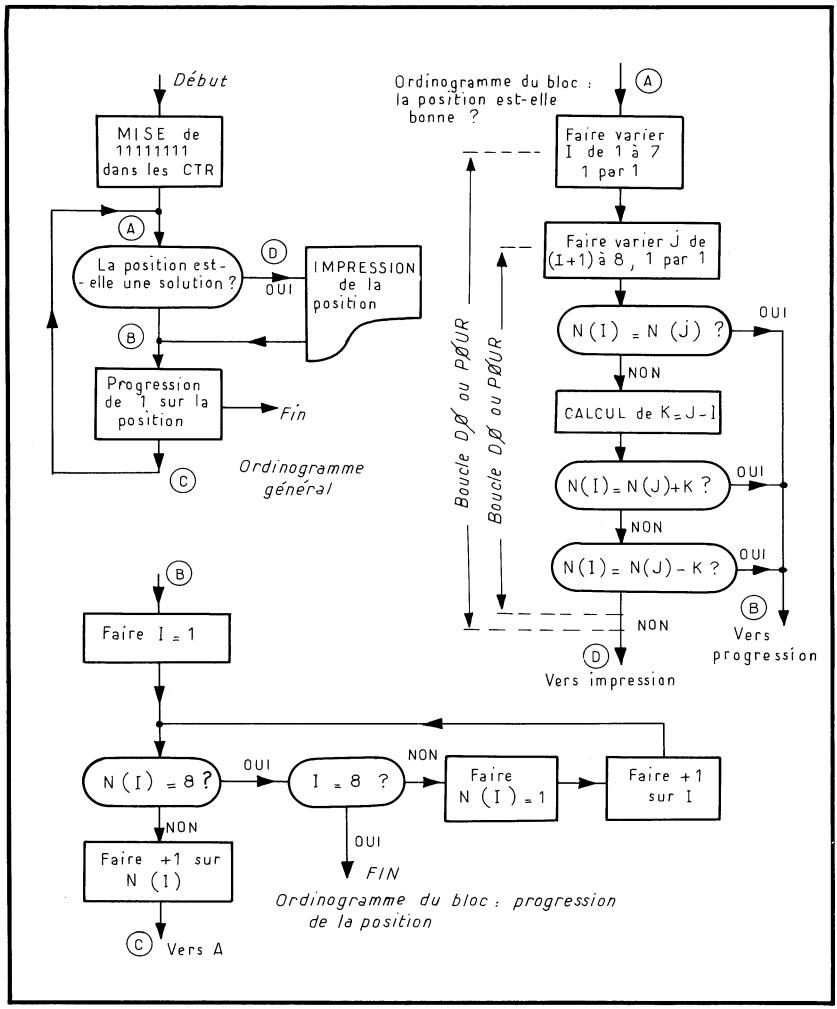


Fig. 4-5. — Ordinogramme du programme pour résoudre le problème des huit Dames.

— Sur l'échiquier il y a des symétries. Ainsi, par exemple, lorsque nous sommes arrivés à avoir N (8) = 5, nous pouvons nous arrêter car toutes les positions qui restent à trouver se déduisent de celle que l'on possède par une symétrie par rapport à l'axe horizontal de l'échiquier. Un ordinateur demandant 3 heures pour exécuter l'ordinogramme, tel que nous l'avons donné, après toutes ces améliorations le déroulera en 10 minutes.

4. — La formulation récursive

On dit qu'un bloc de programme est récursif lorsque dans ces instructions figure l'appel à un sous-programme rigoureusement identique à lui (de même nom). Il se rappelle lui même. C'est avant tout une possibilité supplémentaire des langages évolués ALGOL et PL 1. Cela évite parfois d'écrire les fastidieuses boucles DØ et POUR. Éclaircissons la chose avec l'aide de l'exemple le plus simple : le calcul de Factorielle.

Il est de la forme : $S_1 = \text{Constante}$

$$S_n = f(S_{n-1})$$

avec ici $1! = 1$ donc $S_1 = 1$

et $n! = n \times [(n-1)!]$ $\quad S_n = n \times S_{n-1}$

La méthode itérative classique consiste à calculer S_2 à partir de S_1, puis S_3, puis S_4,..., pour obtenir S_n. Elle exige l'écriture d'une boucle DØ avec un indice variant de 1 à n.

La méthode récursive consiste à écrire dans le langage évolué purement et simplement la formule $S_n = f(S_{n-1})$, et à dire à l'ordinateur : « débrouillez-vous ». Cette attitude est bien plus commode et ô combien plus naturelle.

Par exemple en ALGOL le calcul de Factorielle peut s'écrire de la manière suivante :

REAL PROCEDURE FACTORIEL (N); VALUE N, INTEGER N; BEGIN FACTORIEL : = IF N = 1 THEN 1 ELSE N X FACTORIEL (N — 1) END.

La première ligne ne sert qu'à définir le bloc de programme (la procédure) FACTORIEL et son paramètre N. Le reste est équivalent à :

Si N = 1 LE RÉSULTAT EST 1 AUTREMENT LE RÉSULTAT EST N X FACTORIEL (N — 1).

Le nom de la procédure FACTORIEL figure parmi ces instructions avec le paramètre $(n - 1)$. Le compilateur doit traduire (compiler) cela en langage machine. Le résultat n'est pas évident : Le calcul de FACTORIEL (N — 1) exige l'appel d'un bloc de programme (ou sous-programme) rigoureusement identique à celui que l'on vient d'écrire. Par exemple si au départ $n = 5$ à l'entrée de FACTORIEL (N — 1) le paramètre a la valeur $5 - 1 = 4$. Il devra lui même appeler un autre bloc de programme rigoureusement identique FACTORIEL (N — 2) et ainsi de suite jusqu'à ce que le paramètre ait la valeur 1. Dans le cas où $n = 5$ on a ainsi une chaîne de 5 blocs de programmes (sous-programmes les uns par rapport aux autres), comme dans la figure 4.6.

Deux remarques importantes :

— Le nombre de blocs de programme nécessaires dépend du paramètre n. On ne sait pas au moment de la compilation de combien on aura besoin. On est obligé de les appeler au moment de l'exécution. L'occupation mémoire est variable : on dit que l'on gère « *dynamiquement* » *la mémoire*. On retrouvera cette obligation avec la multiprogrammation et le traitement en temps réel.

— Les 5 blocs de programme précédents forment ce qu'on appelle une « *pile* ». On utilise d'abord le premier avec $n = 5$, puis le deuxième avec

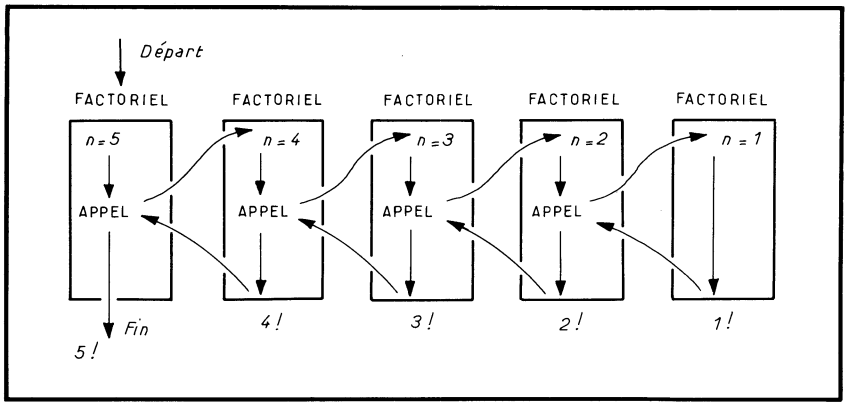


Fig. 4-6. — Chaîne de 5 sous-programmes identiques pour calculer 5!

$n = 4$, ..., enfin le dernier avec $n = 1$. Ensuite on les réutilise mais en sens inverse, l'avant-dernier avec $n = 2$, puis celui avec $n = 3$,..., enfin le premier avec $n = 5$. C'est très exactement ce qui se passe avec une pile d'assiettes. Les assiettes sont mises les unes au-dessus des autres. On les enlève ensuite en commençant par le haut de la pile : la dernière mise.

Bien sûr, la véritable astuce consiste comme dans la figure 4.7., à ne placer en mémoire qu'une seule fois les instructions du bloc de programme.

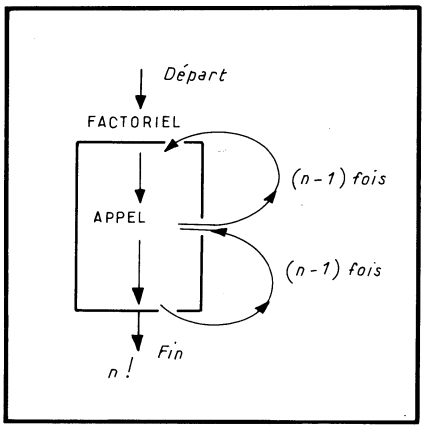


Fig. 4-7. — Une seule copie du sous-programme en mémoire pour calculer *n!*

C'est-à-dire qu'on utilise une suite d'instructions comme celle de la figure 4.8., le registre 1 « REG 1 » contenant la valeur n, et le registre 2 « REG 2 » devant à la fin du calcul contenir le résultat.

APPEL est ici en réalité une « macro-instruction ». C'est-à-dire qu'elle est équivalente à plusieurs instructions machines. En effet lorsqu'un bloc de programme appelle un sous-programme il est nécessaire de lui transmettre l'adresse de retour (là où le sous-programme revient tel un boomerang). Il faut aussi lui transmettre des paramètres. (Bien sûr dans cet exemple très simple il est

possible de conserver les contenus des registres REG 1 et REG 2, dans des exemples plus compliqués il faut très souvent sauvegarder le contenu des registres avant d'appeler le sous-programme).

RÉFÉRENCE	CODE opération	FACTEURS
FACTORIEL	COMPARAISON	REG 1 , 1 Comparaison avec 1
	TEGAL	UN Transfert à UN si égal
	SOUSTRACTION	REG 1 , 1 soustraction de 1 dans REG1
	APPEL	FACTORIEL appel du sous-prog. FACTORIEL
	ADDITION	REG 1 , 1 addition de 1 dans REG 1
	MULTIPLICATION	REG 2, REG 1 multiplication, le résultat allant dans REG 2
	T	FIN Transfert inconditionnel à FIN
UN	METTRE	REG 2 , 1 mettre 1 dans REG 2
FIN	END	Fin du bloc de programme

Fig. 4-8. — Écriture du programme récursif FACTORIEL.

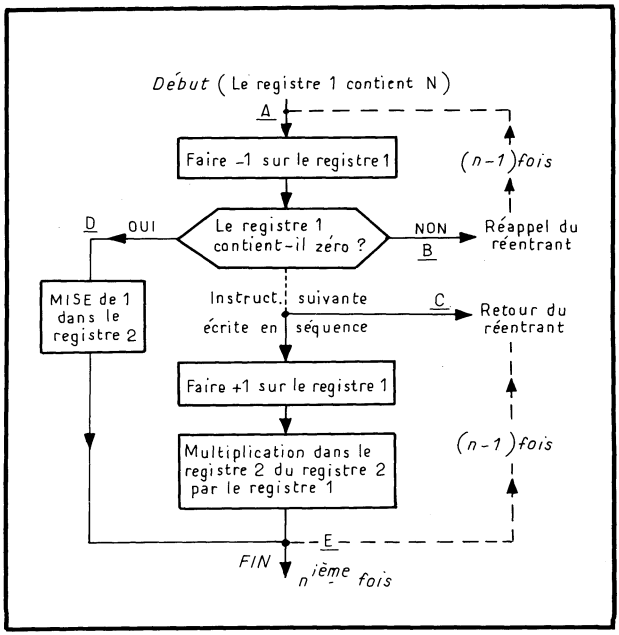


Fig. 4-9. — Ordinogramme du programme récursif FACTORIEL.

Si donc on appelle $(n - 1)$ fois le même programme on lui donne $(n - 1)$ fois une adresse de retour et des paramètres. Si on appelle $(n - 1)$ fois un sous-programme on doit aussi revenir $(n - 1)$ fois. Le principe de la « pile » d'assiettes citée précédemment subsiste pour la gestion des adresses de retour et des paramètres.

La figure 4.9. présente l'ordinogramme du programme récursif FACTORIEL, et la figure 4.10. le déroulement des instructions pour calculer 5!

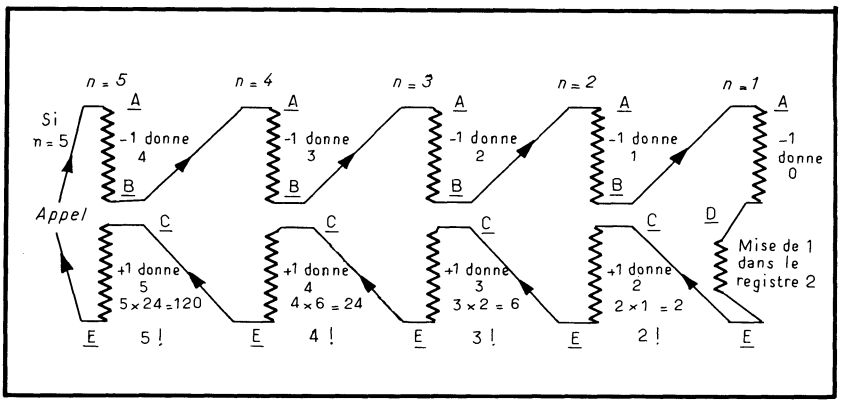


Fig. 4-10. — Déroulement des instructions pour calculer 5!

Les facteurs des instructions précédentes ne sont que des registres ou, pour les transferts, des références d'instruction. Nous verrons au chapitre Multiprogrammation qu'un tel programme est également « réentrant ». (Chapitre VII.5.3.)

5. — La virgule flottante

En gestion, les nombres que l'on traite restent entre des limites que l'on connaît à l'avance. Il suffit de réserver en mémoire des compteurs assez grands pour recueillir les résultats, la position de l'unité est déterminée par une position connue du compteur. On appelle cela de l'arithmétique fixe.

En scientifique, il n'en est pas de même. Les dimensions des nombres peuvent varier entre des écarts très grands. Soit des nombres composés de chiffres significatifs suivis de n zéros avant la virgule pour les grands nombres, soit des nombres composés de p zéros après la virgule avant les chiffres significatifs pour les nombres minuscules. Il n'est pas possible de traiter directement ces nombres en arithmétique fixe, car cela exigerait des réservations mémoires importantes et des unités de calcul spécialisés.

On choisit pour ces nombres une représentation différente. Ils sont artificiellement ramenés à des nombres décimaux compris entre 0,1000 et 1,0000 que l'on multiple ensuite par une puissance de 10 convenable : On considère

pour un nombre donné ses premiers chiffres significatifs. Ils forment ce qu'on appelle la mantisse. En prenant le premier chiffre de cette mantisse comme chiffre décimal (le premier après la virgule) on examine par quelle puissance de 10 il faut multiplier cette *mantisse* pour obtenir le nombre. Cette puissance s'appelle l'*exposant.*

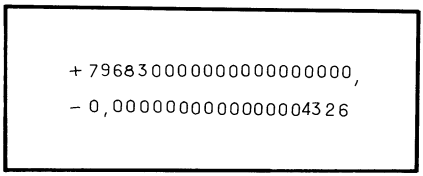


Fig. 4-11. — Les nombres très grands ou très petits. Lilliput et Brobdingnag.

On représente ainsi (fig. 4.12) en mémoire un nombre par 2 valeurs positives ou négatives. Très souvent aussi l'exposant n'est pas considéré comme une puissance de 10 mais une puissance de 2. Cette « arithmétique flottante » est réservée au domaine scientifique et si l'ordinateur possède une arithmétique binaire il y a intérêt à convertir en binaire la mantisse et l'exposant.

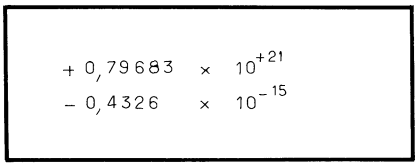


Fig. 4-12. — La mantisse et l'exposant. La virgule flottante est en fait bien fixe.

Les langages MAGE, FORTRAN, ALGOL, PL1, permettent de travailler directement sur du flottant, mais avec des puissances de 10 (il y a donc une conversion en binaire effectuée par le compilateur).

La précision d'un nombre dépend de son nombre de chiffres significatifs. Plus la mantisse comprend de chiffres plus le nombre est défini avec précision (par rapport à lui-même). Très souvent les ordinateurs possèdent des jeux de précision différents suivant le nombre de chiffres significatifs qu'acceptent leurs instructions. Il y a ainsi de la simple précision, de la double précision, etc.

6. — Conclusion

A partir de la propriété qu'a l'ordinateur de répéter un grand nombre de fois la même chose dans un temps très court, la voie lui est ouverte vers des méthodes de calcul automatiques très complexes. Ces méthodes sont actuellement en pleine évolution : analyse numérique, simulation de problèmes aléatoires, schématisation d'un problème par un graphe, par un modèle mathématique... Il suffit d'examiner ce qui se passe dans tous les domaines de la recherche, de l'expérimentation, d'avant-garde : physique nucléaire, domaine de l'espace, domaine médical, domaine de la chimie, des transports... pour se rendre compte que les ordinateurs scientifiques sont liés à l'avancement du progrès.

Actuellement, on distingue 2 types d'ordinateurs scientifiques :

— le petit ordinateur de bureau capable de résoudre les problèmes quotidiens de l'ingénieur;

— l'ordinateur de grande puissance capable de résoudre des problèmes très vastes.

Si l'on compare le scientifique et la gestion, on peut dire que le scientifique fait appel à des connaissances mathématiques poussées, à des calculs qui peuvent être complexes; par contre, une fois que l'analyse du problème est faite, elle est claire et nette. En gestion, l'opération « division » représente un maximum; par contre, l'analyse une fois terminée n'a rien de définitif. Il y a de nombreuses opérations de logique, des cas particuliers qui changent sans cesse au gré du client. Les rapports avec l'humain ne se résument pas aisément à l'aide de signes « = ».

En programmation scientifique on a tendance à introduire toutes les données d'un seul coup en mémoire, puis ensuite seulement de faire les calculs. En programmation de gestion, cela n'est pas possible du fait du très grand nombre de ces données. On lit une faible partie de ces données, on les traite, puis ensuite on revient lire les données suivantes et ainsi de suite. Il y a donc dans ces deux catégories de programmes deux philosophies différentes.

CHAPITRE V

L'ANALYSE

1. — Un exemple d'analyse : une paye

Définition : l'analyse, dans le traitement de l'information, consiste, à partir de documents d'entrée que l'on possède (ou que l'on peut posséder), à déterminer les différents traitements, les différents programmes nécessaires pour obtenir des documents de sortie que l'on désire; autrement dit, à examiner ce qu'il faut faire pour passer d'un nœud d'entrée à un nœud de sortie.

Examinons ici comment la chose peut se traiter sur un exemple classique (non intégré), puisque partout on fait au moins une paye. Aussi doit-on prendre une précaution supplémentaire : « Tous les détails, toutes les remarques que nous allons rencontrer sont purement imaginaires, et s'il y a coïncidence avec ce qui se passe dans une société quelconque, cela ne peut être dû qu'au hasard. »

1.1. — LE NŒUD DE SORTIE

Avant d'analyser un problème il faut d'abord examiner d'où on part et où on va. Il est préférable de commencer par ce que l'on désire obtenir, afin de pouvoir ensuite choisir les entrées. Le premier travail de l'analyste consiste à examiner le nœud de sortie.

Dans notre paye de quoi avons-nous besoin?

— Il est nécessaire de remettre à chaque employé *un bulletin de paye.* Ce bulletin doit contenir : le nom, le prénom, le numéro de matricule de l'employé dans l'entreprise, son numéro de service, sa position, son salaire minimum

garanti, son salaire brut (si on considère que tous les employés sont au forfait mensuel), les différentes retenues courantes (sécurité sociale, mutuelle, retraite), les différentes primes positives ou négatives (transport, déplacement,... grève, absence,...). Ces retenues et ces primes doivent être expliquées par un libellé en clair. Il faut aussi la date : année et mois. En bas du bulletin figure le total déclarable et le total que l'on donne à l'employé. Ces bulletins, afin d'en faciliter la distribution doivent être imprimés par numéro de service et à l'intérieur de chaque service par matricule.

Avant d'aller plus loin une remarque s'impose : l'analyste dans ses différentes recherches se doit pour chaque chose de répondre toujours aux mêmes questions : Quoi? Pourquoi? Quand? Où? Comment? Qui?

Répondons à ces questions pour le bulletin de paye.

Quoi?... le bulletin de paye.

Pourquoi? Pour l'employé c'est un justificatif de la somme qu'il reçoit.

Quand?... Une fois par mois, à la fin du mois.

Où?... Il faut le remettre à l'employé.

Comment? Le bulletin de paye contient tout ce que nous avons décrit précédemment. En séquence par service, par matricule.

Qui? L'ordinateur imprime le bulletin de paye.

Lorsque l'analyste a pu répondre à ces 6 questions, il a complètement défini les bulletins de paye. Nous laissons au lecteur le soin de répondre à ses questions pour les documents suivants :

— Il est nécessaire pour l'entreprise d'archiver les bulletins de paye en séquence par matricule. C'est ce qu'on appelle *l'état de paye.*

— Pour les employés payés directement sur leur compte en banque, il est nécessaire d'avoir un état *bordereau banque* classé par nom de banque, à l'intérieur de chaque banque par agence et à l'intérieur de chaque agence par numéro de compte. En face de chacun de ces numéros, le nom, le prénom et la somme qu'on lui donne. Il faut un bordereau par agence avec en bas le total que l'on remet à l'agence.

— Pour les employés payés en espèces, il est nécessaire d'imprimer un *bordereau du payeur* classé par service et à l'intérieur de chaque service par numéro de matricule avec en face la somme que l'on remet à l'employé. Il faut un bordereau par service avec en bas la somme que l'on place dans le sac attribué au service.

— Pour les employés payés directement sur leur compte chèque postal, il faut un *bordereau chèques postaux* (ou avis de virement) dont le principe est le même que pour les bordereaux banques mais où la forme change.

— Annuellement, il faut par matricule le *cumul*, montant total donné à chaque employé, ceci pour la comptabilité de l'entreprise et aussi pour la déclaration à qui vous savez.

L'analyste, à partir de ces différents documents, cités par l'organisation, doit les détailler complètement sur des papiers lignés (division en colonnes correspondant aux marteaux de l'imprimante, en lignes en respectant l'avancement du papier sur l'imprimante). Il doit dessiner complètement avec précision sur ces papiers lignés un modèle de chacun des documents demandés et les présenter aux services utilisateurs pour accord. A ce sujet, remarquons qu'un état dessiné (obtenu d'une autre manière que par l'ordinateur) peut

donner une fausse idée de ce que sortira l'imprimante en travail réel. La solution préconisée consiste à avoir un petit programme de cartes à imprimante, à faire perforer sur des cartes les modèles de ces états, à les imprimer à l'aide de notre petit programme, et à montrer le résultat à l'utilisateur. « Est-ce bien cela que vous désirez? » Moyennant quoi celui-ci voyant ce que rend l'imprimante aura une idée exacte de la chose et n'aura pas de surprise.

Parallèlement l'archivage, le classement des différents documents dans l'entreprise exige une certaine normalisation, standardisation. L'analyste doit faire approuver par l'organisation les différents modèles d'état qu'il propose. Pour la paye, il faut de plus respecter les lois.

1.2. — LE NŒUD D'ENTRÉE

Les différentes sorties étant fixées, l'analyste doit examiner le nœud d'entrée qu'il possède. Ce nœud d'entrée est constitué d'une part de fichiers déjà existants ou à créer, d'autre part des données nouvelles à traiter dans la chaîne. Ici que nous faut-il?

— Un fichier du personnel en séquence par numéro de matricule avec comme renseignements : le nom, le prénom, le numéro de service, la position dans l'entreprise, le salaire minimum garanti, le salaire brut mensuel, un code spécial indiquant s'il est payé banque, compte postal ou en espèce.

— Un fichier du personnel payé banque en séquence par numéro de matricule avec en face le numéro, le nom de la banque, le numéro de l'agence et le numéro de compte.

— Un fichier du personnel payé chèque postal en séquence par numéro de matricule avec en face le numéro de compte.

On suppose ici dans notre exemple que ces 3 fichiers existent déjà sur des cartes perforées.

— On possède aussi une table des primes et des retenues, également sur cartes perforées, en séquence par code prime avec en face de chaque code le libellé de la retenue ou de la prime.

— Enfin les données proprement dites constituées des cartes primes venant mensuellement du service du personnel. Ces cartes contiennent un numéro de matricule, un code prime et un montant positif ou négatif.

Voilà ce dont dispose l'analyste pour obtenir son nœud de sortie. Il doit d'abord se demander : Est-ce suffisant? Est-ce que les différents détails des états de sortie peuvent être obtenus à partir des fichiers et des données d'entrée? Après examen l'analyste doit pouvoir répondre par l'affirmative. Dans la négative il est en droit de réclamer auprès du service organisation.

L'analyste peut se poser alors une deuxième question : ce nœud d'entrée est-il améliorable? Cette amélioration peut dépendre de l'évolution du matériel. Si on dispose d'un ordinateur à bande magnétique les différents fichiers cartes vont être mis sur bande. On peut profiter de ce changement de support pour améliorer le dessin (le contenu) des enregistrements du fichier.

Mais attention ces modifications ne peuvent se faire qu'avec l'accord de l'organisation. Ce n'est pas par hasard que l'on rencontre ici plusieurs fois le terme « organisation ». La liaison entre l'organisation et l'analyse dans le traitement de l'information est la barrière la plus difficile à définir. En fait

si l'on sait parfaitement définir ce que c'est que *l'analyse :* « détermination des traitements machines entre un nœud d'entrée et un nœud de sortie », *l'organisation* apparaît floue. L'organisation dépend de la manière dont l'utilisateur utilise rationnellement ou non ce merveilleux outil qu'est l'ordinateur. Nous y reviendrons.

Dans l'organisation il doit y avoir un groupe responsable du catalogue des fichiers. Un fichier ne sert pas forcément à une seule chaîne de traitement (une seule application), une synchronisation entre les différentes applications est nécessaire. C'est pourquoi une modification de fichier ne peut avoir lieu qu'après le feu vert de l'organisation et sous le contrôle de l'organisation. C'est du reste dans ce catalogue des fichiers que l'analyste vient chercher les renseignements nécessaires à son nœud d'entrée.

Quelles sont les améliorations que l'analyste peut proposer dans notre exemple?

— N'oublions pas qu'il faut obtenir une statistique annuelle par matricule. Il est possible bien sûr de créer une bande tournante spécialement pour cette statistique, mais ne vaut-il pas mieux modifier le fichier du personnel en ajoutant au bout de chaque enregistrement un compteur contenant le montant progressif depuis le début de l'année?

— Ne peut-on pas non plus donner aux chèques postaux le code banque 98? (en supposant qu'il n'y a pas de code banque supérieur à 97). Nous pouvons ainsi fusionner le fichier banque avec le fichier chèques postaux.

— Enfin il est également possible de modifier les enregistrements du fichier du personnel, de manière à supprimer le fichier banques et le fichier chèques postaux, en y mettant les renseignements de ces 2 fichiers. Cela est possible car la séquence est la même et un numéro de matricule du fichier banques ou du fichier chèques postaux doit se trouver dans le fichier du personnel.

On a intérêt dans la mesure où cela est possible à diminuer au maximum le nombre des fichiers. Les pertes de temps dues au transport d'informations inutiles pour certains programmes sont négligeables devant les gains de temps dus à la simplification que cela entraîne. Il ne faut pas un fichier par programme, mais un même fichier pour différents travaux.

1.3. — LA MATÉRIALISATION DE LA CHAINE : L'ORGANIGRAMME

Remarque : Dans ce qui suit l'expression : « ordinateur périphérique » signifie que l'on peut traiter l'opération soit sur une unité non connectée à l'ordinateur soit sur l'ordinateur lui-même (dans la suite séquentielle des travaux ou bien en simultanéité avec d'autres travaux; voir multiprogrammation).

On suppose ici qu'on a pu réunir en un seul tous les fichiers du personnel, banques et chèques postaux et qu'on peut y mettre les montants progressifs pour la fin de l'année.

Nous avons maintenant les deux bouts de la chaîne, on va pouvoir aller de l'un à l'autre.

On se dit : « Le fichier du personnel est par matricule, les cartes sont en vrac, il nous faut l'état de paye par matricule... » et peu à peu on construit l'organigramme de la figure 5.1.

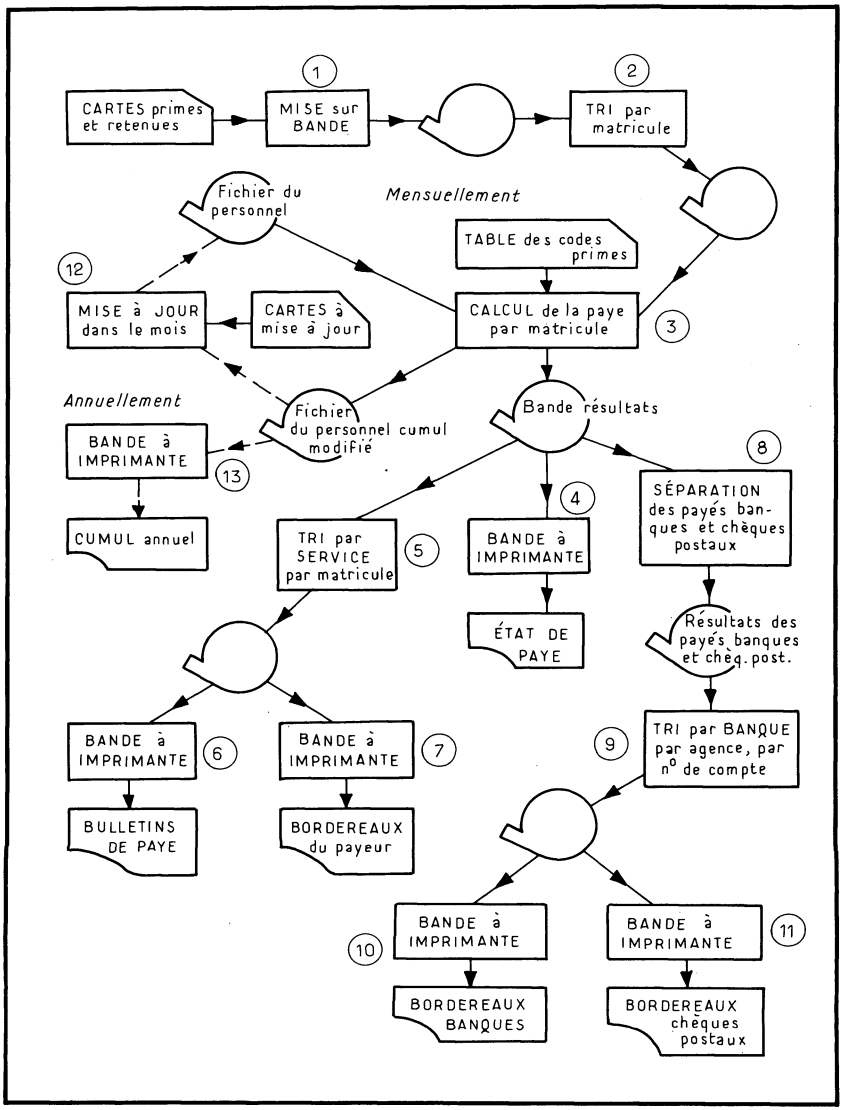


Fig. 5-1. — Premier brouillon de l'organigramme PAYE.

1re opération : Mise des cartes primes et retenues sur bande (sur un ordinateur périphérique).

2e opération : Tri par matricule.

3e opération : Calcul de la paye à partir du fichier du personnel et les données primes et retenues.

4e opération : Impression de l'état de paye : (ordinateur périphérique).

5e opération : Tri par service par matricule.

6e opération : Impression du bulletin de paye (ordinateur périphérique).

7e opération : Impression des bordereaux du payeur pour les payes en espèces (ordinateur périphérique).

8e opération : Séparation des résultats des payes banques et chèques postaux des autres résultats.

9e opération : Tri par banque par agence par numéro de compte.

10e opération : Impression des bordereaux banques (ordinateur périphérique).

11e opération : Impression des bordereaux chèques postaux (ordinateur périphérique).

Toutes les opérations précédentes se font mensuellement au moment « de la paye » (fin du mois).

Entre 2 passages de cette chaîne on effectue : l'opération 12 : mise à jour du fichier du personnel.

A la fin de l'année : l'opération 13 : impression des cumuls annuels (ordinateur périphérique).

L'analyste examine ce premier brouillon d'organigramme. Il doit se demander : est-il satisfaisant? est-il améliorable?

— Il ne faut pas oublier le traitement des anomalies (la bête noire de l'analyse). Il ne suffit pas de se dire « elles ne peuvent pas se produire » ou bien « elles se produisent exceptionnellement ». On doit les tester (détecter) toutes et prévoir un traitement pour chacune. Si on oublie d'en traiter une, elle risque de coûter très cher lorsqu'elle se présentera. (Elle nécessitera des dépouillements d'état, des analyses de fichier, des repassages en machine, enfin des bricolages devant les papiers et devant les ordinateurs, pour retrouver l'anomalie noyée dans la masse des informations, pour remettre le tout d'aplomb.)

Ici, dans la paye, ces anomalies sont d'autant plus importantes qu'on n'a pas le droit comme dans une gestion de stock de les corriger au cours du passage suivant. Il est nécessaire de corriger les anomalies avant même de calculer la paye. Citons les anomalies qui peuvent se présenter :

— Code prime inconnu.

— Numéro de matricule dans les cartes inconnu dans le fichier du personnel.

— Plus de cartes primes pour un même numéro de matricule qu'on ne le prévoit.

— Code service inconnu, inexistant.

— Dépassements de capacité, etc.

Le fait que nous sommes obligés de traiter ces anomalies avant le calcul proprement dit de la paye nous oblige à diviser la chaîne en deux parties : une chaîne de contrôle et une chaîne de calcul de la paye. Dans la chaîne de contrôle on imprime toutes les anomalies. Ces anomalies sont corrigées par le bureau de paye (délai de corrections une demi-journée). Les anomalies rectifiées sont réintroduites dans la paye au début de la deuxième chaîne.

On peut dans l'organigramme de la figure 5.1 supprimer l'opération 8. Il suffit de demander au programme de l'opération 3 de sortir directement la bande « Résultat des payes banques et chèques postaux ». Afin de diminuer les interventions des opérateurs et les fichiers intermédiaires (relecture d'un fichier que l'on vient d'écrire) on a intérêt à grouper le maximum de traitement dans un même programme. (Si la mémoire de l'ordinateur n'est pas assez grande, on peut utiliser des techniques de programmes segmentés ou à plusieurs phases et éventuellement des fichiers intermédiaires de travail.)

Les impressions des états de paye et des bulletins de paye sont des opérations assez longues. Si on dispose de 2 imprimantes, il est peut-être utile d'avoir à la sortie du programme « calcul de paye » 2 bandes résultats. Pendant que l'on commence à imprimer les états de paye avec une bande, on peut trier l'autre par service. Une fois ce tri terminé on pourra commencer l'impression des bulletins de paye sur la deuxième imprimante.

Après ces réflexions on aboutit à l'organigramme de la figure 5.2. L'analyste est maintenant prêt à commencer la rédaction de son dossier d'analyse.

Remarques :

— Notre organigramme peut également se décomposer en trois morceaux et il en est généralement ainsi avec tous les traitements de la gestion classique : un premier morceau pour le contrôle des informations d'entrée (il se trouve ici avant l'arrêt d'une demi-journée), un deuxième morceau pour les calculs proprement dits (ici les calculs sont tous rassemblés dans le programme « calcul de la paye »), et un troisième morceau constitué par les éditions.

— Notre exemple est une paye non intégrée. Dans la pratique une grande partie des informations d'entrée peuvent venir de la production (contrôle de production, suivi de chantier). A la sortie les comptabilités générales et analytiques ont besoin de certains résultats.

1.4. — LE DOSSIER D'ANALYSE

C'est le but final de toute analyse. Il doit contenir :

La présentation de la chaîne : son objet et son but, sa fréquence, le matériel utilisé. La position de cette chaîne par rapport aux autres chaînes de traitement de l'information. La présentation du nœud d'entrée (présentation logique sans entrer dans les détails). L'arrivée de ces informations d'entrée : Quoi? Où? Comment? Quand? Qui? Les services responsables de ces entrées. De la même manière la présentation du nœud de sortie. La ventilation des différents états : Où? Quand? Comment? Les temps de programmation, de mise au point, de démarrage, d'exécutions prévus par l'analyste.

Nous devons nous arrêter un moment pour signaler un détail qui a son importance. En examinant les délais d'arrivée des entrées, de départ des

sorties l'analyste doit examiner si le tout est compatible avec son temps d'exécution (avec une sécurité tenant compte des incidents possibles). Mieux, l'analyste doit se renseigner auprès du centre de traitement de l'information

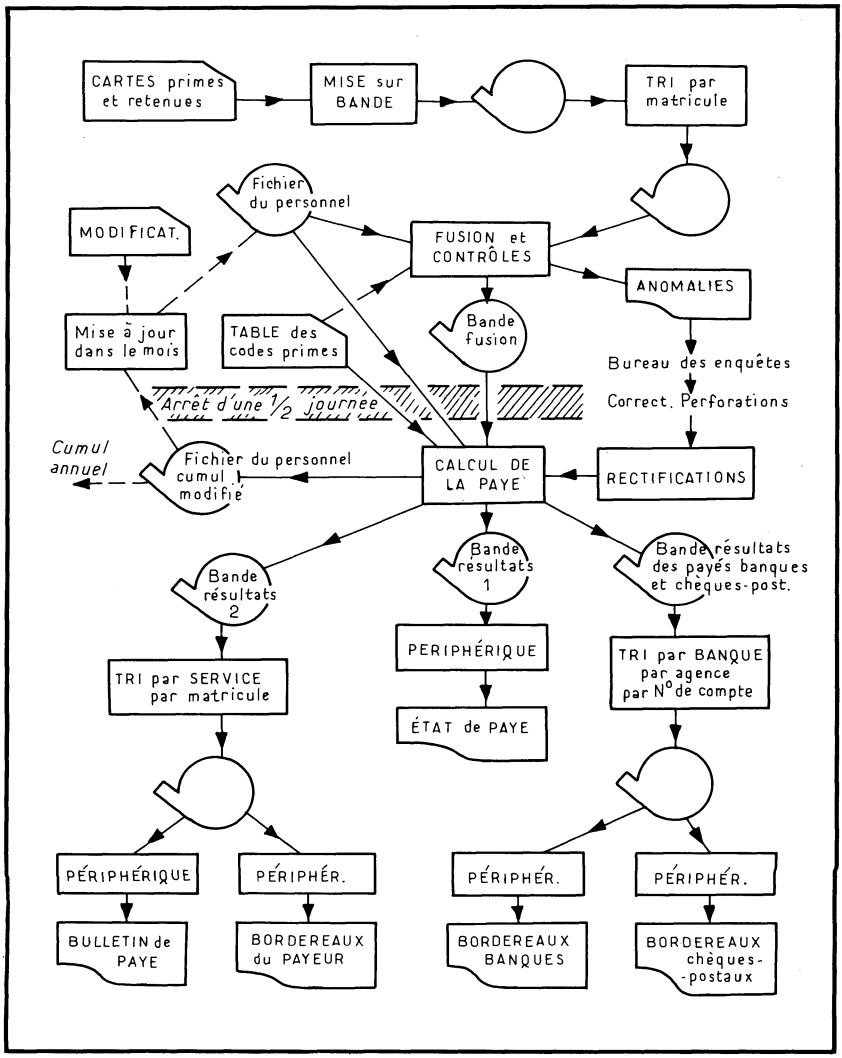


Fig. 5-2. — Organigramme de notre exemple PAYE.

pour savoir si l'atelier peut exécuter le travail. Cela ne concerne pas seulement le matériel mais aussi le planning. Il ne s'agit pas de programmer une chaîne demandant 6 heures de tel ordinateur tous les mardis alors que le planning de cet ordinateur, ce jour-là, est déjà saturé. Dans le cas où la réponse de l'atelier est négative, l'analyste doit poser le problème à l'organisation. Celle-ci pourra envisager soit un achat de nouveau matériel, soit une modification du planning pour les chaînes déjà existantes, soit enfin une modification des problèmes dans la chaîne en cours d'analyse.

L'organigramme vu précédemment. Cet organigramme doit avoir été contrôlé par l'organisation. Lorsque nous employons le mot « contrôle » il ne s'agit pas bien sûr « de contrôler le travail de l'analyste ». Ce n'est pas cela, mais nous estimons que l'informatique est un domaine complexe, toujours améliorable. Il est bon avant de mettre la chose en route, de faire jeter par un autre un coup d'œil sur son travail afin d'être sûr de n'avoir rien oublié.

Un sous-dossier par programme. Dans chaque sous-dossier on commencera par présenter le programme. Toujours les mêmes questions : Quoi? Quand? Où? Comment? Le type d'ordinateur utilisé, le langage de programmation utilisé. Ces choix du matériel et du langage doivent aussi être approuvés par l'organisation.

On mettra ensuite les dessins détaillés des entrées et des sorties pour le programme, rappeler d'où viennent les entrées et où vont les sorties.

Après l'analyste explique comment on passe des entrées aux sorties. Dans ses explications il doit préparer la *logique des blocs* de l'ordinogramme du programmeur. Il doit d'abord dire sommairement le fonctionnement du programme par des enchaînements de blocs. Ensuite détailler un peu plus les blocs en les divisant en sous-blocs, puis enfin arriver aux détails. Il ne faut pas oublier qu'un « beau baratin » pour être très clair doit s'accompagner d'un exemple. Si cet exemple est complet et bien conçu, il pourra servir de jeu d'essai au programmeur.

Ces sous-dossiers d'analyse seront remis aux différents programmeurs. Pour les programmes importants, il est possible de diviser les sous-dossiers en sous-sous-dossiers correspondant aux blocs de programme et de donner les blocs d'un même programme à des programmeurs différents.

Les programmeurs ajouteront dans les sous-dossiers leurs ordinogrammes, leurs listings de compilation et les listings des jeux d'essais et des résultats qui en découlent. L'ensemble de ces sous-dossiers à nouveau réunis avec l'organigramme et la présentation de la chaîne formera ce qu'on appelle *la brochure étude.* En fait toute étude terminée (analyse, programmation, mise au point, démarrage) doit donner lieu à 3 brochures :

— Une *brochure étude :* nécessaire aux analystes et aux programmeurs pour les modifications et les pannes éventuelles (c'est ce qu'on appelle la maintenance de l'étude).

— Une *brochure atelier :* nécessaire au centre de traitement de l'information pour effectuer le travail. On verra ce qu'elle doit contenir.

— Une *brochure de vulgarisation :* nécessaire pour les autres analystes et autres personnes désireuses de connaître rapidement ce qui se fait dans cette chaîne sans entrer dans les détails (il ne s'agit pas « d'en mettre plein la vue » mais de faire comprendre).

Remarque : La chaîne paye précédente peut sembler bien simple. En fait ce qui complique tout ce sont les cas particuliers. Effectuer 50 000 fois la même chose n'est pas plus difficile que de la faire 1 000 fois (temps d'exécution de l'ordinateur mis à part). Ce qui coûte cher c'est le traitement spécial qu'il faut faire une fois de temps en temps. Malheureusement les chaînes de traitement sont trop souvent infestées de cas particuliers.

2. — Les différentes organisations de fichier

Un problème important pour l'analyste est l'organisation de ses fichiers. On distingue pour les fichiers 4 types d'organisation : l'organisation séquentielle (seul ce type d'organisation est valable pour les cartes et les bandes magnétiques), l'organisation à accès direct, l'organisation séquentielle indexée et l'organisation cloisonnée.

2.1. — L'ORGANISATION SÉQUENTIELLE

Tous les problèmes que nous venons de traiter utilisent ce type d'organisation. Les supports auxquels on pense s'appellent la carte perforée et la bande magnétique. En fait avec l'apparition des ordinateurs évolués ou de grandes performances la technique des mémoires à accès sélectif à changé et le support disque, par exemple, est préférable à la bande, même pour le traitement en séquentiel. (Disons pour donner une idée de grandeur qu'actuellement le rapport des vitesses d'entrée-sortie entre la bande et le disque est de 1/3 à 1/10.)

Avec l'organisation séquentielle, l'analyste doit prendre les précautions suivantes :

En tête de chaque fichier un enregistrement spécial qu'on appelle *label de début* contient des renseignements sur les enregistrements qui suivent. En général on y trouve : le nom du fichier, son numéro dans l'entreprise (sa place par rapport aux autres fichiers); la date de création (il est possible par exemple de diviser les dates : en année et en n^{ieme} de jour dans l'année, ou bien de mettre en place un calendrier spécial dans l'entreprise en comptant un par un les jours ouvrables); le nombre de jours pendant lesquels on n'a pas le droit d'effacer le contenu du fichier (le droit de réécrire dessus); un numéro de séquence de volume (de la bande si le support est une bande magnétique) dans le cas où le fichier est obligé de tenir sur plusieurs volumes (plusieurs bandes).

Avant de lire ou d'écrire sur une bande magnétique (ou un fichier séquentiel sur mémoire à accès sélectif) le programmeur doit avant tout vérifier le contenu du label.

S'il s'agit de lire, on doit au moins vérifier le nom du fichier (ou le numéro de série), la date de création, et le numéro de séquence de volume. Cette vérification se fait en comparant le contenu du label de début avec ce que désire le programme. Si le label n'est pas « bon » un message doit prévenir le pupitreur afin de lui dire qu'il s'est trompé de volume et lui offrir la possibilité d'en monter un autre.

S'il s'agit d'écrire, on doit (fig. 5.3) lire le label qui se trouve au début du volume et vérifier que la somme : « nombre de jours de conservation » plus « date de création » est inférieure ou égale à « date du jour ». Si cette condition n'est pas satisfaite un message doit prévenir le pupitreur pour lui dire qu'il doit monter un autre volume. Sinon il faut repositionner le volume à son point de départ, écrire le nouveau label, ensuite on peut commencer le travail. Cette procédure exige d'écrire un label sur les nouveaux volumes, dès leur arrivée dans l'entreprise.

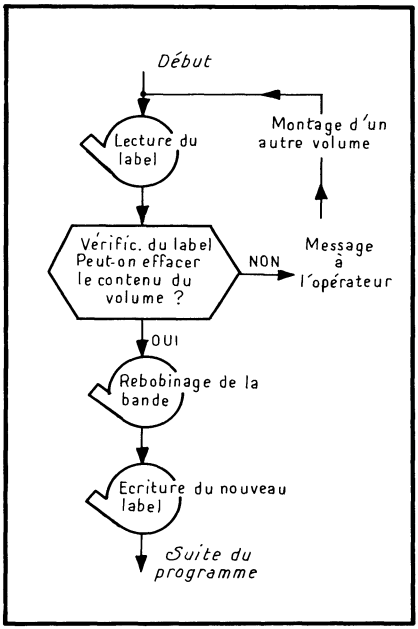


Fig. 5-3. — Vérification du Label d'un fichier.

D'autre part lorsqu'on écrit sur une bande, il est prudent de compter les enregistrements. On place le total qui en résulte, à la fin du fichier dans un enregistrement spécial qu'on appelle le *label de fin*. Lorsqu'on relie le fichier on compte le nombre de ces enregistrements et lorsqu'on arrive à la fin on compare le total que l'on trouve avec le total indiqué dans le label. Si le résultat de la comparaison est différent un message sort au pupitre afin de prévenir le centre de traitement qu'il y a eu une anomalie dans les entrées-sorties.

Généralement ces procédures sont programmées une fois pour toutes dans le software (gestion des données), mais l'analyste doit indiquer le contenu de ces labels.

L'analyste doit aussi fournir à l'atelier un nombre de jours de conservation pour ces bandes fichiers. Il faut prévoir la reconstitution d'une bande fichier en cas de destruction de cette bande. Pour cela deux solutions :

— dupliquer (reproduire) cette bande;

— ou bien conserver la bande venant de la mise à jour précédente (on l'appelle bande-mère) avec les cartes de mise à jour (pour éventuellement recréer la bande-fille).

2.2. — L'ORGANISATION A ACCÈS DIRECT

La mémoire à accès sélectif est divisée : en cylindres et pistes pour les disques, en cellules et feuillets pour les mémoires à cellules... Chaque piste a une adresse. Si on donne à l'unité d'entrée-sortie cette adresse elle peut lire ou écrire presque immédiatement (temps d'accès chiffrable en millisecondes) sur cette piste.

On dit qu'un fichier est organisé en accès direct lorsqu'il existe un sous-programme (ou bloc de programme) permettant à partir d'un indicatif donné (un numéro d'article pour un fichier stock) de donner l'endroit (numéro de cylindre, numéro de piste pour un disque) où doit se trouver l'enregistrement correspondant sur la mémoire à accès sélectif (s'il y est déjà placé).

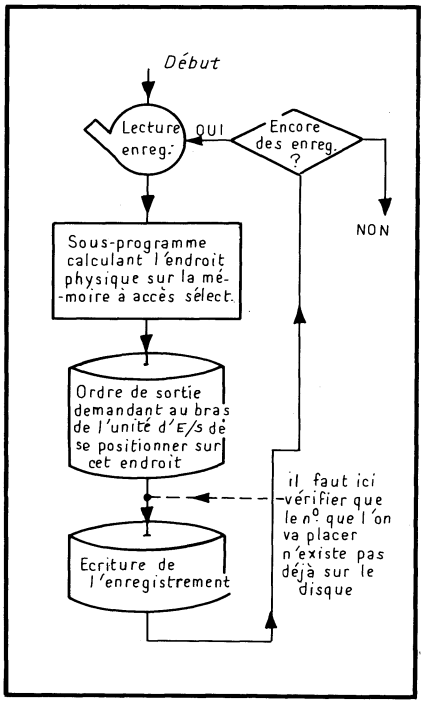


Fig. 5-4. — Mise en place d'un fichier en organisation à accès direct.

Toute la difficulté de cette organisation consiste pour l'analyste à déterminer ce sous-programme. Il faut pour un numéro d'article *donné un seul* numéro géographique sur la mémoire à accès sélectif; pour un numéro géographique un nombre à peu près constant de numéro d'article. Il est possible d'analyser ce sous-programme lorsque la nomenclature est conçue par les responsables de l'informatique. Cela est pratiquement toujours impossible lorsqu'il s'agit d'une nomenclature déjà existante. (Malgré les nombreuses recettes de cuisine : adressage calculé, adressage parallèle, recouvrement, liens de chaînage pour les synonymes, préconisées partout; recettes dont le verbiage est bien plus impressionnant que l'efficacité).

L'Analyse du sous-programme de calcul pour l'accès direct n'est possible que lorsque l'on connaît d'une manière précise les règles qui permettent ou qui ont permis la création de la nomenclature.

Avant de mettre les enregistrements du fichier en place il est nécessaire de préformer les pistes par des enregistrements fictifs. En effet les enregistrements ne vont pas être placés à la suite les uns des autres sur les pistes, il est nécessaire de réserver de la place pour les enregistrements qui vont suivre. C'est le phénomène du parking des voitures. Si on ne trace pas la place de chaque voiture par rapport aux voisines, les premières voitures se mettront là où elles le veulent et il risque de n'y avoir plus de place pour celles qui viendront plus tard. C'est pourquoi on commence toujours par écrire des enregistrements fictifs (ne contenant par exemple que des blancs) de longueur égale à ceux que l'on veut écrire, en séquence sur toutes les pistes que va occuper notre fichier.

Ensuite un programme dont l'ordinogramme est celui de la figure 5.4 met en place le fichier. Les enregistrements sur la bande magnétique, à l'entrée de ce programme peuvent être triés ou non. Pour chaque enregistrement lu on calcule l'endroit où il faut le mettre sur le disque et on y place directement cet enregistrement. (En réalité il faut prendre en plus la précaution de regarder si le numéro d'indicatif de l'enregistrement que l'on va placer sur le disque n'existe pas déjà sur celui-ci. C'est le cas où par erreur 2 enregistrements de même numéro figurent sur la bande. Il faut alors signaler l'erreur et ne placer qu'un de ces enregistrements).

Recherche d'un enregistrement sur la mémoire à accès sélectif

Une fois le fichier en place il est possible de trouver les renseignements correspondant à un article, toujours à l'aide du même sous-programme de calcul de lieu géographique. Ce sous-programme positionne le bras de l'unité d'entrée-sortie sur la piste qui doit contenir le numéro demandé. Il suffit ensuite de lancer un ordre de lecture.

Mise à jour d'un fichier à accès direct

Pour mettre à jour un numéro donné sur un fichier à accès sélectif, on commence par aller le rechercher (comme expliqué précédemment). S'il se trouve sur le fichier on le met à jour dans la mémoire principale, puis on le réécrit sur la mémoire à accès sélectif à la même place. S'il ne s'y trouve pas on dit qu'il s'agit d'une mise à jour en addition, c'est un nouveau numéro que l'on place dans le fichier. La figure 5.5 contient l'ordinogramme commun à la recherche et à la mise à jour.

Les précautions indispensables sur les mémoires à accès sélectif

Il est nécessaire sur ce type de mémoire (plus encore qu'avec les bandes, car on peut en modifier le contenu plus aisément : lire puis écrire au même endroit) de pouvoir reconstituer un fichier en cas de fausse manœuvre. La

méthode la plus classique consiste à dupliquer le contenu de ces mémoires de temps en temps, la fréquence dépendant de la densité d'utilisation. On conserve aussi une trace des modifications : additions et mises à jour.

Avantages et inconvénients de l'organisation à accès direct

L'accès à un enregistrement est très rapide, pour ainsi dire immédiat. Par contre, la mémoire à accès sélectif n'est pas occupée à plein, on peut être satisfait lorsque l'occupation de cette mémoire est de 60 %.

Donc avec l'accès direct : accès immédiat, mais perte de place (sans compter la quasi-impossibilité de trouver le sous-programme liant le numéro d'indicatif à l'adresse physique sur le disque lorsqu'on n'est pas maître de la nomenclature). Avantage et inconvénient à l'opposé de l'organisation séquentielle où le volume est occupé à 100 % mais où le temps d'accès à un enregistrement est très long. D'où l'idée d'un 3e type d'organisation, combinaison de ces deux organisations conservant les avantages de l'une et de l'autre.

2.3. — *L'ORGANISATION SÉQUENTIELLE INDEXÉE*

Pour expliquer le principe de cette organisation nous prendrons des disques, mais le fonctionnement est évidemment le même pour toutes les mémoires à accès sélectif.

Mise en place du fichier

Les enregistrements du fichier sont d'abord triés par numéro indicatif puis placés sur les disques, cylindre par cylindre, et à l'intérieur de chaque cylindre piste par piste, comme pour l'organisation séquentielle (une adresse disque se compose d'un numéro de cylindre suivi d'un numéro de piste). Seulement lorsqu'on arrive à la fin d'une piste on conserve en mémoire le dernier numéro-indicatif que l'on a écrit et le numéro de cette piste. Et on procède ainsi pour toutes les pistes. Lorsqu'on arrive à la fin du cylindre on possède en mémoire une table contenant toutes les adresses des pistes de ce cylindre, avec pour chacune d'elles, le plus grand numéro indicatif que contient la piste. Cette table s'appelle « *l'index piste* » du cylindre. On place cet index piste à un endroit bien déterminé de la pile de disque. (En général on le met au début de la première piste du cylindre considéré; et on a pris la précaution de réserver de la place lorsqu'on a écrit les enregistrements).

Avant de changer de cylindre, on conserve en mémoire le dernier numéro indicatif de l'enregistrement que l'on vient d'écrire, avec à côté le numéro de ce cylindre. Et on procède ainsi à la fin de tous les cylindres. Lorsque le fichier est complètement chargé on possède en mémoire une table où l'on a tous les numéros de cylindre occupés par le fichier, avec pour chacun de ces cylindres le plus grand numéro indicatif qui s'y trouve. On appelle cette table « *l'index cylindre* ». On écrit cet index à un endroit connu sur la pile de disque.

Lorsque cet index cylindre est trop grand parce que le fichier est lui-même très important, il est possible de le diviser en plusieurs morceaux. On garde

alors en mémoire le plus grand numéro indicatif connu par chacun des index cylindres, avec à côté le numéro de cylindre, numéro de piste où l'on a mis l'index cylindre correspondant. On constitue ainsi un *index fichier*.

Enfin, pour les fichiers encore plus monstrueux, il est possible de décomposer à nouveau, suivant le même principe, les index fichiers pour obtenir un super-index fichier et ainsi de suite. Ce qui compte c'est que l'index de plus haut niveau soit placé à un endroit connu sur la pile de disque.

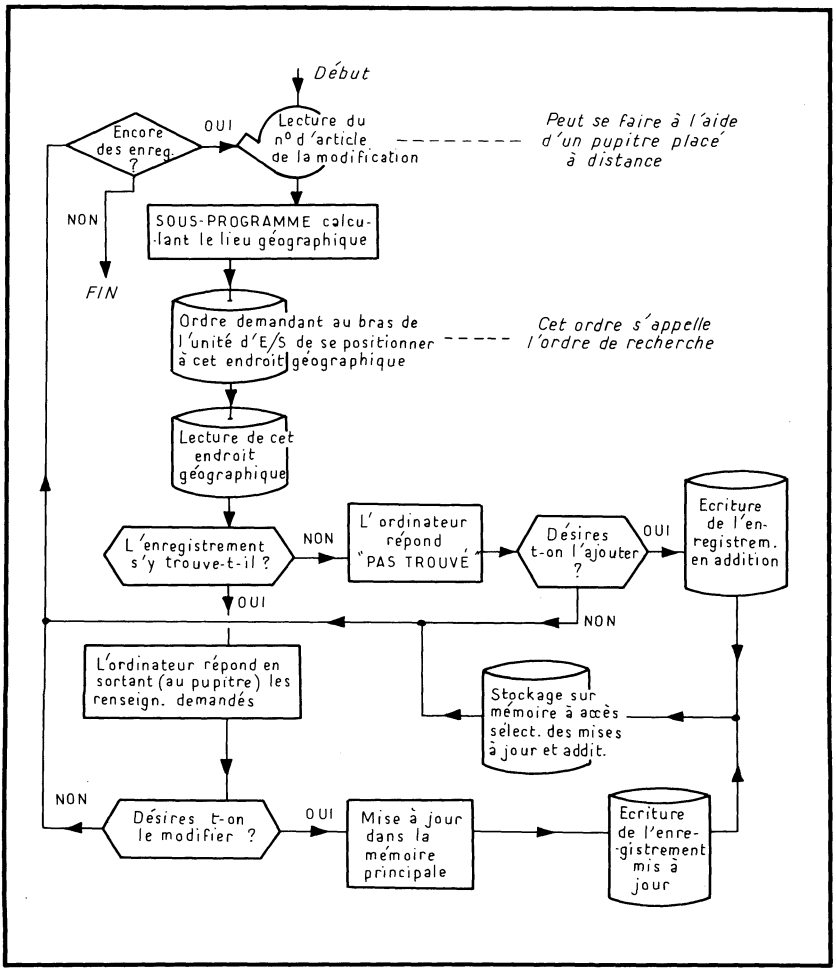


Fig. 5-5. — Recherche et mise à jour d'un fichier en accès direct.

Recherche d'un enregistrement

Avec ce type d'organisation on peut de la même façon qu'avec l'organisation séquentielle rechercher un enregistrement puisque les enregistrements sont placés en séquence. Il suffit de lire le fichier à partir de son début, cylindre par cylindre, à l'intérieur de chaque cylindre, piste par piste, jusqu'à ce que l'on tombe (ou que l'on dépasse) le numéro indicatif que l'on cherche.

On peut aussi, comme avec l'organisation à accès direct, se positionner directement sur un enregistrement. Il suffit de lire l'index de plus haut niveau, (car on sait par définition où se trouve cet index) en le consultant il va nous dire quel index de niveau immédiatement inférieur il faut lire et ainsi de suite... Par exemple, si notre fichier comporte trois niveaux d'index : index fichier, index cylindre et index piste; en consultant l'index fichier celui-ci va nous dire où se trouve l'index cylindre qu'il faut lire. La lecture de cet index cylindre puis sa consultation vont nous donner l'endroit où se trouve l'index piste que l'on doit examiner. Ce dernier index nous donne l'adresse de la piste qui doit contenir notre enregistrement.

Autrement dit, nous avons avec ce type d'organisation le même ordinogramme qu'avec l'organisation à accès direct. Le bloc « sous programme calculant l'endroit physique de la mémoire à accès sélectif à partir de l'indicatif » se décompose en lecture de table sur le disque, consultation dans cette table pour lecture de la suivante et ainsi de suite jusqu'à l'index piste qui nous donne le résultat.

Mise à jour d'un fichier en séquentiel indexé

Avec ce que nous savons déjà du séquentiel indexé nous pouvons après avoir trouvé un enregistrement (soit par recherche séquentielle, soit par recherche au hasard) le mettre à jour en le modifiant. On peut aussi enlever (supprimer) un numéro indicatif (par exemple un numéro d'article périmé) en lui mettant un code spécial à une position bien déterminée de son enregistrement.

Bien plus délicat est le problème de la mise à jour pour les nouveaux indicatifs. Il y a des additions à faire entre des enregistrements existants et cela ne peut pas se faire sur les mêmes pistes, car le fichier a été chargé séquentiellement, il n'y a pas de place pour les nouveaux enregistrements. Il faut trouver une astuce.

En principe, on dispose de deux solutions :

— *La première* consiste à se réserver des pistes supplémentaires que l'on appelle *pistes d'addition.* Les nouveaux enregistrements peuvent être mis sur ces pistes, seulement n'oublions pas que le fichier doit parfois être lu séquentiellement. Si un enregistrement doit être mis sur une piste d'addition, il est nécessaire qu'il y ait un lien entre l'enregistrement qui le précède en séquence et lui-même, un lien entre lui-même et l'enregistrement qui le suit. Généralement, cette addition se fait de la manière suivante : on place dans la table appelée index piste des renseignements supplémentaires. Après chacun des numéros de piste on indique bien sûr le plus grand numéro indicatif sur la piste; mais on réserve aussi de la place pour un 2^{e} numéro indicatif et pour un autre numéro qu'on appelle lien.

Un poste index piste a donc la configuration de la figure 5.6.

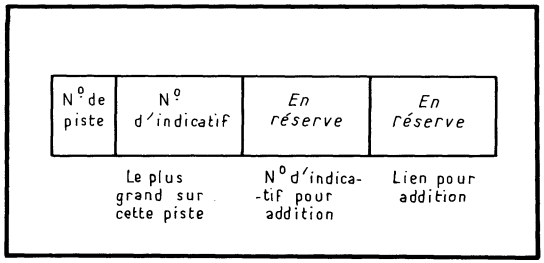


Fig. 5-6. — Configuration d'un poste index piste.

Prenons comme exemple la piste n° 2480 contenant les numéres indicatifs : 57, 64, 78, 87, 95, 98 (fig. 5.7).

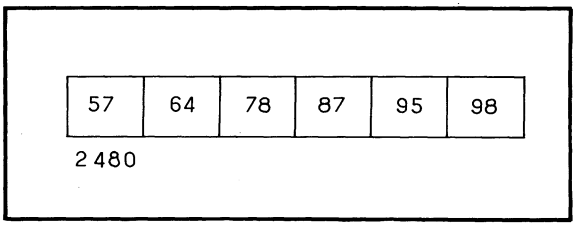


Fig. 5-7. — La piste 2 480 contient au départ 6 articles.

Au départ (après chargement du fichier) le poste correspondant à cette piste sur l'index piste a donc le contenu de la figure 5.8.

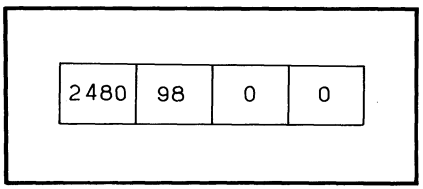


Fig. 5-8. — Contenu initial du poste index de la piste 2 480

On désire maintenant ajouter l'enregistrement ayant pour indicatif 67. On va consulter l'index fichier, puis l'index cylindre qui en découle et de la même manière l'index piste. Celui-ci va nous dire : « l'enregistrement 67 s'il existe doit se trouver sur la piste 2480 ». Nous lisons donc cette piste et nous constatons que l'indicatif 67 n'existe pas encore dans le fichier. C'est donc bien

une addition qu'il faut faire. Dans la mémoire principale on place l'enregistrement 67 entre 64 et 78 en reculant d'un cran les enregistrements 78, 87, 95 et 98. (Comme pour tout ce qui est informatique nous donnons ici une solution, ce n'est pas la seule.)

Et nous réécrivons (fig. 5.9) sur la piste les enregistrements 57, 64, 67, 78, 87 et 95. Il faut écrire l'enregistrement 98 sur la piste d'addition attribuée

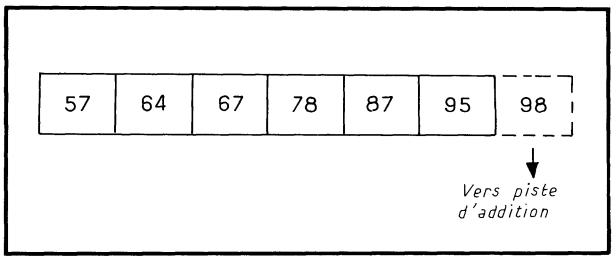


Fig. 5-9. — Contenu de la piste 2 490 après l'addition de l'enregistrement 67.

au cylindre. Supposons que cette piste d'addition porte comme adresse le nº 2572 et qu'elle contienne déjà 2 enregistrements. On va placer l'enregistrement 98 sur cette piste, à la suite de ces enregistrements, en 3e position. On modifie en même temps le poste de la piste 2480 dans l'index piste (fig. 5.10). L'indicatif le plus élevé sur cette piste est maintenant 95, on place ensuite sur ce poste 98 et 3 (place de cet enregistrement sur la piste d'addition).

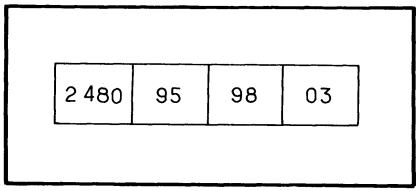


Fig. 5-10. — Contenu du poste index de la piste après l'addition de l'enregistrement 67.

De cette manière lorsqu'on voudra rechercher un enregistrement compris entre 96 et 98 (96, 97 ou 98), l'index piste nous dira qu'il faut aller voir le 3e enregistrement sur la piste d'addition du cylindre.

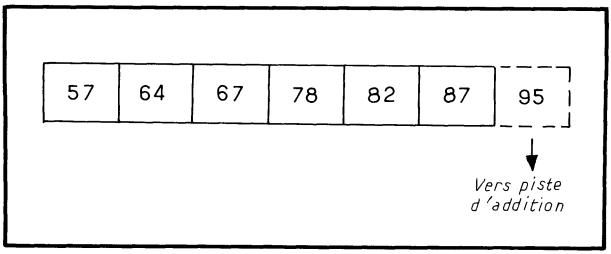


Fig. 5-11. — Contenu de la piste 2 480 après l'addition de l'enregistrement 82.

Supposons que l'on désire ensuite ajouter l'enregistrement ayant pour indicatif 82. Toujours par le même procédé nous allons aboutir à la piste 2480. On va lire cette piste et constater que cet indicatif n'existe pas sur la piste, donc à nouveau une addition. Dans la mémoire principale on place l'enregistrement 82 après l'enregistrement 78 et on décale d'un cran les enregistrements 87 et 95 (fig. 5.11).

On écrit sur la piste 2480 les enregistrements ayant pour indicatif 57, 64, 67, 78, 82 et 87. Il faut placer l'enregistrement 95 sur la piste d'addition attribuée au cylindre au premier endroit de libre. C'est-à-dire ici en 4e position. Mais il ne faut pas oublier que les enregistrements doivent pouvoir être lus en séquence : 95 après 87, 98 après 95, après 98 on doit revenir à la piste 2481 qui suit la piste 2480. Ceci nous amène à introduire les *liens de chaînage* que doivent contenir les enregistrements sur les pistes d'addition. Par exemple, l'enregistrement 95 a un lien de chaînage contenant 03. Cela signifie que l'enregistrement à lire après 95 se trouve en 3e position sur la piste d'addition. (En l'occurence c'est l'enregistrement 98 que nous avons écrit précédemment.) L'enregistrement 98 lui n'a pas besoin de lien de chaînage (ou il a un lien de chaînage contenant zéro) puisqu'il est le dernier maillon de la chaîne pour la piste.

Parallèlement, on modifie le poste de la piste 2480 dans l'index piste. Le numéro le plus élevé sur cette piste est actuellement 87, l'enregistrement à prendre en considération après ce numéro se trouve en 4e position sur la piste d'addition. Cela est indiqué comme dans la figure 5.12.

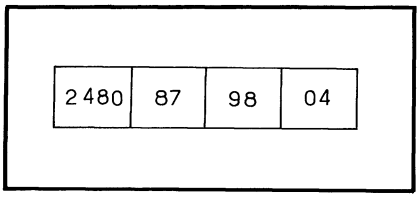


Fig. 5-12. — Contenu du poste index de la piste après l'addition de l'enregistrement 82.

Après le numéro de piste on trouve : l'indicatif le plus élevé sur la piste (87), l'indicatif le plus élevé pour cette piste sur le poste d'addition (98), l'emplacement de l'indicatif le moins élevé en addition pour cette piste (le premier à prendre en considération après 87, ici en 4e position).

Complétons notre gymnastique en voyant ce qui se passe si maintenant on ajoute l'indicatif 96. Notre série de consultations de table par niveaux d'index décroissants va nous amener dans l'index piste de notre cylindre sur le poste 2480. On s'aperçoit que 96 est compris entre 87 et 98. Par conséquent, il faut lire la piste d'addition. On vérifie bien entendu que ce numéro n'existe pas déjà et que c'est bien une addition. On va écrire l'enregistrement ayant pour indicatif 96 au premier emplacement de libre sur la piste d'addition, c'est-à-dire ici la 5e. On lui met comme lien de chaînage 03 car l'enregistrement (98) à prendre en considération après lui se trouve en 3e position. On modifie en même temps le lien de chaînage (95) en lui mettant 05 (après (95) on a maintenant (96)).

En résumé, le lien entre ces différents enregistrements est réalisé de la manière indiquée par la figure 5.13.

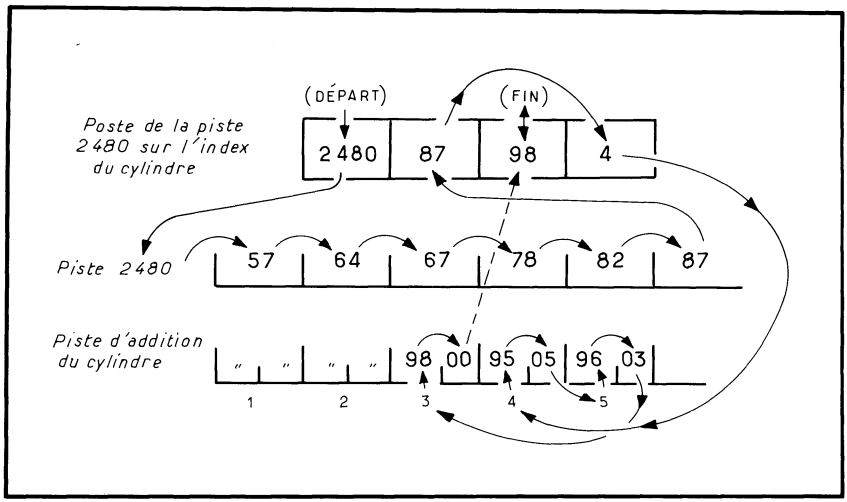


Fig. 5-13. — Chaînage séquentiel entre les enregistrements à partir d'un poste index d'une piste.

Remarques :

Lorsqu'une piste d'addition devient pleine il est nécessaire de reconstituer le fichier en nettoyant les pistes d'addition. Cela peut se faire aisément à l'aide d'un programme « disque à disque » lisant séquentiellement le fichier et le recopiant sur un autre volume, mais en mettant les enregistrements en addition à leurs places logiques.

Il est possible d'avoir plusieurs pistes d'addition pour un même cylindre, il faut alors prévoir en plus un numéro de piste dans les liens de chaînage, mais il ne faut pas non plus avoir trop d'enregistrements en addition, car les recherches risquent d'être plus longues.

— *La deuxième solution* est plus simple mais n'offre pas la souplesse de la première. Elle consiste à conserver de la place à la fin de chaque piste lors de la création du fichier sur disque. (Par exemple en mettant des *enregistrements* avec code de suppression à la fin de chaque piste). Les enregistrements sont mis à leur place logique. Lorsqu'une piste est pleine on peut effectuer une recréation du fichier par un programme « disque à disque » séquentiel, ou bien alors retomber sur la première solution avec des pistes d'addition.

L'avantage de cette méthode par rapport à la première est que les recherches sont moins longues; par contre, on risque d'avoir à recréer plus souvent le fichier dans le cas où les enregistrements vont toujours s'ajouter sur les mêmes pistes.

Conclusion sur l'organisation séquentielle indexée

Ce type d'organisation de fichier possède à la fois les avantages du séquentiel et de l'accès direct. En plus, il est *valable dans tous les cas*, il ne nécessite pas comme l'accès direct une étude préalable pour déterminer la relation entre le numéro indicatif et l'emplacement géographique. La mise en place de cette méthode (les différents niveaux d'index, les additions...) peut sembler longue, en fait ces différentes opérations sont toujours les mêmes et peuvent avoir été prévues dans le software.

L'accès direct et le hardware sur les mémoires à accès sélectif

L'accès direct (au hasard) exige pour chaque recherche (positionnement du bras sur la piste d'un disque, mise d'un feuillet magnétique sur une tête de lecture...) un temps relativement assez long. Il en est de même pour les différentes consultations de table de l'index séquentiel. Ensuite, il y a la recherche de l'enregistrement sur la piste même...

Une solution consiste à livrer aux registres du canal de l'unité d'entrée-sortie un véritable programme canal où les ordres se suivent (comme les instructions d'un programme) dans la mémoire principale. Certains de ces ordres peuvent être logiques, c'est-à-dire se débrancher ou non à un autre point du programme canal en fonction du résultat de l'ordre précédent. Le canal n'interrompt la mémoire principale que pendant le court temps nécessaire à la lecture de l'ordre suivant.

D'autre part, les enregistrements sur disque peuvent être divisés en morceaux. Il peut y avoir pour chaque enregistrement : 3 sous-enregistrements (fig. 5.14).

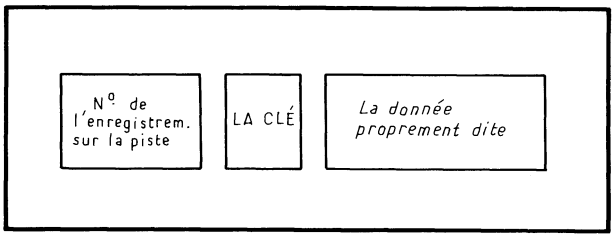


Fig. 5-14. — Les trois morceaux d'un enregistrement sur disque magnétique.

Un premier sous-enregistrement nécessaire pour l'unité de contrôle contient le numéro (géographique) de l'enregistrement sur la piste. Un deuxième contient ce qu'on appelle *la clé*, c'est-à-dire l'indicatif de l'enregistrement s'il n'y a qu'un enregistrement logique dans l'enregistrement physique, ou bien l'indicatif le plus élevé dans l'enregistrement s'il y a plusieurs enregistrements logiques dans l'enregistrement physique (groupage). Le troisième sous-enregistrement contient l'enregistrement proprement dit, on l'appelle *la donnée*.

Un registre du canal contient l'indicatif (la clé) de l'enregistrement que l'on recherche. L'unité de contrôle a la possibilité de comparer la clé venant du canal avec les clés des différents enregistrements de la piste, et suivant le résultat de ces comparaisons de lire ou de ne pas lire la donnée.

Il faut bien voir la synchronisation entre la chose électronique et la chose mécanique que cela exige. Cela explique la complexité des unités de contrôle.

Après avoir positionné le bras du disque sur le cylindre demandé par le programme canal, l'unité de contrôle signale au canal « ordre exécuté ». Celui-ci lui donne le numéro de piste à prendre en considération sur le cylindre et la clé de l'enregistrement à chercher. L'unité de contrôle commande à l'unité de disque l'ordre de lecture de la piste. Mais la lecture d'une piste exige un certain nombre de millisecondes. Il ne s'agit pas de lire la piste, comparer la 1re clé, relire la piste, comparer la 2e clé, relire la piste et ainsi de suite jusqu'à ce qu'on trouve la bonne clé. On lit la première clé sur le disque et l'unité de contrôle compare la clé qui vient d'être lue avec la clé qui vient du canal pendant que l'unité de disque lit l'entre enregistrement placé après la clé. Suivant le résultat de cette comparaison la donnée sera lue ou non (ou bien écrite ou non).

Voilà pourquoi un entre-enregistrement (un espace) est nécessaire entre la clé et la donnée. (Dans les entre-enregistrements se trouvent aussi des informations (bits) pour les contrôles et la synchronisation.) Il en résulte pour les analystes que la capacité d'une mémoire à accès sélectif dépend du nombre de ces enregistrements, c'est-à-dire de la longueur des enregistrements (le groupage diminue le nombre des entre-enregistrements).

2.4. — L'ORGANISATION CLOISONNÉE

Généralement, on ne met pas un fichier sur un volume de mémoire à accès sélectif, mais plusieurs (la capacité de ces mémoires étant très grande). Il est donc nécessaire de pouvoir se positionner sur le fichier que l'on désire traiter. Pour cela, chaque volume de mémoire à accès sélectif comporte sur la 1re piste de son premier cylindre ce que l'on appelle un *label Volume*. Ce label contient différents renseignements pour ce volume et il contient en particulier une table, *un répertoire*, qui contient les noms des différents fichiers avec les adresses des labels de ces fichiers. En consultant cette table, on peut ensuite se positionner sur le label d'un fichier et le lire. Le label du fichier contient les différents renseignements propres à ce fichier : type d'organisation, forme des enregistrements, date de création, nombre de jours de conservation, adresse de l'index de plus haut niveau sur la mémoire à accès sélectif si l'organisation est séquentielle indexée, adresse du premier enregistrement à lire si l'organisation est séquentielle...

Mais parfois le nombre de fichiers dans une mémoire à accès *sélectif peut* être tellement important, que le répertoire du volume au lieu d'indiquer des adresses de label indique des adresses de sous-répertoire. On donne alors aux fichiers ce qu'on appelle des noms composés. C'est-à-dire que si un sous-répertoire s'appelle VERT tous les fichiers qui s'y rapportent s'appellent VERT DE... Enfin, il est possible d'avoir des sous sous-répertoires, on obtient une structure arborescente, comme dans la figure 5.15.

Pour se positionner sur le fichier « VERT DE FEUILLE DE LILAS », le répertoire du volume nous donne l'adresse du sous-répertoire VERT, ce dernier nous

transmet l'adresse du sous-sous-répertoire FEUILLE. Dans ce sous-sous-répertoire on trouve l'adresse du label du fichier que l'on cherche.

Cette structure arborescente est importante, car elle permet dans un software moderne de rechercher directement un fichier sur un volume donné en partant d'un répertoire général englobant tous les volumes montés sur l'ordinateur.

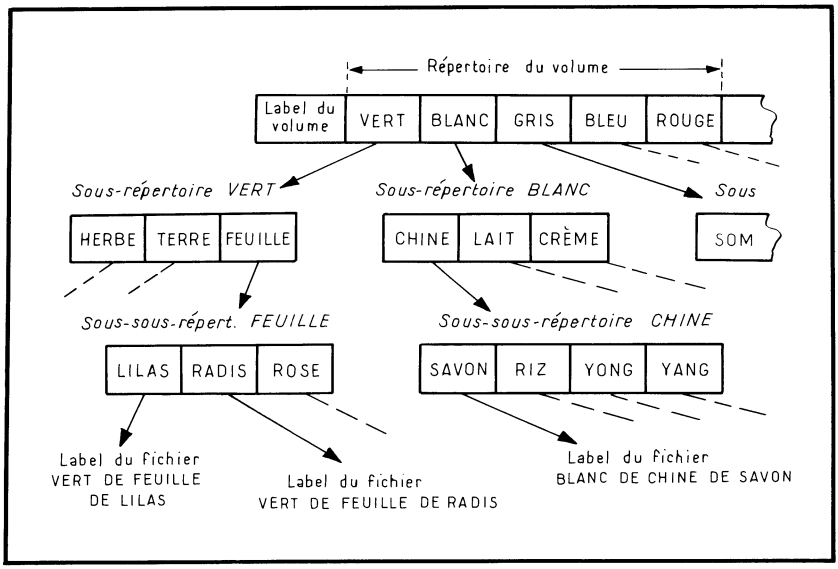


Fig. 5-15. — La structure arborescente de l'organisation cloisonnée.

Enfin, à l'intérieur même d'un fichier on peut décomposer en morceau. On dit alors qu'il a l'organisation cloisonnée. Cela est couramment utilisé lorsque le fichier est une bibliothèque de programmes. Cela peut être aussi le cas pour un fichier stock où les différentes pièces sont classées par famille de pièces, et où l'on désire pouvoir ne traiter qu'une seule famille de ces pièces. Il s'agit de se positionner au début d'un programme donné ou d'une famille de pièces donnée, et de traiter les enregistrements qui viennent ensuite en séquence. Dans ce cas le fichier comporte lui aussi à côté de son label un répertoire indiquant les différents noms des programmes (ou des familles de pièces) avec pour chacun l'adresse sur la mémoire à accès sélectif du premier enregistrement de ce programme (ou de cette famille).

On peut alors se placer sur ce premier enregistrement et traiter « *le membre* » de fichier ainsi sélectionné. Parfois le fichier peut lui aussi comporter des sous-répertoires...

Attention, en cas d'addition, de suppression ou de mise à jour d'un membre, il faut modifier les répertoires en conséquence.

3. — Évolution de l'analyse

Sur les premiers ordinateurs on pouvait résumer les différents organigrammes de gestion de la manière suivante : (comme dans la fig. 5.16) un

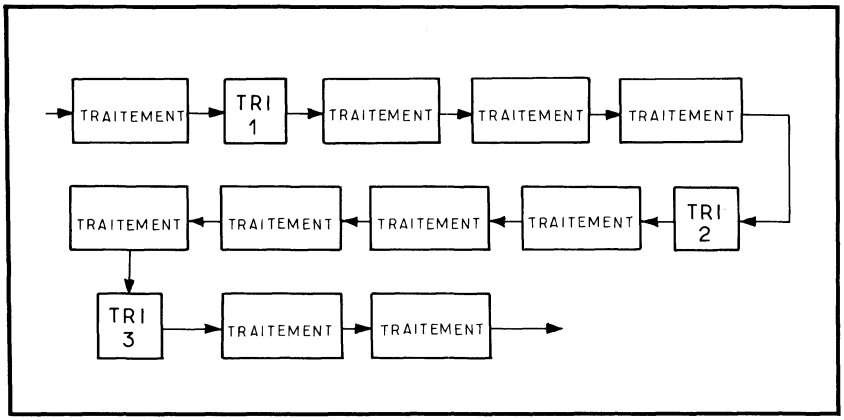


Fig. 5-16. — Organigramme de gestion sur les premiers ordinateurs.

traitement, un tri, plusieurs traitements, un tri, plusieurs traitements... (en séparant les tris des autres traitements).

Peu à peu les mémoires principales se sont agrandies, les analystes ont mûri et les différents traitements entre 2 tris consécutifs ne sont plus devenus qu'un seul traitement. Les organigrammes se résument alors à : un traitement, un tri, un traitement, un tri, un traitement... (fig. 5.17).

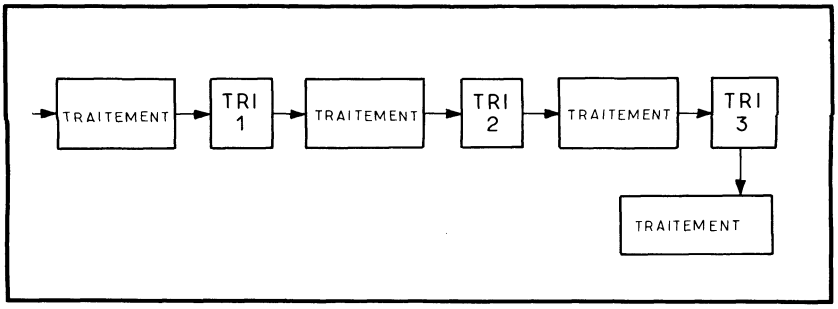


Fig. 5-17. — Organigramme de gestion classique sur les ordinateurs actuels.

Afin de minimiser le temps machine, on doit en effet dans un même programme effectuer le maximum de choses. Avec les mémoires à accès sélectif et les différentes organisations qu'elles apportent (en particulier le merveilleux séquentiel indexé) la moitié des tris ne deviennent plus nécessaires, les seuls conservés sont ceux exigés par les sorties. Il s'ensuit que le schéma précédent est conservé mais avec deux fois moins de tris, donc avec des traitements entre chaque tri 2 fois plus gros. Par suite, les programmes deviennent de plus en plus énormes, et la logique des blocs dans l'ordinogramme s'impose de plus en plus.

Parallèlement, la technique des mémoires à accès sélectif, avec en plus la possibilité qu'offre le hardware d'interrompre à distance le travail en cours dans la mémoire principale afin d'interroger et éventuellement de modifier le contenu d'un enregistrement d'un fichier, permet ce qu'on appelle *le traitement de l'information en temps réel.*

Le traitement de l'information classique consiste à recueillir à l'entrée une certaine quantité d'informations, et une fois seulement que cette quantité est assez importante d'effectuer le traitement.

Le traitement en temps réel, lui, traite les informations au fur et à mesure qu'elles arrivent. Afin d'expliquer le fonctionnement de ce type de traitement, il nous est nécessaire de faire un voyage à travers le software.

Nous pouvons résumer les tendances de l'analyse actuelle à l'aide des 6 points :

1) abandon des bandes magnétiques pour les mémoires à accès sélectif. Celles-ci sont plus rentables pour l'organisation des fichiers en séquentiel. Elles permettent en outre l'accès direct à un enregistrement grâce, en particulier, au séquentiel indexé valable dans tous les cas. Toutefois les bandes restent plus économiques pour le stockage des informations;

2) effectuer des programmes de plus en plus volumineux. On a intérêt à effectuer le maximum de calculs dans un même programme.

3) nécessité absolue de diviser les programmes en blocs;

4) utilisation des langages évolués. On n'utilise les autocodeurs que pour des sous-programmes spéciaux;

5) traitement de l'information en temps réel actuellement en plein développement.

6) intégration des chaînes du traitement de l'information dans un ensemble, dans un « système » intimement lié à l'organisation de l'entreprise.

CHAPITRE VI

LE SOFTWARE

Le software est tout ce qui se trouve entre le hardware et les utilisateurs, c'est une définition floue et vaste. Le software couvre l'ensemble des programmes « utilitaires » : compilateurs, programme de tris, analyses de fichiers. Il couvre aussi ce qu'on appelle le système d'exploitation qui contient : la gestion des « travaux », la gestion des données, la gestion des « tâches », et la possibilité de structurer les programmes.

Le software et le hardware forment un tout : l'ordinateur. Le hardware est ce qui est câblé, il doit être conçu en fonction du software, de même ce dernier doit utiliser toutes les possibilités du premier. Aussi il ne faut pas s'étonner si, lorsqu'on parle de l'évolution du software, on est obligé de parler en même temps de l'évolution du hardware.

Afin d'expliquer ce qu'est le software, il est nécessaire d'en faire l'historique, d'examiner comment il a évolué de la 1re à la 3e génération d'ordinateur, et comment peu à peu il s'est imposé comme une partie de l'ordinateur au moins aussi importante que le hardware.

1. — La préhistoire du software

A son début l'ordinateur digital — qu'il soit à mots ou à caractères — n'avait pas de software ou presque rien. Il fallait envoyer les programmeurs chez les constructeurs suivre des cours fort longs où on leur apprenait le langage absolu de la machine, les formats des instructions, les codes opérations (ou à la rigueur des codes équivalents un peu plus mnémotechniques), les fonctionnements des différents registres.

Enfin dans le cas où la machine était assez compliquée, il était nécessaire d'avoir des élèves habitués aux disciplines de la science mathématique. L'ordinateur ne pouvait fonctionner que chez des utilisateurs ayant un personnel hautement qualifié.

Les instructions absolues étaient écrites sur des bordereaux de perforation. Il fallait compter le nombre de positions prises par chacune de ces instructions pour pouvoir d'une part se transférer à un point donné du programme, se reporter à une constante ou une variable, modifier une instruction par une autre. Le software se limitait à l'existence d'un petit programme appelé « chargeur » permettant de placer en mémoire, à l'endroit voulu, les différentes instructions et zones du programmeur. Ce chargeur était lui-même amené en mémoire à l'aide d'un dispositif spécial du hardware appelé « dispositif de chargement ». Les premières instructions : « l'amorce » sont mises en mémoire par ce dispositif, après quoi cette amorce amène en mémoire le reste des instructions du chargeur.

Peu à peu se sont ajoutés quelques *programmes utilitaires*. Un programme utilitaire est un programme qui peut être utilisé par tout le monde, il n'est pas spécifique d'une application, il a un intérêt général.

Les programmes utilitaires après le chargeur ont été des programmes de tri, des programmes de vidage mémoire (permettant de sortir le contenu de la mémoire principale à un moment donné sur l'imprimante), des programmes d'analyse bande et disque (permettant de sortir le contenu d'une bande ou d'un disque).

Peu après est arrivé le premier compilateur de langage autocodeur de base; puis un deuxième compilateur autocodeur avec cette fois une bibliothèque de macro-instructions (sur cartes puis sur bande). On a introduit dans ces macro-instructions la gestion des entrées-sorties pour les fichiers organisés séquentiellement. C'est le fameux « IOCS » (Input Output Control System) groupant des routines standard nécessaires aux entrées-sorties.

1.1. — L'IOCS

Les contrôles des labels de début et de fin de bande sont toujours les mêmes à la condition de standardiser le dessin (format) de ces labels. Il en est de même pour les routines d'erreur des entrées-sorties. Par exemple, relire n fois un enregistrement lu d'une façon erronée, commander un dépoussiérage sur le lecteur de bande, à nouveau p fois la relecture avant de déterminer que l'enregistrement est effectivement illisible. Laisser alors la possibilité à l'opérateur de corriger l'enregistrement, de le placer sur une bande des illisibles et de continuer le traitement en l'ignorant, ou bien d'essayer à nouveau des tentatives de relecture.

Une routine standard peut être aussi écrite pour les sorties. Une routine standard également pour ce qu'on appelle le *flip flop de bande :* pour les fichiers volumineux tenant sur plusieurs volumes il est nécessaire à la fin de chaque volume de contrôler le label de fin, de donner l'ordre à l'opérateur de monter le volume suivant si ce volume n'est pas déjà monté, de contrôler le label de ce nouveau volume, et enfin si ce contrôle est bon de poursuivre le traitement. Il est possible de monter les volumes d'un même fichier sur des unités d'entrées-sorties différentes. Cela peut même se faire d'une manière cyclique. C'est-à-

dire le premier volume sur le lecteur A, le deuxième volume sur le lecteur B, lorsque le traitement du premier volume est terminé l'IOCS passe automatique sur le B, pendant ce temps on monte le 3e volume sur le lecteur A, plus tard le quatrième volume sera mis sur B et ainsi de suite. On peut de la même façon avoir des cycles ABCABC ou ABCDABCD. La routine spéciale de l'IOCS permet à chaque fois de changer les ordres d'entrée-sortie en y mettant le numéro de lecteur nécessaire.

Une routine de l'IOCS permet, en lecture après avoir détecté le dernier label de fin du fichier de se transférer à l'adresse du programme utilisateur correspondant à : « il n'y a plus d'enregistrement à lire ».

Enfin, et la plus importante de toutes, une routine dégroupe les enregistrements groupés, c'est-à-dire les enregistrements physiques (blocs) contenant plusieurs enregistrements logiques (fiches). Pour le programmeur qui lance sa macro-instruction de lecture ou d'écriture tout se passe comme s'il n'avait accès qu'à l'enregistrement logique. (Il raisonne avec les bandes comme avec les cartes.) On dit couramment que l'IOCS est divisé en 2 étages, « l'IOCS physique » qui s'occupe des entrées-sorties, et l' « IOCS logique » qui s'occupe de la liaison des entrées-sorties avec le programme du programmeur.

1.2. — LE DÉGROUPAGE

Ouvrons ici une parenthèse pour expliquer comment peut s'opérer *ce dégroupage :*

— A l'aide d'une zone de travail (comme dans la figure 6.1).

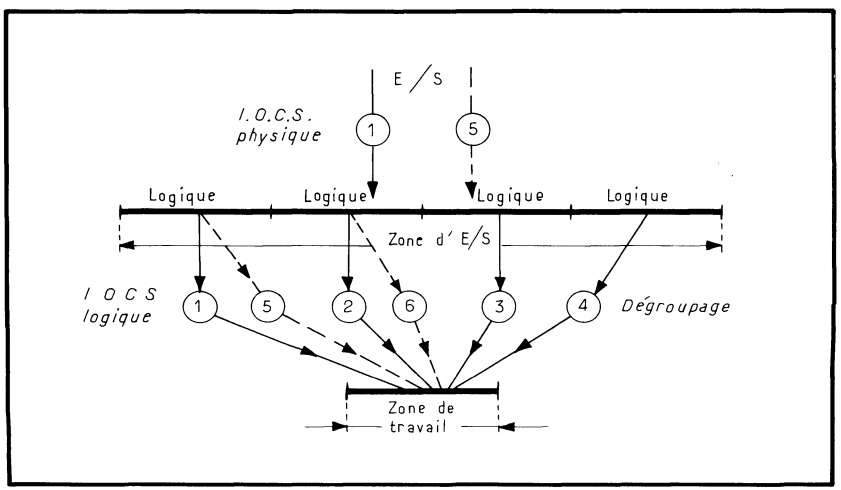


Fig. 6-1. — Mise des enregistrements logiques un par un dans une zone de travail.

Au premier ordre d'E/S logique de lecture du programmeur, l'ordre physique est exécuté. C'est-à-dire que la zone d'E/S est remplie par le premier enregistrement venant de la bande ou du disque. Puis l'IOCS place le premier enregistrement logique de ce bloc dans la zone de travail. Le programmeur peut alors travailler avec cette zone.

Au deuxième ordre d'E/S logique de lecture, le 2e enregistrement logique du bloc est mis par l'IOCS dans la zone de travail, au troisième ordre ce sera le troisième, et ainsi de suite jusqu'à ce que l'on arrive à la fin du bloc. A ce moment-là, l'IOCS physique entre à nouveau en jeu. La zone d'E/S est remplie par le 2e bloc de la bande ou du disque, puis l'IOCS logique recommence au début de la zone d'entrée-sortée, etc.

— A l'aide de l'indexation.

Le principe est le même que précédemment, seulement il n'y a plus de zone de travail. L'adresse de la zone d'E/S est indexée. L'IOCS logique au lieu de transporter les enregistrements logiques dans une autre zone, ajoute à chaque fois dans l'index la longueur de l'enregistrement logique. Au départ, et après chaque ordre physique, il est positionné au début de la zone d'E/S. Le programmeur travaille ainsi directement sur cette zone.

— A l'aide de techniques cycliques (parfois complexes).

Plus récemment, avec des hardwares permettant de diviser en plusieurs morceaux les zones d'E/S, on a mis au point des techniques permettant de minimiser au maximum les mouvements d'information à l'intérieur de la mémoire. Par exemple dans le cas d'un programme effectuant une entrée et une sortie ce sont les mêmes zones qui servent pour l'entrée et pour la sortie. L'IOCS fait passer successivement ces zones par les états : en lecture (entrée), en attente de calcul, en calcul, en attente d'écriture (sortie), en écriture, en attente de lecture, en lecture, etc. (Il modifie pour cela les adresses des zones d'entrée-sortie qu'il transmet aux unités par l'intermédiaire du canal.)

1.3. — LA TECHNIQUE DE LA VERRUE

Cette époque a été l'âge d'or du programmeur. Après avoir écrit son programme il va lui-même le compiler, le corriger, l'essayer sur l'ordinateur. Il se paye le luxe de le modifier directement dans la machine à l'aide du pupitre. Il utilise la *technique de la verrue* pour ajouter des instructions dans le langage absolu.

Soit, figure 6.2, les instructions A, B, C, D, E, elles se suivent en séquence dans le programme. On désire ajouter les deux instructions W, Z entre les instructions C et D.

Le programmeur remplace en absolu l'instruction C par une instruction de transfert vers une position de la mémoire principale inoccupée. On place à partir de cette position de mémoire les instructions C, W et Z. On termine par un transfert à la position de mémoire contenant l'instruction D inchangée du programme.

Après avoir passé un temps parfois assez long à écrire son programme, le programmeur a ainsi la possibilité de venir faire marcher ce qu'il a fait, de mettre en pratique ce qu'il a placé sur papier, de jouir, tel un pilote dans son avion, du frottement du vent sur les voiles, de pouvoir travailler par les

saisons chaudes ou froides dans une salle climatisée. Il a l'avantage d'être semi-bureaucrate et semi-opérationnel. Cette époque est hélas révolue.

1.4. — LA SIMULTANÉITÉ

Il y a eu l'avènement de la simultanéité dans le hardware, le software a dû se concevoir en conséquence. Il y a ce qu'on appelle le *flip flop de zones* dans la mémoire principale. Pour un même fichier, on réserve non pas une zone d'entrée-sortie mais plusieurs. Ces zones d'entrée-sortie sont utilisées d'une manière cyclique. Si par exemple on a les 3 zones A — B — C, on effectue l'ordre d'E/S sur A, puis sur B, puis sur C, à nouveau sur A, B, C... Pendant que l'entrée-sortie s'exécute sur une des zones, le programme a tout le loisir de travailler sur les 2 autres zones. Le software se charge de la synchronisation entre ces zones d'entrée-sortie de manière à ce que jamais l'ordre d'E/S et le programme n'utilisent la même zone.

Afin de bien voir l'avantage de la simultanéité prenons un programme ayant un fichier en entrée et un fichier en sortie. On suppose d'abord que le temps nécessaire à la lecture d'un enregistrement physique en entrée est égal au temps nécessaire à l'écriture d'un enregistrement physique en sortie; enfin que ces deux temps sont égaux au temps du calcul du programme compris entre une entrée et une sortie physiques consécutives.

En l'absence de simultanéité le déroulement dans le temps des opérations est une entrée E 1, un calcul C 1, une sortie S 1, une entrée E 2, un calcul C 2 et ainsi de suite, comme dans la figure 6.3.

Avec la simultanéité il est possible d'effectuer en même temps une entrée, un calcul et une sortie. Après avoir donc exécuté E 1 il est possible d'effectuer E 2 en même temps que C 1, puis E 3 en même temps que C 2 et S 1... C'est ce que montre la figure 6.4.

On va donc dans ce cas 3 fois plus vite. Pratiquement lorsque ces 3 temps ne sont pas égaux on raisonne avec le plus long de ces temps (en fait les temps de calcul sont négligeables devant les temps d'E/S et cette différence est appelée à s'accentuer sans cesse, c'est pourquoi on expliquera un procédé beaucoup plus efficace au chapitre VII.)

1.5. — LA PRIORITÉ DES UNITÉS D'E/S

Afin que la simultanéité fonctionne il est nécessaire dès qu'une entrée-sortie est libre, de lui redonner du travail si on peut le faire. Cela a amené à introduire dans le hardware le dispositif *d'interruption de fin d'entrée-sortie.* Dès qu'une entrée-sortie a terminé le travail qu'elle a à faire, elle *envoie à* l'unité centrale, par l'intermédiaire de l'unité de contrôle, puis du canal un signal : « travail terminé ».

Cela signifie que l'unité centrale doit vérifier que l'ordre d'entrée-sortie s'est exécuté correctement et examiner s'il n'y a pas en réserve dans la mémoire principale un autre travail à donner à cette unité d'E/S. Le hardware déclenche ce qu'on appelle une interruption. Il y a interruption du travail en cours, stockage des différents registres et indicateurs de la machine (l'ensemble de ces registres et de ces indicateurs forme ce qu'on appelle l'état machine),

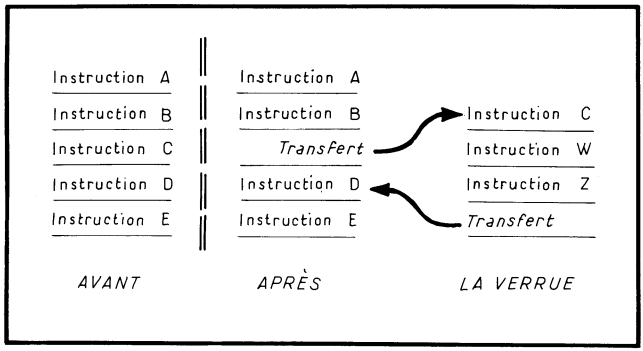


Fig. 6-2. — La verrue permet d'ajouter les instructions W et Z entre les instructions C et D.

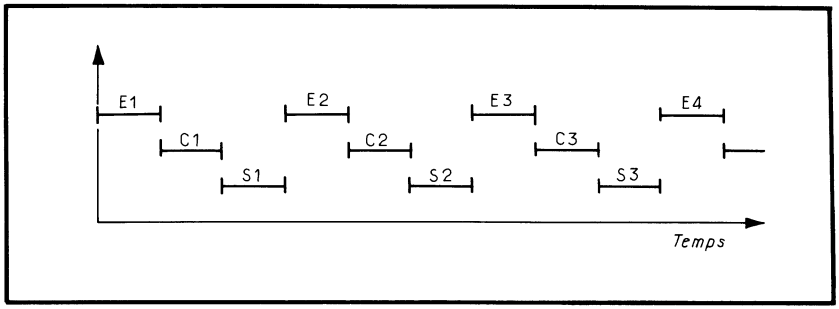


Fig. 6-3. — Déroulement du programme sans aucune simultanéité.

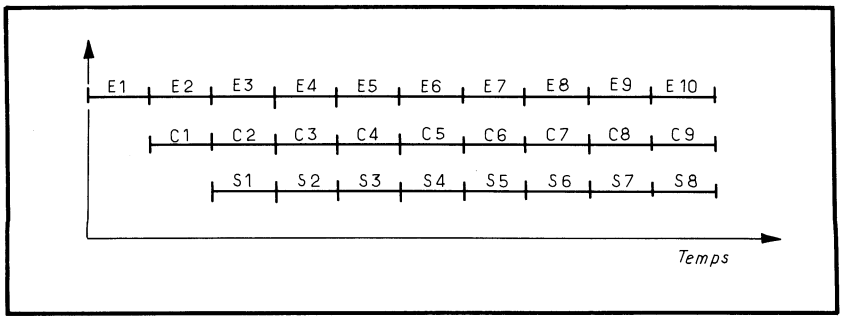


Fig. 6-4. — Déroulement du programme avec simultanéité entre les entrées, les calculs et les sorties.

et transfert automatique à un point bien défini de la mémoire principale. A ce point bien défini le software a placé des routines en permanence dans la machine. Ce sont des routines de l'IOCS pour contrôler l'ordre d'E/S qui vient de s'exécuter, puis des routines de file d'attente pour donner un autre travail à l'unité d'E/S.

Intervient alors ce qu'on appelle : la priorité des unités d'E/S pour un même canal. Lorsqu'un canal ne peut travailler qu'avec une seule unité d'E/S à la fois et que l'on a des ordres à donner aux différentes unités de ce canal, il faut choisir l'unité d'E/S que l'on doit faire travailler avant les autres. Le programmeur a la possibilité de dire dans ses macro-instructions de l'IOCS : « je désire que telle unité passe avant telle autre ». Autrement dit, en même temps qu'il place ses fichiers sur telle unité, tel canal, il déclare les priorités des E/S d'un même canal les unes par rapport aux autres. Il y a là une anomalie (une erreur) car un programmeur ne connaissant pas suffisamment le fonctionnement du hardware peut en se trompant dans les priorités ralentir le traitement de l'ordinateur plutôt que l'accélérer.

Lorsque le software a terminé ce qu'il avait à faire il restaure l'état machine et on revient dans le travail en cours, là où on l'avait laissé.

Nous répétons : il faut bien voir qu'avec ce type de simultanéité unitaire (pour un programme) c'est le plus long des 3 temps : entrée, calcul, sortie qui freine les trois autres. Si par exemple les entrées sont plus longues que les temps de calcul, il faut que les calculs attendent la libération des zones d'entrée, même si on utilise le flip flop de zones. Cette libération doit être contrôlée par l'IOCS, afin de ne pas lire et calculer sur la même zone en même temps.

Il y a là déjà la conception d'un software possédant une gestion des données (IOCS) assez complexe.

1.6. — LA TECHNIQUE DU POINT DE CONTROLE (OU POINT DE REPRISE)

Pour les travaux très longs (supérieurs à la demi-heure) il est nécessaire en cas d'incident (panne machine, coupure de courant, incident sur une unité d'E/S...) de ne pas avoir tout le travail à refaire. La solution consiste à écrire (toutes les demi-heures) l'image de la mémoire principale, et l'état de la machine sur une bande ou sur un disque. On appelle cela l'écriture d'un point de contrôle. On peut toujours repartir au dernier point de contrôle (perte de temps jamais supérieure à la demi-heure) à l'aide d'un programme utilitaire. Il suffit de mettre en place les différents volumes utilisés par le programme au moment du point de contrôle, puis de charger le programme utilitaire. Ce dernier remet en place la mémoire principale et l'état de la machine. Dans les routines de l'IOCS il trouve les compteurs de blocs (nécessaires aux label de fin). Il peut ainsi en comptant les enregistrements positionner convenablement les volumes séquentiels. (S'il y a des cartes, le programme aura dû noter à quelle carte on s'était arrêté).

1.7. — LA NAISSANCE DU CHAINAGE

Parallèlement la vitesse des machines s'est augmentée et on a pu faire, en suivant le travail d'un ordinateur pendant une journée, une constatation capitale :

Le matin il est nécessaire de le mettre en route, de lui apporter et de mettre en place différentes bandes, cartes, papiers pour l'imprimante. Il faut exécuter certaines manipulations, tout cela avant que le premier travail commence.

Une fois que ce programme est terminé, il faut déblayer l'ordinateur des différents fichiers qui sont montés sur lui, les cartes, les bandes et les états et il faut en monter d'autres pour le travail suivant. Et la journée se passe ainsi, les travaux étant séparés entre eux par des temps morts, « entre-travaux ». Supposons (supposition simpliste mais pédagogique) que tous les temps de travaux sont égaux et que les entre-travaux durent deux fois moins de temps que les travaux, le planning d'une journée d'ordinateur peut être schématisé comme dans la figure 6.5.

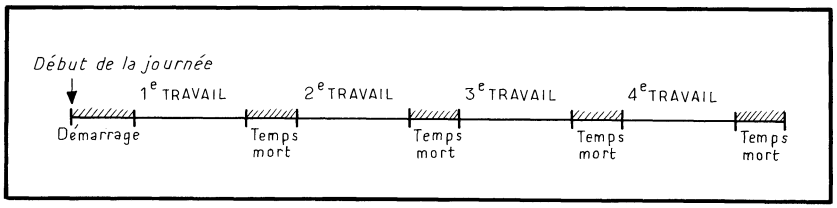


Fig. 6-5. — Planning d'une journée d'ordinateur.

Le rendement de l'ordinateur dans ce cas est de 2/3.

Remplaçons maintenant cet ordinateur par un autre travaillant 2 fois plus vite, les travaux vont durer 2 fois moins longtemps. Par contre, la vitesse de l'homme, elle, ne change pas. Depuis Astérix les Français se conduisent toujours comme des Gaulois, et les « entre-travaux » prennent toujours le même temps. Le schéma précédant devient celui de la figure 6.6.

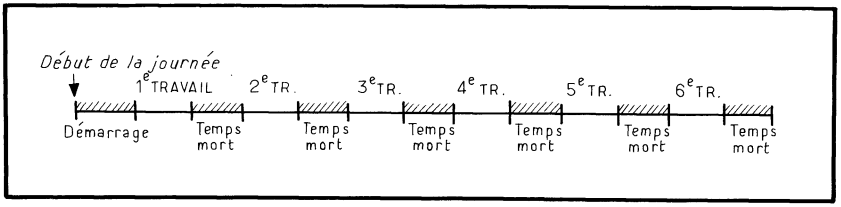


Fig. 6-6. — Planning d'une journée d'un ordinateur deux fois plus rapide que le précédent.

Le rendement n'est plus que de 1/2, on ne travaille pas 2 fois plus vite. Il faut bien se rendre à cette évidence :

— si les temps dans les mémoires, unité centrale, s'expriment en micro-secondes, voire en nanosecondes;

— si les temps des unités d'entrée-sortie s'expriment en millisecondes;

— les temps de l'homme s'expriment *en minutes*.

Il a donc fallu trouver quelque chose pour supprimer ces temps morts. C'est-à-dire préparer le 2e travail pendant que l'on exécute le premier, le 3e pendant que l'on exécute le 2e, et ainsi de suite. Il faut à la fin de chaque travail passer automatiquement au traitement suivant (chaînage des travaux). On a ainsi conçu le système d'exploitation séquentiel.

2. — Le système d'exploitation séquentiel

2.1. — PRÉSENTATION

Ce système ajoute à la gestion des données, déjà connue, la gestion des travaux. Tout tourne autour *d'un moniteur* (appelé aussi parfois superviseur). Il se décompose en deux parties : le moniteur permanent, toujours dans la mémoire, et le moniteur de transition qui ne vient en mémoire qu'entre les différents travaux. (Le moniteur est donc un ensemble de sous-programmes, de « routines ».)

Avec ce moniteur, on trouve, 2e pièce maîtresse de l'œuvre, *une bibliothèque de programmes* (sur bande ou sur disque) contenant après le programme amorce (nécessaire au démarrage du système au début de la journée), le moniteur permanent, le moniteur de transition et les différents programmes utilitaires : compilateurs, programmes de tris, vidage de la mémoire, programme-trace permettant de suivre le déroulement d'un programme, programmes d'analyse bande et disque... A la suite de ces programmes utilitaires, l'utilisateur a la possibilité de placer des programmes à lui (écrits par lui).

Parfois, on ne place pas les programmes du client sur la même bande, ou sur le même disque (on dit sur le même « volume ») que les programmes fournis par le constructeur (software). Dans une entreprise l'ensemble des programmes est tel qu'ils ne tiennent pas tous sur un même volume, d'autre part il n'est pas nécessaire qu'ils soient montés en permanence sur l'ordinateur. (Moins il y a de programmes en place et plus le temps de recherche pour un programme est court). C'est pourquoi avec ce type de système d'exploitation on choisit le plus souvent un volume pour les programmes du système. On l'appelle : *volume système* (bande système si c'est une bande). On prend des volumes différents pour les différentes applications, en groupant ensemble les applications appelées à être traitées à la suite les unes des autres. On a ainsi des *volumes applications* (bandes applications).

Enfin pour compléter le système, la 3e pièce maîtresse de l'œuvre, est ce qu'on appelle les *cartes de commande*. Ces cartes contiennent les différents ordres que le système doit exécuter séquentiellement.

L'ensemble des cartes de commande nécessaires à l'ensemble des travaux de la journée constitue ce qu'on appelle « le batch ». Ce « batch » est divisé en « JOB » : petit paquet de cartes nécessaire pour un travail. Dans ces différents paquets, placés à la suite les uns des autres, il y a toutes les commandes à l'opérateur et au moniteur. On y trouve le nom du programme à exécuter et sur quel volume application il faut aller le chercher. Des cartes commentaires dont le libellé sort au pupitre, disant il faut monter tel papier sur l'imprimante, telle bande de travail, telle bande fichier, enfin il y a les *fameuses cartes* d'assignation des unités d'entrée-sortie.

2.2. — ASSIGNATION DES UNITÉS D'ENTRÉE-SORTIE

Lorsque l'on exécute un travail, il est nécessaire de préparer le travail suivant, cela est possible à la condition que ce dernier n'utilise pas les unités d'E/S occupées actuellement. Le choix des unités d'E/S ne peut donc pas se

faire au niveau du programme. En effet, suivant le planning de l'atelier un programme ne passe pas forcément toujours après le même autre.

Le programme ne donne plus des numéros de canaux, numéros d'unités de contrôle, numéro d'unités d'entrée-sortie. Il donne des noms symboliques. Le compilateur accepte ces noms symboliques et les laisse tels qu'ils sont. Chacun des noms symboliques sera remplacé par un nom absolu (canal, unité de contrôle, unité d'E/S) seulement juste avant le chargement du programme.

La chose se passe de la manière suivante. Le travail en cours se termine, il faut passer au travail suivant. Le moniteur permanent charge en mémoire principale le moniteur de transition (à la place du programme qui vient de s'achever). Le moniteur permanent contient la table de tous les noms physiques des unités d'E/S. Le moniteur de transition contient tous les noms symboliques que le programmeur a le droit d'utiliser pour désigner ses entrées-sorties. Lorsque le moniteur lit les cartes d'assignations, dans ces cartes il trouve un nom symbolique et un (ou plusieurs) nom absolu d'unité E/S. Cette carte veut dire « j'attribue à tel nom symbolique tel numéro d'unité ». Le moniteur de transition place le nom symbolique (après avoir vérifié qu'il est possible) dans la table des numéros d'unités du moniteur permanent, en face du numéro physique correspondant.

Il faut une carte (un nom symbolique) par fichier et on doit retrouver ce nom symbolique dans le programme. Il doit y avoir correspondance entre le programme et la carte de commande du « batch ». Dans le cas où le fichier tient sur plusieurs volumes, il peut y avoir en face du nom symbolique plusieurs numéros d'unités physiques suivant que l'on désire utiliser le flip flop sur 2, 3... lecteurs de bande.

Après ces cartes d'assignation le moniteur de transition lit la carte de contrôle disant : « charger et exécuter tel programme se trouvant sur tel volume ». Le moniteur de transition passe la main (donne le contrôle) à la routine de chargement du moniteur permanent. Celui-ci va chercher le programme sur le volume donné, puis il le charge en mémoire en effaçant (en écrasant) le moniteur de transition. Pendant le chargement il remplace les noms symboliques d'unités du programme par le nom physique placé dans sa table (cela peut aussi se faire lors du premier ordre d'E/S de chacun des fichiers, ordre que l'on appelle « ouverture du fichier »). Après quoi l'exécution du programme peut commencer.

2.3. — DÉROULEMENT DE LA JOURNÉE

On peut résumer le fonctionnement journalier du système d'exploitation séquentiel de la manière qui suit; avec l'aide de la figure 6.7.

Le matin, après avoir mis en route l'ordinateur, avoir monté le volume système, volume application et les volumes nécessaires au premier travail (on ne peut pas supprimer le temps de démarrage matinal), le pupitreur appelle la procédure du hardware mettant en place l'amorce du volume système dans la mémoire principale. Ensuite cette amorce appelle le moniteur permanent. Ce dernier restera dans la mémoire principale toute la journée. Il place à sa suite le moniteur de transition à qui il donne le contrôle. Celui-ci lit les cartes de commande.

En principe la première carte doit contenir la date du jour (ou bien alors cette date est entrée au pupitre). Cette date servira pour les labels de début des fichiers. Ensuite on doit trouver le nom du premier travail suivi de ses cartes d'assignation. Le moniteur de transition décodifie le contenu de ses cartes et place le résultat dans les tables du moniteur permanent. S'il trouve des cartes commentaires, il imprime ces commentaires au pupitre pour le pupitreur. Il lit après la carte d'exécution du programme. Le moniteur de transition recherche sur le volume application le programme donné. Lorsque celui-ci est trouvé, il redonne le contrôle au moniteur permanent, lequel charge en mémoire le programme à la place du moniteur de transition. Le contrôle est ensuite donné à ce programme nouvellement chargé.

— Ce programme peut être aussi bien un programme client (écrit par l'utilisateur) qu'un programme utilitaire (d'intérêt général, le plus souvent écrit par le constructeur). Les programmes utilitaires sont placés sur le volume système et les programmes clients sur les volumes applications mais ce n'est pas obligatoire. Lorsque le programme a terminé son déroulement, il peut se terminer par une fin normale, dans ce cas on revient directement à la routine du moniteur permanent qui charge le moniteur de transition..., et ainsi de suite. Il peut aussi se terminer par une fin anormale (par exemple il ne s'est pas déroulé correctement). Dans ce cas avant de charger le moniteur de transition, le moniteur permanent traite cette fin anormale. (Nous y reviendrons avec les essais chaînés).

A noter que nous avons parlé ici de volumes pour éviter de dire « bandes ou disques », en réalité du fait des cartes d'assignation, il y a complète indépendance entre les programmes et les unités physiques. On peut aisément remplacer une bande par un disque, ou vice-versa si les fichiers sont organisés séquentiellement. Dans le cas où les fichiers sont d'une organisation d'un autre type il faut, bien sûr, des mémoires à accès sélectif. La bande système peut elle aussi être remplacée par une mémoire à accès sélectif.

2.4. — LA TRANSLATION

Avec ce type de software apparaît une structuration très sommaire des programmes. Il est déjà possible d'utiliser la logique des blocs. (On appelle souvent le bloc de programme : un module de programme.) Les blocs de programme, donc, peuvent être compilés séparément.

Nous pouvons ainsi avoir le bloc A, le bloc B, le bloc C et le bloc D et obtenir des « plans de mémoire » ADB ou CBA. Ces plans de mémoire constituent un programme, le moniteur permanent étant placé en tête de la mémoire, comme dans la figure 6.8.

Dans les deux plans le bloc B, par exemple, n'est pas placé au même endroit dans la mémoire. Il doit pouvoir être translaté à l'intérieur de cette mémoire. Cette translation peut se faire au niveau du hardware, ou bien au niveau du software.

— Dans le cas où la translation n'a pas été prévue dans le hardware, le compilateur transforme les instructions du programmeur dans un langage que l'on appelle : « absolu translatable ». C'est-à-dire que les instructions sont compatibles avec le langage machine, mais comme si la première instruction

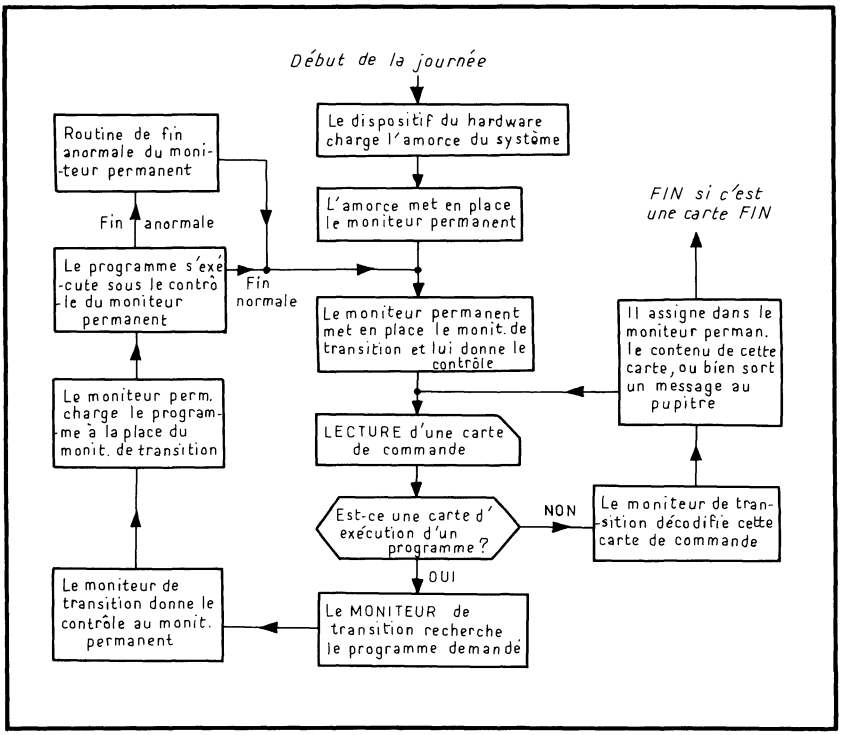


Fig. 6-7. — Fonctionnement journalier du système d'exploitation séquentiel.

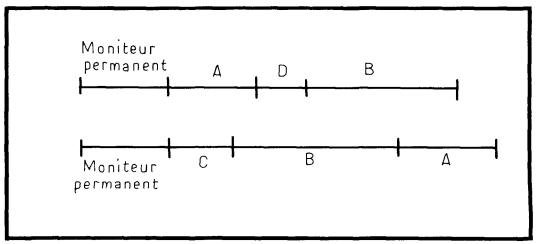


Fig. 6-8. — Les plans de mémoire A D B et C B A.

du bloc de programme était placée dans la position de mémoire 0000. En plus à chaque instruction on attribue un code de translation, suivant que les 2 adresses, ou 1 adresse, ou pas d'adresse de cette instruction doivent être translatées ou non. En effet il ne faut translater comme adresse que celles venant des adresses symboliques du programmeur. Si le programmeur a écrit une adresse en absolu (par exemple un numéro d'index ou une constante) on ne

doit pas modifier cette adresse. Par contre si on place le bloc de programme à partir de la position de mémoire 4500, il faut ajouter 4500 à toutes les adresses venant des adresses symboliques, dans l'absolu translatable.

Il existe ensuite entre le compilateur et le chargement du programme en mémoire un autre travail exécuté par un programme utilitaire que l'on peut appeler « translateur » ou « éditeur de liens ». Ce programme a comme fonction l'assemblage des différents blocs de programme pour fournir un programme. A noter qu'un programme peut ne compter qu'un seul plan de mémoire, on dit dans ce cas qu'il est « monophase ». Il peut aussi en comporter plusieurs. il est alors « multiphase ». Chacun des plans de mémoire s'appelle une phase Pendant l'exécution du programme on passe d'une phase à l'autre (chargement de la 1re phase, exécution de cette 1re phase, chargement de la 2e phase, exécution de cette phase...) sans avoir besoin d'appeler le moniteur de transition.

En même temps, ce « translateur » ou « éditeur de liens » translate les différentes instructions du programme. Le langage absolu translatable devient un langage absolu translaté.

On dit très souvent que l'absolu translatable est chargeable en mémoire mais n'est pas exécutable, alors que l'absolu translaté est chargeable et exécutable.

— Ou bien la translation a été prévue au niveau du hardware par un adressage permettant la translation. Par exemple, figure 6.9, une adresse se

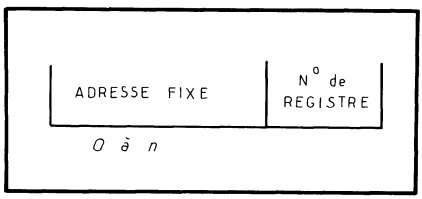


Fig. 6-9. — L'adressage indexé par un registre.

compose de 2 valeurs : une adresse fixe pouvant couvrir une partie limitée de la mémoire (de 0 à *n* positions de mémoire) et un numéro de registre. L'adresse qui est traitée par l'unité centrale est celle qui est égale à la somme de l'adresse fixe et du contenu du registre. Autrement dit, par construction, les adresses symboliques suivent déjà le principe de l'indexation. Le compilateur attribue à un bloc du programme (ou à chaque partie du bloc si le bloc est trop important) un numéro de registre. Ce registre est appelé registre de base du bloc. La première instruction d'un bloc de programme (ou d'une *partie du bloc*) doit être : la mise dans le registre de base de l'adresse du début du bloc. Le compilateur assemble en donnant comme adresse fixe à chaque référence symbolique le nombre de positions comprises entre la position de cette référence et le début du bloc.

Par exemple : prenons un bloc occupant en absolu 2 000 positions de mémoire. Ce bloc commence par une instruction : « mise dans le registre 3 du contenu du registre instruction (compteur ordinal) ». Une carte de commande signale au compilateur que le registre de base de ce bloc est le registre 3.

Supposons, comme dans la figure 6.10, qu'à la 1 500e position de ce bloc le compilateur trouve l'adresse symbolique (du programme) : « RETOUR ». Dans sa table de référence il met en face du symbole « RETOUR » : 1 500 indexé par le registre 3.

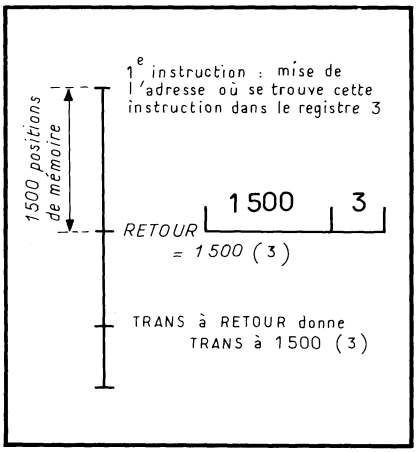


Fig. 6-10. — Le compilateur remplace le symbole RETOUR par l'adresse 1 500 indexée par le registre 3.

Nous pouvons trouver ainsi une instruction dans le programme : TRANS à RETOUR (transfert à l'instruction appelé RETOUR). Le compilateur remplace cette instruction (à l'aide de sa table des références) par : TR 1 500 (3), c'est-à-dire un transfert à l'adresse égale à la somme de 1 500 plus le contenu du registre 3.

Plaçons ce bloc à partir de la position de mémoire 48 000. La 1re instruction du programme met dans le registre 3 la valeur 48 000. Lorsqu'on arrive à l'instruction : TR 1500 (3) on se transfère bien à l'adresse 1500 + 48 000 qui est l'adresse où se trouve RETOUR.

Cela est valable bien sûr à la condition d'entrer dans le bloc par la 1re instruction. Dans le cas où l'on entre par un autre point dans ce bloc, il faut prendre la précaution préalable d'initialiser le contenu du registre de base avant d'exécuter les instructions du bloc.

A l'aide de ce dispositif l'ensemble des instructions d'un bloc est aisément translatable. Il faut seulement faire attention à ce qu'on appelle les constantes adresses. Ce sont les constantes contenant les adresses d'autres constantes, variables ou instructions. Ce type de constante est nécessaire lorsque le programmeur désire manipuler un contenant au lieu d'un contenu. Il est nécessaire lors de la translation d'ajouter ou de retrancher ce qu'il faut dans ces constantes adresses. C'est pourquoi le compilateur génère une table de constantes adresses.

Remarque : Dans certains ordinateurs cet inconvénient n'existe plus. L'adressage dans un programme reste toujours relatif à lui. Sa première position est toujours 000. Un registre spécialisé contient, l'adresse absolue de début de ce programme et lorsqu'il exécute des instructions le hardware ajoute automatiquement aux adresses du programme cette adresse de début.

2.5. — *LA RÉUNION DES DIFFÉRENTS BLOCS DE PROGRAMME*

Elle s'effectue par le programme utilitaire que l'on appelle suivant les cas « translateur » ou « éditeur de liens ».

Ce programme est commandé par des cartes de commande placées dans « le batch » d'entrée. Le compilateur a placé les différents blocs de programme sur des bandes (ou disques) intermédiaires avec, pour chacun, un nom donné par le programmeur. Il suffit de dire au « translateur » ou à « l'éditeur de liens » les noms des blocs que l'on désire et sur quelles unités ils se trouvent. On peut imposer l'ordre de ces blocs dans le plan de mémoire. Le programme utilitaire va chercher ces différents blocs et effectue dessus la translation nécessaire. Dans le cas où un bloc a des constantes adresses (constantes contenant une adresse) il les modifie à l'aide de la table générée par le compilateur.

En même temps que ces translations s'établissent ce qu'on appelle les liaisons entre les différents blocs. Les programmeurs ont donné dans chacun des blocs de leur programme (par des instructions de commande spéciales) des points d'entrée et des points de sortie. Ces points portent des noms, et ces noms sont connus au niveau du « translateur, éditeur de liens » car le compilateur les a conservés.

De deux choses l'une, ou bien la liaison d'un bloc à un autre est toujours la même et, dans ce cas, un point de sortie de bloc contient déjà le nom d'un point d'entrée d'un autre bloc et la liaison se fait automatiquement, ou bien cette liaison est essentiellement variable et il est nécessaire par des cartes de commande de dire au programme utilitaire : « reliez-moi tel point de tel bloc à tel point de tel autre bloc ».

Ces liaisons ne sont pas suffisantes car, lorsqu'on arrive dans un bloc, on est amené à utiliser des références situées dans le bloc que l'on vient de quitter. La chose se complique car lorsque le compilateur a rencontré ces références, elles sont pour lui inconnues. Il faut à nouveau des cartes de commande pour lui dire : ce sont des « *symboles externes* » au bloc. Elles se trouvent dans un autre bloc. De la même manière, lorsqu'une référence est susceptible d'être utilisée par un autre bloc, on la définie comme « *symbole interne* » dans le bloc qui la contient.

Résumons le travail supplémentaire que doit effectuer le compilateur :

— il doit fournir des instructions absolues translatables (si cela n'est pas prévu dans le hardware);

— il doit fournir une table des constantes adresses;

— il doit fournir une table des symboles internes et des symboles externes;

— à la rencontre d'une instruction contenant un symbole externe, il doit laisser tel quel ce symbole.

Au niveau du « translateur — éditeur de liens » on connaît donc ces symboles externes et internes et on peut les remplacer, après la formation du plan de mémoire par leur valeur absolue. On dit qu'un symbole externe est « satisfait » lorsqu'on a pu le remplacer par une valeur absolue, c'est-à-dire que l'on a rencontré dans un autre bloc du plan de mémoire un symbole interne ayant le même nom.

Très souvent on utilise aussi ce qu'on appelle le dispositif d'appel automatique de bibliothèque. Lorsqu'un symbole externe n'est pas satisfait alors que tout le plan de mémoire est en place, on va chercher dans une bibliothèque

(dont le nom et le numéro d'unité ont été donnés par une carte de commande) de blocs de programme, ce symbole. Si on trouve un bloc ayant, parmi ces symboles internes, le symbole correspondant, on amène ce bloc dans le plan de mémoire.

On verra plus loin qu'il n'est pas toujours nécessaire que les symboles externes soient satisfaits avant l'exécution du programme.

Une astuce très souvent utilisée pour passer d'un bloc à un autre, est l'utilisation d'une table de références. On place dans une table toutes les constantes adresses qui doivent être connues par les autres blocs. Par exemple dans la figure 6.11. On a l'adresse du facteur A, du facteur B, du facteur C

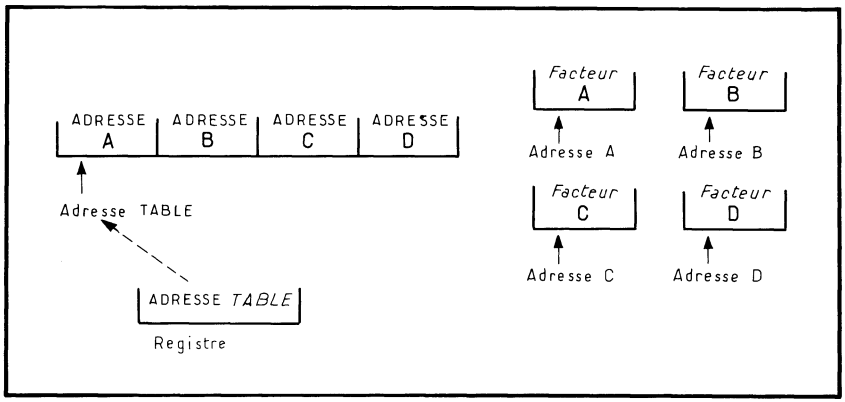


Fig. 6-11. — Table des constantes-adresses A, B, C et D.

et du facteur D. Avant de sortir du bloc, on met l'adresse du début de cette table dans un registre. Les autres blocs peuvent travailler alors avec ce registre pour retrouver les adresses dont ils ont besoin. (Cette manière de considérer les contenus comme des contenants s'appelle, lorsqu'elle est câblée dans le hardware, *l'adressage indirect.*)

2.6. — LA GÉNÉRATION DU SYSTÈME

Le plus souvent la configuration machine chez les différents clients n'est jamais la même (taille mémoire principale, dispositifs spéciaux, canaux, unités d'E/S, disposition de ces unités d'E/S..., etc.). Il est nécessaire d'avoir un software adapté, optimisé en fonction de la configuration machine. Le constructeur n'écrit pas un software pour chacune des configurations, cela serait impossible. Le software est divisé lui aussi en blocs. Comme tout ce qui est informatique, il respecte la logique des blocs. L'ensemble de tous les blocs de programme nécessaires à tous les softwares pour toutes les configurations machines possibles du type d'ordinateur donné est placé sur une « bande maîtresse ». En tête de cette bande maîtresse on place en absolu un moniteur valable pour la configuration minimum (c'est-à-dire capable de fonctionner sur toutes les configurations machines).

Cette bande arrive chez le client, celui-ci doit, dans des cartes de commande, spécifier : sa configuration machine, les programmes utilitaires

qu'il désire avoir dans sa bibliothèque, il peut aussi y ajouter des programmes à lui.

Cette bande est montée sur la machine, on appelle le moniteur de configuration minimum qu'elle contient, il lit les différentes cartes de commande. Petit à petit (parfois par des opérations longues) on compile et on forme le moniteur optimisé pour notre configuration, puis on constitue notre bibliothèque de programmes. On obtient ainsi notre bande système.

Pour les tris, par exemple, il est possible d'obtenir des programmes plus performants en précisant quel type de tris on pratique dans l'entreprise. Moins un programme utilitaire est généralisé (c'est-à-dire utile seulement à quelques applications) et plus il est rentable, car il est optimisé en conséquence.

Le plus souvent un ordinateur n'est pas quelque chose de rigide, de physiquement constant dans le temps. Le hardware est modifié sans cesse. Il y a des modifications techniques, des améliorations, mais aussi on s'aperçoit que tel type de circuit est défaillant lorsqu'on l'utilise de telle façon bien spéciale. Ce qui est vrai pour le hardware l'est aussi pour le software. Dans un ensemble servant à une multitude d'applications, un nombre incalculable de cas particuliers, on trouve toujours des cas défaillants ou améliorables. Par suite, le software n'est jamais stable et la bande maîtresse du constructeur est modifiée sans cesse. C'est pourquoi le client désireux de suivre les progrès techniques doit, de temps en temps, refaire la génération de son système avec la bande maîtresse de dernier niveau.

C'est pourquoi, aussi, nous ne pensons pas que dans l'immédiat le software (ou tout au moins le moniteur permanent) soit entièrement microprogrammé (le software devenant hardware) car alors il resterait figé.

2.7. — CONCLUSION SUR CE TYPE DE SOFTWARE

— Création d'un moniteur et d'une bibliothèque optimisés en fonction de notre configuration machine.

— Chaînage des travaux, on passe automatiquement d'un travail à un autre à l'aide d'une carte de commande.

— Les cartes d'assignation permettent de rendre les unités physiques complètement indépendantes du programme. Cela permet la préparation du travail suivant.

— L'obtention d'un programme en langage absolu se fait en deux temps : compilation des différents blocs, et ensuite assemblage de ces blocs en un tout à l'aide d'un programme utilitaire effectuant la translation et les liens de ces blocs.

3. — Un exemple d'application du système d'exploitation séquentiel : les essais chaînés

Nous l'avons vu, après avoir écrit leurs programmes, les programmeurs doivent les compiler, puis les tester. Lorsque l'ordinateur n'est pas occupé à

plein temps, on conçoit très bien qu'ils viennent eux-mêmes essayer leurs programmes en manipulant le pupitre, avec l'aide d'un opérateur. Mais au fur et à mesure que de nouvelles applications sont mises en place, ces manipulations ne sont plus viables (autrement que pendant les nuits et les week-ends). Il faut mécaniser les essais des programmes. Cette procédure s'appelle : les essais chaînés.

3.1. — LES 4 PARTIES

On divise un essai en quatre parties :

— *la phase compilation* qui comprend les différentes compilations des différents blocs de programme, la réunion de ces blocs en un tout pour obtenir le programme en langage absolu.

— *la phase génération de fichier.* Au cours de ces essais les manipulations doivent être réduites au minimum. Il ne s'agit pas, pour chacun des essais, d'avoir à monter telle bande, puis telle bande, puis tel disque... L'ensemble des jeux d'essais est mis sur un support uniforme : ce support étant généralement le même que les cartes de commande du batch. C'est-à-dire qu'en principe tous les jeux d'essai sont sur cartes. En tête de chacun de ces jeux d'essai, il y a une carte d'appel pour charger en mémoire un programme utilitaire de la bande système : un programme capable de mettre les jeux d'essai sur bande ou sur disque.

Ensuite et en tête de chacun des fichiers d'essai, il y a une carte de commande indiquant sur quelle unité il faut mettre le fichier d'essai qui suit. On place ainsi les jeux d'essai là où il le faut.

— *la phase d'exécution.* Il s'agit d'abord de charger (mettre) en mémoire le programme compilé ou assemblé précédemment. Ensuite de l'exécuter en utilisant les jeux d'essai que nous venons de placer. Éventuellement, il peut y avoir une partie du jeu d'essai ayant comme support le batch lui-même. On place cette partie après la carte de commande qui lance l'exécution du programme.

— *la phase d'analyse.* Cette phase analyse les résultats de la phase précédente. Une partie de cette analyse a pu se faire pendant l'exécution si on a demandé une trace du programme. On peut aussi demander un vidage de la mémoire. Si, par exemple, le programme a demandé « des points de contrôle » on peut lister leur contenu.

Le programmeur peut demander l'analyse d'un fichier de sortie. Cela se fait à l'aide d'une carte d'appel d'un programme utilitaire placé sur la bande système. Ce programme a pour fonction le vidage du contenu des bandes et des disques sur l'imprimante, mais en présentant ces contenus d'une façon claire, c'est-à-dire enregistrement par enregistrement. Après cette carte de commande, dans d'autres cartes le programmeur spécifie quelle unité il désire vider et cela de tel enregistrement à tel enregistrement.

3.2. — EXEMPLE DE « BATCH » NÉCESSAIRE POUR L'ESSAI D'UN PROGRAMME

Prenons un programme ayant 3 fichiers en entrée A, B, C, 1 fichier carte D et 3 fichiers en sortie E, F, G. (Fig. 6.12.) Les lettres A à G sont les noms

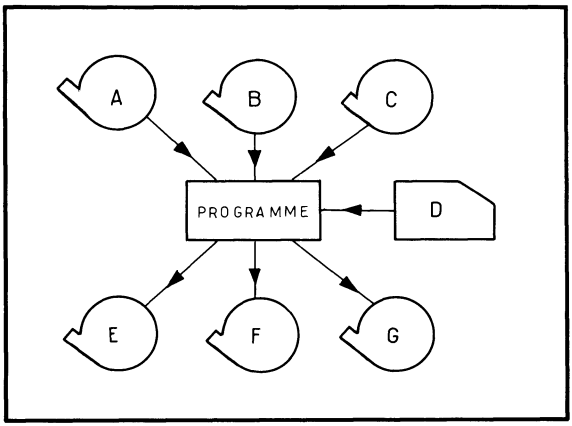


Fig. 6-12. — Programme exemple à mettre au point.

symboliques donnés par le programme à ses unités d'E/S. Ce programme est écrit en 4 blocs par 4 programmeurs différents. Un bloc est écrit en autocodeur et les 3 autres en cobol. On désire tester le programme constitué par ces 4 blocs (préalablement on a pu vérifier le fonctionnement de chacun des blocs en les testant séparément, en remplaçant les symboles externes et les symboles internes par des adresses données par le software).

Le batch des cartes pour cet essai est organisé de la manière suivante :

— *(En tête).* Une carte indiquant qu'il s'agit du début d'un travail. Cette carte contient, entre autres choses, le nom du travail et les noms des programmeurs.

— Des cartes d'assignation pour dire que le symbole A est attribué à telle unité physique, B à telle autre,... que D est confondu avec les cartes qui contiennent le batch.

— *(1re phase).* Une carte d'appel du compilateur cobol qui se trouve sur la bande système.

— Le premier bloc de programme symbolique cobol (avec à la fin une carte de fermeture pour indiquer où il s'arrête dans le paquet de cartes).

— Le deuxième bloc de programme symbolique cobol.

— Le troisième bloc de programme symbolique cobol.

— Une carte d'appel du compilateur autocodeur sur la bande système.

— Le bloc de programme symbolique écrit en autocodeur. (le résultat de ces 4 compilations est placé sur un volume intermédiaire que l'on peut conserver si on ne désire pas recommencer la compilation lorsqu'on se servira à nouveau d'un de ces blocs).

— Une carte d'appel du programme « translateur-éditeur de liens » sur la bande système.

— Les cartes de commande pour ce programme, pour lui dire que l'on désire un plan de mémoire constitué des 4 blocs précédents. On donne en même temps un nom à ce plan de mémoire. Il est placé sur une bande-application.

— *(2e phase)*. Une carte d'appel pour mettre en mémoire et exécuter le programme utilitaire de la bande système, appelé « générateur de fichiers ».

— Une carte de commande pour ce programme, pour lui dire : « les cartes qui suivent sont à mettre sur l'unité symbolique A, avec tel dessin d'enregistrement ».

— Le jeu d'essai de A (avec à la fin une carte de fermeture pour indiquer où il s'arrête dans le paquet de cartes.)

— Une carte commande pour ce programme pour l'unité symbolique B.

— Le jeu d'essai de B.

— Une carte commande pour ce programme pour l'unité symbolique C.

— Le jeu d'essai de C.

— Une carte de commande pour prévenir le moniteur que l'on va exécuter un essai.

— *(3e phase)*. La carte d'appel du plan de mémoire construit précédemment, avec son nom et le nom symbolique de la bande application qui le contient. Le moniteur va chercher ce programme sur cette bande et le charger en mémoire (en même temps on peut demander une trace de telle instruction à telle instruction). Le contrôle est donné à ce nouveau programme et il s'exécute.

— Le jeu d'essai de l'unité d'entrée D.

— *(4e phase)*. Une carte d'appel du programme « analyse de volume » placé sur la bande système.

— Les cartes de commande pour ce programme, pour demander l'analyse des unités symboliques E, F, G.

L'organigramme de la figure 6.13 résume les quatre phases de cet essai.

Les batchs des différents essais de programme sont ainsi placés à la suite les uns des autres. Lorsqu'on en a terminé un, on passe au suivant. Avant de démarrer les essais chaînés, les opérateurs montent le maximum de volume afin que le tout fonctionne sans arrêt.

Il faut aussi prévoir le cas où il y a une anomalie dans une des phases. S'il y a une erreur au niveau de la première ou de la deuxième phase (le compilateur signale certaines erreurs dans le langage symbolique et ne corrige que celles sans gravité, il peut y avoir une erreur dans les cartes de commande du « translateur » ou des jeux d'essai) il est inutile de poursuivre l'essai plus loin. Dans ce cas, on demande au moniteur de lire toutes les cartes qui suivent en les ignorant jusqu'à ce que l'on rencontre la première carte de commande de l'essai suivant. On signale seulement au programmeur la cause de l'erreur.

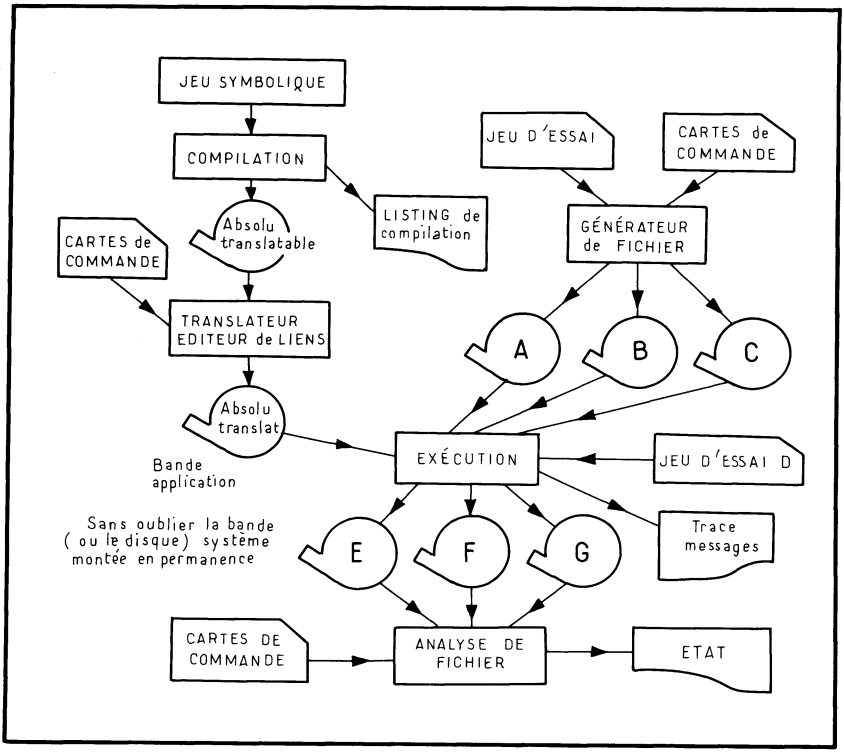


Fig. 6-13. — Organigramme de l'essai de notre programme.

Par contre, s'il y a une erreur au niveau de la 3e phase (au cours de l'exécution du programme) cela n'est pas une anomalie puisque c'est un essai de programme. Il est nécessaire d'analyser la cause de cette anomalie. On sort alors par fin anormale de programme. Le moniteur vide le contenu de la mémoire et l'état machine au moment de cette anomalie, sur une bande ou sur un disque. Le programmeur peut ensuite demander le listage de ce point de contrôle, (à l'aide d'une carte de commande), enfin le moniteur signale dans ses tables qu'il y a eu anomalie.

Lors de la 4e phase, avant de positionner les volumes séquentiels à leur départ pour les analyser, le programme d'analyse doit placer sur ces bandes un label fin. En effet, comme le programme ne s'est pas exécuté jusqu'à sa fin normale, il est fort probable qu'il n'y a pas de label fin sur ces bandes, il est nécessaire de repérer jusqu'où il faut les lister. Si on dispose d'un ordinateur périphérique, on peut effectuer sur cet ordinateur un « cartes à bande » du batch, et un « bande à imprimante » des résultats. De cette manière l'ordinateur de traitement ignore les unités lentes même pour les essais.

Enfin, si on dispose d'une horloge interne à l'ordinateur, on peut chronométrer automatiquement le temps pris par chacun des essais.

Les essais chaînés sont une procédure très efficace, on peut ainsi tester des programmes à distance, par correspondance.

CHAPITRE VII

LE SYSTÈME D'EXPLOITATION A PRIORITÉS LA MULTIPROGRAMMATION

1. — Le principe de fonctionnement

Lorsque l'on examine le fonctionnement d'un programme dans le temps, on constate, figure 7.1, qu'il se décompose en calcul (de durée relativement courte), puis en entrée-sortie (de durée relativement longue), puis à nouveau en calcul, puis à nouveau en E/S..., etc. Et cela malgré la simultanéité car il arrive un point où le calcul a besoin du résultat de l'entrée-sortie pour pouvoir continuer. En gestion surtout, les temps de calculs sont pratiquement négligeables devant les temps d'E/S. Pendant ces derniers la mémoire principale et l'unité centrale sont libres, (sauf pendant les quelques microsecondes nécessaires aux échanges canaux-mémoire principale). D'où l'idée de mettre non pas un mais plusieurs programmes en même temps dans l'ordinateur.

— On effectue les calculs d'un programme pendant que s'exécute les E/S des autres programmes. Pour cela il est nécessaire d'attribuer aux programmes des priorités différentes.

— Prenons par exemple (fig. 7.2) le cas de 3 programmes placés dans la mémoire principale, l'un de priorité 1, un autre de priorité 2, et le troisième de priorité 3. Au départ on exécute le calcul sur le programme de priorité la plus grande, c'est-à-dire celui de priorité 1. Il arrive un moment où il est nécessaire pour ce programme de lancer un ordre d'E/S, et d'attendre l'exécution de cet ordre. On ne peut plus continuer l'exécution de ce programme.

Le moniteur du software donne alors le contrôle au programme 2 car ce programme est plus prioritaire que le troisième. Le programme 2 exécute le calcul qu'il a à faire, au bout d'un certain temps, il doit lui aussi lancer et attendre un ordre d'E/S.

Le moniteur donne le contrôle au programme qui reste, c'est-à-dire le programme 3. Puis arrive un moment où, par exemple, l'entrée-sortie du programme 1 est terminée. A ce point il y a dans la mémoire deux programmes capables de fonctionner : le 1 et le 3. Le moniteur donne le contrôle au programme 1 car c'est le plus prioritaire.

Le programme 1 poursuit son calcul jusqu'à ce qu'à nouveau il ait besoin d'attendre l'exécution d'une E/S, le programme 2 n'est toujours pas prêt car son ordre d'E/S n'est pas terminé, le moniteur donne donc à nouveau le contrôle au programme 3. Celui-ci poursuit son calcul jusqu'à ce que, par exemple, l'ordre d'E/S du programme 2 soit terminé. Le moniteur passe alors la main au programme 2. Si enfin, à nouveau, l'ordre d'E/S du programme 1 se termine c'est ce programme qui reprend le contrôle et ainsi de suite...

On arrive de cette manière à faire en même temps plusieurs travaux, la mémoire principale restant inoccupée seulement pendant les temps très rares où tous les programmes sont en attente d'E/S.

Du point de vue vocabulaire il faut noter que dans l'ordinateur le terme « programme » est à proscrire. En effet, il arrive souvent qu'un même bloc de programme soit utilisé par plusieurs travaux. Il n'est placé qu'une fois en mémoire, et ses instructions sont utilisées comme des ressources au même titre que les E/S. C'est pourquoi on dit que l'on a affaire à des « *tâches* » afin de bien distinguer l'exécution d'un travail d'une part, et les instructions des différents blocs de programme d'autre part. Les différentes tâches se partagent les différentes ressources de l'ordinateur.

Le fonctionnement d'un tel software est à la fois très intéressant et très complexe.

Du point de vue occupation mémoire le moniteur permanent prend une place très importante, comme ordre de grandeur disons que cette place est le 1/3 de la capacité machine.

2. — Les modifications nécessaires dans le hardware

2.1. — LA PROTECTION MÉMOIRE

Plusieurs tâches, donc, vont être mises en même temps dans l'ordinateur Il ne s'agit pas qu'une tâche aille perturber une autre tâche (*ce qui appartient* en propre à cette tâche). C'est pourquoi il est nécessaire d'avoir dans le hardware un dispositif de protection mémoire.

Lorsque le software initialise une tâche il lui donne un numéro et lui attribue de la place en mémoire : une ou plusieurs zones de mémoire. Dans ces zones on place toutes les variables, zones d'E/S, constantes, instructions spécifiques de la tâche. Pour avoir accès à ces zones il est nécessaire d'être positionné sur le numéro de la tâche. Autrement dit, lorsque le moniteur donne le contrôle à une tâche il doit mettre le numéro correspondant dans un des

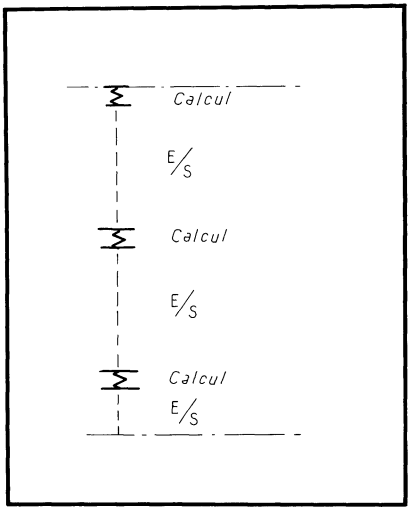


Fig. 7-1. — Fonctionnement d'un programme dans le temps.

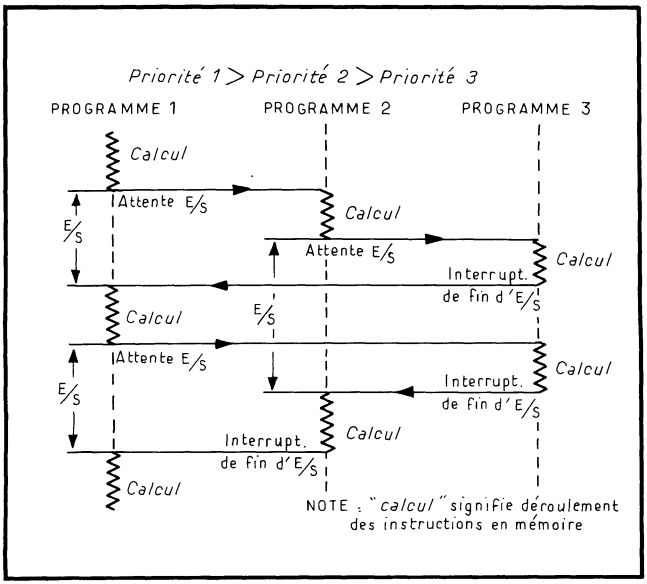


Fig. 7-2. — Multiprogrammation entre trois programmes.

registres de l'unité centrale. (Par exemple, dans certains ordinateurs, à chaque numéro correspond en mémoire, à une position connue à la fois du hardware et du software, un registre. Dans ce registre le software a pu placer l'adresse d'une table de segmentation dans laquelle pour chaque segment se trouve son adresse de début et la dimension de son occupation mémoire). De cette manière chaque tâche peut toucher à ce qui lui appartient mais ne peut pas écrire et lire sur les zones qui appartiennent aux autres.

De cette manière également le moniteur gère l'occupation de la mémoire. Il sait là où il y a de la place disponible. (Il dispose de l'ensemble constitué par les tables de segmentation.)

Il y a aussi les zones de la mémoire communes à toutes les tâches. Ces zones contiennent les blocs de programme utilisables pour toutes. Il n'y aura pas là de protection mémoire. (On verra qu'il y a une catégorie de blocs de programme sur lesquels on n'a besoin que de lire; on dit qu'ils sont « réentrants », sur ces blocs par précaution on peut laisser la protection mémoire en écriture et enlever seulement la protection mémoire en lecture.)

Les tâches du moniteur (le moniteur est lui aussi constitué de tâches, ce sont en principe les tâches les plus prioritaires dans la machine) doivent avoir accès à toutes les positions de la mémoire.

Le moniteur se réserve aussi quelques instructions spéciales (privilégiées). Par exemple il gère l'ensemble des files d'attente des entrées-sorties, il ne faut pas qu'une tâche utilisatrice perturbe ces files d'attente en lançant elle-même directement, un ordre d'entrée-sortie. Seul le moniteur a le droit de lancer des ordres d'entrée-sortie. C'est pourquoi l'ordinateur a 2 modes de fonctionnement : le mode moniteur (ou superviseur) dans lequel c'est une tâche de moniteur qui a le contrôle de l'unité centrale, elle peut utiliser toutes les instructions. Le mode problème dans lequel c'est une tâche de l'utilisateur qui a le contrôle. Elle n'a pas le droit d'utiliser les instructions réservées au moniteur (pas d'ordre d'E/S! ce mystère va s'expliquer avec les interruptions).

2.2. — LES INTERRUPTIONS

Il est nécessaire aussi de pouvoir passer aisément d'une tâche à une autre tâche, n'oublions pas qu'à chaque fois il faut stocker l'état de la machine afin de pouvoir le réinitialiser plus tard (revenir à la tâche). D'où l'intérêt de condenser l'ensemble de cet état machine dans un nombre de ferrites minimum.

Lorsque l'on passe d'une tâche à une autre on doit stocker l'état machine en cours dans une zone de mémoire appartenant à la « tâche en cours »; cette zone s'appelle « bloc de contrôle de la tâche ». Puis on doit prendre dans « le bloc de contrôle » de la tâche où l'on va, l'ancien état machine qui s'y trouve et le mettre dans les registres de l'unité centrale.

C'est pourquoi on a l'habitude de représenter une tâche par le schéma de la figure 7.3 avec en haut à gauche son bloc de contrôle. Ce dernier est une table contenant les images du compteur ordinal, des registres et des indicateurs tels qu'ils étaient la dernière fois qu'on a quitté la tâche.

En réalité ce passage d'une tâche à une autre tâche est bien plus compliqué que cela; il faut aussi passer par le moniteur du software, car c'est lui qui gère la file d'attente, par priorité, des tâches. D'où la quasi-nécessité

d'avoir un basculement automatique des états machines. Ce basculement s'appelle l'interruption. (Nous avons vu déjà que l'interruption de fin d'E/S est nécessaire pour le système d'exploitation séquentiel.)

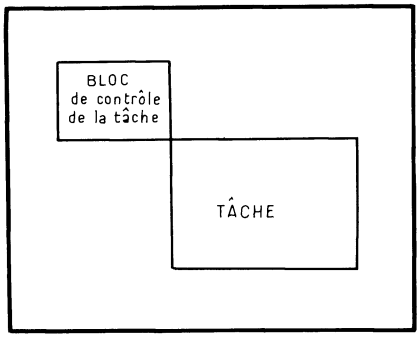


Fig. 7-3. — La tâche et son bloc de contrôle.

Ce basculement s'effectue de la manière suivante : il y a l'état machine en cours et il y a dans la mémoire des places réservées pour les états passés ou à venir. Pour chaque type d'interruption l'ordinateur possède deux zones un ancien état et un nouvel état. Lorsque l'interruption se produit, automatiquement l'état en cours est mis dans l'ancien état, et le nouvel état prend la place de celui en cours. L'état machine contient bien sûr le compteur ordinal, et ce basculement modifie le contenu de ce compteur. On se retrouve après chaque type d'interruption à un point bien particulier du moniteur. Celui-ci peut alors traiter cette interruption et gérer le contenu des anciens et nouveaux états machines. (Afin qu'il n'y ait pas interférence entre plusieurs interruptions se produisant presque en même temps, le moniteur a la possibilité d'en masquer certaines, de les mettre en réserve pour les traiter plus tard.)

2.3. LES DIFFÉRENTS TYPES D'INTERRUPTION

— Nous connaissons déjà l'interruption de fin d'E/S.

— Lorsqu'il y a une anomalie dans une tâche, par exemple une erreur de programmation, un dépassement de capacité dans un calcul, etc., il ne faut pas abandonner pour cela les autres tâches. Il doit donc y avoir un traitement automatique de cette anomalie. C'est pourquoi le hardware prévoit une interruption en cas d'erreur dans une instruction, (soit au moment de la phase analyse, soit au moment de l'exécution, soit à la fin de l'exécution de cette instruction).

Il y a en réalité différents types d' « erreur-programme », et cela est très intéressant lorsque l'on teste (essaye) une tâche. Il est possible de dire au moniteur : « pour tel type d'erreur, revenez à tel point de mon bloc de programme », et le programmeur a placé, à ce point, une routine à lui, qui analyse l'erreur.

— Le moniteur contient tous les ordres d'E/S et il gère toutes les files d'attente qui en résultent. Les tâches problèmes (utilisatrices) sont amenées à appeler le moniteur afin de l'informer sur leurs besoins en ressource (E/S et aussi les blocs de programme commun). C'est pourquoi il est nécessaire, dans le programme, de créer volontairement une interruption (afin de passer du mode problème au mode moniteur). C'est l'interruption d'appel moniteur (ou d'appel superviseur).

— Si l'ordinateur possède des unités d'E/S à distance, pouvant interroger puis modifier un fichier placé sur une mémoire à accès sélectif, il est nécessaire que cette interrogation puisse se faire à tout moment. Cette interrogation déclenche une interruption de télé-traitement.

— Il faut aussi contrôler le temps d'exécution de chaque tâche. Si une tâche dure plus que prévu on peut décider de l'abandonner (de la terminer) immédiatement. Il existe pour cela un dispositif d'horloge interne qui permet au moniteur de mesurer le temps pris réellement par chacune des tâches. Lorsqu'un certain temps est dépassé on peut demander une interruption d'horloge interne.

— En cas de panne-machine il est nécessaire que l'ordinateur essaye avant tout de se dépanner lui-même (dans le cas d'une panne intermittente). La panne-machine crée une interruption d'un type spécial. On se branche à un point du moniteur qui conserve l'état-machine au moment de cette panne puis réessaye l'instruction qui a déclenché la panne. Seules les pannes permanentes arrêtent le fonctionnement de l'ordinateur.

On dit qu'une interruption est « asynchrone » lorsqu'elle est imprévisible dans le temps. Cela est vrai pour toutes les interruptions sauf pour les appels moniteurs. Par antonymie on dit que les tâches sont « synchrones » car elles effectuent généralement des travaux révolutifs : entrée-calcul-sortie, entrée-calcul-sortie...

3. — La gestion des données

La gestion des données comprend toutes les routines « IOCS » nécessaires aux différentes organisations de fichier : séquentiel, accès direct, séquentiel indexé, cloisonné et télétraitement. On distingue l'IOCS physique et l'IOCS logique.

De plus le software peut gérer lui-même l'emplacement de ses fichiers. Le volume (obligatoirement à accès sélectif) qui contient le système possède un répertoire de tous les volumes avec le nom de tous les *répertoires qu'ils* contiennent et aussi le nombre de cylindres ou de pistes encore inoccupées sur ces volumes. L'ensemble fonctionne comme l'organisation cloisonnée. On ne donne plus un numéro d'unité d'E/S au software, on lui donne un nom de fichier. Le système recherche dans son répertoire sur quel volume il faut aller le chercher. Si ce volume n'est pas monté sur l'ordinateur, il envoie un message au pupitre pour demander à l'opérateur de le mettre en place. Ensuite il consulte le répertoire du volume et en descendant les branches de l'organisation cloisonnée, on aboutit au label du fichier, puis au fichier proprement dit.

Inversement, si on désire créer un nouveau fichier. Le software gère l'allocation de place avec les volumes. Il va lui-même décider de le mettre sur tel volume et à tel endroit de ce volume.

L'utilisateur peut bien sûr imposer au système son volume, sa partie de volume, mais il est évident que dans ce cas on perd certains avantages. Le software optimise les entrées-sorties en fonction de la configuration de l'ordinateur, des montages de volume et de l'équilibrage des canaux. (Donner à peu près le même travail à chaque canal). Si l'on veut avoir une gestion automatique de volume, avec le minimum de manipulations, une optimisation des temps d'exécution, il est préférable de laisser le système choisir lui-même la place où il doit mettre ce fichier. La règle d'or est d'imposer le moins de chose possible au software.

Au moment de la compilation (transformation du langage du programmeur en langage machine) l'ordinateur génère dans le programme résultant, pour chaque fichier une table. Cette table s'appelle le « bloc de contrôle du fichier ». Elle a un format standard. On doit y trouver au moment de l'exécution des entrées-sorties un certain nombre de paramètres : organisation du fichier zones d'entrée-sortie, format des enregistrements, contenu du label, technique de travail des zones d'entrée-sortie... Ces paramètres viennent comme l'indique la figure 7.4 de trois endroits différents.

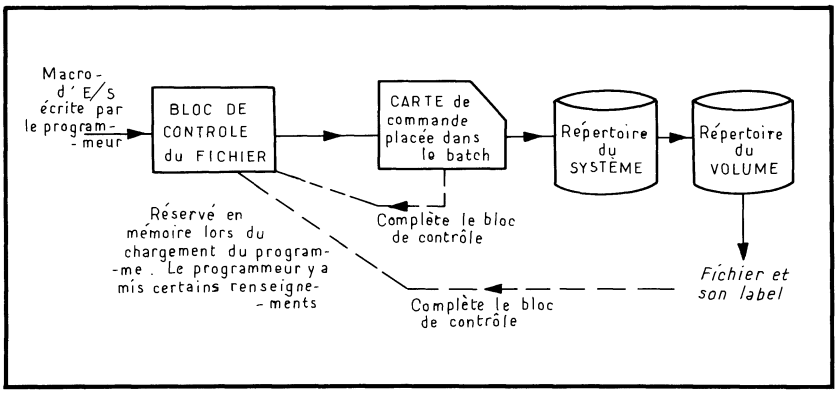


Fig. 7-4. — Le bloc de contrôle du fichier est rempli par le programmeur, la carte de commande du fichier et le label du fichier.

1) Du programme lui-même. Dans celui-ci il y a des renseignements décrivant le fichier. Parmi eux il doit au moins y avoir le nom d'une carte du Batch (de la gestion des travaux).

2) Dans ce Batch on doit normalement trouver après la carte d'appel du programme une carte contenant ce nom. Cette carte contient, elle aussi des renseignements sur le fichier. Elle doit contenir au moins le nom du fichier.

3) Dans le cas où le fichier existe déjà, ce nom est connu du répertoire du software. Par son organisation cloisonnée le système peut se positionner sur le label. Ce label contient lui aussi certains renseignements sur le fichier.

Il peut y avoir des interférences entre ces renseignements. Par exemple : le programme indique un format des enregistrements, et la carte du Batch en indique un autre différent. Il est donc nécessaire d'établir une priorité entre les trois origines.

En principe, le programme a priorité sur la carte, laquelle a priorité sur le label. Il y a intérêt à mettre le moins possible de renseignements dans le programme. De cette manière, le même programme peut servir pour des fichiers différents, une modification dans le fichier n'entraîne pas une modification dans le programme (on dit ici programme car il s'agit de la tâche avant son exécution).

Lorsque le fichier est à créer, il y a intérêt à mettre dans la carte du batch un ordre de grandeur sur le nombre de pistes ou de cylindres qu'occupera le fichier de manière à ce que le système choisisse une place en conséquence.

En réalité, avec ce système, il arrive au bout d'un certain temps que dans un volume les fichiers s'enchevêtrent les uns dans les autres. Par exemple : le fichier A occupe les cylindres 1 à 19, 30 à 59 et 80 à 89, le fichier B les cylindres 20 à 29 et 90 à 129, le fichier C les cylindres 60 à 79 et 130 à 179. C'est pourquoi on dispose de programmes utilitaires permettant de temps en temps, par des disques à disques, de simplifier cet enchevêtrement. Parallèlement, ces programmes mettent à jour les répertoires.

Le système ne connaît que les répertoires, ainsi on peut très bien remettre à blanc les informations contenues dans un fichier sans pour cela supprimer le fichier. Il suffit de ne pas modifier parallélement les répertoires. Par contre, si on efface un nom dans les répertoires sans toucher au fichier proprement dit, le système réagit comme si ce fichier n'existait plus.

Dans tous ces fichiers, dont l'utilisateur ignore l'emplacement exact (mais peut le connaître en listant le répertoire) il peut arriver qu'un fichier soit confidentiel (par exemple : le fichier du personnel). On crée un fichier supplémentaire qu'on appelle le fichier des « mots de passe ». Il y a un mot de passe par fichier protégé. Dans le label de ces fichiers, un signal indique que l'on n'y a pas accès directement. A la lecture du label le système envoie un message à l'opérateur, lui demandant de frapper au pupitre le mot de passe. Après l'entrée de ce mot de passe, le système le compare à celui que l'on trouve pour ce fichier, dans le fichier des mots de passe. S'il y a égalité, le système donne le contrôle à la tâche qui l'a demandé, sinon ce contrôle est refusé.

4. — La gestion des travaux

Le batch est organisé comme pour le système d'exploitation séquentiel : par travaux. A l'intérieur de chaque travail il peut y avoir plusieurs exécutions (étapes). Dans chaque exécution, il faut une carte de commande par fichier.

On donne, en plus, à chaque travail un degré de priorité.

Les différentes exécutions d'un même travail se font toujours d'une manière séquentielle, les unes après les autres, dans l'ordre des cartes de

commande. Il est possible de sauter une ou plusieurs exécutions suivant les résultats des exécutions précédentes. Chaque exécution peut fournir un code « retour » aux exécutions suivantes. Par exemple une compilation donne un code « bon » ou un code « mauvais » au « translateur-éditeur de liens » suivant que la compilation a été bonne ou mauvaise. Dans les cartes de commande des exécutions, on peut tenir compte des codes retours des étapes précédentes, et exécuter ou sauter l'étape.

Dans les cartes de commande du fichier on peut donner au volume occupé par ce fichier certaine qualité suivant que l'on désire conserver d'une façon définitive ce fichier, ne le conserver que pendant le cours du travail ou bien ne pas le conserver du tout (fichier travail). On attribue une qualité spéciale au volume qui contient le software et ses programmes utilitaires.

Le système lit l'ensemble du batch et le place sur un volume à lui. Il gère lui-même la file d'attente des travaux en entrée, en tenant compte des priorités et aussi des volumes déjà montés, enfin à priorités égales et à montages égaux de l'ordre d'entrée de ses travaux.

Au moment de la génération du système on a indiqué un nombre n maximum de travaux pouvant se faire en même temps. Lorsque l'occupation des ressources de l'ordinateur (occupation mémoire principale, unités d'entrée-sortie) le permet, la gestion des travaux sélectionne le premier travail à exécuter. (Ou bien si une tâche vient de se terminer, elle prend l'exécution suivante dans la chaîne des programmes du même travail).

Elle va assigner de la place sur les volumes pour les fichiers à créer, demander à l'opérateur de monter certains volumes. Lorsque tout est prêt, elle le signale à la « gestion des tâches » (à la partie du moniteur gérant les tâches). Celle-ci va allouer au travail à exécuter de la place en mémoire principale, puis va charger les blocs de programme nécessaire. Le tout devient pour le système une tâche.

Le pupitreur a la possibilité, au pupitre, de modifier certains ordres : changer un numéro de volume, changer un numéro de priorité...

Lorsqu'une tâche a des impressions d'état, des perforations de cartes à faire, il n'est pas nécessaire de passer par un ordinateur périphérique. Dans le cadre de la multiprogrammation, les sorties lentes peuvent se faire sur un ordinateur de traitement. Malgré tout la répartition entre les temps d'impression et de perforation ne correspond pas à l'importance des tâches qui les commandent. Il y a intérêt aussi à terminer une tâche le plus rapidement possible, afin de pouvoir la remplacer par une autre. C'est pourquoi le système dépose les sorties lentes sur une mémoire à accès sélectif intermédiaire. Il les imprimera ou les perforera un peu plus tard. Il gère ainsi une file d'attente des sorties en tenant compte des temps libres et des priorités.

(Pour ses différents messages, le système peut se réserver une imprimante, afin de les sortir immédiatement).

Remarque : Il est possible de « cataloguer des travaux ». On dit cataloguer « une procédure ». On écrit une fois pour toutes un ensemble de cartes de commande et on les place dans une bibliothèque (organisation cloisonnée). Alors pour exécuter cette procédure une seule carte du batch suffit : une carte d'appel de la procédure.

5. — Les différentes classes dans les blocs de programme

On distingue 3 catégories de blocs de programme.

5.1. — LE BLOC PROGRAMME UTILISABLE UNE FOIS

On ne peut pas réutiliser ce bloc une fois qu'on l'a utilisé. Il faut le recharger en mémoire à chaque fois. Par exemple, il contient des compteurs et des aiguillages, et dans sa fin de travail ils ne sont pas réinitialisés.

5.2. — LE BLOC DE PROGRAMME RÉUTILISABLE SÉQUENTIELLEMENT

On peut le réutiliser sans avoir à le recharger en mémoire, une fois qu'on l'a utilisé complètement (du début à la fin).

5.3. — LE BLOC DE PROGRAMME RÉENTRANT

C'est un bloc de programme dont on peut interrompre le déroulement à n'importe quel point de ses instructions et ensuite reprendre l'exécution à n'importe quel autre point. Cette définition abstraite va s'éclairer avec l'exemple suivant : Imaginons dans l'ordinateur 2 tâches : la tâche 1 plus prioritaire que la tâche 2. Dans la mémoire se trouve le bloc de programme réentrant C, comme dans la figure 7.5.

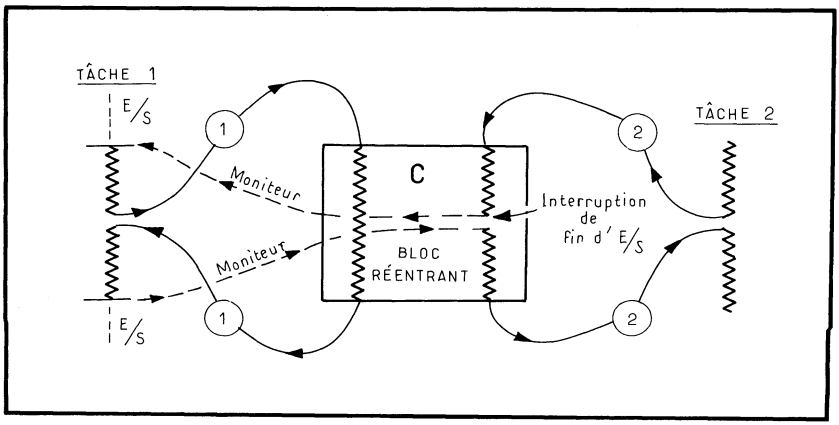


Fig. 7-5. — Le bloc de programme réentrant C est utilisé par la tâche 1 et la tâche 2.

Supposons qu'à un moment donné la tâche 1 est arrêtée car elle attend un événement : la fin d'une entrée-sortie. C'est la tâche 2 qui a le contrôle de

l'unité centrale, elle déroule ses instructions. Elle a besoin du bloc de programme C, elle commence donc à exécuter les instructions de ce bloc. Puis en plein milieu de C, l'entrée-sortie de 1 se termine. Il y a interruption de fin d'entrée-sortie.

Le moniteur prend le contrôle, après avoir vérifié que l'entrée-sortie qui vient de se terminer s'est passée correctement, il redonne du travail, s'il y en a, à cette unité d'entrée-sortie. Puis il consulte la file d'attente des tâches prêtes à fonctionner. Il y a la tâche 1 et la tâche 2, comme la tâche 1 est la plus prioritaire, c'est elle qui prend le contrôle. La tâche 1 déroule ses instructions. Il peut arriver un moment où elle a besoin, elle aussi, du bloc réentrant C. Elle va l'exécuter complètement, puis ensuite revient poursuivre ses instructions. Elle doit alors à nouveau attendre une entrée-sortie. Elle se met en attente et le moniteur passe la main à la tâche 2.

Lorsque le moniteur interrompt une tâche, il place l'état de la machine et le contenu des registres dans le bloc de contrôle de cette tâche. Lorsque ensuite il lui redonne le contrôle, il prend le contenu de ce bloc et réinitialise l'état machine et les registres.

On va donc poursuivre la tâche 2 où on l'a laissée, c'est-à-dire en plein milieu du bloc réentrant C. On continue le déroulement de C puis on revient aux instructions propres à la tâche 2.

On a ainsi décrit une partie de C (tâche 2), puis exécuté complètement C (tâche 1), et terminé C (tâche 2).

Cela explique déjà pourquoi on est amené à faire une distinction entre la tâche et le programme : un bloc de programme peut servir à plusieurs tâches.

Reste à voir comment doit être écrit un bloc de programme réentrant : il est en lecture seulement. Il n'y a pas une instruction qui modifie une autre de ses instructions, il peut posséder des constantes mais ne peut pas avoir de variable. Il fonctionne grâce à l'indexation.

Afin de comprendre il est nécessaire de donner un exemple : imaginons que le programme C calcule une règle de trois :

$$\text{RESULT} = \frac{\text{TAUX} \cdot \text{VALEUR}}{\text{BASE}}$$

Pour écrire un tel programme on peut attribuer à chacune de ses valeurs un numéro d'index (ou de registre).

Prenons :

— le registre 1 pour RÉSULT
— le registre 2 pour TAUX
— le registre 3 pour VALEUR
— le registre 4 pour BASE

et on écrit la figure 7.6.

Autrement dit lorsque le programme réentrant doit travailler avec des variables il utilise des index. (Si le nombre des variables est très grand on construit, dans chaque tâche, une table d'adresses des différents paramètres. Il suffit d'avoir un seul registre qui contient l'adresse de début de table de la tâche en cours.)

La tâche 1 va se réserver dans ses instructions à elle des variables qu'elle peut appeler : RESULT 1, TAUX 1, VALEUR 1, BASE 1. Avant de se brancher

au programme réentrant elle place respectivement dans les registres 1, 2, 3, 4 les adresses de RESULT 1, TAUX 1, VALEUR 1, BASE 1.

La tâche 2 procède de la même manière avec ses propres zones. Le programme réentrant travaille ainsi directement sur les variables qui appartiennent à la tâche en cours, et lorsque l'on passe d'une tâche à une autre il s'y retrouve puisqu'à chaque fois le moniteur restaure le contenu des registres (mis en réserve dans le bloc de contrôle de la tâche).

La protection mémoire ne gêne pas le fonctionnement d'un programme réentrant. En effet elle fait aussi partie de l'état machine, par exemple c'est un registre contenant l'adresse d'une table de segmentation. Ce registre est lui aussi mis en réserve dans le bloc de contrôle de la tâche. Lorsqu'il travaille avec la tâche 1 l'état machine est positionné sur la protection mémoire de cette tâche, il peut ainsi lire ou écrire sur les variables appartenant à la tâche 1. La même constatation est valable avec la tâche 2.

Très importante est la remarque suivante : un bloc de programme réentrant est protégé en écriture par la protection mémoire. Toutes les tâches utilisatrices ont le droit de le lire mais aucune n'a le droit de l'effacer. Il est inaltérable. Au lieu de le placer sur des mémoires à ferrites on peut donc le mettre sur des selfs ou des capacités. On peut le microprogrammer. En particulier certains modules du software comme les méthodes d'accès aux enregistrements des fichiers sont réentrants. Il est possible de les rendre hautement performants, une partie du software devenant ainsi hardware.

Une propriété qui découle de la réentrance s'appelle la *récursivité* que nous avons vu au chapitre IV.4. Un bloc de programme qui peut interrompre le déroulement de ses instructions à n'importe quel point et ensuite reprendre l'exécution à n'importe quel autre point peut évidemment se rappeler lui-même. Comme dans l'exemple du calcul de « FACTORIELLE » que nous avons précédemment donné il ne travaille pas directement sur des variables, il utilise des index. Il suffit donc pour qu'un programme réentrant soit récursif que l'ordinateur possède une gestion des « piles » d'information (stockage des différents paramètres dont l'adresse de retour à chaque appel du bloc de programme) soit une gestion automatique des piles (par hardware) soit une gestion programmée (par software).

6. — La structuration des blocs de programme

Dans le cadre de la multiprogrammation on peut réserver, demander de la mémoire, des blocs de mémoire pour une tâche à 4 moments bien distincts :

— A la compilation (les instructions, les zones générées par le compilateur vont prendre une certaine place).

— A la phase « translation-éditions des liens » (en réunissant les différents blocs compilés on peut ajouter par exemple des blocs venant d'une bibliothèque).

— Au moment du chargement du programme dans la mémoire (par exemple un bloc réentrant, s'il se trouve déjà en mémoire, on ne le recharge pas, dans le cas contraire il faut le mettre.)

— Au moment de l'exécution, il est possible de demander de la place en mémoire seulement lorsqu'on en a besoin. On appelle cela de l'allocation « dynamique » de mémoire. (Par exemple on ne connaît la dimension d'une « pile » pour un bloc de programme récursif qu'au moment de son exécution.)

A partir de ce principe on divise les structures de programmes en 4 catégories.

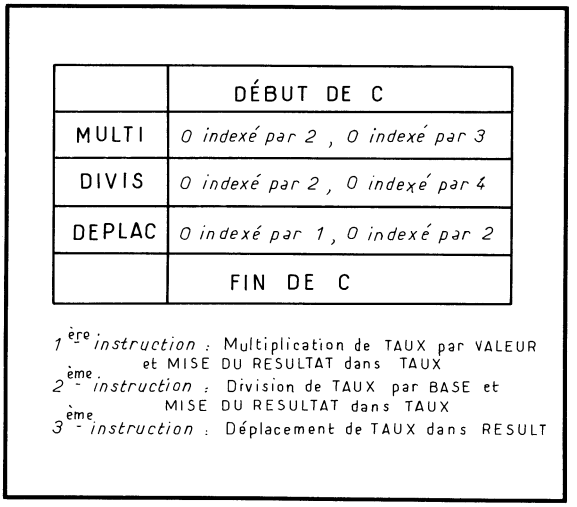


Fig. 7-6. — Écriture d'un bloc de programme réentrant.

6.1. — LA STRUCTURE SIMPLE

Les différents blocs de programme coexistent déjà à la fin de la phase : « translateur-éditeur de biens ». Tous les symboles externes sont satisfaits. Au moment du chargement tous ces blocs sont placés en mémoire en même temps. Au moment de l'exécution l'ensemble ne forme qu'une seule tâche. On passe d'un bloc à l'autre par un système de zones de sauvegarde des registres.

6.2. — LA STRUCTURE A RECOUVREMENT

Ici aussi, au niveau du « translateur-éditeur de liens » tous les symboles externes sont satisfaits, le tout à l'exécution ne donne qu'une seule tâche. Par contre les différents blocs de l'ensemble ne sont pas chargés en même temps.

Il y a toujours un bloc en mémoire qui contient les renseignements nécessaires à tous les autres blocs : variables, zones communes de travail, d'E/S et aussi la table indiquant où il faut charger les autres blocs. Ce segment (les différents blocs de programmes s'appellent ici des segments) est le segment

racine. En cours d'exécution on peut charger un segment à la place d'un autre. On distingue 2 types de structures à recouvrement :

La structure arborescente

Dans l'exemple de la figure 7.7, le segment racine O est toujours en mémoire. A sa suite on peut placer soit le segment A, soit le segment D.

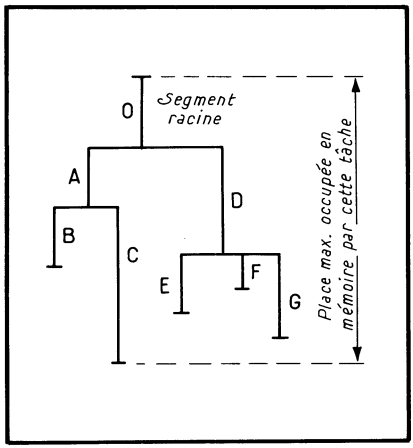


Fig. 7-7. — Structure arborescente d'un bloc de programme.

Si le segment A est en mémoire on peut placer soit le segment B soit le segment C.

Si par contre c'est le segment D ce sera E, F ou G.

La structure par région

Prend un peu plus de place mais est plus souple que la précédente.

On divise la place réservée à la tâche en région de mémoire. Dans notre exemple de la figure 7.8, on a 3 régions. Dans la 1re région on place le segment racine O toujours là, dans la 2e région on y met soit A; soit D, dans la 3e ce sera B ou C ou E ou F ou G.

L'avantage des structures à recouvrement est le gain de place en mémoire, par contre il ne faut pas avoir à recharger des segments sans cesse, car alors cela se traduit par une perte de temps.

6.3. — LA STRUCTURE DYNAMIQUE SÉQUENTIELLE

Au niveau de la phase « translation-édition des liens » certains symboles externes peuvent très bien ne pas être satisfaits. Ils le seront soit lors du chargement, soit lors de l'exécution, au moment où on en a besoin, sous le contrôle de la gestion des tâches. L'ensemble des blocs de programme constitue une seule tâche.

Si un symbole externe doit être satisfait au chargement ou à l'exécution 3 cas sont possibles :

— Ce symbole externe se rapporte à un bloc de programme utilisable une fois, il faut charger ce bloc même s'il y est déjà.

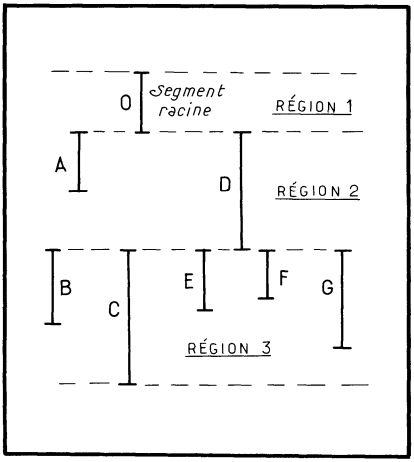


Fig. 7-8. — Structure par région d'un bloc de programme.

— Ce symbole externe se rapporte à un bloc de programme réutilisable séquentiellement. Si ce bloc est déjà en mémoire, on a deux solutions : Ou bien attendre que la tâche qui l'utilise actuellement ait fini de s'en servir, ou bien charger immédiatement une nouvelle copie de ce bloc en mémoire. (On peut limiter le nombre de copie d'un même bloc).

— Ce symbole externe se rapporte à un bloc réentrant. Il est inutile de le charger en mémoire s'il y est déjà.

Le terme « dynamique » signifie aussi rendre la mémoire disponible dès qu'on n'en a plus besoin.

6.4. — LA STRUCTURE DYNAMIQUE PARALLÈLE (MULTI-TACHES)

Cette structure possède les mêmes priorités que la précédente mais en plus l'ensemble des blocs de programme constitue plusieurs tâches.

Supposons par exemple, figure 7.9, un morceau d'organigramme constitué par 3 traitements et 2 tris. On effectue le traitement 1 puis le traitement 2, puis successivement chacun des 2 tris et enfin le traitement 3.

Avec le remplacement des bandes par des mémoires à accès sélectif, les deux tris peuvent être supprimés et les traitements 1, 2 et 3 ne forment plus qu'un seul ensemble de blocs de programme.

A l'intérieur de cet ensemble on peut décider d'exécuter en même temps le traitement 1 et le traitement 2, en les déclarant comme deux tâches différentes.

La première qui arrive au point commun, début du traitement 3, attend l'autre.

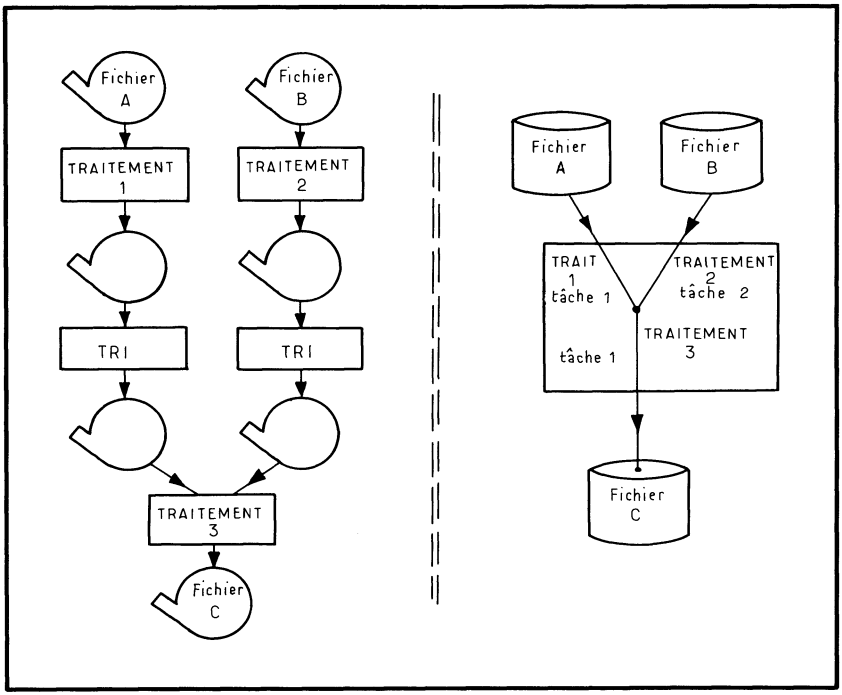


Fig. 7-9. — La structure dynamique parallèle et les mémoires à accès sélectif peuvent permettre d'exécuter en même temps le traitement 1 et le traitement 2.

On gagne ainsi un temps considérable dans l'exécution de cet ensemble.

Cette structure est couramment utilisée dans le traitement en temps réel. On verra qu'il y a une tâche-mère qui appelle des tâches-filles.

7. — La gestion des tâches

Au moment où la gestion des travaux présente à la gestion des tâches une certaine étape d'un travail, celle-ci attribue à la nouvelle tâche un « bloc de contrôle de tâche » qui va contenir les différents renseignements : priorité, état de la machine et contenu des registres lors de la dernière interruption de cette tâche. La gestion des tâches lui attribue aussi un certain nombre de positions de mémoire.

Ainsi donc il y a en même temps dans la mémoire plusieurs tâches qui se partagent les ressources de l'ordinateur : les E/S, les blocs de programme

réentrants et réutilisables séquentiellement, les ferrites de la mémoire principale. Chacune de ces tâches a un bloc de contrôle et le moniteur connaît l'emplacement de ces blocs.

Une tâche peut : — être « *active* », c'est elle qui a le contrôle de l'unité centrale (il n'y en a qu'une à la fois à être dans cet état); — être « *prête* », si le moniteur lui passe la main elle peut dérouler ses instructions; — être « *en attente* », elle attend qu'une E/S soit terminée, ou bien qu'une autre tâche ait fini d'utiliser un bloc de programme réutilisable séquentiellement, ou bien qu'une partie de la mémoire soit libre car elle a demandé dynamiquement le chargement d'un bloc de programme et il n'y a pas de place. Lorsqu'une tâche est « *achevée* » la gestion des tâches la fait disparaître de l'ordinateur en le signalant à la gestion des travaux.

Pour chaque ressource : E/S, bloc de programme réutilisable séquentiellement, occupation mémoire, le moniteur crée une file d'attente des tâches dans laquelle figurent les adresses des différents blocs de contrôle de tâche rangées par ordre de priorité. Lorsqu'un événement a lieu : fin d'E/S, fin d'occupation mémoire, fin d'utilisation d'un bloc de programme réutilisable en séquence, le moniteur donne l'utilisation de la ressource à la tâche la plus prioritaire (parmi celles qui attendaient cet événement).

Dans le bloc de contrôle de chaque tâche, il y a un compteur (géré par le moniteur) indiquant le nombre d'événements que la tâche attend. Lorsqu'une ressource est attribuée à la tâche on décrémente de 1 ce compteur. Lorsque ce compteur est nul la tâche est « prête » et l'adresse de son bloc de contrôle est placée dans la file d'attente (par ordre de priorité) des tâches prêtes. Après chaque événement, le moniteur donne dans cette file d'attente le contrôle à la tâche la plus prioritaire. Cette tâche devient la tâche « active ».

La mesure du temps

Il est possible de mesurer le temps pris par une tâche. L'ordinateur possède une horloge interne. Lorsque le software donne le contrôle à une tâche, l'horloge interne fait en même temps progresser un compteur dans le « bloc de contrôle de cette tâche ». A tout moment on peut donc savoir pendant combien de minutes une tâche a eu le contrôle de la mémoire principale. On peut décréter qu'une tâche donnée ne doit pas durer plus de n minutes. Si le nombre n est dépassé la tâche est considérée comme « terminée ». Cette mesure de temps est aussi utile pour la facturation des différents travaux.

8. — La gestion des tâches vis-à-vis de la gestion des données

8.1. — LES ZONES D'E/S

La gestion des tâches s'occupe aussi de la réservation en mémoire des zones d'entrée-sortie. On distingue 3 étapes dans l'étude de ces zones.

La réservation des zones d'E/S

Le programmeur a le choix entre :

— réserver ses zones d'E/S une fois pour toutes au niveau de la compilation. Il peut déclarer un groupe de zones données pour un seul fichier, ou pour plusieurs fichiers;

— réserver ses zones d'E/S au début de l'exécution (au moment de ce que l'on appelle l'ouverture de fichier);

— réserver ses zones dynamiquement avant chaque entrée-sortie. Cela est très utile lorsqu'on travaille en recherche au hasard avec les mémoires à accès sélectif. Ces recherches sont malgré tout assez longues par rapport au temps de la mémoire principale et il n'est pas nécessaire, par exemple, entre 2 mises à jour, de conserver à la tâche ces zones de mémoire.

La structuration des zones d'E/S

Les zones d'E/S sont structurées suivant les formats des différents enregistrements. Cela peut être fait soit par le programmeur, soit (automatiquement) par le software.

La technique de travail sur les zones d'E/S

Il faut choisir une technique parmi celles que nous avons déjà vues dans le système d'exploitation séquentiel : à l'aide d'une zone de travail, à l'aide d'une indexation, ou bien à l'aide d'une technique cyclique. (Voir chapitre VI-2.)

8.2. — L'IOCS PHYSIQUE ET L'IOCS LOGIQUE — (INPUT-OUTPUT-CONTROL-SYSTEM)

Le programmeur a accès à ces 2 IOCS. Il peut choisir soit l'un, soit l'autre.

S'il utilise l'IOCS logique, le software lui fournit à chaque fois qu'il le demande un enregistrement logique. Il raisonne avec les bandes, et les mémoires à accès sélectif, comme avec les cartes. Le software se charge du dégroupage, des contrôles mais aussi il gère une file d'attente des zones d'E/S pour chacun des fichiers.

S'il utilise l'IOCS physique, le programmeur doit lui-même dégrouper ses enregistrements. Il doit aussi s'occuper de la synchronisation de sa tâche :

S'il écrit en autocodeur, pour chaque ordre physique il doit avoir 3 macro-instructions, comme l'indique la figure 7.10. La première est nécessaire pour lancer l'ordre. Ensuite il peut écrire des instructions qui n'ont pas besoin du résultat de cet ordre. Arrive un point dans le programme, où pour continuer il faut que l'ordre d'E/S soit terminé. Le programmeur écrit une macro-instruction qui place sa tâche en attente de cet événement. Lorsque la fin d'E/S arrive il faut contrôler cette E/S. (L'IOCS physique ne se charge que d'une des parties des contrôles. Très souvent, dans la pratique, les 2 macro-instructions d'attente et de contrôle sont confondues en une seule.)

Supposons un programme effectuant une reproduction de fichier : une lecture, une écriture, une lecture, une écriture... Le programmeur désireux d'utiliser l'IOCS physique (parce que par exemple ses enregistrements sont de formats absolument quelconques) va d'abord écrire : comme dans la figure 7.11,

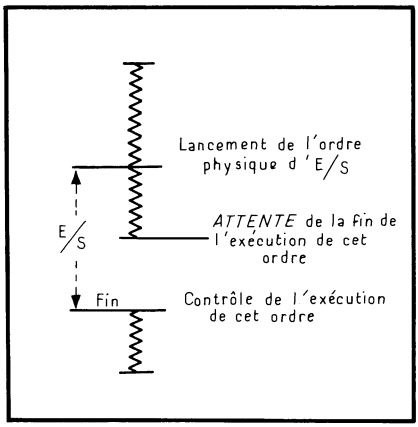


Fig. 7-10. — Pour exécuter une instruction...

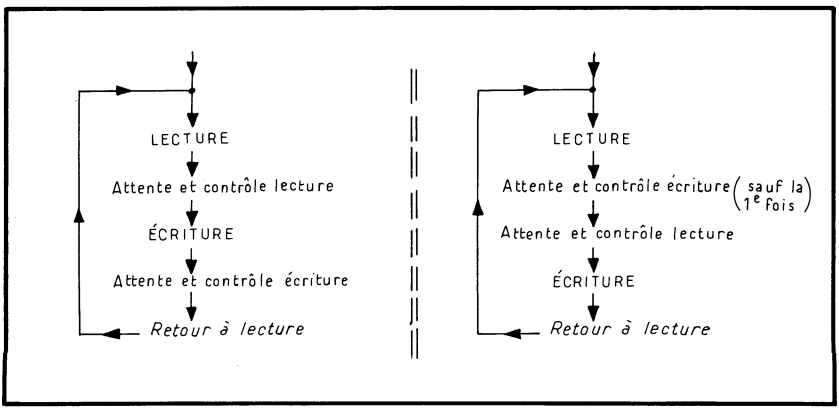


Fig. 7-11. — A gauche le programme n'est pas optimisé. A droite il lit sans attendre la fin de l'écriture de l'enregistrement précédent.

chaque ordre suivi de l'attente, et contrôle de cet ordre. Il s'aperçoit alors qu'il n'a pas besoin de la fin de l'écriture pour lancer l'ordre de lecture, et il modifie en conséquence son ordinogramme.

En agissant de cette manière on optimise la tâche, ce qui ne veut pas dire que l'on optimise l'ensemble de la multiprogrammation puisque de toutes façons les temps d'attente ne sont pas perdus.

Un exemple de synchronisation de tâche est ce qu'on appelle « *l'accès direct asynchrone* » : une tâche demande plusieurs enregistrements disques, elle lance les déplacements de plusieurs bras en même temps. Elle traitera ensuite ces enregistrements dans l'ordre où elle les obtiendra, celui trouvé le plus vite d'abord.

8.3. — L'EXCLUSIVITÉ D'UNE RESSOURCE

Il peut arriver que 2 tâches utilisent le même fichier, par exemple la première y effectue une mise à jour et la deuxième une interrogation. Dans le cadre de la multiprogrammation, et aussi de l'accès au hasard, on ne peut pas savoir si un enregistrement donné sera mis à jour avant d'être interrogé, ou l'inverse. On peut désirer un de ces deux ordres et l'imposer au système :

Soit au niveau du « batch » d'entrée de la gestion des travaux, en indiquant dans une des cartes de commande d'un travail : « je me réserve l'exclusivité de ce fichier ». Pas un seul autre travail ne pourra utiliser le fichier, avant que ce travail là soit terminé.

Soit au niveau de l'enregistrement, pendant une mise à jour. Pour mettre à jour un enregistrement sur une mémoire à accès sélectif, il est nécessaire d'abord de le lire, ensuite de le modifier dans la mémoire principale, puis ensuite de la réécrire. La tâche a la possibilité de se réserver l'exclusivité d'un enregistrement entre cette lecture et cette écriture.

Gestion des travaux, gestion des tâches, gestion des données et structuration des programmes constituent les 4 parties du software sur un ordinateur de 3^{e} génération. Nous avons survolé ici toutes ses possibilités en gestion classique. Quelques années seulement se sont écoulées entre l'apparition des bandes magnétiques et du software et l'avènement de la multiprogrammation, c'est tout simplement vertigineux.

CHAPITRE VIII

LE TRAITEMENT EN TEMPS RÉEL ET LA BANQUE DES DONNÉES

La gestion classique consiste à stocker un grand nombre d'informations et à effectuer seulement ensuite le traitement, cela exige un délai de réponse assez grand. Le traitement en temps réel, lui, traite les informations au fur et à mesure qu'elles arrivent. Il doit tenir compte de deux règles : les messages arrivent d'une manière aléatoire, les réponses doivent être données au bout d'un temps assez court.

Imaginons un stock traité en gestion classique tous les mois. Le réapprovisionnement ne sera déclenché qu'à la fin de chaque mois. Avec le traitement en temps réel, le réapprovisionnement se déclenche au fur et à mesure des besoins. On peut ainsi, en diminuant les délais, supprimer près d'un mois de stock.

1. — Fonctionnement du traitement en temps réel dans le cadre d'un système d'exploitation séquentiel

Les fichiers à traiter ou à consulter sont placés sur une mémoire à accès sélectif. Un enregistrement est accessible directement car les fichiers sont organisés soit en accès direct, soit en séquentiel indexé. Les volumes qui contiennent ces fichiers sont montés en permanence sur l'ordinateur (pendant les heures où ils sont susceptibles d'être interrogés ou d'être mis à jour).

L'ordinateur exécute en permanence des travaux classiques, ou bien s'il n'exécute rien il se place dans un état particulier qu'on appelle état d'attente.

Lorsque sur un pupitre placé à distance ou non, un opérateur appuie sur la touche « demande d'interrogation » il déclenche dans l'ordinateur une interruption. Cette interruption conserve l'état machine en cours et, en mettant dans les registres de l'unité centrale un nouvel état machine, se branche à un point particulier du moniteur. Celui-ci examine la cause de l'interruption et constate que c'est une demande d'interrogation venant de telle ligne.

Deux cas sont possibles :

— Ou bien la mémoire est assez grande et on a en permanence en fin de mémoire principale un programme spécial pour le traitement en temps réel. Le moniteur passe alors le contrôle à ce programme qui va lire le message envoyé par l'opérateur. En principe c'est un numéro indicatif d'un fichier. En fonction de l'organisation du fichier correspondant, notre programme va calculer où il doit se trouver. Il va positionner la tête de lecture-écriture sur cet endroit, puis lancer l'ordre de lecture. Si l'enregistrement ayant ce numéro d'indicatif ne se trouve pas dans le fichier, le programme répond au pupitre de l'opérateur « numéro inconnu ». Si par contre il le trouve, il envoie sur la ligne tous les renseignements qu'il contient. L'opérateur peut alors désirer ajouter ou mettre à jour cet enregistrement. Il le signale par un code spécial, puis il envoie les nouveaux renseignements. Le programme met à jour en conséquence le fichier (on dit très souvent qu'il s'agit d'un fichier dynamique), et (précaution indispensable) il conserve une trace de cette modification. Ensuite il rend le contrôle au moniteur qui restaure l'état de la machine et on continue le travail classique (de routine) en cours.

— Ou bien le programme ne tient pas en permanence en mémoire, car il est trop important pour celle-ci. Il est nécessaire avant de l'exécuter d'aller le chercher. Pour cela le moniteur commence par faire un point de contrôle (ou point de reprise). C'est-à-dire qu'il vide le contenu de la mémoire et de l'état machine sur une bande ou sur un disque, puis il charge le programme spécial à partir d'un volume connu de lui. Ensuite on exécute ce programme de la même façon que précédemment. Lorsqu'on arrive à la fin de ce programme on réinitialise la mémoire et l'état machine à partir du point de contrôle. On continue ensuite le travail classique.

2. — Fonctionnement du temps réel dans le cadre de la multiprogrammation

Le fonctionnement précédent suppose une densité de messages à recevoir très faible, et ne traite pas l'interférence entre ces messages : il ne faut pas qu'un message arrive pendant que l'on est en train d'en traiter un autre.

Avec des systèmes à plusieurs lignes de communication, où la densité de demandes est très importante il est nécessaire de créer des files d'attente. En principe l'ensemble des blocs de programme chargé du traitement en temps réel est divisé en 4 parties :

— Les zones d'entrée-sortie nécessaires pour communiquer avec les différents terminaux et aussi avec les mémoires à accès sélectif.

— Les files d'attente : la file d'attente des messages que l'on vient de recevoir; la file d'attente des réponses prêtes à être envoyées, la file d'attente pour l'interrogation et la mise à jour des mémoires à accès sélectif.

— Les blocs de programme qui sont toujours en place en mémoire. Ils forment le noyau permanent de l'ensemble (tâche-mère).

— Les blocs de programme appelés dynamiquement par le noyau permanent, lorsqu'il en a besoin (tâches-filles).

En multiprogrammation cet ensemble est le plus souvent structuré en dynamique parallèle (plusieurs tâches).

Cette division en plusieurs tâches est souvent nécessaire car les temps d'E/S sont relativement longs. Il y a la réception et l'envoi des messages à distance tributaires de la vitesse des lignes de transmission. D'autre part, l'interrogation des fichiers sur mémoire à accès sélectif a lieu par des recherches au hasard. Les gestions des différentes files d'attente peuvent se faire indépendamment les unes des autres. Ce qu'il faut c'est synchroniser ces différentes gestions entre elles. Il s'agit là d'une technique en plein développement.

Cela nous amène à expliquer un peu plus la structure dynamique parallèle : comment peut-on synchroniser 2 tâches? Lorsqu'une exécution est divisée en deux tâches, 2 cas peuvent se produire : (la tâche-mère est celle qui appelle la tâche-fille).

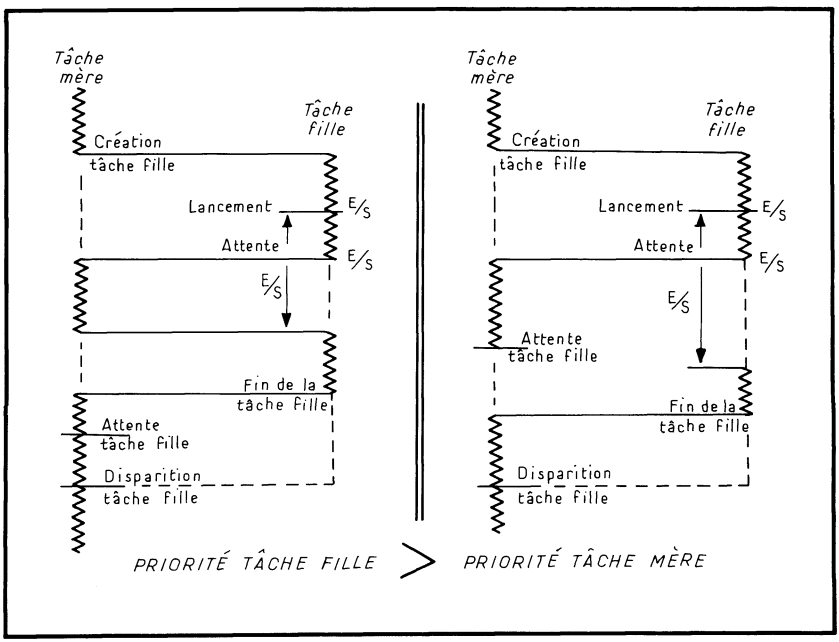


Fig. 8-1. — Synchronisation entre une tâche fille et une tâche mère, la tâche fille exécutant une entrée/sortie.

La tâche-fille est de priorité supérieure à la tâche-mère

Lorsque la tâche-mère (fig. 8.1) crée la tâche-fille, cette dernière prend immédiatement le contrôle de l'unité centrale (elle est plus prioritaire). Elle déroule ses instructions jusqu'à ce qu'elle ait besoin d'attendre la fin d'une E/S pour continuer. Elle se met « en attente » et le contrôle est redonné à la tâche-mère. A la fin de cet E/S on revient dans la tâche-fille qui continue. Lorsqu'elle se termine elle redonne le contrôle à sa tâche-mère qui peut alors signaler au moniteur que sa tâche-fille est terminée.

Il est nécessaire d'avoir une synchronisation entre les 2 tâches. Par exemple ici il y a un point de la tâche-mère où il faut que la tâche-fille soit terminée. Si on rencontre ce point pendant l'E/S de la tâche-fille, les deux tâches sont mises « en attente ». La tâche-fille attend la fin de l'E/S et la tâche-mère attend la fin de la tâche-fille.

La tâche-fille est de priorité inférieure à la tâche-mère

Lorsque la tâche-mère (fig. 8.2) crée sa tâche-fille, elle conserve le contrôle de l'unité centrale. A un moment donné, elle doit attendre la fin d'une E/S.

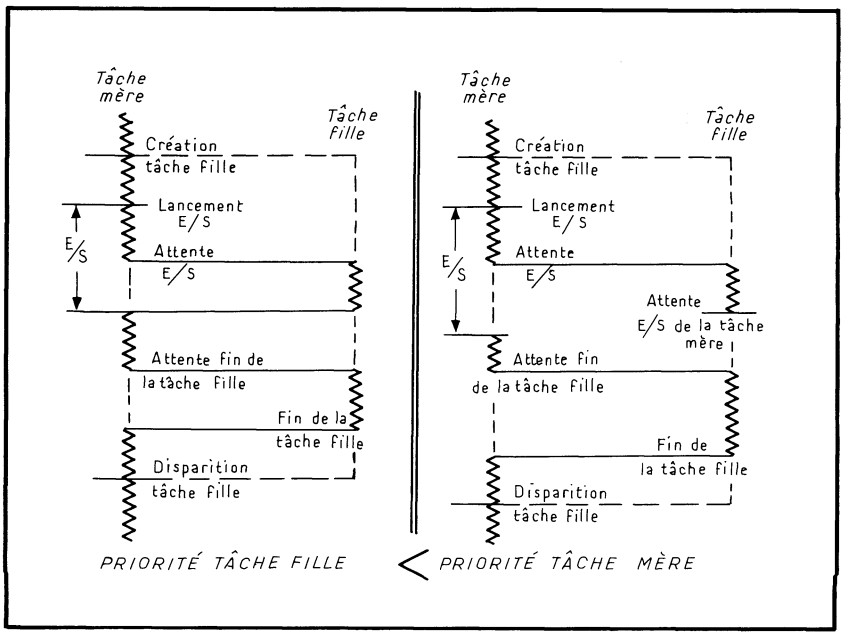


Fig. 8-2. — Synchronisation entre une tâche fille et une tâche mère, la tâche mère exécutant une entrée/sortie.

Le contrôle est donné à la tâche-fille qui s'exécute jusqu'au moment où la fin d'E/S arrive (on suppose ici que la tâche-fille n'est pas terminée à ce niveau).

On continue alors à dérouler les instructions de la tâche-mère jusqu'à ce que l'on ait besoin, pour aller plus loin, de la fin de la tâche-fille. On place la tâche-mère « en attente » et on déroule les instructions de la tâche-fille jusqu'à sa fin. Ensuite on retourne à la tâche-mère, celle-ci prévient alors le moniteur qu'il peut faire disparaître la tâche-fille.

Pour la synchronisation de ces 2 tâches on peut placer dans la tâche-fille une attente d'un point donné de la tâche-mère. Par exemple ici la tâche-fille peut attendre, elle aussi, la fin d'E/S de la tâche-mère. Les 2 tâches peuvent alors être « en attente » en même temps.

En principe on donne la priorité la plus faible à la tâche qui a le plus d'E/S. Il s'agit là d'une programmation assez spéciale où l'on doit encore utiliser le langage autococheur. C'est le deuxième âge d'or de la programmation. La figure 8.3 montre un exemple de plusieurs tâches dans le traitement en temps réel.

3. — L'analyse dans le traitement en temps réel

L'analyse dans le traitement en temps réel est assez délicate. On y rencontre à la fois les difficultés de la gestion et du domaine industriel. Ce n'est pas le matériel qui impose l'application, mais l'application qui impose le matériel. Suivant les cas, on peut être amené à utiliser des dispositifs, des organes, des terminaisons qui ne sont pas de « série ».

Les messages à traiter arrivent à l'ordinateur d'une façon essentiellement variable (d'une manière asynchrone) et il faut répondre au bout d'un temps assez court (exigé par le client). L'analyste doit en premier lieu étudier le format des messages : s'ils sont fixes, leur longueur; s'ils sont variables : leur longueur maximum et leur longueur moyenne.

Il faut ensuite étudier la densité du trafic. C'est un point où on risque d'avoir des surprises si on se contente d'observer les besoins actuels. Il faut tenir compte de l'attrait qu'apporte une technique nouvelle. Un agent de planning responsable d'un stock aura tendance à interroger le fichier plutôt que de consulter ses documents où le renseignement figure déjà. Il faut tenir compte des perspectives futures, de l'expansion des besoins, des facteurs extérieurs ou intérieurs, des perturbations capables de modifier ce trafic. Il faut tenir compte des heures de pointe comme des heures creuses.

Ensuite on peut étudier les terminaux, les lignes et les centraux.

Un terminal peut se résumer par exemple à un clavier et une imprimante, mais alors il faut concevoir une mémoire tampon afin de n'envoyer le message sur la ligne que lorsqu'il est complètement frappé par l'opérateur (de façon à ne pas occuper la ligne pendant tout le temps de frappe). L'imprimante sert à recevoir la réponse de l'ordinateur, mais aussi peut servir à imprimer les modifications envoyées par le terminal, dans le cas où le client désire conserver une trace de ce qu'il a envoyé à distance.

Une bonne solution consiste à utiliser une bande perforée. En principe un groupe de terminaux a 2 modes de fonctionnement : le mode « en local »

dans lequel il fonctionne sur lui-même. Par exemple le clavier imprime seulement sur l'imprimante. Le mode « en ligne » dans lequel le groupe de terminaux peut transmettre et recevoir.

Avec un perforateur-lecteur de bande perforée, l'opérateur peut ainsi « en local », à partir du clavier, perforer une bande, puis ensuite « en ligne » transmettre le message en envoyant le contenu de cette bande. On obtient une mémoire tampon à bon compte.

Le nombre de types de terminaux dans ces domaines est très grand et l'analyste doit choisir en tenant compte des facteurs besoins, performances et prix.

Les lignes constituent le gros point noir du traitement en temps réel. Elles sont un frein à son développement car elles ont généralement des vitesses trop lentes par rapport à celles des autres parties de l'ensemble. D'autre part celles qui existent actuellement ont été prévues pour transmettre la parole, elles ne sont pas faites pour les informations digitales. L'unité de vitesse de transmission s'appelle le baud (du nom du grand télégraphiste français Baudot). C'est le nombre d'intervalles élémentaires (ou bits mais attention il faut aussi ajouter les bits nécessaires aux contrôles et à la synchronisation) transmis en une seconde. Avec les lignes télégraphiques on peut atteindre 200 bauds, avec les lignes téléphoniques 600-1200-2400 et même parfois 4800 bauds. Avec des circuits spéciaux permettant une plus grande largeur de bande passante on peut aller jusqu'à environ 50 000 bauds. Pour augmenter encore la vitesse on peut faire appel à la radio et aux satellites artificiels. (A titre de comparaison les canaux des ordinateurs sur quelques centaines de mètres permettent parfois quelques millions de bauds.)

Il faut aussi étudier sous quel code on envoie les messages, quel est le mode de transmission sur ces lignes : un seul sens de transmission (simplex), ou bien dans les deux sens mais pas en même temps (half duplex), ou bien simultanément dans les deux sens (duplex). Il est nécessaire de moduler (soit la fréquence, soit l'amplitude, soit la phase) sur la ligne : modulation à l'émission et démodulation à la réception. L'appareil qui rassemble ces deux fonctions s'appelle un « modem ».

Les centraux dépendent du nombre de lignes et aussi de la sélection des lignes. Le hardware peut par exemple envoyer de temps en temps (cette fréquence dépend de la densité du trafic) un signal à chacune des lignes pour leur demander si elles ont besoin d'envoyer quelque chose. Si la réponse est « non » on passe à la ligne suivante. Si la réponse est « oui » le contrôle est donné à la première qui répond de cette manière. Dans le cas où le signal de sélection tourne toujours dans le même sens, il faut étudier dans quel ordre il faut placer ces lignes.

A l'autre bout, il faut étudier l'organisation des fichiers, et le format des enregistrements. Ils doivent être accessibles directement, donc organisation à accès direct ou organisation en séquentiel indexé.

Il y a aussi le choix de l'ordinateur. Il ne faut pas que le fonctionnement du traitement en temps réel freine trop considérablement les autres tâches. Il y a l'étude du temps et de l'occupation mémoire. Suivant ces 2 facteurs, il peut y avoir conservation du type d'ordinateur dont on dispose, changement de modèle, ou achat d'un ordinateur spécialisé dans le traitement en temps réel.

Avant de mettre en place un tel système, l'analyste doit en étudier la viabilité. Pour cela il dispose de techniques de simulation ou de calcul.

Dans le dossier d'analyse la logique des blocs est plus importante que partout ailleurs, il y a la division en tâches et la synchronisation de ces tâches.

Enfin, précaution indispensable, il faut pouvoir s'y reconnaître en cas de fausse manœuvre dans une mise à jour. Pour cela la duplication de temps en temps des fichiers, et la conservation des modifications depuis la dernière duplication sont indispensables.

4. — La technique du « temps partagé » (« time sharing »)

Sa définition est à la fois simple et ambitieuse : Les ressources d'un ordinateur sont mises à la disposition, non pas d'un, mais de plusieurs utilisateurs. Ceux-ci ne se trouvent pas près de l'ordinateur et ils doivent pouvoir l'utiliser immédiatement lorsqu'ils le désirent.

Celà pose à la fois un problème de traitement à distance, un problème de traitement en temps réel, et un problème de multiprogrammation à cette différence près qu'il faut donner à chacun la même priorité.

On peut les résoudre de deux manières : soit avec un ordinateur dont le hardware est conçu en conséquence, soit avec un ordinateur ordinaire mais possédant alors un software spécialisé. Dans ce dernier cas on peut aussi mettre en place deux ordinateurs reliés entre eux, le premier orienté « périphériques » gère les messages des utilisateurs et présente les tâches à effectuer au deuxième ordinateur, lequel effectue les traitements.

On divise le temps en tranches égales (par exemple de 100 millisecondes) et pendant chaque tranche un utilisateur à tour de rôle et cela d'une manière cyclique a la priorité la plus grande. C'est-à-dire qu'il a le contrôle de l'unité centrale si sa tâche est prête.

Il faut aussi ordonnancer l'occupation de la mémoire. On ne peut pas y mettre, en même temps, tous les programmes et toutes les données. On divise donc les programmes en petits segments (par exemple de 500 instructions absolues). Lorsqu'on donne le contrôle de l'unité centrale à une tâche il n'y a en mémoire que le segment nécessaire. Pendant la tranche de temps alloué la tâche cliente effectue un traitement à partir des instructions de ce segment. Elle arrêtera son travail soit à la fin de sa tranche de temps, soit parce qu'elle a besoin d'une instruction se trouvant dans un autre segment, soit parce qu'elle se met en attente d'un événement (par exemple la fin d'une entrée-sortie). Dans ces trois cas le contrôle de l'unité centrale est donné immédiatement au client suivant et une nouvelle tranche de temps commence. Pendant le même temps on place en mémoire les segments et les données nécessaires aux tranches suivantes.

La translation doit être souple et exige un système d'adressage spécial car les segments peuvent être chargés n'importe où dans la mémoire, c'est-à-dire là où c'est libre. La protection mémoire doit être particulièrement efficace. Les gestions des données et des tâches sont éminemment complexes. Enfin comme plusieurs clients utilisent le système en même temps il faut une très grande fiabilité et des contrôles à tous les étages.

On est amené à une conception concentrique de la mémoire :

— Au centre du cercle la mémoire rapide (temps de base chiffrable en nanosecondes), de dimension relativement modeste, directement reliée aux unités de calculs et de logiques. C'est dans cette zone que doivent se trouver les instructions et les données nécessaires au traitement.

— Autour une mémoire plus lente (temps de base chiffrable en microsecondes) mais beaucoup plus grande, où sont mis en attente les segments et les données qui seront utilisés ultérieurement.

— A la périphérie viennent les entrées-sorties (temps exprimés en millisecondes), les mémoires à accès direct de très grande capacité et les lignes de transmissions reliées aux terminaux, lesquels sont en liaison directe avec les utilisateurs.

Pendant qu'un segment effectue son traitement à l'aide des unités de calculs et de logiques et avec l'aide de la mémoire rapide, il s'effectue des échanges, d'une part entre la mémoire rapide et la mémoire lente, et d'autre part entre la mémoire lente et les unités périphériques. La mémoire lente place dans la mémoire rapide les segments et les données nécessaires aux tranches de temps suivantes. La mémoire rapide donne à la mémoire lente les résultats de ses traitements. La mémoire lente reçoit les messages des utilisateurs et communique avec les mémoires à accès direct. Elle envoie aux utilisateurs les réponses et les résultats. C'est dans cette mémoire lente que la gestion des tâches gère les files d'attente des données et les files d'attente des segments de programme.

L'originalité profonde d'un tel système est que l'ordinateur vu du client peut être complètement différent de ce qu'il est en réalité. On aboutit à la notion de *machine virtuelle.* Par exemple, étant donné la translation et le fait qu'un segment seulement d'une tâche est en mémoire rapide à un même instant, la dimension des programmes n'est pas limitée par la taille de cette mémoire. On est seulement arrêté par la dimension maximale prévue par l'adressage. Pour l'utilisateur tout se passe comme s'il avait une mémoire rapide de cette taille.

Comme sa tâche est activée une fois toutes les n tranches, si « n » est le nombre de clients à l'instant t, il a l'impression d'avoir un ordinateur possédant un temps de base n fois plus long que le temps de base réel. Enfin comme il a accès directement à l'ordinateur, il travaille comme si cette machine était en face de lui. (Il faut aussi tenir compte du temps pris par le software, et des temps de transmission des lignes.)

Le traitement en temps réel à distance permet ce qu'on appelle le « *mode conversationnel* », c'est-à-dire la possibilité pour l'utilisateur d'échanger directement des informations avec l'ordinateur, de modifier les paramètres d'une courbe ou d'un modèle et de voir immédiatement ce que ces modifications apportent aux résultats. On utilise des « langages conversationnels » : où le programmeur n'est pas esclave des déclarations qu'il a faites au début de son programme, ces déclarations se font au moment où il en a besoin, avec le verbe qui les utilise. Il est ainsi possible de compiler un programme phrase par phrase (« compilateur incrémental »). Dès que le programmeur a terminé la frappe d'une phrase sur son clavier l'ordinateur la transforme en langage machine, il signale immédiatement les erreurs et il peut même exécuter cette phrase. Si le résultat n'est pas satisfaisant le programmeur peut annuler sa ou ses dernières phrases et recommencer. La mise au point des programmes a lieu ainsi à une vitesse optimale.

On peut aussi imaginer, à l'aide d'une entrée par écran de télévision, une représentation de la programmation en deux dimensions pour les petits problèmes scientifiques. C'est une question de normalisation des ordinogrammes et des formules mathématiques.

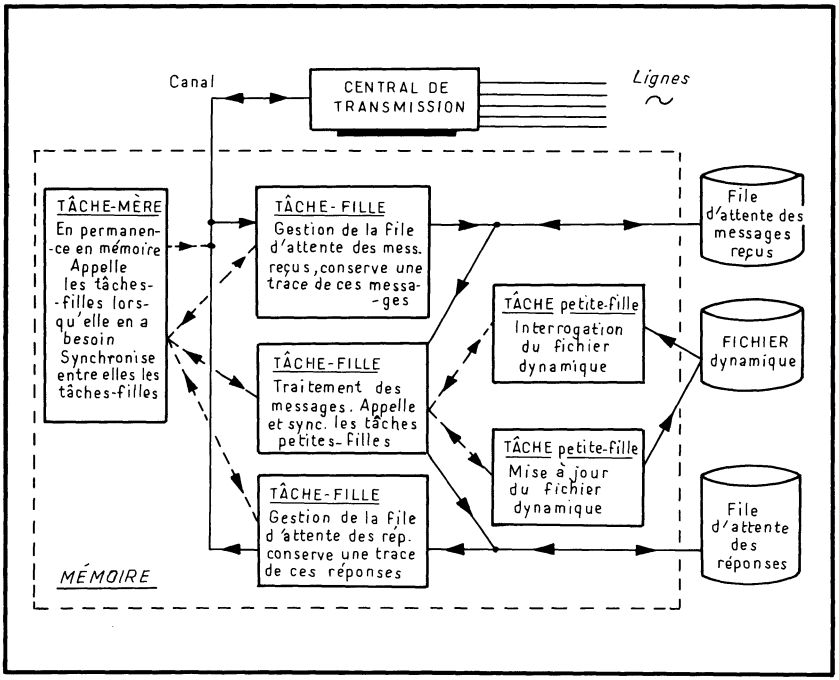


Fig. 8-3. — Exemple « MULTITACHES » dans le traitement en temps réel.

Dans le même esprit ces dialogues avec l'ordinateur peuvent apporter une grande aide à l'enseignement : soit avec « l'enseignement programmé » où tous les chemins possibles de l'élève devant un certain nombre de questions ont été programmés, soit comme exercice pratique où l'élève écrit lui même *le programme en* appliquant la méthode qui vient d'être apprise.

Enfin ce qui suit n'est pas de la science fiction et est parfaitement réalisable. C'est l'ensemble portatif comprenant un téléphone et un télétype. Cet appareil est connectable au réseau commuté par l'intermédiaire d'une prise. Il suffit de composer au cadran le numéro de l'ordinateur et une fois le contact établi d'appuyer sur une touche mettant le télétype en ligne. On frappe alors, avec un langage évolué et simple, ce qu'on désire. L'ordinateur pourra reconnaître l'auteur des messages soit par un système de mots clés, soit par le son de la voix. Un nouveau dialogue commence... et de constater que la télé-informatique permet une centralisation du traitement de l'information tout en permettant une décentralisation des informations et des décisions.

Dans le concept du « mode conversationnel » le langage APL (sigle correspondant au titre modeste de « a programming language »), de l'équipe de KE Iverson, semble occuper une place de choix. Conçu avant tout pour formuler d'une manière logique les problèmes des mathématiques modernes, c'est un outil d'une grande puissance lorsqu'on utilise un clavier possédant les touches des fonctions APL avec un compilateur incrémental.

5. — La banque des données

Le traitement en temps réel exige un certain nombre de fichiers en ligne, accessibles immédiatement. Nous verrons aussi que le concept de « gestion intégrée » demande en plus, des liaisons entre ces différents fichiers. L'entreprise forme un tout, ses informations doivent former un ensemble. C'est pourquoi l'informaticien, au lieu de penser à une organisation fichier par fichier, doit penser à une organisation où tous les fichiers sont liés dans des structures. Si en plus on place toutes ses informations près d'un ordinateur central on réalise « la banque des données de l'entreprise ». C'est le centre du système informatique.

Dans une banque des données :

— chaque information doit être liée à une autre sinon elle serait « un cas isolé » dans l'entreprise;

— chaque information ne doit figurer qu'une seule fois. Dans le cas d'une mise à jour il n'y a à chaque fois qu'une donnée à modifier. Cela est vrai pour les informations proprement dites mais aussi pour les liens qui les unissent. (Ce commentaire bien sûr ne tient pas compte des duplications de fichier nécessaires pour la sécurité.)

5.1. — LES FICHIERS MAITRES ET LES FICHIERS LIENS

On peut diviser les fichiers en deux groupes :

Les fichiers maîtres pour lesquels il est nécessaire d'avoir un accès direct sans passer par des informations intermédiaires contenues dans d'autres fichiers. La connaissance des renseignements contenus dans ces fichiers peut être une fin en soit. Par exemple c'est la recherche des renseignements sur une personne dans un fichier du personnel à partir de son numéro de matricule.

Les fichiers liens : ils contiennent des « informations liens » reliant au moins 2 « informations maîtres ». On n'obtient jamais directement les données contenues dans ces fichiers et leurs connaissances ne constituent généralement pas un but. Par exemple à partir d'un mot clé contenu dans un fichier maître on peut aller dans un fichier lien lequel nous envoie, dans un autre fichier maître, sur les textes ayant un rapport avec le mot clé.

5.2. — LES POINTEURS ET LES CHAINES

Avant d'examiner l'organisation de ces fichiers il est nécessaire d'introduire 2 notions : les pointeurs et les chaînes.

Notion de pointeur : Un pointeur est une zone dans un enregistrement contenant une adresse dans une mémoire à accès sélectif (une adresse disque). A cette adresse se trouve un enregistrement du même fichier ou d'un autre fichier.

Cette adresse est généralement « relative », si elle contient le nombre n elle désigne le n^{eme} enregistrement d'un fichier. La position du pointeur dans l'enregistrement indique quel fichier il faut considérer. A partir de cette adresse logique le software peut calculer l'adresse physique (disque, cylindre, piste). Cette notion « relative » est obligatoire lorsque le software gère lui même, automatiquement, l'emplacement des fichiers. (En effet le programmeur lorsqu'il écrit son programme ignore alors où seront placés ses enregistrements.)

Les enregistrements des fichiers comprennent donc un ensemble de zones constituées par des pointeurs et un ensemble de zones contenant les informations proprement dites. On s'arrange dans la mesure du possible pour travailler avec des enregistrements de longueur fixe et tous les enregistrements d'un même fichier contiennent le même nombre de pointeurs (fig. 8.4).

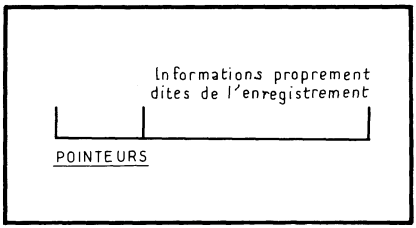


Fig. 8-4. — Dessin d'un enregistrement d'un fichier chaîné.

Notion de chaîne : Une chaîne est une suite d'enregistrements dans un même fichier reliés linéairement par des pointeurs. Dans chaque enregistrement un pointeur contient l'adresse de l'enregistrement suivant de la chaîne (le successeur), comme dans la figure 8.5.

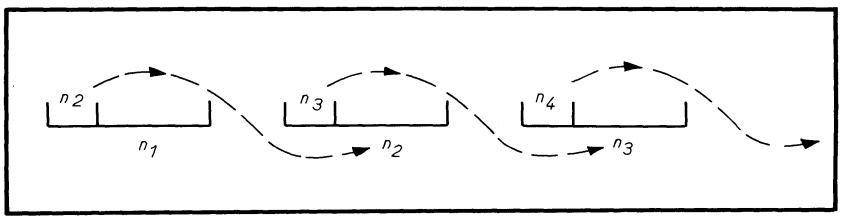


Fig. 8-5. — Chaîne d'enregistrements.

Le plus souvent une chaîne est « bidirectionnelle ». C'est-à-dire que chaque enregistrement possède également un pointeur contenant l'adresse de l'enregistrement précédant dans la chaîne (le prédécesseur). De cette manière lorsqu'on désire supprimer un enregistrement il suffit de prendre les 2 pointeurs qu'il contient pour obtenir le successeur et le prédécesseur. On ressoude la chaîne en modifiant les pointeurs de ces 2 derniers enregistrements.

5.3. — L'ORGANISATION DES FICHIERS

Introduisons au préalable l'idée des *mémoires associatives*. Ce sont des mémoires où il suffit de donner le numéro du contenu pour avoir directement accès à ce contenu. Par exemple il suffit de placer un numéro de matricule dans un des registres de ces mémoires pour avoir accès à l'enregistrement correspondant dans le fichier du personnel. Les mémoires associatives existeront un jour couramment dans le hardware. Elles sont actuellement simulées par software (par exemple à l'aide de l'organisation séquentielle indexée).

L'organisation des fichiers maîtres correspond à l'idée des « mémoires associatives », L'organisation des fichiers liens est en accès direct sous la dépendance des pointeurs contenus dans les enregistrements des fichiers maîtres. Donnons une solution astucieuse et performante à la fois en occupation disque et en temps d'accès.

L'organisation associative des fichiers maîtres est réalisée avec l'aide de 2 fichiers :

un fichier index organisé en séquentiel indexé. Chaque enregistrement contient uniquement le numéro du contenu et un pointeur contenant une adresse dans le deuxième fichier;

un fichier des données organisé en accès direct mais avec une optimisation de la place occupée sur les disques.

Au moment de la création du fichier l'ensemble est chargé séquentiellement suivant l'organisation séquentielle indexée du fichier index. Les enregistrements du fichier des données sont placés séquentiellement sur disque case par case et on place dans les pointeurs du fichier index les numéros des cases correspondantes. L'espace disque occupé par le fichier des données est ainsi entièrement utilisé. Le fichier index qui est, par rapport aux données, un petit fichier est géré comme un fichier séquentiel indexé avec des emplacements réservés pour les enregistrements en addition. Lorsqu'on ajoute un enregistrement le fichier index place son numéro en addition et met parallèlement la donnée correspondante à la suite du fichier des données.

Lorsqu'on désire supprimer des enregistrements on gère le fichier index comme dans l'organisation séquentielle indexée et on marque un emplacement vide dans la case correspondante du fichier des données. A l'intérieur de ce fichier on crée une chaîne des emplacements vides en les reliant par des pointeurs. Lorsque ensuite on ajoute des enregistrements on utilise ces emplacements vides en gérant cette chaîne.

Les fichiers liens sont organisés de la même manière que les fichiers des données des fichiers maîtres. Ce sont les fichiers maîtres qui leur servent de fichiers index.

5.4. — VOICI 3 EXEMPLES :

1er exemple : gestion des commandes en attente dans un problème de distribution

Notre banque des données contient 3 fichiers :

— un fichier maître articles;

— un fichier maître clients;

— un fichier lien des commandes en attente.

C'est l'exemple classique où un fichier lien relie deux fichiers maîtres, figure 8.6.

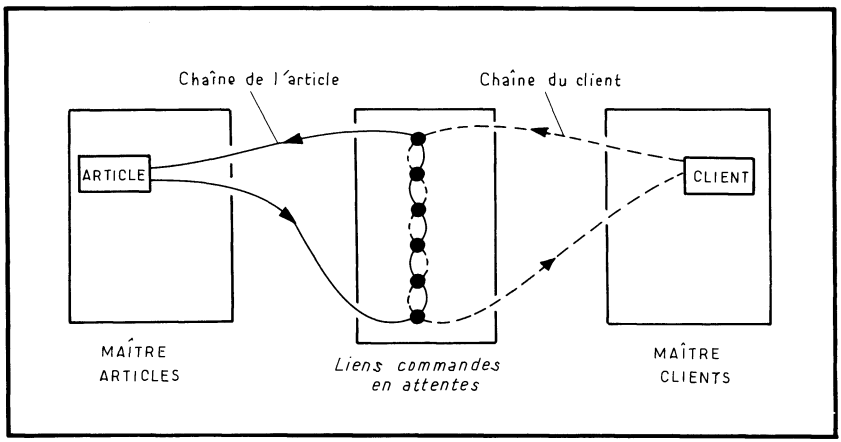


Fig. 8-6. — Les commandes en attente relient le fichier des articles au fichier des clients.

Pour chaque commande en attente on crée dans le fichier lien autant d'enregistrements qu'elle contient d'articles différents. On chaîne entre eux tous les enregistrements d'un même article. Chaque enregistrement du fichier articles contient un pointeur qui indique dans le fichier lien le premier enregistrement de la chaîne correspondante (l'ancre de la chaîne). Un article qui n'a pas de chaîne est un article qui n'est pas demandé.

On exécute la même opération avec les clients; on chaîne entre eux tous les enregistrements liens d'un même client et chaque enregistrement du fichier client possède un pointeur qui désigne le début d'une chaîne. Un client qui n'a pas de chaîne est un client qui ne demande rien.

Chaque enregistrement lien se trouve donc sur 2 chaînes : une pour le client et une pour l'article.

Cette banque des données permet :

— une gestion des commandes en attente par client. A partir d'un numéro de client il est possible d'obtenir la liste de ses commandes et des articles qu'il attend;

— une gestion des commandes en attente par article. Inversement à partir d'un numéro d'article on peut obtenir tous les clients qui le demandent.

Cela amène quelques remarques :

Généralement à partir d'un client on aboutit à plusieurs articles. Si on utilise le vocabulaire de la « théorie des ensembles » on peut dire que nous exécutons une « correspondance » entre les clients et les articles.

Inversement à partir d'un article on peut trouver plusieurs clients, c'est aussi une « correspondance » — correspondance inverse de la précédente car si un client demande un article inversement on doit trouver que cet article est demandé par ce client. (Il est possible de concevoir des banques de données où cette condition n'est pas remplie.)

Considérons dans le fichier lien la chaîne d'un client et la chaîne d'un article. Ces deux chaînes peuvent :

— ne pas se couper. Dans ce cas l'article n'est pas demandé par le client;

— se couper sur un enregistrement. Cet enregistrement est la commande du client concernant cet article;

— se couper sur plusieurs enregistrements. Dans ce cas le client a commandé plus d'une fois l'article.

Enfin il est possible d'organiser ces chaînes de différentes manières, par exemple en séquence sur la date d'arrivée des commandes.

2e exemple : La recherche documentaire automatique

Considérons d'abord une banque de données avec 3 fichiers :

— un fichier maître des mots clés (descripteurs);

— un fichier lien entre ces mots clés et les textes;

— un fichier maître des textes.

Cette banque des données est organisée de la même manière que dans l'exemple précédent. Pour chaque mot clé on crée dans le fichier lien autant d'enregistrements qu'il y a de textes touchés par lui, un enregistrement par texte intéressé. Chaque enregistrement du fichier lien se trouve à la fois dans une chaîne d'un mot clé et dans une chaîne d'un texte. L'ordinateur peut donc à partir de la frappe, sur un clavier d'un mot clé, ou de plusieurs mots clés reliés par des relations logiques « OU » « ET », donner tous les textes correspondants. Il peut aussi donner la liste des mots clés reliés à un texte.

Une chaîne d'un mot clé et une chaîne d'un texte peuvent :

— ne pas se couper. Dans ce cas le mot clé n'a aucun rapport avec le texte;

— se couper sur un enregistrement. Cet enregistrement est le lien entre le mot clé et le texte. L'utilisateur peut y placer des renseignements précisant la nature de ce lien. Par contre ces deux chaînes ne peuvent pas ici se couper sur plus d'un enregistrement. Un seul lien suffit entre un mot clé et un texte.

Cette banque des données à l'inconvénient d'exiger un dictionnaire des mots clés dans lequel l'utilisateur est obligé de choisir ce qu'il désire avant d'interroger l'ordinateur. De plus il y a la mise en place des fichiers. Les documentalistes doivent eux même sortir la « substantifique moelle » de leurs textes. Ils « indexent » les textes en leur attribuant à chacun plusieurs mots

clés. Il est alors possible à partir de ces renseignements de mettre en place la banque des données précédentes. L'ordinateur exécute une phase de « pré-création » ou création d'un fichier lien à blanc.

Cela va nous renseigner sur la valeur documentaire d'un mot clé, d'un descripteur : c'est-à-dire le nombre de textes auquel il se rapporte. Il suffit de dénombrer les enregistrements de sa chaîne dans le fichier lien. Un mot clé qui se rapporte à un très grand nombre de textes risque de ne pas être efficace car il donne beaucoup trop de réponses. Inversement un mot clé qui ne se rapporte qu'à quelques textes est trop fin. Il faut choisir les mots clés dans un juste milieu.

Mais il y a encore mieux. Il existe entre les mots clés eux-mêmes un ensemble de relations, c'est ce qu'on appelle le « thésaurus ». Il peut y avoir des liaisons hiérarchiques : le sens d'un mot clé englobe les sens de plusieurs mots clés; des liaisons de synonymes : les chaînes du fichier lien sont signalées comme pratiquement identiques; des liaisons de ressemblance : un mot clé possède une certaine analogie avec un autre. Un mot clé peut aussi avoir son contraire.

C'est pourquoi il est possible d'ajouter à notre banque des données un quatrième fichier :

— un fichier lien Thésaurus contenant tous les liens entre les mots clés. Au lieu d'être placé entre deux fichiers maîtres, comme dans les exemples précédents, il n'est utilisé que par un seul fichier. Les deux fichiers maîtres sont confondus en un seul. Notre fichier lien matérialise une « correspondance » de l'ensemble des mots clés sur lui même. (On appelle aussi cela parfois « structure en noyau ».)

Pour chaque mot clé on crée dans ce fichier une chaîne à raison d'un enregistrement par lien avec un autre mot clé. Si on considère deux mots clés leurs chaînes peuvent ne pas se couper, ou bien se couper sur un enregistrement et un seul. Dans cet enregistrement le documentaliste peut ici aussi préciser la liaison.

Il est ainsi possible d'enrichir le vocabulaire sans augmenter considérablement le volume des fichiers. Par exemple prenons un groupe de synonymes, il suffit de les relier à un seul d'entre eux, le plus courant, ce dernier sera le seul à posséder une chaîne dans le fichier lien entre les mots clés et les textes. Cette nouvelle banque de données permet par rapport à la précédente une plus grande souplesse et un véritable dialogue entre un utilisateur et l'ordinateur. A partir de la frappe d'un mot clé on peut obtenir les mots clés auxquels il est lié, sa valeur documentaire. Enfin en groupant plusieurs mots clés on obtient presque des phrases de notre langue..., et on arrive en passant du général au particulier, ou bien par des associations d'idées, à demander et à trouver réellement ce que l'on désire.

3e exemple : Le contrôle de production

Notre banque des données, figure 8.7 contient 4 fichiers :

— un fichier maître produit;

— un fichier lien nomenclature;

— un fichier lien gammes de fabrication (opérations);

— un fichier maître poste de travail.

Intéressons nous d'abord à la « nomenclature ». Un produit est composé de plusieurs produits (des sous-produits) lesquels peuvent à nouveau se décomposer. Il y a ainsi plusieurs étages, plusieurs niveaux dans les produits. Déclarer la nomenclature d'un produit c'est donner la liste de tous les produits qu'il contient.

Autrement dit le fichier lien nomenclature sert à relier des enregistrements du fichier maître produit. Il matérialise une « correspondance » de l'ensemble des produits sur lui-même. Par rapport au fichier thésaurus de l'exemple précédent il n'y a ici que des liaisons hiérarchiques.

On ne relie les produits que niveau par niveau : Pour chaque produit on constitue dans le fichier lien une chaîne des produits de niveau inférieur (chaîne analytique), un enregistrement lien dans cette chaîne par produit de niveau inférieur. De la même manière pour chaque produit on constitue dans le fichier lien une chaîne des produits de niveau supérieur (chaîne synthétique), un enregistrement lien dans cette chaîne par produit de niveau supérieur.

Considérons la chaîne analytique du produit A et la chaîne synthétique du produit B. Elles peuvent ne pas se couper. Dans ce cas le produit B n'est pas dans le produit A parmi les produits de niveau immédiatement inférieur, mais cela ne signifie pas que A ne contient pas B, en effet plus d'un niveau peuvent les séparer.

Nos 2 chaînes peuvent se couper sur un enregistrement lien. Dans ce cas B est un sous-produit, de niveau immédiatement inférieur dans la nomenclature de A. Le contenu de l'enregistrement lien peut préciser cette structure, par exemple dire combien de B il y a dans A, le temps qu'il faut pour fabriquer A une fois que B existe.

En principe nos 2 chaînes ne peuvent pas se couper sur plus d'un enregistrement lien. Un seul lien est nécessaire et suffisant pour exprimer que A contient B, mais il faut penser à l'écart entre la théorie et la pratique. Il existe dans toute fabrication des numéros de version, des mises à jour techniques. Un lien peut être valable jusqu'à telle date, un autre à partir de cette date. Ils seront représentés par des enregistrements différents. D'autre part il existe aussi entre la conception d'un produit et sa fabrication des différences. La nomenclature de l'atelier peut être légèrement différente de celle imaginée par le bureau d'études.

Occupons nous maintenant des trois fichiers : maître produit, lien gamme de fabrication (opérations), et maître poste de travail. Ils sont organisés comme les banques des données des exemples précédents. On décompose la fabrication d'un produit à partir de ses sous-produits de niveau immédiatement inférieur en une suite d'opérations. Cette suite s'appelle la gamme de fabrication du produit. On crée dans le fichier lien gamme de fabrication pour chaque produit une chaîne d'enregistrements à raison d'un enregistrement par opérations. Cette chaîne doit être organisée en séquence dans l'ordre d'exécution des opérations. Ensuite pour chaque poste de travail on crée une chaîne des opérations qui s'exécutent sur ce poste.

Une chaîne d'un produit (sa gamme de fabrication) et une chaîne d'un poste de travail peuvent se couper sur plus d'un enregistrement. Plusieurs opérations d'une gamme peuvent s'exécuter sur un même poste de travail.

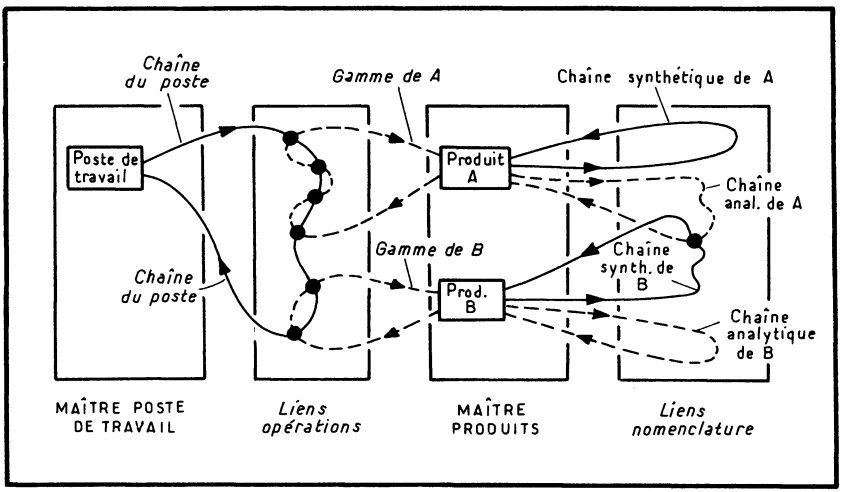


Fig. 8-7. — Banque de données pour un contrôle de production.

Remarque :

Revenons au fichier maître produit. Chaque produit a donc 3 chaînes : 2 dans la nomenclature (une analytique et une synthétique) et sa gamme. Il est possible en plus de chaîner entre eux les produits eux-mêmes dans le fichier maître. En effet il est nécessaire de s'assurer que les nomenclatures n'ont pas des longueurs infinies. C'est par exemple le cas lorsqu'on déclare que A contient B, puis plus loin que B contient A. On commence donc par chaîner entre eux tous les produits qui n'ont pas de chaîne synthétique. Ce sont les produits finis. On marque (par exemple un bit sur la position 1) tous les enregistrements des chaînes analytiques de ces produits. Ensuite on considère les chaînes synthétiques dont tous les enregistrements sont marqués, les produits correspondants sont chaînés entre eux. On marque les enregistrements de leurs chaînes analytiques et on continue l'opération... jusqu'à ce que tous les enregistrements liens du fichier nomenclature soient marqués. (Dans la pratique ce chaînage est établi d'une manière plus simple, car, au moment de la création des fichiers, les nomenclatures sont enregistrées une par une.)

Cette banque des données permet la mise en place d'un véritable système informatique, c'est de la « gestion intégrée » ce n'est pas forcément du « temps réel » :

— *la gestion des nomenclatures*

A partir des numéros de version il est possible de s'assurer une tenue des dossiers techniques. A partir de l'interrogation d'un numéro de produit on obtient au jour le jour sa nomenclature. Inversement il est possible de remonter les chaînes synthétiques et de savoir dans quels produits finis il est utilisé.

— *la gestion des gammes de fabrication*

De la même manière il est possible de gérer les dossiers des gammes de fabrication — d'ajouter dans les chaînes des opérations des relations logiques — si telle opération n'est pas réalisable il faudra exécuter telle autre...

— *le calcul des besoins*

A partir des commandes de produits finis et de ce qui est en cours de fabrication le fichier nomenclature permet de déterminer les programmes de fabrication pour tous les sous-produits. Pour les sous-produits et matières premières venant de l'extérieur on déclanche le réapprovisionnement. Pour les autres produits on lancera la fabrication aux moments voulus.

— *la prévision des charges*

A partir des programmes de fabrication le fichier lien gammes de fabrication (opérations) permet de prévoir dans le fichier maître poste de travail les charges de ces postes. A partir de ces charges il est possible également de prévoir les prix de revient.

— *le suivi de la fabrication*

Aux moments voulus le fichier lien gammes de fabrication permet d'imprimer des bons de travaux où opération par opération l'ordinateur indique ce qu'il faut faire. Le personnel d'exécution complète ces bons en y notant ce qu'il a fait réellement en temps, rendement et consommation. Par un moyen informatique, lecteur de marques ou lecteur optique l'ordinateur relit ces bons et enregistre les écarts entre les prévisions et le réalisé. Il est également possible d'y intégrer la gestion du personnel (paye), la gestion des machines et la gestion des matières consommées.

L'ensemble de tous les fichiers forme une banque des données. Parfois lorsque les problèmes de l'entreprise sont vastes il est possible de faire la distinction entre « base commune des données » et « banque des données ». La base commune des données englobe plusieurs banques des données. Chaque banque des données rassemble toutes les informations d'un secteur, d'un « sous-système » de l'entreprise. Des échanges d'informations entre les différentes banques des données (« interfaces ») sont alors nécessaires.

Une banque des données est décomposable en plusieurs ensembles d'informations, par degré d'accessibilité. Il y a les informations accessibles à tout le monde. D'autres ne le sont que par certains. Certaines informations ne peuvent être modifiées que par tel utilisateur et à tel moment... Pour chaque information d'une banque des données il est nécessaire de définir : qui a le droit de l'interroger et quand; qui a le droit de la modifier et quand. Les contrôles sont plus que jamais nécessaires car une erreur introduite dans une banque des données est difficilement repérable et peu *entraîner d'autres* erreurs sans rapport apparent avec elle.

Cette conception de banque des données est fondamentale. D'une part le software lui-même pourrait constituer un quatrième exemple car il utilise et gère d'une manière très dynamique à l'intérieur de la mémoire une chaîne des tâches, une chaîne des fichiers, une chaîne des espaces occupés de la mémoire..., d'autre part et surtout l'informaticien lorsqu'il étudie un système doit d'abord penser à l'organisation possible de ses fichiers. C'est son premier survol et il est déterminant pour tout le reste.

CHAPITRE IX

LES TRIS

Dans le domaine de l'informatique, le problème des tris occupe une place à part. Leur étude n'est pas indispensable pour le spécialiste amené à utiliser des programmes utilitaires tout faits. Elle constitue malgré tout quelque chose d'agréable pour l'esprit, à comprendre et à expliquer.

Le problème consiste à classer un fichier de n objets amenés pêle-mêle, suivant par exemple leur numéro de matricule croissant. Nous n'avons pas la prétention ici de signaler toutes les méthodes possibles mais seulement de montrer celles qui sont les plus utilisées.

1. — La méthode de la trieuse (fig. 9.1.)

Sur la trieuse du matériel classique, les cartes sont placées dans un magasin de lecture. On dispose d'un balai de lecture (ou cellule photo-électrique) que l'on peut placer sur une des 80 colonnes des cartes. Il y a ensuite autant de cases de réception qu'il y a de perforations différentes dans une colonne.

Par exemple, si on règle la lecture sur la colonne 10 et qu'on lance la lecture des cartes, toutes celles ayant une perforation 1 dans cette colonne tombent en case 1, une perforation 2 en case 2, une perforation 3 en case 3, une perforation 9 en case 9. Si une colonne contient 2 perforations c'est la première lue qui détermine la sélection. Dans chaque case les cartes *restent* dans leur ordre d'arrivée.

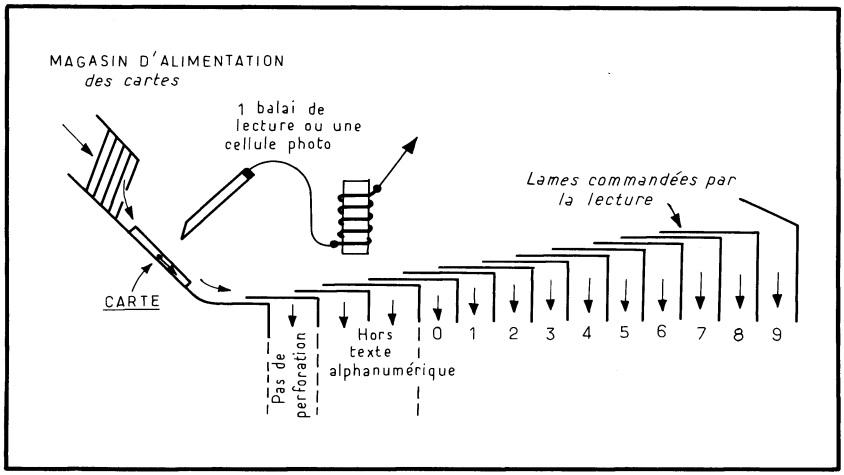


Fig. 9-1. — La trieuse avec son balai de lecture et ses cases de réception.

Avec cette machine, on peut trier de la manière suivante :

Dans un premier passage les numéros sont testés suivant leur unité puis envoyés sur une des dix cases de réception existantes. A la fin de ce passage, les numéros envoyés dans la case 0 sont placés devant ceux de la case 1, ceux-ci devant ceux de la case 2 et ainsi de suite jusqu'à 9. Ils sont pris dans cet ordre pour le 2e passage au cours duquel ils sont testés suivant leur dizaine puis rangés dans l'ordre des cases de réception et ainsi de suite.

Exemple : prenons la séquence 32 - 30 - 21 - 31 - 29, au premier passage on règle sur les unités,

30 tombe en case 0, 21 et 31 en case 1, 32 en case 2, 29 en case 9;

on a l'ordre : 30 - 21 - 31 - 32 - 29;

au 2e passage, on règle les dizaines; 21 et 29 tombent en case 2, 30 - 31 - 32 en case 3;

on a l'ordre : 21 - 29 - 30 - 31 - 32.

Le tour est joué.

Il faut autant de passages que les numéros portent de chiffres significatifs. Cette méthode est très efficace. La difficulté en programmation consiste à matérialiser les cases de la trieuse sans avoir des E/S trop nombreuses.

2. — Méthode théorique donnant le minimum de comparaison

Étant donné plusieurs équipes, comment effectuer le minimum de matchs pour les départager sans avoir d'ex-aequo? On admet ici que le résultat de chacun de ces matchs respecte la force des équipes l'une par rapport à l'autre,

et que l'on a à chaque fois une perte ou un gain. Soit les équipes : A - B - C - D - E et F et supposons que nous ayons la séquence C < B < F < A < D < E, séquence que nous ne connaissons pas au départ et qu'il s'agit de trouver.

Traçons le tableau de la figure 9.2.

	A	B	C	D	E	F
A		1		-1		
B	-1			-1		
C				-1		
D	1	1	1			
E						
F						

1e round	2e round	3e round
1	1	0
-1	-1	-2
0	-1	-1
0	1	3
0	0	0
0	0	0
A/B	C/D	A/D

Fig. 9-2. — Contenu du tableau à la fin du troisième round.

Au premier round nous effectuons le match entre les deux premières A et B. Nous trouvons B < A, nous mettons 1 point pour A (dans la colonne B), — 1 point pour B (dans la colonne A). Après le premier round : A a 1 point, B — 1 point, et tous les autres : 0 point.

Au deuxième round, nous prenons les deux premières équipes du tableau ayant le même nombre de points : ce sont C et D. Nous trouvons C < D donc 1 point pour D et — 1 point pour C. Après ce round, A et D ont 1 point, E et F : 0, B et C : — 1 point.

Pour le troisième round, A et D sont les deux premières ayant le même nombre de points. Le résultat est A < D donc + 1 pour D et — 1 pour A.

Mais ici il faut faire attention à ce qu'on appelle « la transitivité ». En effet, si A < D et si A > B, il est clair que D > B. Cela revient à dire que le 1 placé au croisement de l'horizontale A et de la verticale B, et le 1 placé au croisement de la colonne A et de l'horizontale D génèrent un 1 au croisement de la verticale B et de l'horizontale D.

Par symétrie par rapport à la diagonale principale on a — 1 au croisement de l'horizontale B et de la verticale D. Il faut ensuite vérifier que ces nouveaux 1 et — 1 n'amènent pas à leur tour d'autres 1 et — 1 (transitivité du 2e ordre).

Après le 3e round : D a 3 points, A, E et F ont 0 point, B : — 2, C : — 1.

Pour le 4e round, on aura donc A contre E et ainsi de suite. Finalement après 9 matches on arrive au résultat de la figure 9.3.

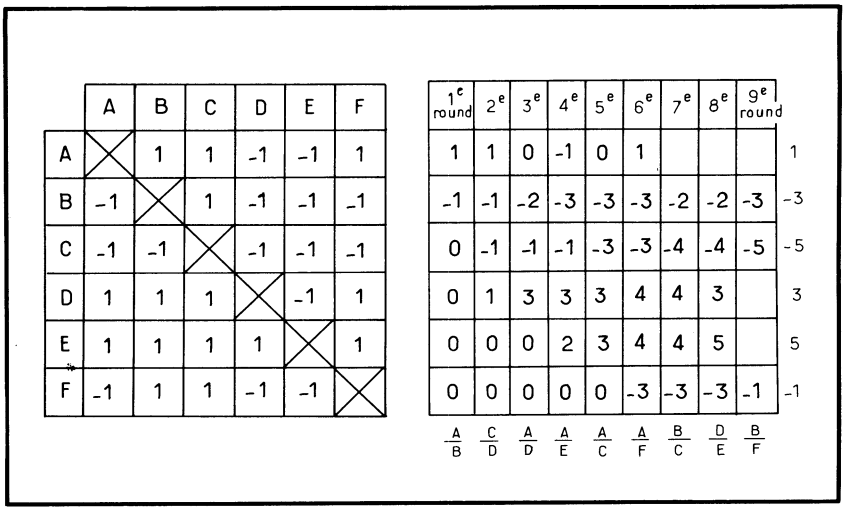


Fig. 9-3. — Résultat après le neuvième round.

A remarquer qu'après le 8e round, nous n'avons pas d'équipe ayant le même total. La plus petite différence est alors 1 entre B et F (on démontre que cette différence ne peut pas être supérieure à 1).

Le résultat donne E = 5, D = 3, A = 1, F = —1, B = —3, C = —5.

Il faut, suivant les cas, 9 ou 10 rounds pour classer 6 objets avec cette méthode.

3. — Les phases d'assignation et de tris internes

Pratiquement on utilise très rarement la méthode théorique précédente : ce ne sont pas les comparaisons qui prennent le plus de temps, il y a aussi le déplacement des informations. D'autre part, en gestion, il n'est pas possible de placer tous les enregistrements d'un seul coup dans la mémoire. Les temps des E/S entrent en jeu, il faut trouver autre chose.

Les programmes de tris sont avant tout des programmes utilitaires, c'est-à-dire qu'ils peuvent fonctionner, non pas dans une, mais dans presque toutes les applications. Il faut donc commencer par leur donner à lire des cartes d'assignation pour leur préciser le travail à faire. Dans ces cartes, il y a certains renseignements sur les fichiers à trier : labels fichiers, format des enregistrements, emplacement des arguments à trier dans ces enregistrements...

En fonction du contenu de ces cartes, le programme de tri que nous avons choisi s'initialise. En particulier, il se réserve des zones d'E/S et des zones de travail.

Après cette initialisation, le programme de tri va tronçonner le fichier à trier en monotonies. Une monotonie est un ensemble d'enregistrements, placés à la suite des uns et des autres, et déjà triés les uns par rapport aux autres. Pour cela il exécute des phases de tri interne.

Au cours de sa phase d'initialisation, il a réservé en mémoire principale une grande zone de travail, zone de tri interne, capable de recevoir plusieurs enregistrements de la bande à trier. Il lit les premiers enregistrements de cette bande et les place à la suite les uns des autres dans la zone de tri interne. Lorsque cette zone est pleine, il trie ces enregistrements. Il obtient ainsi une première monotonie. Il la vide sur une bande ou sur un disque, on verra comment, puis il remplit à nouveau la zone de tri interne en lisant sur le fichier d'entrée les enregistrements suivants. Par un nouveau tri interne, il obtient une 2e monotonie... et ainsi de suite jusqu'à la fin du fichier.

Ces tris internes peuvent s'exécuter de différentes manières, citons deux méthodes (il en existe beaucoup d'autres) :

a) On compare le 1er enregistrement avec, successivement, tous les autres enregistrements. Si on trouve un enregistrement n plus petit que le premier, on les intervertit; c'est-à-dire que l'on place l'enregistrement n à la place du premier, et le premier à la place de l'enregistrement n, puis on continue les comparaisons. A la fin de cette série, l'enregistrement ayant le numéro d'indicatif le plus petit de l'ensemble se trouve à la place du premier. Ensuite, en ne touchant plus à cet enregistrement, on recommence les opérations avec le 2e enregistrement, et ainsi de suite jusqu'à ce que tout soit trié, après $\frac{p\,(p-1)}{2}$ comparaisons s'il y a p enregistrements dans la zone de tri interne.

b) On peut aussi utiliser une méthode arborescente. Pour la simplicité, on suppose ici que le nombre p d'enregistrements est une puissance de 2, par exemple 8 comme sur notre figure 9.4.

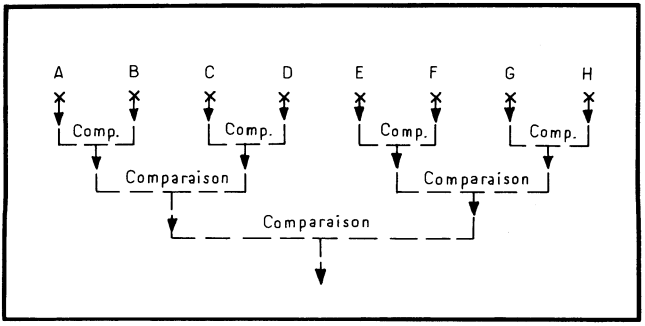


Fig. 9-4. — Méthode arborescente pour les tris internes.

On compare les enregistrements 2 par 2 et on conserve à chaque fois le plus petit, en procédant ainsi étage par étage, on arrive rapidement à sélectionner le premier de la monotonie. On confie cet enregistrement en sortie à l'IOCS logique, puis on lit l'enregistrement suivant à la place de celui que l'on vient de sortir. De deux choses l'une :

1) ou bien l'enregistrement que l'on vient d'entrer a un numéro d'indicatif plus grand que celui que l'on vient de sortir et on recommence les comparaisons en tenant compte de ce nouvel enregistrement. On agrandit ainsi le nombre d'enregistrements dans une même monotonie et on diminue le nombre de monotonies à la fin de la phase de tri interne. Si par exemple, on lit une bande déjà triée, on obtiendra une seule monotonie.

2) ou bien l'enregistrement que l'on vient d'entrer a un numéro d'indicatif plus petit que celui que l'on vient de sortir. Dans ce cas, on ne tient pas compte de ce nouvel enregistrement et on termine la monotonie en triant les enregistrements qui restent à l'aide de la structure arborescente. En prenant le plus petit des deux à chaque palier, en ignorant ceux que l'on vient de sortir.

4. — La réunion des monotonies

Le tri interne précédent a donné plusieurs monotonies, il s'agit maintenant de réunir toutes ces monotonies en une seule, de manière à obtenir le fichier trié. Pour cela plusieurs méthodes, citons :

4.1. — LA MÉTHODE DE LA BALANÇOIRE (FIG. 9.5.)

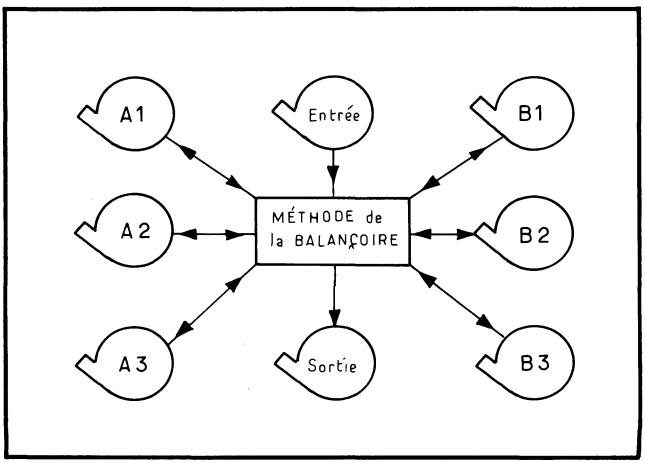


Fig. 9-5. — La méthode de la balançoire.

La phase tri interne ventile les monotonies de la manière suivante, en prenant par exemple 3 bandes de ventilation : A1 - A2 - A3. La première monotonie est mise sur A1, la seconde sur A2, la troisième sur A3, la quatrième sur A1, et ainsi de suite d'une manière cyclique sur A1, A2, A3. Lorsqu'on

arrive à la fin du fichier, on a ainsi n monotonies sur A1, n ou $(n - 1)$ monotonies sur A2, n ou $(n - 1)$ monotonies sur A3.

Ensuite on rebobine à leur position de départ les 3 bandes A1, A2, A3. On dispose aussi de 3 bandes B1, B2, B3. On fusionne les premières monotonies de chacune des 3 bandes A (la première monotonie de A1, la première monotonie de A2, la première monotonie de A3). On obtient ainsi une monotonie égale à la somme de ces 3 monotonies. On la place sur la bande B1. On fusionne ensuite les deuxièmes monotonies de chacune des 3 bandes A et on place le résultat sur B2, puis ensuite les troisièmes sur B3, les quatrièmes sur B1, les cinquièmes sur B2 et ainsi de suite cycliquement sur B1, B2, B3. A la fin de ce passage, on a toujours au $(n - 1)$ près : $\frac{n}{3}$ monotonies sur B1, $\frac{n}{3}$ (ou $\frac{n}{3} - 1$) monotonies sur B2, $\frac{n}{3}$ (ou $\frac{n}{3} - 1$) monotonies sur B3.

Ensuite on rebobine les 6 bandes A1, A2, A3, B1, B2, B3 et on recommence en inversant les rôles des bandes A et des bandes B. A la fin de ce passage on a $\frac{n}{9}$ (au $- 1$ près) monotonies sur les bandes A.

Et ainsi de suite en poursuivant le mouvement de la balançoire : A lecture et B écriture, B lecture et A écriture... on a un moment où le nombre de monotonies sur les bandes d'un des côtés est 1 ou 0. A ce moment il suffit d'effectuer une fusion finale sur la bande de sortie pour obtenir le fichier trié.

Les seuls inconvénients de cette méthode sont les rebobinages des bandes et aussi la nécessité parfois d'effectuer un passage supplémentaire, alors qu'on a plus qu'une seule monotonie sur A ou sur B afin d'obtenir le résultat sur la bande de sortie. (Cette bande de sortie est nécessaire dans le cas où on désire faire du chaînage avec le travail suivant.)

Cette méthode est aussi valable avec les mémoires à accès sélectif. Avec ce support, il est possible de faire des tris uniquement sur les indicatifs accompagnés des adresses des enregistrements. A la fin du tri on va rechercher ces enregistrements.

Note — Le nombre de bandes A ou B s'appelle le nombre de voies, dans l'exemple précédent on avait donc 3 voies. Ce nombre de voies est choisi suivant le nombre de bandes que l'on peut monter, et suivant la dimension du fichier à trier.

4.2. — LA MÉTHODE POLYPHASE

Depuis un certain temps, on a construit des lecteurs de bande capables de lire en lecture arrière (le sens de lecture peut être l'inverse du sens d'écriture). On a imaginé des méthodes de tri capables de profiter de cette amélioration technique.

Avec la méthode Polyphase, on commence par choisir un nombre de voies (qui dépend là aussi du nombre de bandes à monter et de la dimension du fichier à trier). Pour notre exemple, nous prenons : 3 voies (fig. 9.6). A partir de ce nombre le programme construit une table de distribution des monotonies, en prenant le nombre de voies + 1 bande; Ici : 4. On commence par écrire 1 0 0 0

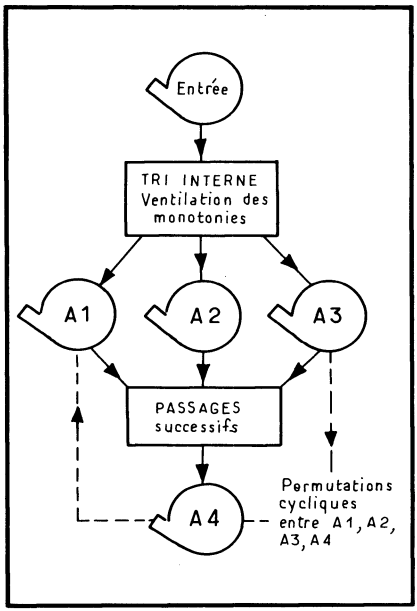


Fig. 9-6. — La méthode polyphase, et la méthode des fusions oscillantes.

C'est la distribution finale : 1 monotonie sur 1 bande et rien sur les autres. A partir de là on passe à l'étage précédent en prenant le plus grand nombre (ici 1) et en l'ajoutant à tous les autres. On remplace d'autre part ce plus grand nombre par 0. On obtient ainsi successivement, la figure 9.7.

1	0	0	0	Etage final
0	1	1	1	Avant dernier étage
1	0	2	2	Avant avant dernier étage
3	2	0	4	"
7	6	4	0	" Chacune des colonnes représentent respecti-vement les bandes
0	13	11	7	" A_1 A_2 A_3 A_4

Fig. 9-7. — Table de distribution des monotonies.

En ventilant les monotonies, le tri interne respecte cette table de distribution. Comme il ne sait pas combien il y a de monotonies en tout, il passe successivement d'un étage à l'autre. Par exemple, à la fin ici on a 17 monotonies, il donne la distribution 7 - 6 - 4. Dans le cas où l'on ne tombe pas

juste sur le nombre de monotonies d'un étage, le tri crée des monotonies fictives (bidon).

Mais ce n'est pas tout, les monotonies sont placées sur chaque bande, la première en séquence croissante, la deuxième en séquence décroissante, la troisième en croissant, la quatrième en décroissant et ainsi de suite (ou bien l'inverse avec la première en séquence décroissante). La ventilation, après chacun des tris internes, met en place tout cela en prévision des passages suivants.

Dans notre exemple, à la fin de cette phase on a ainsi la configuration de la figure 9.8.

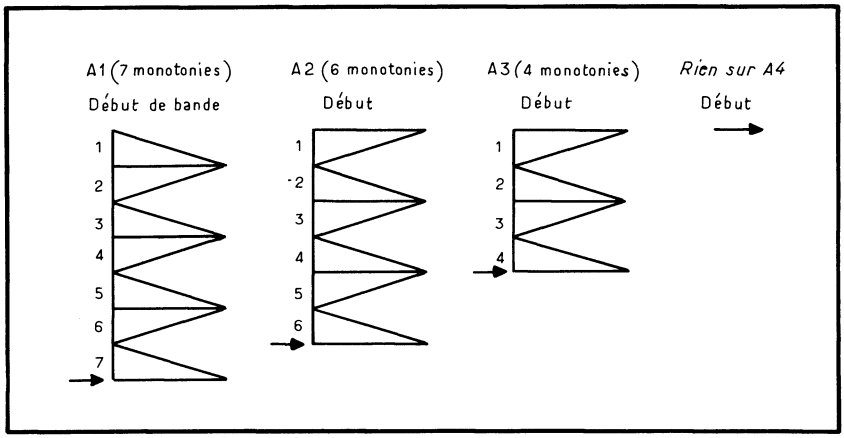


Fig. 9-8. — Distribution des monotonies à la fin du tri interne.

Les triangles indiquent le sens dans lequel les monotonies sont croissantes. Les têtes de lecture écriture sont positionnées à la fin de A1, A2 et A3 (positions indiquées par les flèches) et au début de A4.

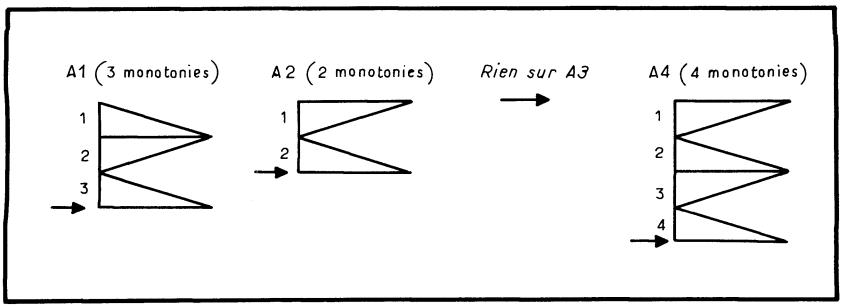


Fig. 9-9. — Distribution des monotonies à la fin du premier passage.

Au premier passage, les bandes A1, A2 et A3 sont lues en lecture arrière, et on va écrire en avant sur A4. On fusionne ainsi en séquence décroissante La septième séquence de A1, sixième séquence de A2, et la quatrième séquence de A3. Cela donne la première séquence de A4. On fusionne ensuite en séquence croissante la sixième séquence de A1, la cinquième séquence de A2 et la troisième séquence de A3. Et ainsi de suite jusqu'à ce que l'on arrive au début de A3, comme dans la figure 9.9.

Distribution 3 - 2 - 0 - 4, on constate ainsi que l'on a changé d'étage dans notre table de distribution.

Au cours du 2e passage, on fusionne en lecture arrière les bandes A1, A2, A4 sur A3. On obtient la figure 9.10.

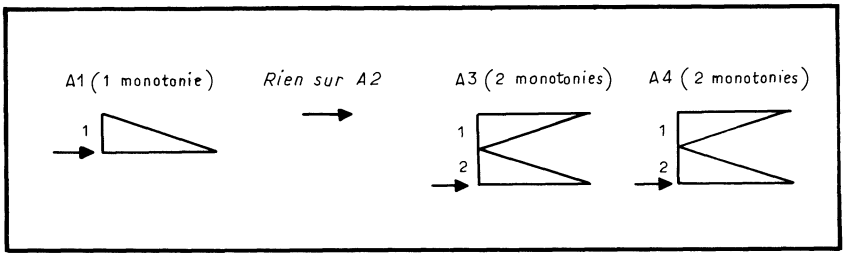


Fig. 9-10. — Distribution des monotonies à la fin du deuxième passage.

Puis on exécute la fusion en lecture arrière de A1, A3 et A4 sur A2 pour obtenir la figure 9.11.

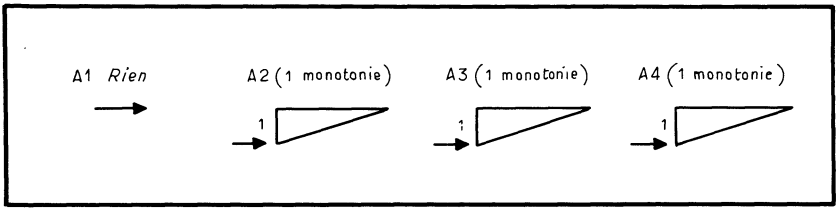


Fig. 9-11. — Distribution des monotonies à la fin du troisième passage.

Il suffit ensuite de fusionner en lecture arrière les bandes A2, A3 et A4 pour obtenir le résultat final, en séquence croissante sur A1. On a effectué tout cela sans un seul rebobinage. Il faut noter aussi le peu de bandes utilisées. Le tri ne peut pas choisir sa bande de sortie sur A1, A2, A3 ou A4 car il ignore au départ à quel étage de sa table de distribution il va s'arrêter. (On peut par contre en prenant une bande supplémentaire obtenir le fichier trié sur cette bande sans avoir de passage supplémentaire.)

4.3. — LA MÉTHODE DES FUSIONS OSCILLANTES

Cette méthode est surtout utilisée sur les gros ordinateurs. Une partie seulement du fichier est considérée à chaque oscillation.

Dans notre exemple, nous prendrons ici aussi 3 voies et le nombre de bandes est encore égal au nombre de voies plus 1. (La disposition des bandes est identique à celle de la méthode précédente).

Dans un premier temps, on place une monotonie sur chacune des bandes A1, A2, A3. On arrête le tri interne. On fusionne A1, A2, A3 en lecture arrière sur la bande A4. On obtient ainsi sur A4 une monotonie égale à la valeur de 3 monotonies (figure 9.12).

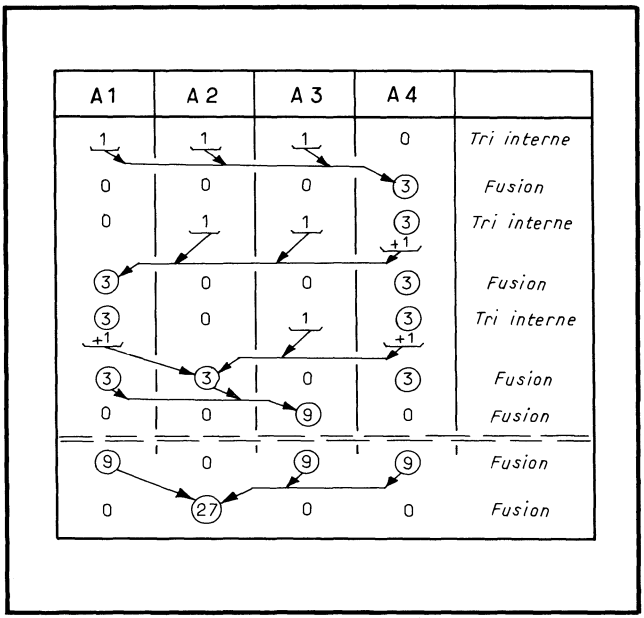


Fig. 9-12. — Déroulement des tris internes et des fusions de la méthode des fusions oscillantes.

On permute ensuite, d'une manière cyclique A1, A2, A3 et A4. Le tri interne reprend et on place une monotonie sur chacune des bandes A2, A3, A4.

On arrête le tri interne et on fusionne en lecture arrière les monotonies que l'on vient de ventiler sur la bande A1.

Et ainsi de suite en permutant à chaque fois, cycliquement, les bandes A1, A2, A3, A4, les fusions se faisant toujours en lecture arrière. Lorsque l'on aura 3 monotonies égales à 3 monotonies venant du tri interne, il y aura une fusion supplémentaire. Et on continue ainsi jusqu'à la fin du fichier.

C'est une méthode efficace pour les gros tris sur gros ordinateurs. Il n'y a pas un seul rebobinage, par contre, on ne sait pas, ici non plus, sur quelle bande on va sortir et on a une perte de temps si on a structuré le programme de tri en phases. A chaque fois il faut recharger la phase tri interne, puis la phase fusion, etc.

4.4. — LA PROGRAMMATION AJOUTÉE DANS LES TRIS

Très souvent il y a dans les programmes utilitaires de tri des « portes de sorties ». Ce sont des points bien déterminés où les programmeurs peuvent ajouter, en utilisant la technique de la verrue, des blocs de programme à eux.

Ainsi, si par exemple un tri se trouve placé après ou avant un traitement simple, on peut placer ce traitement dans le tri en utilisant une « porte de sortie » à l'entrée du tri interne ou bien en cours de la fusion finale. On évite ainsi l'écriture puis la relecture d'un fichier intermédiaire.

5. — Les tris et l'analyse

Examinons le problème suivant :

A la sortie d'un programme de traitement (de calcul) on obtient une bande triée par numéro d'article, par numéro de client (argument majeur : numéro d'article, à l'intérieur de chaque article par numéro de client, ce dernier est l'argument mineur). Les enregistrements contiennent avec différents renseignements un numéro d'article, un numéro de client, un numéro d'agence, un numéro comptable et un numéro de facture.

On désire obtenir 5 états :

— un état 1 par numéro d'article, par numéro comptable, par numéro d'agence par numéro de facture (de l'argument majeur à l'argument mineur);

— un état 2 par numéro d'agence, par numéro de client, par numéro de facture par numéro d'article;

— un état 3 par numéro de client, par numéro de facture, par numéro d'article;

— un état 4 par numéro comptable, par numéro d'article, par numéro de facture;

— un état 5 par numéro de facture, par numéro d'article, par numéro de client, par numéro comptable.

La solution qui vient à l'esprit de l'analyste consiste à effectuer un tri suivant les arguments exigés par l'état 1, puis à imprimer cet état 1; à effectuer un tri suivant les arguments exigés par l'état 2, puis à imprimer cet état 2; ... et ainsi de suite.

On obtient une fin de chaîne constituée de 5 tris et de 5 impressions. Cela est évidemment assez lourd et exige beaucoup de manipulations de la part du service exploitation de la salle des ordinateurs.

Une meilleure solution consiste à créer à la sortie du programme de traitement un fichier 5 fois plus gros : on reproduit 5 fois le même enregistrement. En tête de chacun des enregistrements on ajoute un certain nombre de caractères : une zone. Cette zone contient — pour la première copie de chacun des enregistrements : le chiffre 1 suivit des arguments du majeur au mineur de l'état 1 — c'est-à-dire le numéro d'article de l'enregistrement, puis, dans l'ordre, son numéro comptable, son numéro d'agence, et son numéro de facture. (On réunit tous les arguments en un seul argument de tri.)

— Pour la deuxième copie de chacun des enregistrements : le chiffre 2 suivit des arguments du majeur au mineur de l'état 2.

— Pour la troisième copie de chacun des enregistrements : le chiffre 3 suivit des arguments du majeur au mineur de l'état 3... et ainsi de suite,

Lorsque cette bande est créée il ne reste plus à faire qu'un tri important mais un seul sur cette zone. Tous les enregistrements commençant par le chiffre 1 vont être dans l'ordre et en tête de la bande, les enregistrements 2 ensuite et ainsi de suite.

Il ne reste plus alors qu'à faire un seul listing. On gagne ainsi un temps considérable : beaucoup moins de manipulations, il n'y a plus que 2 programmes au lieu de 10. L'analyste a pris la précaution de classer les états dans l'ordre exigeant le moins de changement de papier à l'imprimante.

Certes on peut penser aussi que ces différents états montrent qu'il y a quelque chose à revoir dans l'organisation de l'entreprise, mais cela c'est un autre problème.

6. — Le Software d'application

En fait les programmes de tri ne sont que les premiers maillons d'une chaîne de programmes généralisés, chaîne que l'on peut appeler « software d'application ». Les problèmes rencontrés dans une entreprise sont le plus souvent semblables aux problèmes rencontrés dans d'autres entreprises. Il est possible de concevoir des programmes, des ensembles de programmes valables pour tous; c'est-à-dire indépendants de l'entreprise et dans une certaine mesure de l'ordinateur (s'ils sont écrits dans un langage évolué). On peut ainsi imaginer : une paye généralisée, un calcul des besoins généralisé, un ordonnancement généralisé, ... Cela peut amener par voie de conséquence des organisations standardisées dans les entreprises. C'est un nouveau domaine appelé à se développer dont il est très difficile de fixer les limites.

CHAPITRE X

L'ORGANISATION

Avertissement préliminaire

Ce livre, bien sûr, n'est pas un traité sur l'organisation. Nous sommes parti des machines et nous aboutissons tout naturellement aux applications. C'est le parcours de *l'informaticien.* Il est possible de faire la démarche inverse, de passer de la gestion des entreprises à la conception d'un « système » puis à l'automatisation. C'est le chemin de *l'organisateur.* Nous renvoyons donc sur des traités d'organisation, mais le lecteur qui aura suivi le premier chemin pourra plus facilement « boucler la boucle » en parcourant le deuxième.

1. — La sous-traitance mécanographique

On classe habituellement les différentes applications des ordinateurs en 3 sections :

— la recherche et les études (secteur dit « scientifique »);

— la gestion (des entreprises);

— l'automatisme (secteur dit « industriel »).

Les premiers ordinateurs ou calculateurs furent conçus pour résoudre des problèmes scientifiques. Ils étaient un outil entre les mains d'un chercheur ou d'un ingénieur. Après avoir lu un énoncé et des données leur rôle se limitait à fournir un résultat.

Parallèlement, on a pensé à utiliser les ordinateurs dans le domaine industriel, pour commander, surveiller et tester les résultats d'autres machines. L'ordinateur participe à la fabrication d'autres ordinateurs! Il y a déjà là un certain progrès : les ordinateurs sont considérés comme des outils intégrés dans une chaîne de fabrication, dans un processus industriel. Il y a une modification de ce processus en fonction de l'arrivée des ordinateurs.

Avec l'avènement de nouvelles unités d'entrée-sortie (notamment les bandes magnétiques) les ordinateurs vont s'étendre et se développer dans un nouveau domaine : la gestion des entreprises. C'est le domaine des ordinateurs de grande série. Ils vont alors remplacer dans des services, appelés « mécanographie » et non « traitement automatique de l'information », les machines comptables classiques. Au début le mot « remplacer » a été le verbe « juste ». On a commencé par transposer, telles quelles, des applications avec cartes perforées sur trieuse, interclasseuse, calculatrice, reproductrice, tabulatrice en applications sur bandes magnétiques. Les enregistrements logiques des bandes ont parfois conservé les 80 colonnes des cartes. Le recyclage des anciens opérateurs, la formation des nouveaux programmeurs, tout cela a dû se faire très vite.

Peu à peu des services, des départements, des directions se sont aperçu que les ordinateurs sont des merveilleuses machines à lister du papier. Il ont donné aux analystes des services mécanographiques de splendides études à faire en leur imposant leurs nœuds d'entrée et leurs nœuds de sortie. Ainsi dans la plupart des entreprises on voit fleurir de magnifiques chaînes de programmes, indépendantes les unes des autres. Le service « informatique » n'est pas dans l'entreprise un service comme les autres. Il est au service de ses semblables que l'on appelle « utilisateurs ».

Qu'en résulte-t-il?

— Les services « utilisateurs » s'aperçoivent sans cesse que les nœuds d'entrée et les nœuds de sortie qu'ils ont fournis aux analystes ne sont pas satisfaisants et ils demandent de les modifier sans cesse. On voit alors des programmeurs passer leur vie à changer, puis à retester leurs programmes, tout cela parce qu'ils ont eu une fois le malheur de les mettre en place. En programmation une modification coûte parfois, aussi cher que la création complète d'un programme.

— Dans des chaînes différentes de services utilisateurs différents, il arrive fréquemment que l'on traite les mêmes problèmes. Par exemple le même fichier peut être mis à jour par des nœuds d'entrée d'origines diverses. Dans une entreprise les différents services forment un tout, par suite les informations rencontrées par l'un peuvent se rencontrer dans l'autre.

— La mise en place d'applications inutiles et non rentables. On a souvent tendance à juger l'importance d'un service au volume du courrier qu'il reçoit. Aussi la mécanographie, service à faire du papier, n'est pas négligée.

— Ranger le service « Informatique » au rang de service de sous-traitance conduit à une psychose de l'ordinateur, il est l'indirect des services indirects (non productifs). On n'a pas confiance en lui. Il nous est même arrivé de constater que l'ordinateur ne soulageait pas le travail des autres services : les calculs se font à la main comme dans le temps. On compare ces résultats avec ceux venant de l'ordinateur. Lorsque, cas très rarissime, les deux résultats sont identiques on se dit : « Il faut bien que cela arrive de temps en temps ».

Si par contre, ils sont inégaux on claironne « l'ordinateur s'est encore trompé, le pôvre ! ». On n'incrimine jamais les données qu'on lui a fournies à l'entrée.

Les différents services forment un ensemble. Il est anormal de dire qu'un service dépend des autres. Ils sont chacun au service de l'entreprise. D'où l'idée d'utiliser le traitement de l'information en général et les ordinateurs en particulier d'une nouvelle manière.

2. — La gestion intégrée

L'informatique n'est plus vu à l'échelon de chacun des services, mais à partir de chacun des circuits d'information de l'entreprise. Il y a des fonctions, et l'information qui circule de fonction en fonction. La fonction implique une position statique chargée de recevoir, de traiter, de ventiler vers d'autres fonctions, différentes informations. Au-dessus de ces fonctions il y a la direction qui doit prévoir la régulation des différents circuits, réagir devant les écarts et devant les événements (fig. 10.1).

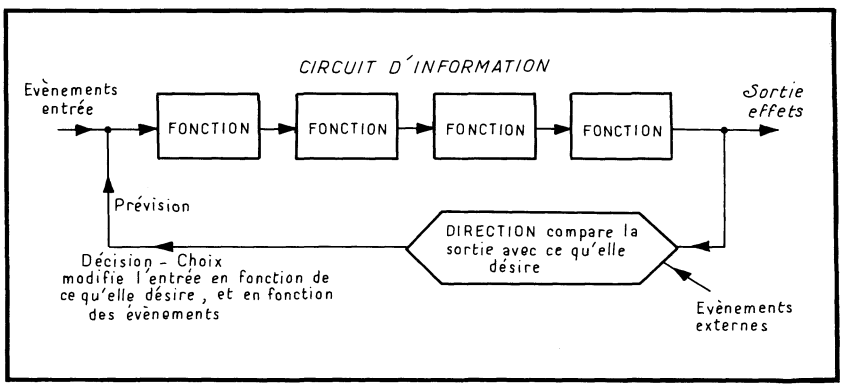


Fig. 10-1. — La Direction d'une entreprise face à un circuit d'information.

Prenons un circuit d'approvisionnement dans une usine. Les informations à l'entrée sont de deux types. Il y a d'une part les commandes qui viennent des circuits commerciaux, et d'autre part les gammes de fabrication amenées par les services de laboratoire et de bureau d'études.

A partir de ces deux sources le service « contrôle de production » va établir les différents besoins d'ensembles de pièces suivant les différentes périodes à venir. Parallèlement, il va établir l'ordonnancement, le planning des différentes chaînes de l'usine. Intéressons-nous plus particulièrement aux achats et à la sous-traitance. Parmi les besoins en stock le contrôle de production établit des demandes d'achats nécessaires : pour les pièces élémentaires, et pour les ensembles ou opérations appelées à être exécutées à l'extérieur de l'entreprise.

Ces « demandes d'achat » parviennent au service des achats qui après des « appels d'offre » chez différents fournisseurs va faire « une commande » chez le ou les fournisseurs choisis en fonction de leur « offre ».

Lorsque les pièces ou ensembles sont livrés à l'usine, ils sont pris en charge par un service « réception et magasin ». Ce service contrôle quantitativement ce qu'il reçoit puis soumet le tout au service « contrôle de qualité ». Après réception définitive le service « comptabilité industrielle » compare le prix de la commande avec le prix de la facture et enregistre le prix définitif. Pendant ce temps les pièces ou ensembles sont stockés « en magasin » en attendant d'être traités par les chaînes de fabrication.

Nous avons défini ainsi un circuit d'information passant par les fonctions : contrôle de production, achats, réception, contrôle de qualité, comptabilité industrielle et magasin.

Si on travaille en mode « sous-traitance mécanographique » que va-t-il se passer?

— Le service « contrôle de production » va demander une chaîne mécanographique pour effectuer son calcul des besoins et établir ses demandes d'achat.

— Le service « achats » va commander une chaîne mécanographique pour imprimer ses commandes sur ordinateur, en espérant que cette chaîne ne se contentera pas de remplacer des dactylos par des perfos.

— Le service « réception » va demander une chaîne mécanographique pour s'assurer que les marchandises livrées correspondent bien à celles qui ont été commandées; pour alerter le service des achats en cas de retard à la livraison.

— Le service « contrôle de qualité » va commander une chaîne mécanographique pour savoir quels sont les lots qu'il doit contrôler, le nombre de pièces qu'il doit vérifier par lot, et les mesures qu'il doit faire sur ces pièces.

— Le service « comptabilité industrielle » va demander une chaîne mécanographique pour comparer les factures et les commandes et pour enregistrer le prix des factures sur son fichier « comptabilité ».

— Le service « magasin » désire contrôler automatiquement ses entrées et ses sorties, la surface nécessaire pour les stocks, les files d'attente des pièces en attente de fabrication...

Il va y avoir ainsi 6 analyses qui vont s'effectuer en parallèle pour optimiser les fonctionnements respectifs de chacun des services.

En fait il s'agit d'un même circuit d'information, traitant des mêmes informations, donc du ou des mêmes fichiers. Pour l'optimiser dans l'ensemble de la société il est nécessaire de la traiter en un seul bloc, et non pas d'optimiser séparément chacunes des parties.

Un circuit d'information se décompose en parties mécanisables et en parties non mécanisables. Les limites de ses parties ne correspondent pas forcément aux frontières des services existants. Il incombe aux ingénieurs en organisation de fournir des nœuds d'entrée et des nœuds de sortie, pour chacune des parties mécanisables aux analystes correspondants; de relier le nœud de sortie au nœud d'entrée de la partie mécanisable suivante par une procédure non mécanisable. Par exemple le traitement automatique peut fournir, pour un numéro d'article donné, une liste des fournisseurs possibles.

Ce nœud de sortie est remis au service « achats » lequel choisit le ou les fournisseurs, et ce choix devient le nœud d'entrée dans le traitement automatique chargé d'obtenir les commandes. Ce choix des fournisseurs, bien que répétitif n'est pas mécanisable car on ne connaît pas la règle de répétition. Il y a en particulier des facteurs dont on n'est pas maître (on peut seulement fixer la probabilité) : citons l'accord du fournisseur.

Les ingénieurs en organisation doivent aussi prévoir les différents liens entre les différentes parties mécanisables. Par exemple une erreur dans un nœud d'entrée doit pouvoir être corrigée dans les chaînes suivantes.

On peut résumer la mise en place du traitement de l'information pour un circuit donné par le planning suivant :

— *Étude d'un circuit d'information.* Comment il est actuellement dans l'organisation existante? Sa position, sa relation par rapport aux autres circuits. L'entreprise formant un tout, les différents circuits doivent être reliés ensemble. Il ne faut pas non plus optimiser un circuit au détriment de tous les autres. On se fixe ainsi les objectifs, les performances que l'on désire.

— *Création d'un avant-projet.* C'est déjà un modèle, présentant la chose d'une manière grossière. Il permet d'améliorer ses propres dimensions par des études quantitatives et qualitatives. On examine les résultats qu'il donne et on compare avec les objectifs.

— Après plusieurs modifications on arrive ainsi *au projet définitif.* Il est bon de noter que la logique des blocs, très utile en analyse et en programmation, est indispensable au niveau de l'organisation. Le circuit est divisé en parties mécanisables et non mécanisables. Pour chaques parties le projet définit les procédures. A chaque niveau il répond aux éternelles questions : quoi? où? quand? comment? Il définit en particulier les nœuds d'entrée et de sortie des parties mécanisables.

— Chacun des couples nœuds d'entrée et de sortie est remis aux analystes correspondants. Ceux-ci peuvent alors commencer *leur analyse.* Pour certains détails concernant les nœuds, ils peuvent demander des précisions aux services intéressés, mais ils ne doivent pas modifier les procédures précédemment définies.

— Après les analyses, les dossiers d'analyse amènent *la programmation,* la mise au point et le test de chaque programme puis de chaque chaîne.

— Vient ensuite la mise en place et *le démarrage* du circuit.

La somme des connaissances nécessaires à l'organisation est très grande. Elle exige des spécialistes dans des domaines très divers : *connaissance des* ordinateurs et de leurs nouveautés techniques, fonctionnement des entreprises, comptabilité, contrôle de production, ordonnancement, achats et ventes, contrôle de gestion..., recherche opérationnelle, statistiques et probabilités, jeux d'entreprise, économie politique, techniques industrielles... Aussi l'organisation ne peut être qu'un travail d'équipe auquel doivent participer, avec les ingénieurs en organisation, l'ensemble des chefs de service ou de département de l'entreprise (spécialiste du management et spécialiste des problèmes de leur secteur), et aussi parfois des spécialistes pouvant provenir

de l'extérieur. Les problèmes rencontrés par une organisation sont pour la plupart semblables à ceux rencontrés par d'autres et il est bon de confronter les solutions déjà prises et les résultats obtenus.

L'organisation est un travail ingrat, car par le fait même qu'elle étudie la dynamique de l'information, elle se heurte à la fonction en place. L'organisation peut dans certains cas être amenée à déplacer ou à modifier cette fonction. D'où l'éternelle méfiance des chefs de service, spécialistes du management, à l'égard de l'ingénieur en organisation. Ils ont l'impression qu'on leur enlève une partie de leur pouvoir.

L'étude du traitement de l'information n'enlève rien au pouvoir de décision de la fonction, mais au contraire permet de faciliter ce choix, cette décision en lui présentant les événements, les effets d'une manière plus complète plus claire et plus rapide. Tout ce qui est automatique, administratif, répétitivement défini, « tout le travail bête » est supprimé. On ne laisse que ce qui exige réflexion, imagination, avec un entourage plus sûr : statistiques, probabilités, prévisions.

Nous pensons qu'une des meilleures manières de faciliter le dialogue entre le personnel d'une entreprise et un groupe d'organisateurs est l'éducation, l'explication de ce que l'on fait. Si par exemple à un magasinier on demande en plus de son travail d'écrire soigneusement sur un bordereau de perforation ce qu'il rentre et ce qu'il sort, on risque à la longue d'avoir beaucoup d'erreurs. Pourquoi voulez-vous qu'il effectue convenablement un travail supplémentaire dont il ne comprend pas la finalité? (si ce n'est pense-t-il dans un but de surveillance). Il est nécessaire le plus souvent de lui expliquer grossièrement ce qu'est l'informatique, ce que sont les ordinateurs afin de l'intéresser à son travail. Ce qui est vrai pour le magasinier l'est aussi pour tous les étages de la société.

Par rapport aux analystes du traitement automatique de l'information, l'ingénieur en organisation est ce que l'urbaniste est aux architectes. Livrés à eux-mêmes ils peuvent faire de belles façades, il n'est pas sûr que l'ensemble fasse une ville.

3. — L'organisation d'un département « Informatique »

On doit retrouver (fig. 10.2), à l'intérieur d'un tel département les parties : organisation — analyse — programmation — et exploitation.

3.1. — LA PARTIE ORGANISATION

Elle comprend :

— le groupe des ingénieurs en organisation des différentes disciplines de ce domaine.

Parmi ces différents spécialistes il en est un qui joue un rôle spécial à l'intérieur même du département, c'est le spécialiste des techniques de traitement automatique de l'information. Il se tient en liaison étroite avec les

services technico-commerciaux des différents constructeurs d'ordinateur. Il ventile ses informations auprès des autres parties du département. Il conseille sur le choix des machines, des techniques d'analyse et des langages de programmation. Il participe aux réunions du groupe organisation, contrôle les organigrammes et les ordinogrammes. La plupart du temps il est aidé par un ou plusieurs programmeurs chargés de tester et de signaler l'existence des nouveaux programmes utilitaires, de dépanner les programmeurs et l'atelier d'exploitation.

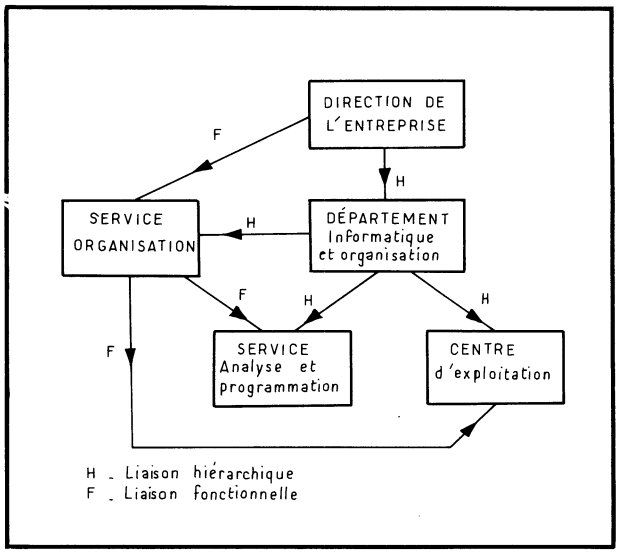


Fig. 10-2. — Organisation interne d'un département informatique et organisation.

— un groupe administratif chargé de gérer le catalogue des fichiers et de veiller à une standardisation des informations qui circulent dans l'entreprise. C'est par exemple lui qui donne aux analystes les dessins des fichiers déjà existants. Comme il connaît la liste de tous les programmes utilisant un fichier, une modification de fichier ne peut pas se faire sans son accord. Il gère aussi la bibliothèque des brochures techniques et les programmes utilitaires.

3.2. — LA PARTIE ANALYSE ET PROGRAMMATION

Le plus souvent l'analyse et la programmation ne forment qu'une seule partie car pendant le temps où les programmeurs travaillent pour un même analyste ils forment un groupe. Ce sont donc des groupes essentiellement variables suivant les besoins du moment. Il y a les groupes chargés d'études nouvelles et les groupes chargés des modifications ou des améliorations d'études.

3.3. — LA PARTIE EXPLOITATION

Elle comprend :

— un groupe administratif chargé de recevoir les différents documents (nœuds d'entrée), et de ventiler les différents nœuds de sortie.

A partir des dates d'arrivée des entrées, et des délais de livraison des sorties, des méthodes (PERT) de calcul permettent d'établir automatiquement le planning de l'atelier. L'ordinateur établit lui-même son plan de travail. Cela est particulièrement avantageux en cas de changement de délai ou d'incident. Une modification dans les données et l'ordinateur refait rapidement son planning. En même temps l'ordinateur crée son paquet de cartes de commande, son « batch » pour le système d'exploitation : les cartes de commande des différents travaux, les cartes d'appel de programmes et de fichiers. Ce groupe administratif comporte aussi un *magasinier* chargé de gérer les différents fichiers et fournitures nécessaires à l'atelier. Le magasinier à partir du planning prépare ce qu'il faut pour chacun des travaux sur des chariots. Les chariots sont placés dans l'ordre de leur utilisation. Pour les remplir il se sert des brochures d'atelier fournies par les analystes.

— L'exécution proprement dite qui comprend l'atelier de perforation et de vérification des cartes, le découpage des états imprimés.

— Enfin la salle des ordinateurs où l'on trouve parmi le personnel : *des pupitreurs*, en principe un par gros ordinateur, capables en cas d'incident de repérer la cause de cet incident et d'agir en conséquence. On y voit également *des opérateurs* capables à partir des brochures d'atelier et des chariots remplis par les magasiniers de fournir du travail à l'ordinateur en respectant le planning.

3.4. — LA BROCHURE D'ATELIER

Comme cette brochure est le centre du fonctionnement de l'atelier nous devons la préciser un peu plus. Elle est le lien entre l'analyse et l'exécution. L'exécution doit pouvoir fonctionner en l'absence des analystes et des programmeurs responsables. Combien de fois avons-nous vu, sur tel arrêt numéroté, dans la brochure, l'expression : « me voir » ou bien « ne peut pas se produire »? Ce sont là des erreurs d'analyse. La brochure d'atelier (une brochure par application) doit contenir :

— Généralités : en général une ligne ou deux, jamais plus d'une page. On indique le titre du travail, son numéro dans l'entreprise et très brièvement son but. Éventuellement après quel autre travail, ou avant quel autre, il doit se produire. On prévient l'atelier d'un cas particulier. On indique le temps moyen d'exécution, le type d'ordinateur utilisé et la configuration minimum nécessaire.

— L'organigramme général. Très général mais très lisible, bien préciser les entrées et les sorties.

— Le tableau des assignations de fichier. Dans le cas où le système d'exploitation choisit lui-même les volumes pour les fichiers ce tableau est évidemment inutile. On ne place donc dans cette rubrique que les fichiers dont on impose le volume au système.

— Les fournitures. C'est pour l'atelier le chapitre le plus important. Il doit permettre aux magasiniers de préparer le travail sur un chariot. On indique donc ici tout ce qu'il faut en support : cartes, bandes, disques, papiers, en qualité et en quantité.

— Les manipulations spéciales correspondant aux cas particuliers signalés dans la rubrique généralités.

3.5. — LE DÉMARRAGE D'UN ORDINATEUR

La densité de travail à fournir dans un département « informatique » est très variable et il arrive très souvent de sous-traiter à tout niveau, de l'organisation à l'exécution. Parmi les périodes de pointe se situe le démarrage d'un ordinateur. Il s'agit de préparer la marche de cette machine avant même son arrivée. Pour cela il y a les problèmes d'éducation du personnel, des nouvelles organisations, des nouvelles études, des reconversions de fichier. La plupart du temps cela se planifie par des méthodes PERT en liaison avec les services technico-commerciaux du constructeur.

Citons les solutions de dépannage permettant de traiter avec un nouvel ordinateur B les programmes de l'ancien ordinateur A.

Il y a d'abord les *simulateurs*. Ce sont des programmes utilitaires, donc purement software permettant de transformer les programmes écrits en autocodeur A en programmes écrits en autocodeur B. Le résultat est généralement « un veau ». Pour les programmes écrits en langage évolué il vaut mieux effectuer les quelques retouches nécessaires.

Avec la *compatibilité* l'ordinateur B possède dans ces circuits tout ce que contient l'ordinateur A, c'est donc un dispositif purement hardware. Dans le cas où les ordinateurs A et B sont de logiques différentes le résultat ne peut être qu'au plus égal au rendement de l'ancien ordinateur A.

Enfin il y a *l'émulateur* qui est un dispositif à la fois hardware (la plupart du temps des microprogrammes supplémentaires) et software (pour initialiser et compléter ces microprogrammes). On peut alors obtenir un rendement nettement supérieur à l'ancien ordinateur sans atteindre celui du nouveau.

4. — Le domaine industriel (l'automatisme)

Dans ce domaine la définition d'un circuit d'information apparaît clairement. Il s'agit d'une chaîne de fabrication divisée en opérations élémentaires. Certaines sont mécanisables, d'autres ne le sont pas. Les difficultés rencontrées par l'ingénieur en organisation sont différentes. On se heurte avant tout à des problèmes techniques. Il est souvent nécessaire d'avoir recours à du matériel spécialisé : des organes d'entrée-sortie, voire des ordinateurs complets construits spécialement pour résoudre tels problèmes, donc très coûteux. Par contre les résultats sont rapidement spectaculaires.

4.1. — L'INFORMATION ANALOGIQUE

Un des problèmes est l'acquisition des données. Les données d'entrée se déplacent dans le temps d'une façon continue comme par exemple la tension ou l'intensité d'un courant. On peut ramener par analogie toutes les variables des différentes énergies aux variables de l'énergie électrique plus souples à manipuler. Devant les performances des ordinateurs digitaux, ceux-ci tendent de plus en plus à remplacer les ordinateurs analogiques. Il faut alors transformer l'information analogique en information digitale (numérique). Cela se fait en décomposant les courbes des variables en dents de scie verticale, comme dans la figure 10.3. Il suffit de prendre un intervalle de temps, déter-

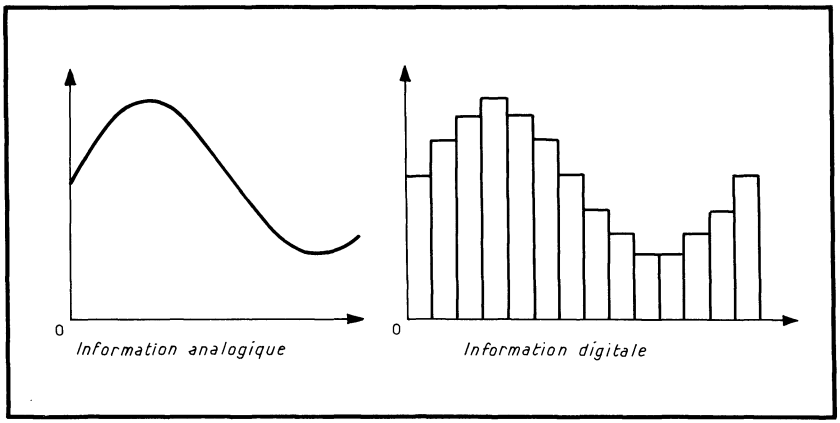


Fig. 10-3. — Transformation de l'information analogique en information digitale. On décompose la courbe en dents de scie.

minant la largeur de ces dents, suffisamment petit pour que l'arrondi effectué pour fixer les hauteurs soient négligeables devant la précision des appareils de mesure.

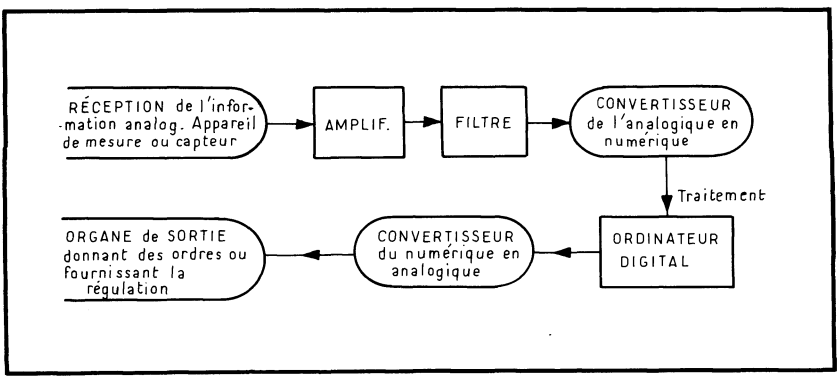


Fig. 10-4. — L'information analogique de l'appareil de mesure ou du capteur est traitée par un ordinateur digital. L'organe de sortie ou de régulation reçoit de l'information analogique.

Inversement à la sortie il peut être nécessaire de transformer l'information numérique en information analogique. Citons les traceurs de courbe, l'affichage d'une courbe sur un écran de télévision. La figure 10.4 montre un circuit d'information analogique traité par un ordinateur digital.

4.2. — LA RÉGULATION

L'ordinateur compare le résultat d'un système avec ce que l'on désire et suivant l'écart enregistré par cette comparaison il réagit sur les commandes du système, comme dans la figure 10.5. Généralement dans un processus industriel on a deux sortes de variables :

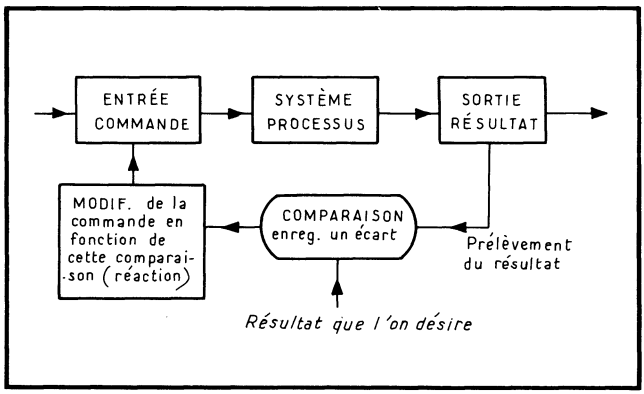


Fig. 10-5. — Principe de la régulation.

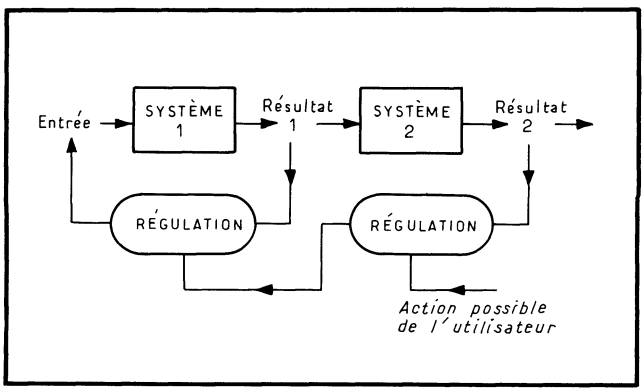


Fig. 10-6. — La régulation en cascade.

— les variables que l'on peut régler, on modifie volontairement le résultat en agissant sur elles;

— les variables perturbatrices arrivant d'une manière aléatoire, elles modifient le résultat d'une façon imprévue. En mesurant le résultat, l'ordinateur agit sur les variables réglables de manière à corriger l'effet des variables perturbatrices. Il y a ainsi plusieurs types de régulation suivant la réaction que l'on donne en fonction de l'écart. Il y a des réactions proportionnelles à l'écart, intégrales, dérivées de l'écart.

Lorsque les processus se suivent, (fig. 10.6), il est possible de faire ce qu'on appelle la régulation en cascade : chaque régulateur agit sur le régulateur précédent.

Enfin dans le cas de systèmes très complexes il est nécessaire de créer des modèles mathématiques afin de calculer les différents asservissements. (Par quel miracle le terme « servo-mécanismes » est-il devenu dans le langage courant « cerveaux électroniques »?)

4.3. — LA COMMANDE PROPREMENT DITE

On peut mécaniser tout ce qui est répétitif d'une manière définie, en particulier tout ce qui est travail en chaîne. La seule limite est le prix des études et du matériel. On a d'abord pensé utiliser l'ordinateur pour contrôler, surveiller la marche des phénomènes complexes. On peut aussi utiliser l'ordinateur au cœur même du processus, dans la chaîne même. Donnons deux exemples.

A la place des machines-outils

Imaginons, comme dans la figure 10.7, une pièce montée sur un étau, capable de tourner autour d'un axe horizontal et d'un axe vertical, capable aussi de se translater dans les 3 directions de l'espace.

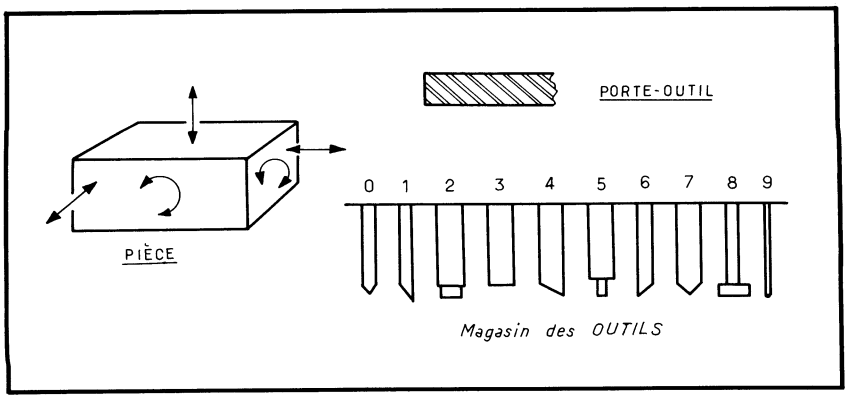


Fig. 10-7. — Fabrication automatique d'une pièce.

Une telle pièce peut se positionner d'une manière quelconque par rapport à un porte-outil. On dispose en plus d'un magasin d'outils où chacun des outils est numéroté. Le porte-outil a la possibilité d'aller prendre et monter sur lui n'importe lequel de ces outils, de travailler avec, et ensuite de le remettre en place. On peut commander les mouvements de la pièce et du porte-outil par un programme enregistré sur une bande perforée par exemple. On peut ainsi rendre automatique tout une série d'opérations.

En effet après le montage de la pièce sur l'étau les opérations s'exécutent toujours de la même manière :

— 1. positionnement de la pièce (sur la position M);

— 2. le porte-outil va chercher l'outil n° x et le met en place;

— 3. travail de l'outil la pièce se déplaçant de la position M à la position M' à la vitesse v;

— 4. remise en place de l'outil.

On peut donc commander chaque opération par une instruction standard. Le programme commence par une mise en place de la pièce, puis ensuite par autant d'instructions qu'il y a d'opérations. On termine le programme par le déchargement de la pièce.

Pour traiter une autre pièce il suffit de recommencer le programme. Si celui-ci est sur bande perforée on peut la mettre sous la forme d'une boucle.

Le problème délicat est la programmation car une erreur peut endommager le matériel, pour cela 2 solutions :

— La simulation : par exemple sur un ordinateur digital classique un programme de simulation contrôle le bon fonctionnement du programme machine-outils.

— Le programme de la première pièce : c'est l'ordinateur qui construit lui-même son programme. On exécute avec les commandes manuelles de la machine la première pièce. L'ordinateur enregistre ce que l'on fait. Il pourra ensuite répéter indéfiniment ce qu'on lui a montré.

Contrôles automatiques de circuits (fig. 10.8)

Il s'agit ici de contrôler des circuits ou des ensembles complexes. L'ordinateur possède une unité de lecture sur laquelle arrivent, par une rampe mécanique, les unes après les autres, les différentes pièces. Il lit à un endroit bien déterminé le numéro de la pièce. Il va chercher ce numéro dans un fichier sur disque magnétique. On trouve dans ce fichier les mesures qu'il faut faire et les résultats qu'il faut trouver. La pièce avance et passe devant une ou plusieurs unités de mesure. Elles effectuent les mesures demandées par l'ordinateur. Celui-ci compare le résultat avec ce que l'on doit trouver. En cas d'inégalité la pièce ou ensemble est éjectée de la chaîne, parallèlement un état sort sur l'imprimante précisant sur quelle mesure se trouve l'erreur. Cela permet de corriger rapidement en amont de la chaîne, l'opération défaillante.

Les avantages apportés par l'ordinateur industriel sont des gains dans la précision, dans les délais par la diminution des temps morts, la réduction du parc machine et une diminution considérable du travail humain à la chaîne.

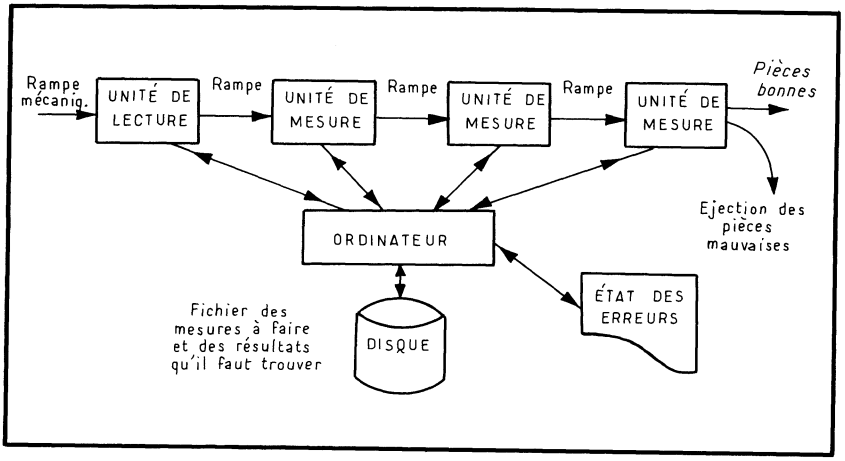


Fig. 10-8. — Contrôles automatiques des circuits.

5. — Applications et limites des ordinateurs

5.1. — LISTE DES APPLICATIONS

Le nombre des différentes applications des ordinateurs est immense. On les utilise dans tous les domaines. Donnons *en vrac* une liste suffisamment éloquente :

La géophysique, la géologie mathématique, la mécanique : le calcul des contraintes et de résistance des matériaux, les jeux d'entreprise, étude de la mise en place d'un ordinateur, politique optimale face à des facteurs divers, étude d'investissement, loi de réapprovisionnement en fonction d'une loi de demande, études de phénomènes aléatoires, l'ordonnancement et le lancement de fabrication, l'enseignement programmé sur ordinateur, classement d'une bibliothèque permettant par des systèmes de mots clés de retrouver rapidement une série de livres soit d'un auteur donné, d'un sujet donné, d'une langue donnée ou d'une époque donnée... Études de plans d'investissement avec choix entre plusieurs matériels, minimisation des durées de déplacements, plannings de machines, d'avions ou de chemins de fer en fonction des demandes ou des besoins, problèmes de linguistique, études des problèmes de trafic (théorie des graphes), problèmes de distribution, de livraison, réservation automatique de places, problème de stockage, affectation du personnel, étude de chemin critique, étude de communications entre des êtres humains, résultats lors des réunions sportives : jeux olympiques, 24 heures du Mans, Tour de France, organisation d'une cession d'examen en fonction des inscrits, pointage du personnel (intégré dans le calcul de la paye), industrie laitière, industrie hôtelière (réservation de chambres d'hôtels, de places dans les différents transports), industrie aéronautique : simulation de vols *spatiaux*,

études en soufflerie, essais d'un réacteur, étude de vibration, contrôle de la navigation aérienne (choix d'un trajet pour un avion, contrôle et modification de ce trajet à partir de stations placées à terre), poursuite de satellite... industrie chimique, industrie de la viande, étude des affaiblissements de terrains en fonction des constructions en cours, identification des personnes par leurs empreintes, composition automatique des textes en imprimerie, étude de terrains, recherche de gisement (pétrole), classification des dossiers médicaux dans les hôpitaux, la pédagogie médicale, recherche d'un diagnostic médical, prévision des performances d'un ordinateur devant les besoins d'un client.

La théorie des probabilités : l'étude de la circulation dans une ville, le choix des feux rouges et des sens uniques; l'étude des facteurs ayant une influence sur le fonctionnement d'une machine, sur les qualités d'un produit, sur les prix de revient, de vente; étude des prévisions en fonction des modifications de ces facteurs, étude sur les causes de ces modifications : prix, nombre de clients, chiffre d'affaires, les impondérables, les tendances, les cycles saisonniers. On distingue « le calcul de corrélations »; influence relative de plusieurs facteurs variant en même temps, et « le calcul de régressions » : étude d'une équation reliant tous les facteurs d'un modèle. La lexicographie explore complètement toutes les évolutions possibles d'un système dans le but de trouver une suite séquentielle d'évolution ayant certaines propriétés (par exemple suite de travaux à donner à une machine-outil).

Si les paramètres d'un modèle mathématique sont reliés par des relations linéaires on est dans le cas de la « programmation linéaire ». Cette programmation dispose d'algorithmes et est utilisée dans des domaines très variés. La « programmation dynamique » est l'étude des variations d'un modèle en fonction des variations d'une de ces variables, par exemple le temps. Dans le cadre des « programmations non linéaires » (il existe au moins une relation non linéaire entre les variables) on utilise des méthodes d'exploration pour trouver l'optimum. Les « tables de décision » permettent, si n conditions se présentent en même temps, de dire il y aura p résultats ou bien il faut faire q actions.

Il est aussi difficile de définir en quelques lignes le terme « recherche opérationnelle » que le terme « informatique » tellement la diversité des problèmes qu'elle traite est grande. Ce que l'on peut dire c'est qu'elle est une science appliquée, amenée à utiliser ses méthodes propres mais aussi celles des autres sciences. Elle est amenée en général à traiter des problèmes ayant un grand nombre de paramètres et le plus souvent utilise l'ordinateur comme outil. Sans cet outil elle ne donnerait que des résultats imprécis obtenus par des calculs fastidieux. Peut-être la recherche opérationnelle est-elle tout simplement la résolution d'une manière scientifique d'un problème dont l'énoncé n'a apparemment rien de scientifique : recherche d'*une solution* optimale en examinant toutes les solutions, protection des convois maritimes contre les sous-marins, étude d'un train de laminoir, étude de la production d'une usine, étude de la vente d'un produit...

Les statistiques, étude des grands nombres, des séries qui reviennent sans cesse, des événements exceptionnels, l'économie politique, le recensement, les sondages d'opinion, étude des jeux de hasard, des phénomènes économiques et sociaux, la météorologie, la courbe de Gauss, les sciences humaines, la physique moderne, l'astronomie, la conduite d'une guerre, les

contrôles de qualité sur les fabrications industrielles (le contrôle ne se fait que sur un échantillon), enfin tout ce qui présente une grande quantité et pour lequel on désire une certaine fiabilité; les élections, le choix d'une politique commerciale, abandon ou construction d'un produit avant même le lancement de la fabrication, procédés nouveaux de calcul, de raisonnement, méthodes de prévisions économétriques (science statistique des prévisions), les assurances, le PMU...

Nous arrêtons là cette liste qui bien sûr est loin d'être complète.

5.2. — RÉFLEXION SUR LES LIMITES DES ORDINATEURS

Ce qu'il faut aussi savoir, afin de domestiquer complètement « la plus belle conquête de l'homme du xx^{e} siècle », ce sont ses limites. Lorsque par exemple nous disons « diagnostic médical » il est faux de penser que l'ordinateur remplacera le médecin. Du reste, quel malade accepterait d'être examiné par une machine? L'ordinateur se borne à partir de différents paramètres (symptômes) d'énumérer les différentes maladies possibles et de placer en face de chacune d'elle la probabilité relative. Au médecin ensuite de choisir.

La comparaison entre l'homme et l'ordinateur nous fait toujours sourire. S'il est exact que nous possédons des organes d'entrée-sortie et un milieu interne, on ne peut pas pousser l'analogie plus loin. La mémoire d'un ordinateur est une mémoire « tout ou rien », l'information est ou n'est pas dans les ferrites. La mémoire de l'homme est beaucoup plus vague et cette mémoire est conditionnée par énormément d'effets internes et externes. Malgré des progrès techniques fantastiques, nous sommes très éloignés de la miniaturisation de la cellule vivante. Combien de ferrites faut-il pour représenter un visage et de combien de visages avons-nous souvenance? La forme d'un visage est-elle enregistrée chez nous comme une information analogique ou une information digitale?... La construction des ordinateurs n'a pas suivi le chemin prévu par un des aspects de la cybernétique : étudier l'homme pour pouvoir construire des machines. Nous connaissons parfaitement le fonctionnement des ordinateurs, nous sommes encore loin de comprendre le fonctionnement de notre cerveau. S'il avait fallu attendre des connaissances suffisantes sur les êtres vivants pour créer des machines semblables, malgré les progrès fantastiques effectués, nous serions encore loin de posséder des ordinateurs.

L'ordinateur ne sait faire que ce qu'on lui a montré. C'est un merveilleux perroquet capable d'exécuter un très grand nombre de fois les mêmes opérations *Il* est capable de nous battre au jeu du « morpion », ou au jeu de « Marienbad » car les mathématiciens connaissent les lois de la meilleure politique à adopter. Par contre, s'il est capable de déplacer les pièces du jeu d'échecs il se fait battre par un joueur même débutant. On ne connaît pas encore de règles valables dans tous les cas aux échecs et on ne peut donner à l'ordinateur que des principes parfois vrais, mais aussi parfois faux. L'ordinateur n'est pas capable de découvrir, d'imaginer, que le principe général dans le cas particulier de telle position n'est pas à appliquer. Récemment un match d'échecs sur 4 parties a eu lieu entre les ordinateurs russes et les ordinateurs américains. Ce match était bien sûr un test entre les meilleurs principes programmés, par exemple : « il ne faut pas laisser une pièce en prise », « il faut occuper le centre »...

On a aussi pensé à enregistrer les coups de telle manière que l'ordinateur s'améliore au fur et à mesure qu'il joue. On se heurte alors très rapidement à la capacité des mémoires. En fait les échecs sont l'exemple même de ce qu'un ordinateur ne peut pas faire. Un joueur d'échecs ne raisonne pas par coup, il ne calcule pas à chaque fois le meilleur coup possible, c'est pour lui impossible. Il adopte une stratégie, une idée, un plan d'ensemble fonction de son adversaire et de lui-même. Un plan? Une idée? un ordinateur en est incapable. Il ne peut que composer une musique « à la manière de... » à partir de morceaux caractéristiques de cet auteur en les mélangeant différemment, il ne possède pas notre sens artistique.

Ce qu'il est juste de dire c'est qu'une découverte, un progrès ne peut plus se faire sans que la plus belle conquête de l'homme du xx^{e} siècle y participe. La puissance économique, militaire d'un pays se mesure au nombre de ses ordinateurs multiplié par leur puissance moyenne. L'ordinateur modifie chaque jour notre manière de vivre : les systèmes de numération que nous devons enseigner à nos bacheliers candidats programmeurs sont déjà connus de nos gamins en culotte courte. Le traitement en temps réel favorisera la décentralisation des usines en province tout en laissant le siège social à Paris. Toute entreprise ignorant l'ordinateur ou incapable d'utiliser ses possibilités est appelée à disparaître...

L'ordinateur construit par l'homme, pour l'homme, ne supprime pas l'homme. En supprimant tout ce qui est administratif, répétitif, travail à la chaîne, il remplace l'homme-esclave par un homme socialement plus élevé. L'homme ne peut plus se passer de l'ordinateur, mais l'ordinateur ne peut pas travailler sans l'homme.

Terminons sur une note gaie à l'aide de 3 tableaux humoristiques.

5.3. — NOSTRADAMUS

1963. Le rayon Z d'un grand magasin parisien est plein à craquer. Les hauts-parleurs claironnent partout : « Venez voir au rayon Z les machines électroniques qui ont révolutionné l'Amérique ! » Dans l'agglutinement de la foule, du bruit, sous des néons rouges, jaunes, verts, des femmes ayant beaucoup de hardware mais bien peu de software, attendent leur tour. Un écriteau, long de deux mètres « Ne pas toucher : machines électroniques », lettres blanches sur fond rouge, les préviennent d'un grand danger. Le Tout-Paris est là pour assister au nouveau miracle. Pensez donc ces machines prédisent l'avenir, votre avenir !

Encastrées entre des tables recouvertes de tapis de velours noir une perforatrice et une trieuse, d'avant-guerre et non d'avant-garde, *aux larges* pieds sculptés, sont le centre de tous les regards. Le tout sent le cuivre et le laiton. Un peu plus loin, dans des bacs, des cartes commentaires préperforées attendent d'être utilisées.

Le miracle par personne demande un franc nouveau. Il faut donner son nom et sa date de naissance. Peut-on prévoir sans date de naissance? Les personnes passent 12 par 12, et à chaque personne on perfore une carte avec les renseignements donnés. Comme par hasard, une perforation existe déjà dans une colonne bien déterminée des cartes, et parmi les 12 cartes prises

à chaque fois il n'y a pas 2 perforations identiques. L'opérateur prend ces 12 cartes et place derrière une poignée de cartes commentaires. Il place le tout dans la trieuse et appuie sur départ. Rien ne va plus. C'est l'instant critique qui décide de l'avenir de 12 vies. Seront-elles roses ou noires, passionnées violemment ou passionnées doucerettement? Les cartes des clientes tombent une à une dans chaque case, puis derrière les cartes commentaires se répartissent suivant les cases. On donne le contenu de chaque case à la cliente correspondante : « Après un grand amour vous aurez un gros bébé qui fera pipi et caca partout ». Ah que c'est beau l'automatisme, quel progrès tout de même. Il y aura toujours des margoulins pour exploiter la crédulité des gens.

5.4. — LA PRÉCISION ZÉRO

Le Professeur Omégajust est un grand mathématicien. Du haut de sa barbichette il adore parler de ces merveilleuses lignes droites, sans épaisseur, qui s'enfoncent à l'infini, vers les points cycliques ou ailleurs. Il aime beaucoup travailler avec l'ordinateur. Il a toujours plein les poches de ces problèmes extraordinaires, agréables à l'esprit mais dont on ne voit jamais l'utilité pratique. Lorsqu'il a un algorithme à programmer il demande la précision « zéro ». On s'arrête lorsque le résultat d'un pas est rigoureusement égal au résultat du pas précédent. Pour le Professeur un résultat ne s'exprime pas par une fourchette. Il aime les chiffres nets et tranchants comme une lame de couteau. Lorsqu'il utilise l'ordinateur il ne dit pas qu'il fait de l'informatique, il dit qu'il « calcule » et en prononçant ce mot on aperçoit comme un éclair dans ses yeux noirs. Pourtant le Professeur a des doutes sur la valeur des résultats donnés par les « calculateurs ». Dernièrement on a changé d'ordinateur et sur le nouveau, les dernières décimales de ses résultats ne correspondent pas exactement avec celles données par le précédent. D'autre part, il constate que certaines séries divergentes sont données comme convergentes...

5.5. — UN CAS

Le programmeur Jojo-la-Bricole sait tout faire. Il est indispensable. Il rend compte directement aux services « utilisateurs » de la mécanographie. Il manipule les langages de programmation et les langages absolus en particulier, avec dextérité. Il ne fait jamais d'ordinogramme car il est assez fort pour s'en passer. Devant l'ordinateur il se sent vivre, il est le maître après Dieu. Plus il y a de boutons à manipuler et plus il est heureux. Il ignore les jeux d'essai et travaille sur les fichiers existants. Il traite ses semblables et surtout les opérateurs avec dédain. Il est la bête noire des inspecteurs et des technico-commerciaux du constructeur. Pour lui, le software doit être utilisé à sa convenance et il est l'ennemi de tout chaînage. Il aime bien mettre au point ses programmes entre deux travaux car il estime qu'il ne fait pas perdre de temps à l'atelier. Lorsqu'il demande 5 minutes d'ordinateur il n'est pas rare de voir ces minutes se transformer en heures. Fervent lecteur des journaux sportifs, portant moustaches fines et nœud papillon haut, dans son costume clair coupé sur mesure, il conduit avec nervosité sa décapotable rouge. C'est l'As des as de la Mécano...

TABLE DES MATIÈRES

Chapitre VI. — **Le software**

Chapitre VII. **Le système d'exploitation à priorités; la multiprogrammation**

Chapitre VIII. — **Le traitement en temps réel et la banque des données**

Chapitre IX. — **Les tris**

Chapitre X. — **L'organisation**

LES MEILLEURS LIVRES D'ÉLECTRONIQUE

CIRCUITHEQUE D'ELECTRONIQUE N° 1 : CIRCUITS INTEGRES LINEAIRES, par **H. Lilen.** — Recueil de schémas d'utilisation : amplificateurs, montages B.F. et Hi-Fi; circuits fondamentaux, sous-ensembles radio et T.V. en A.M. et F.M., montages professionnels et industriels.
196 pages, format 16-24 ..... **33,00 F**

CIRCUITHEQUE D'ELECTRONIQUE N° 2 : CIRCUITS INTEGRES NUMERIQUES, par **H. Lilen.** — Principes et applications des circuits intégrés numériques : familles, fonctions, utilisations.
416 pages, format 16-24 ..... **60,00 F**

CIRCUITHEQUE D'ELECTRONIQUE N° 3 : GUIDE MONDIAL DES CIRCUITS INTEGRES, par **H. Lilen.** — Caractéristiques détaillées et schémas de brochage des circuits intégrés numériques et linéaires. Listes d'équivalence. Indication du fabricant.
256 pages, format 16-24 ..... **60,00 F**

CIRCUITHEQUE D'ELECTRONIQUE N° 4 : THYRISTORS ET TRIACS, par **H. Lilen.** — Principes et applications des thyristors, triacs, diacs, SBS, photothyristors, etc., avec schémas d'applications.
264 pages, format 16-24 ..... **48,00 F**

CIRCUITHEQUE D'ELECTRONIQUE N° 5 : CIRCUITS INTEGRES MOS (principes et applications), par **H. Lilen.** — Le transistor MOS et les diverses technologies utilisées pour les circuits intégrés. Avantages et applications des CI/MOS.
196 pages, format 16-24 ..... **45,00 F**

CIRCUITS DE LOGIQUE, par **R. Damaye.** — Fonctions logiques; algèbre de Boole; simplification des fonctions; décodage et affichage; réalisation et protection des circuits; protection contre les parasites industriels.
372 pages, format 16-24 (2e édition) ..... **48,00 F**

COURS D'ELECTRICITE POUR ELECTRONICIENS, par **P. Bleuler** et **J.-P. Fajolle.** — Un ouvrage écrit pour les étudiants qui se spécialisent en électronique.
388 pages, format 16-24 (2e édition) ..... **39,00 F**

COURS FONDAMENTAL DE LOGIQUE ELECTRONIQUE (l'algèbre de Boole et le calcul binaire dans l'industrie électronique) par **R. Amato.** — Notions d'informatique, calcul logique, algèbre de Boole, logique générale, calcul binaire, logique opérationnelle, applications diverses, annexes.
328 pages, format 16-24 ..... **72,00 F**

INITIATION A L'INFORMATIQUE, par **R. Quinqueton.** — Enseignement de la science de l'informatique à partir d'exemples très simples (hardware et software).
272 pages, format 16-24 ..... **39,00 F**

L'INFORMATIQUE INDUSTRIELLE N° 1 : ACQUISITION ET TRAITEMENT DE DONNEES, par **H. Soubies-Camy.** — Les techniques numériques utilisées en instrumentation électronique dans les systèmes industriels de centralisation de données, de surveillance ou de conduite automatique.
240 pages, format 16-24 ..... **33,00 F**

L'INFORMATIQUE INDUSTRIELLE N° 2 : LOGIQUE ELECTRONIQUE ET CIRCUITS INTEGRES NUMERIQUES, par **R. Damaye.** — Technologie, principe de fonctionnement électrique et logique, schémas d'utilisation des circuits intégrés de logique.
488 pages, format 16-24 ..... **63.00 F**

L'ELECTRONIQUE DANS L'AUTOMOBILE par **G. Bredow.** — Montages de réalisation simple permettant de personnaliser son véhicule en ajoutant à son confort.
96 pages, format 21-14,5 ..... **21,00 F**

MATHEMATIQUES POUR ELECTRONICIENS, par **F. Bergtold.** — Ouvrage spécialisé ne nécessitant pas de connaissances particulières. Les difficultés sont dosées depuis les opérations élémentaires et les équations jusqu'aux imaginaires, au calcul graphique, au calcul différentiel et à l'algèbre de Boole.
324 pages, format 16-24 ..... **42,00 F**

MESURES ELECTRONIQUES, par **A. Haas.** — Notions de métrologie, mesures des grandeurs électriques, mesures des composants actifs et passifs, mesures des amplificateurs, sources de courant stabilisées.
264 pages, format 16-24 ..... **27,00 F**

THEORIE ET PRATIQUE D'AUTOMATISMES NUMERIQUES (de la logique aux ordinateurs), par **Ch. Korsakissok.** — Les trois aspects de la logique : mathématique, élément constituant, utilisation; automatismes logiques, séquentiels, programmés.
296 pages, format 16-24 ..... **48,00 F**

AJOUTER 10 % POUR FRAIS D'ENVOI

SOCIÉTÉ DES ÉDITIONS RADIO

9, rue Jacob, 75006 PARIS — C.C.P. Paris 1164-34

Après une brève description du principe des ordinateurs, cet ouvrage explique l'informatique, son emploi et ses buts. Des exemples simples y donnent ensuite les différents principes de la programmation et de l'analyse. Le lecteur comprendra alors sans peine ce que sont la multiprogrammation, le temps réel, le temps partagé, les banques de données, les programmes utilitaires,... et finalement la place de l'ordinateur dans l'entreprise.

Ce livre lève le voile qui entoure l'informatique. Il démystifie les fameuses barrières du langage et des mathématiques. Tous les chapitres y sont exposés d'une manière simple et claire; en termes compréhensibles pour chacun, et pour les lire, point n'est besoin de posséder des connaissances particulières. Il franchit aussi la barrière constituée par l'ordinateur lui-même. Ce qui compte c'est bien l'usage qu'on en fait. Les utilisations seront bonnes ou mauvaises; l'important en informatique, ce sont les hommes et leur compétence.

Ce cours, bien sûr, s'adresse aux utilisateurs présents ou futurs de l'ordinateur. C'est-à-dire aux membres du service informatique (programmeurs, analystes, organisateurs) mais aussi aux responsables et correspondants informatiques des autres services. Il sera également très utile à l'amateur et à l'étudiant désireux de connaître les principes d'une technique qui, plus qu'à la mode, est indispensable.