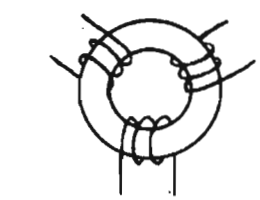

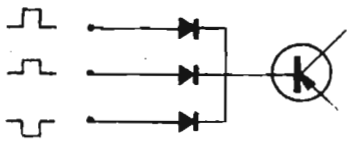

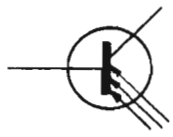

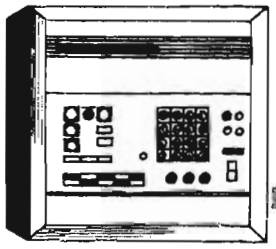

INITIATION AU CALCUL ELECTRONIQUE

COMMENT PARLER AUX MACHINES

LE software a accédé à son troisième âge et la courte histoire de son développement n'est pas rigoureusement parallèle à celle du hardware.

Le software du premier âge n'en portait pas encore le nom ; il s'agissait plus simplement de programmation. Cette dernière était faite par des programmeurs, individualistes comme des artisans, chacun ayant ses propres astuces et étant seul à savoir comment fonctionnait son programme. Ce premier âge dura jusqu'à l'avènement du 1401 IBM, en 1959.

Le second âge est caractérisé par une véritable explosion dans l'emploi des ordinateurs, principalement dans la gestion (naissance du Cobol), mais aussi dans le calcul scientifique (apparition de l'Algol et du Fortran).

Les premières sociétés de software se créent à cette époque et se développent dans un certain désordre ; mais elles ne sont pas véritablement acceptées, car on se fait alors beaucoup d'illusions sur l'avenir de ce mal transitoire, la programmation. De cette époque datent plusieurs mythes, à la vie plus ou moins dure, comme le « Cobol-langage-quasi-naturel-qui-élimine-le-programmeur » ou le « faites-un-programmeur-avec-n'importe-qui-en-trois-semaines ». Même les constructeurs avaient, en ce temps, largement sous-estimé le niveau des connaissances exigibles d'un professionnel de la programmation : certains ont payé très cher cette erreur.

L'avènement du troisième âge ne coïncide pas exactement avec la sortie des ordinateurs de la troisième génération, les mutations technologiques n'ayant pas d'équivalence dans le software.

La fabrication du software passe, dans ce troisième âge, de l'ère artisanale à l'ère industrielle : le software, en plus du « sur-mesure » devient un produit qui se vend sous une forme commerciale, les « packages », quelquefois même par correspondance. L'amateurisme n'a plus sa place dans la profession, la production du software s'organise, se mesure, se contrôle.

Les sociétés atteignent des niveaux industriels, la profession se structure, des chambres syndicales professionnelles sont constituées: les problèmes se posant à cette nouvelle profession sont caractéristiques d'une industrie qui émerge de l'artisanat mais où le pourcentage d'artisans et d'artistes restera toujours beaucoup plus élevé que dans n'importe quelle autre industrie.

PLUS D'UN MILLION D'INSTRUCTIONS

L'utilisation d'un ordinateur se fait dans le cadre d'un software, terme par lequel on désigne :

— Un ensemble de conventions d'utilisation : format de stockage des informations en mémoire principale et auxiliaire, langages de description des algorythmes et des programmes, mode d'exploitation et dialogue avec les opérateurs.

— Les outils associés à ces conventions : manuels de référence, système d'exploitation, programme de service.

Au fur et à mesure que la puissance des ordinateurs a crû, le volume du software a crû. Les raisons essentielles de cette croissance sont multiples :

— La déspécialisation progressive de l'utilisateur, les conventions du software étant de plus en plus proches des problèmes et de l'utilisateur humain ;

— L'intégration progressive des fichiers nécessitant de plus en plus d'outils de base pour la gestion et la sécurité de ces fichiers ;

— L'évolution des méthodes d'exploitation destinées à assurer le meilleur rendement des ordinateurs de plus en plus puissants (multiprogrammation, time-sharing) ;

— La connexion de périphériques éloignés en temps réel.

La croissance du software peut être schématisée par les chiffres suivants du volume du software pour un ordinateur :

1954 : 5 000 instructions ;
1956 : 20 000 instructions ;
1961 : 100 000 instructions ;
1964 : 500 000 instructions ;
1969 : plus d'un million d'instructions.

DE MULTIPLES LANGAGES

Tout problème à résoudre à l'aide d'un ordinateur est défini, par l'utilisateur, dans un langage qui lui est familier. La solution de tout problème traité par ordinateur devra, elle aussi, être formulée dans un vocabulaire adapté à l'utilisateur.

L'ordinateur n'est pas capable de comprendre directement cette formulation. Pour la résolution des problèmes, il a besoin d'un programme, c'est-à-dire d'une succession d'instructions exactes et détaillées, rédigées dans ce que l'on appelle le langage-machine. Les mots de ce langage sont les instructions-machines.

Chaque ordinateur dispose d'un jeu précis et limité d'instructions de cette nature, à l'aide desquelles il faut poser le problème à ré-

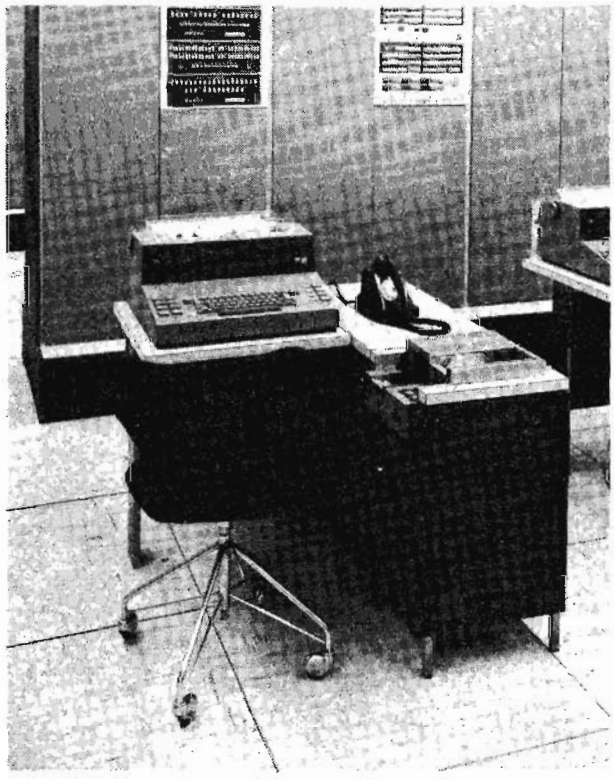

Photo 1

soudre. Le problème doit être programmé, c'est-à-dire que le langage adapté au problème doit être traduit en langage-machine. Une telle opération se déroule en trois phases :

● On analyse tout d'abord le problème ; on définit les procédés et les moyens d'en obtenir la solution.

Prenons, par exemple, le cas du calcul de la racine carrée d'un nombre A quelconque.

Supposons le problème résolu, et soit x la racine carrée du nombre A. Par définition même de la racine carrée, on a :

$x^2 = A$, c'est-à-dire que le carré de la racine carrée d'un nombre est égale au nombre même. Donc :

$$x = \frac{A}{x}$$

On peut encore écrire que :

$$\frac{x}{2} = \frac{1}{2}\frac{A}{x}$$

Donc, comme $x = \frac{x}{2} + \frac{x}{2}$, il vient :

$$x = \frac{x}{2} + \frac{A}{2x}$$

On va prendre le problème inverse. Soit un nombre x_0, et calculons :

$$x_1 = \frac{x_0}{2} + \frac{A}{2x_0}$$

A étant un nombre quelconque ; puis successivement :

$$x_2 = \frac{x_1}{2} + \frac{A}{2x_1}$$

$$x_3 = \frac{x_2}{2} + \frac{A}{2x_2}$$

$$x_n = \frac{x_{n-1}}{2} + \frac{A}{2x_{n-1}}$$

On montre mathématiquement que lorsque n est grand, x_n est à peu près égal à la racine carrée du nombre A.

On vient d'analyser le problème : pour calculer la racine carrée d'un nombre A, on part d'un nombre quelconque x_0, puis on effectue le calcul des nombres x_0, x_1, x_2,... x_n et on arrête le calcul lorsque x^2_n est égal à A, à la précision du calcul près (que l'on se fixera à priori). D'où la seconde étape de la programmation :

● Il faut concevoir et planifier le programme, en établissant un plan de procédure, appelé ordinogramme (Fig. 1).

● Enfin on procède à la codification. On écrit le programme : c'est la formulation du problème dans un langage compréhensible par la machine.

Le langage-machine est constitué d'instructions-machines et de quelques règles d'enchaînement de ces instructions, extrêmement primitives. La langue d'origine, que l'on doit traduire en langage-machine, ne peut pas être la langue courante : celle-ci est trop riche. Il faut donc créer un langage artificiel moins riche, permettant une formulation exacte des problèmes, qui soit néanmoins aussi proche que possible du langage courant ou professionnel. Ce langage artificiel est le langage de programmation (Fig. 2).

La traduction, par la machine, d'un langage de programmation en son propre langage, nécessite un programme de traduction. Ce programme connaît les deux langues. Il en possède, en quelque sorte, un lexique complet, avec toutes les règles de syntaxe. De plus, le programme de traduction doit détecter, et dans la mesure du possible, corriger toutes les infractions aux règles du langage de programmation, c'est-à-dire toutes les erreurs formelles.

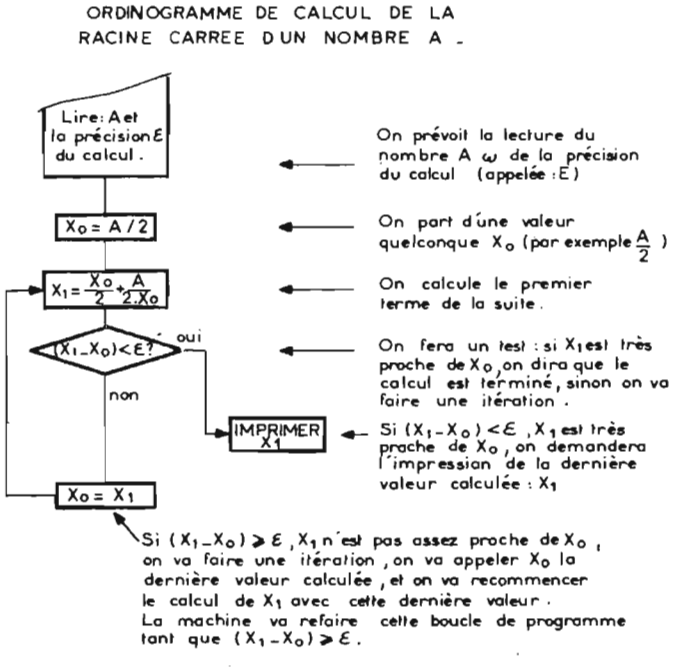

Fig. 1. — *Ordinogramme du calcul de la racine carrée d'un nombre A*

Photo 2. — *Il ne suffit plus de jouer aux cartes (même si celles-ci sont binaires) pour savoir programmer !*

Pratiquement, la traduction en langage-machine s'effectue avant l'exécution du programme.

Les langages de programmation se divisent en deux grandes classes : les uns, dits langages orientés, tiennent compte du type de machine utilisé, et diffèrent donc d'un ordinateur à l'autre. Les autres, langages universels, sont valables sur n'importe quel ordinateur.

LES LANGAGES ORIENTES

Le programmeur utilise des instructions-machine commodes, symboliques, et faciles à comprendre, emploie des abréviations mnémotechniques, telles que ADD pour addition, MULT pour multiplication et il dresse la mémoire avec des noms (prix, taux) qu'il peut choisir librement.

La programmation en langage orienté permet d'optimaliser un programme au point de vue de l'encombrement et du temps machine.

Les langages orientés simples assurent seulement une correspondance biunivoque entre une instruction exprimée symboliquement et l'instruction machine. Dans les systèmes plus évolués, ces langages s'enrichissent de « macro-instructions », constituées d'instructions élémentaires. Le programmeur peut créer lui-même des macro-instructions qu'il utilisera comme des entités linguistiques, au même titre que les instructions élémentaires.

Par exemple, le programmeur aura intérêt à créer une macro-instruction, qu'il appellera par exemple, RAC, permettant le calcul d'une racine carrée. Il n'aura, dans la suite de son programme, qu'à écrire RAC pour calculer une racine carrée.

Le programme de traduction d'un langage orienté s'appelle un assembleur. Il traduit les données en valeurs binaires et leur attribue un emplacement mémoire.

LES LANGAGES UNIVERSELS

Parmi les langages universels les plus connus, on citera : Basic, Algol, Fortran, Cobol.

Le Fortran est sans conteste le langage scientifique le plus employé. Il facilite l'emploi des modes de calculs fondamentaux et permet l'utilisation de fonctions d'usage très courant (racines, sinus, fonctions exponentielles et logarythmiques).

Le Cobol permet d'établir de façon simple des programmes adaptés aux problèmes commerciaux. Un programme Cobol s'articule autour de quatre divisions : identification, environnement, données et procédure. La division « identification » a un rôle documentaire. La division « environnement » permet, avec des modifications simples, de changer la composition du système, ou même le système de traitement lui-même. La division « données » permet d'écrire, d'une façon très souple, les informations à traiter. Enfin, la division « procédure » exprime le déroulement du programme proprement dit.

Le programme de traduction d'un langage universel s'appelle un compilateur. Il fait correspondre à chaque élément du programme en langage universel, une séquence d'instructions-machine.

PROGRAMME EN LANGAGE BASIC DU PROBLEME DE LA RECHERCHE DE LA RACINE CARREE DE A

```
 Ø REM : Calcul de la racine carrée de A        ← titre
1Ø INPUT A, E                                   ← on lit A et ε
2Ø X Ø = A/2
3Ø X 1 = (X Ø + A/X Ø)/2
4Ø IF ABS (X1 – X Ø) < E                        ← test : si (X1 – XO) < ε on va à
   THEN 7Ø                                        la ligne 70, sinon on opère en
                                                  séquence (ligne 50)
5Ø X Ø = X1
6Ø GOTO 3Ø
7Ø PRINT X1
8Ø END
```

CALCUL DE 2 + 2 EN FORTRAN ET PL/1

FORTRAN	PL/1
I = 2 J = 2 K = I + J IMP = 5 WRITE (IMP, 1000) K 1000 FORMAT (1 H,/K =/) STOP END	EXEMPLE : PROCEDURE OPTIONS (MAINS) ; DECLARE (A, B, C) ; A = 2 ; B = 2 ; C = A + B ; PUT DATA (C) ; END EXEMPLE ;

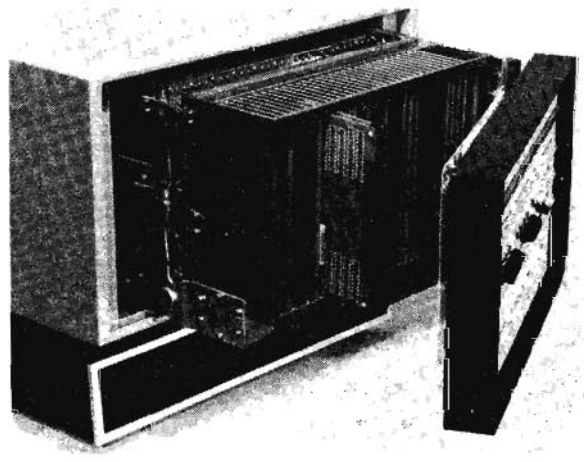

Photo 3. — *L'ordinateur : une machine qui ne comprend pas les langages évolués. Il faut tout lui traduire en langage-machine*

LE PLUS RECENT : LE PL/1

Un effort a été effectué récemment pour bâtir un langage ordinateur universel. C'est en ce sens que l'on peut parler de PL/1 (Programming Language 1). La réussite du PL/1 résulte du fait que ce langage couvre non seulement les domaines d'application du Fortran, Algol et Cobol réunis, mais offre de plus, certaines facilités nouvelles d'écriture (Fig. 3). D'ailleurs, dans PL/1 (comme en Algol) toutes les instructions pourraient être écrites les unes à la suite des autres, pourvu qu'elles soient séparées par un point-et-virgule, ce qui nous rapproche aussi de l'écriture ordinaire.

Le programme PL/1 de la figure 3 porte l'étiquette **exemple** et s'étend du mot **exemple** au mot **end.** L'instruction **déclare,** fournit à la machine la liste des noms des variables utilisées par le programme, réserve les emplacements de mémoire correspondants. **Put data** demande à la machine d'éditer le résultat désiré.

On peut bien sûr envisager d'aller plus loin, de construire un super-compilateur permettant d'extraire d'un programme rédigé en langage libre, une suite d'instructions machine non ambigües. On pourrait même imaginer un dialogue entre l'auteur du programme et la machine, suscité par la machine, qui pourrait souhaiter des précisions ou détecter des impossibilités. Quel que soit l'avenir de ces vues, elles ne vont pas à l'encontre de l'idée qu'il existera toujours un abîme entre la conscience humaine, mélange de raison et de sentiments, et le hardware, « quincaillerie » inerte incapable de saisir autre chose que la forme des messages.

Marc FERRETTI