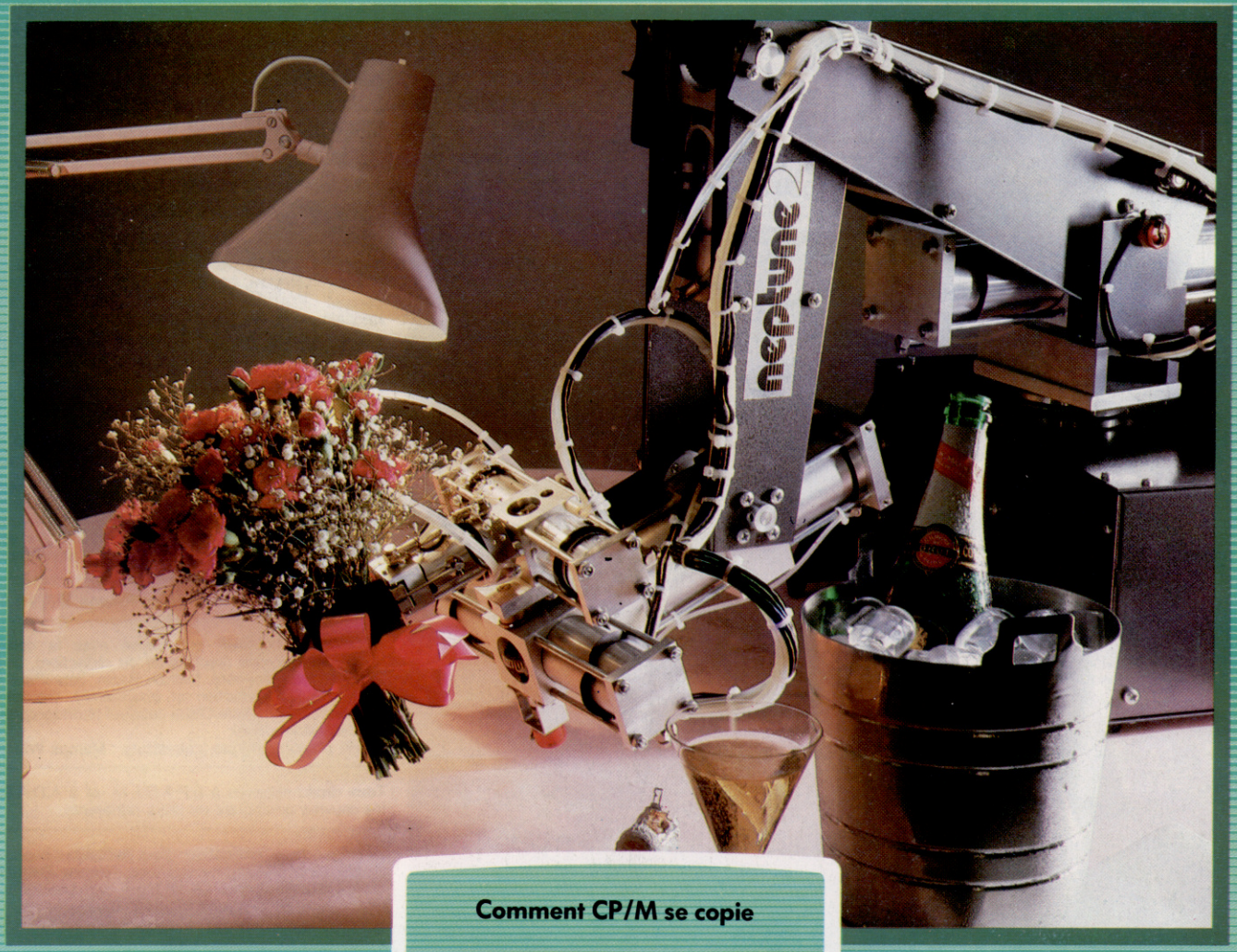
ABC INFORMATIQUE

N° 91

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Comment CP/M se copie

Les débits d'octets

Flux de données sur Spectrum

Des livres sur logo

EDITIONS ATLAS

M 6062-91-12,50 F 90 FB - 3,80 FS - $ 1.95

Apprendre à apprendre

Comment mettre au point un système capable d'apprendre par lui-même? Passons en revue les critères nécessaires à la création d'un tel système, et examinons Beagle, un « moteur d'inférences ».

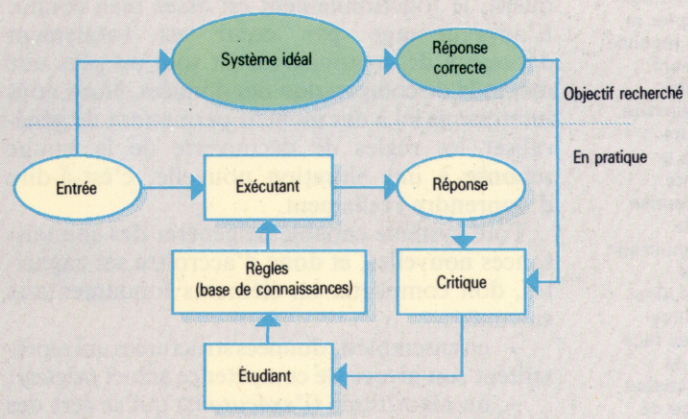
Si, face à une tâche donnée, un ordinateur est capable d'améliorer ses résultats au cours d'une période de temps donnée, on est en droit de dire qu'il a « appris », dès lors qu'il n'a pas été utile de le reprogrammer. Notons cependant que nous aurons besoin d'une norme, une référence, pour mesurer les « progrès » de l'appareil. Apprentissage est un mot vide de sens, si l'on ne dispose pas de procédures d'évaluation. Un algorithme permettant d'apprendre devrait être capable de maîtriser un ou plusieurs des éléments suivants :

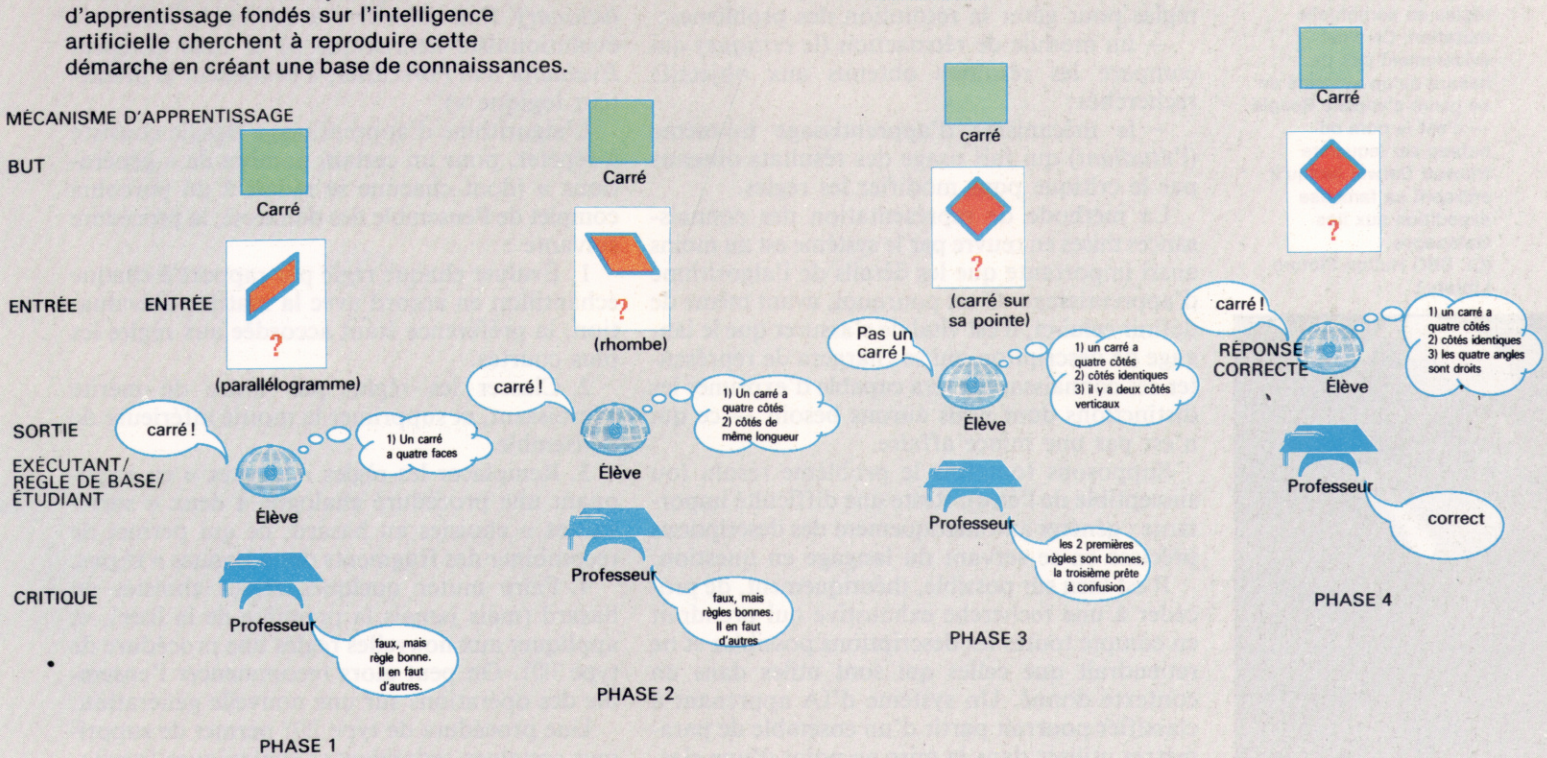
- Couvrir de nombreux problèmes.
- Donner des solutions plus précises.
- Obtenir des réponses à moindre coût.
- Simplifier les connaissances déjà acquises.

Ce dernier point implique que la simplification soit toujours positive, même quand elle ne se traduit pas par une amélioration des performances de l'ordinateur. Cela peut se révéler exact s'il commence avec un ensemble de règles pour terminer avec un autre, aussi efficace, qui sera plus aisément compréhensible pour nous.

Les armes de la critique

On ne sait pas très bien comment les enfants apprennent : le processus est complexe. Les psychologues pensent toutefois qu'ils ont recours à des ensembles de règles, des « schémas » souvent mis au point par essais successifs; les hypothèses (les « règles ») sont mises à l'épreuve, et conservées si elles donnent des résultats corrects. Dans bien des cas, cette méthode est guidée par quelqu'un qui connaît déjà le résultat recherché (le *critique*). Il est ici représenté sous la forme d'un enseignant, qui assiste l'enfant (l'*étudiant*) afin de lui permettre d'évaluer ses propres règles, de façon à accroître ses performances face à une tâche donnée. Les systèmes d'apprentissage fondés sur l'intelligence artificielle cherchent à reproduire cette démarche en créant une base de connaissances.





L'apprentissage automatisé a de nombreux domaines d'application ; les systèmes de ce type ont connu leurs plus grands succès face à des problèmes de classification. Leur objectif reste d'analyser les données de départ et de les identifier, de les interpréter ou de les classifier d'une façon ou d'une autre.

On a déjà cherché à mettre au point diverses méthodes permettant un accroissement automatique de leurs possibilités. Arthur Samuel reste l'un des précurseurs en ce domaine avec des études restées classiques sur le jeu de dames qui recouraient à deux des procédés les plus simples : l'apprentissage par cœur et l'ajustement paramétrique. Avant lui, Rosenblatt et Selfridge, parmi d'autres, avaient déjà créé des systèmes de reconnaissance des formes aux capacités d'apprentissage rudimentaires, comme le Perceptron.

L'ajustement paramétrique a pour principe de base l'ajustement répété des coefficients et des paramètres du programme, afin d'améliorer ses résultats. C'est en fait une simple technique d'optimisation, étudiée de façon exhaustive en mathématiques appliquées, et dont, par conséquent, le fonctionnement est assez bien connu. L'apprentissage par cœur est totalement dépourvu de créativité : c'est tout au plus une méthode de compression des données. Nous nous intéressons ici à des moyens permettant de généraliser les règles de découverte de la bonne réponse à une situation nouvelle, c'est-à-dire d'apprendre réellement.

Tout système capable de générer des connaissances nouvelles, et donc d'accroître ses capacités, doit comporter les éléments fondamentaux suivants :

– un ensemble de données structurées qui représentent son niveau de compétence actuel *(règles)* ;

– un algorithme (l'*exécutant*) qui se sert des règles pour gérer la résolution des problèmes ;

– un module de rétroaction (le *critique*) qui compare les résultats obtenus aux objectifs recherchés ;

– le mécanisme d'apprentissage lui-même (l'*étudiant*) qui fait usage des résultats obtenus par le critique pour modifier les règles.

La méthode de représentation des connaissances mises en œuvre par le système est au moins aussi importante que les détails de l'algorithme d'apprentissage. C'est pourquoi, avant même de définir celui-ci, il est vital de s'assurer que le langage de description qui se chargera de représenter ces connaissances sera capable d'exprimer les distinctions dont nous aurons besoin — ce qui n'est pas une mince affaire.

Supposons toutefois le problème résolu (ou susceptible de l'être). Reste une difficulté importante : générer automatiquement des descriptions précises en se servant du langage en question.

Il est bien sûr possible, théoriquement, de procéder à une recherche exhaustive qui prendrait en compte toutes les descriptions possibles et ne retiendrait que celles qui sont utiles dans un contexte donné. Un système d'IA apprenant à classifier pourrait partir d'un ensemble de paramètres utilisés dans la mise au point d'une classification. Dans sa phase d'apprentissage, il pourrait générer et évaluer d'autres descriptions en combinant d'une façon ou d'une autre les paramètres d'origine, retenant celles qui aident à définir une classification correcte et négligeant les autres. Mais, hélas, leur nombre peut être astronomique ; plus le langage de description est expressif, plus ce problème combinatoire se révélera explosif.

De toute évidence, il nous faudra trouver un moyen de guider la recherche et donc d'ignorer la très grande majorité des descriptions potentielles qui ne présentent aucun intérêt. Un certain nombre de procédés sont tout à fait performants dans des situations d'apprentissage où les classifications sont bien définies, et où les zones d'incertitude entre classes sont très restreintes. Mais l'information comporte toujours des « parasites », et les choses se compliquent aussitôt. Voyons d'un peu plus près une méthode assez simple capable de traiter ce type de problème.

Beagle (*Biological Evolutionary Algorithm Generating Logical Expressions,* « algorithme évolutionniste biologique générateur d'expressions logiques ») est un système informatisé qui génère des règles de prises de décisions par induction, à partir d'une base de données. Il prend donc en compte un problème intéressant : d'où viennent les informations de base ?

Beagle fonctionne suivant le principe de la « sélection naturelle » : les règles qui ne sont pas conformes aux données sont annihilées, et remplacées par des « mutations » de règles supérieures, ou par des règles nouvelles, nées de la fusion de deux règles mieux adaptées. Il s'agit en fait d'expressions booléennes représentées par des structures arborescentes.

Les logiciels d'origine, écrits en PASCAL, étaient au nombre de deux : Herb (*Heuristic Evolutionary Rule Breeder,* « moteur d'inférence évolutionniste heuristique ») et Leaf (*Logical Evaluator and Forecaster,* « évaluateur et prédicteur logique »).

L'algorithme d'apprentissage Beagle consiste à répéter, pour un certain nombre de « générations » (dont chacune se réduit à un parcours complet de l'ensemble des données), la procédure suivante :

1. Évaluer chaque règle par rapport à chaque échantillon en accord avec la matrice d'évaluation, la préférence étant accordée aux règles les plus courtes.

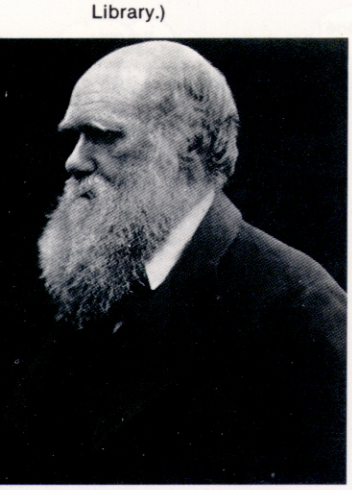
2. Classer les règles par ordre de mérite décroissant, et supprimer la moitié inférieure de l'ensemble.

3. Remplacer les règles « mortes » en appliquant une procédure analogue à deux « survivantes » choisies au hasard, ce qui permet de recombiner des fragments des « bonnes » règles.

4. Faire muter quelques règles choisies au hasard (mais jamais la première de la liste), et appliquer aux nouvelles règles une procédure de type TIDY. On peut alors recommencer l'ensemble des opérations sur une nouvelle génération.

Une procédure de type TIDY permet de supprimer certaines redondances syntaxiques inévita-

Charles Darwin (1809-1882)
Pour la théorie de l'évolution darwinienne, c'est la sélection naturelle qui permet aux espèces de s'adapter et de s'améliorer en réponse à leur environnement. Seuls les plus forts peuvent espérer survivre et transmettre leurs caractéristiques à la génération suivante. Cette idée a été reprise avec succès par la théorie de l'apprentissage automatique : des systèmes d'IA ont ainsi pu réussir à améliorer leurs performances face à des problèmes de classification en faisant usage d'ensembles de règles en perpétuelle mutation. Ce n'est évidemment pas par hasard qu'un système de ce genre s'appelle Beagle — c'est le nom du bateau sur lequel se trouvait Darwin lorsqu'il entreprit sa fameuse expédition aux îles Galápagos.
(Cl. BBC Hulton Picture Library.)



bles, ainsi les doubles négations et les expressions constantes ; l'ensemble des règles, qui a une structure arborescente, est « élagué » mais reste presque semblable à ce qu'il était, à ceci près que la formulation est désormais plus succincte. Toute cette méthode repose sur les idées darwiniennes relatives à l'évolution : seules les règles les mieux « adaptées » survivent à la génération suivante. De cette façon, on peut déterminer si un élément appartient à une classe recherchée.

Prévision météo

Un bon exemple de système capable d'apprendre est un programme pouvant utiliser les données relatives au temps qu'il fait aujourd'hui afin de prédire si, oui ou non, il pleuvra demain. Il doit commencer avec une série de données de base : chutes de pluie, temps d'ensoleillement, vitesse maximale du vent, pression atmosphérique (le tout étant relevé, disons, à midi). Afin que l'ordinateur puisse apprendre les règles qui lui permettront de faire des prédictions exactes, il doit être alimenté en données d'apprentissage, qu'il faut donc recueillir préalablement. Chaque enregistrement précisera s'il a plu le lendemain. On aura par exemple :

Chute de pluie	0
Ensoleillement	7,2
Vent	22
Pression	1017

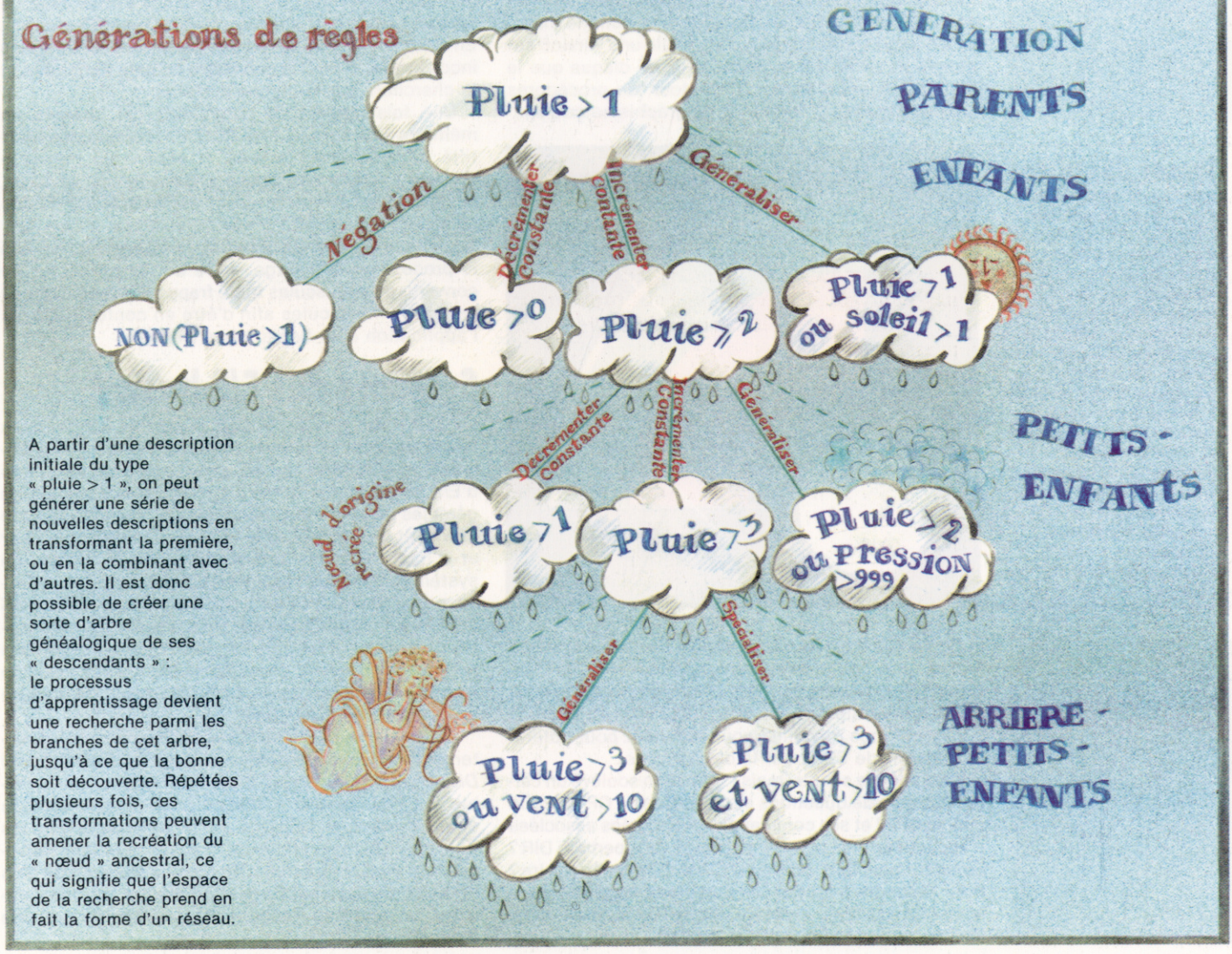
Le système doit créer des règles (au besoin, dans le premier cas, au hasard), et les vérifier en fonction des données. Les règles peuvent servir d'éléments, ainsi dans :

Caractéristiques	Vent
Constantes	10
Opérateurs de comparaisons	> ou <
Opérateurs logiques	ET, OU et NON

Le tout peut être combiné dans des expressions du genre :

ensoleillement < 4 ET pression < 1 000.

Après chaque parcours de l'ensemble de données, les bonnes règles peuvent être conservées, et les mauvaises abandonnées. Elles sont alors remplacées par d'autres. L'ensemble « évolue » à chaque fois et accroît ses performances ; peu à peu l'ordinateur devrait donc améliorer la précision de ses prévisions du temps.



A partir d'une description initiale du type « pluie > 1 », on peut générer une série de nouvelles descriptions en transformant la première, ou en la combinant avec d'autres. Il est donc possible de créer une sorte d'arbre généalogique de ses « descendants » : le processus d'apprentissage devient une recherche parmi les branches de cet arbre, jusqu'à ce que la bonne soit découverte. Répétées plusieurs fois, ces transformations peuvent amener la recréation du « nœud » ancestral, ce qui signifie que l'espace de la recherche prend en fait la forme d'un réseau.

Gabrielle Izen

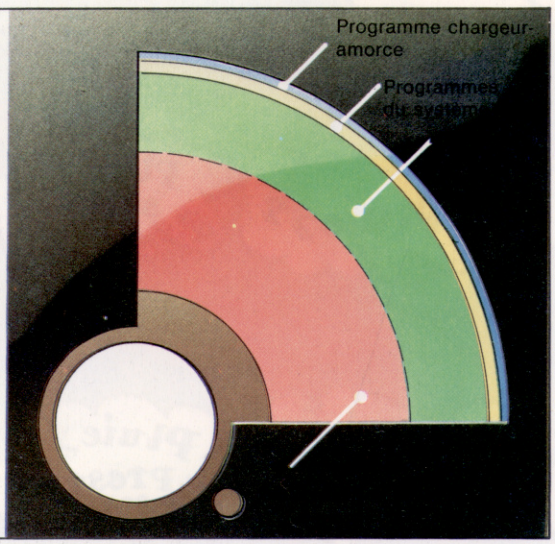
Encore le même

La commande SYSGEN permet à CP/M de se copier lui-même d'un disque à l'autre. Suivons le flux des informations à travers le système.

Nous avons vu jusqu'ici les diverses fonctions disponibles sous CP/M, et nous sommes maintenant à même d'effectuer toutes les fonctions importantes, en micro-informatique professionnelle ou familiale. Nous pouvons charger, éditer, sauvegarder et transférer les divers types de fichiers définis par CP/M, et aussi les examiner. Il reste cependant une fonction vitale que nous n'avons pas effectuée, la copie du système d'exploitation CP/M lui-même. Souvenez-vous que CP/M est conçu pour être « transparent » à l'utilisateur : il n'apparaît pas sur un listage normal de répertoire, ce qui signifie qu'on ne peut y avoir accès par les opérations des commandes transitoires. Il est essentiel que nous puissions transférer le système sur d'autres disques, non seulement pour disposer d'une copie de sauvegarde au cas où le disque d'exploitation viendrait à être endommagé, mais également parce que de nombreuses applications supposent le transfert d'une version CP/M sur le disque. On peut citer comme exemple Dr. Logo sur Amstrad qui accueille une version sur mesure de CP/M sur la même face de disque que le langage, lequel dispose, à son tour, de procédures supplémentaires d'édition et de graphisme couleur.

Tracer des pistes

Bien que la quasi-totalité d'un disque de données CP/M soit disponible pour les fichiers utilisateur, les pistes les plus externes sont réservées au système d'exploitation. La piste zéro contient le programme chargeur-amorce qui charge automatiquement le CP/M à l'allumage ou après réinitialisation. Les pistes 1 et 2 contiennent le système CP/M. Viennent ensuite les pistes répertoire dont les FCB indiquent exactement l'emplacement des enregistrements des fichiers.



Kevin Jones

Copier CP/M est tout à fait simple : le disque système fournit une commande transitoire appelée SYSGEN. L'exécution de cette dernière affiche une suite de messages à l'opérateur. Selon la configuration du système sur lequel vous travaillez, vous répondez pour obtenir le transfert de CP/M sur le disque cible.

Cela semble simple, mais l'opération soulève un certain nombre de questions sur la nature de CP/M. Pourquoi le CP/M et ses commandes incorporées associées n'apparaissent-ils pas sur un listage de répertoire DIR ? En allant plus loin, pourquoi CP/M lit-il un répertoire ? Pour répondre à toutes ces questions importantes, il nous faut examiner en détail ce que l'on pourrait appeler les nœuds et les verrous de CP/M.

Le système d'exploitation chargé dans l'ordinateur se divise en trois parties, chacune étant responsable d'un domaine particulier du fonctionnement du disque. La partie dont vous êtes le plus familier est le processeur de commandes de la console (PCC). Il s'agit en effet de la partie visible du système, l'endroit d'où partent les informations à destination du moniteur vidéo, et où les commandes entrées au clavier sont interprétées. Le PCC est donc la face visible de CP/M.

En réalité, il en fait plus dans les coulisses. Lorsque des informations sont tapées au clavier, le processeur suppose qu'il s'agit de commandes. Les informations sont donc transférées, via une autre partie du système appelée BIOS *(Basic Input/Output System)*, « gestion des Entrées/Sorties », que nous verrons ultérieurement), vers la zone-tampon du PCC. Ce dernier passe alors en revue la liste des commandes résidentes pour vérifier que la commande Entrée y figure. Si c'est bien le cas, elle est immédiatement exécutée. En revanche, si elle ne fait pas partie des commandes incorporées, le PCC la considère comme transitoire et la cherchera sur le disque système.

Une fois repérée, la commande est chargée en mémoire et exécutée. Si le PCC n'a toujours pas identifié la commande passée, il génère un message d'erreur « fichier non trouvé ». Dans le cas de CP/M, le PCC affiche le nom de la commande en majuscules, suivi d'un point d'interrogation. La raison pour laquelle la commande apparaît en majuscules est que le processeur se charge, entre autres fonctions, de convertir en majuscules toute frappe d'une commande en lettres minuscules afin d'être en conformité avec l'application d'un fichier sous CP/M.

Système des E/S basic

Le BIOS est constitué d'un ensemble de routines (connues sous le nom de sous-programmes de commandes — *drivers*), qui assurent toutes les fonctions d'E/S de CP/M. Sa raison d'être fondamentale est de gérer les périphériques connectés au système (dont le clavier et le moniteur — le PCC se contente d'envoyer et d'interpréter les informations via le BIOS). Le système utilise les paramètres fournis par le processeur. Pour adapter CP/M à une machine, c'est le BIOS qu'il faut modifier. La raison en est (outre les contraintes propres au CP/M), que les méthodes de gestion des périphériques varient considérablement d'une machine à l'autre. La modification du BIOS permet de prendre en compte ces différences.

La troisième partie du CP/M est responsable directement de la gestion disque. Il s'agit du BDOS *(Basic Disk Operating System)*. C'est l'élément le moins visible de CP/M. Sans lui, CP/M ne pourrait atteindre son objectif principal. Le BDOS est le logiciel qui gère les fichiers du disque, alloue de l'espace pour leur sauvegarde et tient le répertoire. En prélude d'une étude plus détaillée du système de gestion disque, voyons comment se distribue l'espace de sauvegarde sur une disquette.

En termes matériel, un disque comprend quarante pistes. Les versions modernes de CP/M prolongent cette subdivision en secteurs logiciel, ainsi nommés du fait que c'est le logiciel qui détermine les secteurs. Les disques système plus anciens subdivisaient les pistes en secteurs matériel lus mécaniquement par la machine. Naturellement, les pistes les plus extérieures d'un disque comportent davantage de secteurs que les pistes plus internes. La plupart des pistes d'un disque servent à sauvegarder des informations.

Moniteur résident

Le CP/M se réserve quant à lui trois pistes (sur une disquette 5 pouces). Ce sont les pistes les plus externes (numérotées de 0 à 2). Elles comportent le court programme chargeur-amorce *(bootstrap loader),* appelé également le moniteur résidant, qui permet au CP/M de se charger lui-même dans l'ordinateur en l'absence d'un système d'exploitation. Ces pistes contiennent également le système CP/M lui-même. C'est l'information de ces trois pistes qui est mise en place sur le disque par la commande SYSGEN.

Chaque secteur (appelé « enregistrement » en termes CP/M) comporte 128 octets d'information. Les enregistrements peuvent être groupés en unités contenant chacune jusqu'à 128 enregistrements. Un fichier CP/M peut en comporter 16 au maximum. Un fichier est donc constitué de 128 × 128 × 16 octets (256 K).

A moins d'être vraiment minuscule, un fichier ne figurera pas sur un seul enregistrement. Il ne sera pas non plus possible, dans la pratique, de faire figurer tous les enregistrements d'un fichier par ordre séquentiel sur le disque. C'est pourquoi un système d'exploitation doit pouvoir se donner une méthode d'identification des secteurs (quels secteurs correspondent à tel fichier).

Cette information s'appelle, pour le CP/M, le bloc de contrôle fichier (FCB). Sur une disquette 5 pouces, ces FCB figurent sur la quatrième piste, appelée piste répertoire. Le système de gestion disque (BDOS) gère cette méthode d'indexation des fichiers (sa lecture, son écriture et ses modifications). Un bloc de contrôle fichier comporte jusqu'à 33 octets d'information et contient toutes les données nécessaires à l'identification et à la localisation de n'importe quel fichier par le BDOS sur le disque.

Le FCB comporte d'abord l'information de saisie, un seul octet qui donne l'identification du lecteur où réside le disque du fichier. Les huit prochains octets contiennent le nom du fichier. Ils sont suivis de trois octets réservés à l'extension du fichier. Les octets 12 et 14 indiquent la longueur de ce dernier, le quinzième contient le compte du nombre d'enregistrements. Les 15 octets suivants comportent la table d'allocation des secteurs pour le fichier. Enfin, le dernier octet donne le prochain numéro d'enregistrement, lorsque le fichier occupe plus de 16 K (taille maximale que peut décrire un bloc de contrôle).

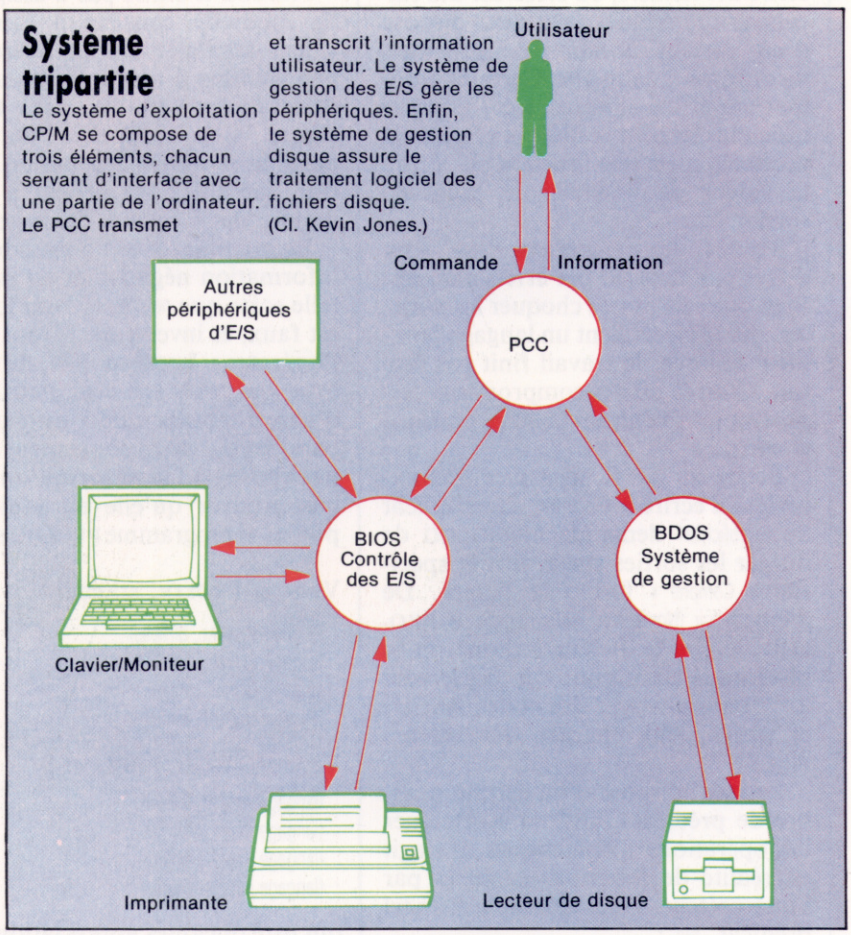
Les fichiers

Lorsqu'un fichier est ouvert par l'exécution d'une commande, le processeur des commandes de la console crée une version du FCB dans la zone mémoire habituellement réservée aux commandes transitoires. Les autres informations qui concernent le fichier, sa longueur, son nombre d'enregistrements et l'allocation de ses secteurs, sont fournies par le système de gestion disque (BDOS) lorsqu'il trouve le fichier (c'est-à-dire un fichier de même nom que celui créé par le PCC). L'intérêt d'un FCB en mémoire est que les données qu'il contient peuvent être rapidement mises à jour par le BDOS lorsque le fichier est utilisé par les commandes de CP/M. La version définitive du FCB est écrite sur disque par le BDOS à la fin de ces opérations, en tenant compte de la modification survenue dans la longueur de l'enregistrement et dans l'allocation des secteurs.

Nous pouvons maintenant revenir sur la manière dont CP/M restitue les fichiers. Lorsque le processeur reçoit une commande en provenance du terminal, il interprète le premier mot comme une instruction. Si la commande n'est pas résidente en mémoire, le processeur transmet le nom du fichier au système de gestion disque, via le système « messager » de traitement des E/S. Le BDOS passe alors en revue la piste répertoire jusqu'à ce qu'il trouve une occurrence correspondant au nom du fichier dans un des FCB. Si le fichier est de type COMmande, le système de gestion disque examine la table d'allocation des secteurs du disque pour découvrir quels secteurs il occupe. Il les trouve et les copie en mémoire. C'est alors que la commande est exécutée.

Système tripartite

Le système d'exploitation CP/M se compose de trois éléments, chacun servant d'interface avec une partie de l'ordinateur. Le PCC transmet et transcrit l'information utilisateur. Le système de gestion des E/S gère les périphériques. Enfin, le système de gestion disque assure le traitement logiciel des fichiers disque. (Cl. Kevin Jones.)



Cela signifie que le BDOS doit rechercher un fichier différent pour le traitement, et qu'il le trouvera et le chargera également. Lorsqu'un fichier est chargé en mémoire, le système de gestion disque fournit également les données appropriées de FCB à une version vide de FCB mise en place par le processeur. Ce bloc de contrôle fichier sera mis à jour avec le déroulement de l'exécution. A la fin du programme de commande, la version mise à jour est recopiée par le BDOS.

Nous étudierons prochainement comment les divers composants de CP/M sont disposés dans la mémoire de l'ordinateur, et comment cet espace mémoire est organisé pour tirer profit au mieux du peu d'espace disponible.

Effets annexes

Nous allons analyser comment prolog incorpore des effets annexes et des propriétés « non logiques » tels que les opérations « couper » et « échec ».

PROLOG est avant tout un langage logique qui ne convient pas à certaines fonctions procédurières. La logique des prédicats s'accommode en effet difficilement de certaines fonctions nécessaires à un langage de programmation : par exemple, lire et écrire des fichiers, ou effectuer des calculs arithmétiques. PROLOG dispose d'un certain nombre de prédicats incorporés à cette fin. Certains utilisent des effets annexes, écrire (Terme) par exemple. Ils sont réalisés et effectuent accessoirement une fonction (ici, écrire la valeur de Terme sur le canal de sortie).

Toutes les entrées/sorties sont gérées par PROLOG par effets annexes. Bien que cela puisse choquer les puristes, qui préféreraient un langage purement logique, le travail finit par être fait. C'est là un des compromis nécessaire pour obtenir un langage pratique et efficace.

Écrire sur des fichiers avec PROLOG revient à écrire à l'écran. Un prédicat dire (Nomfichier) demande à PROLOG de diriger les sorties sur le fichier spécifié. « Compris » ferme le fichier. De même, voir (Nomfichier) demande à PROLOG de lire le fichier indiqué, et vu restaure le statu quo. On utilise mettre (Code) pour envoyer des codes ASCII, et prendre pour charger des valeurs ASCII.

PROLOG dispose d'un certain nombre de prédicats destinés à effectuer des opérations arithmétiques. Il s'agit en réalité de fonctions évaluées par l'interpréteur. Voici l'exemple d'un tel prédicat, fait :

```
C1 fait C + 1.
```

Ce qui équivaut, en BASIC, à :

```
C1 = C + 1.
```

Remarquez que C fait C + 1 échouerait toujours avec PROLOG, alors qu'avec le BASIC, C = C + 1 fonctionne. Cela est dû au fait que des langages tels que le BASIC, le FORTRAN, le PASCAL, l'ALGOL et le langage C utilisent des affectations destructrices : en assignant le résultat de l'expression (la partie droite de l'affectation) à la variable (la partie gauche), la valeur qu'avait cette dernière est recouverte en écriture. C'est pourquoi le fait que la même variable apparaisse dans la partie droite de l'expression n'a pas d'importance.

PROLOG n'utilise pas d'assignation. Sa méthode consiste à étendre la valeur attribuée une première fois à une variable à toutes les occurrences de la même variable, dans toute la clause. Cela s'appelle l'unification après la première attribution. Aussi, dire que C fait C + 1 revient à vouloir donner deux valeurs distinctes à C.

Le prédicat non sert à introduire une information négative et est défini de telle sorte que non (X) est vrai lorsque X est faux, et inversement, tout comme l'opérateur booléen NON du BASIC. Pour que non (X) soit vrai, PROLOG doit d'abord trouver que X est faux. Il y parvient par un raisonnement simple : une chose est fausse lorsqu'on ne peut pas prouver qu'elle est vraie. Supposons le programme PROLOG suivant :

Version PROLOG standard :

```
langage (PASCAL, difficile)
langage (COBOL, difficile)
langage (BASIC, simple)
langage (PROLOG, simple)
```

Version MICRO-PROLOG :

```
(langage PASCAL difficile)
(langage COBOL difficile)
(langage BASIC simple)
(langage PROLOG simple)
```

qui liste quelques appréciations sur la facilité de programmation de divers langages. Nous pourrions poser des questions telles que :

```
langage(BASIC, DIFFICULTÉ)
```

où DIFFICULTÉ est une variable. PROLOG répond :

```
DIFFICULTÉ = SIMPLE.
```

Mais si nous demandons :

```
langage(BASIC, facile).
```

PROLOG nous répond non. En effet, le but que nous nous sommes fixé ne peut être prouvé, n'étant pas explicitement indiqué dans la base de données ni susceptible d'être déduit à partir d'autres faits et règles. PROLOG ne comprend pas, bien sûr, la signification des mots. Il ne peut distinguer simple et facile, et encore moins comprendre que les deux mots disent la même chose... En outre, il considère que sa base de données est un univers clos. C'est-à-dire que tout ce qui peut être connu l'est certainement, et donc que tout ce qui ne peut être prouvé est nécessairement faux.

L'idée de la négation considérée comme étant établie par l'échec est un des points faibles de PROLOG. Elle est due à la méthode de preuve logique utilisée par l'interpréteur. Cette méthode s'appelle résolution et ne peut traiter directement les informations négatives. Prenons par exemple la règle :

```
non(A) :- B, C, D.
```

Elle dit que A n'est pas le cas recherché si B, C et D sont vrais. Cela ne peut s'écrire avec PROLOG car une négation ne peut figurer en tête de clause.

PROLOG n'est pas un langage purement déclaratif. Nous avons déjà vu comment la disposition des clauses du code source peut affecter le déroulement du programme, notamment pour les procédures récursives. En fait, l'orthodoxie de sa logique peut être détournée par l'utilisation de ses caractéristiques hors-logique échec et couper. Le premier, échec, est un prédicat qui entraîne systématiquement l'échec. Quel intérêt ? Sa principale raison d'être est d'empêcher PROLOG de s'interrompre. Si nous reprenons l'exemple du programme des langages, et si nous demandons :

```
langage(Lang,simple).
```

PROLOG répond :

```
Lang = BASIC
```

et s'interrompt. Il trouve la première clause dans sa base de données. Elle correspond au but fixé. Ayant prouvé

l'objectif, PROLOG ne va pas plus loin. Et pourtant, nous aurions bien aimé avoir la liste des langages simples. Nous pouvons l'obtenir en ajoutant une règle telle que :

Version PROLOG standard :

```
listesimple :– langage(Lang,simple),
écrire (Lang),nl, échec
```

version MICRO-PROLOG :

```
((listesimple)(langage X simple) (PX)
PP ECHEC))
```

Si nous tapons listesimple, PROLOG répond :

```
BASIC
PROLOG
non
```

PROLOG a d'abord essayé de prouver le but listesimple en prouvant langage(Lang,simple). Il trouva le fait langage(BASIC,simple) et donna la valeur BASIC à la variable Lang. Ainsi, le premier sous-objectif était prouvé. Il passa au suivant, et, avec BASIC pour Lang, trouva écrire(BASIC). Parce que écrire réussit automatiquement (écrivant subsidiairement BASIC), il atteignit le dernier sous-objectif, échec. PROLOG naturellement échoue et revient à écrire. Puisque écrire et les autres prédicats ne sont pas une deuxième fois couronnés de succès en remontant dans le programme, PROLOG va plus haut. Il remonte jusqu'à langage(Lang, simple) et s'efforce de trouver un autre moyen de le prouver.

L'autre caractéristique, couper (écrit !), sert également à contrôler cette démarche qui consiste à remonter dans le programme. C'est un prédicat qui, cette fois, réussit à tous les coups. Cependant, comme effet annexe, il empêche l'interpréteur de remonter plus haut. Les choix de valeurs pour les variables ayant eu lieu précédemment dans une procédure sont gelés. Il empêche également qu'une autre clause de la même procédure ne soit essayée comme alternative. Une utilisation importante de couper est en conjonction avec échec pour s'assurer qu'une clause ayant échoué ne soit pas à nouveau sollicitée. Nous pouvons définir un prédicat manque par ! et échec :

version PROLOG standard :

```
manque(Clause) :– Clause, !, échec.
manque(Clause).
```

version MICRO-PROLOG :

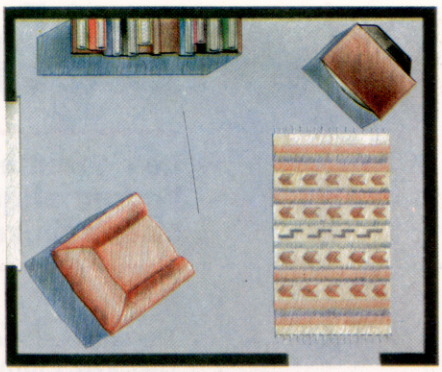
```
((manque X) (?X)/ÉCHEC)
(manque X)
```

est vrai lorsque la clause donnée en argument manque à la base de données, et faux dans le cas inverse. En mettant cela en œuvre dans notre programme sur les langages, nous pouvons demander :

```
(manque(langage(BASIC,simple)).
```

Pour prouver manque(langage(BASIC,simple)), nous devons prouver langage(BASIC,simple). Ce dernier est vrai, figurant dans la base de données.

Sans le prédicat couper, PROLOG serait remonté dans le programme et aurait essayé de trouver un autre moyen de satisfaire Clause. Et couper se dresse sur son chemin; mais il n'y a pas d'autre manière d'atteindre le but, c'est l'échec. En outre, à cause de couper, PROLOG ne cherche pas d'autres clauses et manque(Clause) se révèle faux (parce que Clause était vrai), et nous n'avons pas perdu de temps à chercher d'autres solutions.



KEVIN JONES

De place en place
Nous mettons au point un programme qui déplace un robot dans une pièce (le dessin ci-dessus). Les positions sont sauvegardées selon le format « position(porte) », etc. Le robot se déplace par la procédure « aller-(position1, position2) ».
Ce qui suppose de prouver d'abord le sous-objectif « position (position 1) », c'est-à-dire, prouver que la position 1 existe et que le robot peut en partir. C'est bien le premier objectif à établir avant de mettre en place un plan de route.

Variantes de prolog

La version de PROLOG utilisée ici s'appelle DEC-10 PROLOG (selon le nom de l'ordinateur qui mit en œuvre en premier ce langage). Elle est très largement reconnue comme étant un standard. Malheureusement, la version plus répandue de PROLOG (MICRO-PROLOG) est une des rares à être très différente du standard. L'autre variante importante, PROLOG-1, est très proche du standard DEC-10 (comme son successeur PROLOG-2). Les programmes écrits avec MICRO-PROLOG figurent entièrement sous forme de listes. Ils ressemblent beaucoup en cela à LISP. Cependant, le programmeur n'a pas à utiliser cette notation compliquée : il exécute un programme d'en-tête appelé « Simple ». « Simple » est écrit avec MICRO-PROLOG et fournit une syntaxe alternative.
Il existe de nombreuses petites différences entre MICRO-PROLOG et la version standard, ce qui rend les deux versions très différentes. MICRO-PROLOG met le prédicat en premier dans la liste. Par exemple :

```
origine_de(Mars, Martiens).
```

donne :

```
(origine_de Mars Martiens)
```

On peut préférer l'une ou l'autre des syntaxes. MICRO-PROLOG laisse de côté les signes tels que « :- », ou les virgules. Une clause comme :

```
origine_de(Planète, Créature):–
  née(Créature,Ville)
  sur(Ville,Planète).
```

s'écrirait de la sorte :

```
((origine_de x y) (née y z) (sur z x))
```

ce qui peut se lire comme une liste constituée de trois sous-listes, la première étant l'en-tête de la clause. Le programme « Simple » rend plus compréhensible cette lourde syntaxe. Avec sa notation d'implantation pour les prédicats, la clause va alors devenir :

```
x origine_de y si y née z et z sur x
```

Cela serait encore plus facile à comprendre si MICRO-PROLOG n'acceptait pas que x, y, z, X, Y et Z (ou l'un de ceux-ci suivis d'un chiffre), comme noms de variables.
Parmi les autres différences, MICRO-PROLOG utilise des parenthèses et non des crochets pour les listes, il sépare les arguments et les éléments de liste par des espaces et non des virgules; il utilise «/» au lieu de «!» pour « couper », et donne les prédicats incorporés en majuscules.
Ces derniers ont également des noms non standards, et quelquefois ne font pas exactement la même chose; mais tout cela est très courant entre versions d'un même langage de programmation. MICRO-PROLOG comporte néanmoins une caractéristique très utile que n'a pas DEC-10 PROLOG (mais d'autres versions en disposent avec la syntaxe standard) : il nous permet de créer des modules de programme.
Les modules permettent l'écriture de programmes sous la forme d'unités fonctionnelles.
Ces dernières peuvent ensuite être librement combinées sans avoir à se préoccuper de conflits éventuels de noms.
Cela donne à PROLOG une certaine structure, même si MICRO-PROLOG est loin d'être un vrai langage structuré.

Débit d'octets

Les commerçants viennent de plus en plus nombreux à l'informatique. La SEITA propose aux débits de tabac des services et des moyens adaptés à leurs besoins.

De nombreux commerçants sont déjà familiarisés avec l'informatique, dont ils ont découvert tous les avantages, en particulier pour la gestion, la comptabilité, voire la correspondance avec les clients et les fournisseurs. Les logiciels spécialisés ne manquent pas dans ce domaine et surtout leur qualité s'améliore.

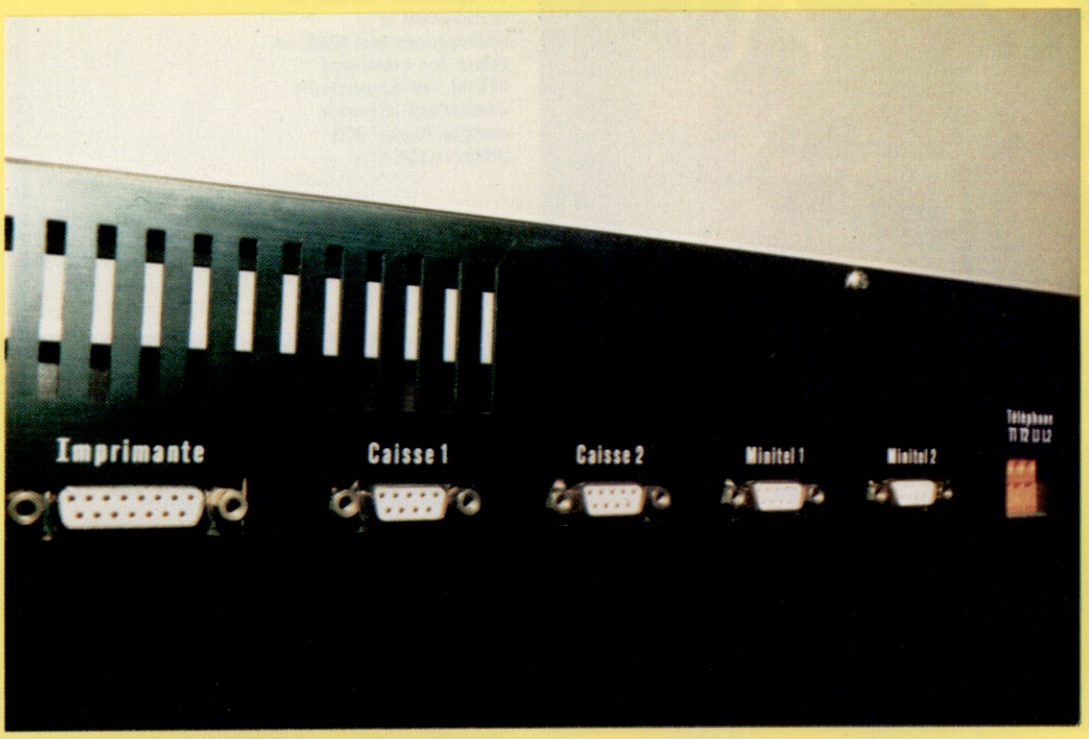
Depuis le début de 1985, la SEITA (Société d'exploitation industrielle des tabacs et des allumettes) a commencé à mettre en place des terminaux points de vente dans les bureaux de tabac. Ces équipements, appelés SEITAMAT, ont été conçus par les ingénieurs informaticiens de la SEITA, en collaboration avec la société Spinel, qui a réalisé l'étude matériel, et Le Comptoir des programmes (L.C.P.) pour le logiciel.

L'ensemble comprend une caisse enregistreuse, Strator 20, permettant la saisie des opérations de caisse, la mémorisation des articles en vente, et disposant d'une autonomie d'au moins une journée. Cette caisse, conçue à partir d'une machine Olivetti Mercator 50, comprend un microprocesseur 6809, une sortie série RS232 C, une mémoire de 16 K CMOS, 64 K RAM et 32 K PROM comprenant le programme. Mille articles peuvent être mémorisés, dont cinq cents en mémoire CMOS et cinq cents en RAM. Le grand clavier comporte trente-cinq touches personnalisées pour les articles les plus courants. Les touches numériques permettent d'accéder aux autres articles par l'intermédiaire de codes. Le libellé correspondant, mémorisé par la caisse, ainsi que le prix de l'article s'affichent sur un petit écran à diodes électroluminescentes de vingt caractères. Une prise est prévue pour l'utilisation d'un lecteur de code à barres. Les habitudes commerciales du futur proche sont donc prises en compte.

Un ordinateur polyvalent

Pour faire la gestion des stocks, l'inventaire, les contrôles, les approvisionnements et les synthèses financières, un micro-ordinateur Strator 200 peut être connecté à une ou deux caisses enregistreuses. Cet ordinateur est également fondé sur un microprocesseur 6809. Il comprend en outre deux mémoires à bulles (voir encadré) de 128 K chacune, pouvant être étendues à quatre, ainsi que 16 K de mémoire CMOS, 64 K de RAM et 32 K de PROM, quatre sorties série RS232 C, une sortie parallèle de type Centronics, un modem synchrone 1 200 bauds semi-duplex. Le FORTH,

L'ensemble SEITAMAT, présenté au Spécial Sicob 1985 par Le Comptoir des programmes (L.C.P.), comporte un terminal de vente Strator 20, le micro-ordinateur Strator 200 et un Minitel pour la commande et l'affichage. (Photo L.C.P.)



A l'arrière du Strator 200 se trouvent cinq prises : pour l'imprimante, deux caisses enregistreuses, deux Minitels, ainsi qu'une prise modem. Les prises étant standard, elles peuvent être connectées à d'autres périphériques, suivant l'application. (Photo L.C.P.)

son langage de programmation, est ultrarapide; il lui permet de tirer le meilleur parti de ses puces électroniques.

Le Strator 200 est connecté à un Minitel qui lui sert d'écran d'affichage, et permet la consultation et éventuellement des modifications à distance, offrant ainsi au magasin la possibilité de communiquer avec ses partenaires, fournisseurs, siège, équipés en informatique.

Mais ce micro-ordinateur n'est pas limité à la gestion commerciale. Il peut avoir bien d'autres fonctions. En particulier, il est capable de se transformer en serveur vidéotex. Il peut également assurer la gestion des alarmes ou des anomalies en milieu industriel, afin de prévenir les pannes éventuelles. En gestion de taxation téléphonique, il peut convenir à des hôtels de plusieurs centaines de chambres. On peut aussi l'utiliser pour le contrôle de température des chaudières, des réfrigérateurs ou congélateurs dans les grandes surfaces, ainsi que le contrôle anti-intrusion.

Multitâche, le Strator 200 peut, en effet, assurer simultanément jusqu'à huit tâches. Pour cela, des cartouches ROM peuvent être chargées sur l'avant de la machine. Par cet intermédiaire, il est possible de brancher un disque dur.

Grâce au Strator, le commerçant pourra accroître son chiffre d'affaires, ciblera mieux sa clientèle, trouvera les meilleurs partenaires, sera mieux informé, et surtout gagnera beaucoup de temps.

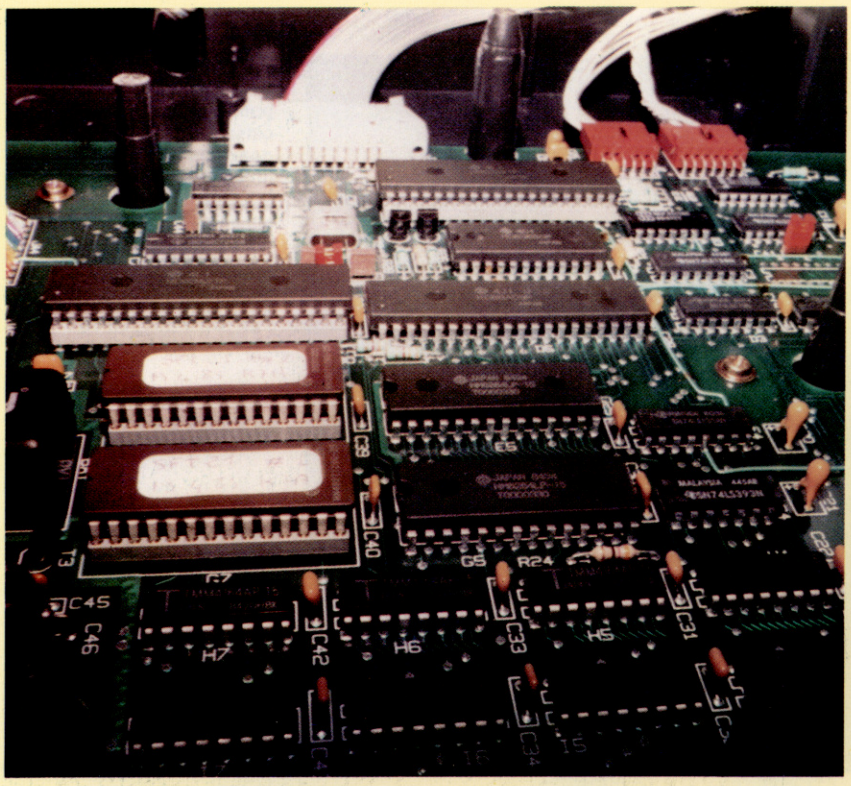
Le Comptoir des programmes assure la commercialisation du produit et fournit, en outre, la formation, la réalisation d'une programmation adaptée à l'organisation et aux spécificités envisagées, et l'étude de « réseaux » pour les liaisons à d'autres ordinateurs ou systèmes.

Les mémoires à bulles

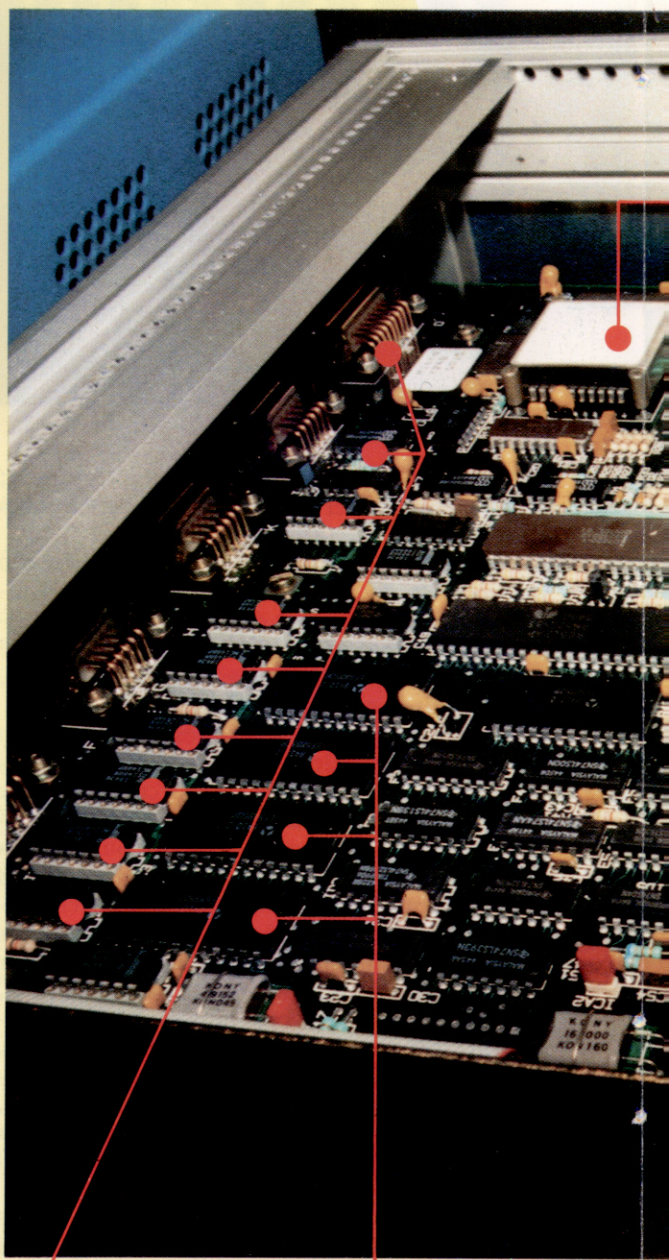
Les mémoires à bulles confèrent au Strator 200 une fiabilité et une robustesse sans égales. Mise au point en 1967 par une équipe des laboratoires Bell, aux États-Unis, la mémoire à bulles est, contrairement aux disquettes et disques durs, dépouillée de toute mécanique en mouvement. En outre, c'est une mémoire non volatile, c'est-à-dire qu'elle conserve l'information même en l'absence d'alimentation. La technologie des mémoires à bulles met à profit la propriété qu'ont certains matériaux magnétiques à s'organiser en zones d'aimantation opposée, déterminant ainsi, au sein d'un même matériau, deux états opposés. Sous l'action d'un champ dit « de polarisation », une des orientations magnétiques est favorisée, ne laissant plus que des zones cylindriques très petites, d'aimantation opposée : ce sont les « bulles ». Elles constituent l'élément d'information de ces mémoires : la présence d'une bulle correspond à la valeur 1, son absence à la valeur 0. L'écriture, la lecture et l'effacement sont effectués au moyen de champs variables, se superposant au champ de polarisation permanent, et créés par des courants. Le déplacement des bulles à l'intérieur du matériau se fait également par l'intermédiaire de champs variables, excluant ainsi toute mécanique.

Le support de la mémoire à bulles est enfermé dans un boîtier hermétique, mettant cet élément à l'abri de la poussière et de l'humidité, ce qui lui confère une robustesse sans égale.

En France, les mémoires à bulles sont fabriquées par la Société d'applications générales d'électricité et de mécanique (SAGEM).



Intérieur de la caisse enregistreuse Strator 20, comportant le microprocesseur 6809, en blanc, les mémoires PROM, les connecteurs permettant la liaison avec le Strator 200. (Photo L.C.P.)



entrées/ sorties (arrière)

contrôleurs d'entrées/sorties

Caisse enregistreuse Strator 20

CARACTÉRISTIQUES GÉNÉRALES

Clavier : 82 touches.
1 000 produits en configuration complète, 500 en « caisse seule ».
Appels directs : 44.
Serrure : 8 positions.
Caractéristiques principales : rapports horaires, journaliers, mensuels.
Jusqu'à 31 familles.
10 + 10 registres financiers.
4 modes de paiement.
5 taux de taxe.
Calcul taxes incluses ou hors-taxes.
Libellés articles sur 12 caractères.

SÉCURITÉ

Protection des opérations sur coupure de courant.
Batterie mémoire : conservation 3 mois.
Test d'autodiagnostic.
Horloge/calendrier.

PÉRIPHÉRIQUES

Écran opérateur : 10 chiffres, 10 indicateurs de fonction.
Tourelle client : 20 caractères.
Lignes d'impression : 22 colonnes.
Lignes de visualisation : 32 colonnes.

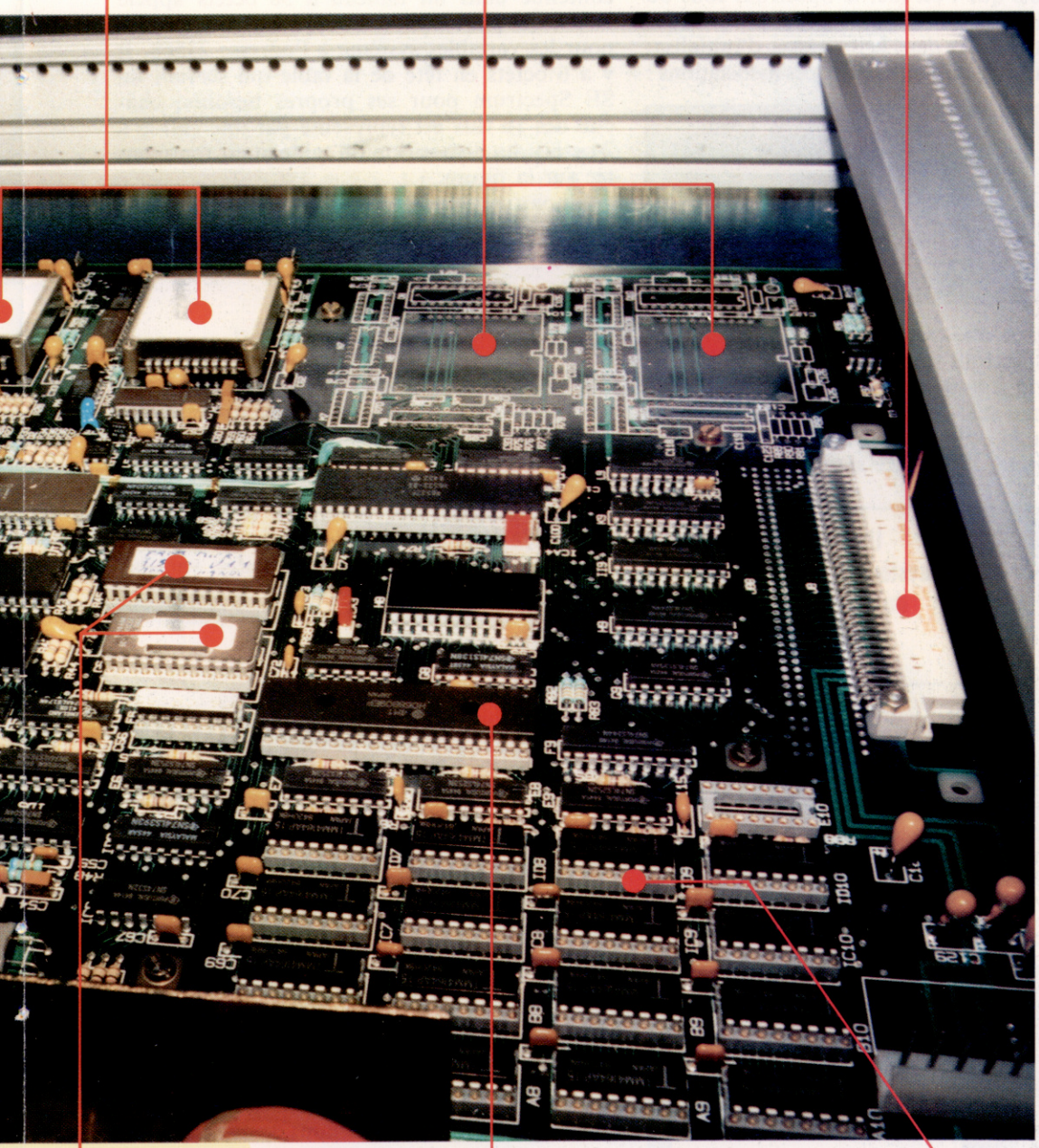
LIAISON MICRO

RS232 C.
Téléchargement automatique.
Transfert permanent des quantités vendues.
Transfert en fin de journée des cumuls financiers et produits.

2 boîtiers de mémoire à bulles

emplacement pour 2 boîtiers supplémentaires de mémoire à bulles

connecteur ROMPACK accessible par une trappe située à l'avant de l'appareil



PROM contiennent les programmes

Microprocesseur 6809

RAM

Micro-ordinateur Strator 200

DIMENSIONS

19 pouces.

MICROPROCESSEUR

6809. RAM 128 K.

MÉMOIRES A BULLES

2 × 128 K.

LIAISONS SÉRIE

4 RS232C.
1 ou 2 caisses.
1 Minitel.
1 libre.

LIAISON PARALLÈLE

1.

CONNEXION IMPRIMANTE

INTERFACE CENTRONICS SUR LIAISON PARALLÈLE

COMMUNICATION

Modem 1200 bauds semi-duplex intégré.
Réponse automatique.
Protocole IBM BSC 3780.

SYSTÈME D'EXPLOITATION

Multitâche.
Multiposte.
Langage de programmation : FORTH.

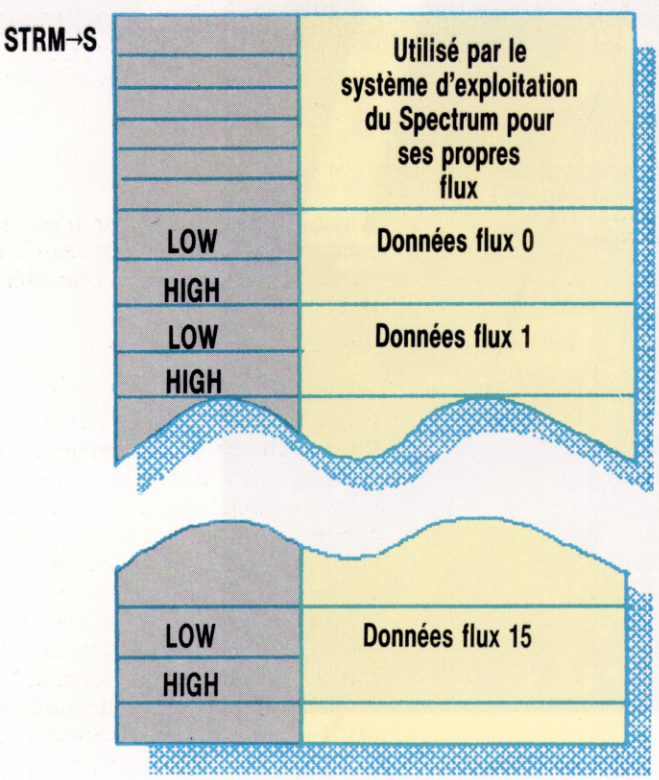
Flux en ligne

Attardons-nous sur les flux de données que le Spectrum envoie par ses canaux d'entrées/sorties vers l'imprimante et l'écran, ou reçoit à partir du clavier.

Chaque fois que l'on met le Spectrum sous ou hors tension, certains flux sont associés à des canaux particuliers dans le processus d'initialisation. Le tableau suivant montre ces associations :

Canal	Flux	Description de canal
K	0	Clavier et partie inférieure de l'écran.
K	1	
S	2	Partie principale de l'écran.
P	3	Imprimante ZX.

Cependant d'autres flux qui ne servent pas dans un Spectrum standard sont disponibles. Il faut noter que la description qui suit se réfère à un Spectrum sans l'Interface 1 ; tout est changé lorsque nous utilisons des réseaux locaux, des Microdrive ou l'interface série.



Les autres flux sont numérotés de 4 à 15, et ils peuvent aussi être associés à tout canal E/S que nous voudrions interfacer.

Le système d'exploitation du Spectrum comporte un enregistrement des flux qui sont associés à chaque canal utilisé. Cette information est contenue dans un tableau à 38 octets appelé STRMS, qui peut se trouver dans la zone réservée pour les variables système à l'adresse 23568. Il y a 6 octets en tête de la table qui servent au SE Spectrum pour ses propres besoins ; chacun des 16 flux est représenté par une entrée à 2 octets. La disposition du tableau est représentée sur la figure.

La partie de code BASIC suivante imprimera le contenu de cette zone de mémoire pour vous permettre de l'examiner :

```
10 FOR I=23568 TO 23604 STEP 2
20 PRINT PEEK I+256*PEEK(I+1)
30 NEXT I
```

Elle imprime toutes les entrées à 2 octets. Les trois premières sont les détails des flux internes du Spectrum, puis nous voyons les entrées pour les flux 0 à 15.

Les valeurs de cette dernière partie de tableau agissent comme des pointeurs à différentes entrées dans une autre table localisée ailleurs en mémoire et pointées par une autre variable système appelée CHANS. Celle-ci se trouve à l'adresse 23631, et l'adresse du tableau CHANS peut se trouver à l'aide de :

```
PRINT PEEK 23631+256*PEEK 23632
```

Cette variable système ne doit *jamais* être POKée. L'expression que nous venons d'utiliser vous donne le début du *tableau d'enregistrement de canal* (CRT), qui contient les diverses informations pour chaque canal. La valeur lue à partir du tableau STRMS est utilisée pour indexer ce CRT. Une entrée du tableau STRMS donne le décalé+1 à partir du début du tableau d'enregistrement de canal de l'information concernant le canal utilisé avec un flux particulier. Ainsi, l'entrée de tableau STRMS pour Canal 3 (vers l'imprimante) est 16. En soustrayant 1, nous obtenons un décalé de 15. Aussi les données concernant le canal P se trouveront-elles à l'adresse :

```
(PEEK 23631+256*PEEK 23632)+15
```

(en supposant que nous venions de mettre l'ordinateur sous tension et que nous n'ayons pas modifié les associations flux-canaux après initialisation).

Si les données dans une entrée de tableau STRMS sont nulles, cela indique que le flux n'a été associé à aucun canal. Sur un Spectrum non étendu, c'est le cas pour les flux 4 à 15 après initialisation. Chaque entrée dans le CRT se présente sous la forme suivante :

octet lo	Adresse de la routine de sortie pour le canal.
octet hi	
octet lo	Adresse de la routine d'entrée pour le canal.
octet hi	
n	Octet unique représentant le code canal en format ASCII, soit K, S ou P.

Une entrée de tableau STRMS pointe le second octet de chaque entrée de CRT — d'où la nécessité de soustraire 1 avant de pouvoir l'utiliser. Ce tableau est au cœur du système d'entrées/sorties du Spectrum. Si un canal n'est pas capable d'effectuer l'une ou l'autre des opérations d'E/S, alors l'entrée d'adresse appropriée dans le tableau CRT pointera une routine dans la ROM Spectrum qui génère le message d'erreur Invalid I/O Device. Donc, pour un canal capable de sortie seulement — tel que le canal S — l'entrée d'adresse de routine pointera cette routine d'erreur. Sinon, la routine ROM appropriée pour la sortie ou l'entrée sera pointée pour devenir ces entrées de tableau.

Examinons l'entrée du canal P dans le CRT. Rappelons que, normalement, ce n'est qu'un canal de sortie qui envoie des données vers l'imprimante ZX. Les données pour ce canal dans l'entrée CRT sont :

(CHANS) + 15 →	8′ F 4	Pointe GP. Affiche routine à adresse 09F4 en ROM.
	8′ 09	
	8′ C4	Pointe routine générant erreur à adresse 15C4 en ROM.
	8′ 15	
	8′ 50	Code ASCII pour « P ».

L'entrée pour le canal S est à (CHANS)+5, et la donnée canal K est à (CHANS)+0 — (CHANS) est l'adresse contenue dans la variable système CHANS.

En utilisant les commandes OPEN# et CLOSE#, nous pouvons associer les flux inusités à l'un des canaux que nous avons considérés jusqu'ici. Mais l'usage réel de ces deux commandes consiste à associer des flux à d'autres canaux dans un système étendu, tels que ceux fournis par l'Interface 1. Toutefois, nous jetterons un bref coup d'œil à l'utilisation de ces commandes sans l'Interface 1 connectée. Par exemple :

```
OPEN #4,«S»
```

associera le flux 4 au canal S. Si vous émettez cette commande, puis examinez le tableau STRMS, vous trouverez qu'une entrée a été faite dans la position qui se réfère au flux 4. Il est également autorisé d'utiliser des variables dans une telle instruction, ainsi :

```
10 LET n=4
30 LET c$=«S»
30 OPEN #n,c$
```

qui effectuera exactement la même opération que la commande OPEN. Une fois un canal ouvert de la sorte, les commandes PRINT# et LIST# vous permettront d'envoyer des données au canal via ce nouveau flux. Ainsi, vous pouvez ajouter :

```
40 PRINT #4; «Hello»
```

qui affichera à l'écran. LIST#4 listera le programme à l'écran de manière habituelle. Si un flux a été associé à un canal qui permet l'entrée aussi bien que la sortie, alors la commande INPUT# pourrait aussi être utilisée. Ces instructions étendues PRINT et INPUT ne sont réellement utiles que lorsque les canaux supplémentaires fournis par l'Interface 1 sont disponibles.

Une fois que nous avons fini avec une combinaison particulière de flux et canaux, et que nous voulons « dissoudre l'association » entre eux, nous utilisons la commande CLOSE#. Ainsi, après notre commande précédente OPEN, une commande CLOSE#4 dissocierait le flux 4 du canal S. Toute tentative pour envoyer des données à ce flux « fermé » générerait un message d'erreur.

Les commandes PRINT#, LIST#, INPUT# et INKEY$# peuvent également être utilisées avec les flux 0 à 3. Ces flux sont effectivement ouverts en permanence — comme vous le verrez ici :

```
10 CLOSE #2:REM doit fermer canal S
20 PRINT #2;«Hello»
```

Aucun message d'erreur n'est généré — le SE s'assure qu'il ouvre à nouveau le canal avant d'y passer de l'information. Considérons maintenant la commande LIST#.

```
LIST#3
```

fait normalement la même chose que LLIST — le programme est listé au flux 3 qui, comme nous l'avons vu, est généralement associé au canal P. LIST #0 et PRINT #0 sont des commandes très intéressantes, puisqu'elles affichent dans la partie inférieure de l'écran (qui est normalement en dehors des limites de l'instruction PRINT). Notez que tout ce qui est affiché sur cette partie de l'écran sera « écrasé » lorsque le SE Spectrum affichera un message d'entrée ou d'erreur. PRINT #3 est identique à LPRINT en temps normal.

Enfin, il est possible de modifier le comportement d'un canal sur le Spectrum pour lui faire faire autre chose que ce pourquoi il a été initialisé. Par exemple, le canal P est souvent modifié pour permettre d'interfacer correctement une imprimante Centronics au Spectrum. Si nous modifions l'entrée de « routine de sortie » de l'entrée du CRT approprié, alors LPRINT et LLIST enverront les données à une routine qui manie l'imprimante Centronics.

Programme d'affichage haute résolution

Le programme d'affichage haute résolution donné ici intercepte les données émises le long du canal P sur le Spectrum et les détourne vers une routine qui fournit à l'utilisateur une possibilité d'affichage haute résolution. Le listage peut être entré à l'aide d'un assembleur standard (CHAMP, DevPac ou Picturesque, par exemple). Si vous n'avez pas d'assembleur, vous devrez entrer directement le codage hex, soit en utilisant un moniteur, soit à l'aide du Chargeur Hex donné ici. Après avoir tapé le programme Chargeur Hex et l'avoir exécuté, on vous demandera de taper l'adresse de début (qui est 65068). Il vous faudra alors taper le codage hex, ce qui peut être fait seulement deux chiffres à la fois. Par exemple, la ligne de code :

2130FE

devra être tapée comme suit :

21 (ENTER)
30 (ENTER)
FE (ENTER)

Lorsque tout le code aura été entré, tapez S et le programme sera terminé. On vous demandera alors le total de contrôle « CHECKSUM » (qui est égal à 29155), et cela indiquera si vous avez fait une erreur de frappe.

La nouvelle commande fournie par le programme a le format :

LPRINT AT X,Y;«MESSAGE»

où X doit être compris entre 0 et 168, et Y entre 0 et 248.

Les coordonnées X et Y fonctionnent de la même manière que pour la commande PLOT du Spectrum.

Chargeur Hex

```
 10 CLEAR 63999
 15 POKE 23658,8
 17 DEF FN H(H$)=16*(CODE (H$)-48-7*(H$
(1)>"9"))+(CODE (H$(2))-48-7*(H$(2) "9")

 20 INPUT "Adresse de début ";a
 25 LET Z=A
 30 INPUT (a); LINE a$
 35 IF A$="S" THEN GOTO 90
 40 PRINT a;a$; POKE 23692,255
 50 FOR Z=1 TO LEN A$/2
 60 POKE a,FN H(a$(((z*2)-1) TO)): LET
a=a+1
 70 NEXT z
 80 GOTO 30
 90 INPUT "CHECKSUM:";D
100 LET C=0: FOR B=Z TO A-1: LET C=C+PE
EK B: NEXT B
110 IF C=D THEN PRINT "Le code est exact!":
 STOP
120 PRINT "Hélas ! le code est incorrect.":
 STOP
```

Listage d'assemblage

```
                ORG   65068
          PIXAD EQU   22AAH
          UDG   EQU   23675
          CHARS EQU   23606
                ;     "Make the data in CHANS point to our routine"
2A4F5C    ENABL LD    HL,(23631)
010F00          LD    BC,15
09              ADD   HL,BC
013AFE          LD    BC,DO-IT
71              LD    (HL),C
23              INC   HL
70              LD    (HL),B
C9              RET
                ;     "Preserve regesters and call new print routine"
E5        DO-IT PUSH  HL
C5              PUSH  BC
D5              PUSH  DE
F5              PUSH  AF
CD46FE          CALL  DOIT1
F1              POP   AF
D1              POP   DE
C1              POP   BC
E1              POP   HL
C9              RET
                ;     "Check to see if a holdsan AT control code"
F5        DOIT1 PUSH  AF
3A1DFF          LD    A,(ATFLG)
FE00            CP    0
200B            JR    NZ,GETXP
F1              POP   AF
FE16      ATCHQ CP    22
201D            JR    NZ,CRCHQ
3EFF            LD    A,255
321DFF          LD    (ATFLG),A
C9              RET
FEFE      GETXP CP    254
2809            JR    Z,GETYP
F1              POP   AF
3216FF          LD    (XPOSI),A
211DFF          LD    HL,ATFLG
35              DEC   (HL)
C9              RET
F1        GETYP POP   AF
3217FF          LD    (YPOSI),A
3E00            LD    A,0
321DFF          LD    (ATFLG),A
C9              RET
                ;     "Check for end of print data (signified by 13)"
FE0D      CRCHQ CP    13
2001            JR    NZ,VCHRQ
C9        SKIPC RET
                ;     "If the code in a is    between 32 & 128 or if"
                ;     "it is a UDG code then  find its data in memory"
FE20      VCHRQ CP    32
380C            JR    C,PRNT?
FE80            CP    128
381C            JR    C,FCHR
FE90      UDGCQ CP    144
3804            JR    C,PRNT?
FEA5            CP    165
3804            JR    C,FUDGC
                ;     "For all other codes a ?will be printed"
3E3F      PRNT? LD    A,63
1810            JR    FCHR
                ;     "Find the data for the  UDG or character in mem"
D690      FUDGC SUB   144
210000          LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A7B5C          LD    HL,(UDG)
19              ADD   HL,DE
180C            JR    PRNIT
210000    FCHR  LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A365C          LD    HL,(CHARS)
19              ADD   HL,DE
010700    PRNIT LD    BC,7
09              ADD   HL,BC
221BFF          LD    (CHRAD),HL
                ;     "Check to see that the X& Y positions are valid"
3A16FF          LD    A,(XPOSI)
FEF9            CP    249
D231FF          JP    NC,ERRB
3A17FF          LD    A,(YPOSI)
FEA9            CP    169
3078            JR    NC,ERR5
ED4B16FF        LD    BC,(XPOSI)
CDAA22          CALL  PIXAD
321AFF          LD    (PIXPO),A
2218FF          LD    (DFADD),HL
                ;     "Get the first pixel slice of char being printed"
0608            LD    B,8
C5        PRNLP PUSH  BC
2A1BFF          LD    HL,(CHRAD)
7E              LD    A,(HL)
2B              DEC   HL
221BFF          LD    (CHRAD),HL
6F              LD    L,A
                ;     "If the pixel slice doesn't need moving within"
                ;     "the display byte then  jump forward"
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAE6FE          JP    Z,PUTIT
                ;     "Move the slice along in the display bytes to"
                ;     "the correct pixel position"
47              LD    B,A
2600            LD    H,0
CB3D      ROTLP SRL   L
CB1C            RR    H
A7              AND   A
10F9            DJNZ  ROTLP
                ;     "Put the slice at the   right screen location"
ED5B18FF  PUTIT LD    DE,(DFADD)
1A              LD    A,(DE)
AD              XOR   L
12              LD    (DE),A
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAFAFE          JP    Z,PST
13              INC   DE
1A              LD    A,(DE)
AC              XOR   H
12              LD    (DE),A
1B              DEC   DE
2A18FF    PST   LD    HL,(DFADD)
CD1EFF          CALL  ULINE
2218FF          LD    (DFADD),HL
C1              POP   BC
10C4            DJNZ  PRNLP
                ;     "Add 8 to the X positionso that the next char"
                ;     "is printed in a clear  space"
3A16FF          LD    A,(XPOSI)
C608            ADD   8
                ;     "If the end of the line is reached go back to"
                ;     "the start BUT DO NOT   move down a line"
FEF9            CP    249
DA12FF          JP    C,STAIT
3E00            LD    A,0
3216FF    STAIT LD    (XPOSI),A
C9              RET
00        XPOSI DEFB  0
00        YPOSI DEFB  0
0000      DFADD DEFW  0
00        PIXPO DEFB  0
0000      CHRAD DEFW  0
00        ATFLG DEFB  0
                ;     "Move the address in HL one pixel up the screen"
F5        ULINE PUSH  AF
7C              LD    A,H
25              DEC   H
E607            AND   7
200A            JR    NZ,END
7D              LD    A,L
D620            SUB   32
6F              LD    L,A
3804            JR    C,END
7C              LD    A,H
C608            ADD   8
67              LD    H,A
F1        END   POP   AF
C9              RET
                ;     "Create various errors"
CF        ERRB  RST   8
                DEFB  10
CF        ERR5  RST   8
04              DEFB  4
          FINIS END
```

Précisions sonores

Nous commençons la conception et la construction d'une interface MIDI pour permettre au C64 de communiquer avec plusieurs instruments électroniques numériques.

L'interface numérique d'instrument musical (MIDI) définit un standard de communication qui permet à divers instruments de communiquer entre eux.

Puisque la communication est de nature numérique, nous pouvons introduire un micro domestique dans le système MIDI afin de commander d'autres instruments, ou afin de stocker des données dans sa mémoire — sur bande ou disquette.

Les caractéristiques techniques de l'interface permettent de la construire facilement et à faible coût sur une carte.

Après avoir construit la carte, l'ordinateur pourra être programmé afin qu'il puisse commander d'autres unités du système MIDI en écrivant le logiciel approprié.

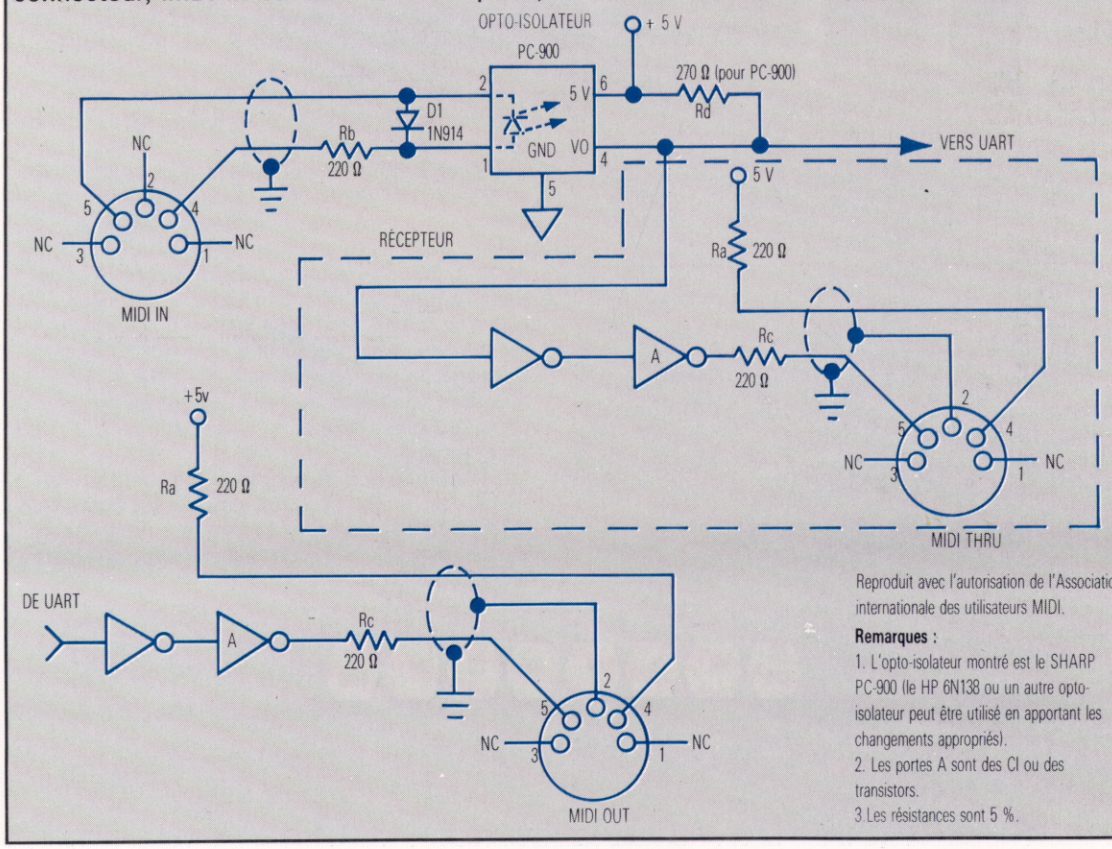
le monde musical a fait l'objet de nombreuses hypothèses. Voilà pourquoi la nature de la communication entre les divers instruments musicaux n'a pas véritablement préoccupé les musiciens qui, pour la plupart, sont (à juste titre) intéressés uniquement par les possibilités tangibles d'un système particulier au niveau musical.

Ici, nous expliquerons les normes MIDI dans des termes plus familiers aux programmeurs qu'aux musiciens; de plus, nous vous fournirons des éléments d'informations élémentaires normalement destinés aux divers concepteurs de matériel désireux de créer des produits compatibles avec ce standard.

Nous devons d'abord connaître la nature des données communiquées par MIDI. Pour faciliter la compréhension, présentons les divers com-

Matériel standard MIDI

L'interface est généralement formée de deux sections : l'une reçoit les données et l'autre les transmet. Chacune a son propre connecteur, MIDI IN et MIDI OUT. En option, il envoie simplement la copie des données apparaissant à MIDI IN. Cela sert à faciliter la connexion de récepteurs multiples enchaînés sans la présence de MIDI OUT multiples sur l'émetteur. Le diagramme du circuit de l'interface apparaît ci-dessous.



Reproduit avec l'autorisation de l'Association internationale des utilisateurs MIDI.

Remarques :

1. L'opto-isolateur montré est le SHARP PC-900 (le HP 6N138 ou un autre opto-isolateur peut être utilisé en apportant les changements appropriés).
2. Les portes A sont des CI ou des transistors.
3. Les résistances sont 5 %.

Kevin Jones

posants d'un système musical comme des périphériques d'ordinateur. Les différences entre ces composants peuvent ne pas être évidentes au départ — par exemple, un synthétiseur polyphonique moderne se présente normalement en deux parties distinctes, bien que celles-ci soient généralement contenues dans le même boîtier. Ces deux éléments sont le clavier (l'unité d'entrée) et les circuits de génération sonore (l'unité de sortie). En mode de fonctionnement normal, l'unité d'entrée est directement connectée à l'unité de sortie — appuyer sur une touche produit immédiatement une sortie sonore.

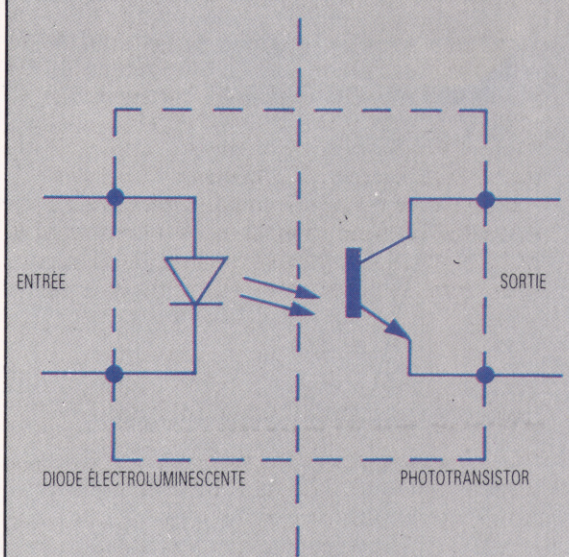
Cette situation est analogue à l'utilisation d'une machine à écrire électronique sophistiquée, où deux composants — le clavier et le mécanisme d'impression — peuvent être connectés afin de reproduire immédiatement l'entrée de données. Cependant, le clavier et l'imprimante se définissent facilement comme deux unités périphériques potentielles pour un ordinateur central qui lit les données d'entrée et les traite avant de les envoyer à une unité périphérique de sortie. L'utilisation de programmes de traitement de texte nous a tous familiarisés avec ce processus.

Dans le cas du synthétiseur, il est important de noter que l'interface (interne) entre le clavier et les générateurs sonores est purement numérique. Le clavier agit uniquement en contrôleur du synthétiseur, comme le clavier de la machine à écrire commande le mécanisme d'impression. Il est également possible d'insérer un ordinateur entre deux unités musicales, formant ainsi un « système de traitement musical ».

La musique, dans ce cas, n'est plus représentée comme un ensemble de variations de pression d'air (comme celles produites par un disque ou un disque compact sur lequel de la musique aurait été enregistrée numériquement), mais comme une séquence d'événements entrés au clavier et transmis ultérieurement à l'unité de sortie. Il revient aux circuits de génération sonore de convertir ces événements en signaux électroniques pouvant être reproduits par un système amplificateur/haut-parleur. Ainsi, MIDI introduit pour la première fois de sérieuses possibilités pour des contrôleurs autres que des claviers, tels les instruments à cordes et à vent, pourvu que les sorties respectives répondent aux normes MIDI.

Un bon exemple bien connu, pourtant rudimentaire, d'un système de traitement musical remonte à une période qui se situe bien avant l'ère informatique. Le piano mécanique pouvait jouer un morceau de musique enregistré sous la forme

Circuit opto-isolateur

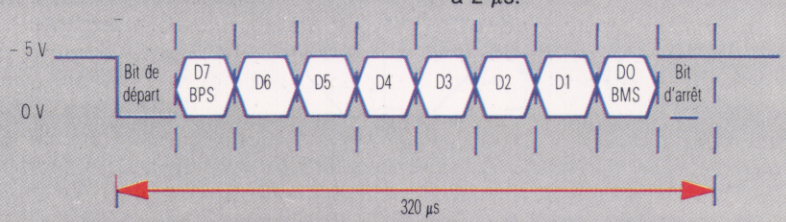


La seconde exigence matérielle de l'interface MIDI spécifie que la ligne d'entrée doit être isolée de l'équipement récepteur afin d'empêcher la création de boucles de mise à la terre dans le système. Elles doivent être évitées puisqu'elles risquent de produire des bourdonnements audibles de la fréquence secteur (50 Hz). Ce problème peut être résolu avec un dispositif nommé « opto-coupleur » ou « opto-isolateur ». Comme son nom l'indique, cette unité est un circuit intégré doté d'une connection optique (et non électrique) entre l'entrée et la sortie. Un diagramme de l'unité est présenté.

Les deux broches d'entrée sur la puce de l'opto-isolateur sont connectées à une DEL interne, similaire aux voyants indicateurs utilisés à l'avant des unités de disquettes. Une DEL émet de la lumière lorsqu'une tension est appliquée dans la direction positive. Dans le coupleur-optique, la DEL n'est pas visible à l'extérieur, mais illumine un phototransistor qui autorise le passage du courant lorsqu'il est éclairé. Le signal logique (bas ou haut) à l'entrée est transféré à la sortie sans connexion électrique.

MIDI spécifie que le coupleur optique doit nécessiter moins de 5 mA de courant pour allumer la DEL, et que les temps de montée et de descente de la sortie doivent être inférieurs à 2 µs.

Octet de données série MIDI



Kevin Jones

d'une série de trous sur un rouleau de papier. En introduisant le papier à une vitesse constante dans le piano, un mécanisme d'analyse détectait la séquence et la combinaison des trous, dont chacun représentait une note particulière. Les marteaux du piano frappaient ensuite les cordes appropriées. La musique était donc codée sur le papier comme une séquence de notes.

La distance latérale du trou représentait la hauteur de la note, et la distance longitudinale représentait la durée pendant laquelle la note devait être jouée. Il était alors possible de modifier les morceaux de musique ! Par exemple, si une mauvaise note était perforée, elle était localisée et recouverte, ensuite la bonne note était perforée sur le rouleau. Ce mode de codage des données présente de nombreuses analogies avec les premiers systèmes de stockage de données informatiques qui utilisaient des cartes perforées.

Un système de traitement musical est en quelque sorte une version améliorée d'un piano mécanique. Le papier est remplacé par une mémoire d'ordinateur et un support de stockage magnétique, et le piano est remplacé par un synthétiseur muni d'une interface d'ordinateur. L'édition est maintenant simplement effectuée en traitant les données dans la mémoire de l'ordinateur au moyen d'un programme approprié.

Avant de pouvoir le faire — même avant de pouvoir entrer les données dans l'ordinateur — nous devons connaître deux caractéristiques importantes de MIDI. La première est le type de transmission de données utilisé, et la seconde est le format des données. Ces deux facteurs correspondent aux composantes matérielle et logicielle des spécifications de l'interface; cet article n'abordera que les aspects matériels du projet.

Nous devons d'abord savoir comment est transmis un octet de données par la liaison. MIDI agit comme une interface série asynchone fonctionnant à 31,25 Kbauds. Cela signifie qu'il n'est possible de transmettre qu'un seul bit à la fois, et que le récepteur et l'émetteur ne sont pas synchronisés l'un par rapport à l'autre par des signaux d'horloge communs. Cette méthode comporte le désavantage d'une vitesse lente de transmission. Cependant, puisque la liaison ne consiste qu'en deux fils, elle ne nécessite que des câbles et connecteurs très simples et peu coûteux.

MIDI spécifie l'utilisation de connecteurs DIN à 5 broches 180° (avec en option des connecteurs audio professionnels XLR). Cela permet d'abaisser les coûts de production de l'interface, et encourage son utilisation dans des instruments peu coûteux.

Les principales critiques à l'égard de MIDI portent sur sa faible vitesse de transmission et, comme nous le verrons, sur les retards de transmission qui peuvent devenir assez importants sur un gros système. Cependant, sur le plan positif, le débit de transfert est suffisamment lent pour permettre le traitement des données par des ordinateurs domestiques.

Les données MIDI sont envoyées par groupes de 10 bits, chacun représentant un octet de données, plus un bit de départ et un bit d'arrêt. La ligne est normalement définie à une tension élevée (+ 5 V pour MIDI) lorsqu'aucune donnée n'est envoyée. Le début d'une transmission est signalé par le passage au niveau bas de la ligne (0 V) pendant la période d'un bit (le bit de départ). Les huit bits de données sont envoyés — d'abord le BPS (bit le plus significatif) —, suivis d'une période d'un bit à un niveau bas afin de signaler la fin des données. Chaque octet est donc transmis pendant une période de 10 bits, ce qui équivaut à $10/31{,}25\ K = 320\ \mu s$. Voilà un nombre que les programmeurs MIDI doivent garder à l'esprit.

Lorsque l'interface est en attente, la ligne est à un niveau de tension élevé. Le récepteur contrôle continuellement la ligne à la recherche de l'apparition d'un niveau bas — le bit de départ de la transmission. Le récepteur sait alors qu'un octet de données est envoyé et peut solliciter son compteur, qui compte une période d'un bit et demi à partir de la détection du bit de départ. Lorsque cette période s'est écoulée, le récepteur se situera à la moitié de la période de bit contenant le BPS des données et pourra alors détecter de façon fiable s'il s'agit d'un 1 ou d'un 0. Les autres bits sont reçus en comptant successivement sept périodes d'un bit et en détectant le niveau correspondant à chacune de ces périodes.

Finalement, le bit d'arrêt est analysé afin de vérifier s'il se situe à un niveau élevé, confirmant ainsi le bon « cadrage » des données par les bits de départ et d'arrêt. Grâce à cette méthode de communication, les horloges de l'émetteur et du récepteur n'ont pas à être synchronisées avec précision. Cependant, la différence ne doit pas être suffisamment importante pour entraîner des erreurs de synchronisation à la fin de l'octet; MIDI spécifie donc une tolérance de 1 % au niveu de la fréquence d'horloge.

L'interface est relativement rapide, sa nature série (RS232) fonctionne à un débit maximal de 19,2 Kbauds; la longueur des câbles de connexion ne doit donc pas dépasser 15 m pour éviter de créer des temps de montée et de descente trop lents.

Délais de transmission

Nous devons maintenant parler de la principale contrainte de MIDI, c'est-à-dire les délais de transmission entre instruments. En général, un message on/off d'une note est formé d'un total de 3 octets. L'envoi de ce message prend donc $3 \times 320\ \mu s = 0{,}96$ ms. Examinons maintenant le cas d'un séquenceur à canaux multiples où, par exemple, vingt messages de ce type doivent être envoyés simultanément. Cela implique un retard de 20 ms pour la dernière note.

Plusieurs séquenceurs employant un enregistrement en temps réel utilisent une résolution de synchronisation allant jusqu'à 3 ou 4 ms pour reproduire les nuances de synchronisation d'une exécution humaine. Par conséquent, un retard de 20 ms pourrait entraîner une synchronisation erronée, perceptible lors de l'exécution de l'enregistrement.

Pour approfondir logo

Pour aller plus loin en logo et découvrir les multiples facettes de ce langage particulièrement intéressant et formateur pour l'esprit, voici quelques livres.

L'alternative logo

Une utilisation de l'informatique à l'école élémentaire, premier cycle et éducation spécialisée, expérimentée par un groupe de l'I.R.E.M. Paris-Nord.

Le langage LOGO est un instrument privilégié pour un apprentissage dépassant de loin les seules techniques de programmation : il permet de donner à l'enfant la maîtrise de l'ordinateur et met en jeu rigueur et imagination.

Au-delà d'une initialisation à l'informatique, la multiplicité d'activités et de projets offerts dans ce livre autorise l'enseignant à exploiter toutes les possibilités d'un micro-ordinateur.

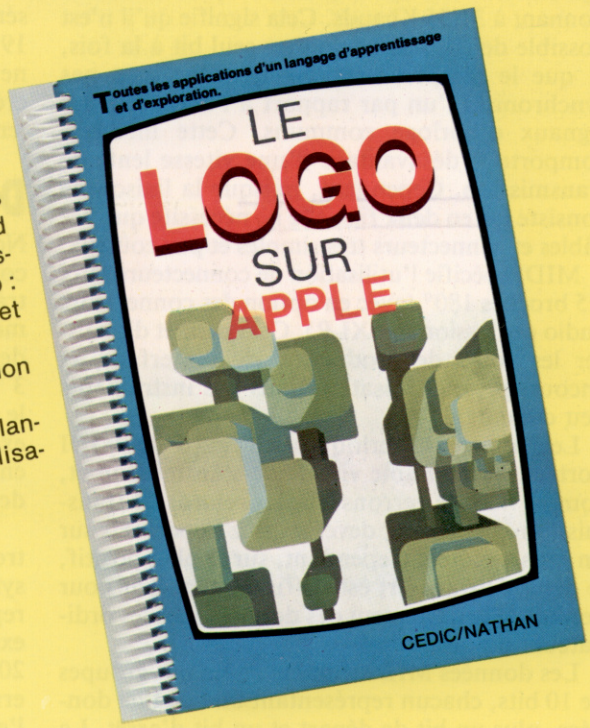
Par M. Bourbion.
190 pages. Format 13,5 × 21 cm.
Armand Colin — Bourrelier.
Pratique pédagogique.

Le logo sur Apple

Conçu par Seymour Papert et ses collaborateurs du M.I.T., LOGO est un langage inspiré des idées de Jean Piaget sur l'auto-apprentissage. Harold Abelson distingue trois étapes dans l'apprentissage du langage et de la programmation LOGO :

- les bases de la définition de procédures et la façon d'utiliser les graphiques tortue;
- l'écriture de procédures et leur conservation en fichiers sur disque;
- les aspects particuliers de la syntaxe du langage LOGO, l'emploi de la récursivité et l'utilisation de listes.

Par Harold Abelson.
270 pages. Format 15 × 23 cm.
Cedic/Nathan.



Initiation à logo

LOGO est un langage simple, immédiatement utilisable, qui se développe au fur et à mesure de votre compréhension. Ce langage puissant, aux caractéristiques sophistiquées, vous familiarisera avec les notions de récursivité, de procédure, de liste... Le graphique de la tortue est un monde riche, permettant d'apprécier d'un seul regard le résultat des programmes.

Cette initiation à la programmation LOGO vise à introduire les idées et méthodes principales de ce langage, appliqué aux micro-ordinateurs TO7, TO7-70 et MO5. Un livre à la hauteur du langage qu'il décrit.

Par Doris Avram et Michèle Weidenfeld.
160 pages. Format 15 × 23 cm.
Cedic/Nathan.

Logo, des ailes pour l'esprit

« Mon intention, en écrivant cet ouvrage, allait au-delà d'un manuel pratique pour des étudiants ou pour des personnes intéressées par le sujet; je visais plutôt à introduire une modalité particulière dans l'utilisation des ordinateurs et à contribuer à élaborer une nouvelle relation avec la pensée individuelle. »

Ainsi l'auteur présente-t-il ce livre très abordable, organisé en petites unités qui introduisent séparément des concepts fondamentaux, des groupes d'exemples illustratifs ou des applications spécifiques.

Des détails et des notes accompagnant chacune de ces unités s'adressent au lecteur désireux de les approfondir.

Par Horacio C. Reggini.
200 pages. Format 13 × 20 cm.
Cedic/Nathan.

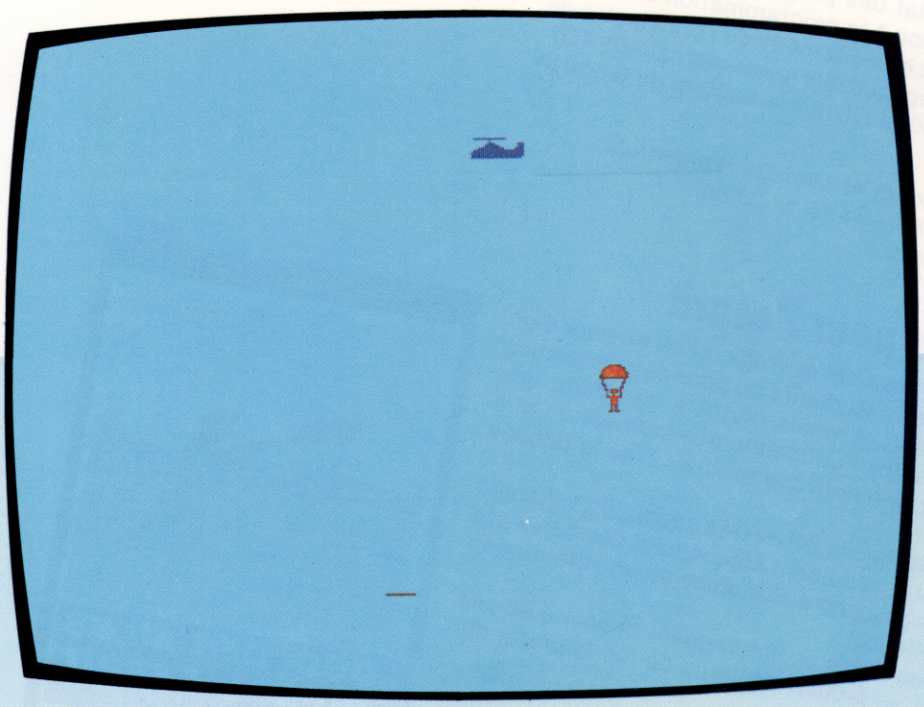
Lire logo

Un langage puissant pour traiter les listes et les mots. C'est de ce point de vue, et non de celui de la tortue, qu'est écrit cet ouvrage qui représente de manière très systématique les concepts du LOGO : traitements de listes, procédures, récursivité, calculs, et enfin les grands principes de la programmation en LOGO. Le tout est illustré de nombreux exemples mettant en évidence la richesse et la convivialité de ce langage.

Par André Myx.
110 pages. Format 15 × 23 cm.
Cedic/Nathan.

Parachute

Ce jeu d'action existe pour de nombreux modèles de micro-ordinateurs. Nous vous présentons ici le programme destiné au MO5 de Thomson.



Essayez, en sautant d'un hélicoptère en vol, d'atteindre la cible qui se trouve au sol. Une première pression sur une touche vous permet de descendre verticalement en chute libre. Une seconde pression entraîne l'ouverture du parachute. La descente continue alors plus lentement et avec un angle de 45° car le vent vous pousse. Plus vous attendrez pour ouvrir votre parachute et moins vous serez déporté. Mais n'attendez pas trop longtemps car, au-dessous de 100 m, le parachute ne s'ouvre plus...

```
10 REM *************
20 REM * PARACHUTE *
30 REM *************
40 GOSUB 470
50 HH=HH-4
60 IF HH=0 THEN PUT (0,1)-(27,9),R
70 IF HH=0 THEN HH=288
80 PUT (HH,1)-(HH+27,9),H
90 D$=INKEY$
100 IF D$="" THEN 140
110 IF PV>100 THEN 140
120 IF SP=1 THEN OP=1 ELSE SP=1
130 IF OP=0 THEN PV=10:PH=HH
140 IF SP=0 THEN 230
150 IF OP=0 THEN PV=PV+8
160 IF OP=1 THEN PV=PV+1:PH=PH-1
170 IF PV>167 OR PH<1 THEN 260
180 IF OP=1 THEN 210
190 PUT (PH,PV)-(PH+14,PV+23),PF
200 GOTO 50
210 PUT (PH,PV)-(PH+14,PV+23),PO
220 GOTO 50
230 FOR I=1 TO 50
240 NEXT I
250 GOTO 50
260 IF ABS(PH-A)>4 THEN 320
270 FOR I=1 TO 1000
280 NEXT I
290 S=S+10
300 GOSUB 670
310 GOTO 50
320 CLS
330 SCREEN 1,2,2
340 LOCATE 10,10
350 ATTRB 1,1
360 PRINT "SCORE :";S;
370 LOCATE 10,16
380 PRINT "UNE AUTRE ?";
390 ATTRB 0,0
400 IF INKEY$<>"" THEN 400
410 D$=INKEY$
420 IF D$="" THEN 410
430 IF D$<>"N" THEN RUN
440 CLS
450 SCREEN 4,6,6
460 END
470 LOCATE 0,0,0
480 SCREEN 4,6,6
490 DEFINT A-Z
500 DIM H(27,8)
510 DIM PF(14,23)
520 DIM PO(14,23)
530 DIM R(27,8)
540 CLS
550 DRAW "C4BM51,50R14L6G4L2G1L1G1D1F1R2
1E1U5L2D1G2L5H1L1H1L1H1L1"
560 PAINT (58,56),4
570 GET (50,50)-(77,58),H
580 GET (100,50)-(127,58),R
590 CLS
600 DRAW "C1BM54,50G1L1G1D1G1D1R12U1H1U1
H1L1H1L4"
610 PAINT (56,54),1
620 DRAW "C1BM51,56D1F1D1F1D3R2D6L1R1U3R
2D3R1L1U4L1U4L1U1R2D1L1D4R1U2R2U3E1U1E1U
1"
630 GET (50,46)-(64,69),PO
640 CLS
650 DRAW "C1BM54,61R1D6L2D2U2R3D2L1D1R2U
1L1U2R3D2U2L3U3D3R1U6R1"
660 GET (50,46)-(64,69),PF
670 CLS
680 HH=288
690 HV=1
700 A=INT(RND*191)+10
710 DRAW "COBM"+STR$(A)+",191R12"
720 SP=0
730 OP=0
740 PV=0
750 PH=0
760 RETURN
```