NORBERT RIMOUX

GUIDE DE L'UTILISATEUR

ALICE ET ALICE 90

SYBEX

Paris • Berkeley • Düsseldorf • Londres

Photo de couverture : Dominique Weachter

ISBN : 2-7361-0078-8

Le domaine de la micro-informatique familiale est aujourd'hui en pleine expansion. L'accroissement du nombre de machines de cette catégorie promet d'être spectaculaire dans les trois années à venir.

Dans un tel contexte, les machines à bas prix comme Alice représentent un marché non négligeable.

En effet, il est préférable de s'initier à la programmation sur un micro-ordinateur bon marché plutôt que de faire l'acquisition d'un système plus évolué et surtout plus spécialisé, qui risque fort de terminer sa carrière en tant que simple console de jeu.

Le but de cet ouvrage n'est pas de faire découvrir au lecteur un langage aussi courant que le BASIC, la documentation du constructeur remplit fort bien cette mission, mais plutôt d'exposer les techniques générales de programmation adaptées à Alice ou à son jumeau, le MC 10 de TANDY.

La démarche adoptée s'inspire plus de la pratique que du cours magistral, tout en développant certaines notions générales d'informatiques indispensables à une bonne compréhension des processus de fonctionnement d'un ordinateur. Bien évidemment, avant de s'attaquer directement aux astuces de la programmation, un tel apprentissage passe d'abord par la maîtrise d'un langage évolué, en l'occurrence le BASIC. Mais bien peu de projets ambitieux sont réalisables sans aucune connaissance des arcanes du fonctionnement interne d'un appareil spécifique. Pour cette raison, ce livre contient une introduction au fonctionnement du micro-processeur 6803 ainsi qu'à sa programmation. Quoique relativement déroutante pour le débutant, cette section est sans conteste la plus riche d'enseignements quant à la manière de tirer le meilleur profit d'un système informatique.

1
GÉNÉRALITÉS

QU'EST-CE QU'UN ORDINATEUR

Un ordinateur est un appareil électronique sophistiqué dont la tâche est de traiter l'information, au sens large du terme. Cela signifie, en clair, qu'il est théoriquement capable de tout faire. Mais nous n'en sommes pas encore à ce niveau. Pratiquement, Alice peut traiter des informations numériques, des textes, voire de l'image. Dans certains cas particuliers, et moyennant un équipement additionnel, ce micro-ordinateur est capable de piloter de l'appareillage externe, cafetière, alarme, thermostat d'ambiance, etc.

Ces possibilités restent toutefois réservées aux bricoleurs de l'électronique, à moins qu'un constructeur ne commercialise un dispositif particulier "prêt à l'emploi".

Pour beaucoup, l'ordinateur est devenu un mythe souvent un peu angoissant d'autant que notre vie est en passe de devenir tributaire de l'informatique. Mais un ordinateur n'est qu'un outil, certes performant, mais parfaitement obéissant comme vous serez en mesure de vous en rendre compte tout au long de ce livre. Il est particulièrement pratique pour les tâches monotones et répétitives où sa rapidité de traitement et sa "patience" infinie en font un auxiliaire indispensable. Mais sa vocation n'est pas toujours axée sur des applications professionnelles, surtout pour un modèle familial qui passe près de soixante pour cent de son temps à des applications purement ludiques.

STRUCTURE INTERNE

Les capacités de traitement d'Alice sont dues à un seul petit *composant* électronique : le microprocesseur. Il est en fait le véritable "cerveau" de la machine, le décisionnaire en quelque sorte.

Malheureusement pour lui, il n'est capable d'effectuer qu'un nombre très restreint de tâches élémentaires... seulement si on lui en donne l'ordre. Seul, il est complètement inexploitable car il n'a absolument aucune *mémoire* ; de ce fait, il ne lui est même pas possible de se souvenir d'un ordre reçu. Il est donc évident qu'un ordinateur se doit d'en posséder une. Dans ce cas précis ce sont d'autres composants qui remplissent cette fonction. Il en est des ordinateurs, comme des humains ;

plusieurs types de mémoires sont nécessaires. Imaginez un instant que vous soyez obligé de vous rappeler toutes les étapes de la digestion puis de les effectuer ; il ne vous resterait alors que bien peu de temps pour vivre. Il existe donc bien une mémoire inconsciente qui dicte à notre organisme la bonne manière de fonctionner. Dans un micro-ordinateur, cette mémoire est celle qui contient toutes les instructions de démarrage lors de la mise sous tension. Les boîtiers qui tiennent ce rôle sont les ROM ou MEM (*mémoires mortes*). Les ROM sont normalement inaltérables, le processeur ne peut qu'y lire des instructions, mais en aucun cas, il n'en modifie le contenu.

Parallèlement, le processeur est doté d'une mémoire beaucoup plus souple qui constitue son aire de travail. La RAM ou MEV (*mémoire vive*) est analogue à la feuille de papier sur laquelle nous jetons les résultats de nos raisonnements les plus profonds. Le processeur peut y écrire, la lire, l'effacer et y effectuer d'autres opérations aussi indispensables.

La technologie employée pour la fabrication des composants de RAM actuels présente un petit inconvénient. La mémorisation de l'information provient de la charge de plusieurs dizaines de milliers de minuscules condensateurs intégrés sur une pastille de silicium de quelques millimètres carrés. Si le courant d'alimentation est coupé, les condensateurs se déchargent complètement et l'information qu'ils représentaient est perdue. Dès que l'on éteint un ordinateur, toutes les données ou programmes introduits y sont irrémédiablement perdus.

C'est à ce moment que l'on prend conscience de l'utilité d'un moyen de stockage permanent. La mémoire dite *de masse* assure cette tâche, au même titre que les livres sans lesquels notre savoir serait des plus restreints.

La mémoire de masse n'est pas un dispositif réellement actif. Le processeur doit d'abord charger son contenu en mémoire vive avant de pouvoir l'exploiter.

Les problèmes techniques que posent les mémoires de masse sont particulièrement ardus et sont fonction de la méthode employée. A l'heure actuelle, trois supports se partagent la faveur des constructeurs :

- la mini-cassette standard ;
- le disque souple ;
- le disque dur.

L'éventail de leurs performances est aussi large que celui de leur prix.

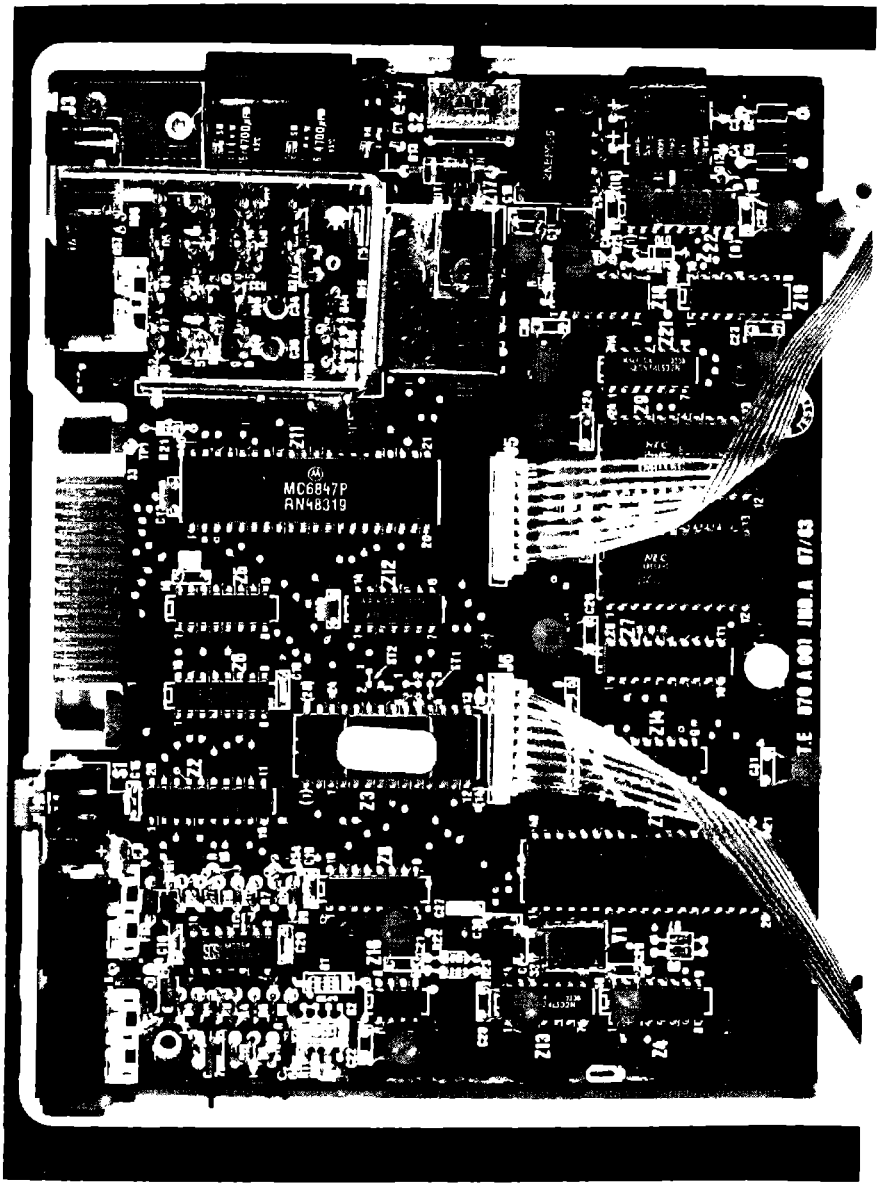


Figure 1.1. Photo de l'intérieur de l'Alice. On aperçoit distinctement les différents composants.

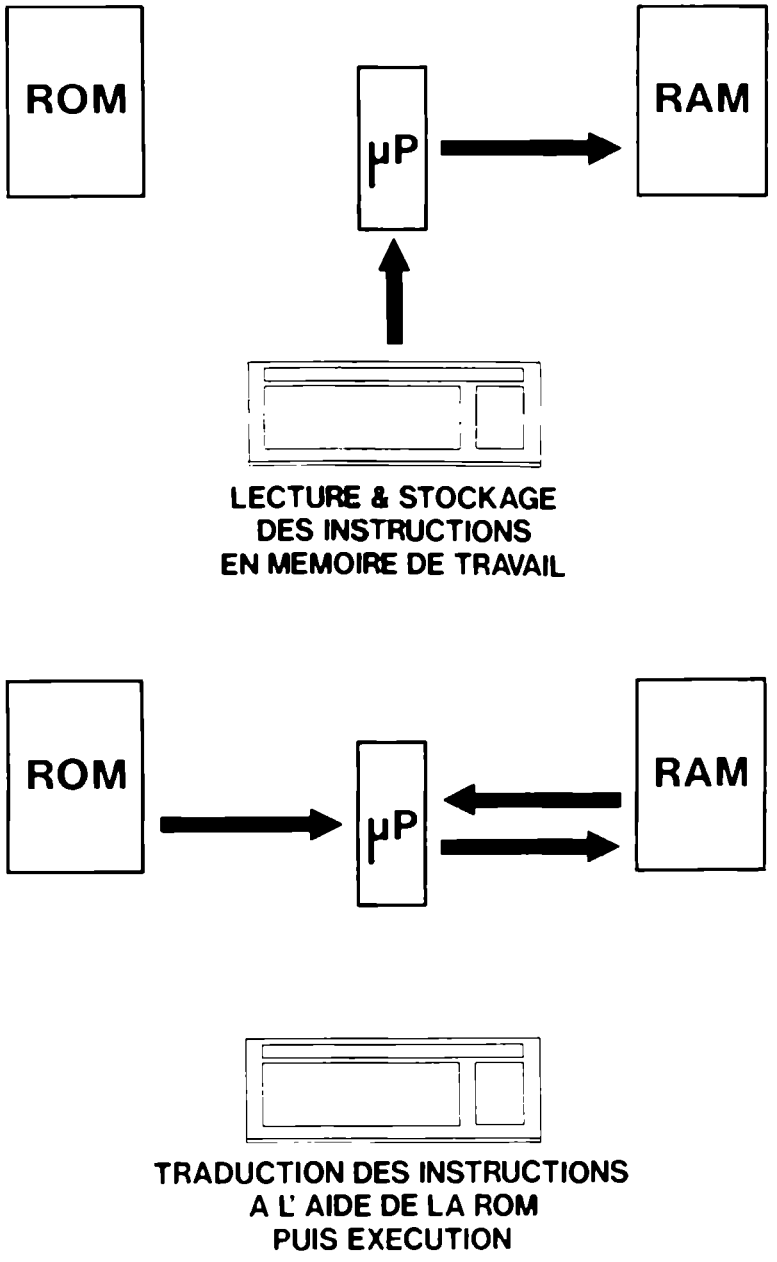


Figure 1.2. Schéma du fonctionnement interne.

Nous nous bornerons à examiner la façon dont procède Alice pour sauvegarder des informations sur une bande magnétique standard, puisqu'aucun autre moyen n'existe pour cette machine.

Il manque encore au microprocesseur les capacités de communication avec l'utilisateur ; ses sens en quelque sorte.

La liaison avec l'utilisateur se fait par le biais des deux appareils *périphériques* de l'ordinateur : l'écran de visualisation et le clavier. Dans ce domaine également les disparités sont spectaculaires.

Heureusement, les ordinateurs à vocation familiale sont tous bâtis sur un modèle quasiment commun.

L'écran de visualisation, qui gère le flot de données du processeur vers l'utilisateur, est généralement un téléviseur couleur classique. Cette solution est sans aucun doute la plus valable puisque bon nombre d'utilisateurs potentiels disposent déjà d'un tel périphérique.

Dans le cas d'Alice, le clavier et l'ordinateur sont intégrés. Bien qu'il ressemble par bien des côtés à celui d'une machine à écrire, il possède certaines touches spécifiques susceptibles de varier d'un modèle à l'autre. Nous verrons plus loin le détail de celui de l'Alice.

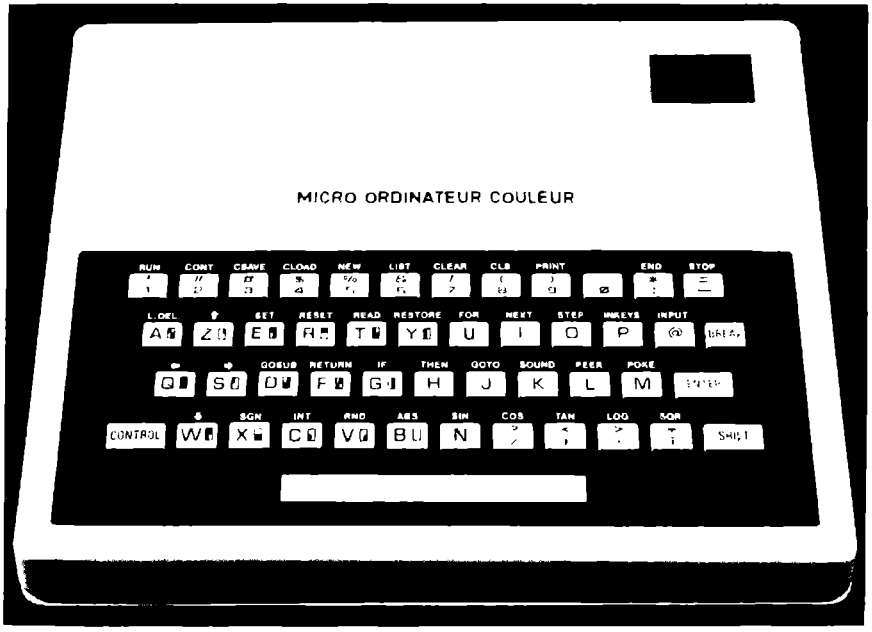


Figure 1.3. Le clavier de l'Alice. On remarque les touches spéciales ENTER, CONTROL...

Figure 1.4. Imprimante thermique.

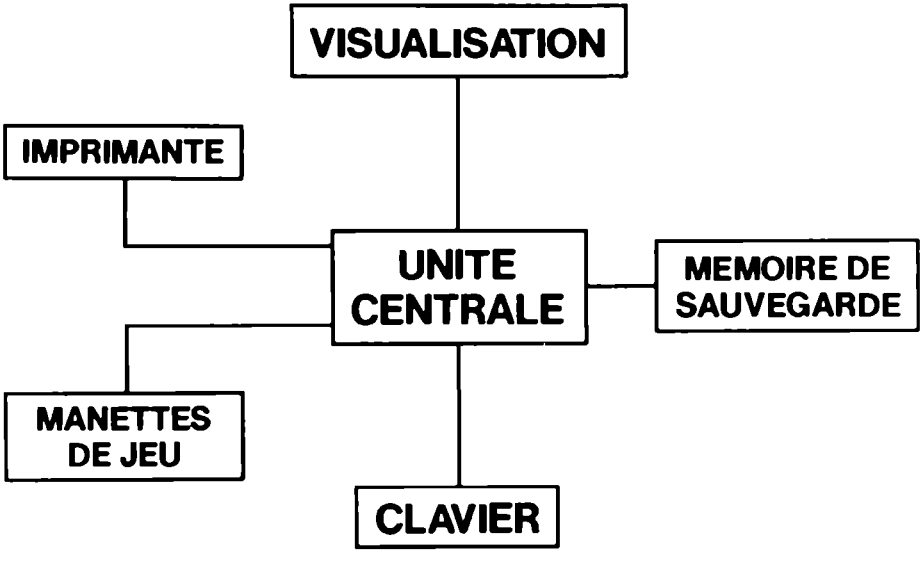


Figure 1.5. Synoptique de configuration.

L'unité centrale (UC) accompagnée de son écran et de son magnétophone à cassette est prête à fonctionner dans de bonnes conditions.

Mais il n'est pas rare de rencontrer des systèmes informatiques munis de périphériques supplémentaires. Ainsi, Alice peut être connecté à une petite imprimante thermique très utile pour l'archivage des données sur papier.

Une configuration classique répond à la synoptique suivante, et Alice n'y fait pas exception.

LA COMMUNICATION AVEC L'ORDINATEUR

Le microprocesseur est un composant programmable, c'est-à-dire qu'il est possible de lui fournir une liste d'instructions qu'il est capable de "comprendre". Cette liste représente un *programme* dans le langage de la machine.

Malheureusement pour nous, la CPU (*Central Processing Unit*) ne reconnaît que des quantités numériques. Ainsi, chaque instruction de base est-elle codée par un nombre. Un programme en *langage machine* n'est donc qu'une suite de chiffres peu représentatifs de leurs effets. C'est pour cette raison que les informaticiens ont conçu des *langages évolués.*

Quoique beaucoup plus compréhensible pour le programmeur, un langage de programmation n'est qu'un pâle substitut au langage naturel (généralement l'anglais). Notre manière de parler comporte bien trop d'ambiguïtés et de "sous-entendus" pour être facilement analysée par un programme d'ordinateur. A l'heure actuelle, il existe plus d'une centaine de langages évolués "tournant" sur des ordinateurs. Mais le plus simple à apprendre est sans nul doute le BASIC. Ce n'est pas sans raisons que l'immense majorité des constructeurs propose ce langage intégré en version de base sur leur machine. Avant d'approfondir ce langage en particulier, voyons d'abord les caractéristiques d'un langage.

Schématiquement, il est possible de dire que chaque langage possède un domaine dans lequel il excelle. Mais il n'en existe aucun qui soit universel, loin s'en faut. Seul la bonne connaissance de la programmation permet de se faire une bonne idée de ses besoins en matière de langage. Mais plus un langage est performant, plus son mode d'em-

ploi est ésotérique. Pour vous en convaincre, il suffit de regarder comment est fait un programme APL,FORTH ou, dans une moindre mesure, un programme en Pascal.

Toujours est-il que la connaissance des méthodes de programmation est bien plus importante que celle de la syntaxe d'un langage donné.

C'est pourquoi le BASIC est très bien adapté à l'apprentissage de la programmation pourvu que l'on se souvienne qu'il n'en est que l'un des supports possibles.

PRÉSENTATION DU BASIC

Le BASIC est le langage résident dans la ROM de l'Alice. C'est donc lui qui nous servira de référence tout au long de ce livre.

Il est constitué d'un certain nombre de mots clés exécutant des tâches précises.

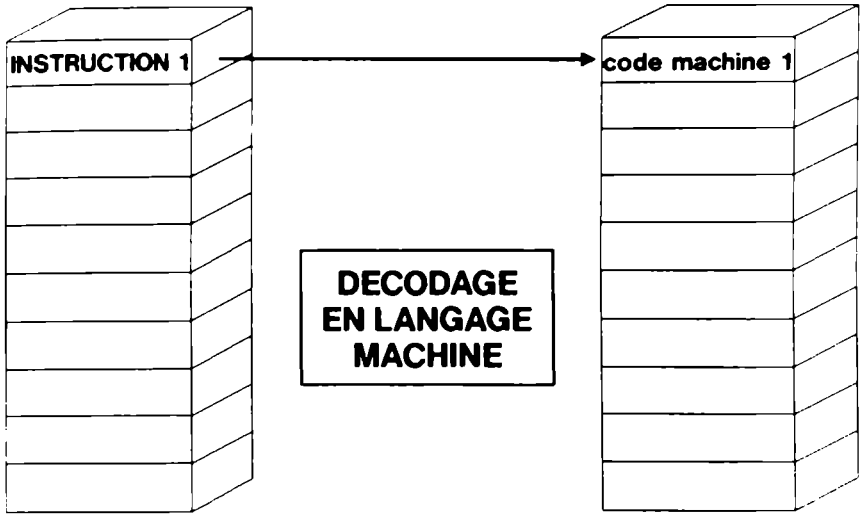
Ces instructions, proches de l'anglais, décrivent aussi fidèlement que possible leurs effets.

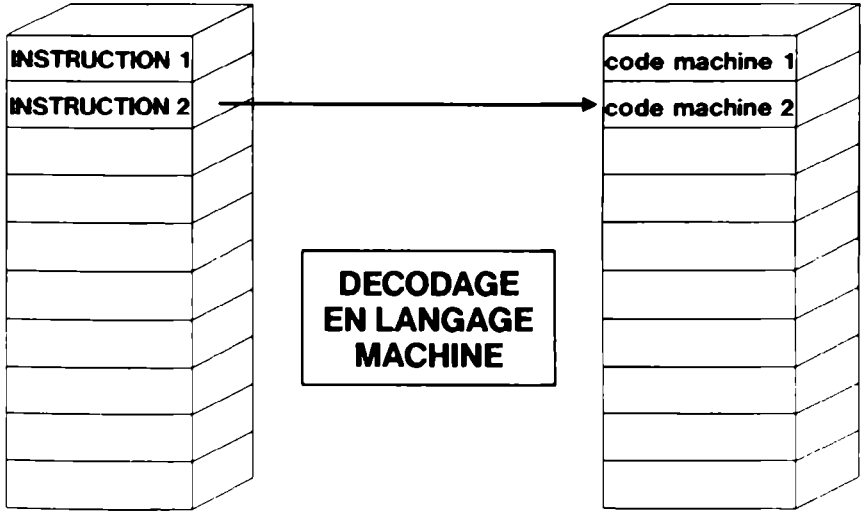
Par exemple, PRINT signifie imprimer dans la langue de Shakespeare et l'instruction BASIC PRINT "BONJOUR" affichera BONJOUR sur l'écran de visualisation.

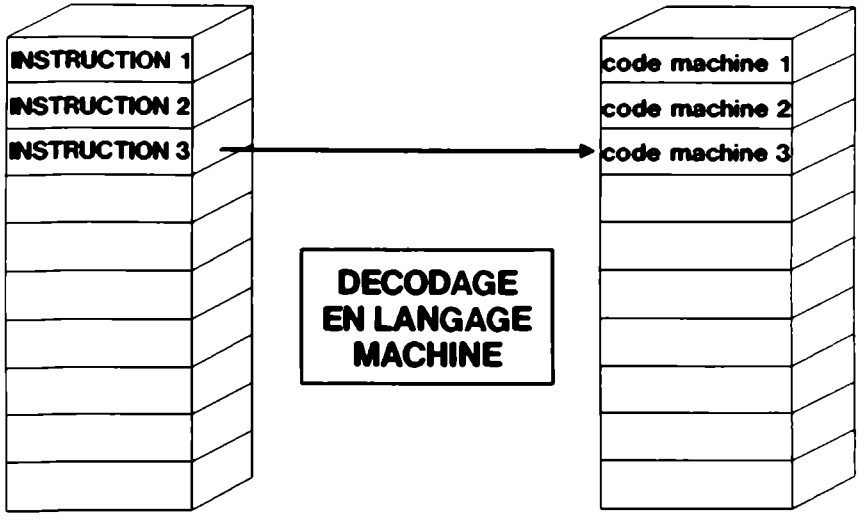
Le BASIC a un grand nombre de détracteurs.

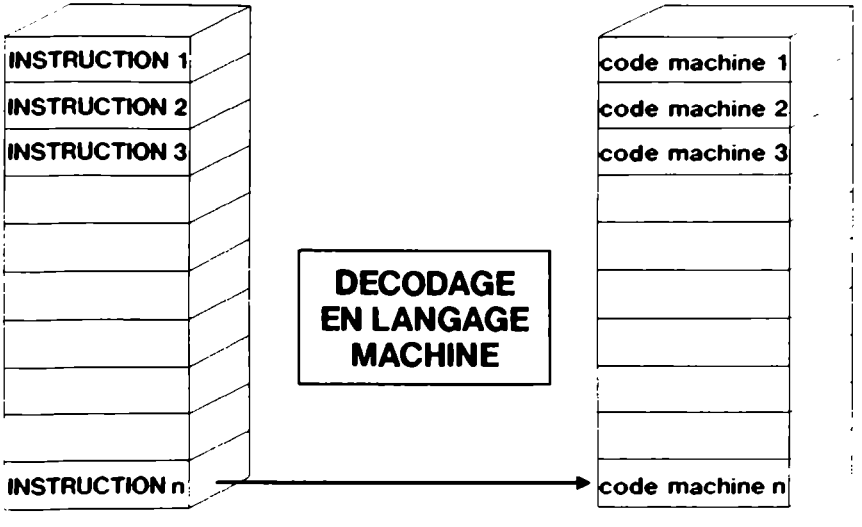
Le reproche le plus courant qui lui est fait est sa lenteur. Ceci provient du fait qu'il est presque toujours *interprété*, c'est-à-dire que chaque instruction est préalablement décodée avant d'être exécutée ; au contraire des langages dits *compilés*, qui traduisent d'abord globalement l'ensemble des instructions du programme avant d'exécuter le tout. Il y a donc un gain de temps considérable lors de l'exécution de celui-ci.

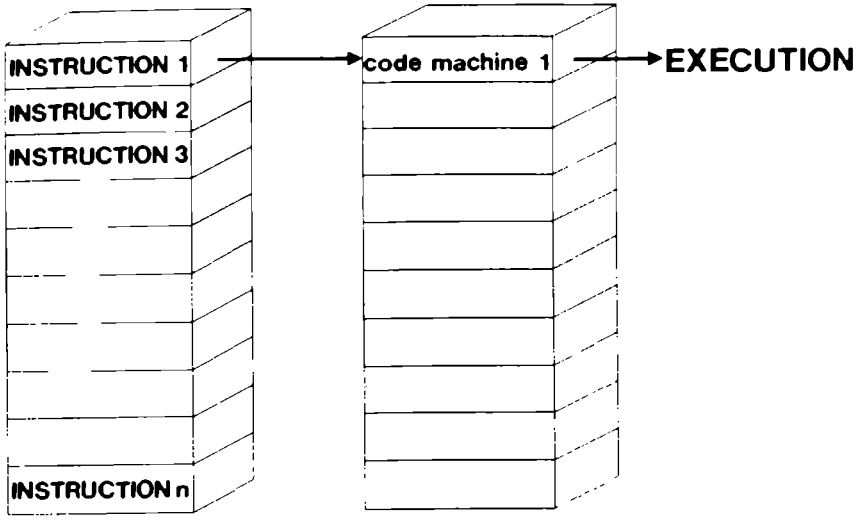
Mais un compilateur nécessite beaucoup plus de place en mémoire qu'un interpréteur, ce qui est inacceptable sur une machine comme Alice relativement limitée dans ce domaine.

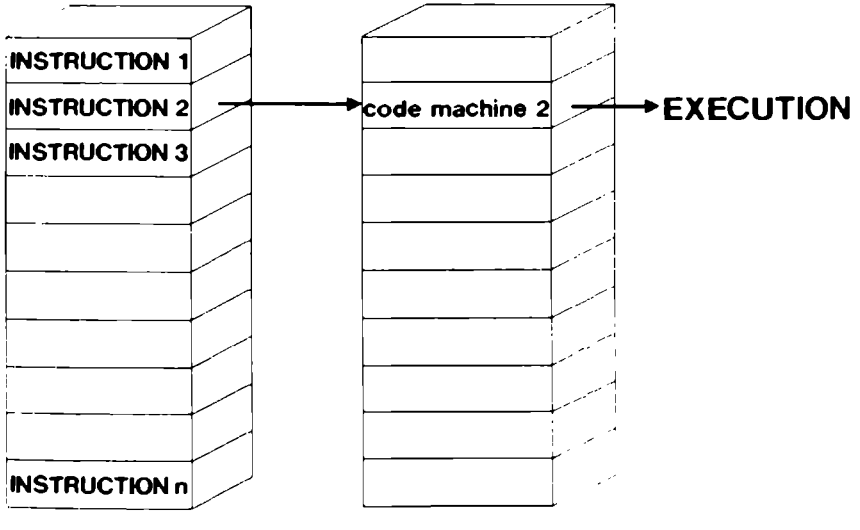

INSTRUCTION 1
code machine 1
DECODAGE
EN LANGAGE
MACHINE

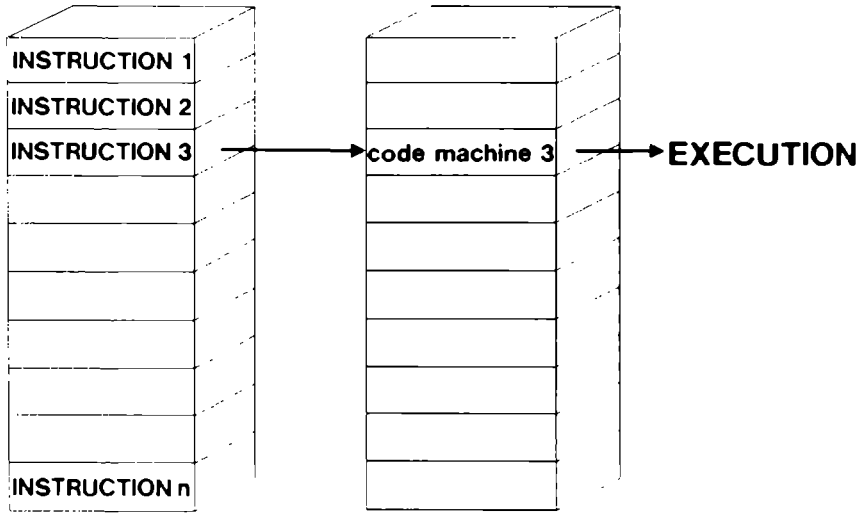

INSTRUCTION 1
INSTRUCTION 2
code machine 1
code machine 2
DECODAGE
EN LANGAGE
MACHINE


INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
DECODAGE
EN LANGAGE
MACHINE
code machine 1
code machine 2
code machine 3


INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
INSTRUCTION n
DECODAGE
EN LANGAGE
MACHINE
code machine 1
code machine 2
code machine 3
code machine n


INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
INSTRUCTION n
code machine 1
EXECUTION


INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
INSTRUCTION n
code machine 2
EXECUTION



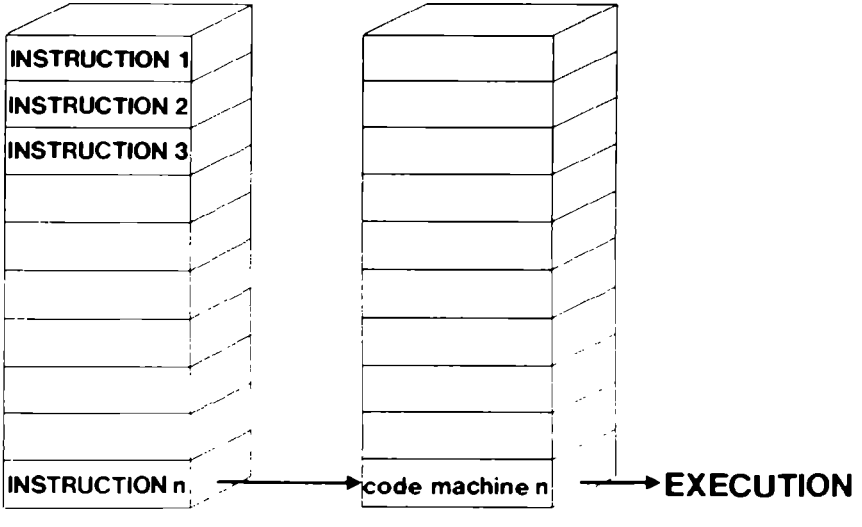


Figure 1.6. Schéma comparatif interpréteur-compilateur.

Le BASIC n'est capable de travailler que sur deux types de données différents :

- les données numériques ;
- les données alphanumériques.

Les données numériques sont interprétées selon leur vrai sens, c'est-à-dire comme autant de nombres susceptibles d'être traités arithmétiquement. Dans le second cas, l'interpréteur BASIC ne cherche en aucune manière leur signification.

Ces données sont considérées comme des symboles sans aucune signification.

Les alphanumériques peuvent être n'importe lequel des caractères ou groupe de caractères obtenu à partir du clavier.

Pour achever cette description générale du BASIC, rappelons que ce type de programmation impose certaines règles très rigides.

- Les instructions sont regroupées par *lignes* de programme.
- Chaque ligne est numérotée.
- L'exécution se fait dans l'ordre croissant des numéros de lignes, sauf spécification contraire du programmeur.
- Si aucun numéro de ligne n'est spécifié, la commande est immédiatement exécutée et n'est pas mémorisée.

2

FONCTIONNEMENT DU BASIC

FONCTIONNEMENT DE L'INTERPRÉTEUR BASIC

L'interpréteur BASIC est un programme en langage machine qui assure le décodage des ordres du programmeur et les exécute.

Il peut fonctionner selon deux modes :

- le mode immédiat ;
- le mode différé.

Dans le premier cas, l'instruction ou la suite d'instructions est stockée à la fois sur l'écran et dans une zone spécialisée de la mémoire dénommée *buffer*. Le processus est ainsi répété jusqu'au moment où la frappe de la touche ENTER est détectée. A ce moment l'ordinateur va analyser le contenu du *buffer* puis exécuter les instructions qu'il contient. En fin d'exécution, celui-ci est réinitialisé et les instructions entrées par l'opérateur sont perdues. Il faut signaler, et cela reste valable quelle que soit la machine, que la taille du *buffer* ne dépasse jamais 255 caractères (127 pour Alice). Le nombre d'instructions exécutables "simultanément" en mode immédiat est donc très limité.

Le mode immédiat est assez peu usité. C'est en fait le mode différé ou mode programme qui fait tout l'intérêt d'un ordinateur.

Lorsqu'une ligne d'instructions est précédée par un nombre, l'ordinateur réagit différemment. Lorsque l'entrée d'une ligne a été validée par la touche ENTER, celle-ci est simplement recopiée dans la mémoire vive RAM d'Alice. Puis l'interpréteur va à nouveau scruter le clavier sans exécuter la ligne de programme.

RUN est la commande qui ordonne à l'ordinateur d'exécuter les instructions stockées. Après l'exécution complète, le programme est intact et peut être réutilisé autant de fois que nécessaire.

L'interpréteur va tout d'abord chercher à reconnaître les "vraies" quantités numériques puis les instructions BASIC, et enfin les quantités variables. Si à un moment donné une erreur est détectée, il stoppe l'exécution et affiche sur l'écran un message décrivant le type d'erreur et l'endroit où elle est localisée (voir Annexe 2).

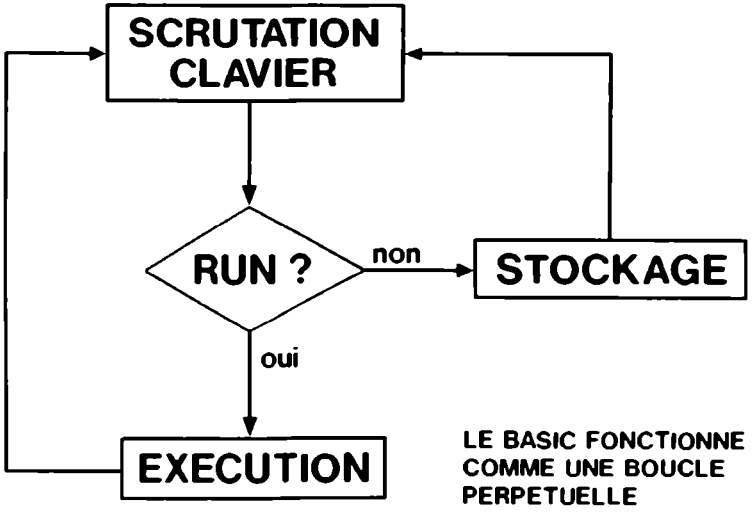


Figure 2.1. Diagramme de fonctionnement du BASIC.

ÉDITION D'UN PROGRAMME

Il est nécessaire de respecter un certain nombre de conventions sous peine d'obtenir un message d'erreur. Ainsi, les numéros de lignes doivent impérativement être compris entre 0 et 63999. Les raisons de cette limitation apparaîtront plus clairement lorsque nous aborderons les chapitres sur le langage machine.

A titre d'exemple entrez les lignes suivantes :

```
 -1 PRINT ENTER
 10 PRINT ENTER
64000 PRINT ENTER
```

et observons la réaction de l'ordinateur.

Avant "d'attaquer" un programme il est nécessaire de connaître un certain nombre d'instructions et de "ruses". Mis à part l'exécution, il est indispensable de visualiser le programme résident dans la mémoire vive. C'est le rôle du mot clé LIST qui est susceptible d'être utilisé de plusieurs manières ; on dit aussi qu'il dispose de plusieurs *syntaxes* d'utilisation.

• *LIST* utilisé seul fait défiler sur l'écran l'ensemble du programme résident en mémoire RAM.

• *LIST suivi d'un numéro de ligne* ne visualisera que la ligne concernée si elle existe.
• *LIST No - No* imprimera toutes les lignes dont les numéros sont compris entre les deux nombres mentionnés, limites comprises.

Entrez le programme ci-dessous et essayez plusieurs variantes.

```
10 PRINT
40 PRINT
50 PRINT


LIST

30 PRINT
40 PRINT
50 PRINT


LIST 40

40 PRINT


LIST 40-100

40 PRINT
50 PRINT
```

Si on désire supprimer une ligne, il suffit de taper son numéro et de le valider. Le processus est exactement le même en cas de modification, la nouvelle ligne se substituera à l'ancienne. A ce sujet, il est interdit d'employer des numéros de ligne décimaux. Dans ce cas, le BASIC ne signalera pas l'erreur mais risque de détruire une ligne existante.

```
10.5 PRINT ENTER
     LIST ENTER
10.5 PRINT
```

Seule la partie entière a été interprétée comme un numéro de ligne.
Ainsi, s'il existait une ligne 10 celle-ci a été détruite par une ligne erronée de surcroît.

Il faut savoir que le BASIC stocke et exécute les lignes de programme selon leur numérotation, du plus petit au plus grand nombre.

```
10 PRINT ENTER
30 PRINT ENTER
25 PRINT ENTER
 5 PRINT ENTER
   LIST ENTER
 5 PRINT
10 PRINT
25 PRINT
30 PRINT
```

Pour cette raison, une bonne habitude à prendre consiste à rédiger ses programmes en utilisant une numérotation de ligne de 10 en 10. Ainsi, il est facile d'insérer de nouvelles instructions si besoin est, sans avoir à réajuster le programme déjà présent en mémoire.

Notons que l'instruction NEW efface complètement les lignes de programme présentes en mémoire. Il est donc fortement conseillé de l'utiliser avant de commencer un nouveau programme pour éviter les interactions avec des instructions résiduelles. Mais son emploi en mode différé est à proscrire impérativement, car un programme ainsi conçu s'autodétruirait sans possibilité de le récupérer.

GÉOGRAPHIE DE L'ÉCRAN

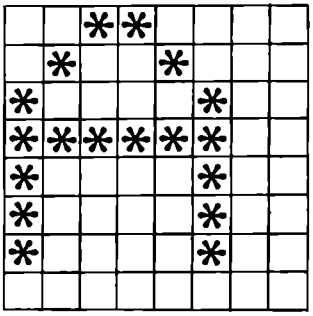
L'écran de visualisation, dans le cas d'Alice, est un simple téléviseur couleur équipé d'une prise *péritélévision*. En réalité, tout le processus de visualisation est confié à un circuit spécialisé dit CRTC (*Cathod Ray Tube Controler*). Nous n'entrerons certes pas dans le détail de son fonctionnement qui importe peu au niveau de la programmation.

Mais plus importante est la manière dont l'unité centrale gère l'affichage.

Une portion de mémoire est réservée pour l'image vidéo. Cette RAM vidéo est en fait une représentation numérique de l'écran tel que nous le voyons sur le téléviseur. Chaque caractère possède un numéro de code.

Lorsque l'un d'entre eux est destiné à être affiché, son code est introduit dans la mémoire vidéo. A ce moment, une autre partie de la mémoire est lue, qui contient l'image du caractère sous forme de points, puis le CRTC s'occupe de transformer ces informations en un signal compatible avec le téléviseur.

Par exemple, le code du "A" est 65 (voir annexe 1), la 65^{e} forme dans la mémoire des caractères sera :



Chaque caractère occupe une case de la mémoire encore appelée *octet*.

La mémoire vidéo d'Alice peut en contenir 512 répartis en 16 lignes de 32 caractères.

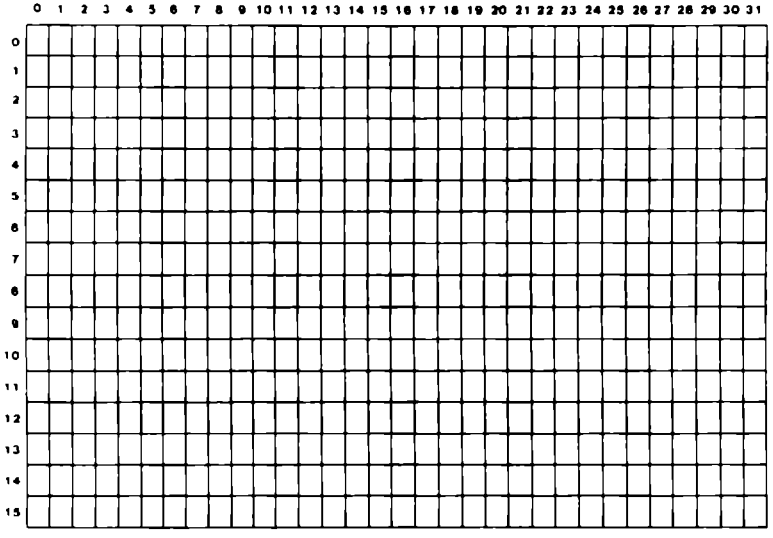


Figure 2.2. RAM vidéo.

La position d'affichage sur l'écran est symbolisée par un curseur clignotant alternativement vert et bleu foncé. Pour des raisons de confort d'utilisation, il est préférable de changer le vert du curseur en unc autre couleur. Il suffit d'appuyer simultanément sur les touches CONTROL et 0.

Si vous répétez cette opération, le curseur deviendra tour à tour : jaune ; bleu clair ; rouge ; blanc ; bleu pâle ; magenta ; orange ; vert.

La couleur de l'affichage des caractères est toujours noire sur fond vert sauf lorsqu'on tape les touches SHIFT et 0 en même temps, auquel cas, les caractères apparaîtront verts sur fond noir. Ce mode n'est présent que pour signaler le passage à un affichage des caractères en minuscules. Si ceux-ci sont destinés à être envoyés vers une imprimante, par exemple. Le BASIC ne reconnaît plus les mots clés s'ils sont entrés dans ce mode.

VISUALISATION À L'ÉCRAN

La première instruction BASIC qu'il est nécessaire d'analyser est bien entendu celle qui permet de visualiser une donnée à l'écran.

Il s'agit bien entendu de PRINT que nous avons déjà introduit dans le premier chapitre.

LES DONNÉES DE PRINT

PRINT ne peut manipuler que deux types de données : des nombres ou des caractères sans aucune signification pour l'ordinateur.

```
10 PRINT 3
```

inscrira un trois à la position courante sur l'écran.

Par contre :

```
10 PRINT A
```

n'imprimera qu'un 0.

L'explication en est simple : le BASIC cherche à interpréter tous les messages entrés à partir du clavier. Pour lui, A n'est ni un nombre, ni une instruction qu'il soit en mesure d'exécuter. Si l'on désire

que la lettre A soit imprimée, il faut lui signaler qu'il ne doit en aucune manière lui chercher une signification dans le cadre du BASIC.

Ce rôle est attribué aux guillemets.

Ainsi :

```
10 PRINT "A"
```

fera bien apparaître la lettre A sur le téléviseur.

Nous pouvons généraliser cette notion en disant que tout caractère entre guillemets est considéré comme un simple symbole sans aucune signification et destiné à être reproduit tel quel par l'ordinateur.

Ces données purement symboliques sont appelées *chaînes de caractères ou alphanumériques* (alphabétique-numérique).

LES SYNTAXES DE PRINT

PRINT peut également être employé avec diverses syntaxes :
Essayons ces trois programmes :

```
10 PRINT "BONJOUR"
20 PRINT "VOUS"

10 PRINT "BONJOUR";
20 PRINT "VOUS"

10 PRINT "BONJOUR",
20 PRINT "VOUS"
```

Les trois messages suivants seront respectivement affichés :

```
BONJOUR
VOUS

BONJOURVOUS

BONJOUR VOUS
```

Si aucune ponctuation n'est employée en complément de PRINT, le programme efface la fin de la ligne puis envoie un *retour-chariot* et l'impression se fera dans la première colonne de la ligne suivante.

Le point virgule annule ce dernier effet et l'affichage se fait à la position immédiatement consécutive.

Quant à la virgule, son action est un peu plus complexe. Elle sépare l'écran en deux portions indépendantes. Elle agit comme une tabulation automatique.

Mais plutôt que d'épiloguer à ce sujet, mieux vaut expérimenter son effet en écrivant un programme du type :

```
10 PRINT "BONJOUR",
20 PRINT "VOUS",
30 PRINT "LA VIE EST BELLE",
40 PRINT "POUR LES PROGRAMMEURS";
```

Sur une même ligne de programme, une seule instruction PRINT suffit à afficher plusieurs données, pourvu qu'elles soient séparées par l'une des deux ponctuations standard. De cette façon, le programme précédent aurait pu être condensé en une seule ligne.

```
10 PRINT "BONJOUR","VOUS","LA VIE EST BELLE",
   "POUR LES PROGRAMMEURS";
```

L'affichage des nombres présente aussi quelques caractéristiques spéciales qui ne se retrouvent pas toujours dans les autres BASIC.

Lors de la visualisation, un nombre quelconque sera précédé par un espace séparateur plus un caractère "-" si le nombre est négatif, ou un autre espace s'il est positif sauf si un point-virgule a été employé. Dans ce cas, l'espace séparateur est éliminé.

```
10 PRINT "NUMERO";3
20 PRINT "NUMERO";-3
```

affichera :

```
NUMERO 3
NUMERO-3
```

LA TABULATION

Il est heureusement possible de positionner les données à imprimer à sa guise. L'instruction PRINT@ permet de sélectionner l'une des 512 positions de l'écran. Si H représente la coordonnée horizon-

tale entre 0 et 31 et V la verticale entre 0 et 15, la position se calcule selon l'équation :

$$32 \times V + H$$

Le programme visualise les deux quantités alphanumériques en lignes 10 et 12, colonne 5 (6e colonne).

```
CLS32768

    PRINT @ 32*10+5,"BONJOUR"
    PRINT @ 32*12+5,"VOUS"
```

Une autre option est représentée par l'utilisation conjointe de PRINT et TAB. TAB émet autant d'espace que spécifié par son opérande. Le résultat en est une tabulation relative par rapport à la position courante d'affichage, mais avec effacement des caractères qui pourraient être entre les deux.

```
10 PRINT@10*32+5,"BONJOUR"
20 PRINT@12*32+5,"BONJOUR"
30 PRINT@ 12*32," ";
40 PRINT TAB(8);"SOIR"
```

Remarquez la ligne 30 qui permet de tabuler l'écran à la ligne 12 sans effacer le message déjà inscrit (utilisation du point-virgule).

L'effacement complet de l'écran s'obtient avec la commande CLS. Le curseur est alors positionné en ligne 0 et colonne 0. CLS peut être suivie d'un nombre, auquel cas, la couleur du fond de l'écran est modifiée. Cet opérande est un nombre compris entre 0 et 255, mais seules les valeurs de 0 à 8 sont significatives.

CALCUL NUMÉRIQUE

Les nombres que peut traiter Alice sont des réels positifs ou négatifs dont la plage de variation est située entre les bornes :

10^{38} et -10^{38}

La notation informatique de ces nombres en *virgule flottante* suit les règles suivantes :

- E représente l'exposant ou la puissance de dix associée.
- Le point est l'équivalent de notre virgule.

Ainsi 1000 peut s'écrire 1E+3,

0,001 1E−3.

La représentation sur l'écran est au maximum de neuf chiffres. Il faut signaler que dans le cas où cette limite est atteinte, la dernière décimale est arrondie.

Les performances d'Alice, et de tous les autres ordinateurs familiaux d'ailleurs, sont nettement inférieures à celles des calculettes spécialisées.

Bien entendu, le BASIC dispose des opérateurs algébriques classiques :

Addition	+
Soustraction	−
Division	/
Multiplication	*

La méthode de programmation des calculs est exactement calquée sur notre façon de travailler à l'aide d'une feuille de papier. Le calcul s'effectue de gauche à droite, les opérateurs / et * sont prioritaires sur + et −. Les parenthèses modifient ces priorités.

Exemple 1

```
10 PRINT 10+3
```

```
- > 13
```

Exemple 2

```
10 PRINT 10+3*5
```

```
- > 25
```

Exemple 3

```
10 PRINT (10+3)*5
```

```
- > 65
```

Exemple 4

```
10 PRINT "8+7="
20 PRINT 3+2
```

```
-> 8+7=5
```

Cet exemple démontre bien la différence que fait l'ordinateur cntre une chaîne de caractères et des quantités numériques.

Attention, si vous désirez voir le résultat, il est nécessaire d'ordonner à la machine de le visualiser à l'écran par un PRINT.

LES VARIABLES

LES VARIABLES NUMÉRIQUES

Certaines quantités numériques ne peuvent pas être directement incorporées dans un programme BASIC, car elles ne sont pas constantes. Il est alors nécessaire de les *paramétrer*. En clair, cela signifie que l'on fait référence à sa représentation et non plus à sa valeur.

Prenons par exemple une équation simple qui nous permet de retrouver la distance parcourue en fonction de la vitesse moyenne et du temps. Elle peut se symboliser sous la forme :

$$D = V \times T$$

avec D : distance ; V : vitesse ; T : temps.

Sous cette forme, l'équation reste très générale. Pour une application donnée, il suffit de remplacer les quantités symboliques par leur valeur numérique.

D, V et T sont des variables.

Le BASIC est prévu pour employer des variables. Lors de l'exécution, toute variable rencontrée est remplacée par la valeur qu'elle contient à ce moment.

Il suffit qu'elle soit mentionnée pour qu'elle soit créée. Dans le cas de figure qui nous préoccupe, une variable est une zone de la mémoire de travail réservée. Elle occupe sept cases de mémoire dont deux pour le nom de la variable et cinq pour la valeur résidente.

Le label mentionné dans le programme peut avoir un grand nombre de caractères, mais seuls les deux premiers sont significatifs.

Exemple 1

```
10 ADRESSE = 10
20 PRINT ADMISE
```

```
-> 10
```

ADresse et ADmise sont un seul et même label pour l'interpréteur. Cette caractéristique est la cause de bon nombre d'erreurs difficiles à découvrir.

Le BASIC cherche d'abord à détecter les mots clés de son vocabulaire, le nom d'une variable ne doit donc pas contenir le nom d'une instruction.

Exemple 2

```
10 PRINTEMP = 0
```

sera interprété comme PRINT EMP = 0.

Nous verrons que cette ligne de programme ne "perturbe" absolument pas l'interpréteur ; aucun message d'erreur n'est émis, mais l'effet produit n'a plus aucun rapport avec le but du programmeur. C'est l'une des deux ou trois erreurs les plus difficiles à mettre en évidence dans un programme un tant soit peu évolué.

Il est conseillé d'utiliser des variables dont le label ne dépasse pas deux caractères. Si le listing résultant est moins lisible, les possibilités d'erreurs en sont d'autant réduites.

Les variables imposent les contraintes suivantes :

- Le nom doit commencer par un caractère alphabétique. AB, A1, B5 ...
- Seuls les deux premiers caractères sont significatifs.
- Le nom ne doit pas contenir un des mots clés du BASIC.

En revanche, l'utilisation des variables numériques en BASIC est extrêmement souple. Elle est basée sur le symbole d'affectation " = ", une autre syntaxe est également tolérée ; LET ... = . Mais *attention*, = n'est pas ici l'opérateur mathématique. A = B est différent de B = A. L'expression de droite est d'abord évaluée puis stockée dans la variable à gauche du signe.

Ceci explique que des expressions comme A = A + 1 sont parfai-

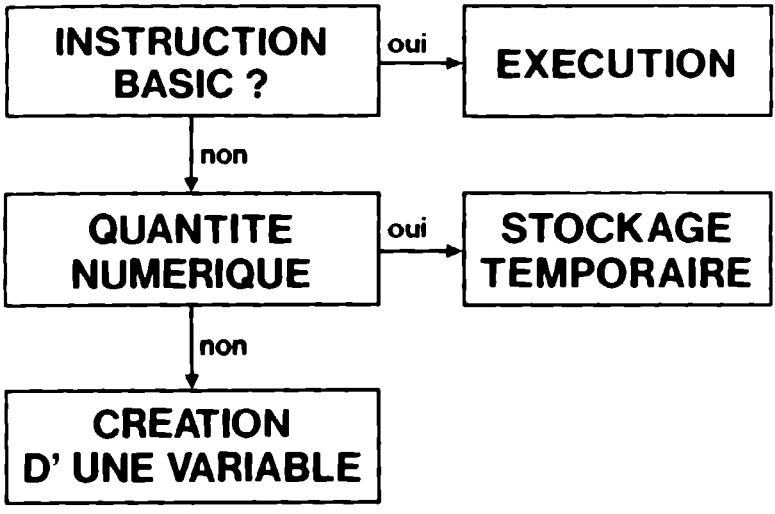


Figure 2.3. Analyse de l'interprétation.

tement exactes en informatique ; la valeur finale de A sera égale à sa valeur initiale plus 1.

Exemple 3

```
10 A=10
20 B=22
30 C=A+B
40 PRINT A;"+";B;"=";C
```

On obtient :

```
-> 10+22=32
```

Signalons enfin qu'à sa création une variable numérique contient la valeur 0 et ce, jusqu'à ce qu'une valeur lui soit affectée. L'instruction de démarrage RUN réinitialise les variables.

LES VARIABLES ALPHANUMÉRIQUES

Les données numériques ne sont pas les seules à pouvoir être symbolisées. Il en va de même des chaînes de caractères. Toutes les règles générales restent valables dans ce cas. Il faut cependant toujours tenir compte du type de la variable, car selon les instructions

qui sont paramétrées, celles-ci réclament un opérande soit numérique, soit alphanumérique. Si une erreur survient en cours d'exécution, un message d'erreur est affiché (voir Annexe 2).

L'instruction " + " est une des exceptions à cette règle. Mais elle agit différemment selon le type de variable utilisée.

La différenciation se fait à l'aide d'un symbole particulier : "$".

Les variables A et A$ sont donc traitées différemment par l'interpréteur.

Exemple 1

```
10 A$= "BON"
20 B$= "JOUR"
30 C$= "SOIR"
40 D$= A$+B$
50 E$= A$+C$
60 PRINT "A$=";A$;
70 PRINT "B$=";B$;
80 PRINT "C$=";C$
90 PRINT D$;"  ";E$
```

On obtient :

```
- >      A$=BON B$=JOUR C$=SOIR
       BONJOUR BONSOIR
```

Le signe " + " est un opérateur de *concaténation* pour les chaînes de caractères. Il met bout à bout les deux chaînes avant de les stocker dans la variable réceptrice.

STRUCTURE ET CODE ASCII

La structure des variables alphanumériques est plus complexe que celle des numériques. Si l'apparence reste la même, deux octets pour le label et cinq pour le stockage, le processus de lecture et d'écriture se fait en temps. Les cinq octets contiennent la longueur de la chaîne de caractères ainsi que son emplacement dans une autre mémoire réservée à cet effet.

Nous verrons comment mettre à profit une telle structure de manière rusée. Mais les chaînes sont de taille réduite, au maximum 255 caractères (retenez bien ce nombre car il est l'une des clés de la

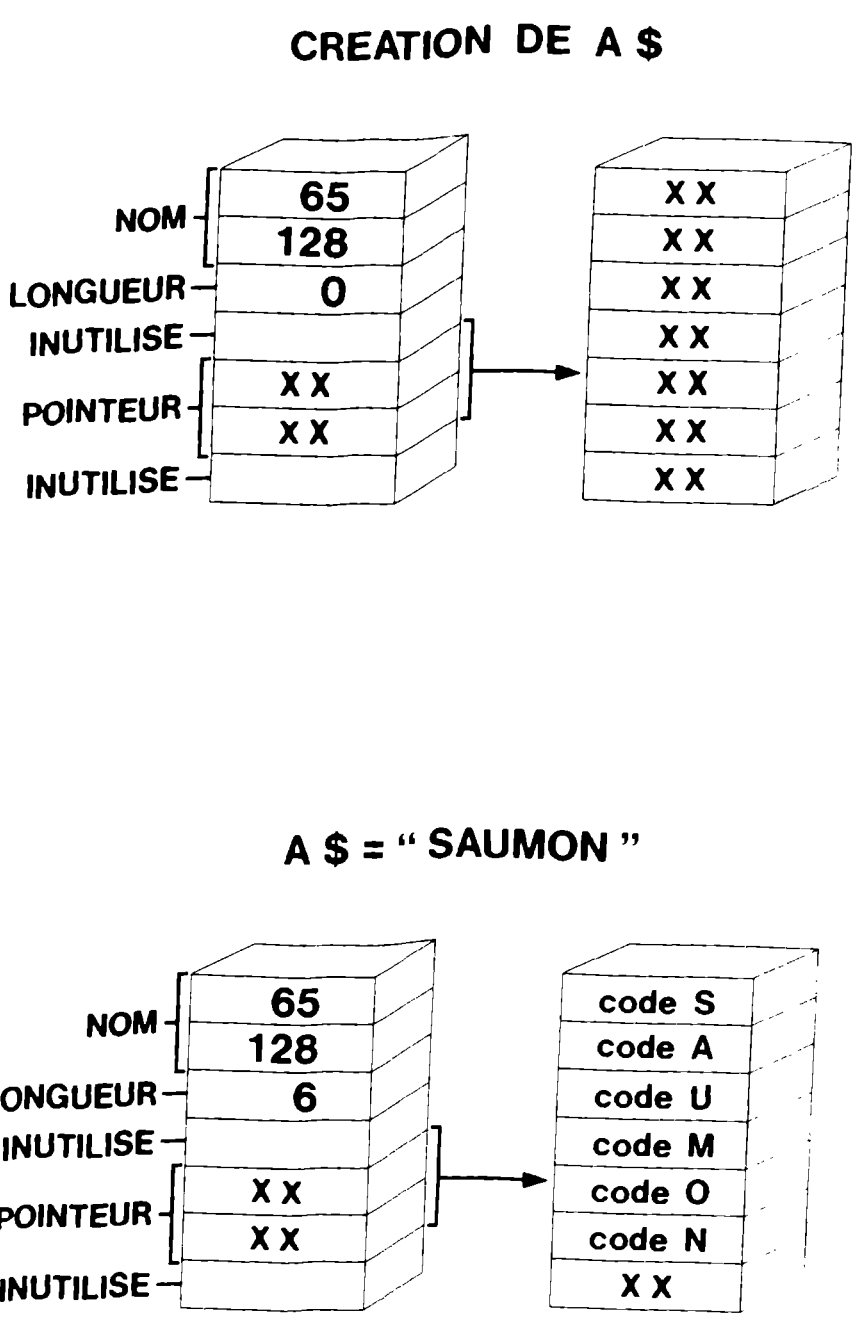


Figure 2.4. Écriture des variables alphanumérique.

micro-informatique). Mais pour parvenir à cette limite, il est nécessaire d'intervenir directement au niveau de la variable.

En utilisation normale, le nombre de caractères susceptibles d'être emmagasinés dans la mémoire spéciale est de 100. En cas de dépassement, un message d'erreur apparaît à l'écran. Il est heureusement possible de pallier cet inconvénient à l'aide de l'instruction CLEAR suivie d'un nombre qui représente la taille en nombre de caractères, de la mémoire des variables alphanumériques. En fait, CLEAR a d'autres effets secondaires :

- Mise à 0 des variables numériques.
- Les variables alphanumériques sont égales à la chaîne vide "".

La mémoire restreinte d'Alice ne permet pas toujours de jongler comme on le voudrait avec les chaînes de caractères. L'illustration en sera faite lorsque nous verrons la réalisation d'un fichier d'adresses.

Nous avons vu que les caractères étaient enregistrés dans la mémoire vive (RAM). Mais la forme sous laquelle ils le sont n'est pas très simple à comprendre, du moins pour le débutant.

En fait, le processeur n'est pas capable de traiter des données autres que des nombres. Il a donc fallu user d'un artifice pour qu'il traite des textes divers.

La transition entre le symbole d'imprimerie et la donnée informatique est réalisée grâce à un codage numérique standardisé des touches du clavier ; c'est la norme ASCII (*American Standard Code for Information Interchange*).

L'ensemble des codes est répertorié dans l'Annexe 1. La valeur du code associé à un caractère est comprise entre 0 et 255. Le processeur peut ainsi travailler sur des nombres, et réaliser une simulation de traitement des textes.

Dans le cas des variables alphanumériques, ce sont les codes de tous les caractères qui sont enregistrés. La visualisation s'opère selon le processus décrit au Chapitre 1.

LES TABLEAUX

Dans beaucoup de programmes, il est pratique d'avoir à sa disposition une série de variables entre lesquelles il existe une corrélation. Une table de multiplication, un déterminant de matrice, un tableau comparatif sont des structures nécessitant ce type de variables. Le

BASIC dispose de cette option sous le nom de *tableau* ou plus rarement de *vecteur*. Un tableau informatique est un ensemble de variables élémentaires qui sont repérées en fonction de leur indice.

Un tableau peut avoir plusieurs dimensions. Une liste n'est qu'un tableau à une dimension, une table à deux dimensions, etc.

Chacune des cases du tableau se comporte exactement comme une variable numérique ou alphanumérique. Mais il est nécessaire de signaler à l'interpréteur le nombre maximum d'éléments qu'est susceptible de contenir un vecteur donné. Il réservera alors une place dans la mémoire de travail. Cette opération est le *dimensionnement* ; il est effectué par l'instruction DIM.

Exemple 1

Si nous désirons fabriquer un tableau numérique nommé TA possédant deux dimensions avec respectivement dix et douze éléments pour chacune d'entre elles, il suffit de taper l'instruction

```
10 DIM TA (10,12)
```

Le redimensionnement au cours d'un programme est formellement interdit, aussi est-il préférable de dimensionner les tableaux une fois pour toutes en début de programme. Si aucune instruction DIM n'a été mentionnée, un tableau sera automatiquement formaté à 10 par dimensions, soit onze éléments numérotés de 0 à 10.

Exemple 2

```
10 DIM B(1,1)
20 B(0,0)=1
30 B(1,0)=2
40 B(0,1)=3
50 B(1,1)=4
60 PRINT B(0,0)
70 PRINT B(1,0)
80 PRINT B(0,1)
90 PRINT B(1,1)
```

On obtient :

```
 - >   1
       2
       3
       4
```

Comme on le constate, chaque élément constitutif du tableau peut être considéré comme indépendant. L'avantage de ce procédé est de pouvoir sélectionner une donnée par calcul.

Exemple 3

Si nous rajoutons les lignes :

```
100 A=1
110 PRINT B(A,A-1)
```

On obtient :

```
- >    1
       2
       3
       4
       2
```

Les tableaux peuvent aussi être déclarés avec des variables alphanumériques avec les mêmes conventions que pour les variables simples.

Le nombre de dimensions utilisables pratiquement est limité à 10, mais il faut savoir que la mémoire risque d'être bien vite saturée avec des tableaux moins vastes mais possédant un nombre plus élevé d'éléments par dimension.

FONCTIONS MATHÉMATIQUES

Malgré des performances moyennes en calcul numérique pur, Alice est doté de tout un éventail de fonctions mathématiques pures.

Un petit détail peut s'avérer gênant lors de l'utilisation des fonctions trigonométriques. Le calcul fait appel à des développements limités.

Malheureusement, aucune conversion de mesure d'angle n'est prévue, il faut donc utiliser des opérandes en *radians*.

Exemple 1

```
10 PI=3.14159
20 PRINT COS (PI/2)
30 PRINT SIN (PI/2)
40 PRINT TAN (PI/2)
```

On obtient :

```
- >     1.32613524E-06
        1
        754070.905
```

L'imprécision du premier résultat provient du fait que notre PI n'est pas très précis, mais la méthode de calcul entraîne également une erreur.

Les autres fonctions ne posent aucun problème. Pour leur syntaxe, référez-vous à l'Annexe 3.

3

LES BASES DE LA PROGRAMMATION

Généralement, un programme d'ordinateur ne peut pas toujours avoir ses données de travail incorporées.

Considérons par exemple un distributeur de billets de banque. Il s'agit d'un petit ordinateur qui ne possède qu'un programme. Celui-ci a besoin de données extérieures pour être capable d'effectuer le traitement pour lequel il a été conçu. En l'occurence, le numéro de code de l'utilisateur.

Un programme doit donc pouvoir suspendre temporairement son exécution et redonner le contrôle au clavier jusqu'à ce que l'entrée des données soit terminée.

Bien évidemment, il existe une instruction BASIC qui assure ce rôle de messager : INPUT.

Elle dispose de presque autant de syntaxes différentes que PRINT.

Il est nécessaire de lui indiquer une ou plusieurs variables réceptrices des données externes. Dans le cas où plusieurs variables sont concernées, il est impératif qu'elles soient séparées par une virgule.

Examinons plutôt la signification d'une ligne de programme comme :

```
10 INPUT A$
```

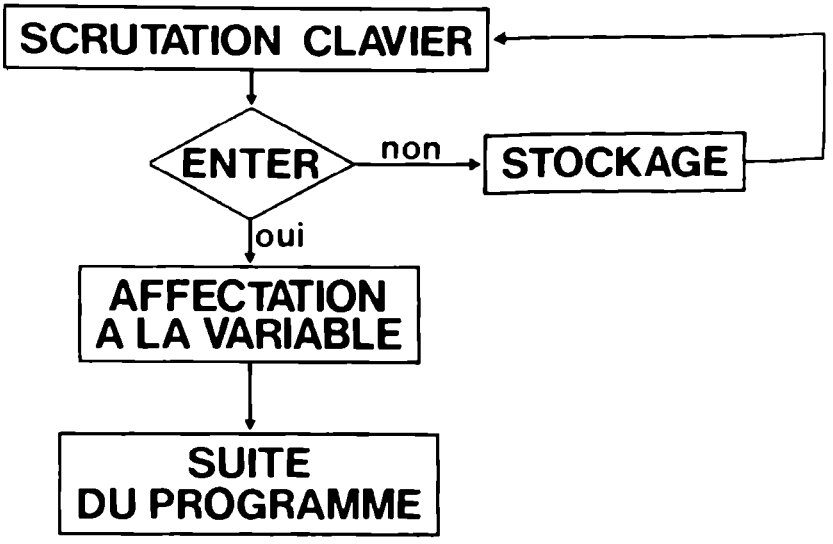


Figure 3.1. Organigramme INPUT.

Si la donnée est incompatible avec la variable réceptrice, INPUT refuse l'entrée et en demande une nouvelle en affichant un message REDO.

Nous verrons au chapitre suivant une méthode de programmation permettant de pallier cet inconvénient.

INPUT affiche toujours un point d'interrogation sur l'écran pour signifier l'attente d'une entrée.

Une bonne habitude consiste à afficher un message descriptif dans le but de guider l'opérateur.

Exemple

```
10 PRINT "QUEL EST VOTRE NOM";
20 INPUT N$
30 PRINT N$;"VOUS ETES BELLE"
40 PRINT
50 PRINT "DANS LE DOUTE"
60 PRINT "SOYONS GALANT !!"
```

On obtient :

```
- >    QUEL EST VOTRE NOM ? MIMI ENTER
       MIMI VOUS ETES BELLE
       DANS LE DOUTE
       SOYONS GALANT !!
```

Il n'y a plus de dilemme, vous ne risquez plus de rentrer le prix du kilo de colle forte en lieu et place de votre nom (à moins d'en avoir un vraiment original).

INPUT est prévu pour afficher une telle phrase optionnelle. On ne peut malheureusement pas profiter de toutes les possibilités de tabulation de PRINT.

Exemple

```
10 INPUT "VOTRE NOM";N$
20 PRINT N$;"VOUS ETES";
30 PRINT "TOUJOURS BELLE"
```

On obtient :

```
- >    VOTRE NOM ? MIMI ENTER
       MIMI VOUS ETES TOUJOURS BELLE
```

Mais certains programmes requièrent l'entrée d'une seule donnée sans interruption visible du traitement. Vous avez certainement déjà joué avec une console de jeu, voire une machine de café. Votre action sur le clavier ou les manettes de jeu est prise en compte par le programme sans que celui-ci ne s'arrête.

Cette saisie au vol est le rôle de INKEY$. La variable réceptacle doit obligatoirement être alphanumérique.

En réalité, INKEY$ va observer le buffer du clavier (un octet) et voir si une touche a été actionnée depuis la précédente scrutation. Si tel est le cas, la valeur de la touche est affectée à la variable, sinon celle-ci est réinitialisée à l'élément vide "".

L'ensemble du processus ne prend que quelques millièmes de seconde, ce qui explique son caractère quasi instantané. INKEY$ est plus souple que INPUT en ce sens que chaque caractère entré peut être testé.

L'Annexe 3 donne un exemple de simulation de INPUT grâce à INKEY$.

PRISE DE DÉCISION

Bien entendu, un programme doit posséder des options de décision. Reprenons l'exemple du distributeur de billets de banque. Si le code est conforme, l'opération se déroule correctement. Si, par contre, la donnée est erronée, la carte sera avalée.

Face à une action précise, le programme de gestion peut réagir selon deux schémas directeurs, en fonction de l'état de l'environnement.

Il est donc capable de tester et de décider en fonction des conséquences de ces tests.

```
SI CODE VALABLE ALORS ACTION SUIVANTE
SINON AVALER LA CARTE
```

Le programme aurait pu être écrit dans un tel langage.

Mais le BASIC est d'origine anglo-saxonne, et l'instruction BASIC se traduit par IF THEN.

La partie située entre IF et THEN est un test de véracité. Si le résultat est faux (0 pour Alice), l'interpréteur se déroutera sur la ligne suivante sans exécuter les instructions en aval de THEN. Sinon, celles-ci sont exécutées.

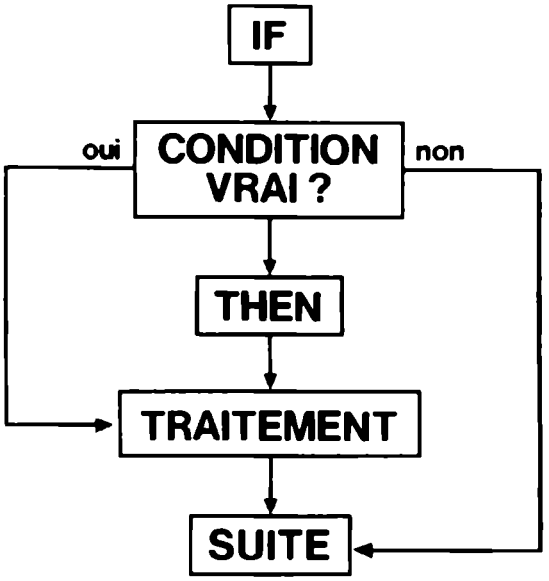


Figure 3.2. Diagramme de IF THEN.

Si la programmation de l'alternative est maintenant possible, encore faut-il pouvoir réaliser un test ou une comparaison.

Le BASIC possède un nombre confortable d'opérateurs de comparaison :

- = test d'égalité ;
- > strictement supérieur ;
- < strictement inférieur ;
- < > différent de ;
- > = supérieur ou égal ;
- < = inférieur ou égal.

Ces tests génèrent les valeurs 0 si le test est faux ; -1 dans les autres cas. Notons qu'ils sont aussi applicables aux quantités alphanumériques au sein desquelles ce sont les codes ASCII eux-mêmes qui sont comparés un à un.

Exemple

```
10 A$="CLONE"
20 B$="EDEN"
30 IF A$=B$ THEN PRINT "ÉGALITÉ"
40 IF A$<>B$ THEN PRINT "DIFFÉRENT"
50 IF A$<B$ THEN PRINT "INFÉRIEUR"
60 IF A$>B$ THEN PRINT "SUPÉRIEUR"
```

On obtient :

```
- >    DIFFÉRENT
       INFÉRIEUR
```

Le codc de C est plus petit que celui de E (voir Annexe 1), donc CLONE est bien inférieur à EDEN. Essayez plusieurs valeurs de A$ et B$.

BRANCHEMENTS

Nous avons vu que l'interpréteur exécutait les instructions en fonction de leur numéro de ligne. Du plus petit au plus grand. Il est toutefois utile de pouvoir le dérouter à certains moments à l'intérieur du programme. L'instruction GOTO le forcera à examiner la ligne mentionnée dans l'opérande. Toutefois, il est possible d'omettre GOTO si le branchement est conditionnel, c'est-à-dire s'il est placé juste après un THEN.

```
 5 INPUT "VALEUR DE A:";A
10 IF A=10 THEN GOTO 50
20 PRINT "A EST DIFFÉRENT DE 10": GOTO 60
50 PRINT "A ÉGAL 10"
60 GOTO 5
```

On obtient :

```
- >    8 ENTER
       A EST DIFFERENT DE 10
       10 ENTER
       A EGAL 10
```

La ligne 10 aurait pu être écrite :

```
10 IF A=10 THEN 50
```

EXEMPLE DE STRUCTURE ALTERNATIVE

Nous allons aborder ce sujet par l'étude d'un exemple d'application. Voyons comment fabriquer un programme de nombre mystérieux. Il peut se résumer ainsi :

CHOIX D'UN NOMBRE ALÉATOIRE
SCORE À 0
ENTRÉE D'UN NOMBRE
AUGMENTER SCORE
SI DONNÉE INFÉRIEURE AU NOMBRE MYSTÉRIEUX
ALORS IMPRIMER "PLUS GRAND" PUIS RETOURNER
À L'ENTRÉE D'UN NOMBRE
SI DONNÉE SUPÉRIEURE ALORS IMPRIMER
"PLUS PETIT" PUIS RETOURNER À L'ENTRÉE
D'UN NOMBRE
ÉCRIRE BRAVO AINSI QUE LE SCORE
FIN

Pour ceux qui préfèrent les schémas, voici l'organigramme du programme.

Traduite en BASIC, la petite analyse précédente donne le listing suivant :

```
5  SC=0
10 CLS
20 A=RND(999)
30 INPUT "NOMBRE";N
35 SC=SC+1
40 IF N<A THEN PRINT
   "PLUS GRAND":GOTO 30
50 IF N>A THEN PRINT
   "PLUS PETIT" GOTO 30
60 PRINT "BRAVO"
70 PRINT "VOTRE SCORE
```

RND génère un nombre au hasard, entre 1 et 999 dans ce cas. Les deux points signalent à l'interpréteur que plusieurs instructions sont présentes sur la même ligne de programme.

Comme on peut le constater, si le test n'est pas vérifié, *ce sont toutes les instructions après THEN qui sont ignorées.* Bien entendu, le but du jeu est de découvrir le nombre mystérieux. Nous avons bien affaire à un programme décisionnel, puisqu'il est capable de réagir différemment selon la donnée introduite.

OPÉRATEURS LOGIQUES

Les structures alternatives sont une des bases de la programmation et, souvent, plusieurs paramètres doivent être testés avant toute indirection (branchement). Jusqu'à maintenant, nous avons vu que si une seule comparaison était facile à mettre en œuvre, plusieurs critères sont plus difficiles à évaluer.

Reprenons une dernière fois ce fameux guichet automatique. L'algorithme de ce programme suit le schéma suivant :

```
SI CODE CORRECT ET SI COMPTE APPROVISIONNÉ
ALORS DONNER LES BILLETS
```

ET constitue un opérateur logique permettant de regrouper les différents tests.

L'ensemble est vrai si et seulement si les deux tests sont vrais. Dans tous les autres cas, la proposition globale est considérée comme fausse.

L'effet de ET peut être symbolisé par une table de vérité où, par convention, V signifie vrai et F, faux.

Soient les résultats de deux tests A et B :

AND	B	V	F
A			
V		V	F
F		F	F

AND étant l'opérateur BASIC "ET".

Si A vrai, B vrai alors A AND B vrai.
Si A faux, B vrai alors A AND B faux.
Si A vrai, B faux alors A AND B faux.
Si A faux, B faux alors A AND B faux.

Un autre opérateur logique est utilisable sur Alice, qui représente le OU du langage courant : OR.

Les programmeurs l'appellent OU inclusif.

En voici la table de vérité :

OR	B	V	F
A			
V		V	V
F		V	F

Si A vrai, B vrai alors A OR B vrai.
Si A vrai, B faux alors A OR B vrai.
Si A faux, B vrai alors A OR B vrai.
Si A faux, B faux alors A OR B faux.

Un nombre quelconque de tests peut ainsi être chaîné pourvu que la longueur totale ne dépasse pas 127 caractères (limite d'une ligne de programme). Les parenthèses peuvent privilégier certaines des séquences de l'instruction. La prise de décision peut ainsi être fonction de la valeur d'une dizaine de paramètres.

Si nous avions à programmer une sortie au restaurant nous pourrions le faire de cette façon :

```
SI (COMPTE APPROVISIONNÉ OU INVITATION) ET
ACCOMPAGNÉ PAR JOLIE FILLE
ALORS ALLER RESTAURANT.
```

Il suffit d'ajuster les paramètres et de traduire le texte en anglais,

pour obtenir la version BASIC de cette évaluation de la situation. La quantité entre parenthèse est la première à être "calculée" et cc, quelle que soit sa position dans le test.

Autrement, le traitement est effectué de la gauche vers la droite.

SIMULATION DE BOUCLE

Les ordinateurs sont bien connus pour leur capacité à mener à bien des tâches répétitives.

Il faut malgré tout les programmer.

```
10 PRINT "BONJOUR"
20 PRINT "BONJOUR"
```

Est bien entendu une tâche répétitive. Mais imaginez que vous ayez à imprimer un message un nombre infini de fois.

Votre vie n'y suffirait pas, ni la capacité de votre appareil.

Il est bien plus intéressant de dérouter l'interpréteur vers le traitement à effectuer dès la fin de celui-ci.

```
10 PRINT "BONJOUR"
20 GOTO 10
```

Le cahier des charges est bien rempli, mais la boucle de programme n'est pas contrôlée. Seule une interruption par la touche BREAK, INIT ou SHIFT @ peut stopper le programme.

L'étape suivante est de réaliser une boucle paramétrée.

Pour se faire, il faut disposer d'une variable qui compte les "tours" jusqu'à une valeur limite spécifiée par l'utilisateur.

```
10 INPUT "VALEUR LIMITE";VL
20 CO=1
30 PRINT "BONJOUR"
40 CO=CO+1
50 IF CO <= VL THEN 30
60 PRINT "FIN DE BOUCLE"
```

On obtient :

```
- >    VALEUR LIMITE?3 ENTER
       BONJOUR
       BONJOUR
       BONJOUR
       FIN DE BOUCLE
```

Si une valeur 1000 avait été mentionnée, 1000 "bonjour" auraient été imprimés. La boucle est ainsi maîtrisée.

Il suffit de paramétrer de la même façon la valeur inférieure de la boucle.

Vous avez sans doute remarqué que le compteur augmentait de 1 à chaque passage (les informaticiens appellent cette opération l'*incrémentation*). Il est évident qu'en changeant cet incrément, il est aisé d'écrire tous les nombres compris entre 1 et 1000 avec un "pas" de 0.5.

```
10 INPUT "DE,JUSQU'À PAS";VI,VL,PA
20 CO=VI
30 PRINT CO
40 CO=CO+PA
50 IF CO <= VL THEN 30
60 PRINT "FIN DE BOUCLE"
```

On obtient :

```
- >    DE,JUSQU'À,PAS ?2,5,0.5 ENTER
       2
       2.5
       3
       3.5
       4
       4.5
       5
       FIN DE BOUCLE
```

Nous sommes en mesure de faire répéter une tâche autant de fois que nécessaire, en contrôlant l'intégralité des paramètres de la boucle.

LES BOUCLES

La méthode que nous venons de décrire a le mérite de fonctionner, mais elle est grosse consommatrice de variables et de lignes de programme donc de place en mémoire centrale.

En fait, tout le travail que nous venons d'accomplir n'a qu'un intérêt purement didactique, car le BASIC est capable de réaliser ce type de tâche grâce à des instructions spéciales incorporées.

FOR indique quelle variable fait office de compteur et lui fournit sa valeur initiale.

TO précise la valeur finale du compteur.

STEP sert à introduire le pas de l'incrémentation. Il est optionnel et, dans le cas où il n'est pas utilisé, le pas est automatiquement de 1.

NEXT délimite la fin de la boucle, et renvoie l'interpréteur au début de celle-ci, tant que le compteur n'a pas atteint la valeur limite.

Le processus suivi est absolument identique à celui que nous avons simulé au paragraphe précédent.

```
10 FOR CO=1 TO 10
20 PRINT "BONJOUR"
30 NEXT CO
```

- > Imprimera dix fois BONJOUR sur le téléviseur.

La mention du label de la variable indice de boucle est optionnelle dans le cadre de NEXT, mais il vaux mieux prendre l'habitude de procéder ainsi pour que l'interpréteur puisse différencier les boucles au cas où plusieurs d'entre elles sont présentes dans une phase du programme.

Il est bon de prendre certaines précautions lors de l'emploi des fonctions de boucle, notamment lorsqu'elles sont imbriquées.

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 10
30 PRINT "TEST"
40 NEXT J
50 NEXT I
```

- > Cent fois TEST.

Le programme est parfaitement reconnu par l'interpréteur car les deux boucles se comportent exactement comme des "poupées russes".

```
10 FOR I = 1 TO 9
20 FOR J = 1 TO 10
```

```
30 PRINT "RATAGE"
40 NEXT I
50 NEXT J

>  RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   RATAGE
   ?NF ERROR IN 50
```

Les boucles ne sont pas correctement imbriquées, ainsi, lors de l'exécution de la première boucle en I, l'erreur n'est pas encore mise à jour. En revanche, une fois arrivé en ligne 50, il rencontre un NEXT J qui, pour lui, ne correspond plus à rien. Un message d'erreur est alors émis.

Il faut signaler qu'une seule instruction NEXT peut gérer plusieurs indices de boucle.

```
10 FOR I = 1 TO 3
20 FOR J = 1 TO 2
30 PRINT "TEST"
40 NEXT J,I

>  TEST
   TEST
   TEST
   TEST
   TEST
   TEST
```

Mais l'ordre des indices est extrêmement important, pour les raisons précédemment exposées. Il est donc fortement recommandé d'oublier cette option, du moins jusqu'à une maîtrise du BASIC suffisante.

LES TRIS

Tri et échec sont sans doute les domaines où les spécialistes ont

le plus fait preuve de ténacité et de créativité. S'il est hors de question d'écrire un jeu d'échec en BASIC, surtout sur Alice, certaines méthodes de tri numérique ou alphabétique, bien adaptées à cet ordinateur, nous sont maintenant accessibles.

Les deux exemples que nous allons développer dans ce paragraphe traitent des listes alphabétiques. La généralisation aux tris numériques en est évidente.

La première étape consiste à établir une liste des données à trier.

Un tableau alphanumérique semble être le moyen le mieux adapté pour parvenir à ce résultat.

Mais plus importante est la méthode de tri utilisée. En voici l'exemple le plus simple :

```
PRENDRE LE PREMIER ÉLÉMENT
COMPARER AVEC LE SECOND
S'IL EST PLUS GRAND, ÉCHANGER CES ÉLÉMENTS
DU TABLEAU
PRENDRE LE TROISIÈME ÉLÉMENT
COMPARER AVEC LE PREMIER
ÉCHANGER S'IL Y A LIEU
PRENDRE LE ÉNIÈME
COMPARER AVEC LE PREMIER
ÉCHANGER S'IL Y A LIEU
PRENDRE LE SECOND ÉLÉMENT
RECOMMENCER LE PROCESSUS
PRENDRE LE ÉNIÈME ET RECOMMENCER LE PROCESSUS
```

Après la première passe, le premier élément est dans le bon ordre ; après la seconde, les deux premiers éléments sont ordonnés.

Ce programme demande toujours :

$$1+2+3+4.....+ N$$

soit $N/2 \times (N+1)$ passages, ou N représente le nombre de données à trier moins une.

Le gros défaut provient du fait que, même si la liste est ordonnée ou partiellement ordonnée, le programme s'exécutera dans son ensemble. Si ce n'est pas trop grave dans le cas où le nombre de données reste restreint (environ cinquante), imaginons le temps nécessaire au classement de vingt mille mots compte tenu du fait qu'environ trois secondes sont nécessaires pour neufs prénoms féminins.

Voyons la programmation en tant que telle.

```
5 DIM AD$(30):I=0
10 CLS
20 INPUT "MOT SUIVANT";B$
30 IF B$="" THEN 100
40 AD$(I)=B$
50 I=I+1:IF I=30 THEN PRINT
   "FIN DE LISTE":GOTO 100
60 GOTO 20
100 CLS
110 PRINT@32*8+6,"TRI EN COURS"
120 FOR J=0 TO I
130 FOR K=J+1TO I
140 IF AD$(J)   AD$(K) THEN
    B$= AD$(K):AD$(K)=AD$(J):
    AD$(J)=B$
150 NEXT K
160 NEXT J
170 CLS
200 FOR J=0 TO I
210 PRINT AD$(J)
215 IF J=10 THEN PRINT,  INPUT
    "TAPEZ ENTER";A$
220 NEXT J
230 INPUT "TAPEZ ENTER";A$
240 CLS
250 PRINT:PRINT
260 PRINT"  IMPRESSION (O/N)
270 INPUT A$
280 IF A$<>"O" THEN 10
290 FOR J=0 TO I STEP 2
300 LPRINT AD$(J),AD$(J+1
400 NEXT J
420 CLEAR:GOTO 10
```

Les lignes 10 à 60 représentent l'établissement de la liste. Celle-ci est volontairement limitée à 31, place mémoire oblige. Si un message OS ERROR survient, CLEAR (Annexe 3) permettra peut-être de régler ce petit problème.

Si ENTER seul est actionné, ligne 30, le tri commence, lignes 100-160.

Le résultat est ensuite visualisé, lignes 170-230.

Le résultat peut être sauvegardé sur une imprimante en employant la commande LIST.

```
      LISTE

MALIKA              SYLVIE
SOPHIE              CAROLE
EUPHRASIE           BO
EMMA                NATHALIE
CAROLINE


     LISTE TRIEE
BO                  CAROLE
CAROLINE            EMMA
EUPHRASIE           MALIKA
NATHALIE            SOPHIE
SYLVIE
```

Deux améliorations peuvent facilement être faites à ce programme.

L'une sur le fond, un algorithme de tri différent ; l'autre sur un détail : en effet, lors de la visualisation des résultats, il n'est pas possible de stopper le défilement du listing. Si la liste est longue, il devient donc vital de prévoir une pause toutes les dix données pour que l'utilisateur ne soit pas frustré.

Toutes ces modifications sont contenues dans le listing suivant :

```
5 DIM AD$(30):I=0
10 CLS
20 INPUT "MOT SUIVANT";B$
30 IF B$="" THEN 100
40 AD$(I)=B$
50 I=I+1:IF I=30 THEN PRINT
   "FIN DE LISTE":GOTO 100
60 GOTO 20
100 CLS
110 PRINT@32*8+6,"TRI EN COURS"
120 FOR J=0 TO 1
130 FOR K=0 TO I
140 IF AD$(K) > AD$(K+1) THEN
    B$=AD$(K):AD$(K)=AD$(K+1):
    AD$(K+1)=B$:FL=1
150 NEXT K
155 IF FL=0 THEN 170
157 FL=0
160 NEXT J
```

```
170 CLS
200 FOR J=0 TO I
210 PRINT AD$(J)
215 IF J=10 THEN PRINT;: INPUT
    "TAPEZ ENTER";A$
220 NEXT J
230 INPUT "TAPEZ ENTER";A$
240 CLS
250 PRINT:PRINT
260 PRINT"  IMPRESSION (O/N) :";
270 INPUT A$
280 IF A$<>"O" THEN 10
290 FOR J=0 TO 1 STEP 2
300 LPRINT AD$(J),AD$(J+1)
400 NEXT J
420 CLEAR:GOTO 10
```

La ligne 215 crée une pause dans l'affichage jusqu'au moment où l'utilisateur tape ENTER.

Si cette méthode est valable dans le cas où moins de vingt éléments ont été triés, elle n'est pas générale.

Une des solutions de ce problème serait de dire :

DIVISER COMPTEUR PAR 10
PARTIE ENTIÈRE DE COMPTEUR DIVISE PAR DIX
SI ÉGAL, ALORS PAUSE

Si la valeur du compteur est égale à 0 modulo 10, c'est-à-dire si elle est un multiple de dix, alors il faut suspendre le défilement.

Si la division génère un nombre entier, alors la condition est réalisée.

La ligne 215 devient alors

```
215 IF J/10 = INT (J/10) THEN PRINT;: INPUT
"TAPEZ ENTER";A$
```

Ce genre de détails fait la différence entre un logiciel qui fonctionne correctement, et un autre qui peut être utilisé par tout le monde.

Le président de la société américaine Microsoft (voyez le petit message en haut de l'écran), Bill Gates, a coutume de dire qu'un logiciel est commercialisable si sa vieille grand-mère est capable de l'utiliser sans problème.

Pour ce faire, il convient donc de documenter au maximum chaque phase du programme et d'utiliser au maximum des messages descriptifs au cours de son déroulement.

Mais l'amélioration majeure est représentée par la nouvelle méthode de tri introduite dans les lignes 120-160. Elle est dénommée tri à bulle.

Chaque élément de la liste est comparé tour à tour avec le suivant et ils sont permutés s'il y a lieu. Après un passage, les éléments les plus petits ont été hissés d'un rang vers le haut.

Il suffit de réitérer N fois la boucle pour que le tri soit totalement effectué. Mais le principal avantage du tri à bulle consiste en sa capacité à détecter si la liste est ordonnée avant d'avoir parcouru tout le cycle, d'où gain de temps parfois important sur la première méthode proposée.

En effet, si au cours d'une boucle aucune permutation n'a eu lieu, cela implique que la liste est triée. C'est la variable FL qui assure cette détection et la ligne 155 clôt le tri s'il y a lieu.

Le tri à bulle fait partie des méthodes les plus simples à mettre en œuvre mais, si la liste dépasse une cinquantaine d'éléments, elle s'avère trop lente. Nous n'exposerons pas ici les autres classiques car Alice n'est pas capable de contenir une liste susceptible de mettre leurs avantages en valeur.

Mais sachez que de nombreux ouvrages fort volumineux traitent uniquement cet aspect de la programmation.

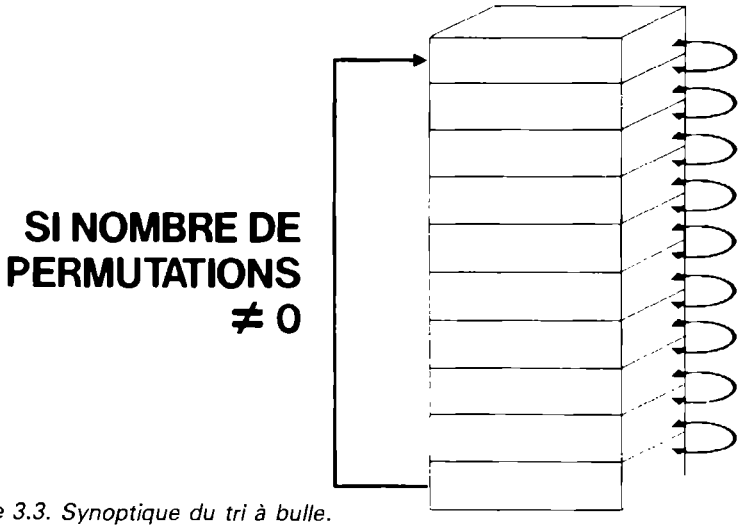


Figure 3.3. Synoptique du tri à bulle.

4
STRUCTURATION

STRUCTURATION DES PROGRAMMES

En BASIC, si on n'y prend garde, un listing peut n'être plus qu'un fouillis de tests et autres branchements anarchiques.

Une des règles de base de la programmation est de structurer ses programmes. Si cela ne pose pas de problème avec d'autres langages qui ne supportent pas d'autre méthode (Forth, Pascal...), il est souvent tentant, en BASIC, d'adopter une attitude brouillonne sous prétexte que les lignes de programme peuvent directement être rentrées sans travail préparatoire et, ma foi, si quelque chose ne va pas, il est toujours possible de rajouter quelques tests ou branchements à l'aide d'une ligne intermédiaire.

De cette façon, l'écriture d'un logiciel relativement complexe demandera trois fois plus de temps de développement que s'il avait été préalablement structuré sur papier. La recherche des erreurs en est facilitée et, plus important, celles qui sont du domaine même de la conception s'avèrent quasiment inexistantes.

Mais comment structurer ?

En fait, la méthode est simple.

Il suffit de procéder comme les généalogistes. Un programme est constitué de tâches principales, elles-mêmes subdivisées en plusieurs petits modules spécialisés, etc.

La structuration consiste à isoler chacun des composants capables de travailler de façon autonome du reste du programme. Une fois ce travail effectué, il suffit de relier entre eux chacun des modules pour obtenir le produit final.

Reprenons le tri du chapitre précédent.

Il peut se décomposer comme suit :

```
Examiner une paire.
Inverser l'ordre s'il y a lieu.
En cas d'inversion, mettre un drapeau à 1.
Examiner la paire suivante.
Recommencer le processus jusqu'en fin de liste.
Si le drapeau est à 0, le tri est terminé.
Sinon recommencer les opérations précédentes en mettant le drapeau à 0.
```

Figure 4.1. Décomposition du tri.

A ce stade, quatre-vingts pour cent du travail est fait. Il ne reste plus qu'à écrire le programme. Cette logique de programmation est universelle en informatique et, une fois bien maîtrisée, le passage d'un langage à un autre n'est que de l'apprentissage de vocabulaire et de syntaxe. Bien sûr, les spécificités du langage rentrent toujours en ligne de compte, mais leur incidence reste faible dans le cadre de la programmation structurée.

Trois outils sont à notre disposition pour structurer un programme en BASIC :

```
GOSUB
ON...GOTO
ON...GOSUB
```

NOTION DE SOUS-PROGRAMME

Un sous-programme est une partie de programme entièrement autonome qui n'est reliée avec l'extérieur que par un seul "point d'entrée". Sa tâche reste très spécialisée et, s'il a été bien pensé, il est susceptible d'être réutilisé tel quel dans un programme différent.

Un exemple simple de sous-programme serait celui qui réalise la conversion de mesure d'angle degré-radian.

Celui-ci a une grande utilité sur Alice. En effet, toutes les fonctions trigonométriques voient leur argument exprimé en radian, alors que, mis à part les mathématiciens, on a coutume d'employer les degrés.

Le sous-programme s'écrit :

```
10 PI = 3.141592654
20 AN = AN * PI/180
```

PI est une constante locale du sous-programme, et AN est la variable qui réalise la liaison entre ce sous-programme de conversion et le programme principal. Elle en constitue le seul point d'entrée.

Tel qu'il est, ce sous-programme peut être implanté dans un programme quelconque, à condition de renuméroter les lignes.

Il n'y a plus qu'à réaliser le chaînage. Nous utiliserons à cet effet les instructions GOSUB et RETURN.

Exemple

```
10 CLS
20 INPUT "VALEUR DE L'ANGLE:";
   AN
30 GOSUB 100
40 PRINT "COSINUS:";COS (AN)
50 PRINT "SINUS:";SIN (AN)
60 PRINT "TANGENTE:";TAN (AN)
70 PRINT
80 INPUT "UNE AUTRE VALEUR (O/N)
   :";A$
90 IF A$="O" THEN 20
95 END
100 PI=3.141592654
110 AN=AN*PI/180
120 RETURN
```

Mais si la valeur 90 est entrée, la ligne 60 implique une division par 0, ce qui génère un message d'erreur. Il devient donc nécessaire de prévoir l'erreur. Ceci est possible par l'adjonction d'un sous-programme.

```
10 CLS
20 INPUT "VALEUR DE L'ANGLE:";
   AN
30 GOSUB 100
40 PRINT "COSINUS:";COS (AN)
50 PRINT "SINUS:";SIN (AN)
55 IF COS(AN)< 3E-9 THEN GOSUB
   200:GOTO 70
60 PRINT "TANGENTE:";TAN (AN)
70 PRINT
80 INPUT "UNE AUTRE VALEUR (O/N)
   :";A$
90 IF A$="O" THEN 20
95 END
100 PI=3.141592654
110 AN=AN*PI/180
120 RETURN
200 PRINT "TANGENTE INDETERMINEE
   "
210 RETURN
```

Ces programmes peuvent être insérés à n'importe quel endroit d'un programme de calcul trigonométrique.

Comme vous l'avez sûrement remarqué, la séquence GOSUB RETURN possède des propriétés originales.

Si GOSUB se comporte apparemment comme GOTO, cette impression est bien vite infirmée par l'expérience. En effet, l'interpréteur est bien dérouté vers le numéro de ligne mentionné comme argument mais, dès qu'il rencontre l'instruction RETURN, il revient au point de départ et l'exécution continue normalement.

Nous en déduisons donc que GOSUB effectue le branchement désiré, mais qu'il stocke quelque part en mémoire, le point de départ du branchement. RETURN se comporte également comme un branchement, mais il va chercher une opérande dans la mémoire de sauvegarde des "adresses" de retour.

Rien ne vous empêche d'avoir plusieurs niveaux de sous-programmes.

Le processus répond alors au principe suivant :

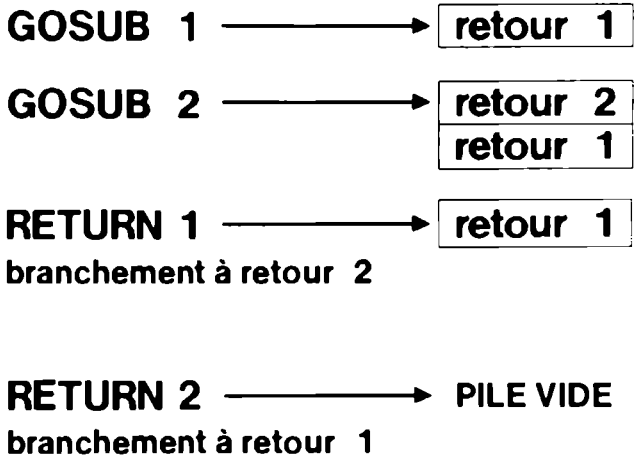


Figure 4.2. GOSUB imbriqués.

Les adresses de retour sont littéralement empilées les unes sur les autres à chaque GOSUB exécuté, ensuite, les RETURN *vont se servir sur le haut de la pile.*

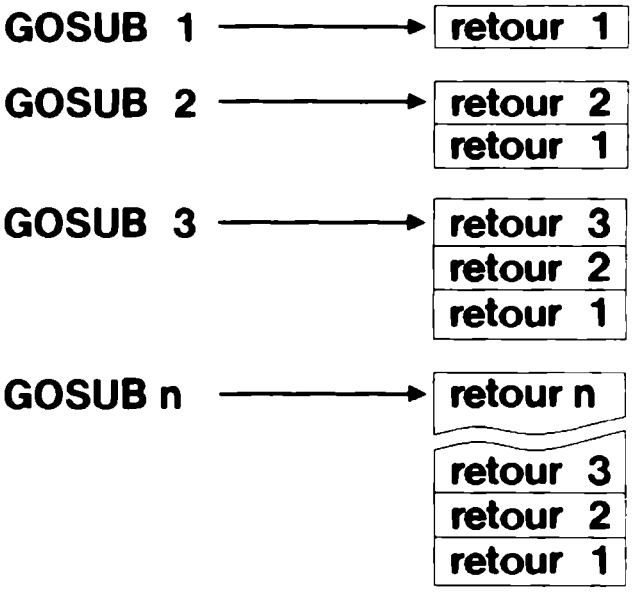


Figure 4.3. Empilement par RETURN.

Essayons le petit programme ci-dessous.

```
5 CLS
10 PRINT "TEST DE DEPASSEMENT"
20 GOSUB 100
100 PRINT"SOUS PROGRAMME ACTIF"
120 I=I+1
130 PPINT I
140 GOTO 10
150 RETURN
```

Le programme fonctionne parfaitement 422 fois, puis Alice, surchargé, signale que sa mémoire est saturée : OM ERROR.

Si vous ajoutez la ligne

```
15 REM**********************
```

la boucle ne sera parcourue que 419 fois.

Nous pouvons maintenant donner l'explication de ce résultat.

Si on analyse soigneusement le listing, on s'aperçoit que le RETURN de la ligne 150 n'est jamais exécuté. Il est "court-circuité" par l'indirection de la ligne 140.

En conséquence de quoi, les GOSUB continuent sans cesse à empiler des adresses de retour. Celles-ci occupent deux octets chacune. La mémoire d'Alice étant limitée, il n'est pas possible d'entasser ainsi des nombres sans la saturer complètement.

D'autant plus qu'une partie de la mémoire contient le programme lui-même. Si celui-ci est rallongé, la place disponible pour stocker les adresses de retour diminue d'autant.

Ce type d'erreur est parfois très difficile à mettre en évidence, et il ne faut pas chercher plus loin les raisons de bien des investigations laborieuses.

Il est donc très vivement déconseillé d'employer une indirection GOTO vers un numéro de ligne situé hors du sous-programme. Signalons toutefois que certains BASIC disposent d'une instruction POP qui efface la première adresse de retour sur l'empilement. Son usage peut éviter certains désagréments ; malheureusement Alice n'en possède pas.

MENUS

La plupart des programmes disposent de plusieurs options de traitement sélectionnables par l'utilisateur. Un moyen fort répandu pour effectuer ce choix est illustré par la notion de menu.

Les options apparaissent sur l'écran, précédées d'un numéro. Il suffit d'entrer le numéro correspondant pour accéder au traitement voulu.

Le programme ci-dessous est un exemple simple de menu :

```
10 CLS
20 PRINT:PRINT:PRINT
30 PRINT TAB(5);"1) ADDITION"
40 PRINT TAB(5);"2) SOUSTRACTION
   "
50 PRINT TAB(5);"3) MULTIPLICATI
ON"
60 PRINT TAB(5);"4) DIVISION"
70 PRINT:PRINT
80 INPUT "VOTRE CHOIX (1-4) ";C
90 IF C>4 OR C<1 THEN 10
95 CLS:INPUT "VALEURS A,B:";A,B
100 IF C=1 THEN 200
110 IF C=2 THEN 300
```

```
120 IF C=3 THEN 400
130 GOTO 500
200 CLS
210 PRINT A;"+";B;"=";A+B
220 END
300 CLS
310 PRINTA;"-";B;"=";A-B
320 END
400 CLS
410 PRINT A;"X";B;"=";A*B
420 END
500 CLS
510 PRINT A;":";B;"=";A/B
520 END
```

On obtient :

```
- >    1) ADDITION
       2) SOUSTRACTION
       3) MULTIPLICATION
       4) DIVISION
       VOTRE CHOIX (1-4) ? 3 ENTER
       - > VALEURS A,B:? 56,87 ENTER
       - > 56 x 87 = 4872
```

La ligne 90 teste si les valeurs entrées par l'utilisateur sont compatibles.

Il est toutefois possible de s'affranchir de tous les tests des lignes 100 à 130, en utilisant une seule instruction :

Remplaçons ce groupe par la ligne :

```
100 ON C GOTO 200,300,400,500
```

Le programme fonctionnera exactement de la même manière.

Selon la valeur de la variable placée entre ON et GOTO, le programme se déroulera en :

```
C=0  - >  200
C=1  - >  200
C=2  - >  300
C=3  - >  400
C=4  - >  500
C>4  - >  200
```

Il est impératif de tester la valeur de l'option avant d'exécuter un ON GOTO, sous peine de résultats incorrects.

Le ON peut tout aussi bien convenir à un branchement paramétré au sous-programme.

TRAITEMENT DES CHAINES

Le BASIC de l'Alice dispose d'instructions supplémentaires qui permettent de traiter les chaînes.

Elles peuvent être subdivisées en trois catégories :

- Conversion ASCII – > alphanumérique
  alphanumérique – > ASCII
- Conversion alphanumérique – > numérique
  numérique – > alphanumérique
- Extraction de chaînes de caractères.

CONVERSIONS

Illustrons d'abord les conversions.

```
10 CLS
20 FOR I=1 TO 255
30 PRINT I;" ";CHR$(I);
40 IF I/10=INT(I/10) THEN INPUT
   "TAPEZ ENTER";A$
50 PRINT
60 NEXT I
```

Les codes ASCII de tous les caractères sont affichés, ainsi que les caractères correspondants.

CHR$ convertit un code ASCII en son symbole associé. On remarque que les trente-deux premiers codes ne génèrent pas grand-chose mis à part le code 13 qui est équivalent à la frappe de la touche ENTER. Les codes sont tous compris entre 0 et 255.

```
10 CLS
20 A$=INKEY$
30 IF A$="" THEN 20
40 PRINT ASC(A$);" ";A$
50 GOTO 20
```

Inversement, ASC fournit le code ASCII du premier caractère d'une chaîne. L'exemple ci-dessus imprime les codes des touches du clavier enfoncées, ainsi que le symbole associé.

Une autre conversion fondamentale est celle qui permet de transférer des données d'une variable numérique dans une variable chaîne de caractères, et vice versa.

C'est le rôle de STR$ et VAL.

Le principal problème rencontré avec STR$ provient du format des nombres.

Souvenez-vous qu'un nombre lorsqu'il est imprimé est accompagné par un espace (si le nombre est positif) réservé au signe – s'il y a lieu.

Lors de la conversion en chaîne de caractères, celui-ci est pris en compte.

```
10 A$="23"
20 A=23
   B$=STR$(A)
   PRINT A$
   PRINT B$

23
 23
```

On obtient :

```
B$= "  " + A$
```

Ce petit détail risque de vous réserver certaines surprises désagréables si vous l'oubliez.

L'instruction VAL en est la réciproque, mais attention, "E" et "." sont considérés comme partie prenante d'un nombre. Si lors de la conversion un symbole non numérique est rencontré, la conversion est terminée. Si le premier symbole est non numérique, le résultat est un 0. Ceci peut être exploité dans un test. Nous pouvons savoir, par exemple, si l'ordinateur interprète une donnée comme un nombre ou du texte.

```
' CLS
10 INPUT"TEST";A$
20 IF VAL (A$) THEN PRINT
   "NUMERIQUE":GOTO 50
30 PRINT "ALPHANUMERIQUE"
50 END
```

On obtient :

```
- >    TEST? POLI ENTER
       ALPHANUMÉRIQUE

- >    TEST? 23ADE ENTER
       NUMÉRIQUE
```

EXTRACTION DE CHAÎNE

La manipulation de chaînes de caractères est un des domaines les plus spectaculaires de la micro-informatique. Avec un petit peu d'imagination, il est possible de créer des programmes extrêmement intéressants.

Ces fonctions sont indispensables et l'éventail de leur utilisation va du traitement d'erreur au traitement de texte en passant par les routines de conversions mathématiques.

Toutes les options existent en ce domaine :

- extraction de la partie droite (RIGHT$) ;
- extraction de la partie gauche (LEFT$) ;
- extraction quelconque (MID$).

Lors de l'entrée d'une donnée en cours de programme, il faut la tester pour savoir si l'utilisateur n'a pas fait d'erreur. Il est important avec un INPUT de ne pas entrer une donnée alphanumérique lorsque le programme attend un nombre. La solution la plus sûre est de faire appel à une variable alphanumérique pour le stockage, quitte à faire la conversion ultérieurement.

```
10 CLS
20 INPUT "NOMBRE ENTRE 1 ET 4";A$
30 IF VAL(A$) < 1 OR VAL(A$) > 4 THEN PRINT "TRICHEUR"
40 END
```

Le programme est maintenant sûr.

RÉALISATION

Testons nos nouvelles acquisitions en élaborant un programme illustrant ces caractéristiques.

Il s'agit, en l'occurrence, d'une version numérique du célèbre mastermind.

Analysons d'abord le programme : cherchons ses différents sous-ensembles.

- GÉNÉRATION D'UN NOMBRE ALÉATOIRE DE 6 CHIFFRES
- CONVERSION EN CHAINE DE CARACTÈRES
- AFFECTATION DE CHACUN DES CARACTÈRES DANS UN TABLEAU POUR TEST AVEC BOUCLE
- ENTRÉE DU NOMBRE
- COMPTEUR DE SCORE
- TEST DES CHIFFRES EN BONNE PLACE
- TEST DES CHIFFRES DÉSORDONNÉS
- AFFICHAGE DES CARACTÉRISTIQUES
- TEST DE GAIN OU DE PERTE DE LA PARTIE

La phase finale consiste à écrire le programme BASIC. La seule difficulté réelle réside dans le choix de la méthode de test sur l'ordre ou le désordre des chiffres. La solution proposée ici a le mérite d'être anthropomorphique.

Dans un premier temps, chacun des caractères de la donnée entrée est comparé avec son équivalent dans le tableau NO$. Chaque fois qu'il y a égalité, une variable OD est incrémentée et l'élément correspondant d'un tableau de test FL est mis à 1.

Nous connaissons alors le nombre de caractères dans le bon ordre.

Reste à déterminer s'il en existe d'autres, communs à la donnée entrée et au nombre aléatoire.

Il suffit de comparer un à un :

- Les chiffres qui n'ont pas le même indice dans le tableau.
- Ceux dont la valeur test de même indice est à 0.

Le reste du programme n'est que de la mise en page. Il est vrai qu'en général le traitement complet dépasse rarement la moitié de la taille du programme. La présentation étant souvent la partie la plus fastidieuse à écrire. Mais la qualité du logiciel en dépend grandement.

```
10 CLS
20 CO=0
30 A=RND(999999)
35 A$=RIGHT$ (STR$(A),6)
```

```
40 PRINTA;" ";
50 PRINT@40," "
70 FOR I=1 TO 6
80 NO$(I)=MID$(A$,I,1)
90 FL(I)=0
100 NEXT I
110 PRINT@463,"        "
120 PRINT@450," ";
130 INPUT "VOTRE NOMBRE:";B$
140 IF VAL (B$)=0 THEN 130
150 CO=CO+1
160 OD=0:DE=0
170 FOR I= 1 TO 6
180 IF MID$(B$,I,1)=NO$(I) THEN
    FL(I)=1:OD=OD+1
190 NEXT I
200 FOR I=1 TO 6
210 FOR J=1 TO 6
220 IF MID$(B$,I,1)=NO$(J) AND
    FL(J)=0 THEN FL(J)=1:DE=DE+1
    :GOTO 240
230 NEXT J
240 NEXT I
250 PRINT@96+32*CO," ";
260 FOR I= 1 TO 6
270 PRINT MID$(B$,I,1);" ";
280 FL(I)=0
290 NEXT I
300 IF OD <> 6 THEN 350
310 PRINT@480,"GAGNE        ";
320 CLEAR
330 INPUT A$
340 GOTO 10
350 PRINT@110+32*CO,"O:";OD;" D:
    ";DE;
360 IF CO < 10 THEN 400
370 PRINT@480," PERDU     ";
380 PRINT@32," ";
385 FOR I=1 TO 6
390 PRINT NO$(I);" ";
395 NEXT I
397 INPUT A$   GOTO 10
400 GOTO 110
```

Voici une liste des variables et des sous-ensembles du *Mastermind.*

20	Compteur à 0.
30	Choix du nombre aléatoire. (voir Annexe 3 pour RND).
35	Elimination du blanc à l'extrême-gauche.
40	Affichage du nombre. Cette ligne n'est nécessaire que pour la mise au point. Il faut l'effacer pour jouer.
70-100	Extraction des chiffres et stockage dans le tableau comparatif.
130-140	Entrée de la donnée du joueur.
170-190	Test des chiffres dans l'ordre.
200-240	Test des chiffres dans le désordre.
250-290	Impression des caractéristiques du coup.
310-340	Partie gagnée.
350-400	Partie perdue.

CO	Score.
A	Nombre aléatoire.
A$	Stockage de A après conversion.
NO$	Tableau comparatif.
FL	Tableau de stockage des tests.
OD	Nombre de chiffres dans l'ordre.
DE	Chiffres dans le désordre.
B$	Donnée du joueur.

FICHIERS INTERNES

Il est parfois pratique de pouvoir stocker certaines données dans un programme BASIC. Lorsque le programme est sauvegardé, les variables sont remises à zéro. Une solution simple consisterait à commencer un programme par le remplissage d'un tableau. Si cette méthode convient avec trois données, il vous faudra bien du courage pour en introduire cinquante.

Si le nombre de données à stocker reste raisonnable, le BASIC permet de les incorporer dans un *fichier* interne. Elles sont au cœur du programme.

Lors de son évaluation par l'interpréteur, le programme va être subdivisé en deux parties distinctes :

- les instructions et leurs opérandes ;
- les données du fichier interne.

Les DATA peuvent être réparties avec des numéros quelconques. Mais leur ordre propre en dépendra.

```
  10 DATA MACHINE,FRANCE,OVNI
 112 DATA RESTAURANT,45,"56",FILM
1230 DATA PARANOIAQUE,MÉGALOMANE
```

Ces lignes de DATA seront regroupées séquentiellement, conformément au schéma suivant :

```
MACHINE
FRANCE
OVNI
RESTAURANT
45
56 (alphanumérique)
PARANOIAQUE
MÉGALOMANE
```

La lecture des données est assurée par READ suivi d'une variable réceptacle. Il faut bien évidemment respecter les règles régissant l'affectation des données à une variable.

La lecture elle-même n'est pas très souple. Il existe un élément de mémoire qui fait office de "flèche" vers la prochaine donnée lue par READ. Cette petite mémoire est le *pointeur des DATA*. Après chaque instruction READ, ce pointeur désigne le mot suivant dans la liste. Lorsque celle-ci a été épuisée, il faut remettre le pointeur en début de fichier interne. Ceci est réalisé par RESTORE.

```
10 DATA FLECHE,MOT,SIGNE
```

Le fichier aura cette forme.

```
pointeur - - >    FLÈCHE
                  MOT
                  SIGNE
              READ A$ A$ contient "FLÈCHE"
                  FLÈCHE
```

```
pointeur - - >     MOT
                   SIGNE
                 READ A$ A$ contient "MOT"
                   FLÈCHE
                   MOT
pointeur - - >     SIGNE
                 READ A$ A$ contient "SIGNE"
```

Le pointeur est positionné hors du fichier

```
                   RESTORE
pointeur - - >     FLÈCHE
                   MOT
                   SIGNE
```

```
5 CLS
10 DATA FRANCE,ECOSSE,SENEGAL
20 DATA MALI,RFA,BURUNDI
30 FOR I=1 TO 6
40 READ A$
50 PRINT A$
60 NEXT I

FRANCE
ECOSSE
SENEGAL
MALI
RFA
BURUNDI
```

Pour lire le énième élément d'un fichier DATA, il faut lire tous ceux qui le précèdent. Pour ce faire, le moyen le plus commode est d'introduire une boucle de 1 à N.

FICHIERS D'ADRESSES

Les fichiers internes ne sont qu'un expédient commode, utilisables dans certains cas très particuliers.

Mais l'informatique fait souvent appel à la notion de gestion de fichier.

Ce rôle est dévolu à des programmes relativement complexes dont la structure est issue des recherches théoriques en ce domaine. La gestion d'un fichier est entièrement tributaire de la mémoire de masse disponible, disque dur, disque souple ou cassette.

Il ne faut pas se faire d'illusion, mais un appareil comme Alice ne peut être comparé à un APPLE II ou un IBM PC. Nous ne verrons donc pas ou peu les structures générales de ces programmes professionnels. Mais il est tout de même possible de réaliser un petit logiciel efficace mais limité par la taille de la mémoire d'Alice et surtout par la sauvegarde magnétophone.

FONCTIONNEMENT DE LA SAUVEGARDE

Le processus de sauvegarde est relativement simple et fait appel à des principes désormais classiques.

Comme dans le cas d'une chaîne stéréophonique, un signal électrique sous forme d'impulsions est envoyé sur le périphérique de sauvegarde. La tête de lecture/écriture se comporte comme une petite bobine et va engendrer un champ magnétique qui perturbe l'ordonnancement des particules ferromagnétiques de la surface de la bande. L'information sera donc présente sous la forme d'un arrangement particulier des molécules d'oxyde de fer.

La lecture, elle, suit le même protocole, mais en sens inverse cette fois.

La bande, en défilant sous la tête, va induire de petites impulsions électriques qui seront décodées par l'ordinateur.

L'information a été restituée. En informatique, les signaux à traiter sont beaucoup moins complexes que le son de la *Cinquième* de Beethoven. Il est préférable d'opter pour un magnétophone standard et des bandes standards (sauf pour le MC 10).

En effet, les amplificateurs très performants font subir un tel nombre de contraintes au signal qu'une simple donnée informatique peut être perdue car considérée comme un parasite.

Voici pour la technologie, mais qu'en est-il du côté de l'ordinateur ?

Lors de la sauvegarde, celui-ci écrit un en-tête au programme ; le HEADER. Il sert à reconnaître ultérieurement le niveau sonore et la gamme de fréquences sous lesquels le programme a été stocké. L'ordinateur peut ainsi ajuster, dans une faible mesure, les paramètres qui lui permettront de relire la bande.

A la fin du HEADER, certaines informations en rapport avec le programme sont incorporées ; le nom (6 caractères au maximum), le type de données (programme ou tableau). L'ensemble du fichier est terminé par une séquence codée signalant la fin du programme.

Le BASIC offre à l'utilisateur le contrôle de cette entrée/sortie (voir Annexe 3 CSAVE,CLOAD...)

FICHIER INFORMATIQUE

Un gestionnaire de fichier fonctionne avec des principes identiques à ceux des fichiers des anciens fonctionnaires.

Imaginez une grande boîte comprenant un certain nombre de feuilles cartonnées. Celles-ci sont rangées dans l'ordre alphabétique des noms de famille par exemple. Le fonctionnaire peut rapidement trouver une fiche précise, l'exploiter, et la remettre à sa place.

Un fichier utilise la même méthode de base. Mais les fiches sont un ensemble de cases spécialisées réservées pour chacune d'entre elles.

Il existe deux possibilités de stockage :

- Soit le fichier est sur la mémoire de masse, auquel cas la recherche a lieu dans la mémoire RAM, puis la fiche est chargée pour une opération ultérieure.
- Soit toutes les fiches sont présentes dans la RAM.

La première méthode exige une mémoire de masse à accès rapide, disques souples ou durs. Elle est donc exclue dans le cas d'Alice.

Malheureusement, le second cas de figure est grand consommateur de mémoire centrale.

La taille du fichier est donc directement limitée par celle de la RAM de l'unité centrale.

Le carnet d'adresses que nous allons fabriquer sera limité à dix fiches. Son extension ne pose aucune difficulté de programmation.

Examinons les différents sous-ensembles du carnet d'adresses.

1. *Le masque de saisie*

Il représente le format de la fiche avec les différentes rubriques qu'elle comporte. Bien sûr, il suffit de stocker les informations et non le format. Il est l'équivalent du formulaire de la Sécurité sociale que vous avez rempli la semaine dernière, après avoir soigné votre rage de dents.

2. *L'éditeur*

Ce sous-programme gère l'entrée des données de telle sorte que le curseur apparaisse face aux rubriques concernées.

3. *Les modules de travail*

Chacun d'entre eux représente une phase de travail.

- Edition et mise à jour d'une fiche.
- Recherche d'une fiche.
- Listage sur imprimante.
- Eventuellement tris divers.

4. *La sauvegarde magnétophone*

Voici le listing du programme de gestion de carnet d'adresses.

```
10 CLS
20 AD$(0,1)="1"
40 GOTO 1000
45 CLS
50 PRINT:PRINT"NOM:"
60 PRINT:PRINT"PRENOM:"
70 PRINT:PRINT"ADRESSE:"
80 PRINT:PRINT"VILLE:"
90 PRINT:PRINT"CODE:"
100 PRINT:PRINT"TEL:"
110 RETURN
200 FOR K=1 TO 11 STEP 2
210 PRINT@32*K+12,"--------------
------";
220 NEXT K
230 RETURN
250 RU$=AD$(IN,R)
260 PRINT@32*V+12,"^";
270 A$=INKEY$
280 IF A$="" THEN 270
290 IF A$=CHR$(13) OR H=31 THEN
    PRINT@32*V+H,"-";:PRINT@32*V
    +12,RU$;:RETURN
295 IF A$=CHR$(9) THEN H=12+LEN
    (RU$):PRINT@32*V+12,RU$;
    GOTO 360
300 IF A$="@" THEN R=0:RETURN
310 IF A$="^" THEN FL=1 :RETURN
```

```
320 IF A$<>CHR$(8) THEN 350
330 H=H-1:IF H<12 THEN H=12
335 RU$=LEFT$(RU$,H-12)
340 PRINT@32*V+H+1,"-";
345 GOTO 360
350 H=H+1
355 RU$=RU$+A$:PRINT@32*V+12,RU$
    ;
360 PRINT@32*V+H,"^";
370 GOTO 270
400 R=1
405 CLS:GOSUB 50
410 GOSUB 200
420 V=R*2-1
430 H=12
440 GOSUB 250
450 IF R=8 THEN AD$(IN,0)="1":
    RETURN
460 IF FL=1 THEN FL=0:RETURN
470 AD$(IN,R)=RU$
480 R=R+1:IFR>6 THENR=1
490 GOTO 420
500 FOR I= 1 TO VAL(AD$(0,1))
510 IF VAL (AD$(I,0))=0 THEN IN=
    I:GOTO 550
520 NEXT I
530 AD$(0,1)=STR$(VAL(AD$(0,1))+
    1):IN =VAL(AD$(0,1))
550 GOSUB 400
560 GOTO 1000
600 FOR I=0 TO VAL(AD$(0,1))
610 IF AD$(I+1,0)="0" THEN 660
620 FOR J=1 TO 6
630 PRINT@32*2*(J-1)+44,AD$(I+1,
    J)
640 NEXT J
645 A$=INKEY$:IF A$="" THEN 645
650 IF A$="E" THEN IN=I+1:GOTO
    550
655 IF A$="S" THEN IN=I+1:GOTO
    700
660 NEXT I
```

```
670 GOTO 1000
700 FOR I=1 TO 6
710 AD$(IN,I)=""
720 NEXT I
730 AD$(IN,0)="0"
740 GOTO 1000
800 CLS
810 PRINT"     RECHERCHE "
820 PRINT:PRINT
830 PRINT "1) PAR NOM"
840 PRINT"2) PAR PRENOM"
843 PRINT"3) PAR ADRESSE"
845 PRINT"4) PAR VILLE"
847 PRINT"5) PAR CODE"
849 PRINT"6) PAR TEL"
850 INPUT"VOTRE OPTION";A$
860 PRINT:INPUT"CRITERE";B$
870 FOR I=1 TO VAL (AD$(0,1))
880 IF AD$(I,VAL(A$))<> B$ THEN
    950
890 CLS : GOSUB 50
900 FOR J=1 TO 6
910 PRINT@32*2*(J-1)+44,AD$(I,J)
920 NEXT J
930 C$=INKEY$:IF C$="" THEN 930
940 IF C$="E" THEN IN=I:GOTO 550
945 IF C$="S" THEN IN=I:GOTO
    700
950 NEXT I
960 GOTO 1000
1000 CLS
1010 PRINT:PRINT"1) EDITION"
1020 PRINT:PRINT"2) SOS"
1030 PRINT:PRINT"3) RECHERCHE"
1040 PRINT:PRINT"4) LISTAGE"
1070 PRINT:PRINT
1080 PRINT"VOTRE OPTION: ";
1090 A$=INKEY$:IF A$="" OR
     A$<"1" OR A$>"4" THEN 1090
1100 MO=VAL(A$)
1110 PRINT MO
1115 FOR I=1 TO 200:NEXT I
```

```
1120 IF MO=4 THEN CLS:GOSUB50:
     GOTO 600
1500 ON MO GOTO 500,1800,800
1510 GOTO 1000
1800 CLS
1810 PRINT:PRINT"E) EDITE"
1820 PRINT"S) SUPPRIME"
1830 PRINT">) FIN DE MOT"
1840 PRINT"@) VALIDATION FICHE"
1850 PRINT"<) RETOUR CARACTERE"
1870 FOR I=1 TO 2500:NEXT I
1880 GOTO 1000
```

Comme vous l'avez sans doute remarqué, les tableaux ne sont pas dimensionnés, le nombre de fiches utilisables est donc limité à dix.

Pour pallier cet inconvénient, l'auteur vous laisse le soin d'introduire une instruction DIM judicieusement paramétrée.

La structuration est ici réalisée pour une part égale par des GOTO et des GOSUB. L'éventail des fonctions utilisées est particulièrement vaste, bien que nous ayons omis la sauvegarde sur la mémoire de masse.

Ce point peut être aisément réglé avec CSAVE* et CLOAD* (voir Annexe 3).

Il suffirait de greffer les sous-programmes de tri du chapitre précédent pour remplir complètement le cahier des charges que nous nous sommes fixé.

Vous vous rendrez vite compte que la programmation en elle-même n'est pas très difficile ; en revanche, la détermination des modalités d'emploi peut très vite devenir un véritable casse-tête.

Pour notre part, l'accès aux différentes phases de travail se fait de façon mixte par menus ou par commandes intégrées dans chaque module.

Quatre options sont disponibles à la mise en route.

1) ÉDITION
2) SOS
3) RECHERCHE
4) LISTAGE

OPTION 3

Un second menu décrit les possibilités du module.

RECHERCHE
1) PAR NOM
2) PAR PRÉNOM
3) PAR ADRESSE
4) PAR VILLE
5) PAR CODE
6) PAR TÉL

Il suffit de donner le critère de choix et sa valeur pour que la fiche, si elle existe apparaisse sur l'écran.

A ce niveau plusieurs traitements peuvent être réalisés :

• Affichage des fiches suivantes, s'il y a lieu, la frappe d'une touche quelconque, à l'exception de E,S et BREAK, implique ce choix.

• Mise à jour de la fiche par la touche E pour entrer dans l'éditeur.

• Suppression de la fiche par la touche S. La prochaine fiche créée prendra en priorité la place vacante.

OPTION 4

Elle provoque le défilement de toutes les fiches présentes en mémoire. Les commandes intégrées sont les mêmes que précédemment. L'ordre d'apparition à l'écran est l'ordre chronologique d'entrée des fiches sauf dans le cas où l'une d'entre elles a été effacée et une autre créée.

Le changement du PRINT de la ligne 630 en LPRINT entraînerait la création d'un listing sur imprimante.

OPTION 2

Un message d'aide décrivant succinctement les différentes commandes intégrées apparaît quelques secondes sur l'écran.

OPTION 1

C'est l'option maîtresse du programme. Elle contient l'éditeur, le masque de saisie et le gestionnaire du tableau contenant les informations.

L'éditeur est plus performant que celui qui équipe Alice d'origine.

Le curseur vient se placer en face de chaque rubrique à remplir.

La flèche à droite le positionne à la fin de la donnée inscrite ; la flèche à gauche efface un caractère de la donnée.

Enfin, ENTER permet de passer à la rubrique suivante. S'il s'agit de la sixième, le curseur se positionnera en début de fiche.

L'incorporation de la fiche est dû à la frappe de la flèche en haut ou du "a commercial".

Toutes les fiches sont stockées dans un tableau AD$ à deux dimensions.

Sa première rubrique contient le nombre de fiches enregistrées.

Le premier élément de chaque fiche est en fait un état d'occupation. S'il est nul, la place est disponible, autrement, il est à 1.

Détail du programme

— 1000 Point d'entrée du menu principal.
— 500 Point d'entrée du module d'édition.
— 600 Point d'entrée du module de listage.
— 800 Point d'entrée du module de recherche.
— 1800 Point d'entrée du module d'aide.
— 50 Point d'entrée du module de masque de saisie.
— 250 Point d'entrée de l'éditeur.
— 400 Point d'entrée du module de gestion.

Bien vite, les limites de la capacité mémoire d'Alice sont atteintes.

Tout commence par un message OS ERROR.

La solution temporaire consiste à débuter le programme avec une instruction CLEAR (voir Annexe 3).

La complémentation du programme, ainsi que sa capacité à traiter un nombre de fiches correct, passent obligatoirement par l'acquisition d'une extension mémoire 16K.

5
MATHÉMATIQUES

Ce chapitre est destiné à la manipulation des fonctions mathématiques que nous avons survolées dans le Chapitre 2.

Quelques exemples relativement simples serviront à les illustrer. Mais vous constaterez très vite que l'écriture de programmes mathématiques est très aisée. En effet, le BASIC travaille exactement comme nous le ferions sur une feuille de papier.

ÉQUATION DU SECOND DEGRÉ

Nous ne développerons pas ici les tenants et les aboutissants de la théorie mathématique des équations du second degré ou polynômes de degré 2.

Il faut simplement savoir qu'il existe une méthode efficace de calcul des solutions. A titre indicatif, mentionnons le fait que les équations du troisième degré peuvent également être résolues exactement grâce à la méthode de CARDAN, mais aucune méthode générale n'est valable pour les polynômes de degré supérieur.

```
10 CLS
20 PRINT "COEFFICIENTS A,B,C"
30 PRINT "DE L'EQUATION"
40 PRINT "  2     "
50 PRINT "AX + BX + C = 0"
60 INPUT "A=";A
70 INPUT "B=";B
80 INPUT "C=";C
90 CLS
100 PRINT A;"X^2+";B;"X+";C;"=0"
110 PRINT
120 PRINT"^ SIGNIFIE PUISSANCE"
130 PRINT:PRINT
140 INPUT "CORRECT (O/N)";A$
150 IF A$<>"O" THEN 10
160 CLS
170 PRINT:PRINT:PRINT:PRINT
180 PRINT TAB(5);
    "CALCUL EN COURS"
190 REM***DISCRIMINANT****
200 DE= B*B-4*A*C
210 IF DE < 0 THEN 300
220 X1= (-B-SQR(DE))/(2*A)
```

```
230 X2= (-B+SQR(DE))/(2*A)
250 PRINT
260 PRINT
270 CLS :PRINT:PRINT
285 IF DE=0 THEN PRINT TAB(5);
    "RACINES DOUBLES"
295 PRINT "X1=";X1:PRINT "X2=";
    X2
298 INPUT A$:GOTO 10
300 X1= -B/(2*A)
310 Y1= SQR(ABS(DE))/2*A
320 CLS
330 PRINT:PRINT
340 PRINT TAB(5);" ";
350 PRINT "RACINES COMPLEXES"
360 PRINT:PRINT
370 PRINT "X1=";
380 PRINT X1;"+    *";Y1
390 PRINT "X2=";
400 PRINT X1;"-  ) *";Y1
410 PRINT:PRINT
420 INPUT A$:GOTO 10
```

Ce programme calcule également les valeurs complexes des racines dans le cas où le discriminant est négatif.

On obtient :

```
- >   COEFFICIENTS A,B,C
      DE L'ÉQUATION
      2
      AX + BX + C = 0
      A=? 4.6  ENTER
      B=? -32 ENTER
      C=? -7  ENTER
- >   4.6×↑2+ -32× -7
      ↑ SIGNIFIE PUISSANCE
      CORRECT (O/N)?O ENTER
- >   ×1= -.212272671
      ×2=7.16879441
```

La présentation de l'équation peut être plus soignée en employant les fonctions de traitement de chaînes de caractères et en testant la valeur du signe des constantes A,B,C.

RÉSOLUTION D'ÉQUATIONS PAR ITÉRATIONS

Pour les équations de degré supérieur, la méthode dite par *itérations* est bien adaptée à l'ordinateur.

Elle est basée sur le fait qu'une valeur particulière est calculée à l'aide de la formule générale.

Cette valeur est ensuite réinjectée dans la formule, puis le processus recommence.

Au bout d'un certain nombre de boucles, la valeur calculée ne change quasiment plus. Cette valeur est la solution approchée de l'équation.

Le programme proposé peut résoudre des polynômes de degré dix.

```
10 CLS
20 INPUT "DEGRE DE L'EQUATION";D
30 CLS
40 FOR I=D TO 0 STEP -1
50 PRINT "X^";I;"=";
60 INPUT IN(I)
80 NEXT I
90 CLS
100 FOR I=D TO 0 STEP -1
103 IN$(I)=STR$(IN(I))
105 IF LEFT$(IN$(I),1)=" " THEN
    IN$(I)="+"+ RIGHT$(IN$(I),
    LEN (IN$(I))-1)
110 PRINT IN$(I);"(X ^";I;")";
120 NEXT I
130 PRINT "=0"
140 PRINT:PRINT
150 INPUT "CORRECT (O/N)";A$
160 IF A$<>"O" THEN 10
180 CO=1:X=1
185 X1=0
190 FOR I=0 TO D-1
200 X1=X1+IN(I)*X^I
210 NEXT I
220 X1=-X1/IN(D)
225 X1=SGN(X1)^D*(ABS(X1)^(1/D))
```

```
230 PRINT CO,X1
235 CO=CO+1
240 IF ABS(X-X1)<1E-8 THEN 300
250 X=X1
260 GOTO 185
300 REM
310 PRINT:PRINT
320 PRINT X1
```

En choisissant un polynôme de degré 4 avec comme coefficients – 5, 8, 8, 31, 3; trente-quatre itérations sont nécessaires pour parvenir au résultat.

```
-5(X    4  )+8(X    3  )+8(X    2  )+3
1(X    1  )+3(X ^ 0  )
I                        N
-------------------------------------
 1                    1.77827941
 2                    2.25117399
 3                    2.52919309
 4                    2.69052414
 5                    2.78345804
 6                    2.83676526
 7                    2.86726765
 8                    2.88469659
 9                    2.89464739
 10                   2.90032603
 11                   2.90356583
 12                   2.90541393
 13                   2.90646806
 14                   2.9070693
 15                   2.90741221
 16                   2.90760778
 17                   2.90771932
 18                   2.90778294
 19                   2.90781922
 20                   2.90783991
 21                   2.90785172
 22                   2.90785845
 23                   2.90786228
 24                   2.90786447
 25                   2.90786572
 26                   2.90786644
```

```
27              2.90786684
28              2.90786707
29              2.9078672
30              2.90786728
31              2.90786732
32              2.90786735
33              2.90786736
34              2.90786737
```

La suite des nombres ne convergera pas toujours à la même vitesse. Si le programme est trop long, il est possible de diminuer la précision du résultat en modifiant la ligne 240, et de limiter le nombre d'itérations en fixant une valeur limite à CO.

```
245 IF CO > 20 THEN PRINT:PRINT "ITERATION STOPPEE":
GOTO 300
```

GRAPHE D'UN POLYNÔME QUELCONQUE DE DEGRÉ 3

L'analyse mathématique est très souvent représentée par des graphiques en tout genre. En effet, il est plus pratique de voir l'allure générale d'une courbe qu'un simple tableau de valeurs, infiniment moins expressif.

Les possesseurs d'une imprimante peuvent exploiter leur périphérique avec efficacité.

En effet, la taille de l'écran étant réduite, il est beaucoup plus logique d'imprimer le graphe sur une bande de papier qui n'est pas limitée en longueur. Bien sûr, l'amplitude des ordonnées doit être circonscrite dans l'intervalle de trente-deux caractères équivalent à une ligne du téléviseur.

La largeur de la bande représentera donc les ordonnées et la longueur, les abscisses.

La courbe ne sera représentative qu'à la condition expresse qu'un nombre suffisant de points ait été calculé.

Pour notre part, 32 est un chiffre suffisant pour de nombreuses fonctions mathématiques. Mais vous verrez qu'il est extrêmement simple d'affiner le calcul.

La méthode employée n'est pas véritablement complexe, mais elle requiert un soin extrême lors de son écriture, sous peine de générer des graphiques pour le moins curieux.

Le programme se décompose comme suit :

- Entrée de l'intervalle de travail.
- Découpage de cet espace en trente-deux valeurs.
- Calcul de l'ordonnée correspondant à chacune d'entre elles.
- Détermination des valeurs maximale et minimale.
- Impression des X et des Y.
- Calcul des tabulations en fonction de l'amplitude des Y et de la valeur traitée.
- Impression d'un point sous la forme d'un astérisque.

Le graphe est subdivisé en deux parties correspondant aux ordonnées positives et négatives. Le fait d'utiliser des rapports dans le calcul de l'argument de tabulation indique bien que la représentation de la courbe est optimisée en fonction de la largeur de la bande de papier.

Les premières fonctions analysées sont des polynômes de degré inférieur à 4.

```
5 DIM Y(32)
10 CLS
15 LPRINT "      X","      Y"
16 LPRINT"-----------------------
----------"
17 LPRINT
20 INPUT "INTERVAL SUP,INF";S,IM
30 FOR I=3 TO 0 STEP -1
40 PRINT "COEFF DE 'X^";RIGHT$
   (STR$(I),1)
45 INPUT A(I)
50 NEXT I
55 K=0
60 FOR I=IM TO S STEP (S-IM)/32
65 Y(K)=0
70 FOR J=0 TO 3
80 Y(K)=Y(K)+I^J*A(J)
90 NEXT J
95 IF I=IM THEN MN=Y(K):MY=Y(K)
97 K$=STR$(K):K$=RIGHT$(K$,LEN(
   .K$)-1)
98 IF LEN(K$)<2 THEN K$="0"+K$
99 LPRINT K$;":";I,Y(K)
100 IF Y(K)>MY THEN MY=Y(K)
110 IF Y(K)<MN THEN MN=Y(K)
115 K=K+1
120 NEXT I
150 LPRINT
160 LPRINT"  -------------------
-----------"
170 FOR I=1 TO 32
180 LPRINT I;":";
190 H=15+INT(15/(MY-MN)*Y(I))
200 LPRINT TAB(H);"*"
210 NEXT I
```

```
INTERVAL -5,+5
FONCTION 2X^3+2X^2+2

      X                    Y
-----------------------------------

00:-5                  -198
01:-4.6875             -160.04834
02:-4.375              -127.199219
03:-4.0625             -99.0864258
04:-3.75               -75.34375
05:-3.4375             -55.6049805
06:-3.125              -39.5039062
07:-2.8125             -26.6743164
08:-2.5                -16.75
09:-2.1875             -9.36474609
10:-1.875              -4.15234375
11:-1.5625             -.746582031
12:-1.25                1.21875
13:-.9375               2.10986328
14:-.625                2.29296875
15:-.3125               2.13427734
16: 0                   2
17: .3125               2.25634766
18: .625                3.26953125
19: .9375               5.40576172
20: 1.25                9.03125
21: 1.5625              14.5122070
22: 1.875               22.2148438
23: 2.1875              32.5053711
24: 2.5                 45.7500001
25: 2.8125              62.3149414
26: 3.125               82.5664063
27: 3.4375              106.870605
28: 3.75                135.59375
29: 4.0625              169.102051
30: 4.375               207.761719
31: 4.6875              251.988965
32: 5                   302
```

```
INTERVAL -1,1
FONCTION X^2-3X+1
      X                    Y
-----------------------------------

00:-1                      5
01:-.9375                  4.69140625
02:-.875                   4.390625
03:-.8125                  4.09765625
04:-.75                    3.8125
05:-.6875                  3.53515626
06:-.625                   3.265625
07:-.5625                  3.00390625
08:-.5                     2.75
09:-.4375                  2.50390625
10:-.375                   2.265625
11:-.3125                  2.03515625
12:-.25                    1.8125
13:-.1875                  1.59765625
14:-.125                   1.390625
15:-.0625                  1.19140625
16: 0                      1
17: .0625                  .81640625
18: .125                   .640625
19: .1875                  .47265625
20: .25                    .3125
21: .3125                  .16015625
22: .375                   .015625
23: .4375                 -.12109375
24: .5                    -.250000001
25: .5625                 -.37109375
26: .625                  -.484375
27: .6875                 -.589843752
28: .75                   -.687500001
29: .8125                 -.777343752
30: .875                  -.859375
31: .9375                 -.933593751
32: 1                     -1
```

Les deux fonctions particulières donnent une bonne idée de l'intérêt de la représentation graphique.

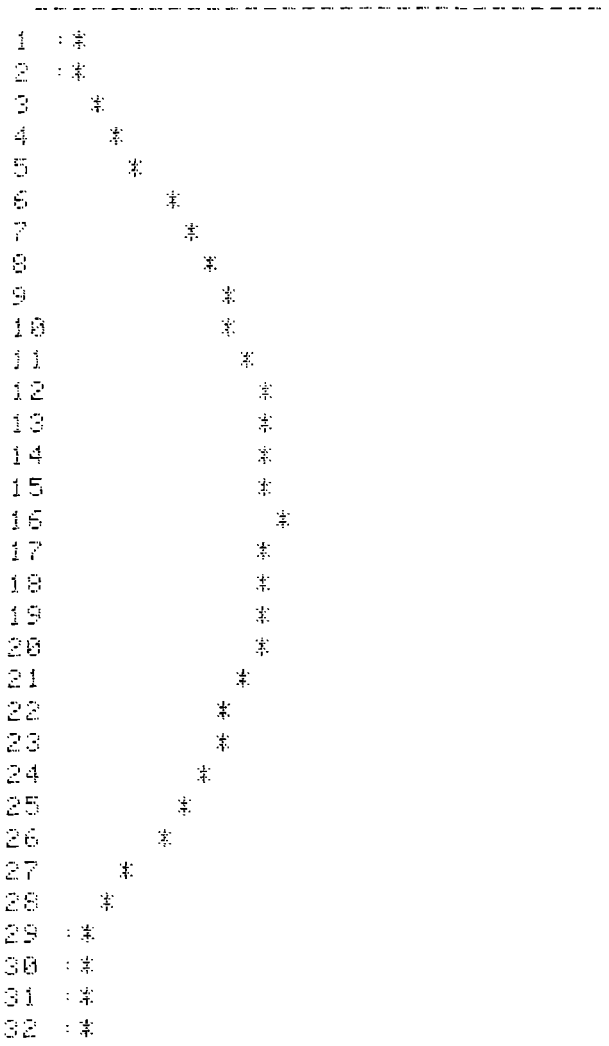
Mais voyons le détail du programme.

10-50	Entrées des paramètres, intervalle et coefficients du polynôme. La ligne 40 a pour but d'extraire l'espace contenu dans la variable numérique I. Cette opération est effectuée à des fins de présentation plus "propre".
65-90	Calcul d'une valeur du polynôme.
95	Détermination des valeurs initiales du maximum et du minimum dans l'intervalle considéré.
97-98	Encore une manœuvre de formattage de l'impression du rang du point calculé. Si ce nombre est inférieur à 10, il est complété par un 0, ainsi 3 sera affiché 03.
99	Impressions des X et Y calculés.
100	Détermination du maximum.
110	Détermination du minimum.
170-210	Écriture du graphe sur imprimante.
190	Calcul de la tabulation. C'est l'élément fondamental du programme.

Pour en finir avec les polynômes, en voici un exemple dont le graphe est particulièrement significatif.

```
INTERVAL -20,20
FONCTION -3X^2
      X                    Y
-------------------------------------

00:-20                -1200
01:-18.75             -1054.6875
02:-17.5              -918.75
03:-16.25             -792.1875
04:-15                -674.999999
05:-13.75             -567.1875
06:-12.5              -468.749999
07:-11.25             -379.6875
08:-10                -300
09:-8.75              -229.6875
10:-7.5               -168.75
11:-6.25              -117.1875
12:-5                 -75.0000001
13:-3.75              -42.1875
14:-2.5               -18.75
15:-1.25              -4.6875
16: 0                  0
17: 1.25              -4.6875
18: 2.5               -18.75
19: 3.75              -42.1875
20: 5                 -75.0000001
21  6.25              -117.1875
22: 7.5               -168.75
23: 8.75              -229.6875
24: 10                -300
25: 11.25             -379.6875
26: 12.5              -468.749999
27: 13.75             -567.1875
28: 15                -674.999999
29: 16.25             -792.1875
30: 17.5              -918.75
31: 18.75             -1054.6875
32: 20                -1200
```



La généralisation de ce programme à des polynômes de degrés plus élevés ne pose aucun problème de principe.

Il est toujours nécessaire d'apporter quelques modifications.

Les lignes 30, 40 et 70 sont liées au degré du polynôme.

EXEMPLE DEGRÉ 6

```
30 FOR I=6 TO 0 STEP -1
```

40 Pas de modification car le degré peut être décrit par un seul caractère.

```
70 FOR J=0 TO 6
```

En outre, si le degré dépasse 10, il est obligatoire de dimensionner correctement le tableau A( ).

L'accroissement de la précision ne présente pas plus de difficultés.

Les lignes à modifier sont les suivantes :

5, 60, 98 et 99 si le nombre de points dépasse 99, et 170.

GRAPHE D'UNE FONCTION QUELCONQUE

Les grandes lignes du programme précédent peuvent être reprises pour cette généralisation. Malheureusement, la fonction étudiée devra être préalablement introduite comme une ligne de programme dont le numéro est 70.

Le grand changement réside donc dans la simplification du module de calcul des ordonnées de la fonction.

Un problème apparaît toutefois avec les fonctions dotées d'un point de singularité dans l'intervalle de travail.

Une division par zéro peut avoir lieu et le programme est stoppé par l'émission d'un message d'erreur /0 ERROR. Le traitement d'erreur devra être adapté à chaque fonction en particulier.

Par exemple, si la fonction étudiée est la TANGENTE entre $-PI$ et $+PI$, il faut tester la valeur du COSINUS et "sauter" le calcul de la valeur de la fonction si celui-ci est nul.

Le traitement de SINUS entre $-PI$ et $+PI$ se fait simplement avec la ligne 70 Y(K)=SIN(I).

```
INTERVAL -PI,PI
FONCTION SIN(X)

   X                    Y
-----------------------------------
-
00:-3.1459          4.307332455E-03
01:-2.94941594   -.190995992
02:-2.75293187   -.378949419
03:-2.55644781   -.552320134
04:-2.35996375   -.704436498
05:-2.16347969   -.829444773
06:-1.96699562   -.922584402
07:-1.77051156   -.980123113
08:-1.5740275    -.99999478
09:-1.37754344   -.981384703
10:-1.18105937   -.925009035
11:-.984575311   -.83303722
12:-.788091248   -.709008513
13:-.591607185   -.55769578
14:-.395123123   -.384921828
15:-.19863906    -.197335336
16:-2.15499778E-03
-2.15499662E-03
17: .194329065    .193108272
18: .390813127    .380940358
19: .58729719     .554113119
20: .783781252    .705962529
21: .980265315    .830645128
22: 1.17674938    .923362887
23: 1.37323344    .980547846
24: 1.5697175     .999999418
25: 1.76620157    .980969067
26: 1.96268563    .924189119
27: 2.15916969    .831844577
28: 2.35565375    .707489037
29: 2.55213782    .555907943
30: 2.74862188    .382934431
31: 2.94510594    .195224856
```

```
 1                    *
 2                   *
 3                 *
 4                *
 5               *
 6               *
 7              *
 8              *
 9              *
10               *
11               *
12                *
13                 *
14                    *
15                     *
16                      *
17                        *
18                         *
19                           *
20                            *
21                             *
22                             *
23                               *
24                               *
25                               *
26                             *
27                             *
28                            *
29                           *
30                         *
31                        *
32                       *
```

Le programme correspondant est le suivant :

```
5 DIM Y(32)
10 CLS
20 LPRINT "    X","     Y"
30 LPRINT"----------------------
-----------"
40 INPUT "INTERVAL INF,SUP";IM,S
50 K=0
60 FOR I=IM TO S STEP (S-IM)/32
70 Y(K)=SIN(I)
80 IF I=IM THEN MY=MN=Y(K)
90 IF Y(K)<MN THEN MN=Y(K)
100 IF Y(K)>MY THEN MY=Y(K)
120 K$=STR$(K):K$=RIGHT$(K$,LEN
    (K$)-1)
130 IF LEN(K$)<2 THEN K$="0"+K$
140 LPRINT K$;":";I,Y(K)
145 K=K+1
150 NEXT I
160 LPRINT
170 LPRINT"----------------------
-----------"
180 FOR I=1 TO 32
190 LPRINT I;":";
200 H=15+INT(15/(MY-MN)*Y(I))
210 LPRINT TAB(H);"*"
220 NEXT I
```

Si la plage de variation est trop élevée, le graphe risque d'être trop plat ou trop dispersé ; en tout cas peu visible.

Ce problème peut souvent être résolu par un choix judicieux de l'intervalle.

Ainsi la courbe d'équation :

$$XSIN(X)\, e^{(-X/X)^2}$$

est, elle, spectaculaire dans les limites 2 à 18, mais elle n'est plus qu'un agrégat désordonné d'étoiles pour certains autres cas.

```
INTERVAL 2,18
FOCTION XSIN(X)*EXP(-I/I^2)
   X                    Y
-------------------------------
-
00:  2                 1.10303354
01:  2.5               1.00291969
02:  3                 .303350712
03:  3.5              -.922619709
04:  4                -2.35759351
05:  4.5              -3.52235218
06:  5                -3.92550397
07:  5.5              -3.23535469
08:  6                -1.41912067
09:  6.5               1.19889069
10:  7                 3.98669015
11:  7.5               6.15684415
12:  8                 6.98484471
13:  8.5               6.03383415
14:  9                 3.31901842
15:  9.5              -.64260464
16:  10               -4.92250658
17:  10.5             -8.39770122
18:  11               -10.0440095
19:  11.5             -9.22923126
20:  12               -5.92405098
21:  12.5             -.765285323
22:  13                5.05775822
23:  13.5              10.0763535
24:  14                12.9124469
25:  14.5              12.6525925
26:  15                9.12523222
27:  15.5              3.00029777
28:  16               -4.32736216
29:  16.5             -11.0538129
30:  17               -15.4100899
31   17.5             -16.1251887
32:  18               -12.787263
```

Nous ne pousserons pas plus loin l'étude de l'informatique appliquée aux mathématiques. Mais même dans le cas où cette branche ne vous est pas utile, les méthodes de développement d'un programme ainsi que certaines petites "ruses" de présentation ou conception devraient maintenant vous être familières.

Si on ne dispose pas d'une imprimante, il est possible de parvenir à un résultat comparable, non pas à l'aide de l'écran texte, mais avec les caractères semi-graphiques dont est équipée Alice. Seuls les modules d'impression d'un point changeront profondément, mais la base du programme reste exactement la même.

Si on adopte cette solution, il faut savoir qu'un calcul de coordonnées d'écran (0 à 31) sera probablement indispensable.

6

DESSIN ET MUSIQUE

Dans une large mesure, les ordinateurs familiaux doivent leur succès à la capacité qu'ils ont de générer des images colorées et des sons.

Alice, du moins dans sa version de base, est relativement pauvre dans ce domaine.

Malgré tout, certaines applications amusantes, voire même utiles, sont réalisables.

Deux programmes qui concrétisent la description du fonctionnement interne des "organes" audio-visuels d'Alice vous seront présentés dans ce chapitre.

LE GRAPHISME

Au cours du Chapitre 1, nous avons vu la configuration de l'écran du téléviseur ainsi que de la mémoire vidéo chargée de stocker les données de l'écran. Il existe une autre possibilité ; elle consiste dans l'affichage de petits "pavés" colorés.

```
10 CLS0
20 FOR I=1 TO 8
30 SET (10,2*I,I)
40 NEXT
```

– > Affichage de pavés de couleur

On en déduit les codes des différentes couleurs

1 – – > vert
2 – – > jaune
3 – – > bleu
4 – – > rouge
5 – – > blanc
6 – – > bleu clair
7 – – > magenta
8 – – > orange

La géographie de l'écran est apparemment modifiée.

En effet, le nombre de pavés affichables est deux fois plus élevé dans les deux dimensions que le nombre de caractères. C'est-à-dire 64×32 pavés dont les coordonnées varient entre 0,63 et 0,31.

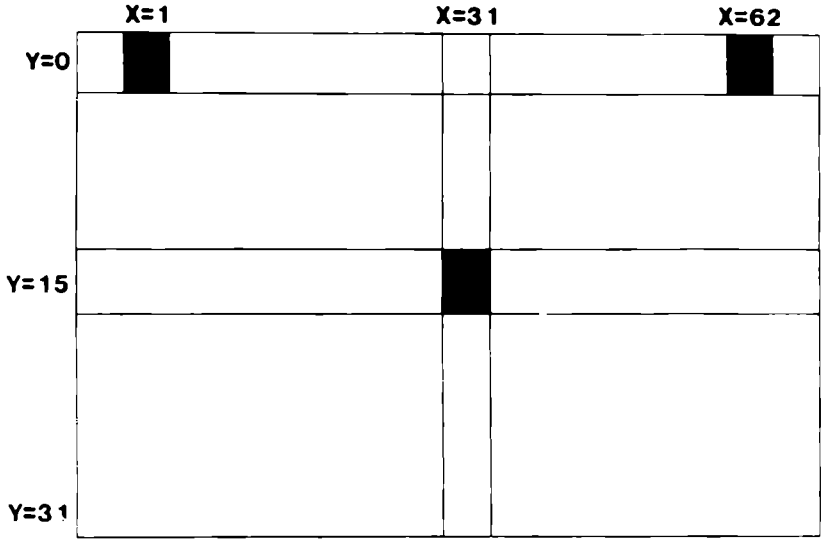


Figure 6.1. Écran basse résolution.

```
10 CLS0
20 FOR I=1 TO 8
25 SET (20,I,I)
30 SET (10,2*I,I)
40 NEXT
50 GOTO 20
```

Les apparences sont trompeuses.

– > *La colonne précédente apparaît bien correctement, mais celle que nous avons rajoutée scintille, et seulement cinq couleurs sont nettement visibles.*

Un doute effleura alors Indiana Jones, qui lut le bas de la statue : et il vit écrit

```
10 CLS
20 FOR I=128 TO 191 STEP 8
30 FOR J=I TO I+7
40 PRINT CHR$(J);" ";
50 NEXT J
55 PRINT
57 PRINT
60 NEXT I
70 A$=INKEY$:IF A$="" THEN 70
```

_ > *Nous voyons apparaître sur l'écran quatre fois la même séquence, à la couleur près.*

La première est verte
La seconde est jaune
La troisième est bleue
La quatrième est rouge

Ce n'est évidemment pas une coïncidence. Si on y réfléchit bien, les petits pavés ont exactement la taille d'un quart de caractère texte. Si on ne considère qu'un graphisme monochrome, le petit rectangle peut être soit allumé, soit éteint.

L'écran est subdivisé en 512 cases de la taille d'une lettre.
Affichons un pavé aux coordonnées 5,1 ,
Ceci correspond à un affichage dans le coin inférieur gauche de la case texte numéro 2. Si l'écran.était vidé par une instruction CLS0 (fond uniformément bleu foncé), il suffit d'afficher le caractère correspondant au pavé allumé en case texte 2.
Nous n'avons donc que l'illusion du graphisme.
En fait, le processus d'affichage est un peu plus complexe.

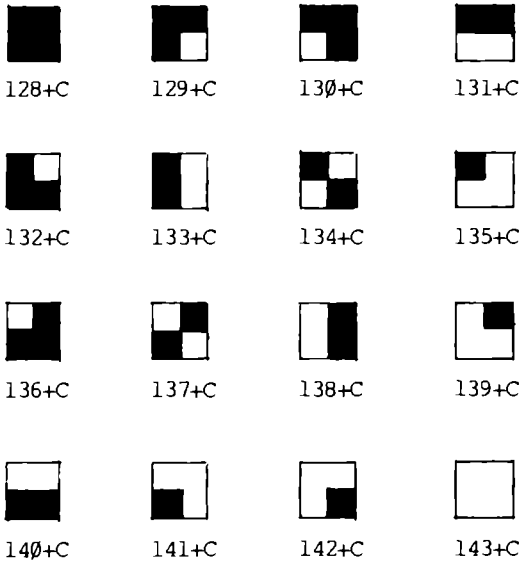


Figure 6.2. Combinaison de 4 pavés.

Le sous-programme qui en est chargé va d'abord examiner si la case texte ne possède pas d'abord un pavé allumé. Auquel cas, le caractère affiché sera le complément de celui qui est déjà résident.

Vu la taille des pavés, seuls seize caractères sont nécessaires pour décrire toutes les combinaisons.

Comme il existe huit couleurs (le bleu foncé est réservé au fond), il faut huit groupes de seize configurations de "points" pour parvenir à l'illusion d'une page graphique.

Au total, 128 caractères spéciaux sont réservés à cet usage. Nous en avons visualisé la moitié dans l'exemple précédent. Leur code ASCII est compris entre 128 et 255.

Nous sommes maintenant en mesure d'expliquer pourquoi la colonne que nous avions greffée scintille.

- Le pavé jaune s'affiche
- Le bleu va être étudié
- Le pavé jaune est détecté
- Le caractère choisi complémente la case texte
- La configuration sera choisie dans la couleur bleue (code 165)
- Le pavé jaune devient bleu
- Le processus continue avec l'affichage des autres pavés.
- Le programme reboucle, et le jeu de couleur alternatif recommence, ce qui provoque le scintillement.

Il faut parfois prendre plus de précautions avec les instructions graphiques. Pour plus de renseignements, consulter l'Annexe 3 aux rubriques SET, RESET, POINT.

Comme d'habitude, la réalisation d'un programme vaut mieux que tous les exercices.

En l'occurrence, il s'agit bien d'un logiciel dessinateur.

C'est un système de commandes intégrées qui est ici adopté.

La première ligne sert de "STATUS" comme disent les Anglo-Saxons, c'est-à-dire qu'elle fournit les renseignements sur l'état du programme à tout moment.

```
COULEUR DE L'ENCRE
MODE DESSIN
MODE CURSEUR
RAPPEL DES COMMANDES
```

Bien sûr, le curseur est clignotant et n'efface pas le fond, sauf dans certains cas relevant du problème évoqué dans le paragraphe précédent.

```
10 CLS0
15 DIM T(63,31)
20 COL=1
30 MO=-1
35 PRINT@16,"COM: Z0SW M G "
40 X=32:Y=16
50 A$=INKEY$
60 IF A$<"0" OR A$>"8" THEN 150
70 COL=VAL (A$)
150 IF A$="Z" THEN Y=Y-1:IF Y<2
    THEN Y=31:GOTO 250
160 IF A$="W" THEN Y=Y+1:IF Y>31
    THEN Y=2:GOTO 250
170 IF A$="Q" THEN X=X-1:IF X<0
    THEN X=63:GOTO 250
180 IF A$="S" THEN X=X+1:IF X>63
    THEN X=0:GOTO 250
190 IF A$="M" THEN MO=MO*-1:GOTO
    250
200 IF A$="G" THEN RESET(X,Y):
    GOTO 250
210 IF A$=CHR$(9) THEN 400
220 IF A$=CHR$(8) THEN 500
230 IF A$="N" THEN CLS0:GOTO 35
250 A=POINT(X,Y): IF A<>0 THEN
    270
260 SET (X,Y,COL):RESET (X,Y):
    GOTO 280
270 RESET (X,Y):SET (X,Y,A)
280 IF MO=1 THEN SET (X,Y,COL):
    PRINT@0,"DESSIN";:GOTO 300
290 PRINT@0,"CURSOR";
300 PRINT@7,"COULEUR";: SET (29,
    0,COL):SET (30,0,COL)
350 GOTO 50
400 FOR I=0 TO 63
410 FOR J=2 TO 31
420 T(I,J)=POINT (I,J)
430 NEXT J
440 NEXT I
445 GOSUB 450:CSAVE*T,"DESSIN"
447 GOTO 50
450 CLS
```

```
460 PRINT@64,"POSITIONNEZ LE LEC
TEUR"
470 PRINT:PRINT
480 INPUT"TAPEZ RETURN";K$
490 RETURN
500 GOSUB 450
510 CLOAD*T,"DESSIN"
520 FOR I=0 TO 63
530 FOR J=2 TO 31
540 IF T(I,J)=0 THEN 600
550 SET (I,J,T(I,J))
600 NEXT J
610 NEXT I
620 GOTO 50
```

Le déplacement est réalisé avec les touches

Z
Q S
W

En outre, notre dessinateur dispose des commandes suivantes :

M – – >	Sélection du mode, soit dessin, soit déplacement sans tracé
G – – >	Bascule du mode d'effacement. Cette commande n'est efficace qu'en mode déplacement
N – – >	Effacement d'écran
0 – 8 – – >	Sélection de la couleur
CTRL Q – – >	Chargement d'un dessin à partir du magnétophone
CTRL S – – >	Sauvegarde du dessin sur magnétophone

DÉTAIL

50-230	test de commande
250-270	clignotement du curseur sans effacement du fond
280-350	tracé du pavé s'il y a lieu
400-447	sauvegarde
500-620	chargement d'un dessin. Celui-ci sera superposé à celui qui réside déjà sur l'écran.

La sauvegarde a été effectuée à l'aide d'un tableau. L'écran est préalablement analysé, et si un pavé est allumé, son code de couleur est placé à un endroit équivalent à sa position "géographique", dans ce tableau.

Le chargement suit le processus inverse. Seule différence, il se contente de réafficher les points. Si une position du tableau est nulle, un pavé occupant les mêmes coordonnées ne sera pas effacé, ce qui provoque la superposition.

LE SON

L'instruction sonore d'Alice est extrêmement facile à décrire, aussi ne nous y attarderons-nous pas.

Les deux paramètres de SOUND spécifient respectivement la fréquence et la durée du son.

La solution la plus attrayante pour illustrer ce domaine est d'écrire un programme où sont mixés graphismes et sons.

```
                    
10 REM****MUSICIEN******
13 X=21
14 Y=23
20 REM
25 DIM NO(11)
26 NO(1)=255
27 FOR I=1 TO 10
28 NO(I)=I*18
29 NEXT I
30 CLS0
35 FOR J=10 TO20
40 FOR I=20 TO 48 STEP 4
50 SET(I,J,3):SET(I+1,J,3)
60 NEXT I
65 NEXT J
70 FOR J=10 TO 18
80 FOR I=23 TO 48 STEP 4
90 SET(I,J,5)
100 NEXT I
110 NEXT J
120 A$=INKEY$
```

```
125 IF A$<"2" THEN 120
130 IF A$>"9" THEN 200
135 A=ASC(A$)
140 SOUND NO(A-47),3
145 RESET(X,22)
146 X=(A-47)*4+8
147 SET(X,22,3)
150 GOTO 120
200 IF A$="Z"THEN NO=30:GOTO300
210 IF A$="E"THEN NO=45:GOTO300
220 IF A$="R"THEN NO=60:GOTO300
230 IF A$="T"THEN NO=75:GOTO300
240 IF A$="Y"THEN NO=90:GOTO300
250 IF A$="U"THEN NO=105:GOTO300
260 IF A$="I"THEN NO=120:GOTO300
270 GOTO120
300 RESET (Y,8)
310 SOUND NO,3
320 Y=(NO/15)*4+14
330 SET(Y,8,5)
340 GOTO 120
```

Le programme va tout d'abord dessiner un clavier du type piano (une bonne dose d'imagination est nécessaire pour le reconnaître en tant que tel).

Un curseur pour chaque rangée de touches se positionnera sur celle qui vient d'émettre une note.

NB : Cette petite illustration graphique est particulièrement appréciée des moins de quatre ans...

Il n'y a plus qu'à pianoter sur le clavier pour enfanter une symphonie synthétique digne des plus grands.

Mais trêve de plaisanteries, voyons plutôt si certaines astuces de programmation peuvent être réutilisées ultérieurement.

Ce programme illustre bien un fait courant en programmation.

Comparez les modules 125-140 et 200-310. Leur travail est absolument identique, la génération d'une note à partir de la valeur de A$.

Dans un cas, quatre lignes de programmes, dans l'autre neuf, et ne parlons pas de la rapidité d'exécution.

La raison en est simple. Les touches numériques permettent d'établir un algorithme général très court pour choisir la valeur d'une note.

La deuxième rangée, par contre, est formée par le groupe de lettres :

Z E R T Y U I

Et là, il est difficile de trouver une relation simple entre ces valeurs d'où découlerait la valeur de la note. Une solution peu élégante s'impose donc : tester chaque touche possible et effectuer le traitement correspondant.

Mais si cette méthode ne consomme que peu de matière grise, elle handicape fortement le programme en matière de rapidité d'exécution et surtout d'encombrement en mémoire centrale.

NB : L'auteur décline toute responsabilité dans le cas où une oreille mélomane serait blessée.

Vu ses connaissances musicales, celui-ci suggère de changer certains paramètres susceptibles de ne pas être en accord avec les règles de l'art.

A titre indicatif, voici une table d'équivalence entre la valeur de l'opérande et la note réelle.

Note	Valeur	Note	Valeur	Note	Valeur	Note	Valeur
FA	5	FA	133	FA	197	FA	230
	19		140		200		231
SOL	32	SOL	147	SOL	204	SOL	232
	45		153		207		234
LA	58	LA	159	LA	210	LA	236
	69		165		213		237
SI	79	SI	170	SI	216	SI	238
DO	89	DO	176	DO	219	DO	239
	99		180		221		240
RÉ	108	RÉ	185	RÉ	223	RÉ	242
	117		189		225		243
MI	125	MI	193	MI	227	MI	244

7

NOTIONS DE MÉMOIRE

SYSTÈMES BINAIRE ET HEXADÉCIMAL

Comme nous l'avons vu dans le Chapitre 1, le cerveau de l'ordinateur est ce fameux composant tant vanté dans les publicités de machines à laver.

Il fonctionne selon une logique à deux états. Mais clarifions un peu les choses.

Schématiquement, les composants informatiques ne peuvent détecter qu'un courant électrique donné. Pour eux, soit il y a établissement d'une tension, soit il n'y en a pas.

Cette manière de voir les choses est exactement calquée sur celle des logiciens pour qui, seul le vrai et le faux représentent les éléments fondamentaux de toutes choses.

En fait, ce raisonnement se retrouve partout, même chez les Chinois avec leur philosophie du Yin et du Yang.

Mais en informatique, la philosophie est difficilement applicable.

Ce sont les mathématiques qui, comme d'habitude, ont fourni les outils nécessaires au traitement de l'information. Cette dualité peut être décrite dans le cadre de la théorie des nombres.

SYSTÈME DÉCIMAL

L'homme ayant généralement dix doigts, il est parvenu à décrire toute entité numérique avec dix symboles.

Cela n'a pas été sans mal, et maints systèmes ont vu le jour tout au long du développement de la civilisation.

Celui qui est universellement adopté par tous nous vient des algébriciens arabes.

Grâce à lui, tout nombre peut être décrit par une combinaison adéquate des dix chiffres :

0,1,2,3,4,5,6,7,8,9

Ce système est dit de *base* 10 ou décimal. Examinons de plus près un nombre décimal quelconque, 123 par exemple.

Nous pouvons le décomposer sous la forme suivante :

$$123 = 100 + 20 + 3$$
$$123 = 10 \times 10 \times 1 + 10 \times 2 + 3$$

Nous avons déjà fait apparaître la base. Mais poussons plus loin le raisonnement, en employant la convention suivante.

$10^0 = 1$
$10^1 = 10$
$10^2 = 10 \times 10$
$10^3 = 10 \times 10 \times 10$
...
$10^n = 10 \times 10 \times ..... \times 10$ n fois

Notre nombre peut maintenant s'écrire :

$$123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

Recommencons le processus avec 1348.

$1348 = 1000 + 300 + 40 + 8$
$1348 = 10 \times 10 \times 10 + 3 \times 10 \times 10 + 4 \times 10 + 8$
$1348 = 1 \times 10 + 3 \times 10 + 4 \times 10 + 8 \times 10$

Nous constatons que la valeur du nombre dépend en fait de trois paramètres : la base, les symboles et leur *position* dans le nombre.

Nous sommes désormais en mesure de donner une formulation générale d'un nombre quelconque du système binaire.

Le *rang* d'un chiffre correspond à sa position diminuée de 1.

Pour 123, 3 est le chiffre numéro 1, son rang sera donc 0 ; le rang de 2 sera 1, celui de 1 sera 2, etc.

Ainsi, la valeur d'un nombre sera la somme de tous les symboles multipliés par la base élevée à la puissance du rang du symbole associé.

SYSTÈME BINAIRE

Un ordinateur ne connaît que deux symboles, le 1 ou le 0 ou, si vous préférez, le vrai et le faux, voire une tension et pas de tension.

Le système de numération employé est de base 2.

Si on applique les règles précédentes, il est très facile de convertir un nombre écrit en base 2, dans le système décimal.

EXEMPLE

1011 peut se décomposer comme suit :

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$1011 = 1\times8+0\times4+1\times2+1\times1$$
$$1011 = 8+0+2+1$$
$$1011 = 11 \text{ en base } 10$$

Le processus de comptage reste toujours le même. Le fonctionnement des opérateurs arithmétiques est indépendant de la base employée.

$$\begin{array}{r} 101011 \\ +\,110010 \end{array}$$

s'effectuera de la gauche vers la droite ;

$$\begin{array}{r} 1+0 = 1 \\ 1+1 = 0 \text{ retenue } 1 \\ 1+0+0 = 1 \\ 1+0 = 1 \\ 0+1 = 1 \\ 1+1 = 0 \text{ retenue } 1 \end{array}$$

Le total sera donc : 1011101

En informatique, les symboles 0 et 1 sont les *bits*. S'ils représentent l'unité élémentaire d'information, le traitement se fait simultanément sur un groupe de huit bits qui est l'octet dont nous avons déjà parlé et dont nous connaissons maintenant la signification exacte.

Un octet peut décrire tous les nombres compris entre 0 et 11111111.

Si nous convertissons dans le système décimal,

$$11111111 = 1\times2^7+1\times2^6+1\times2^5+1\times2^4+1\times2^3+1\times2^2$$
$$+1\times2^1+1\times2^0$$
soit $128+64+32+16+8+4+2+1$
$$255$$

Un octet est donc suffisant pour représenter tous les nombres entre 0 et 255 (souvenez-vous du nombre d'or de l'informatique).

Ce qui explique pourquoi un caractère a besoin d'un octet pour être stocké.

CONVERSION DÉCIMALE-BINAIRE

Cette opération suit exactement le même principe de décomposition.

Convertissons 254 en binaire.

254:2 = 127 reste 0
127:2 = 63 reste 1
63:2 = 31 reste 1
31:2 = 15 reste 1
15:2 = 7 reste 1
7:2 = 3 reste 1
3:2 = 1 reste 1

D'où 254 = 11111110, le dernier quotient est bien entendu à prendre en compte.

SYSTÈME HEXADÉCIMAL

A moins d'avoir un esprit particulièrement tortueux, on remarque que les calculs en binaire sont peu adaptés à notre propre mode de fonctionnement. Aussi les informaticiens ont-ils simplifié le problème en introduisant une nouvelle base numérique : la base 16 ou système hexadécimal.

Nous verrons pourquoi, malgré son apparente complexité, ce système est particulièrement adapté aux ordinateurs.

Bien sûr, seize symboles sont nécessaires pour décrire un nombre : les dix chiffres classiques auxquels viennent se greffer les six premières lettres de l'alphabet.

0 1 2 3 4 5 6 7 8 9 A B C D E F.

avec les correspondances suivantes :

Base 10	Base 16	Base 10	Base 16
0	0	8	8
2	2	9	9
3	3	10	A
4	4	11	B
5	5	12	C
6	6	13	D
7	7	14	E
		15	F

Le système hexadécimal est moins gros consommateur de symboles que la base 10 et à fortiori la base 2.

Les conversions se font toujours selon le même principe.

$$AADE = 10 \times 16^3 + 10 \times 16^2 + 13 \times 16^1 + 9 \times 16^0$$
$$AADE = 10 \times 4096 + 10 \times 256 + 13 \times 16 + 9$$
$$AADE = 43737$$

Par contre, la conversion hexadécimal-binaire, et vice versa, peut être grandement facilitée pourvu que l'on remarque une intéressante relation entre ces deux systèmes.

$1111 = 8 + 4 + 2 + 1$
$1111 = 15$ en base 10
$1111 = F$ en base 16

Un seul symbole hexadécimal suffit à décrire un groupe de quatre bits.

La conversion est donc chose aisée avec le tableau suivant.

Binaire	Hexadécimal	Binaire	Hexadécimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

EXEMPLE

BAAE = 1011101010101110

inversement

101101 = 0010 1101
101101 = 2 D
101101 = 2D en hexadécimal.

Un octet est donc un nombre compris entre 0 et FF en hexadécimal.

INTRODUCTION AU FONCTIONNEMENT INTERNE D'ALICE

Le microprocesseur qui équipe Alice est un MOTOROLA 6803. Il est dérivé du 6800 et les ouvrages qui traitent de celui-ci sont équivalents au niveau de la programmation.

Le processeur est capable d'exécuter une cinquantaine d'instructions élémentaires au niveau de l'octet de donnée, voire même des bits qui le constituent.

Tous les traitements s'effectuent entre la mémoire de travail, externe au processeur, et des mémoires intégrées privilégiées, les *registres* du processeur. La description des caractéristiques spécifiques du 6803 est le sujet du chapitre suivant.

Toujours est-il que la programmation du processeur consiste à implanter dans la mémoire des codes numériques (*codes opératoires*) qui, lorsqu'ils sont lus par le processeur, le forcent à exécuter une tâche bien précise. Généralement ces codes sont rentrés en mémoire sous forme hexadécimale. Ce type de programmation qui se situe à la source même de la machine s'appelle la programmation en langage machine. Le chapitre suivant en est une introduction, mais les techniques de programmation en hexadécimal peuvent occuper des ouvrages entiers, aussi nous contenterons-nous d'en exposer les principes de base.

MÉMOIRE

Mais comment le processeur gère-t-il sa mémoire externe ?

En fait de moyens magiques, ce ne sont que seize (hasard ?) fils conducteurs qui assurent la sélection des cases de mémoires destinées à travailler.

Chaque octet de mémoire occupe un emplacement numéroté que l'on nomme son *adresse mémoire*, à ne pas confondre avec une ligne de programme BASIC.

Le processeur va envoyer le numéro de la case sélectionnée, codé sur ses seize fils (*bus d'adresse*).

Nous pouvons de ce fait calculer la taille maximum de la mémoire que peut gérer le processeur.

Chacun des seize fils peut contenir un bit d'information ; le nombre ainsi représenté se situe dans l'intervalle binaire

0 à 1111 1111 1111 1111

ou 0 à 65535

ou 0 à FFFF

selon le système de numération considéré. En termes de documentation, le maximum de mémoire directement adressable par le processeur de l'Alice est de 64 Kilo-octet, étant bien entendu que Kilo signific 1024 en informatique.

Pour des raisons de commodités, nous ferons précéder tout nombre hexadécimal par le symbole "$".

Mais Alice n'utilise pas au maximum les performances du processeur au niveau de l'adressage de la mémoire.

La mémoire interne est répartie comme suit :

Huit K-octets de mémoire ROM contenant l'interpréteur BASIC, et les divers programmes de gestion des périphériques.

Quatre K-octets de mémoire RAM, en version de base.

Celle-ci comprend une zone de stockage des lignes de programme, la zone des variables numériques et celle des variables alphanumériques.

Il ne faut pas non plus oublier la RAM de l'écran vidéo qui n'utilise pas moins de 512 octets.

Seize K-octets de RAM d'extension optionnelle, mais indispensable pour obtenir la haute résolution graphique.

La mémoire de travail est donc portée à près de 20 K-octets, ce qui est nettement plus confortable pour le programmeur. Un seul programme de ce livre ne tient pas dans la version de base, il s'agit d'un utilitaire du prochain chapitre, dit "désassembleur 6803". De toute façon, l'exploitation des renseignements de ce chapitre requiert l'extension 16 K-octets.

BASIC

Nous savons déjà que l'interpréteur BASIC est un gros programme écrit en langage machine.

Une des fonctions d'un langage évolué est justement d'affranchir le programmeur du type de processeur employé pour l'écriture d'un logiciel.

C'est le langage qui s'occupe automatiquement de la liaison.

Malgré tout, aucun langage n'est parfait. Aussi est-il préférable de lui adjoindre des fonctions offrant à l'utilisateur d'intervenir directement au cœur de la machine.

Ce sont les PEEK, POKE, EXEC, USR.

Nous en verrons un peu plus loin des exemples d'utilisation. Auparavant il nous faut signaler que ces directives laissent totalement l'initiative au programmeur. Aussi faut-il bien se rendre compte que l'on ne dispose plus du "garde-fou" que représente l'ensemble des tests de validité du BASIC. Si une erreur est commise, ses conséquences seront beaucoup plus néfastes. Il est donc nécessaire de faire preuve d'un soin et d'une attention redoublés lors de l'emploi de telles fonctions.

Nous n'approfondirons pas leur mode d'emploi. Pour plus de précision, il est préférable de consulter l'Annexe 3.

Les premiers outils de travail en ce domaine sont les routines de conversion décimal à hexadécimal et inversement.

DÉCIMAL-HEXADÉCIMAL

La méthode est très simple et basée sur les fonctions de chaînes de caractères.

En effet, toutes les instructions d'Alice attendent un opérande décimal.

L'utilité de cette routine est de pouvoir visualiser la valeur hexadécimale sur l'écran.

Pour ce faire, il est naturel d'utiliser une variable numérique comme point d'entrée du sous-programme, et la variable alphanumérique qui lui est associée, comme point de sortie.

Comme le BASIC ne reconnaît pas les symboles A,B,C,D,E,F en tant que valeurs numériques, leur affectation sera entièrement à notre charge.

Le stockage des symboles surnuméraires se fera par le biais d'un tableau CO$.

Son initialisation doit avoir lieu au début du programme principal.

La routine de conversion proprement dite suit le processus énoncé ci-dessous :

- Affecter la valeur décimale dans la variable A$.
- Déterminer le coefficient de 16^1.
- Déterminer le coefficient de 16^0.

• Extraire les parties droite et gauche du chiffre.
• Si la valeur est inférieure à 9, alors il n'y a pas de changement.
• Si la valeur est supérieure, on extrait le symbole correspondant du tableau CO$.

Nous l'écrirons, non pas à titre de sous-programme, mais sous une forme autonome ; son réemploi dans un programme plus complexe ne pose que très peu de problèmes, comme nous le verrons par la suite.

```
10 REM ****CONVERSION****
20 REM ****DECIMALE A****
30 REM ****HEXADECIMAL***
32 DIM CO$(7)
33 FOR I=1 TO 6
34 CO$(I)=CHR$(I+64)
35 NEXT I
36 CLS
40 INPUT"VALEUR DECIMALE";A$
45 IF VAL(A$)>255 THEN 36
50 A= INT(VAL(A$)/16)
60 B= VAL(A$)-A*16
65 C$=RIGHT$(STR$(A),1)
67 B$=RIGHT$(STR$(B),1)
70 IF A>9THEN C$=CO$(A-9)
80 IF B>9 THEN B$=CO$(B-9)
90 PRINTA$;" HEX:";C$;B$
100 GOTO40
```

```
EXEMPLE D'UTILISATION
VALEUR DECIMALE:10 HEX:0A
VALEUR DECIMALE:20 HEX:14
VALEUR DECIMALE:30 HEX:1E
VALEUR DECIMALE:40 HEX:28
VALEUR DECIMALE:50 HEX:32
VALEUR DECIMALE:100 HEX:64
VALEUR DECIMALE:200 HEX:C8
VALEUR DECIMALE:254 HEX:FE
VALEUR DECIMALE:255 HEX:FF
VALEUR DECIMALE:
```

Le problème est ici un peu différent. Généralement cette routine est destinée à faciliter la vie de l'utilisateur qui n'aura pas à effectuer le calcul à la main.

Le point d'entrée est obligatoirement une variable alphanumérique, puisque le BASIC ne "comprend" que le système décimal. Par contre, la variable de sortie sera numérique.

Encore une fois, la méthode reste simple.

- Si la chaîne est plus petite que deux caractères, rajouter "0" en première position.
- Extraire les symboles de droite et de gauche.
- Comparer ceux-ci aux différents éléments du tableau.
- S'il y a égalité, affecter une valeur numérique fonction de l'indice de la case du tableau comparée.
- Sinon affecter la valeur numérique associée.
- Recalculer la valeur finale en fonction des deux nombres extraits.

Ici, la méthode est illustrée par un véritable sous-programme.

Essayez de l'incorporer dans l'un de vos programmes, cela constituera un excellent exercice.

```
100 DIM CO$(7)
110 FOR I=1 TO 6
120 CO$(I)=CHR$(I+64)
130 NEXT I
3100 IF LEN(A$)<2 THEN A$="0"+A$
3110 B$=RIGHT$(A$,1)
3120 C$=LEFT$(A$,1)
3130 FOR L=1 TO 6
3140 IF C$=CO$(L) THEN C$=STR$(
     9+L)
3150 IF B$=CO$(L) THEN B$=STR$(
     9+L)
3160 NEXT L
3170 A=VAL(C$)*16+VAL(B$)
3180 RETURN
```

FORMAT DES VARIABLES

Explorons un peu la manière dont fonctionnent les variables en BASIC. Nous disposons de la fonction VARPTR qui fournit la première adresse en mémoire de la variable opérande.

```
10 A=10
20 B=10
30 PRINT VARPTR (A)
40 PRINT VARPTR (B)
```

– > *Il en résulte deux nombres dont la différence est égale à sept.*

Sachant que le label utilise deux octets, nous pouvons en déduire que la valeur numérique en occupe cinq.

```
5 CLS
10 INPUT "VALEUR";A
20 L=VARPTR(A)
30 FOR I=L-2 TO L+5
40 PRINT PEEK (I);
50 NEXT I
60 PRINT
70 GOTO 10
```

Ce petit programme permet de visualiser l'évolution du contenu de la variable en cas de modifications.

Si nous le modifions pour traiter les variables alphanumériques :

```
5 CLS
10 INPUT "VALEUR";A$
20 L=VARPTR(A$)
30 FOR I=L-2 TO L+5
40 PRINT PEEK (I);
50 NEXT I
60 PRINT
70 GOTO 10
```

A l'usage, on voit que le premier octet donne la longueur de la chaîne de caractères. Les octets 2 et 3 sont en fait l'adresse de la mémoire où sont stockés les caractères. C'est un *pointeur*.

Remarquez que le BASIC différencie les deux types de variables par la valeur du dernier octet du label. Il y ajoute 128 pour les variables alphanumériques.

D'autre part, les autres octets n'ont pas de signification dans ce cas précis.

Vérifions cette particularité.

Il suffit pour cela d'aller lire le contenu de la mémoire pointée par le corps de la variable.

```
10 CLS
20 INPUT A$
30 A=VARPTR (A$)
40 N=PEEK(A)
50 AD=PEEK (A+2)*256+PEEK(A+3)
60 PRINT AD;" ";
65 FOR I=AD TO AD+N
70 PRINT PEEK (I);" ";
80 NEXT I
90 GOTO20
```

– > *liste des codes ASCII constituant la variable.*

Maintenant que ces variables ne sont plus un mystère pour nous, mettons à profit les particularités précédemment décrites.

En effet, en modifiant le pointeur, il est possible de visionner tout le contenu d'une portion de 255 octets de mémoire à l'aide d'une seule instruction PRINT. La variable alphanumérique va nous servir de fenêtre que nous pouvons ouvrir à notre guise dans la ROM et la RAM.

Une adresse occupe deux octets en mémoire ; de plus, elle est codée en hexadécimale. Il est donc nécessaire d'extraire la valeur de ces deux octets à partir d'une adresse de départ décimale.

ADRESSE = PARTIE HAUTE × 256 + PARTIE BASSE

```
10 REM**DUMP MEMOIRE*******
20 REM**AVEC UNE    *******
30 REM**VARIABLE    *******
40 REM**ALPHANUMERIQUE*****
45 A$=""
50 CLS
60 INPUT"ADRESSE DE DEPART";AD
70 AH = INT (AD/256)
80 AB = AD - AH*256
90 PRINT "PAGE:";AH;" ADRESSE:";
   AD
110 A = VARPTR (A$)
```

```
120 POKE A,255
130 POKE A+2,AH
140 POKE A+3,AB
150 PRINT A$
160 INPUT B$
170 AD=AD+256:IF AD>65535 THEN
    END
180 GOTO 70
```

DÉTAIL

60	entrée de l'adresse de visualisation
70-80	calcul des deux "composants" de l'adresse
120	ajustement à 255 du nombre d'octets à visualiser (maximum)
130-140	ajustement du pointeur vers la zone de mémoire à visualiser
150	visualisation
170-180	page suivante

Une notion vient d'apparaître : celle de page de mémoire.

Pour une raison de commodité, la mémoire est divisée en page de 256 octets numérotés de 0 à 255.

Si l'on utilise la notation hexadécimale, il est très facile de décomposer une adresse.

L'octet de valeur la plus élevée est le numéro de page, et celui de plus faible poids, le numéro au sein de cette page.

EXEMPLE

01 03 est le quatrième octet de la seconde page de mémoire.

En usant de cette notation, le 6803 peut disposer de 256 pages de 256 octets, soient 65 536 caractères.

La mémoire ainsi découpée est équivalente à un grand livre, plus facile à gérer qu'une liste.

Généralement, PEEK est une instruction sans problème puisqu'elle se contente de lire la mémoire.

Il en est tout autrement de POKE qui perturbe la mémoire centrale. Son utilisation ne doit en aucun cas être brouillonne. Si par malheur un problème survient, la touche INIT constitue souvent un dernier recours.

Pour vous fixer les idées, voici un POKE particulièrement spectaculaire.

```
POKE 49088,80
```

Si votre percepteur vous rend visite, exécutez le petit programme suivant et il ne sera plus capable d'étudier votre déclaration d'impôts.

```
10 POKE 49088,80
20 POKE 49088,154
30 GOTO 10
```

D'autres valeurs de l'argument engendreront des effets assez sympathiques dont nous reparlerons plus en détail dans le prochain chapitre.

LES OUTILS DE TRAVAIL

Pour travailler efficacement, il faut disposer d'un certain nombre de programmes utilitaires destinés à simplifier les procédures de programmation. Dans le cas de ce chapitre, nous allons réaliser deux types de programmes indispensables :

Un *dump* de la mémoire, c'est-à-dire une visualisation sur imprimante ou sur l'écran d'une zone de mémoire sous forme mixte ASCII et hexadécimale.

Un *moniteur*, qui va permettre d'implanter des codes directement en mémoire, sans avoir à calculer d'adresses complexes, et d'ajuster l'argument des instructions POKE.

Lors de la réalisation, il faut songer à faciliter au maximum l'usage de chaque logiciel, même si le temps de développement en est prolongé d'autant.

DUMP DE LA MÉMOIRE

La disposition adoptée permet l'affichage de dix lignes. Celles-ci contiennent l'adresse de départ en décimal, les valeurs hexadécimales des quatre octets adjacents et les équivalents en ASCII pour détecter la présence de texte implanté en mémoire.

Plusieurs fonctions intégrées facilitent l'exploration de la mémoire.

Il faut pouvoir continuer le listage sans avoir à redéfinir l'adresse de début. Celle-ci a donc pour valeur par défaut la prochaine position de mémoire. Le défilement quasi automatique est ainsi réalisé.

Mais il peut être nécessaire de changer complètement cette adresse ou de comparer élément par élément, deux zones de mémoire.

Nous en arrivons à l'analyse suivante :

- Entrer l'adresse.
- L'imprimer.
- Lire les quatre octets correspondants.
- Convertir les données en hexadécimal.
- Convertir les données en ASCII.
- Les afficher sur une ligne.
- Recommencer dix fois le cycle de lecture/impression.
- Tester le clavier.
- Si "Z" retour à la zone précédente, enlever 80 à l'adresse.
- Si "N", entrer la nouvelle adresse de départ.
- Si "Q", retour au BASIC.
- Si "W", continuer le listage séquentiellement en fonction de la dernière adresse lue.
- Si "C", branchement au sous-programme de comparaison.
- Entrer les paramètres suivants : adresse de départ ; adresse d'arrivée ; longueur de la zone de comparaison.
- Comparer un à un les éléments correspondants des deux zones.
- Si différent, afficher adresses et valeurs.
- Si le nombre d'éléments comparés est inférieur à la longueur spécifiée, recommencer le processus.
- Rctourner au programme principal.

Une traduction possible en BASIC peut être :

```
10 REM****DUMP DE LA*****
20 REM***** MEMOIRE *****
32 DIM CO$(7)
33 FOR I=1 TO 6
34 CO$(I)=CHR$(I+64)
35 NEXT I
36 CLS
```

```
40 GOTO 100
50 A= INT(VAL(A$)/16)
60 B= VAL(A$)-A*16
65 C$=RIGHT$(STR$(A),1)
67 B$=RIGHT$(STR$(B),1)
70 IF A>9THEN C$=CO$(A-9)
80 IF B>9 THEN B$=CO$(B-9)
90 RETURN
100 PRINT "ADRESSE DE DEPART";
105 INPUT AD
110 CLS
115 FOR K=1 TO 10
120 PRINTAD;
130 FOR R=0TO 3
140 A$=STR$(PEEK(AD+R))
150 GOSUB 50
160 PRINTC$;B$;"
170 NEXT R
171 PRINT" ";
172 FOR R=0 TO 3
173 D=PEEK(AD+R):IFD>127 THEN D=
D-128
174 IF D<32 THEN D=46
175 D$=CHR$(D)
177 PRINTD$;" ";
178 NEXT R
180 AD=AD+4
185 PRINT" "
190 NEXT K
200 A$=INKEY$
210 IFA$="Z"THEN AD=AD-80:GOTO11
0
220 IFA$="Q"THEN END
230 IFA$="W"THEN115
240 IFA$="N"THENPRINT"NOUVELLE A
DRESSE:";:INPUTAD:GOTO115
250 IF A$="C" THEN GOSUB 300
290 GOTO 200
300 CLS
310 PRINT"    COMPARAISON     "
320 PRINT:PRINT
330 PRINT"ADRESSE DE DEPART";
340 INPUT DE
```

```
350 PRINT"ADRESSE DE COMPARAISON
";
360 INPUTAR
370 PRINT"LONGEUR DE LA ZONE";
380 INPUT LO
390 CLS
400 FOR J=DE TODE+LO
420 IF PEEK(J)<>PEEK(AR+J-DE)THE
N LPRINT J;":";PEEK(J),AR+J-DE;"
:";PEEK(AR+J-DE)
430 A$= INKEY$
440 IFA$=CHR$(13) THEN INPUTC$
450 NEXT J
460 RETURN
```

DÉTAILS

32-90	Sous-programme de conversion décimal à hexadécimal.
100-110	Entrée de l'adresse.
115-190	Boucle pour les dix lignes.
130-170	Boucle pour la lecture/impression/conversion des codes en hexadécimal.
172-178	Boucle pour la lecture/impression/conversion des codes en ASCII.
200-290	Test du clavier pour sélectionner les commandes intégrées.
300-460	Sous-programme de comparaison de zones de mémoire.

A propos de la conversion des octets en ASCII, il faut noter que les lignes 173 et 174 éliminent les valeurs supérieures à 127 ou inférieures à 32.

Cette démarche a été développée car les codes écran ne sont pas forcément les codes ASCII associés.

Nous pouvons profiter de ce programme pour tester une zone particulièrement intéressante.

EXEMPLE D'UTILISATION

LISTING DES COMMANDES

```
49200 79 EF 80 79   y o  y
49204 EF 75 7B F0   o u { p
49208 F1 7B F1 C8   q { q H
49212 7F F5 56 50    u V P
49216 EA 8E 46 EA     N F
49220 8D 46 4F D2
F O R
49224 47 4F 54 CF   G O T O
49228 47 4F 53 55   G O S U
49232 C2 52 45 CD   B R E M
49236 49 C6 44 41   I F D A
49240 54 C1 50 52   T A P R
49244 49 4E D4 4F   I N T O
49248 CE 49 4E 50   N I N P
49252 55 D4 45 4E   U T E N
49256 C4 4E 45 58   D N E X
49260 D4 44 49 CD   T D I M
49264 52 45 41 C4   R E A D
49268 4C 45 D4 52   L E T R
49272 55 CE 52 45   U N R E
49276 53 54 4F 52   S T O R
49280 C5 52 45 54   E R E T
49284 55 52 CE 53   U R N S
49288 54 4F D0 50   T O P P
49292 4F 4B C5 43   O K E C
49296 4F 4E D4 4C   O N T L
49300 49 53 D4 43   I S T C
49304 4C 45 41 D2   L E A R
49308 4E 45 D7 43   N E W C
49312 4C 4F 41 C4   L O A D
49316 43 53 41 56   C S A V
49320 C5 4C 4C 49   E L L I
49324 53 D4 4C 50   S T L P
49328 52 49 4E D4   R I N T
49332 53 45 D4 52   S E T R
49336 45 53 45 D4   E S E T
49340 43 4C D3 53   C L S S
49344 4F 55 4E C4   O U N D
```

```
49048 45 58 45 C3   E X E C
49352 53 4B 49 50   S K I P
 9856 C6 54 41 42   F T A B
49360 A8 54 CF 54   ( T O T
49364 48 45 CE 4E   H E N N
49368 4F D4 53 54   O T S T
49372 45 D0 4F 46   E P O F
49376 C6 AB AD AA   F + - *
49380 AF DE 41 4E    ^ A N
49384 C4 4F D2 BE   D O R >
49388 BD BC 53 47   = < S G
49392 CE 49 4E D4   N I N T
49396 41 42 D3 55   A B S U
49400 53 D2 52 4E   S R R N
49404 C4 53 51 D2   D S Q R
49408 4C 4F C7 45   L O G E
49412 58 D0 53 49   X P S I
49416 CE 43 4F D3   N C O S
49420 54 41 CE 50   T A N P
49424 45 45 CB 4C   E E K L
49428 45 CE 53 54   E N S T
49432 52 A4 56 41   R $ V A
49436 CC 41 53 C3   L A S C
49440 43 48 52 A4   C H R $
49444 4C 45 46 54   L E F T
49448 A4 52 49 47   $ R I G
49452 48 54 A4 4D   H T $ M
49456 49 44 A4 50   I D $ P
49460 4F 49 4E D4   O I N T
49464 56 41 52 50   V A R P
49468 54 D2 49 4E   T R I N
49472 4B 45 59 A4   K E Y $
49476 4D 45 CD 00   M E M
```

COMPARAISON DE LA ZONE DE MEMOIRE D'UNE LONGUEUR DE 11 OCTETS, COMMENCANT A L'ADRESSE 49000,AVEC LA ZONE DEBUTANT EN 50000

```
49000   255     50000   50
49001   255     50001   129
49002   255     50002   32
49003   255     50003   39
49004   255     50004   251
49005   255     50005   160
49006   255     50006   0
49007   255     50007   39
49008   255     50008   246
49009   255     50009   129
49010   255     50010   128
```

Comme vous pouvez le constater, il s'agit là de la table des mots clés du BASIC débutant à l'adresse décimale 49220. Si vous observez les codes hexadécimaux, vous constaterez que la dernière lettre d'un mot clé voit son code augmenté de 128. C'est un moyen classique de recherche.

En effet, l'interpréteur va comparer caractère par caractère la commande et un mot clé. Si à un moment un caractère diffère, il passe au mot suivant dans la liste en testant toutes les valeurs. Si cette valeur dépasse 128 cela signifie la fin du mot clé en cours de comparaison et le début du suivant de la liste.

Si on utilise le dump par variable alphanumérique, le dernier caractère de chaque mot du vocabulaire apparaît sous la forme d'un pavé semi-graphique coloré.

Ceci explique pourquoi nous avons été obligés de ramener tous les codes ASCII à des valeurs inférieures à 128.

Les codes inférieurs à 32 ne sont pas affichés par le BASIC, aussi sont-ils indiqués par un symbole spécial.

Si on continue plus avant notre exploration, on rencontre la liste des messages d'erreur et le logo d'initialisation d'Alice :

```
MICROCOLOR BASIC 1.0
COPYRIGHT 1982 MICROSOFT
```

MONITEUR

Le programme précédent cantonne l'utilisateur dans un rôle passif d'observateur. Aussi est-il préférable de le complémenter en développant un programme dit moniteur. Celui-ci doit permettre d'interagir directement avec la mémoire.

La manière d'entrer les données est primordiale lorsqu'il est question d'un travail soutenu en profondeur.

La solution adoptée est celle de l'éditeur pleine page.

Onze lignes de codes hexadécimaux précédées de l'adresse d'implantation par groupe de huit octets sont visualisées sur l'écran. Un curseur clignotant se place alors sur la première position courante, c'est-à-dire celle du premier octet. Tout le temps que le curseur est positionné sur un octet, celui-ci alterne avec la valeur courante de l'octet précisé.

Cette méthode a l'avantage de tracer l'évolution du contenu d'une case mémoire en fonction des traitements en cours.

Les flèches à droite et à gauche déplacent le curseur dans toutes les directions de ce tableau, sans effacer la valeur précédente. Si à un moment donné, une valeur est entrée sous forme hexadécimale, celle-ci sera implantée en mémoire à l'adresse correspondant à la position courante du curseur.

Une telle disposition est particulièrement intéressante, mais le temps de développement du programme en est considérablement accru.

Les commandes intégrées sont très simples d'emploi, puisqu'elles ne sont qu'au nombre de 3 :

- Flèche droite.
- Flèche gauche.
- Flèche en bas pour faire défiler les zones de travail.

L'analyse du programme peut être menée de la façon suivante :

- Entrée de l'adresse de départ.
- Lecture de données successives en mémoire.
- Conversion en hexadécimal.
- Affichage.
- Positionnement du curseur.
- Entrée des commandes.
- Si valide, ajuster la position du curseur.
- Sinon, considérer l'entrée comme une valeur à implanter en mémoire.
- Conversion hexadécimal à décimal.
- Affichage et implantation en mémoire.
- Avancer d'un cran la position du curseur.
- Recommencer la scrutation du clavier.

```
5 H=5:V=0
10 DIM CO$(7)
20 FOR I=1 TO 6
30 CO$(I)=CHR$(I+64)
40 NEXT I
50 CLS
60 INPUT "ADRESSE";A$
65 CLS
70 AD=VAL (A$)
80 ST=AD
90 AD=AD-8
100 FOR I=1 TO 11
110 AD=AD+8
120 AH=INT(AD/256):AL =AD-AH*256
130 A$=STR$(AH):GOSUB 500:AD$=A$
140 A$=STR$(AL):GOSUB 500:AD$=
    AD$+A$
150 PRINT AD$;" ";
160 FOR J=1 TO 8
170 A=PEEK(AD+J-1+(I-1)*8)
180 A$=STR$(A)
190 GOSUB 500
200 PRINT A$;" ";
210 NEXT J
220 PRINT
230 NEXT I
240 PRINT@32*V+H,"--";
250 A$=INKEY$
270 IF A$=CHR$(10) THEN AD=ST+88
    :H=5:V=0:CLS:GOTO 80
280 IF A$<>CHR$(8) THEN 320
290 GOSUB 700:H=H-3
300 IF H<5 THEN H=26:V=V-1:IF V<
    0 THEN V=10
310 GOTO 240
320 IF A$<>CHR$(9) THEN 360
330 GOSUB 700:H=H+3
340 IF H>26 THEN H=5:V=V+1:IF V>
    10 THEN V=0
350 GOTO 240
360 PRINT@32*V+H,A$;
```

```
370 IF A$="" THEN GOSUB 700:
    FOR I=1 TO 80:NEXT I:GOTO
    240
375 SOUND 40,2
380 E$=A$
390 A$=INKEY$:IF A$="" THEN 390
395 SOUND 40,2
400 E$=E$+A$:A$=E$
410 GOSUB 600
420 POKE (ST+8*V+(H-5)/3),A
430 GOTO 330
500 A=INT(VAL(A$)/16)
510 B=VAL(A$)-A*16
520 C$=RIGHT$(STR$(A),1)
530 B$=RIGHT$(STR$(B),1)
540 IF A>9 THEN C$=CO$(A-9)
550 IF B>9 THEN B$=CO$(B-9)
560 A$=C$+B$
570 RETURN
600 IF LEN(A$)<2 THEN A$="0"+A$
610 B$=RIGHT$(A$,1)
620 C$=LEFT$(A$,1)
630 FOR I=1 TO 6
640 IF C$=CO$(I) THEN C$=STR$(
    5+I)
650 IF B$=CO$(I) THEN B$=STR$(
    9+I)
660 NEXT I
670 A=VAL(C$)*16+VAL(B$)
680 RETURN
700 A=PEEK(ST+8*V+(H-5)/3)
710 A$=STR$(A)
720 GOSUB 500
730 PRINT@32*V+H,A$;
740 RETURN
```

DÉTAIL

5-40 Initialisation des positions horizontale et verticale du curseur ainsi que du tableau de conversion.
60-90 Entrée de l'adresse de départ.
100-230 Boucle d'affichage des onze lignes du contenu mémoire.
240-430 Editeur du moniteur.
500-570 Conversion décimal à hexadécimal.
600-680 Conversion hexadécimal à décimal.
700-740 Calcul de l'adresse en fonction de la position du curseur.

Précisons un peu la description succinte que nous venons de faire en disséquant certains blocs stratégiques du programme.

240-430 L'éditeur est sans conteste la partie la plus délicate à réaliser car il faut bien calculer les positions du curseur sous peine de voir le listage prendre une apparence malsaine, voire illisible.

240 Affichage du curseur.
270 Test flèche en bas.
280 Test pour module flèche à gauche.
290 Restauration de la valeur de l'octet et décalage à gauche du curseur.
300 Test si curseur arrive au début, alors curseur va à la fin.
320 Traitement pour flèche à droite.
330 Restauration de l'octet et déplacement à droite du curseur.
340 Si curseur en fin de tableau, alors curseur va au début.
360-370 Restauration de l'octet de la position courante. Ceci provoque le clignotement.

En essayant le programme avec l'adresse 16384 des caractères parasites vont venir infester l'écran de visualisation. C'est normal, puisque la RAM vidéo commence à cette adresse. A chaque fois que l'on implante un code hexadécimal à cet endroit, le CRTC va le considérer comme le code ASCII d'un caractère à afficher.

En fait, il y a une petite différence entre code ASCII et code d'écran, mais elle ne sera pas analysée dans cet ouvrage.

Les heureux possesseurs d'une extension mémoire 16 K-octets peuvent regrouper ces deux programmes en un seul, voire même y connecter le désassembleur du chapitre suivant. Ils disposeront ainsi d'un logiciel indispensable pour les premiers pas en langage machine.

8

NOTIONS DE LANGAGE MACHINE

PRÉSENTATION DU 6803

Le but de ce chapitre est de familiariser le lecteur avec le langage machine d'Alice. Mais il n'est pas question d'en faire un traité de programmation en assembleur, ce qui occuperait beaucoup plus de place que ce livre ne peut en offrir. Malgré tout, les spécificités du microprocesseur 6803 sont décrites pour que le lecteur qui souhaiterait aller plus loin puisse avoir une base de départ. Dans ce cas, il est conseillé de se procurer l'ouvrage de Rodnay Zaks et Daniel Jean David qui traite de la programmation du 6800 (paru chez le même éditeur).

Le 6803 est un processeur fabriqué par Motorola. Il est dérivé du très classique 6800.

Mais avant de poursuivre plus avant l'énoncé des caractéristiques de ce processeur, voyons quelques règles générales sur la programmation en langage machine.

Nous savons déjà que le processeur dispose d'un jeu d'instructions élémentaires qui sont représentées par un code opératoire hexadécimal.

Comme il n'est pas très commode de programmer numériquement, le constructeur propose un jeu de symboles sur trois ou quatre caractères qui est censé décrire l'effet d'une instruction donnée.

Le symbole en question ou *mnémonique* est l'abréviation (en anglais) de la commande utilisée.

EXEMPLE

```
CPX  - >  compare X
INC  - >  incrément
```

Il est beaucoup plus facile d'écrire une suite d'instructions à l'aide des mnémoniques puis de les traduire en codes hexadécimaux à l'aide d'une table avant de les implanter en mémoire à l'aide d'un programme spécialisé, le moniteur du Chapitre 7 par exemple.

Dans ce domaine existent des programmes spécialisés qui assurent la traduction automatique des mnémoniques en codes opératoires. Ce sont des programmes assembleurs.

Le processeur est équipé d'un ensemble de cases mémoire privilégiées qui assurent la gestion d'un programme en langage machine ; ce sont les registres du microprocesseur.

Le 6803 dispose des registres suivants :

• Les registres A et B qui sont les accumulateurs. Ils sont les registres à tout faire du processeur, et la plupart des opérations passent par leur biais.

Leur taille est d'un octet.

Ils peuvent servir au stockage d'une donnée, aux additions et soustractions simples, et à divers autres traitements.

• Le registre d'index X. Sa taille est de seize bits soit deux octets. Son rôle est très spécialisé. Il permet une plus grande souplesse pour la gestion de données dans une table de la mémoire centrale dont il peut contenir l'adresse de départ.

• Le registre SP, pointeur de pile (*Stack Pointer*). Il va désigner l'adresse du début de la *pile* du processeur en mémoire centrale. Sa taille de deux octets permet de placer la pile à un endroit quelconque de la mémoire.

Souvenez-vous de l'instruction BASIC GOSUB : nous avions vu qu'elle stockait les adresses de retour les unes sur les autres ; les RETURN, eux, prennent le premier élément du haut pour dérouter l'interpréteur BASIC. Cette mémoire de stockage particulière est une pile, et SP permet de repérer son état à un moment donné.

• Le compteur ordinal PC, également sur deux octets, sert à suivre la progression d'un programme.

• Le registre d'état sur huit bits décrit les effets résultant d'une instruction quelconque.

Chacun des bits a une signification précise.

Bit 0 C, indicateur de retenue, s'il est à 1, cela signifie qu'une opération a généré une retenue qui est stockée à cet endroit. Exemple : 255 + 1 = 0 retenue 1, pour une addition sur un octet.

Le résultat peut être écrit en base 16 :

$FF + $1 = $01 00

Bit 1 V, indicateur de dépassement (overflow).

Bit 2 Z, indicateur de 0. Si une opération a généré un résultat nul, ce bit est mis à un.

Bit 3 N, indicateur de signe. Si une opération a généré un nombre dont le bit 7 est à 1, N contiendra 1. En effet, en arithmétique binaire, le bit 7 est réservé au signe du nombre traité.

Bit 4 I, il sert à gérer les interruptions.

Bit 5 H, indicateur de demi-retenue. Celle-ci est utilisée lors de la programmation d'opérations arithmétiques complexes par exemple.

Les autres bits ne sont pas utilisés.

Le registre d'état est particulièrement adapté aux tests de l'état du système.

Il permet la création de structures alternatives en langage machine.

LES MODES D'ADRESSAGE

Une instruction nécessite souvent un opérande précis.

Par exemple LDA A $FFFE chargera (*Load*) l'accumulateur A avec le contenu de la mémoire situé à l'adresse décimale 65534. Mais cette instruction peut charger un opérande d'une autre manière.

```
LDA A # $0A
```

chargera directement l'accumulateur A avec la valeur décimale 10.

La façon dont une instruction charge un opérande s'appelle le *mode d'adressage.*

Il en existe plusieurs sur le 6803 :

Adressage implicite. Il n'est pas utile de spécifier l'opérande. TAB transfère le contenu de A dans l'accumulateur B.

Adressage immédiat. Un registre est immédiatement chargé avec une valeur numérique. En représentation symbolique, l'opérande est précédé du symbole "dièse"

```
LDA B # $0B
```

charge immédiatement l'accumulateur B avec la valeur décimale 11.

Adressage en page zéro. L'opérande est contenu dans une adresse en page zéro de la mémoire, c'est-à-dire une adresse sur un octet. LDA A $FF chargera l'accumulateur avec le *contenu* de l'adresse décimale 255.

Adressage absolu. C'est une généralisation de l'adressage en page 0. L'adresse de l'opérande est spécifiée sur deux octets.

```
LDA B $4000
```

chargera l'accumulateur B avec le *contenu* de l'adresse décimale 16384.

Adressage indexé. L'adresse de l'opérande est trouvée en additionnant le contenu du registre d'index X et un déplacement mentionné dans l'instruction

```
LDX # $ FF00
LDA A $0A.X
```

chargera l'accumulateur A avec le contenu de l'adresse $FF + $0A soit

```
$FF0A
```

Adressage relatif. Il est utilisé par les opérations de branchement relatif.

L'opérande est un nombre sur un octet précisant la distance du saut par rapport à la position courante.

LE JEU D'INSTRUCTIONS

Nous ne décrirons pas le jeu d'instructions du 6803, mais seulement ses fonctions additionnelles par rapport au 6800 (voir Annexe 4).

ABX : addition du contenu de B et X, le résultat est rangé dans X. L'indicateur C peut être affecté.

ADDD : addition sur seize bits. Les accumulateurs A et B sont concaténés pour former un registre accumulateur sur seize bits nommé D (Double).

ASLD : décalage à gauche du pseudo-registre D. Le bit 7 de A est mis dans l'indicateur C et le bit 0 de B est mis à zéro.

LDD : charge le pseudo-registre D.

LSRD : décalage à droite du pseudo-registre D. Le bit 0 de B est stocké dans l'indicateur C. Le bit 7 de A est mis à zéro.

MUL : multiplication de A par B, le résultat est stocké dans le pseudo-registre D.

PSHX : place le contenu de X sur la pile.

PULX : place les deux premiers octets de la pile dans le registre X.

STD : place le contenu du pseudo-registre D à l'adresse spécifiée.

SUBD : soustraction sur deux octets à l'aide du pseudo-registre D.

La plupart des fonctions additionnelles proviennent du fait que, sur le 6803, les registres A et B peuvent en former un seul sur seize bits résultant de leur concaténation dans le sens A – B. Cette possibilité est particulièrement avantageuse à plus d'un titre et fait du 6803 un processeur beaucoup plus facile à mettre en œuvre que le 6800.

Nous arrêterons là la description du 6803, étant entendu que nous n'avons pas développé l'aspect programmation de ce microprocesseur.

Ce paragraphe était destiné à donner un complément d'information à l'utilisateur désireux de s'attaquer à la programmation assembleur à l'aide d'un ouvrage spécialisé. Il faut savoir surtout que toutes ces explications un peu ésotériques ne sont en fait que le reflet d'un jargon spécialisé. La programmation en langage machine est considérée par beaucoup comme un croquemitaine, mais en fait, sans être chose vraiment aisée, elle est abordable par tout un chacun, pourvu que le facteur temps ne soit pas un problème.

RÉALISATION D'UN DÉSASSEMBLEUR 6803

Nous avons vu qu'un assembleur est un programme spécialisé chargé de traduire les mnémoniques en codes opératoires. Mais il est souvent très utile de faire l'opération inverse, c'est-à-dire de désassembler un programme en langage machine.

En effet, la ROM d'un micro-ordinateur contient souvent un grand nombre de sous-programmes en langage machine très utiles. Aussi faut-il pouvoir les *lister* sous forme mnémonique pour en bien compren-

dre le fonctionnement et pouvoir ainsi les mettre en œuvre au sein de vos propres réalisations en assembleur.

Le programme, écrit en BASIC, proposé dans ce chapitre est un exemple de désassembleur 6803. Même si on ne veut pas savoir comment il fonctionne, on peut le rentrer tel quel, il se révélera bien vite un auxiliaire précieux.

```
5 DIM CO$(7)
10 DIM AN$(15)
20 DIM IN$(12)
30 DIM EN$(3,15)
40 MO=1
50 FOR I=0 TO 12
60 READ IN$(I)
70 NEXT I
80 FOR I=0 TO 15
90 READ AN$(I)
100 NEXT I
110 FOR I=0 TO 6
120 A$(I)="  "
130 NEXT I
140 FOR I=1 TO 6
150 CO$(I)=CHR$(I+64)
160 NEXT I
170 FOR I=0 TO 3
180 FOR J=0 TO 15
190 READ EN$(I,J)
200 NEXT J
210 NEXT I
220 GOTO 400
300 C$=STR$(C)
310 A=INT(VAL(C$)/16)
320 B=VAL(C$)-A*16
330 A$=RIGHT$(STR$(A),1)
340 B$=RIGHT$(STR$(B),1)
350 IF A>9 THEN A$=CO$(A-9)
360 IF B>9 THEN B$=CO$(B-9)
370 C$=A$+B$
380 RETURN
400 CLS
410 INPUT "ADRESSE";A$
420 GOSUB 3000
```

```
430 CLS
440 FOR I=1 TO 10
450 CI$=INKEY$:IF CI$="L" THEN
    MO=-1*MO
460 TE=PEEK(AD)
470 AH=INT(AD/256)
480 C=AH:GOSUB 300
490 IF MO=-1 THEN LPRINT C$;
500 PRINT C$;
510 AL=AD-AH*256
520 C=AL:GOSUB 300
530 IF MO=-1 THEN LPRINT C$;
540 PRINT C$;
550 C=TE:GOSUB 300:A$(0)=C$
560 GOSUB 700
570 AD=AD+1
580 NEXT I
590 A$=INKEY$:IF A$="" THEN 590
600 IF A$=CHR$(13) THEN 440
610 IF A$=CHR$(8) THEN 400
620 GOTO 590
700 TI=INT(TE/16):TJ=TE-TI*16
710 IF TI<8 THEN 1200
720 IF TJ>12 THEN 1000
730 A$(3)=IN$(TJ)
740 IF TJ=12 AND TI>11 THEN
    A$(3)="LDAD"
750 IF TJ=3 AND TI>11 THEN
    A$(3)="ADDD":GOTO 790
755 IF TJ=3 THEN 790
760 IF TJ=12 THEN 790
770 IF TI<12 THEN A$(4)="A ":
    GOTO 790
780 A$(4)="B "
790 ON TI-7 GOTO 800,950,900,850
    ,800,950,900,850
800 A$(5)="#"
810 AD=AD+1:TF=PEEK(AD)
820 C=TF:GOSUB 300
830 A$(1)=C$:A$(6)=C$
840 GOTO 2000
850 AD=AD+1:TF=PEEK(AD)
860 C=TF:GOSUB 300:A$(1)=C$
```

```
870 AD=AD+1:TF=PEEK(AD):C=TF:
    GOSUB 300:A$(2)=C$
880 A$(5)=A$(1)+A$(2)
890 GOTO 2000
900 AD=AD+1
910 TF=PEEK(AD):C=TF:GOSUB 300
920 A$(5)=C$:A$(1)=C$
930 A$(6)=",":A$(7)="X"
940 GOTO 2000
950 AD=AD+1:C=PEEK(AD):A$(1)=C$
960 A$(5)=C$
970 GOTO 2000
1000 IF TJ<>14 THEN 1050
1010 IF TI<12 THEN A$(3)="LDS":
     GOTO 790
1020 A$(3)="LDX":GOTO 790
1050 IF TJ<>15 THEN 1100
1060 IF TI=8 OR TI=12 THEN A$(3)
     ="???":GOTO 2000
1070 IF TI<12 THEN A$(3)="STS":
     GOTO 790
1080 A$(3)="STX":GOTO 790
1100 IF TI=9 OR TI=12 THEN A$(3)
     ="???": GOTO 2000
1110 IF TI>12 THEN 1160
1120 IF TI>9 THEN A$(3)="JSR":
     GOTO 790
1130 A$(3)="BSR":GOTO 1620
1160 A$(3)="STD":GOTO 790
1200 IF TI<4 THEN 1600
1210 A$(3)=AN$(TJ)
1220 IF A$(3)="???" THEN 2000
1230 IF TI>5 THEN TI=TI+4:GOTO
     790
1240 IF TJ=14 AND TI<6 THEN A$(3
     )= "???":GOTO 2000
1250 IF TI=4 THEN A$(4)="A  ":
     GOTO 2000
1260 A$(4)="B ":GOTO 2000
1600 A$(3)=EN$(TI,TJ)
1610 IF TI<>2 THEN 2000
1620 AD=AD+1
```

```
1630 C=PEEK(AD):GOSUB 300:A$(1
     =C$
1640 IF C>127 THEN 1700
1650 AB=AD+1+C:GOSUB 1800
1660 GOTO 2000
1700 AB=AD+1-(256-C):GOSUB 1800
1710 GOTO 2000
1800 AH=INT(AB/256):C=AH:GOSUB
     300
1810 A$(5)=C$
1820 AL=AB-AH*256:C=AL:GOSUB 300
1840 A$(5)=A$(5)+C$
1850 RETURN
2000 FOR K=0 TO 7
2010 IF MO=1 THEN 2030
2020 LPRINT " ";A$(K);
2030 PRINT " ";A$(K);
2040 A$(K)="  "
2050 NEXT K
2060 PRINT
2070 IF MO=-1 THEN LPRINT
2080 RETURN
3000 IF LEFT$(A$,1)<>"$" THEN
     AD=VAL(A$):RETURN
3010 AL$=MID$(A$,4,2)
3020 AH$=MID$(A$,2,2)
3030 A$=AH$:GOSUB 3100
3040 AD=A*256
3050 A$=AL$:GOSUB 3100
3060 AD=AD+A
3070 RETURN
3100 IF LEN(A$)<2 THEN A$="0"+A$
3110 B$=RIGHT$(A$,1)
3120 C$=LEFT$(A$,1)
3130 FOR L=1 TO 6
3140 IF C$=CO$(L) THEN C$=STR$
     (9+L)
3150 IF B$=CO$(L) THEN B$=STR$
     (9+L)
3160 NEXT
3170 A=VAL("&"+C$)*16+VAL(B$)
3180 RETURN
```

```
5000 DATA SUB,CMP,SBC,SUBD,AND,B
IT,LDA,STA,EOR,ADC,ORA,ADD
5001 DATA CPX
5010 DATA NEG,???,???,COM,LSR,??
?,ROR,ASR,ASL,ROL,DEC,???,INC,TS
T,JMP,CLR
5020 DATA ???,NOP,???,???,LSRD,A
SLD,TAP,TPA,INX,DEX,CLV,SEV,CLC,
SEC,CLI,SEI
5030 DATA SBA,CBA,???,???,???,??
?,TAB,TBA,???,DAA,???,ABA,???,??
?,???,???
5040 DATA BRA,???,BHI,BLS,BCC,BC
S,BNE,BEQ,BVC,BVS,BPL,BMI,BGE,BL
T,BGT,BLE
5050 DATA TSX,INS,PUL A ,PUL B ,
DES,TXS,PSH A ,PSH B ,PUL X ,RTS
,ABX,RTI,PSH X ,MUL,WAI,SWI
```

Voilà, par exemple, le listing désassemblé d'un sous-programme de la ROM commençant à l'adresse $F058 et se terminant par un RTS (*Return from Subroutine*, similaire au RETURN du BASIC) à l'adresse $F08A.

```
F058 C6 0A     LDA B   # 0A
F05A 7E E2 38  JMP     E238
F05D CE 00     LDX     # 00
F05F 8C A6     CPX     # A6
F061 04        LSRD
F062 97 97     STA A   97
F064 A6 03     LDA A   03    X
F066 A7 04     STA A   04    X
F068 A6 02     LDA A   02    X
F06A A7 03     STA A   03
F06C A6 01     LDA A   01
F06E A7 02     STA A   02
F070 96 96     LDA A   96
F072 A7 01     STA A   01    ,
F074 CB 08     ADD B   # 08
F076 2F E8     BLE     F060
F078 96 96     LDA A   96
F07A C0 08     SUB B   # 08
F07C 27 0C     BEQ     F08A
```

```
F07E  67  01      ASR     01    X
F080  66  02      ROR     02    X
F082  66  03      ROR     03 , X
F084  66  04      ROR     04    X
F086  46          ROR  A
F087  5C          INC  B
F088  26  F4      BNE     F07E
F08A  39          RTS
F08B  81  00      CMP  A  # 00
F08D  00          ???


E725  27  3F      BEQ     E766
E727  81  40      CMP  A  # 40
E729  26  0B      BNE     E736
E72B  BD  FC  29  JSR     FC29
E72E  BD  00  F3  JSR     00F3
E731  27  33      BEQ     E766
E733  BD  EA  2F  JSR     EA2F
E736  27  3B      BEQ     E773
E738  81  A1      CMP  A  # A1
E73A  27  50      BEQ     E78C
E73C  81  2C      CMP  A  # 2C
E73E  27  34      BEQ     E774
E740  81  3B      CMP  A  # 3B
E742  27  5E      BEQ     E7A2
E744  BD  E9  1A  JSR     E91A
E747  96  96      LDA  A  96
E749  36          PSH  A
E74A  26  06      BNE     E752
E74C  DD  F4  26  JSR     F426
E74F  BD  ED  05  JSR     ED05
E752  8D  57      BSR     E7AB
E754  32          PUL  B
E755  5D          TST  B
E756  26  09      BNE     E761
E758  BD  00  F3  JSR     00F3
E75B  81  2C      CMP  A  # 2C
E75D  27  15      BEQ     E774
E75F  8D  58      BSR     E7B9
E761  BD  00  F3  JSR     00F3
E764  26  D2      BNE     E738
E766  86  0D      LDA  A  # 0D
```

```
E768  20 54     BRA     E7BE
E76A  BD FA 7B  JSR     FA7B
E76D  27 F7     BEQ     E766
E76F  96 96     LDA A   96
E771  26 F3     BNE     E766
E773  39        RTS
E774  BD FA 7B  JSR     FA7B
E777  27 0A     BEQ     E783
E779  D6 D6     LDA B   D6
```

ANALYSE DU PROGRAMME

Si l'on étudie bien la table des codes opératoires du 6803 (Annexe 4), il est possible de la séparer en plusieurs zones caractéristiques en fonction de I et J.

Pour I de 8 à $F
et J de 0 à $B

Chaque colonne contient une seule mnémonique s'appliquant à A ou B, celle-ci est fonction de I. Les modes d'adressages dépendent de J.

Les cas particuliers dont il faut tenir compte sont dans les colonnes J = 3 et J = 7. Il est possible de mettre au point une méthode générale simple pour cette zone.

Pour I de 4 à 7
et J de 0 à $F

Là aussi, les mnémoniques sont facilement exploitables, mise à part la colonne J = $E.

Pour I de 8 à $F
et J de $C à $F

Une méthode générale peut être mise au point pour le mode d'adressage, mais le choix de la mnémonique est fonction de la valeur de I. Il n'est pas possible de trouver un rapport simple pour ce choix.

Pour I de 0 à 3
et J de 0 à $F

C'est l'anarchie, sauf pour la ligne I = 2 dont le mode d'adressage est toujours relatif.

Trois tableaux peuvent donc être utilisés pour stocker les mnémoniques.

AN$, IN$? et EN$, qui est un tableau à deux dimensions représentatif de la dernière zone décrite.

Détail du programme

5-220	Initialisation des différents tableaux dont celui de conversion décimal-hexadécimal, et vice versa. Les mnémoniques sont lues dans un fichier interne DATA commençant en 5000.
300-380	Sous-programme de conversion décimal à hexadécimal.
400-580	Désassemblage de dix instructions.
590-620	Test des commandes intégrées, si ENTER, suite du désassemblage ; si CONTROL Q, nouvelle adresse de désassemblage.
700-790	Traitement de la première zone.
1000-1160	Traitement de la deuxième zone.
1200-1260	Traitement de la troisième zone.
1600-1710	Traitement de la dernière zone.
1800-1850	Calcul de la valeur de chaque octet constitutif d'une adresse.
2000-2080	Affichage d'une ligne désassemblée à l'écran.
3000-3180	Conversion hexadécimal à décimal.
790	Cette ligne sélectionne les modes d'adressages immédiats, en page zéro, indexé ou en page zéro.
800-840	Adressage immédiat.
850-890	Adressage absolu.
900-940	Adressage indexé.
950-970	Adressage en page zéro.
1620-1710	Adressage relatif avec calcul de l'adresse du branchement.

L'adresse de départ peut être entrée sous forme hexadécimale pourvu qu'elle soit précédée du symbole "$" et qu'elle soit constituée de quatre symboles hexadécimaux.

Par exemple, $00FA est correct mais $FA ne l'est pas.

Un tirage des listings formatés est prévu sur imprimante. En cours de listing, la trappe de la touche "L" bascule le programme en mode imprimante ou inhibe celui-ci. Dans le programme, c'est la variable MO qui fait office de bascule entre les deux modes. MO peut prendre les valeurs 1 ou -1 selon les cas.

Le contrôle imprimante est réalisé par les lignes de programmes suivantes :

490, 530, 2010, 2070

La ligne 450 teste le clavier et bascule le logiciel en mode impression ou écran selon la valeur précédente de la variable MO.

Signalons pour terminer que la frappe de la touche ENTER désassemble dix instructions de plus à partir de la dernière adresse traitée.

Une nouvelle adresse de désassemblage peut être sélectionnée par la flèche à gauche (CONTROL Q).

ANNEXE 1

TABLE DES CODES ASCII

NUMÉROS DES BITS								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	HEX 1 / HEX 0	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*		J	Z	j	z
			1	0	1	1	11	VT	ESC	+	,	K	[	k	{
			1	1	0	0	12	FF	FS	,	<	L	\	l	|
			1	1	0	1	13	CR	GS		=	M	]	m	}
			1	1	1	0	14	SO	RS	.	>	N	^	n	~
			1	1	1	1	15	SI	US	/	?	O	_	o	DEL

LES CARACTÈRES DE CONTROLE ASCII

NUL — Null
SOH — Start of Heading
STX — Start of Text
ETX — End of Text
EOT — End of Transmission
ENQ — Enquiry
ACK — Acknowledge
BEL — Bell
BS — Backspace
HT — Horizontal Tabulation
LF — Line Feed
VT — Vertical Tabulation
FF — Form Feed
CR — Carriage Return
SO — Shift Out
SI — Shift In
DLE — Data Link Escape
DC — Device Control
NAK — Negative Acknowledge
SYN — Synchronous Idle
ETB — End of Transmission Block
CAN — Cancel
EM — End of Medium
SUB — Substitute
ESC — Escape
FS — File Separator
GS — Group Separator
RS — Record Separator
US — Unit Separator
SP — Space (Black)
DEL — Delete

ANNEXE 2

TABLE DES MESSAGES D'ERREUR

MESSAGE	SIGNIFICATION
BS	L'indice d'une fonction est hors limite. Exemple : si un tableau A n'est pas dimensionné, une instruction A(12) provoquera le message.
CN	La machine ne peut accomplir une directive CONT.
DD	Un tableau a été redimensionné, utilisez CLEAR si c'est possible, ou changez la ligne contenant DIM.
FC	L'opérande d'une instruction n'est pas correct.
FM	Fichier incompatible. Ce message peut apparaître lors du chargement d'un programme à partir du lecteur de cassette.
ID	Une commande qui ne peut être accomplie a été utilisée en mode immédiat ; READ répond à ce critère.
IO	Erreur d'entrée/sortie. Elle apparaît lorsqu'une anomalie a été détectée pendant une transmission avec un périphérique.
LS	Une chaîne de caractères a une longueur supérieure à 255.
NF	Il manque une instruction FOR dans une boucle.
OM	La mémoire est saturée. Utilisez ":" pour compacter le programme, supprimez les REM, employez le maximum d'opérandes possibles avec chaque instruction. Si ceci échoue, acheter une extension de mémoire vive. Dans certains cas particuliers, le dépassement peut être dû à une mauvaise utilisation des GOSUB.
OD	Typique d'une configuration READ-DATA. Le pointeur de fichier DATA a dépassé la fin. Voir les limites de boucles s'il y a lieu ou utiliser RESTORE.
OS	La mémoire des chaînes de caractères est saturée. Pour y remédier utilisez CLEAR No en début de programme.
OV	Dépassement de la capacité lors d'un calcul numérique.

RG	Il manque un GOSUB dans un appel à un sous-programme. Ceci est souvent dû à des indirections GOTO anarchiques.
SN	La plus répandue. La syntaxe d'une instruction n'est pas respectée.
ST	Une opération sur des chaînes est trop complexe. La création de variables intermédiaires pour fragmenter le traitement résoud toujours le problème.
TM	Les données traitées ne sont pas compatibles. Exemple : A = "BONJOUR". Une chaîne de caractères ne peut être affectée dans une variable numérique, et vice versa.
UL	Un branchement à une ligne inexistante a été tenté.
/0	Tentative de division par zéro.

ANNEXE 3

RÉSUMÉ DES COMMANDES BASIC

LISTE ALPHABÉTIQUE

(
*
+
–
/
<
=
>
ABS
AND
ASC
CHR$
CLEAR
CLOAD
CLS
CONT
COS
CSAVE
DATA
DIM
END
EXEC
EXP
FOR
GOSUB
GOTO
IF
INKEY$
INPUT
INT
LEFT$
LEN
LET

LIST
LLIST
LOG
LPRINT
MEM
MID$
NEW
NEXT
NOT
OFF
ON
OR
PEEK
POINT
POKE
PRINT
READ
REM
RESET
RESTORE
RETURN
RIGHT$
RND
RUN
SET
SGN
SIN
SKIPF
SOUND
SQR
STEP
STOP
STR$
TAB
TAN
THEN
TO
USR
VAL
VARPTR

,
;
:

(

La parenthèse donne un niveau de priorité maximal pour un traitement donné. Celle-ci doit être utilisée conjointement avec la parenthèse fermée comme délimiteur ")".

Attention, un trop grand nombre de parenthèses imbriquées peut amener à une erreur du type ST, formule trop complexe.

Exemple 1

```
10 PRINT (5+3)*(8-2)
```

On obtient :

```
- >    48
```

Exemple 2

```
10 A$="BONJOUR"
20 T=1:U=3
30 PRINT LEFT$(A$,(U-T)*2)
```

On obtient :

```
- >    BONJ
```

*

Opérateur de la multiplication. Sa mise en œuvre ne pose pas de problème particulier.

Exemple

```
10 PRINT 2*3
```

On obtient :

```
- >    6
```

Contexte : +, -, /, ↑

NB : * est prioritaire sur + et − ; en revanche / a le même degré de priorité.

Pour une opération complexe, il est conseillé d'employer les parenthèses.

Enfin, l'élévation à la puissance ↑ est toujours calculée en premier.

+

Opérateur d'addition. Voir *.

−

Opérateur de soustraction. Voir +.

/

Opérateur de division. Il est possible de simuler la division entière en procédant comme suit.

```
10 A=23
20 B=12
30 Q=INT(A/B)
40 R=A-Q*B
50 PRINT "QUOTIENT:";Q
60 PRINT "RESTE:";R
```

On obtient :

```
- >    QUOTIENT:1
       RESTE:11
```

Voir * pour complément d'information.

<

Opérateur "strictement inférieur à". Lors de la comparaison de deux nombres, ce sont 0 ou −1 qui seront générés si la proposition est fausse ou vraie. Dans le cas d'une chaîne de caractères, ce sont les codes ASCII qui sont comparés caractère par caractère.

Contexte : <, =, IF, THEN, <=, >=

Exemple

```
10 A$ = "AMI"
20 B$ = "BAIE"
30 PRINT A$ < B$
```

On obtient :

```
- >    0
```

En effet, le code de A est 65, celui de B, 66. Donc "AMIE" n'est pas inférieur à "BAIE". La proposition est fausse.

=

Instruction d'affectation d'une donnée à une variable. Ou test d'égalité.

Exemple

```
10 A = 10
20 PRINT A
```

On obtient :

```
- >    10
```

Exemple

```
10 A = 5
20 B = 5
30 PRINT A = B
```

On obtient :

```
- >    - 1
```

Dans ce cas, " = " s'est comporté comme un test de comparaison et non comme une instruction d'affectation.

LET peut éventuellement précéder un " = ", auquel cas, celui-ci ne fonctionnera qu'en mode d'affectation.

<

Opérateur "strictement inférieur à". Voir > pour complément d'information.

ABS

Valeur absolue d'un nombre. La valeur du nombre est extraite, indépendamment de son signe.

Exemple

```
10 A=5
20 B=5
30 PRINT ABS (A)
40 PRINT ABS (B)
```

On obtient :

```
- >    5
       5
```

Contexte : INT, SIN, COS, TAN, LOG, EXP

AND

Opérateur logique permettant de regrouper le résultat de plusieurs comparaisons. Il correspond au ET des mathématiciens.

Exemple

```
10 A=5
20 B=6
30 C=7
40 PRINT A > B AND B > C
50 PRINT A < B AND B < C
```

On obtient :

```
- >    0
      -1
```

Chacune des deux propositions de la ligne 50 est vraie, l'ensemble, lié par AND est vrai, ce qui n'est pas le cas ligne 40. Voir Chapitre III.

Contexte : OR, NOT, IF, THEN

ASC

Fournit le code ASCII d'un caractère ou du premier caractère d'une chaîne.

Exemple

```
10 A$="ALICE"
20 PRINT ASC (A$)
```

On obtient :

```
- >    65
```

Soixante-cinq est bien le code du A (voir Annexe 1).

Contexte : CHR$, RIGHT$, LEFT$, MID$, LEN, VAL

CHR$

C'est le contraire de la fonction ASC. A partir d'un code, il restitue le caractère correspondant.

Exemple

```
10 FOR I=0 TO 255
20 PRINT CHR$(I);
30 NEXT I
```

- > *Tous les caractères que peut imprimer ALICE*

Contexte : ASC, VAL, LEN, RIGHT$, LEFT$, MID$

CLEAR

Cette instruction a pour effet de "nettoyer" variables et tableaux. Tous sont réinitialisés à 0. Les précédentes instructions DIM sont inhibées.

CLEAR peut aussi être utile pour redimensionner la taille de la mémoire des chaînes de caractères. Celle-ci aura une longueur égale à l'opérande spécifié. Un opérande additionnel, avec la virgule pour séparateur, permet de fixer l'adresse mémoire la plus haute utilisable par le BASIC.

Exemple

```
10 CLEAR 200
```

- > *La zone des variables alphanumériques est alors de 200 octets.*

CLOAD-CLOAD*

Instruction de chargement à partir de la cassette. Elle peut être utilisée à partir d'un programme. Il suffit de mentionner le nom du programme à charger entre guillemets.

Exemple

```
CLOAD "INVADERS"
```

– > *Chargera le programme INVADERS.*

Parallèlement aux programmes, il est possible de charger des données organisées en tableaux. C'est l'ordre CLOAD* qui nécessite le nom du tableau qui sera affecté comme opérande supplémentaire. Le chargement sous forme binaire peut être réalisé avec CLOADM suivi optionnellement de l'adresse de chargement en mémoire.

Exemple

```
CLOAD* TA$, "PRENOM"
```

– > *Le fichier PRÉNOM sera chargé dans le tableau TA$. Bien entendu, celui-ci doit être suffisamment grand pour contenir toutes les données. Toutes les restrictions d'usage à propos des tableaux s'appliquent à ce cas précis. Le programme résident en mémoire n'est pas affecté par le chargement.*

Contexte : SKIPF, CSAVE, CSAVE*

NB : Alice utilise n'importe quel lecteur de cassettes du commerce, à la différence du Tandy MC 10 qui a besoin du magnétophone Tandy.

CLS

Effacement de l'écran. Si aucun opérande n'est spécifié, l'écran reste vert, couleur d'affichage du texte.

En revanche, la couleur du fond peut varier selon le paramètre employé avec CLS.

```
CLS 0 – > BLEU FONCÉ
CLS 1 – > VERT
```

```
CLS 2 - > JAUNE
CLS 3 - > BLEU
CLS 4 - > ROUGE
CLS 5 - > BLANC
CLS 6 - > BLEU CLAIR
CLS 7 - > MAGENTA
CLS 8 - > ORANGE
```

Pour des valeurs supérieures à 8, le système affiche le copyright du BASIC sur fond vert. Si l'opérande dépasse 255, un message d'erreur sera généré (FC error).

CONT

Après une interruption du programme, il est possible de continuer son exécution à l'endroit où il s'est arrêté en utilisant cette directive. Le contenu des variables et tableaux est sauvegardé, à l'opposé de ce qui se passe avec RUN.

Exemple

Si la touche BREAK est enfoncée, CONT permet la reprise de l'exécution.

Si la reprise n'est pas possible, un message d'erreur est envoyé (CN error).

COS

Fonction mathématique cosinus. L'argument est en radian, il est donc conseillé d'introduire une variable PI = 3.14159265 avant de commencer les calculs.

Exemple

```
10 PI=3.14159265
20 PRINT COS (PI/4)
- >    .707106782
```

Soit racine de 2 sur 2.

Contexte : SIN, TAN, SQR, LOG, EXP

CSAVE-CSAVE*

Ordre de sauvegarde sur bande magnétique. CSAVE concerne les programmes BASIC. Il suffit de mentionner le nom de sauvegarde entre guillemets.

Exemple

```
CSAVE "INVADERS"
```

Effectue une copie du programme résident en mémoire sous le nom INVADERS.

Les données peuvent aussi être sauvegardées sous la forme d'un tableau. Un opérande supplémentaire pour CSAVE* est nécessaire, pour indiquer dans quel tableau se trouvent les données à sauvegarder.

Exemple

```
10 DIM A$(3)
20 A$(0) = "MONIQUE"
30 A$(1) = "CAROLE"
40 A$(2) = "SYLVIE"
50 A$(3) = "SOPHIE"
60 CSAVE* A$,"PRENOM"
```

Contexte : CLOAD, CLOAD*, SKIPF

DATA

Lorsque cette instruction est rencontrée, toutes les données présentes sur la ligne de programme sont stockées les unes après les autres (séquentiellement) dans une portion de la mémoire réservée à cet effet. Si plusieurs lignes de DATA existent dans un programme, les données de chacune d'entre elles sont rangées en fonction du numéro de la ligne de programme.

Le numéro de ligne importe peu. Mais il est conseillé de regrouper toutes les lignes de DATA en début ou en fin de programme. Bien entendu, si l'ensemble des données excède 127 caractères, plusieurs lignes de DATA sont nécessaires. La relecture du fichier interne se fait par READ.

Attention, un fichier DATA n'est pas modifiable par le programme BASIC.

Exemple

```
10 DATA 1,3,PILOTE,MERINGUE
20 .....
:
:
80 .....
90 DATA 6,SYBEX,ALICE
```

– > *Le fichier interne sera de la forme*
1 3 PILOTE MERINGUE 6 SYBEX ALICE

Contexte : READ, RESTORE

DIM

Dimensionnement d'un tableau. DIM réserve un emplacement de mémoire au tableau concerné. Il est nécessaire de fixer le nombre d'éléments par dimension. Si aucune instruction DIM n'est mentionnée, un tableau sera automatiquement positionné à dix éléments par dimension. Le nombre d'éléments par dimension est égale à l'argument plus 1.

Exemple

```
10 DIM A(3,3)
20 FOR I=0 TO 3
30 FOR J=0 TO 3
40 A(I,J)=J
50 PRINT A(I,J);
60 NEXT J
70 NEXT I
```

On obtient :

– > 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3

Attention : il est interdit de redimensionner un tableau pendant le déroulement d'un programme.

END

Lorsque cette directive est rencontrée, le programme s'arrête et le contrôle est redonné à l'utilisateur.

EXEC

Appel d'un programme en langage machine. L'opérande est l'adresse décimale du début du programme.

Contexte : USR, LANGAGE MACHINE

EXP

Opérateur d'exponentiation. Fonction inverse du logarithme népérien.

Exemple

```
10 PRINT EXP(1)
20 PRINT EXP(LOG(5))
```

On obtient :

```
- >    2.71828183
       5
```

Contexte : LOG, SIN, COS, TAN, SQR ...

FOR

Une des instructions fondamentales du BASIC. Dans une boucle, FOR sert à préciser le label de la variable qui fait office de compteur.

Contexte : =, TO, STEP, NEXT

GOSUB

Cette instruction est indispensable pour l'écriture de programmes complexes. Elle sert à délimiter des "morceaux" de programme indépendants du programme principal.

Ce sont les sous-programmes. A chaque fois qu'un traitement doit être répété, il est plus simple de le placer dans un sous-programme. Ainsi, il suffira de l'appeler grâce à GOSUB toutes les fois qu'il est nécessaire de le faire, d'où un gain de place mémoire important. Il est conseillé de placer tous les sous-programmes en début de programme.

La différence entre GOSUB et GOTO provient du fait que GOSUB mémorise l'endroit d'où le programme est parti. Dès qu'un RETURN est rencontré, le BASIC se branche à l'endroit où a eu lieu l'appel.

Exemple

```
10 PI=3.141592654
20 GOTO 50
30 A=A*PI/180
40 RETURN
50 INPUT "ANGLE EN DEGRÉ";A
70 GOSUB 30
80 PRINT COS (A)
```

On obtient :

```
- >    ANGLE EN DEGRÉ? 45
       .707106781
```

Le sous-programme en lignes 20 à 30 s'occupe de la conversion degrés-radians. Si le programme était plus complexe, un seul appel à ce sous-programme suffirait à la conversion, pourvu que le paramètre A soit géré avec soin.

Attention : l'adresse de retour est sauvegardée en mémoire. Si aucune instruction RETURN n'est interprétée, la mémoire qui stocke ces adresses se retrouve vite saturée. Cette erreur est souvent difficile à détecter. Un message OM error est émis.

Ce problème survient souvent lorsqu'un GOTO mal utilisé effectue un saut hors d'un sous-programme, ou lorsqu'une instruction RETURN a été oubliée.

Exemple

```
10 GOTO 50
20 I=I+1
30 GOTO 50
40 RETURN
50 GOSUB 20
```

On obtient :

```
- >    OM error
```

La valeur finale de I est de 430. Mais celle-ci dépend de l'encombrement du programme résident.

Contexte : GOTO, ON GOSUB, ON GOTO, RETURN

GOTO

Branchement à un numéro de ligne. L'interpréteur est dérouté, et l'exécution se poursuit au numéro de ligne mentionné.

Exemple

```
10 PRINT "A"
20 GOTO 10
```

– > *Ce programme imprimera des A jusqu'à la fin des temps, ou jusqu'à ce qu'il soit interrompu.*

Contexte : GOSUB, ON GOTO, IF, THEN

IF

Début de test, tout traitement situé après IF sera considéré comme une proposition logique dont les seuls résultats peuvent être – 1 ou 0, vrai ou faux. Si, malgré tout, le traitement n'est pas un véritable test, son résultat déterminera la validité de l'ensemble. Si nul, alors la condition est fausse, sinon elle est vrai quel que soit le résultat.

Exemple

```
10 A=1
20 B=3
30 IF A=B THEN PRINT "VRAI": GOTO 50
40 PRINT "FAUX"
50 END
```

– > FAUX

Exemple

```
10 IF 3+2 THEN PRINT "VRAI": GOTO 30
20 PRINT "FAUX"
30 END
```

```
- >    VRAI
```

En effet, le résultat de l'opération est différent de zéro, il est donc considéré comme une proposition vraie.

Contexte : THEN, GOTO, <, >, =, <=, >=

INKEY$

Entrée d'un caractère à partir du clavier lors du déroulement d'un programme. Le programme n'est pas arrêté, mais si, à un moment quelconque, une touche du clavier est enfoncée, sa valeur sera affectée à la variable alphanumérique associée.

Cette fonction est particulièrement utile pour les jeux ou la saisie d'une donnée caractère par caractère.

Exemple 1

```
10 PRINT "?";
20 A$=INKEY$
30 IF A$="" THEN 20
40 IF A$=CHR$(13) THEN 70: REM** ENTER ?******
45 PRINT A$;
50 B$=B$+A$
60 GOTO 20
70 PRINT
80 PRINT B$
```

On obtient :

```
>    ?AZERTY
     AZERTY
```

Nous venons de simuler la fonction INPUT B$, mais il est possible de tester tous les caractères un à un au fur et à mesure de leur entrée ; ce qui est impossible avec INPUT.

Exemple 2

```
10 B$="RAME":CLS
20 PRINT "MOT DE PASSE:";
30 FOR I = 1 TO 4
40 A$=INKEY$: IF A$="" THEN 40
```

```
 50 IF A$ < > MID$(B$,I,1) THEN 100
 60 NEXT I
 70 END
100 PRINT  @  170,"FRAUDEUR"
110 SOUND 15,5
120 PRINT  @  170,"           "
130 SOUND 20,5
140 GOTO 100
```

– > *Si lors de l'entrée d'une lettre, celle-ci s'avère différente de celle qui lui correspond dans le mot de passe, un message clignote et une sirène hulule. Il n'y a pas d'écho sur l'écran des caractères entrés.*

Contexte : INPUT

INPUT

Entrée d'une donnée en cours d'exécution de programme. Pour une simulation de son fonctionnement, voir INKEY$. Il est possible de lui adjoindre une phrase optionnelle qui indique à l'utilisateur du programme quelle donnée est en attente. Plusieurs données peuvent être entrées avec une seule instruction INPUT.

Exemple

```
10 INPUT "NOM,PRENOM,AGE:"; N$,P$,A
20 PRINT N$
30 PRINT P$
40 PRINT A
```

On obtient :

```
– >  MINET,ROBERT,40 ENTER
     MINET
     ROBERT
     40
```

Si une donnée n'est pas compatible avec la variable associée, un message REDO est émis et l'instruction INPUT est réexécutée. Si le nombre de données entrées est supérieur à ce qu'attendait l'INPUT, seules les premières sont prises en compte et un message EXTRA IGNORED est généré.

Contexte : INKEY$

INT

Renvoie la partie entière d'un nombre quelconque.

Exemple 1

```
10 PRINT INT(4.78)
```

```
- >    4
```

Exemple 2

```
10 A = 13.45
20 EN = INT(A)
30 DE = A - EN
40 PRINT "PARTIE ENTIÈRE:";EN
50 PRINT "PARTIE DÉCIMALE:";DE
```

On obtient :

```
- >    PARTIE ENTIÈRE:13
       PARTIE DÉCIMALE:.45
```

Nous venons de simuler la fonction complémentaire de INT.

Contexte : ABS, SGN ....

LEFT$

Extrait la partie gauche d'une chaîne de caractères. Il faut spécifier le nombre de caractères à extraire.

Exemple

```
10 A$ = "BONJOUR"
20 PRINT LEFT$(A$,4)
```

On obtient :

```
- >    BONJ
```

Contexte : RIGHT$, MID$, LEN, VAL, ASC, CHR$

LEN

Donne la longueur d'une chaîne de caractères.

Exemple

```
10 A$ = "BONJOUR"
20 PRINT LEN (A$)
```

```
- >    7
```

Contexte : LEFT$, RIGHT$, MID$, VAL, ASC, CHR$

LET

Option pour l'affectation d'une valeur à une variable.

```
LET A = 10
```

est équivalent à

```
A = 10
```

LIST

Affiche sur l'écran l'ensemble du programme résident en mémoire de travail. Il est possible de ne mentionner qu'une ligne ou un groupe de lignes. Pour stopper le défilement taper SHIFT @.

Contexte : RUN, NEW, LLIST

LLIST

Même fonctionnement que LIST, mais le listing est envoyé vers l'imprimante.

LOG

Fonction logarithme népérien ou naturel. Pour plus d'informations, voir EXP.

LPRINT

Même fonctionnement que PRINT, mais le texte est envoyé vers l'imprimante.

MEM

Donne la valeur en octets de la mémoire vive libre.

MID$

Instruction permettant d'extraire n'importe quel sous-programme d'une chaîne de caractères. Deux paramètres sont nécessaires, la position du premier caractère et la longueur de la chaîne à extraire.

Exemple

```
10 A$ = "BONJOUR"
20 PRINT MID$(A$,1,2)
30 PRINT MID$(A$,2,2)
40 PRINT MID$(A$,3,5)
```

On obtient :

```
- >   BO
      ON
      NJOUR
```

Contexte : ASC, VAL, LEN, CHR$, LEFT$, RIGHT$

NEW

Effacement de la mémoire de travail. S'il y avait un programme précédemment, il est définitivement perdu.

Contexte : RUN, LIST

NEXT

Un des composants d'une boucle. Il délimite la fin de la boucle et teste si le compteur de boucle dépasse la limite finale. Si ce n'est pas le cas, l'interpréteur est renvoyé au début de la boucle, sinon le programme se déroule séquentiellement.

Contexte : FOR, STEP, TO, =

NOT

Inverse le sens d'une proposition logique.

Exemple

```
10 A = 6
20 B = 7
```

```
30 PRINT A < B
40 PRINT NOT (A < B)
```

On obtient :

```
- >     -1
        0
```

La proposition de la ligne 30 est vraie, elle est donc inversée en ligne 40 et devient fausse.

Contexte : AND, OR, >, <, =, < >, > =, < =

ON

Instruction de paramétrage des GOTO et GOSUB. Elle permet d'appliquer les règles de la programmation structurée. La ligne de branchement est fonction de la valeur de la variable paramètre de UN.

Exemple

```
10 FOR I=1 TO 3
20 ON I GOSUB 50,70,90
30 NEXT I
40 END
50 PRINT "LIGNE 50"
60 RETURN
70 PRINT "LIGNE 70"
80 RETURN
90 PRINT "LIGNE 90"
100 RETURN
```

On obtient :

```
- >    LIGNE 50
       LIGNE 70
       LIGNE 90
```

Selon la valeur de la variable, ici I, le branchement se fera au 1er, 2e ou 3e numéro.

OR

Opérateur logique OU inclusif. Permet de juxtaposer plusieurs tests à l'intérieur d'une structure de décision du type IF THEN. Un

ensemble de propositions logiques liées par OR est vrai si au moins l'une d'entre elles l'est.

Exemple

```
10 A=10
20 B=5
30 IF A=10 OR B=4 THEN PRINT "VRAI":GOTO 50
40 PRINT "FAUX"
50 IF A=2 OR B>10 THEN PRINT "VRAI":GOTO 70
60 PRINT "FAUX"
70 END
```

On obtient :

```
- >    VRAI
       FAUX
```

Contexte : AND, IF, THEN, <, >, =, < >, < =, > =

PEEK

Action directe sur la mémoire de l'ordinateur. PEEK permet de lire la valeur d'un octet de mémoire.

Son utilisation est abondamment illustrée dans la seconde partie de ce livre.

Contexte : POKE, USR, EXEC

POINT

Instruction graphique détectant la présence et la couleur d'un pavé graphique.

Exemple

```
10 CLS 0
20 SET (10,10,3)
30 PRINT POINT (10,10)
```

On obtient :

```
- >    3
```

Si aucun pavé n'est spécifié, POINT générera 0 si le fond est bleu foncé, et −1 si le fond est celui de l'affichage du texte (vert).

Contexte : SET, RESET, CLS

POKE

Il s'agit du complémentaire de la directive PEEK. Elle permet de placer un octet de valeur quelconque à un endroit spécifié de la mémoire.

Exemple

```
10 CLS
20 FOR I=0 TO 255
30 POKE 16384+I,I
40 NEXT I
```

— > *L'écran affiche la police de caractères disponible sur Alice.*

Nous avons imprimé un message directement dans la mémoire vidéo sans utiliser d'instruction PRINT.

Contexte : PEEK, USR, EXEC

PRINT

Impression d'une donnée sur l'écran. PRINT peut traiter des quantités numériques et alphanumériques. Une seule instruction permet d'afficher plusieurs données, pourvu qu'elles soient séparées par une virgule ou un point-virgule.

La virgule provoque une tabulation automatique de l'écran en deux colonnes indépendantes de seize caractères.

Le point-virgule annule le "retour chariot", et l'impression suivante sera directement contiguë.

Exemple

```
10 CLS
20 PRINT "VIRGULE","OU"
30 PRINT "POINT-VIRGULE"
40 PRINT "TELLE";"EST";
50 PRINT "LA QUESTION"
```

On obtient :

```
- >     VIRGULE OU
        POINT-VIRGULE
        TELLE EST LA QUESTION
```

Voir Chapitre 2 pour plus de détails.

Contexte : PRINT @, TAB

READ

Lecture d'un fichier interne sous forme de DATA. Si N instructions READ ont été employées, la suite va lire le N + 1ème élément du fichier interne s'il existe ; sinon un message d'erreur sera généré (OD error).

La valeur lue est affectée dans la variable mentionnée avec READ, il faut donc toujours vérifier si la donnée et la variable sont du même type.

Une fois arrivé en fin des données, il faut remettre le pointeur des DATA à zéro à l'aide de RESTORE. Dans ce cas, la lecture recommencera au début du fichier interne.

Exemple

```
10 DATA 1,2,3,4,5,6
20 FOR I=1 TO 3
30 READ NO
40 NEXT I
50 RESTORE
60 PRINT NO

- >    3
```

Ce programme lit le troisième élément du fichier interne, l'affecte à la variable NO, puis laisse le fichier dans l'état initial. On remarque qu'il a été nécessaire de lire tous les nombres intermédiaires des DATA.

Comme pour INPUT, il est possible de mettre plusieurs variables dans une instruction READ.

Contexte : DATA, RESTORE

REM

Signale à l'interpréteur que la ligne de programme est un commentaire et ne doit pas être analysée. Les commentaires n'apparaîtrons qu'au cours du listage du programme.

Contexte : LIST

RESET

Instruction graphique destinée à effacer un pavé sur l'écran. Celui-ci devient bleu foncé, couleur du fond de dessin.

Exemple

```
10 CLS 0
20 SET(10,10,5)
30 FOR I=1 TO 1000
40 NEXT I
50 RESET(10,10)
60 GOTO 20
```

– > *Un petit carré blanc va clignoter indéfiniment aux coordonnées horizontale 10, et verticale 10.*

Contexte : SET, POINT, CLS

RESTORE

Remet le pointeur du fichier interne à sa position initiale.

Contexte : READ, DATA

RETURN

Cette directive va dérouter l'interpréteur BASIC vers l'endroit du dernier GOSUB rencontré.

Contexte : GOSUB, ON GOSUB

RIGHT$

Fonction de traitement de chaîne de caractères. Cette instruction extrait la partie droite d'une chaîne alphanumérique, sur une longueur spécifiée par le paramètre.

Exemple

```
10 A$ = "MARINADE"
20 PRINT RIGHT$(A$,6)
```

On obtient :

```
- >     RINADE
```

Contexte : STR$, LEN, VAL, MID$, LEFT$, CHR$

RND

Générateur de nombre aléatoire. Selon la valeur de l'opérande, le nombre aura les valeurs suivantes.

RND (0) - > nombre N tel que 0 < N < 1
RND (N) - > nombre compris entre 1 et N au sens large.

Exemple

```
10 PRINT "LANCE D'UN DE"
20 A = RND (6)
30 PRINT "LE RÉSULTAT EST:";A
40 FOR J = 1 TO 1000:NEXT J
50 GOTO 10
```

L'écran affiche une infinité de nombres entre 1 et 6 choisis au hasard.

Contexte : INT, ABS, SGN

RUN

Ordonne le début de l'exécution d'un programme BASIC résident en mémoire. RUN réinitialise toutes les variables et tableaux à 0 ou "".

Il est possible de commencer l'exécution au numéro de ligne mentionné si on utilise un opérande.

Exemple

```
10 PRINT "LIGNE 10"
20 PRINT "LIGNE 20"
RUN 20 ENTER
```

On obtient :

```
- >     LIGNE 20
```

L'exécution commence ligne 20.

Contexte : LIST, NEW

SET

Affiche un pavé graphique de la couleur spécifiée, aux coordonnées indiquées. X varie de 0 à 63 et Y de 0 à 31. La couleur, elle, est codée de 0 à 8.

Exemple

```
10 X=10
20 Y=12
30 C=5
40 SET (X,Y,C)
```

On obtient :

```
- >     Pavé blanc en 10 et 12
```

Contexte : CLS, RESET, POINT

SGN

Teste le signe d'une quantité numérique : 1 si le nombre est positif ; 0 s'il est nul ; – 1 s'il est négatif. Il peut être utilisé dans des calculs ou comme test pour déterminer si un nombre est nul.

Exemple

```
10 INPUT "NOMBRE À TESTER";A
20 IF SGN (A) THEN PRINT "TEST POSITIF":GOTO 40
30 PRINT "TEST NÉGATIF"
40 END
```

On obtient :

```
- >     NOMBRE À TESTER?0 ENTER
       TEST NÉGATIF
```

```
NOMBRE À TESTER? – 3 ENTER
TEST POSITIF
```

Contexte : ABS, RND, INT

SIN

Fonction trigonométrique SINUS. L'argument doit être donné en RADIANS.

Pour plus de détails voir la fonction COS.

Contexte : COS, TAN, EXP, LOG ....

SKIPF

Contrôle des programmes sur bande magnétique. Positionne la lecture en fin de l'enregistrement mentionné.

Contexte : CSAVE, CSAVE*, CLOAD, CLOAD*

SOUND

Génération d'un son. Les deux paramètres précisent respectivement la fréquence et la durée du son. Leurs valeurs varient entre 1 et 255.

Voir INKEY$ pour un exemple d'utilisation.

SQR

Fonction racine carrée, attention aux nombres négatifs.

Exemple

```
10 PRINT SQR (4)
20 PRINT SQR (16)
```

On obtient :

```
– >   2
      4
```

Contexte : COS, SIN, TAN, EXP, LOG ....

STEP

Instruction optionnelle de boucle. STEP précise le pas de la boucle. C'est-à-dire la valeur ajoutée au compteur de boucle à chaque passage.

Lorsque le pas de la boucle n'est pas précisé, sa valeur par défaut est 1.

Exemple

```
10 FOR I=10 TO 0 STEP -1
20 CLS
30 PRINT I
40 SOUND 50,5
50 NEXT I
60 CLS
70 PRINT TABL(5),"IGNITION"
```

– > Compte à rebours sonorisé avec la mention IGNITION, en fin de programme.

Contexte : FOR, TO, NEXT

STOP

Interruption momentanée d'un programme. La poursuite de l'exécution à partir du point d'arrêt est réalisée par CONT. Le contenu des variables et tableaux est préservé. Cette instruction peut être utile lors de la mise au point d'un programme.

Contexte : END, LIST, RUN, CONT

STR$

Directive complémentaire de VAL. Traduit une valeur numérique en une chaîne de caractères sans signification pour la machine.

Attention : "E" et "." sont considérés comme des symboles numériques.

Exemple

```
10 A=1045
20 A$=STR$(A)
```

```
30 PRINT A+1
40 PRINT A$+"1"
```

On obtient :

```
- >    1046
       10451
```

Contexte : VAL, LEN, CHR$, LEFT$, RIGHT$, MID$

TAB

Tabulation horizontale de l'écran. En réalité, elle émet autant de caractères espace qu'il est mentionné dans l'opérande. Le séparateur avec la donnée d'impression est le point-virgule.

Exemple

```
10 FOR I=1 TO 3
20 PRINT TAB(I);I
30 NEXT I
```

On obtient :

```
- >    1
       2
       3
```

Contexte : PRINT, PRINT @

TAN

Fonction trigonométrique TANGENTE. L'opérande doit être en *radian* (voir GOSUB pour un exemple de conversion). Mathématiquement, TANGENTE = SINUS/COSINUS. Pour plus de détails, voir COS.

Contexte : SIN, COS, EXP, LOG ....

THEN

Délimiteur de test dans une structure de décision IF THEN. Si le test qui précède THEN s'avère vrai, l'interpréteur exécutera les instructions placées après THEN. Sinon, il se branchera à la ligne de programme suivante.

Voir IF pour plus de détails.

Contexte : IF, =, >, <, < >, > =, < =

TO

Sert à préciser la valeur finale d'un compteur de boucle dans une structure FOR NEXT.

Contexte : FOR, STEP, NEXT

USR

Appel d'un sous-programme en langage machine avec transmission de paramètres.

Contexte : PEEK, POKE, EXEC

VAL

Fonction complémentaire de STR$. Convertit une donnée alphanumérique en un nombre directement utilisable par la machine.

Si un symbole non numérique est rencontré, la conversion s'arrête, ou égale 0 si le symbole est le premier de la chaîne de caractères. E et "." sont considérés comme des symboles numériques.

Exemple

```
10 A$ = "1045"
20 A = VAL(A$)
30 PRINT A + 1
40 PRINT A$ + "1"
```

On obtient :

```
- >    1046
       10451
```

Contexte : STR$, RIGHT$, LEFT$, MID$, LEN, CHR$

VARPTR

Donne l'adresse du premier emplacement d'une variable.

Voir les chapitres sur le traitement interne de la machine.

Contexte : POKE, PEEK, USR, EXEC

Elévation à la puissance. 3 ↑ 5 = 3*3*3*3*3. Très utile pour certains calculs mathématiques, elle est toutefois relativement lente et peut introduire un facteur d'erreur discernable dans certains cas.

C'est pourquoi il est conseillé, toutes les fois où cela est possible, d'écrire l'opération avec des multiplications.

L'exponentiation est la plus prioritaire des opérations élémentaires.

Exemple

*Mieux vaut écrire 5*5 que 5 ↑ 2 ; par contre 5 ↑ 16 est plus judicieux que 5*5*5*5*5*5*5*5*5*5*5*5*5*5*5*5*

Contexte : +, −, *, /

Séparateur d'instructions sur une même ligne de programme. Mais l'ensemble des instructions sur cette ligne ne doit pas dépasser une longueur de 127 caractères.

ANNEXE 4

TABLE DES INSTRUCTIONS DU 6803

ABA	•	•	•	•	•	2	•
ABX	•	•	•	•	•	3	•
ADC	•	2	3	4	4	•	•
ADD	•	2	3	4	4	•	•
ADDD	•	4	5	6	6	•	•
AND	•	2	3	4	4	•	•
ASL	2	•	•	6	6	•	•
ASLD	•	•	•	•	•	3	•
ASR	2	•	•	6	6	•	•
BCC	•	•	•	•	•	•	3
BCS	•	•	•	•	•	•	3
BEQ	•	•	•	•	•	•	3
BGE	•	•	•	•	•	•	3
BGT	•	•	•	•	•	•	3
BHI	•	•	•	•	•	•	3
BIT	•	2	3	4	4	•	•
BLE	•	•	•	•	•	•	3
BLS	•	•	•	•	•	•	3
BLT	•	•	•	•	•	•	3
BMI	•	•	•	•	•	•	3
BNE	•	•	•	•	•	•	3
BPL	•	•	•	•	•	•	3
BRA	•	•	•	•	•	•	3
BSR	•	•	•	•	•	•	6
BVC	•	•	•	•	•	•	3
BVS	•	•	•	•	•	•	3
CBA	•	•	•	•	•	2	•
CLC	•	•	•	•	•	2	•
CLI	•	•	•	•	•	2	•
CLR	2	•	•	6	6	•	•
CLV	•	•	•	•	•	2	•
CMP	•	2	3	4	4	•	•
COM	2	•	•	6	6	•	•
CPX	•	4	5	6	6	•	•
DAA	•	•	•	•	•	2	•
DEC	2	•	•	6	6	•	•
DES	•	•	•	•	•	3	•
DEX	•	•	•	•	•	3	•
EOR	•	2	3	4	4	•	•
INC	2	•	•	6	6	•	•
INS	•	•	•	•	•	3	•

INX	•	•	•	•	•	3	•
JMP	•	•	•	3	3	•	•
JSR	•	•	5	6	6	•	•
LDA	•	2	3	4	4	•	•
LDD	•	3	4	5	5	•	•
LDS	•	3	4	5	5	•	•
LDX	•	3	4	5	5	•	•
LSR	2	•	•	6	6	•	•
LSRD	•	•	•	•	•	3	•
MUL	•	•	•	•	•	10	•
NEG	2	•	•	6	6	•	•
NDP	•	•	•	•	•	2	•
ORA	•	2	3	4	4	•	•
PSH	3	•	•	•	•	•	•
PSHX	•	•	•	•	4	•	•
PUL	4	•	•	•	•	•	•
PULX	•	•	•	•	•	5	•
ROL	2	•	6	6	•	•	•
ROR	2	•	6	6	•	•	•
RTI	•	•	•	•	•	10	•
RTS	•	•	•	•	•	5	•
SBA	•	•	•	•	•	2	•
SBC	•	2	3	4	4	•	•
SEC	•	•	•	•	•	2	•
SEI	•	•	•	•	•	2	•
SEV	•	•	•	•	•	2	•
STA	•	•	3	4	4	•	•
STD	•	•	4	5	5	•	•
STS	•	•	4	5	5	•	•
STX	•	•	4	5	5	•	•
SUB	•	2	3	4	4	•	•
SUBD	•	4	5	6	6	•	•
SWI	•	•	•	•	•	12	•
TAB	•	•	•	•	•	2	•
TAP	•	•	•	•	•	2	•
TBA	•	•	•	•	•	2	•
TPA	•	•	•	•	•	2	•
TST	2	•	•	6	6	•	•
TSX	•	•	•	•	•	3	•
TXS	•	•	•	•	•	3	•
WAI	•	•	•	•	•	9	•

ANNEXE 5

LISTE PAR CODE OPÉRATION HEXADÉCIMAL (IJ)

I / J	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		NOP			LSR D	ASL D	TAP	TPA	INX	DEX	CLV	SEV	C,C	SEC	CLI	SEI
1	SBA	CBA					TAB	TBA		DAA		ABA				
2	BRA		BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
3	TSX	INS	PUL A	PUL B	DES	TXS	PSH A	PSH B	PUL X	RTS	ABX	RTI	PSH X	MUL	WAI	SWI
4	NEG A			COM A	LSR A		ROR A	ASR A	ASL A	ROL A	DEC A		INC A	TST A		CLR A
5	NEG B			COM B	LSR B		ROR B	ASR B	ASL B	ROL B	DEC B		INC B	TST B		CLR B
6	NEG INDX			COM INDX	LSR INDX		ROR INDX	ASR INDX	ASL INDX	ROL INDX	DEC INDX		INC INDX	TST INDX	JMP INDX	CLR INDX
7	NEG ABS			COM ABS	LSR ABS		ROR ABS	ASR ABS	ASL ABS	ROL ABS	DEC ABS		INC ABS	TST ABS	JMP ABS	CLR ABS
8	SUB A IMM	CMP A IMM	SBC A IMM	SUB D	AND A IMM	BIT A IMM	LDA A IMM		EOR A IMM	ADC A IMM	ORA A IMM	ADD A IMM	CPX IMM	BSR	LDS IMM	
9	SUB A PAGE Z	CMP A PAGE Z	SBC A PAGE Z	SUB D	AND A PAGE Z	BIT A PAGE Z	LDA A PAGE Z	STA A PAGE Z	EOR A PAGE Z	ADC A PAGE Z	ORA A PAGE Z	ADD A PAGE Z	CPX PAGE Z		LDS PAGE Z	STS PAGE Z
A	SUB A INDX	CMP A INDX	SBC A INDX	SJB D	AND A INDX	BIT A INDX	LDA A INDX	STA A INDX	EOR A INDX	ADC A INDX	ORA A INDX	ADD A INDX	CPX INDX	JSR INDX	LDS INDX	STS INDX
B	SUB A ABS	CMP A ABS	SBC A ABS	SUB D	AND A ABS	BIT A ABS	LDA A ABS	STA A ABS	EOR A ABS	ADC A ABS	ORA A ABS	ADD A ABS	CPX ABS	JSR ABS	LDS ABS	STS ABS
C	SUB B IMM	CMP B IMM	SBC B IMM	ADD D	AND B IMM	BIT B IMM	LDA B IMM		EOR B IMM	ADC B IMM	ORA B IMM	ADD B IMM	LDA D		LDX IMM	
D	SUB B PAGE Z	CMP B PAGE Z	SBC B PAGE Z	ADD D	AND B PAGE Z	BIT B PAGE Z	LDA B PAGE Z	STA B PAGE Z	EOR B PAGE Z	ADC B PAGE Z	ORA B PAGE Z	ADD B PAGE Z	LDA D	STD	LDX PAGE Z	STX PAGE Z
E	SUB B INDX	CMP B INDX	SBC B INDX	ADD D	AND B INDX	BIT B INDX	LDA B INDX	STA B INDX	EOR B INDX	ADC B INDX	ORA B NDX	ADD B INDX	LDA D	STD	LDX INDX	STX INDX
F	SUB B ABS	CMP B ABS	SBC B ABS	ADD D	AND 3 ABS	BIT B ABS	LDA B ABS	STA B ABS	EOR B ABS	ADC B ABS	ORA B ABS	ADD B ABS	LDA D	STD	LDX ABS	STX ABS

ANNEXE 6

TABLES DE BRANCHEMENTS RELATIFS

BRANCHEMENTS RELATIFS EN AVANT

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

BRANCHEMENTS RELATIFS EN ARRIERE

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

ANNEXE 7

TABLE DE CONVERSION HEXADÉCIMALE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	106	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

TABLE DES MATIÈRES

4
STRUCTURATION

5
MATHÉMATIQUES

6
DESSIN ET MUSIQUE

7
NOTIONS DE MÉMOIRE

8
NOTIONS DE LANGAGE MACHINE

ANNEXES