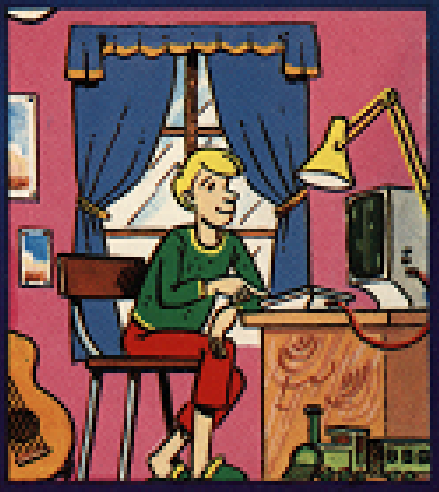
TOUT SUR LE MO 5

J.F. Bieber, A. Perbost, G. Renucci

BASIC
GRAPHISME
ASSEMBLEUR



TH. DUVAL

Edimicro

TOUT SUR LE MO 5
Basic, Graphisme, Assembleur

Collection animée par Benoît de MERLY.

MO 5 est une marque déposée de Thomson.

Première édition

ISBN : 2-904457-22-4

EDIMICRO

DÉPARTEMENT ÉDITIONS DE F.D.S. SARL

121-127, avenue d'Italie, 75013 Paris

TOUT SUR LE MO 5

par

Jean-François BIEBER
Alain PERBOST
Gilles RENUCCI

Collection « Ordinateurs personnels »

Edimicro

Page blanche

Sommaire

Page blanche

CHAPITRE **1**

Initiation au BASIC par le jeu

1.1 INTRODUCTION

Découvrir la micro-informatique, c'est pénétrer dans un monde fascinant. Un peu d'expérience et de pratique vous permettront d'en entreprendre l'exploration avec votre Thomson MO5.

Que vous soyez passionné de mathématiques, de gestion, de graphisme, de jeux ou même de musique, vous pourrez pratiquer sans contrainte votre discipline préférée. Il est impossible de dresser une liste exhaustive des innombrables applications de la micro-informatique ; les seules limites que vous rencontrerez seront celles de votre imagination...

Le but de ce chapitre est de vous montrer quelques-uns des principes fondamentaux de la programmation, et leur mise en œuvre pour la création d'un jeu en BASIC. Vous initier tout en vous divertissant, telle est donc notre ambition.

1.2 PROGRAMMATION ET LANGAGE BASIC

1.2.1 Généralités

Les progrès de l'informatique et le nombre croissant de ses applications ont contribué à créer au niveau du grand public le mythe de l'Ordinateur.

Il est bien sûr impressionnant de voir un ordinateur jouer merveilleusement bien aux échecs ou résoudre en une fraction de seconde des problèmes mathématiques très complexes. Admirer cet ordinateur, c'est toutefois commettre une erreur aussi grossière que d'admirer un violon et non le virtuose qui en joue.

L'ordinateur n'est qu'un instrument dans les mains de l'homme. De même qu'un violon prend vie dans les mains d'un musicien, un ordinateur ne doit son « intelligence » qu'à son programmeur. Livré à lui-même, un ordinateur n'est qu'un fatras de circuits intégrés. Seule l'action de l'homme peut permettre à cet assemblage de réaliser une fonction donnée, jouer aux échecs par exemple.

Il est fondamental d'insister sur ce fait, afin que vous compreniez bien que la mise en œuvre des possibilités de votre micro-ordinateur MO5 dépend entièrement de vous, et de vous seul. C'est à vous de le programmer intelligemment pour lui faire exécuter exactement ce que vous désirez. Si vous voulez faire jouer votre MO5 aux échecs, il faut que vous écriviez le programme qui lui permettra de le faire. Lorsque vous verrez votre micro-ordinateur jouer, vous saurez alors que c'est votre programme qui est en train de « tourner », et vous qui jouez, en définitive, par son intermédiaire.

Cette totale dépendance de la machine envers son programmeur a bien sûr quelques inconvénients :

- Si vous omettez ne serait-ce qu'une virgule, un espace ou même un guillemet, votre MO5 sera totalement désorienté, incapable de comprendre ni d'exécuter ce que vous lui demandez.

- Vous ne pourrez jamais accuser votre MO5 de se tromper car il en est incapable. Les erreurs ne peuvent provenir que de vous, de même que les fausses notes sont dues au musicien et non pas à son instrument. Inutile donc d'être de mauvaise foi et injuste avec votre

micro-ordinateur, il n'y est pour rien et de plus tout cela lui est égal...

Ces « inconvénients » sont mineurs en comparaison des avantages que procure la maîtrise d'un micro-ordinateur. Au gré de votre fantaisie, le Thomson MO5 se transformera en un agréable et infatiguable compagnon de jeux, en un mathématicien modèle ou même en un dessinateur doué ! La programmation vous ouvre des horizons infinis.

Plus rien ne vous limitera quand vous saurez qu'il est impossible d'endommager un micro-ordinateur par une mauvaise programmation. Des messages d'erreurs vous signaleront vos fautes éventuelles. Le pire qui puisse vous arriver est d'être obligé de réinitialiser le système, ce qui n'est tout de même pas dramatique.

Alors n'hésitez pas à taper sur le clavier tout ce qui vous passe par la tête. Dès qu'une idée vous vient, essayez donc de la réaliser. De nombreux exemples sont donnés dans ce livre, utilisez-les et améliorez-les.

Nous espérons que bientôt vous éprouverez autant de plaisir que nous à programmer, et que vous saurez très vite exploiter au mieux les énormes possibilités de votre MO5.

1.2.2 Programmation et langage BASIC

*** Qu'est-ce donc qu'un programme ?**

Le paragraphe précédent a mis l'accent sur l'importance de la programmation. Il reste maintenant à définir la notion de programme.

Un programme est une suite d'instructions que l'utilisateur souhaite voir exécuter dans un ordre déterminé pour réaliser une certaine fonction, jouer aux échecs par exemple.

*** Comment peut-on programmer un micro-ordinateur ?**

Un ordinateur ne raisonne qu'en termes de courant (ou de tension) ; le courant passe (niveau haut) : 1, ou ne passe pas (niveau bas) : Ø. Il existe dans un langage propre à la machine (que l'on appelle « langage machine ») qui permet de donner, par une suite de 1 et de Ø, des ordres exécutables par l'ordinateur. Cette façon de procéder s'avère généralement longue et complexe.

Des langages évolués de programmation ont été développés afin de permettre une programmation simple et rapide. Des interpréteurs ou des compilateurs traduisent ensuite les programmes écrits en langage évolué, en langage machine exécutable par l'ordinateur.

Le Thomson MO5 utilise le langage de programmation BASIC (*B*eginners *A*ll-purpose *S*ymbolic *I*nstruction *C*ode). Mis au point au cours des années 60, ce langage est devenu au fil des ans de plus en plus puissant et s'est répandu dans le monde entier. Votre MO5 dispose de l'une des plus puissantes versions actuelles du BASIC. Il s'agit du BASIC niveau 5 Microsoft, développé spécialement pour le TO7 (version 1.Ø) puis adapté au MO5.

*** Comment apprend-on un langage de programmation ?**

L'apprentissage du BASIC est très semblable à l'apprentissage d'une langue étrangère. Il est nécessaire d'acquérir un certain vocabulaire et de connaître les règles de syntaxe du langage.

Ne vous faites pas de souci, le BASIC est un langage très facile à apprendre et très simple à utiliser. Son vocabulaire n'est pas très étendu ; les règles de grammaire sont précises, strictes et ne comportent pas les multiples exceptions qui rendent si difficile l'étude d'une langue étrangère.

Le chapitre suivant de ce livre donne une analyse détaillée du BASIC disponible sur le Thomson MO5. Vous pouvez vous y reporter pour des détails précis, ce qui suit n'ayant pas la prétention d'être plus qu'une présentation générale du BASIC

1.2.3 Démarrage du BASIC

Mettez sous tension le téléviseur et le MO5.

Le message « MO5 BASIC 1.Ø
(c) Microsoft 1984

apparaît alors sur l'écran.

Le symbole « OK » apparaît aussi, indiquant que l'interpréteur BASIC est prêt à fonctionner. Votre MO5 est prêt à servir en mode calculateur ou en mode programme.

1.3 MODE CALCULATEUR - MODE PROGRAMME

1.3.1 Mode calculateur

Le mode calculateur, appelé aussi mode immédiat ou mode commande, ressemble beaucoup au mode de fonctionnement d'une calculatrice. Ce mode permet en effet de demander à votre MO 5 d'éxécuter immédiatement un calcul, une instruction ou une commande.

*** Tapez, par exemple, PRINT "BONJOUR" puis pressez la touche [ENTREE] : le caractère " s'obtient en pressant simultanément la touche jaune et [2̈].**

Vous remarquez que le message BONJOUR apparaît aussitôt sur l'écran de votre téléviseur. Que s'est-il passé ?

Le MO5 dispose d'un interpréteur BASIC qui lui permet de décoder et d'exécuter les ordres qui lui sont donnés en BASIC. Le symbole OK signifie que l'interpréteur BASIC est actif.

A chaque fois que vous avez tapé l'un des caractères de l'instruction PRINT "BONJOUR", le MO5 a rangé ce caractère en mémoire tout en le faisant apparaître sur l'écran du téléviseur.

Lorsque vous avez pressé la touche [ENTREE], l'interpréteur BASIC s'est mis au travail. Ressortant de la mémoire, les uns après les autres, les caractères entrés précédemment, l'interpréteur a reconnu la commande PRINT qui lui ordonne d'afficher un message sur l'écran du téléviseur (PRINT signifie imprimer en anglais). Il a ensuite reconnu entre les guillemets le message à afficher.

Ayant ainsi totalement décodé l'ordre donné, l'interpréteur BASIC a pu passer à la phase d'exécution, ce qui explique l'apparition immédiate du message BONJOUR sur l'écran.

Le symbole OK signifie que l'interpréteur BASIC est à nouveau prêt pour recevoir et exécuter tout calcul ou instruction.

Remarque : le MO5 possède un clavier orienté BASIC. Pour taper PRINT, vous pouvez appuyer simultanément sur la touche

BASIC et sur la touche I. Le résultat est alors équivalent à celui obtenu en tapant PRINT lettre à lettre.

* Exécutons quelques calculs élémentaires en mode calculateur.

Par exemple :

PRINT 3 + 2 [ENTREE]

L'interpréteur BASIC reconnaît l'ordre d'afficher immédiatement sur l'écran du téléviseur le résultat de l'addition des deux nombres 3 et 2.

Ce résultat apparaît donc aussitôt : 5 suivi du classique OK.

PRINT ((3 + 2) * 5 ↑ 2) / 7 − 1 [ENTREE]

donne sur l'écran

16.8571

Le MO5 permet ainsi de calculer la valeur d'expressions mathématiques très complexes. Pour des détails plus complets sur les opérateurs arithmétiques et leur hiérarchie, reportez-vous au chapitre suivant.

Remarque : l'opérateur ↑, élévation à la puissance (X ↑ Y signifie X à la puissance Y), est obtenu en pressant simultanément la touche jaune et [â].

1.3.2 Mode programme

Le deuxième mode de fonctionnement du MO5 est le mode programme. Ce mode permet, comme son nom l'indique, de mettre en mémoire des programmes écrits en BASIC.

*** Ecriture d'un programme**

Une ligne de programme est constituée par un numéro de ligne suivi d'une ou de plusieurs instructions.

Sur le MO5 le numéro de ligne devra obligatoirement être un nombre entier compris entre Ø et 63 999. La ligne ne devra pas comporter plus de 255 caractères.

— Tapez

1Ø PRINT "BONJOUR" [ENTREE]

Cette fois le message BONJOUR n'apparaît pas sur l'écran du téléviseur. Contrairement à ce qui s'était passé en mode calculateur, la pression de la touche [ENTREE] n'a pas mis l'interpréteur BASIC en action. L'introduction du numéro de ligne 1Ø a en effet indiqué au MO5 qu'il s'agissait d'une ligne de programme devant être stockée en mémoire et non pas d'une instruction exécutable immédiatement.

— Tapez

RUN [ENTREE]

Le message BONJOUR apparaît alors sur l'écran du téléviseur. La commande RUN demande à l'interpréteur BASIC d'exécuter le programme qui se trouve en mémoire. Ce programme est constitué pour l'instant par la seule ligne 1Ø PRINT "BONJOUR". Cette ligne est donc décodée et exécutée.

Le symbole OK réapparaît, indiquant ainsi la fin de l'exécution du programme et la disponibilité du MO5 pour recevoir de nouvelles commandes.

— Tapez

```
 5 INPUT "QUEL EST VOTRE PRENOM"; PS
   [ENTREE]
15 GOTO 5 [ENTREE]
```

— Tapez

LIST [ENTREE]

Cette commande ordonne au MO5 d'afficher sur l'écran le programme contenu dans la mémoire.

Vous voyez alors :

```
 5 INPUT "QUEL EST VOTRE PRENOM"; P$
1Ø PRINT "BONJOUR"!
15 GOTO 5
```

Bien que mises en mémoire après la ligne 1Ø, les lignes 5 et 15 sont rangées par le MO5 dans l'ordre croissant des numéros de lignes.

— Faites maintenant

RUN

Vous avez certainement compris que la pression de la touche

[ENTREE] est indispensable pour valider une ligne de programme ou une instruction. Nous ne vous indiquerons donc plus explicitement qu'il faut presser cette touche mais n'oubliez pas pour autant de la faire !

La commande RUN, ayant pour effet de lancer l'exécution du programme, la question "QUEL EST VOTRE PRENOM ?" s'affiche sur l'écran.

Remarque : le point d'interrogation est placé automatiquement après la question car le délimiteur utilisé est un point-virgule. Pour éviter ce point d'interrogation, inutile dans certains cas, il suffit de mettre une virgule en tant que délimiteur.

Répondez à la question posée (sans oublier de presser [ENTREE] pour valider votre réponse). Le MO5 vous dit alors poliment bonjour puis vous redemande aussitôt votre prénom.

*** Arrêt de programme**

Le programme précédent ne peut s'arrêter tout seul en raison du saut inconditionnel GOTO de la ligne 15. C'est donc à l'utilisateur d'en arrêter l'exécution.

En pressant la touche [STOP], on peut "stopper" momentanément l'exécution d'un programme. La relance du programme se fera en appuyant sur une quelconque des touches du clavier à l'exception de [CNT], la touche jaune et bien sûr [STOP].

On peut aussi presser simultanément les touches [CNT] et [C]. Dans ce cas le message BREAK ON n° de ligne apparaît. La relance du programme n'est alors possible que par la commande RUN ou la commande CONT (abréviation de CONTinuer).

Avant de passer à la suite, arrêtez donc l'exécution du programme en pressant simultanément [CNT] et [C].

*** Modification du programme**

Vous aimeriez peut-être que le MO5 vous dise bonjour en vous appelant par votre prénom. Il faut pour cela modifier la ligne 1Ø pour la transformer en :

```
1Ø PRINT "BONJOUR "; P$
```

Plusieurs méthodes sont possibles :

— Tapez simplement

```
1Ø PRINT "BONJOUR "; P$
```

Cela aura pour effet d'« écraser » l'ancienne ligne 1Ø par la nouvelle.

— Vous pouvez aussi utiliser l'éditeur « plein écran » du MO5.

Les quatre touches [→], [←], [↑], [↓], vous permettent de placer le curseur sur la ligne à modifier et de l'amener exactement là où des modifications doivent être faites.

Amenez ainsi le curseur sous le guillemet qui suit la lettre R. Pressez ensuite la touche [INS], ce qui a pour effet d'insérer un espace entre le R et le guillemet, comme cela était voulu.

Déplacez ensuite le curseur jusqu'à la fin de la ligne et tapez ; P$. N'oubliez pas de valider la ligne en pressant la touche [ENTREE].

— Faites maintenant LIST et vérifiez que le programme est maintenant conforme à vos désirs.

— Vous pouvez en lancer l'exécution avec RUN. Si un message d'erreur survient (ERROR 2 par exemple), corrigez la ligne incriminée à l'aide des méthodes exposées ci-dessus.

Comme moyen de correction, vous disposez également de la touche [EFF] qui permet de supprimer le caractère situé au-dessus du curseur.

Remarque : si vous maintenez la pression sur une touche pendant plus d'une seconde environ, il y a répétition automatique du caractère donné par cette touche. Cela peut être mis à profit pour le déplacement du curseur.

*** Insertion de lignes — supression de lignes**

— Une bonne habitude à prendre dès le début est de numéroter les lignes d'un programme que l'on est en train d'écrire, de dix en dix par exemple. Cela permet d'insérer par la suite de nouvelles lignes dans le programme.

Le programme précédent est :

```
 5 INPUT "QUEL EST VOTRE PRENOM"; P$
1Ø PRINT "BONJOUR "; P$
15 GOTO 5
```

On peut donc insérer :

```
 7 PRINT
12 PRINT
```

Faites LIST et vous aurez alors :

```
 5 INPUT "QUEL EST VOTRE PRENOM"; P$
 7 PRINT
1Ø PRINT "BONJOUR"; P$
12 PRINT
15 GOTO 5
```

Les deux lignes rajoutées ont pour intérêt d'aérer la présentation sur l'écran du téléviseur.

— Si vous désirez voir ce programe ne s'exécuter qu'une fois, la ligne 15 devient totalement inutile. Vous pouvez la supprimer en tapant simplement 15 puis [ENTREE]. La suppression d'une ligne se fait simplement en tapant son numéro puis [ENTREE].

1.4 PRINCIPALES INSTRUCTIONS BASIC

1.4.1 Constantes et variables

*** Constantes**

Le BASIC permet d'utiliser des constantes numériques et des constantes « chaînes ».

Les constantes numériques peuvent être de type entier, réel ou double précision mais ce n'est pas fondamental pour une première approche. Des détails plus précis (notations, plage de variations, etc.) seront données dans le chapitre suivant.

Exemple : 18 est une constante réelle.

Une constante « chaîne » est une suite de caractères encadrée par des guillemets.

Exemple : "BONJOUR" est une constante « chaîne ».

*** Variables**

Une variable est un emplacement particulier de la mémoire auquel le programmeur peut donner le nom qu'il désire. Cet emplacement peut voir son contenu « varier », d'où le nom de variable.

Les noms de variables peuvent comporter de 1 à 255 caractères, le premier caractère étant obligatoirement une lettre, les autres peuvent être des lettres ou des chiffres. Cela permet de donner aux variables des noms « parlants » qui permettent de savoir ce que représente exactement la variable considérée.

Dans un jeu, par exemple, il est parfois nécessaire de créer une variable SCORE qui retiendra le résultat du joueur, une variable RECORD qui contiendra le meilleur score réalisé, etc.

SCORE et RECORD sont donc des noms possibles de variables.

2TAB est impossible car ce nom commence par un chiffre alors que TAB2 est possible.

Un nom de variable ne doit être ni un mot-clé du langage ni commencer par un mot réservé du langage. Ainsi le nom TOTO est impossible car commençant par TO (instruction FOR-TO-NEXT).

Seuls les quinze premiers caractères du nom d'une variable sont réellement pris en compte. Deux variables dont les noms commencent par quinze caractères identiques seront considérés par le BASIC comme identiques.

MISE
MISERE } sont des noms de variables différentes.

ABCDEFGHIJKLMNOX
ABCDEFGHIJKLMNOZ } représentent la même variable.

Les variables peuvent avoir, comme les constantes, différents types : entier, réel et chaîne. Retenons surtout pour l'instant les variables « chaînes » dont le type est indiqué par le caractère $, placé à la fin du nom de la variable. Ainsi A$, REPONSE$, sont des noms de variables « chaînes ».

*** Tableaux**

Un tableau est un ensemble de variables de même type, référencées par un nom unique.

Chaque élément du tableau est représenté par le nom du tableau suivi de un ou plusieurs indices (il faut autant d'indices qu'il y a de dimensions au tableau).

Exemples : A(2) est un élément d'un tableau à une seule dimension.

T(3,5) est un élément d'un tableau T à deux dimensions.

Vous pourrez utiliser des tableaux ayant de 1 à 255 dimensions, chacun des indices pouvant prendre des valeurs de Ø à 32 767 (dans la mesure où la mémoire disponible le permettra).

Pour des tableaux ayant au plus deux dimensions avec des indices ne dépassant pas 1Ø, il n'est pas nécessaire de déclarer ces tableaux. Cela est obligatoire pour des tableaux plus importants. La déclaration d'un tableau se fait à l'aide de l'instruction DIM.

Exemples : 1Ø DIM A(2Ø) déclare le tableau A à une dimension et pouvant contenir 21 éléments A(Ø), A(1),..., A(2Ø).

Remarque : il est fortement conseillé de déclarer les tableaux que l'on utilise, même lorsque cela n'est pas obligatoire. Il est en effet dangereux de s'exposer à des fautes par simple paresse !

1.4.2 Instruction d'affectation

L'instruction d'affectation = permet d'attribuer à une variable une valeur donnée.

Exemples :

A = 1Ø donne à la variable A la valeur 1Ø

P$ = "BONJOUR" donne à la variable « chaîne » la valeur "BONJOUR".

Autrefois le BASIC possédait l'instruction d'affectation LET.

Mais les informaticiens, étant en général de grands paresseux (est-ce un défaut... ?), ont estimé qu'il n'était plus nécessaire d'indiquer LET pour exécuter une affectation. Cela a malheureusement pour effet de conduire à des expressions du type I = I + 1. D'un point de vue purement mathématique c'est bien sûr très choquant car I = I + 1 entraîne 1 = Ø, ce qui est une absurdité.

En fait il faut se souvenir que le LET est sous-entendu et que le = signifie, dans ce cas, affectation. I = I + 1 signifie simplement que la variable I est incrémentée d'une unité (nouvelle valeur de I = ancienne valeur de I + 1).

Exemple : A = 2 * A + 1 est totalement différent de A = −1.

```
1Ø A = Ø
2Ø A = 2 * A + 1
3Ø PRINT "A1"; A
4Ø A = −1
5Ø PRINT "A2"; A
6Ø END
```

L'exécution de ce petit programme fait apparaître sur l'écran du téléviseur

```
A1      1
A2      −1

OK
```

Il eût été judicieux de créer un symbole spécial pour l'instruction d'affectation plutôt que l'utiliser le signe égal. Certains langages l'ont fait et permettent d'écrire, par exemple, I ← I + 1 ou I:=I + 1.

Remarque : si vous tapez, comme cela est vivement conseillé, les petits programmes d'exemples, n'oubliez pas de faire au préalable NEW pour effacer les anciens programmes de la mémoire, et [RAZ] pour effacer l'écran. Vous éviterez ainsi quelques réactions pour le moins bizarres de votre MO5.

1.4.3 Instructions d'entrée-sortie : INPUT, PRINT

Un micro-ordinateur est, comme tout ordinateur, une machine destinée au traitement de l'information. Le travail d'un micro-ordinateur comporte trois phases essentielles : l'acquisition des données, le traitement de ces données et l'affichage des résultats obtenus.

Les phases d'acquisition des données et de sortie des résultats sont les phases de « dialogue » entre l'utilisateur d'un programme et le programme lui-même. Ces phases nécessitent des instructions spéciales, appelées instructions d'entrée-sortie. En BASIC ces instructions sont INPUT et PRINT.

*** Instruction d'entrée INPUT**

L'instruction INPUT est fondamentale pour la phase d'acquisition des données. C'est elle en effet qui permet à l'utilisateur d'introduire des données à partir du clavier en cours d'exécution d'un programme.

Si par exemple vous désirez voir votre MO5 vous dire bonjour en vous appelant par votre prénom, il est nécessaire de fournir au MO5 votre prénom. Cela justifie la ligne 5 du premier programme de ce chapitre :

```
5 INPUT "QUEL EST VOTRE PRENOM "; P$
```

Lors de l'exécution de cette ligne 5, le MO5 affiche sur l'écran du téléviseur la question, constituée par le message placé entre guillemets, et attend votre réponse. La réponse que vous lui donnez est alors rangée dans la variable « chaîne » P$

Les applications de l'instruction INPUT sont innombrables, ne serait-ce qu'en raison de la rareté des programmes ne comportant pas de phase d'acquisition de données. Les jeux sur ordinateur font énormément appel à cette instruction. Donnons quelques exemples qui sont utilisés ultérieurement.

Exemples :

Combien d'argent le joueur veut-il miser ?

```
1Ø INPUT "MISE"; M
```

Quel niveau de jeu le joueur désire-t-il ?

```
1Ø INPUT "NIVEAU" ; N
```

*** Instruction de sortie PRINT**

L'instruction PRINT est indispensable pour la phase de sortie des résultats obtenus à la fin du traitement demandé. Elle permet en effet d'obtenir, sur l'écran du téléviseur, les résultats de calculs, l'affichage de messages et de dessins variés, etc.

Le ligne 1ØPRINT "BONJOUR " ; P$ complète la ligne 5 donnée précédemment. Voilà donc, rapidement fait, un petit programme qui dit poliment bonjour.

Les deux utilisent beaucoup l'instruction PRINT pour afficher toutes sortes de choses

Exemples :

```
12Ø PRINT "SCORE"; SCORE
1ØØØ PRINT "VOUS AVEZ GAGNE!!!"
```

1.4.4 Instructions de branchement : GOTO, GOSUB

Un ordinateur exécute toujours un programme de façon séquentielle, c'est-à-dire par ordre de numéros de lignes croissants, sauf exceptions : sauts inconditionnels, tests, sous-programmes.

Les tests seront vus dans le paragraphe suivant. Intéressons-nous tout d'abord aux sauts inconditionnels et aux sous-programmes.

*** Instruction de saut inconditionnel : GOTO**

L'instruction GOTO n permet de sauter directement de la ligne où est située cette instruction à la ligne de numéro n.

Cette instruction est souvent très utile bien qu'elle soit — à juste titre il faut l'avouer — remise en cause par les partisans de la programmation structurée. Utiliser abondamment l'instruction GOTO est céder à une certaine facilité qui peut se révéler nuisible. Un programme comportant un trop grand nombre de GOTO devient en effet incompréhensible et inextricable. Sa mise au point s'avère alors difficile et d'éventuelles modifications quasiment impossibles.

Le premier programme de ce chapitre donne une application simple de l'instruction GOTO et montre comment permettre à un programme de tourner sans fin.

```
15 GOTO 5
```

c'est-à-dire n° ligne de fin GOTO n° ligne de début.

L'instruction GOTO est souvent utilisée en même temps que les instructions de test qui seront décrites plus loin. Il est en effet nécessaire, suivant le résultat d'un test, de passer à une ligne ou à une autre.

Donnons un exemple d'application de GOTO dans un programme de jeu :

```
:
:
1ØØ PRINT "VOUS GAGNEZ!!!"
1Ø GOTO 1ØØØ
:
:
2ØØ PRINT "VOUS PERDEZ!!!"
21Ø GOTO 1ØØØ
:
:
1ØØØ INPUT "VOULEZ VOUS REJOUER"; R$
```

*** Instruction d'appel d'un sous-programme : GOSUB**

L'analyse d'un problème permet souvent de le diviser en plusieurs petits problèmes pouvant être résolus séparément, la mise en commun des solutions de chacun d'eux permettant ensuite de résoudre le problème initial. On peut donc écrire un programme principal et plusieurs «petits» programmes appelés sous-programmes (en fait souvent plus longs que le programme principal). Les sous-programmes sont appelés les uns après les autres par le programme principal afin de résoudre les différents petits problèmes, le programme principal se chargeant de trouver la solution globable.

Cette façon de procéder est appelée programmation structurée. Elle permet d'écrire des programmes modulaires, simples, clairs, justement mis au point module par module, et aisément modifiables.

L'instruction d'appel d'un sous-programme est l'instruction GOSUB n où n est le numéro de la première ligne du sous-programme.

Le sous-programme doit se terminer par l'instruction RETURN afin de redonner le contrôle au programme principal dès qu'il a fini son travail.

Remarques :

— il est souvent utile de donner des noms aux différents sous-programmes que l'on peut être appelé à utiliser. Ce nom sera mis dans une instruction REM (REMarque) en tant que première ligne du sous-programme. Cela facilitera ensuite grandement la lecture du programme et permettra de savoir très vite quelles sont les fonctions réalisées par les différents sous-programmes.

— De façon générale, n'hésitez pas à employer fréquemment l'instruction REM. Un programme bien commenté est nettement plus facile à comprendre qu'une suite d'instructions sans aucune explication.

Exemple : programme de calcul de la circonférence et de la surface d'un cercle connaissant son rayon.

```
10 PI = 3.14
20 INPUT "RAYON" ; R
30 GOSUB 100
40 GOSUB 200
50 PRINT "CIRCONFERENCE = " ; C
60 PRINT "SURFACE = " ; S
70 END
100 REM CIRCONFERENCE
110 C = 2 * PI * R
120 RETURN
200 REM SURFACE
210 S = PI * R ↑ 2
220 RETURN
```

1.4.5 Instructions de test : IF. THEN, ELSE

Nous avons vu que le travail d'un ordinateur comporte trois phases distinctes : l'acquisition de données, le traitement des données et la sortie des résultats. Les instructions d'entrée-sortie ont été décrites, il reste maintenant à parler des instructions fondamentales du BASIC pour la phase de traitement des données.

Dans tout traitement il y a des calculs intermédiaires dont les résultats influent sur la suite du déroulement du programme. Il y a aussi des interventions de l'utilisateur (par des instructions INPUT) qui forcent le choix de telle ou telle option, jouer au niveau le plus difficile ou au niveau le plus facile, par exemple. Un programme doit donc avoir les moyens d'effectuer des tests et d'agir en fonction des résultats de ces tests.

L'instruction de test en BASIC est l'instruction IF /condition/ THEN /instruction 1/ ELSE /instruction 2/.

Cela peut se traduire en français par : SI la condition est réalisée (résultat non nul, expression logique « vraie ») ALORS exécuter l'instruction 1 SINON exécuter l'instruction 2.

Le petit programme suivant met en évidence la plupart des modes d'utilisation de l'instruction de test.

```
1Ø INPUT "A" ; A
2Ø INPUT "B" ; B
3Ø PRINT
4Ø IF A = B THEN PRINT "A EGAL B" ELSE
   PRINT "A DIFFERENT DE B"
5Ø PRINT
6Ø IF A THEN PRINT "A EST NON NUL" : PRINT
7Ø IF A * B < Ø THEN IF A < B THEN PRINT "A EST
   NEGATIF, B EST POSITIF" ELSE PRINT "A EST
   POSITIF, B EST NEGATIF"
8Ø PRINT
9Ø IF A > B OR A < Ø GOTO 1Ø ELSE GOTO 1ØØ
1ØØ IF A < B AND B > Ø THEN GOTO 1Ø
2ØØ GOTO 1Ø
```

— les lignes 1Ø et 2Ø permettent d'entrer les deux nombres A et B sur lesquels vont porter les tests.

— les lignes 3Ø, 5Ø et 8Ø ont pour but d'aérer la présentation des résultats sur l'écran du téléviseur.

— la ligne 4Ø est une application typique de l'instruction de test et n'a pas besoin d'explication supplémentaire.

— la ligne 6Ø teste si A est non nul. Si c'est le cas le message A EST NON NUL apparaît, sinon il y a passage direct à la ligne 7Ø. On peut donc ainsi tester si le résultat d'un calcul est nul ou non et, suivant la réponse du test, exécuter une instruction ou une autre.

— la ligne 7Ø présente une possibilité extrêmement intéressante de l'instruction de test. On peut en effet mettre plusieurs tests en cascade.

— Si le produit A * B est positif il y a passage à la ligne 8Ø sans exécuter le reste de la ligne 7Ø.

Par contre, si le produit A * B est négatif, alors le programme teste si A est plus petit que B. Si c'est le cas, le programme en déduit que A est négatif et B positif (A * B < Ø entraîne A et B de signes différents, A < B entraîne A négatif et B positif), sinon c'est A qui est positif et B négatif.

La mise en cascade de tests est particulièrement intéressante pour optimiser la vitesse de certaines phases du traitement. Certains tests ne sont à effectuer que si le résultat d'un test précédent le permet. Il est donc inutile d'effectuer ces tests systématiquement. La mise en cascade est une solution élégante et rapide à ce problème.

— La ligne 9Ø introduit deux nouvelles notions : l'instruction IF sans l'instruction THEN et les expressions logiques.

L'instruction IF... GOTO... est identique à IF... THEN GOTO..., il n'y a pas lieu d'en dire plus.

Le test porte cette fois-ci sur une expression logique : A > B OR A < Ø. Si A est plus grand que B *ou* si A est négatif, l'expression logique est dite « vraie » et l'instruction suivante est exécutée. Si A est plus petit que B *et* A est positif (expression logique contraposée de l'expression logique précédente) alors il y a passage directement à la ligne suivante.

— La ligne 1ØØ donne un exemple d'utilisation de l'opération logique AND (c'est-à-dire ET). Si A est plus petit que B *et* si B est positif alors aller à la ligne 1Ø.

1.4.6 Boucles FOR, TO, NEXT, STEP

Certains calculs en traitement doivent être exécutés plusieurs fois. Il est alors intéressant de créer une boucle qui sera parcourue autant de fois que cela est nécessaire. Les instructions FOR, TO, NEXT, STEP permettent de créer de telles boucles.

La syntaxe est la suivante :

FOR/variable numérique/= X TO Y STEP Z
NEXT/variable numérique/

La variable numérique qui suit le mot-clé FOR est la variable de contrôle de la boucle. Cette variable joue le rôle de compteur.

X est la valeur initiale du compteur, Y est la valeur finale.

L'expression STEP Z et la /variable numérique/ qui suit le mot-clé NEXT sont facultatifs. Par défaut le pas (STEP) Z est pris égal à 1.

Comment fonctionne une boucle FOR — NEXT ?

— la variable de contrôle prend la valeur C
— la boucle est exécutée
— lorsque le programme arrive sur NEXT qui veut dire « passer à la valeur suivante de la variable de contrôle », il y a incrémentation de cette variable : variable de contrôle = variable de contrôle + Z (Z égale 1 par défaut).

Le programme teste ensuite si la variable de contrôle est plus petite que la valeur finale Y. Si c'est le cas il y a retour au début de la boucle, sinon le programme « sort » de la boucle et passe aux instructions suivantes.

Remarque : il est vivement recommandé, dans le cas de boucles imbriquées, de donner des noms différents à chacune des variables de contrôle des boucles imbriquées. Il faut aussi vérifier qu'à chaque instruction FOR correspond une instruction NEXT. N'oubliez pas non plus de préciser explicitement après un NEXT le nom de la variable de contrôle concernée bien qu'il soit en théorie possible de ne pas le faire. Ces diverses précautions permettent d'éviter des erreurs de programmation parfois longues et difficiles à retrouver.

1.4.7 Premier jeu : REVERSE

Vous connaissez maintenant les instructions les plus importantes du BASIC. Vous êtes désormais potentiellement capable de programmer tout ce que vous voulez.

Le paragraphe suivant vous montrera comment créer un jeu en BASIC, de l'analyse du problème à l'écriture du programme en passant par le dessin d'un organigramme. Le programme ci-dessous a simplement pour but de vous montrer de quelle façon les instructions décrites précédemment peuvent être mises en œuvre pour programmer un jeu. Nous espérons que vous tirerez profit de la lecture de ce programme, mais aussi que vous vous amuserez en l'utilisant.

*** Programme de jeu reverse**

```
  5 REM PROGRAMME REVERSE
 1Ø DIM A(21) : CLS
 2Ø INPUT "ENTREZ UN NOMBRE N"; N
 3Ø N = INT (ABS(N))
 4Ø FOR K = 1 TO N
 5Ø V = RND
 6Ø NEXT K
 7Ø FOR I = 1 TO 9
 8Ø R = INT (RND * 9 + 1)
 9Ø IF A(R) < > Ø THEN GOTO 8Ø
1ØØ A(R) = I
11Ø NEXT I
12Ø GOSUB 1ØØØ
13Ø INPUT "ROTATION" ; ROT
14Ø IF ROT > 9 THEN GOTO13Ø
15Ø IF ROT = Ø THEN CLS : STOP
16Ø FOR I = 1 TO INT (ROT/2)
17Ø Z = A(I)
18Ø A(I) = A(ROT - I + 1)
19Ø A(ROT - I +1) = Z
2ØØ NEXT I
21Ø GOSUB 1ØØØ
22Ø FOR I = 1 TO 9
23Ø IF A(I) <> I THEN GOTO 13Ø
```

```
 24Ø NEXT I
 25Ø PRINT "GAGNE"
 26Ø END
1ØØØ PRINT
1Ø1Ø FOR I = 1 TO 9
1Ø2Ø PRINT A(I) ;
1Ø3Ø NEXT I
1Ø4Ø PRINT "_ _ _ _" (4 espaces)
1Ø5Ø PRINT
1Ø6Ø RETURN
```

*** Règles du jeu**

Les chiffres 1, 2, 3, 4, 5, 6, 7, 8, 9 sont placés dans un ordre quelconque par le MO5.

Le but du jeu est d'ordonner ces chiffres par ordre croissant à partir de la gauche.

Pour ce faire vous devez demander au MO5 d'effectuer les inversions de votre choix.

Exemples :

```
* 1 3 5 7 9 2 8 4 6
  ROTATION ? 4
  7 5 3 1 9 2 8 4 6
```

```
* 2 3 4 5 1 6 7 8 9
  ROTATION ? 4
  5 4 3 2 1 6 7 8 9
  ROTATION ? 5
  1 2 3 4 5 6 7 8 9
  GAGNE
```

Bien sûr le jeu refusera toute rotation supérieure à 9.

Au début du jeu un nombre N vous est demandé. Cela permet d'obtenir les séquences de nombres aléatoires différentes lors du tirage de la suite initiale. Deux nombres différents donnent deux suites initiales différentes. Si vous désirez rejouer plusieurs fois avec une suite initiale donnée, il vous suffira de redonner à chaque

fois le même nombre N au début du jeu pour obtenir cette suite initiale

*** Remarques**

• Le but de ce jeu étant essentiellement de vous donner l'occasion d'étudier un programme, la structure du programme REVERSE n'est donnée qu'en annexe.

• De plus ce jeu est volontairement dépouillé de tout artifice de présentation. De nombreuses améliorations peuvent être faites : comptage du nombre de coups nécessaires pour ordonner la suite, couleurs, sons, affichage des règles du jeu, etc... C'est un bon exercice que d'essayer de faire vous-même quelques-unes des modifications suggérées ci-dessus.

1.5 CRÉATION D'UN JEU : JEU DU 21

Nous avons vu que programmer un micro-ordinateur consiste à mettre dans la mémoire de ce micro-ordinateur une série d'instructions écrites dans un langage compréhensible par la machine. Vous connaissez maintenant les instructions fondamentales du langage évolué de programmation BASIC. Il reste encore à savoir comment passer d'un problème donné à l'écriture d'un programme qui résoudra ce problème.

Quand on leur soumet un problème, les gens cèdent très souvent à la tentation de jeter immédiatement sur une feuille une longue série d'instructions sans même savoir exactement ce qu'ils se proposent de faire. Une nouvelle idée vient à leur passer par la tête et voilà sur la feuille quelques lignes de plus sans préoccupation aucune de savoir si ces lignes sont compatibles ou non avec les lignes qui les précèdent. Après de nombreuses heures perdues à tenter de mettre au point un programme qui, étant conçu en dépit du bon sens, refuse généralement de « tourner », ces gens-là sont obligés de tout effacer et de repartir à zéro.

Programmer doit être un plaisir, mais programmer sans méthode n'est qu'une perte de temps et une source d'erreurs.

La programmation comporte quatre étapes fondamentales qu'il convient d'effectuer systématiquement si on veut être efficace et rapide.

* analyse du problème,
* écriture d'un organigramme,
* codage,
* mise au point du programme.

Ces différentes étapes ont bien sûr plus ou moins d'importance suivant les problèmes posés. Afficher le message "BONJOUR" sur l'écran du téléviseur ne demande pas une analyse « poussée » ni même l'écriture d'un organigramme. Des problèmes plus complexes, comme c'est le cas pour la plupart des jeux sur micro-ordinateur, nécessitent par contre une analyse très fine et l'écriture d'un organigramme complet. C'est à ce prix qu'on peut être sûr d'aboutir à un programme qui « tourne » bien et remplit exactement le cahier des charges.

Nous allons montrer sur un exemple comment mettre en œuvre les différentes étapes de la programmation en programmant un jeu très connu : le jeu du 21.

1.5.1 Analyse du jeu

L'analyse est la première étape de la programmation et on pourrait presque dire que c'est la plus importante. C'est d'elle que dépend en effet toute la suite des opérations.

L'analyse consiste dans un premier temps à définir exactement le problème que l'on veut résoudre. Quelles sont les questions posées ? Quels résultats va-t-on en attendre ? Quelles sont les règles du jeu ? Quels sont les rôles respectifs du joueur et du micro-ordinateur ?

Le problème étant clairement posé, il s'agit ensuite de le décomposer en plusieurs sous-problèmes. Cette décomposition descendante se poursuit jusqu'à l'obtention de problèmes élémentaires ou « modules » ne nécessitant que des traitements simples.

Procédons à l'analyse du jeu du 21 que nous nous proposons de programmer.

*** Définition du problème : règles du jeu**

Le jeu du 21 est un jeu de dés (ou de cartes) connu. Les règles que nous allons donner sont peut-être différentes de celles qui ont cours dans les casinos mais cela n'a que peu d'importance.

L'essentiel est d'arriver à définir des règles précises et simples et d'obtenir un programme de jeu plaisant à utiliser.

- Le joueur lance un dé à six faces autant de fois qu'il le désire. La somme de ces lancers ne doit pas dépasser 21 mais on doit s'en approcher le plus possible.
- Si le total du joueur dépasse 21, le joueur perd.
- Lorsque le joueur estime un total suffisant, il peut arrêter de jouer le dé et c'est au MO5, qui joue le rôle de la banque, de lancer le dé à son tour.
- Si le total de la machine dépasse le total du joueur sans pour autant dépasser 21, le joueur gagne.
- Si le total de la machine dépasse 21, le joueur gagne.
- Au début du jeu, le joueur se voit créditer d'un avoir, mille francs par exemple. A chaque tour le joueur est obligé, pour pouvoir jouer, de miser une certaine somme. S'il ne mise rien le jeu s'arrête.
- La mise ne doit pas dépasser l'avoir du joueur.
- Quand le joueur gagne, il peut gagner jusqu'à cinq fois sa mise.
- Lorsque le joueur perd, il perd sa mise.
- Si par malheur l'avoir du joueur s'annulait, le jeu s'arrête (un joueur ruiné n'a en effet plus les moyens de miser...). Cela n'est pas dramatique car toute relance du jeu recrédite l'avoir du joueur.

*** Récapitulation des cas de gain et de perte du joueur**

- Le joueur gagne si la machine dépasse 21 en essayant de dépasser le total du joueur. Dans ce cas le joueur peut gagner jusqu'à cinq fois sa mise.
- Le joueur perd si son total dépasse 21 ou si la machine obtient un total supérieur à celui du joueur, sans dépasser 21. Dans ce cas le joueur perd sa mise.
- Si le joueur et le MO5 ont tous les deux 21, le coup est nul.
- Le jeu s'arrête si des revers successifs ont acculé le joueur à la ruine et réduit son avoir à zéro.

*** Déroulement du jeu**

- Il est nécessaire dans un premier temps de créditer l'avoir du joueur. C'est la phase d'initialisation.
- Le joueur peut lancer ensuite le dé autant de fois qu'il le désire. C'est la phase de jeu du joueur. Cette phase ne cesse que si le joueur le désire ou si, en voulant trop tenter sa chance, le joueur obtient un total dépassant 21.
- La machine lance le dé à son tour. C'est la phase de jeu de la machine.
- Enfin les cas de gain et de perte du joueur sont traités. L'avoir du joueur est ajouté à sa nouvelle valeur. Il y a ensuite : soit retour au début pour un nouveau jeu, soit arrêt du programme pour cause de ruine de joueur.

Nous venons donc de définir quatre phases principales dont l'enchaînement permettra de jouer au jeu du 21 avec un micro-ordinateur MO5 dans le rôle de la banque.

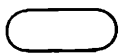
1.5.2 Ecriture d'un organigramme

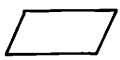
Ecrire ou plutôt dessiner un organigramme permet de visualiser les résultats de l'analyse. Celle-ci a mis en balance plusieurs « modules ». L'organigramme montre l'organisation de ces modules, les liaisons qui existent entre eux, les sauts, les branchements, les tests qui permettent de passer des uns aux autres afin de réaliser la fonction souhaitée.

Plusieurs organigrammes peuvent être dessinés. Un organigramme, dit organigramme principal, représente la suite logique des étapes nécessaires pour la résolution du problème. Plusieurs organigrammes secondaires représentent l'enchaînement des traitements nécessaires pour effectuer chacune des étapes intervenant dans l'organigramme principal.

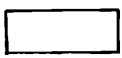
Les organigrammes suivent donc une démarche descendante similaire à celle adoptée pour l'analyse. La décomposition s'arrête lorsque les traitements de niveau le plus bas peuvent être réalisés directement par les instructions du langage de programmation utilisé. Il sera alors temps de passer à la phase de codage du programme.

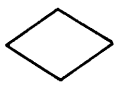
Dessiner n'importe comment un organigramme ferait perdre à celui-ci tout son intérêt. Des symboles spéciaux ont été mis au point pour dessiner les organigrammes. Quatre d'entre eux sont fondamentaux et interviennent très fréquemment.

*  représente un point terminal (début ou fin du programme). Ce symbole ne comporte qu'une arrivée ou un départ suivant le cas.

*  représente une opération d'entrée-sortie (nécessité d'entrer une donnée avec INPUT ou d'afficher un message avec PRINT).

  Ce symbole n'a qu'une arrivée et un départ.

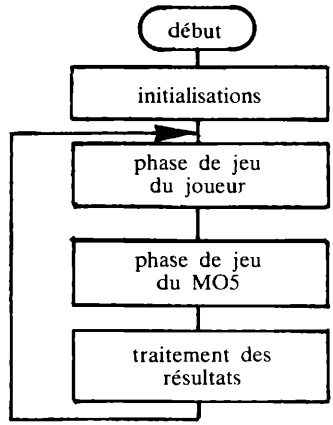
*  représente un ensemble de traitements à effectuer. Ce symbole ne comporte qu'une arrivée et un départ.

*  représente un test (possibilité de prendre une décision en fonction du résultat du test). Ce symbole a une entrée et deux sorties, l'une ou l'autre des sorties étant prise selon le résultat du test.

On trouve aussi parfois les symboles ♁ et ♀ qui indiquent que l'organigramme a une suite qui se trouve sur la page dont le numéro est indiqué dans le symbole (ex. : ♁ ②) ou bien que l'organigramme et la suite de l'organigramme se trouvent sur une autre page (ex. : ② ♀).

Nous possédons maintenant les moyens de dessiner l'organigramme du jeu du 21 dont l'analyse a été faite précédemment.

Organigramme principal du jeu du 21



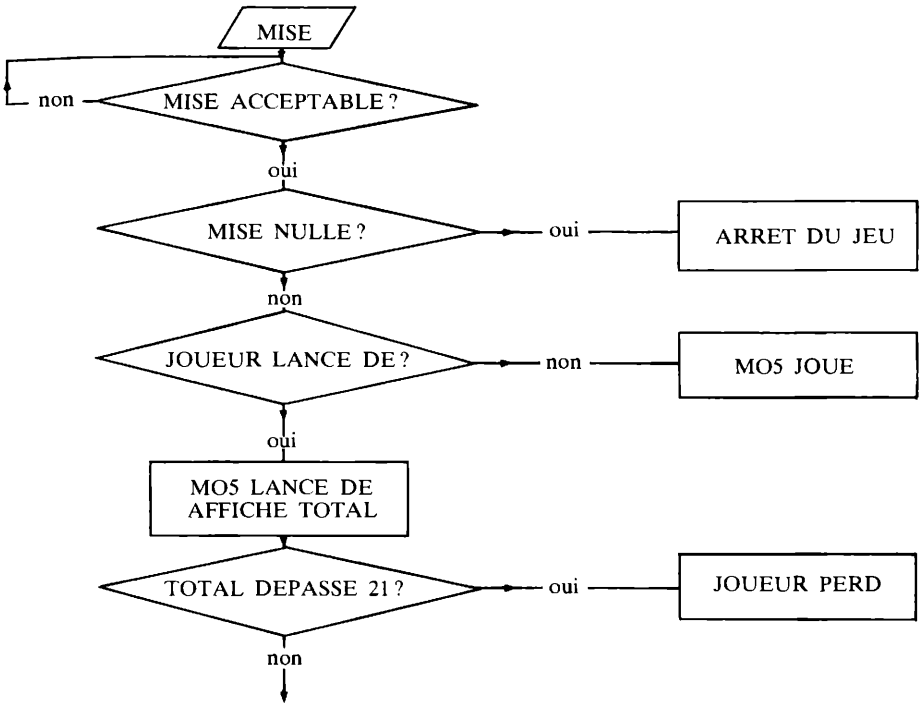
Il s'agit maintenant de dessiner les organigrammes secondaires correspondant aux différents modules que l'organigramme principal met en évidence.

La phase d'initialisation ne pose pas de problème. Il suffit de donner à une variable, que l'on pourra appeler AVOIR, la valeur initiale 1000.

*** Organigramme de la phase de jeu du joueur**

- Le joueur doit d'abord miser. Sa mise ne doit pas dépasser son avoir. Si la mise est nulle, le jeu s'arrête.
- Le joueur peut ensuite soit lancer le dé soit laisser le MO5 jouer.
- Si le joueur lance le dé et obtient un total dépassant 21, le joueur a perdu. Si le total est inférieur à 21, le joueur se voit proposer un nouveau lancé

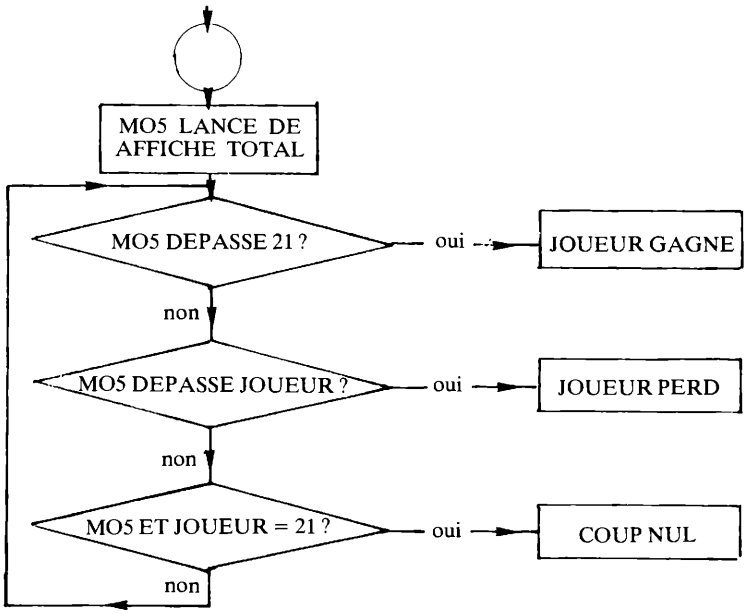
L'organigramme est donc :



*** Organigramme de la phase de jeu du MO5**

- Le MO5 lance le dé.
- Si son total dépasse 21, le joueur a gagné.
- Si son total dépasse le total du joueur sans dépasser 21, le joueur a perdu.

• L'organigramme est donc :



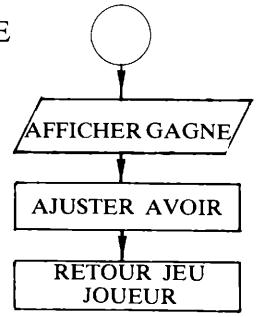
*** Organigramme de la phase de traitement des résultats**

• L'analyse et les organigrammes ci-dessus montrent que deux cas se présentent : le joueur gagne ou le joueur perd. L'avoir du joueur est ajusté en conséquence.

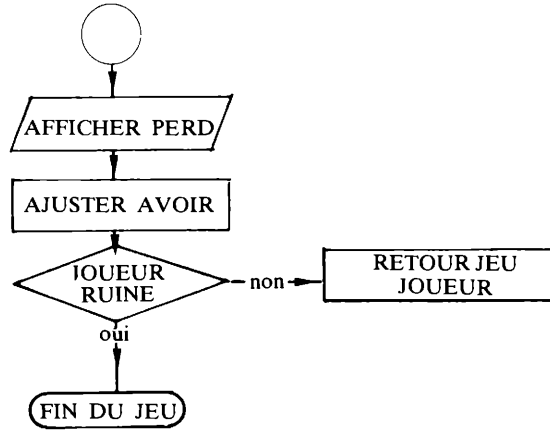
• Si le joueur est ruiné le jeu s'arrête, sinon le joueur a la possibilité de rejouer.

• On peut donc dessiner deux petits organigrammes, l'un pour JOUEUR GAGNE, l'autre pour JOUEUR PERD.

• JOUEUR GAGNE



• JOUEUR PERD



La décomposition est maintenant suffisante pour que l'on puisse passer à la phase de codage.

1.5.3 Codage

* Commençons dans un premier temps par écrire les instructions qui effectuent les initialisations nécessaires.

```
1Ø REM
2Ø REM JEU DU 21
3Ø REM
4Ø REM INITIALISATIONS
5Ø AVOIR = 1ØØØ
6Ø MICRO = Ø
7Ø JOUEUR = Ø
```

• Nous mettons volontairement beaucoup de REM car un programme très commenté et aéré est beaucoup plus lisible.

• La variable AVOIR, qui contient le crédit du joueur, est initialisée à 1ØØØ au début du jeu.

• Les variables MICRO et JOUEUR sont les totaux réalisés en lançant le dé par la MO5 et le joueur. Ces totaux sont évidemment nuls au début du jeu.

Remarquons tout de suite que ces totaux doivent être remis à zéro entre chaque tour de jeu.

* Ecrivons, maintenant les instructions permettant d'exécuter les phases de jeu du joueur ; regardez en même temps, si possible, l'organigramme correspondant.

```
 8Ø REM JOUEUR LANCE LE DE
 9Ø PRINT "AVOIR :" ; MISE
1ØØ INPUT "MISE" ; MISE
11Ø IF MISE > AVOIR GOTO 1ØØ
12Ø IF MISE = Ø GOTO ARRET DU JEU
```

• Nous voyons que la ligne 12Ø pose un problème : l'organigramme exige, dans le cas où la mise est nulle, de se brancher sur ARRET DU JEU.

Cette partie du programme n'est pas encore écrite, nous ne savons donc pas quel est pour l'instant le numéro de ligne à mettre après l'instruction GOTO. Mettons donc pour l'instant l'« étiquette » ARRET DU JEU. Nous pourrons remplacer cette étiquette par un numéro de ligne quand le programme sera entièrement écrit.

De manière générale, chaque fois qu'un branchement sera exigé alors que nous ne connaissons pas le numéro de ligne correspondant, nous mettrons une étiquette. Les étiquettes seront remplacées par des numéros de ligne quand le programme sera terminé.

```
13Ø INPUT "JET" ; J$
14Ø IF J$ <> "" GOTOMO 5 JOUE
15Ø JOUEUR = JOUEUR + INT(RND * 6 + 1)
16Ø PRINT "JOUEUR : __" ; JOUEUR
17Ø IF JOUEUR > 21 GOTO JOUEUR PERD
18Ø GOTO 13Ø
```

• Jusqu'à présent nous avions utilisé des instructions que vous connaissiez. La ligne 15Ø fait appel à deux fonctions nouvelles :

INT(X) donne la partie entière du nombre X, c'est-à-dire le plus grand entier inférieur ou égal à X.

RND donne un nombre réel aléatoire compris entre zéro et un, un n'étant jamais atteint.

INT(RND * 6 + 1) donne donc un nombre entier compris entre 1 et 6, simulant ainsi le lancer d'un dé à six faces.

* Passons maintenant à la phase de jeu du MO5.

```
19Ø REM MO 5 LANCE LE DE
2ØØ MICRO = MICRO + INT(RND * 6 + 1)
21Ø PRINT "MO 5: __" ; MICRO
22Ø IF MICRO > 21 GOTO JOUEUR GAGNE
23Ø IF MICRO > JOUEUR GOTO JOUEUR PERD
24Ø IF MICRO + JOUEUR - 42 = Ø GOTO COUP NUL
25Ø GOTO 19Ø
```

* Il ne reste plus qu'à coder les phases JOUEUR GAGNE, JOUEUR PERD et COUP NUL.

```
26Ø REM JOUEUR GAGNE
27Ø PRINT "VOUS GAGNEZ !!!"
28Ø AVOIR = AVOIR + INT(RND * 5 + 1) * MISE
29Ø GOTO 6Ø
3ØØ REM JOUEUR PERD
31Ø PRINT "VOUS PERDEZ..."
32Ø AVOIR = AVOIR - MISE
33Ø IF AVOIR > Ø GOTO 6Ø
34Ø PRINT "VOUS ETES RUINE !!!"
35Ø END
36Ø REM COUP NUL
37Ø PRINT "COUP NUL !!!"
38Ø GOTO 6Ø
```

* Il faut maintenant remplacer les étiquettes que nous avons placées tout au long du programme par les numéros de lignes correspondantes.

La ligne 12Ø devient :

```
12Ø IF MISE = Ø GOTO 35Ø
```

La ligne 14Ø devient :

```
14Ø IF J$ <>"" GOTO 19Ø
```

La ligne 17Ø devient :

```
17Ø IF JOUEUR > 21 GOTO 3ØØ
```

La ligne 22Ø devient :

```
22Ø IF MICRO > 21 GOTO 26Ø
```

La ligne 23Ø devient :

```
23Ø IF MICRO > JOUEUR GOTO 3ØØ
```

La ligne 24Ø devient :

```
24Ø IF MICRO + JOUEUR - 42 = Ø GOTO 36Ø
```

* Vous pouvez dès à présent utiliser ce programme pour jouer.

Entrez votre mise et pressez [ENTREE]. Si votre mise est égale à zéro, le jeu s'arrête.

Pour lancer le dé, pensez simplement "ENTREE".

Lorsque vous estimez votre total suffisant, pressez une touche quelconque puis [ENTREE]. Le MO5 lance alors le dé à son tour.

1.5.4 Amélioration du jeu du 21

Plusieurs modifications peuvent être faites sur le programme précédent pour en rendre la présentation plus attrayante.

Profitons du fait que les lignes sont espacées de dix en dix, nous allons pouvoir insérer de nouvelles lignes intervenant essentiellement sur l'affichage.

Nous vous donnons tout de suite le « listing » du programme modifié. Vous trouverez ensuite des explications détaillées sur chacune des modifications apportées.

Vous pouvez effectuer directement sur le programme précédent les modifications, celles-ci n'étant que des insertions de ligne ou d'instructions sur des lignes déjà existantes. Il ne s'agit donc pas d'un nouveau programme mais du même programme « étoffé ».

JEU DU 21

```
1Ø REM
2Ø REM JEU DU 21
3Ø REM
4Ø REM INITIALISATIONS
42 CLS
44 CLEAR ,, 2
45 GOSUB 5ØØ
5Ø AVOIR = 1ØØØ
6Ø GOSUB 7ØØ
65 CLS : MICRO = Ø
7Ø JOUEUR = Ø
8Ø REM JOUEUR LANCE LE DE
9Ø PRINT "AVOIR : " ; AVOIR : PRINT
1ØØ INPUT "MISE" ; MISE : MISE = INT(ABS(MISE))
11Ø IF MISE > AVOIR GOTO 1ØØ
12Ø IF MISE = Ø GOTO 35Ø
13Ø PRINT : INPUT "JET" ; J$
14Ø IF J$ <> "" GOTO 19Ø
15Ø JOUEUR = JOUEUR + INT(RND * 6 + 1)
16Ø PRINT "JOUEUR : __" ; JOUEUR
17Ø IF JOUEUR > 21 GOTO 3ØØ
18Ø GOTO 13Ø
19Ø REM MO5 LANCE LE DE
2ØØ MICRO = MICRO + INT(RND * 6 + 1)
2Ø5 GOSUB 7ØØ
21Ø PRINT : PRINT "MO5 : __" ; MICRO
22Ø IF MICRO > 21 GOTO 26Ø
23Ø IF MICRO > JOUEUR GOTO 3ØØ
24Ø IF MICRO + JOUEUR - 42 = Ø GOTO 36Ø
25Ø GOTO 19Ø
26Ø REM JOUEUR GAGNE
27Ø PRINT : PRINT "VOUS GAGNEZ!!!"
28Ø AVOIR = AVOIR + INT(RND * 5 + 1) * MISE
29Ø GOTO 6Ø
3ØØ REM JOUEUR PERD
31Ø PRINT : PRINT "VOUS PERDEZ..."
32Ø AVOIR = AVOIR - MISE
```

```
33Ø IF AVOIR > Ø GOTO 6Ø
34Ø PRINT "VOUS ETES RUINE!!!"
35Ø PRINT : INPUT "VOULEZ VOUS REJOUER__" ;
R$
352 IF R$ = "" GOTO 1Ø
355 END
36Ø REM COUP NUL
37Ø PRINT "COUP NUL!!!"
38Ø GOTO 6Ø
5ØØ REM AFFICHAGE
51Ø SCREEN 3, Ø, Ø
52Ø ATTRB 1, 1
53Ø LOCATE 1Ø, 14, Ø : PRINT "JEU DU 21"
54Ø DEFGR$ (Ø) = 255, 129, 165, 129, 129, 165, 129, 255
55Ø DEFFR$ (1) = 255, 129, 161, 129, 129, 133, 129, 255
56Ø LOCATE 5, 5, Ø : COLOR 1 : PRINT GR$(Ø)
57Ø LOCATE 35, 2Ø, Ø : COLOR 5 : PRINT GR$(1)
58Ø ATTRB Ø, Ø
59Ø LOCATE Ø, 24, 1 : COLOR 2 : INPUT "ENTREZ UN
   NOMBRE QUELCONQUE" , N
6ØØ N = INT(ABS(N))
61Ø FOR I = 1 TØ N
62Ø Y = RND
63Ø NEXT I
64Ø RETURN
7ØØ REM TEMPORISATION
71Ø FOR I = 1 TØ 5ØØ
72Ø NEXT I
73Ø RETURN
```

- L'instruction CLS, que l'on trouve dans les lignes 42 et 65, a pour but d'effacer l'écran du téléviseur et de positionner le curseur dans le coin supérieur gauche de l'écran.

- L'instruction CLEAR ,, 2 permet de réserver de la place mémoire pour la définition de deux caractères graphiques.

- De nombreux PRINT ont été ajoutés (lignes 9Ø, 13Ø, 21Ø, 27Ø, 31Ø, 35Ø) afin d'aérer la présentation du jeu.

- Un sous-programme d'affichage, apporté par la ligne 45, est placé à partir de la ligne 5ØØ. Ce sous-programme fait appel à

beaucoup d'instructions particulières du BASIC du MO5. Ces instructions permettent d'exploiter les très bonnes possibilités graphiques du MO5 :

— définition de caractères graphiques : DEFGR$
— couleurs de l'écran et des caractères : SCREEN et COLOR.
— taille des caractères ATTRB.
— positionnement du curseur : LOCATE.

Vous trouverez les détails complets sur ces instructions et leurs syntaxes particulières dans le chapitre 3.

Une instruction INPUT vous demande d'entrer un nombre quelconque. Cela a pour effet de faire « tourner » un certain nombre de fois le générateur aléatoire afin d'obtenir dans la suite du jeu des séquences différentes.

• Un sous-programme de temporisation est placé à partir de la ligne 7ØØ. Il s'agit simplement d'une boucle vide parcourue 500 fois.

• Ce sous-programme est appelé par les lignes 6Ø et 2Ø5. Il est en effet nécessaire de ralentir artificiellement le jeu sinon cela va trop vite pour le joueur : affichage trop fugitif, tirage de dé trop rapide pour le MO5.

• Vous venez d'assister à la création complète d'un jeu en BASIC. Vous avez pu constater que cela n'est vraiment pas difficile si on procède avec méthode.

Le jeu que nous vous avons proposé n'est certainement pas parfait. Nous espérons que vous aurez des idées pour l'améliorer et que vous n'hésiterez pas à les mettre en application.

Nous souhaitons que tout ce qui précède vous aura donné envie de programmer vous-même vos propres jeux, tel en tout cas était notre but.

Page blanche

CHAPITRE **2**

2.1 INTRODUCTION

Vous voilà maintenant un peu familiarisé avec votre Thomson MO 5. Poursuivons avec l'étude, en détail, du BASIC du MO5.

Cette étude se compose de quatre parties, à savoir :

1) Description des « éléments »
2) Différentes catégories d'instruction BASIC
3) Notions de sous-programmes et de fonctions
4) Entrées-sorties généralisées

2.2 DESCRIPTION DES ÉLÉMENTS

Qu'entend-on par « éléments » ? Dans tout programme on est amené à manipuler : des matrices, des nombres entiers, réels etc. et/ou des chaînes de caractères sur lesquels on effectue des

opérations ; toutes ces données constituent « les éléments ». Parmi eux on distingue les opérandes et les opérateurs. Ainsi, afin de décrire les éléments, on procèdera en deux temps.

— Description des opérandes

— Description des opérateurs.

2.2.1 Description des opérandes

Parmi ceux-ci il y a deux types ; d'une part les constantes, d'autre part les variables. La différence entre ces deux types ayant décrite dans le chapitre précédent (cf 1.4.1), on rappellera simplement pour mémoire qu'il y a :

— les constantes
  - numériques qui sont entières*, réelles, ou hexadécimales
  - chaînes de caractères

— les variables
  - numériques qui sont entières* ou réelles
  - chaînes de caractères
  - matricielles (aussi appelées tableaux)
    (Il convient de noter que les matrices sont variables à double titre :
    — par leur contenu
    — par leur structure
    Ceci sera d'ailleurs explicité ultérieurement).

Les divers opérandes vont être explicités en détail.

*** Constantes et variables entières**

La notion (mathématique) de nombre entier s'identifie au comptage d'objets (ex. : nombre de poules dans une basse-cour). En informatique, cette notion est « généralisée » aux « entiers négatifs » (ex. : un compte bancaire présentant un découvert de 1 300 francs est considéré comme étant à − 1 300 francs (moins 1 300 francs)). Le MO5 accepte les entiers compris entre − 32768 et 32767 inclus, soit :

* « entier » en langage informatique correspond à un entier relatif (ou rationnel) au sens strictement mathématique.

– 32768, – 32767, – 32766, ..., – 2, – 1, Ø, 1, 2, ..., 32766, 32767

Les variables entières (ou les cases mémoires réservées aux nombres entiers) sont repérées par l'ordinateur par le suffixe % (qui se trouve ajouté à leur nom.

Exemple :

A% désigne une variable entière,
A ne désigne pas une variable entière.

Remarques : 1. De nombreux ordinateurs codent les entiers sur deux octets, soit 16 bits. Un bit sert à déterminer le signe du nombre (positif ou négatif), les 15 autres déterminent sa valeur, ce qui explique les valeurs limites (2 ↑ 15 = 32768).

2. Si l'utilisateur désire travailler avec des nombres entiers plus importants, il faut qu'il utilise les réels associés (cf ci-après).

*** Constantes et variables réelles**

Les nombres réels sont utiles dès qu'il faut diviser les nombres (au sens strictement mathématique, cette démarche n'est pas exacte, il y aurait lieu de raisonner en termes de racine carrée au lieu de division). Il est important de retenir qu'il existe des opérateurs qui, appliqués à des entiers, conduisent à des résultats non entiers, à savoir les réels (ex. : 1/2 ne correspond pas à un entier, il est impossible de diviser une voiture en deux parties sans que celle-ci perde son caractère auto-mobile). Néanmoins dans beaucoup de cas il faut réaliser ce type de division (ex. : l'achat d'un demi-kilo de carotte à 5 francs coûte 5/2 francs). Pour représenter le résultat de telles divisions, il s'agit des nombres possédant (ou pouvant posséder) une partie décimale. Le MO5 accepte des nombres réels entre

1 EE 38 (soit 100.000.....000) (jusqu'à 38 zéros) et – 1 EE 38 (idem, précédé d'un signe moins, "—",

possédant jusqu'à 5 chiffres décimaux. Une place mémoire réservée à un réel (ou variable réelle) est repéré par le MO5 par le suffixe ! ajouté à son nom. Il est à préciser que ce suffixe est optionnel (en pratique, il est toujours omis, ce qui n'est pas un

inconvénient, car l'ordinateur considère d'autorité que les noms de variables ne se terminant pas par % ou $ sont des réels.

Exemples :

A est une variable réelle
Score est une variable réelle
1.56 est une constante réelle

Remarques : les nombres réels sont codés (par le MO5) sur 4 octets.

*** Constantes hexadécimales**

En supplément de la base 1Ø habituellement utilisée le MO5 accepte des constantes (et non des variables) exprimées en base 16.

Remarque : Les ordinateurs calculent en base deux. Tout nombre est exprimé avec deux chiffres Ø et 1. Les humains ont l'habitude de calculer en base 1Ø (tout nombre est exprimé avec 10 chiffres Ø, 1, 2, ..., 9).

Il existe une relation entre les différentes bases qui permet de transcrire tout calcul exprimé dans une base dans une autre base. Pour être prise en compte par le MO 5 toute constante exprimée en base 16 doit être précédée du symbole &H.

Enfin, dans tous calculs ou affectations (cf ci-après) les constantes exprimées en base 16 sont restituées en base 1Ø au niveau des organes de sortie (de l'ordinateur) : écran, imprimante etc.

Exemples : Avec les instructions :

PRINT &HFF [ENTREE]

le MO5 imprime :
255
OK

(En effet 255 en base 1Ø vaut FF en base 16)

Les différents types de variables abordés jusqu'à présent

correspondent aux types fondamentaux ou de base. On peut les résumer dans le tableau suivant.

Remarque : la structure des noms de variables a été décrite en détail dans le chapitre précédent. Ainsi dans le tableau ci-dessous AB a un caractère symbolique et désigne un nom de variable.

TYPE	NOM DE VARIABLE	VALEURS EXTREMES
Entier	AB %	− 32768 ; + 32767
Réel	AB ! (ou AB)	± 9.99999 E + 38
Chaîne de caractères	AB$	de 0 à 255 caractères

*** Variables matricielles**

(synonymes : matrices, tableaux)

Une matrice permet de regrouper un ensemble de variables *de même type* sous un nom commun. Ainsi il existe des matrices d'entiers, de réels ou de chaînes de caractères. Le nom commun sous lequel est regroupé l'ensemble des variables est le nom de la matrice. En BASIC il répond aux mêmes restrictions que celui d'une variable de base (cf. 1.4.1).

Exemples :

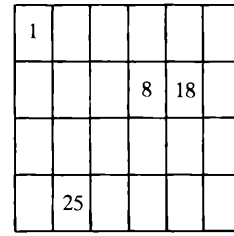
AB% désigne une matrice d'entiers
AB$ désigne une matrice de chaînes de caractères

Remarque : AB a un caractère symbolique.

Pour rentrer ou ressortir les variables placées dans la matrice, il est nécessaire de désigner les emplacements dans lesquels elles se trouvent. On ne peut pas mettre en vrac des variables dans un tableau, sans quoi on ne saurait plus les distinguer les unes des autres. Ainsi on localise une variable au moyen d'indices. Le nombre d'indices nécessaires pour localiser une variable définit la dimension de la matrice. A titre d'exemple, des entiers ont été

placés dans une matrice à deux dimensions (cf figure 1). Dans celle-ci le nombre 8 est en deuxième ligne et quatrième colonne. Les indices de la case où se trouve 8 sont donc 2, 4.

Figure 1.



Remarques :

• Il ne faut pas confondre 2, 4 et 4, 2. En effet 4, 2 repèrent la case où figure 25.

• Une case peut être « vide ». La lecture de son contenu se traduit dans certains cas par l'affichage d'un message d'erreur (sur le MO5 ; en général la lecture du contenu d'une case « vide » donne Ø).

• Un indice est toujours un nombre entier positif (ou nul). Si on désigne une case par des indices :

— non entiers l'ordinateur les transforme en les indices entiers les plus proches.

— négatifs l'ordinateur affiche le message d'erreur :

```
Error 5
OK
```

Sur le MO5 on peut utiliser des tableaux dont la taille ou dimension est variable. Si pour une matrice, il y a plus de trois dimensions ou des indices supérieurs à 1Ø, il faut « prévenir » le MO5. Ceci se fait au moyen d'une déclaration de matrice(s). Elle se fait :

— dans l'écriture du programme

— en principe au début ; du moins à un endroit du programme tel que lors de l'exécution la déclaration soit prise en compte avant toute opération avec les tableaux concernés.

La syntaxe d'une déclaration est donnée en détail dans le manuel de référence du MO5, elle est rappelée ci-après :

DIM matrice/ [, /matrice/...]

avec

/matrice/ = /nom de variable/ (indice [,indice]...)

Exemple :

1Ø DIM A% (1Ø), B$ (2,5)

déclare une matrice

— A% d'entiers à une dimension dont l'indice varie de Ø à 1Ø

et (une matrice)

— B$ de chaînes de caractères à deux dimensions dont les indices varient de Ø à 2 et de Ø à 5.

Remarques :

1) Si un indice dépasse la valeur maximale qui lui est permise le MO5 affiche :

Error 9
OK

Exemple : reprendre la ligne 1Ø de l'exemple ci-dessus avec la ligne 2Ø suivante :

2Ø PRINT A% (11)

Ce programme donne à l'exécution

Error 9 In 2Ø

(2Ø est le numéro de la ligne où est apparue l'erreur)

L'erreur est due au fait que le tableau A % ne contient que 11 cases (de Ø à 1Ø). Par conséquent la case numéro 11 n'existe pas.

2) Chaque tableau déclaré a une dimension fixée (après la prise en compte de la déclaration). De ce fait, pour tout travail sur les tableaux (déclarés), le nombre d'indices devra toujours être égal à la dimension du tableau.

Exemple :

A l'exécution du programme suivant

```
1Ø DIM B$ (2, 3, 1Ø)
2Ø PRINT B$ (1, 2, 8, 5)
```

L'ordinateur affiche le message suivant :

```
Error 9 In 2Ø
```

3) La place de variation d'un indice (ou de plusieurs indices) peut être rendue variable par l'utilisation d'une (ou de plusieurs) variable(s) auxiliaire(s).

Exemple :

```
1Ø INPUT "PLAGES DE VARIATION" ; N ; P ; Q
2Ø DIM A (N, P, Q, 1Ø)
```

Par ces deux instructions, on déclare un tableau A ayant 4 dimensions dont les indices peuvent varier respectivement de Ø à N, de Ø à P, de Ø à Q et de Ø à 1Ø. Les valeurs de N, P, Q sont à fixer lors de l'exécution du programme.

Note : cette possibilité (de rendre variables les plages de variation) est intéressante pour un jeu de bataille navale. Elle permet de disposer d'un océan de taille variable.

*** Conversion de type**

Elle est spécifique des variables numériques (les chaînes de caractères ne sont pas concernées). En quoi consiste-t-elle ? En principe toute opération, comme par exemple l'addition, ne peut être effectuée qu'entre des opérandes de même type numérique. C'est pourquoi le MO5 réalise automatiquement — si nécessaire — la conversion des types sur les opérandes.

Exemple :

En écrivant

```
A% = 5 : B = 8 [ENTREE]
PRINT A% + B [ENTREE]
```

Le MO5 affiche

13
OK

Pour ce faire, il convertit le nombre 5 entier figurant dans A% en le nombre 5 réel et peut ainsi réaliser l'addition avec le 8 réel figurant dans B.

D'autre part les noms des emplacements mémoire, dans lesquels se trouvent placées des valeurs, sont caractéristiques du type de variables qu'ils contiennent

Exemple :

A$ est un emplacement mémoire ne contenant que des chaînes de caractères.

B% ne contient que des entiers.

Ainsi, lors de la copie de la valeur d'une mémoire dans une autre (ou lors du stockage d'une constante en mémoire), il se peut que le type de l'élément à recopier (ou à stocker) soit différent du type d'éléments contenus dans l'emplacement mémoire où il va figurer. Pour pouvoir faire la copie (ou le stockage), le MO5 procède automatiquement à une conversion de type.

Exemple :

A = A% (un entier est copié dans une mémoire A réservée à des réels)

A% = 56.8 (un nombre non entier doit être stocké dans une mémoire ne pouvant contenir que des entiers. Dans ce cas le MO5 va transformer 56.8 en l'entier le plus proche (soit 57) et stocker celui-ci).

Le principe de conversion de type automatique caractérise la souplesse d'utilisation du BASIC. L'utilisateur (de BASIC) n'a pas à s'en soucier (car il se fait automatiquement) mais il est utile qu'il en soit conscient. En effet, dans d'autres langages, tels que le PASCAL, celle-ci ne se fait pas automatiquement et cela pose quelquefois des problèmes.

*** Définition implicite du type de variable**

DEFINT/DEFSNG/DEFDBL/DEFSTR

Syntaxe : { DEFINT / DEFSNG / DEFSTR } { /lettre/ [−/lettre/] [,] }

Ces quatre instructions sont pratiques dans la mesure où beaucoup de variables de même(s) type(s) sont utilisées dans un programme. Elles permettent respectivement de définir les variables dont la première lettre (de leur nom) fait partie d'un certain ensemble commé étant de type :

— entier,
— réel,
— chaîne de caractères.

Ceci permet d'omettre les suffixes %, !, $

Exemple :

DEFINT A−B, O−Q

permet de lire les variables dont le nom commence par A, B, O, P ou Q comme étant de type entier.

2.2.2 Description des opérateurs

Il y a quatre types d'opérateurs, à savoir opérateurs numériques, opérateurs sur chaînes de caractères, opérateurs relationnels et opérateurs logiques.

Ils sont abordés successivement (dans l'ordre ci-dessus).

*** Opérateurs numériques**

Ces opérateurs sont très connus, ils sont rappelés dans l'ordre de priorité décroissante.

( ) : parenthèses qui permettent de délimiter des expressions.

− : opérateur unaire qui permet de changer le signe d'une constante. Exemple : −3

↑	: élévation à la puissance (X ↑ Y)
* /	: multiplication (*) et division (/)
@ MOD	: division entière (@) et modulo
+ −	: addition et soustraction

La division entière et le «modulo» sont des opérations qui nécessitent deux opérandes (exemple : 3 @ 2 ; 5 MOD 2). Elles donnent respectivement, après avoir arrondi les opérandes (si nécessaire), le quotient et le reste de la division.

Exemples :

3 @ 2 donne 1 car 3 = *1* * 2 + 1
5 MOD 2 donne 1 car 5 = 2 * 2 + *1*

Remarques :

- Toute division par ∅ entraîne l'affichage

  Error 11

- Un dépassement de capacité de l'ordinateur avec des nombres trop grands est signalé par le message

  Error 6

Exemple :

En entrant

PRINT 1 EE + 38 ↑ 2 [ENTREE]

on obtient

Error 6
OK

*** Opérateurs sur chaînes de caractères**

Il y a un seul opérateur appelé la concaténation. Il exige deux opérandes (deux chaînes de caractères) qu'il accolle. Son symbole est + (comme pour l'addition).

Exemple :

PRINT "BONJOUR__" + "MONSIEUR" [ENTREE]

donne

BONJOUR__MONSIEUR
OK

Remarque : le symbole __ représente un « espacement » (ou caractère blanc).

*** Opérateurs relationnels**

Ils servent à faire des tests et seront très utiles dans tout programme. Ils sont donc très importants. Ils exigent tous deux opérandes, qui sont soit deux nombres (entiers, réels), soit deux chaînes de caractères. Le résultat d'une opération (relationnelle) ne peut prendre que deux valeurs : soit « vrai » soit « faux ». (Le MO5 représente « vrai » par le chiffre − 1 et « faux » par Ø).

Ces opérateurs ont tous la même priorité et elle est inférieure à celle de tout opérateur numérique. Les différents opérateurs sont :

>	« strictement supérieur à »
<	« strictement inférieur à »
>=	« supérieur ou égal à »
<>	« différent de »
=	« égal à »

Exemple :

Souvent les programmes de jeux utilisent deux mémoires (spécifiques), appelées SCOREA et SCOREB, qui contiennent les scores des joueurs, A et B. A la fin du programme il existe une instruction qui permet de déterminer le gagnant. Cette instruction utilise un opérateur relationnel.

Les manières d'utiliser ces opérateurs sont très variées.

a) D'une manière générale, du fait que le résultat d'une « comparaison » est un nombre Ø ou − 1, les opérateurs relationnels (ou « comparaison ») peuvent être introduits dans les suites numériques.

Exemple :

```
1Ø INPUT ; A, B
2Ø PRINT A = B
3Ø C = 3 * (A = B) + (A < B)
4Ø PRINT "C =" ; C
```

Les instructions ci-dessus sont correctes.

Cette possibilité d'inclure dans les expressions numériques des opérations relationnelles est très importante. Le lecteur s'en rendra compte en programmant. Cependant il faut faire attention à la mise entre parenthèses, sans quoi l'on risque de ne pas être en accord avec le but fixé.

Exemple :

```
3 * (A = B)
```

ne donne pas le même résultat que

```
3 * A = B
```

En effet l'instruction 3 * (A = B) vaut − 3 si A est égal à B et Ø sinon. Par contre 3 * A = B prendra la valeur − 1 si A est égal au tiers de B et Ø sinon.

b) On peut combiner des opérations relationnelles au moyen d'opérateurs logiques (cf. ci-après).

c) Les opérateurs relationnels sont indispensables pour les instructions de branchements conditionnels (cf. ci-après).

*** Opérateurs logiques**

Ces opérateurs, sauf un, exigent deux opérandes qui sont en théorie des booléens, et le résultat (d'une opération logique) est un booléen. Un booléen peut prendre par définition deux valeurs : soit la valeur « vrai » (qui est représentée par − 1 sur le MO5), soit la valeur « faux » (qui est représentée par Ø sur le MO5).

Exemples de booléens :

La proposition « un chat est un chien » est un booléen qui a la valeur « faux ».

La proposition « 2 + 2 = 4 » (en base 1Ø) est un booléen qui a la valeur « vrai ».

Les différents opérateurs sont :

AND appelé « et logique »
EQV appelé « équivalent logique »
IMP appelé « implication logique »
OR appelé « ou logique »
XOR appelé « ou exclusif logique »
NOT appelé « négation » (c'est le seul opérateur exigeant un seul opérande)

Ils sont entièrement définis par le tableau ci-dessous (cf. figure 2).

1er booléen (A)	2e booléen (B)	A AND B	A EQV B	A IMP B	A OR B	A XOR B
Ø	Ø	Ø	– 1	– 1	Ø	Ø
Ø	– 1	Ø	Ø	– 1	– 1	– 1
– 1	Ø	Ø	Ø	Ø	– 1	– 1
– 1	– 1	– 1	– 1	– 1	– 1	Ø

et

booléen (A)	NOT A
Ø	– 1
– 1	Ø

Figure 2

Ces opérateurs servent essentiellement à combiner des opérations relationnelles.

Exemple :

(A = B) AND (B > C) est une expression utilisant la combinaison de deux opérations relationnelles.

Néanmoins, avec le MO5, ces opérations sont « extensibles » à des valeurs numériques. Pour comprendre ces possibilités il faut connaître « la représentation des nombres ».

Qu'est-ce que la représentation des nombres ?

L'ordinateur ne travaille qu'avec les chiffres Ø et 1. De ce fait tout ce qui est entré par le clavier est converti en une suite de Ø et de 1 (pour que l'ordinateur puisse comprendre). En particulier les nombres entiers sont convertis (en suites de Ø et de 1). Les suites nécessaires pour coder les entiers utilisent 16 chiffres (Ø ou 1). Il est possible de vérifier que si le codage binaire d'un nombre entier se termine par :

un Ø	le nombre entier (codé) est divisible par 2 (= 2 ↑ 1)
deux Ø	le nombre est divisible par 4 (= 2 ↑ 2)
trois Ø	le nombre est divisible par 8 (= 2 ↑ 3)
un 1	le nombre est impair etc.

Ainsi le codage binaire (qui est la représentation des nombres) fournit des renseignements sur la parité et la divisibilité des nombres entiers.

Par suite les instructions suivantes

— Y = X AND 264 (264 est codé par 1111111111111Ø, X désigne un entier)
— Z = AND 262 (262 est codé par 1111111111111Ø)
— U = X OR 1 (1 est codé par ØØØØØØØØØØØØØØ1)

auront pour effets respectifs de :

— remplacer le dernier chiffre du codage binaire de X par un Ø. De ce fait Y est le plus grand nombre entier inférieur à X divisible par 2.

— remplacer les deux derniers chiffres du codage de X par deux Ø. Z est donc le plus grand entier inférieur à X divisible par 4.

— remplacer le dernier chiffre du codage de X par un 1. U est donc le plus petit entier supérieur à X impair.

2.3 DIFFÉRENTES CATÉGORIES D'INSTRUCTIONS BASIC

Après avoir, si besoin est, établi un organigramme aussi détaillé que possible, il faut le traduire en une suite d'instructions compréhensibles par l'ordinateur (afin de rendre le programme exécutable). Cette traduction nous l'avons vu, nécessite la connaissance de ce qui est appelé un langage. Dans le cas présent il s'agit du BASIC Microsoft version 1.Ø.

Parmi ces instructions on distingue :

— les instructions d'assignation
— les instructions de contrôle, de branchement
— les instructions de bouclage (ou les boucles)
— les instructions d'entrée-sortie

etc.

Dans les paragraphes suivants la syntaxe de chaque instruction est rappelée brièvement pour faire place à une plus ample explication. Pour chaque syntaxe ce qui est entre parenthèses () est optionnel ; les termes entre crochets[] sont à remplacer par les valeurs appropriées (ex : [variable] peut être remplacé par SCOREA%). Enfin, pour les termes superposés entre accolades, un seul doit être conservé à chaque utilisation de l'instruction. Si une instruction est introduite avec une mauvaise syntaxe, le MO5 affiche le message d'erreur suivant :

Error 2 in

où X est le numéro de la ligne du programme dans laquelle figure l'instruction erronée.

2.3.1 Instructions d'assignation

Il s'agit de mettre une valeur dans une variable. Deux cas peuvent se présenter :

— la variable n'a encore jamais été utilisée

— la variable a déjà été utilisée, ou bien il s'agit d'un tableau encore inutilisé mais déclaré (DIM).

Dans le premier cas, il n'y a qu'une possibilité d'assignation. Elle utilise l'instruction :

=

(syntaxe [non de variable] = [expression])

Cela signifie que l'expression évaluée va être placée dans la mémoire correspondant à la variable.

Dans le deuxième cas il y a une possibilité supplémentaire. Avant de l'étudier il est nécessaire d'examiner la manière dont le MO5 mémorise les valeurs des variables.

Un ordinateur comporte un ensemble de mémoires dont une partie est réservée à l'utilisateur. Toutes ces mémoires sont « numérotées » (pour la partie utilisateur, la numérotation va de 8192 à 4Ø959 inclus). Dans cette partie se situe tout ce que vous créez : variables, programmes. Ainsi chaque variable, chaque instruction est rangée dans une ou plusieurs cases mémoire, selon la taille (de la variable, de l'instruction). De ce fait, pour changer la valeur (d'une variable) il suffit de demander à l'ordinateur l'adresse de celle-ci, puis de mettre dans cette (ou ces) case(s) mémoire la nouvelle valeur. Pour procéder de cette manière à une assignation, il est nécessaire de disposer de plusieurs instructions. Trois sont utilisables, à savoir : VARPTR, PEEK, POKE.

* **VARPTR** (VARiable PoinTeuR)

Syntaxe VARPTR [nom de variable]

Elle donne l'adresse de la première case mémoire dans laquelle se trouve la variable. Comme il a été précisé auparavant, une variable peut occuper plusieurs cases mémoire. En fait le nombre de cases utilisées est lié au type de la variable :

— un entier occupe deux cases mémoire :

- la première contient les bits de poids fort
- la deuxième ceux de poids faible

— un réel occupe quatre cases mémoire :

Exposant	Bits de poids fort de la mantisse	Bits de poids moyen	Bits de poids faible
Première case	Deuxième case	Troisième case	Quatrième case
Exposant	Mantisse		

— une chaîne de caractères occupe un nombre de places dépendant de sa longueur (dans ce cas la première case contient la longueur de la chaîne).

En résumé une variable entière occupe les cases VARPTR (nom de variable) et VARPTR (nom de variable) + 1 etc.

*** PEEK**

Syntaxe PEEK (I) où I est un entier positif inférieur à 65 535 (à valeur entière).

Elle donne la valeur contenue dans la case mémoire numéro I.

Exemple :

```
1Ø INPUT "DONNEZ UN NOMBRE ENTIER" ; A%
2Ø ADR = VARPTR (A%)
3Ø PRINT PEEK (ADR) ; PEEK (ADR + 1)
4Ø END
```

Ce programme permet de visualiser la façon dont les entiers sont mémorisés. Il est possible de vérifier que dans la mémoire numéro

ADR il y a le *reste* de la division entière de A% par 256
ADR + 1 il y a le *quotient* de cette même division

Remarque : la valeur 256 résulte du codage sur 2 octets (1 octet représente 8 bits) des entiers. Ceci n'est pas fondamental dans une première approche.

*** POKE**

Syntaxe POKE I, J

Elle place dans la case mémoire numéro I la valeur J. Le nombre

I doit être positif inférieur à 65535 ; quant à J, il doit être entier positif inférieur à 255 (une case mémoire contient 1 octet, soit 8 bits, soit 2 ↑ 8 ou 256 valeurs possibles, d'où la valeur maximale de J). Si les valeurs de I et J introduites ne sont pas correctes, le MO5 affiche :

```
?
Error 5
OK
```

En utilisant VARPTR et POKE, il est possible de modifier le contenu d'une variable.

Exemple :

```
1Ø INPUT "NOMBRE"; A%
2Ø ADR = VARPTR (A%)        (ADR est l'adresse de A%)
3Ø POKE ADR, Ø              (Ø est placé dans la case mémoire ADR)
4Ø POKE ADR + 1, Ø          (Ø est placé dans la case mémoire ADR + 1)
5Ø PRINT "A% =" ; A%
```

Ce programme met Ø dans la variable A%. En effet à l'exécution si à la question de l'ordinateur :

NOMBRE ?

on répond

35 [ENTREE]

le résultat obtenu est :

```
A%=Ø
OK
```

Remarque : Pour modifier :

— un réel il faut 4 instructions POKE car il y a 4 cases mémoire concernées ;

— une chaîne de caractères, le nombre dépend de la longueur de la chaîne.

Dans un premier temps l'assimilation des notions précédentes n'est pas indispensable. Mais elle est importante pour approcher davantage le fonctionnement de l'ordinateur. Ceux qui sont intéressés par les parties « cachées » du MO5 (i.e. les parties mémoire qui ne sont pas destinées à l'utilisateur) peuvent mettre à profit PEEK et POKE.

*** CLEAR**

Syntaxe CLEAR (I) ; (J) ; (K)

Remarque préliminaire : I, J, K sont des valeurs numériques en théorie entières et optionnelles (une au moins doit être présente). Elles seront arrondies si nécessaire par le MO5. Par contre, si elles sont négatives ou si K est supérieur à 128, une erreur est détectée (Error 5).

Ses fonctions sont multiples :

— La présence de I ordonne au MO5 de réserver I octets ou I « cases mémoires » en place mémoire pour manipuler des chaînes de caractères. Il est conseillé de l'utiliser dès qu'un programme aura à effectuer la concaténation de chaînes (longues).

— La présence de J interdit au MO5 d'utiliser les cases mémoires réservées à l'utilisateur (cf. POKE, PEEK, VARPTR) au-delà de la case mémoire de numéro J. Ceci est utile pour réserver de la place pour les sous-programmes écrits en langage machine (cf. dernier chapitre).

— La présence de K indique à l'ordinateur que l'utilisateur réserve de la place mémoire pour créer des caractères graphiques (cf. chapitre suivant).

Exemples d'utilisation : cf. chapitre 1 jeu du 21.

2.3.2 Instructions de contrôle et de branchement

Bien que les instructions de contrôle soient différentes de celles de branchement, il est plus aisé de les étudier conjointement. En effet en BASIC, dans la majeure partie sinon la totalité des cas, une instruction de contrôle se trouve associée à une instruction de branchement.

— Par définition une instruction de contrôle réalise un test ou contrôle.

Exemple : déterminer entre deux joueurs A et B lequel a gagné nécessite une instruction de contrôle.

— Une instruction de branchement s'utilise en général en mode programme. Par définition elle permet de modifier le séquence-

ment de l'exécution d'un programme. En effet, un programme s'exécute ligne par ligne, et normalement dans l'ordre croissant des numéros de ligne (i.e. : ligne 1 puis ligne 2, puis 3, puis 4, etc.). Cette façon de procéder est modifiable par une instruction de branchement. Une telle instruction indique à l'ordinateur que la prochaine ligne à exécuter est celle mentionnée dans cette même instruction (de branchement).

Ces deux types d'instruction, décrites sommairement, sont fondamentaux pour la programmation. Ils permettent de réaliser des programmes capables de décision.

Les instructions de branchement et de contrôle sont les suivantes :

GOTO [numéro de ligne]		Instructions de branchement
GOSUB [numéro de ligne]		
IF [expression logique]	THEN [suite d'instructions] (ELSE [suite d'instructions])	Instruction de contrôle
IF [expression logique]	GOTO [numéro de ligne] (ELSE [suite d'instructions])	
ON [expression logique]	GOTO [suite de numéros de ligne]	Instruction de contrôle et de branchement
	GOSUB [suite de numéros de ligne]	
ON ERROR GOTO	[numéro de ligne]	

Une partie de ces instructions a été abordée dans le chapitre précédent.

* GOTO

Syntaxe GOTO [numéro de ligne]

Elle doit toujours être suivie d'un numéro de ligne, sans quoi le MO5 affiche :

Error 2

Elle donne l'ordre à l'ordinateur de continuer l'exécution du programme au numéro de ligne indiqué. Si ce numéro correspond à une ligne vide, l'ordinateur imprime :

Error 8

Une utilisation abusive de GOTO est à déconseiller, car elle conduit à des programmes inextricables, comparables à de véritables labyrinthes. Néanmoins le BASIC conduit à une utilisation forcée.

Note : la ligne repérée par le numéro mentionné après GOTO est appelée ligne de branchement.

Exemple :

```
10Ø IF SCOREA> SCOREB THEN PRINT "A EST
   GAGNANT" ELSE GOTO 2ØØ
 :
 :
2ØØ IF SCOREA = SCOREB THEN PRINT "JOUEURS
   EX AEQUO" ELSE PRINT "B EST GAGNANT"
```

La ligne 2ØØ n'est exécutée que si SCOREA est inférieur ou égal à SCOREB

*** GOSUB-RETURN**

Syntaxe : GOSUB [numéro de ligne]
RETURN

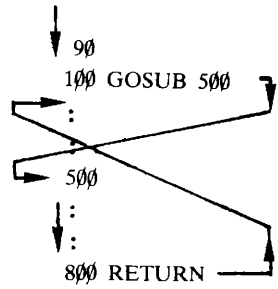
GOSUB est toujours suivi d'un numéro de ligne et la ligne considérée ne doit pas être vide (cf. GOTO). Cette instruction produit deux effets sur l'ordinateur :

— A l'exécution il mémorise le numéro de la ligne où cette instruction apparaît.

— Il continue l'exécution à partir du numéro de ligne indiqué et

cela jusqu'à ce qu'une instruction RETURN (signifiant Retour) soit rencontrée. Dans ce dernier cas il retourne à la ligne correspondant au numéro (de ligne) mémorisé (cf. premier effet). A partir de là il reprend l'exécution normale.

Il est possible de schématiser ceci (sur un exemple) :



Remarque : la partie du programme située entre le numéro de ligne mentionné dans l'instruction GOSUB (5ØØ dans l'exemple) et la première instruction RETURN constitue un sous-programme.

Cette instruction, dont l'utilisation est à recommander, est très importante. Son emploi nécessite le fractionnement de l'organigramme en sous-parties. Une « méthode » de fractionnement fera l'objet de la dernière partie de ce chapitre. Les sous-programmes doivent être placés le plus souvent en fin de programme pour la lisibilité de l'ensemble. En fait un sous-programme est un programme.

Comme il est absurde de mélanger deux programmes, il convient de placer programme et sous-programmes les uns à la suite des autres, pour ne pas les enchevêtrer.

*** IF ...THEN...ELSE ...**

Syntaxe :

IF [expression logique] { THEN [numéro de ligne] [suite d'instructions] / GOTO [numéro de ligne] } { (ELSE [numéro de ligne] [suite d'instructions]) }

Une suite d'instructions est une succession d'instructions séparées chacune par deux points.

IF...THEN...ELSE est commune à de nombreux langages de programmation tant elle est importante. L'essentiel est de bien comprendre sa signification ; rappelons-la :

si (IF)
l'expression logique est vraie
alors (THEN)
l'ordinateur — exécute les instructions situées après THEN
ou
— saute à la ligne indiquée (cas IF... GOTO)
sinon (ELSE)
il — exécute les instructions situées après ELSE
— saute à la ligne indiquée

Il faut savoir que toute expression syntaxiquement correcte et logique (utilisant des IF ... THEN ... ELSE ...) est autorisée. Une liste non limitative de cas d'emplois a été donnée au chapitre précédent (cf. 1.4.5).

* ON GOSUB — ON GOTO

Syntaxe :

ON [expression numérique]	GOSUB	[numéro de ligne] [,[numéro de ligne]]
	GOTO	[numéro de ligne] [,[numéro de ligne]]

L'expression numérique est évaluée. Notons V sa valeur. Le terme GOTO (ou GOSUB) est suivi par une liste de numéros de ligne séparés par des virgules ; appelons N le nombre de numéros (de ligne). Plusieurs cas se présentent :

— Si V n'est pas entier l'ordinateur l'arrondira. Dans la suite V est considéré comme entier (s'il ne l'est pas il faut l'arrondir).

— Si V est strictement négatif ou supérieur à 255, le message d'erreur suivant apparaît :

```
Error 5 (In...)
OK
```

— Si V est égal à Ø ou strictement supérieur à N l'exécution se poursuit en séquence (i.e. : à la ligne suivant l'introduction ON GOSUB ou ON GOTO).

— Dans les cas restants (Ø ≤ V ≤ N) l'exécution saute à la ligne indiquée par le V^e numéro (avec mémorisation du numéro de ligne de départ pour pouvoir revenir : cas ON GOSUB).

Remarque : il faut que les différents numéros de ligne correspondent à des lignes non vides, sinon après l'essai de saut à cette ligne le MO5 arrête l'exécution et affiche

Error 8

Exemple : le programme suivant détermine si une année est bissextile.

```
1Ø INPUT "ANNEE" ; AN%
2Ø PRINT "L'ANNEE" ; AN% ;
3Ø ON (AN% MOD 4 + 1) GOSUB 6Ø, 8Ø, 8Ø, 8Ø
4Ø PRINT "BISSEXTILE" (si l'année est bissextile, elle est
   divisible par 4)
5Ø END
6Ø PRINT "EST";
7Ø RETURN
8Ø PRINT "N'EST PAS ";
9Ø RETURN
```

*** ON ERROR GOTO-RESUME**

Syntaxe : ON ERROR GOTO [numéro de ligne]
RESUME [NEXT]
[numéro de ligne]

En principe si le MO5 détecte une erreur il affiche un message d'erreur et stoppe toute exécution entreprise. Par l'intermédiaire de cette instruction le déroulement du traitement ne sera pas stoppé, mais l'exécution continuera au numéro de ligne indiqué après GOTO, tout en conservant systématiquement en mémoire le numéro de ligne comportant l'erreur. Il faudra de plus inscrire dans le programme, à partir de la ligne de branchement (au moins), une ligne utilisant l'instruction RESUME. En effet il faudra préciser à l'ordinateur ce qu'il doit faire :

— recommencer l'instruction qui a provoqué l'erreur ; pour ce faire il faut écrire :

RESUME

— continuer l'exécution à la ligne suivant « l'instruction erronée » : il faut écrire :

RESUME NEXT

— continuer à un numéro de ligne quelconque : il faut écrire :

RESUME [numéro de ligne]

Remarques :

— Après avoir exécuté l'instruction RESUME, l'ordinateur abandonne l'adresse de la ligne où est apparue l'erreur. Si par suite d'une programmation « défectueuse » le MO5 rencontrait une nouvelle erreur avant RESUME il stopperait l'exécution, et afficherait un message d'erreur. En effet dans ce cas il serait obligé de mémoriser les deux adresses correspondant aux deux erreurs, ce qui est impossible de par sa conception et détermine son arrêt.

— Il est conseillé d'insérer cette instruction (ON ERROR GOTO) en début de programme afin que le MO5 sache dès le commencement de l'exécution qu'il ne doit pas s'arrêter en cas d'erreur.

Exemple :

En reprenant l'exemple précédent (année bissextile) le programme peut être modifié comme suit :

```
 5 ON ERROR GOTO 6Ø
1Ø INPUT "ANNEE" ; AN%
2Ø PRINT "L'ANNEE" ; AN%
3Ø ON (AN% MOD 4)/(AN% MOD 4)
   GOSUB 8Ø      (si l'année est bissextile AN% MOD 4 =
                 Ø, la décision en ligne 3Ø est impossible)
4Ø PRINT "BISSEXTILE"
5Ø END
6Ø PRINT "EST__" ;
7Ø RESUME NEXT (ou RESUME 4Ø)
8Ø PRINT "N'EST PAS__" ;
9Ø RETURN
```

*** ERR-ERL**

Il s'agit des deux variables dans lesquelles le MO5 place le nom (ERR) (et l'adresse (ERL) en mode programme uniquement) de la dernière erreur rencontrée.

Remarque : l'instruction ERR peut être utilisée conjointement avec une instruction ON ERROR GOTO. En effet elle permet d'effectuer des branchements différents suivant le type d'erreur rencontré.

*Exemple *:*

```
1Ø ON ERROR GOTO 3ØØ
:
:
3ØØ IF ERR = 3 THEN PRINT "IL MANQUE UN END
    DANS LE PROGRAMME"
31Ø IF ERR = 57 THEN PRINT "LE FICHIER N'EST
    PAS OUVERT"
32Ø RESUME
```

*** ERROR**

Syntaxe : ERROR [expression numérique]

Elle permet de simuler l'erreur indiquée par la valeur de l'expression numérique

Exemple :*

```
1Ø ERROR 2
```

Ce « programme » affichera

```
Error 2
```

2.3.3 Les boucles FOR... NEXT

Une boucle est une partie du programme qui pourra être répétée au cours de l'exécution. Une telle partie est délimitée par deux lignes comportant des termes particuliers, à savoir :

* Pour les codes d'erreur se reporter à l'annexe C.

- FOR — TO — STEP —(à son début)
- NEXT — (à sa fin).

Pour compléter ces « termes » il est nécessaire d'introduire des variables entières ou réelles (qui sont les seules autorisées, sinon le MO5 affiche : ?)

```
Error 13

FOR ... TO ... (STEP ...)
 :
NEXT
```

Syntaxe :

```
FOR [nom de variable numérique] = X TO Y (STEP Z)
:
:
NEXT ([nom de variable          )(,[nom de variable )
          (              numérique])(        numérique])
```

(X, Y et Z, peuvent être des constantes ou des expressions numériques).

Remarque préliminaire : si l'instruction STEP Z est omise le MO5 considérera d'autorité que Z vaut 1.

Cette instruction signifie :

— faire varier la variable numérique de X à Y par incréments de Z, *et cependant*
— exécuter les instructions inscrites dans la boucle.

Pour que cette boucle soit exécutée au moins une fois il n'y a que deux cas possibles :

1. X est inférieur ou égal à Y ($X \leqslant Y$) et le pas (Z) est positif. (note : si X est égal à Y la boucle ne sera exécutée qu'une fois).
2. X est supérieur ou égal à Y ($X \geqslant Y$) et le pas (Z) est négatif. (même note que ci-dessus).

On peut souvent (ex. : au court de manipulations de matrices ayant plus de deux dimensions) avoir besoin de plusieurs boucles en même temps. Dans ce cas, il ne faut pas oublier la règle suivante :

Des boucles ne doivent jamais s'entrelacer, mais elles peuvent être emboîtées.

Exemple 1 :

```
1Ø  DIM A(3,3)
2Ø  FOR I = 1 TO 3
3Ø  FOR J = 1 TO 3
4Ø  A (I, J) = 1
5Ø* NEXT J
6Ø  NEXT I
```

Cette boucle remplit les cases de la ligne I avec des 1

Cette boucle fait varier le numéro de ligne de 1 à 3 ; cependant la ligne est remplie avec des 1

* Les lignes 5Ø et 6Ø peuvent être remplacées par :

```
5Ø next J, I
```

Exemple 2:

```
1Ø FOR I = 1 TO 3 STEP 0.1
2Ø FOR J = 1 TO 4
3Ø  :
4Ø  :
5Ø NEXT I
6Ø NEXT J
```

Dans ce deuxième exemple, les boucles sont enchevêtrées, et de ce fait le MO5 affichera :

```
Error 1 in 5Ø
```

L'ordinateur n'a pas trouvé le Next attendu (à savoir Next J).

Remarque : il doit toujours y avoir un NEXT à la suite d'un FOR sinon le message d'erreur : Error 23 sera affiché.

2.3.4 Instructions d'entrée-sortie

Elles permettent d'établir le dialogue entre l'utilisateur et l'ordinateur. Il faut distinguer les instructions d'entrée de celles de sortie.

INSTRUCTIONS D'ENTRÉE

Elles servent à introduire dans l'ordinateur (l'unité centrale) les données (au sens large du terme).

* **INPUT**

Syntaxe :

INPUT (;) [" chaînes de caractères "] $\left\{ ; \right\}$ variable, [variable] ...

Remarque : le premier point-virgule est sans objet avec le BASIC du MO5. Comme il est nécessaire pour d'autres BASIC il a été conservé en option pour respecter la compatibilité des programmes.

Lors de l'exécution de l'instruction l'ordinateur imprime la chaîne de caractères (suivie d'un point d'interrogation si on a placé un point-virgule à la suite de cette chaîne dans le programme). Puis il attend que l'utilisateur frappe sur la touche [ENTREE] pour pouvoir reprendre l'exécution. Deux cas se présentent :

— l'utilisateur a tapé correctement le jeu de données qu'attendait l'ordinateur et l'exécution reprend.

— l'utilisateur n'a pas tapé correctement le jeu de données attendu et le message suivant est affiché :

? Redo
[chaîne de caractères]

Le MO5 considère la première (fausse) entrée comme nulle et redemande le jeu de données jusqu'à obtenir une introduction correcte. Une mauvaise introduction peut se produire de deux manières :

— l'utilisateur a introduit trop ou pas assez de données.

— l'utilisateur a introduit des données de types différents de celles attendues et l'ordinateur n'a pas pu réaliser la conversion de type comme lorsqu'il reçoit une chaîne de caractères au lieu d'une valeur numérique (attendue).

Remarques :

— L'introduction des données peut être interrompue en frappant simultanément sur les touches [CNT] et [C]. En cas de reprise de l'exécution par l'instruction CONT il faudra réintroduire en entier les jeux de données.

— Un tableau s'introduit variable par variable.

— Une chaîne de caractères doit en principe être encadrée par des guillemets. Ces derniers ne sont pas utiles si la chaîne ne contient pas de caractères blancs (espacements) à son début et à sa fin.

Exemple 1:

Considérons le début de programme suivant :

```
1Ø INPUT "MISE, NOM" ; MISE%, NOMS
2Ø...
```

Si à l'exécution (après affichage de la question MISE, NOM ?) l'utilisateur tape 5Ø, JEAN [ENTREE] l'ordinateur place dans la mémoire MISE% la valeur 5Ø et la chaîne "JEAN" dans NOM$, et il n'y a pas d'erreur d'introduction.

Exemple 2 : Si avec le même début de programme que ci-dessus l'utilisateur tape

```
5Ø, JEAN, 3 [ENTREE]
```

l'ordinateur n'acceptera pas cette introduction car il y a trois données alors que deux seulement sont attendues.

* **LINE INPUT** (ou LINEINPUT)

Syntaxe :

$$\left\{\begin{array}{l}\text{[LINEINPUT]}\\ \text{[LINE INPUT]}\end{array}\right\} \text{ [;] [chaine de caractères] } \left\{\begin{array}{l};\\ \\ \text{[variable chaine]}\end{array}\right\}$$

Remarques :

— Cf première remarque de l'instruction INPUT

— Contrairement à l'instruction INPUT aucun point d'interro-

gation n'apparaît à l'exécution (seule la chaîne de caractères est affichée).

— La virgule et le point-virgule sont interprétés de la même manière par le MO5.

L'instruction LINE INPUT permet d'introduire une chaîne d'au plus 255 caractères. Cette chaîne n'a pas à être encadrée par des guillemets (contrairement à INPUT), et de plus elle peut comporter tous les caractères sans restriction. C'est donc une instruction adaptée à l'introduction de chaînes de caractères.

Exemple :

```
1Ø LINEINPUT A$
2Ø PRINT A$
```

A l'exécution si vous tapez

```
2 * 3 + 4 [ENTREE]
```

Vous obtiendrez à l'affichage :

```
2 * 3 + 4
OK
```

*** READ-DATA-RESTORE**

Syntaxe :

```
READ [variable] [,[variable]] ...
```

Cette instruction est à utiliser avec ceux autres, à savoir : DATA et RESTORE (DATA signifie : données).

Syntaxes :

```
DATA [constante] (,[constante]) ...
RESTORE ([numéro de ligne])
```

Parfois dans certains programmes il existe des jeux de données qui ne varient pratiquement jamais. Ils peuvent être introduits directement dans l'écriture du programme ; cela permet de ne pas avoir à les réécrire (réintroduire) à chaque exécution : c'est l'objet des trois instructions présentées ci-dessus.

L'instruction DATA permet d'inscrire les données dans le

programme. Il faudra créer une ou plusieurs ligne(s) de programme dans laquelle (lesquelles) la (les) donnée(s) seront inscrite(s) à la suite de DATA.

Exemple :

```
10Ø DATA 3.14159 , 365, "MOT MAGIQUE"
11Ø DATA "PI", 2.71828
```

L'instruction READ (qui signifie *lire* en anglais) permet de lire les données. A chaque instruction READ le MO5 lira dans les lignes repérées par DATA les premières valeurs qui n'auront pas déjà été lues (par une instruction READ antérieure). Il faut que le type de la variable (dans laquelle va être mis l'élément lu) et le type de la donnée soient compatibles.

Exemple :

```
 2Ø READ A, B$
 3Ø READ F%
  :
1ØØ DATA 1ØØ, "JO"
11Ø DATA "DAPHNIS", "ZERO", 32767
```

Le programme contient une erreur : il est demandé en ligne 2Ø et 3Ø de lire successivement un réel, une chaîne de caractères et un entier ; or les instructions DATA contiennent successivement un réel, trois chaînes de caractères et un entier. Au moment de la lecture de « l'entier » (NOTE%) le MO5 va trouver une chaîne de caractères, d'où le message d'erreur.

```
Error 2 in 11Ø
OK
```

Remarque : si le nombre d'éléments à lire est supérieur au nombre de données, l'ordinateur affiche

```
Error 4
OK
```

Cependant le MO5 peut recevoir l'ordre de relire plusieurs fois un même jeu de données : c'est le but de l'instruction RESTORE.

Après RESTORE la lecture reprendra :

— au début de la première instruction DATA si aucun numéro de ligne n'est précisé (cf. syntaxe).

— au début de la première instruction DATA située après le numéro de ligne si ce dernier est précisé (si ce numéro de ligne ne correspond pas à une instruction DATA, il faut qu'il corresponde à une ligne existant dans le programme, sinon le MO5 affiche :

Error 8

Exemple :

```
1Ø READ A, B$, NOM$, PI
 :
5Ø RESTORE 12Ø (RESTORE 13Ø serait plus correct au
   point de vue style de programmation)
6Ø READ PI
 :
 :
11Ø DATA 1ØØ, "JO", "DAPHNIS"
12Ø PRINT "BONJOUR"
13Ø DATA 3.1415
```

Après l'exécution de la ligne 5Ø le MO5 lira les données situées en ligne 13Ø (premier DATA situé après la ligne 12Ø).

Remarques :

— Pour changer les jeux de données, il faut les modifier dans le programme.

— Il est conseillé de rassembler les instructions DATA et de les mettre à la fin du programme.

*** INPUT$**

Cette fonction permet de lire une chaîne de caractères, et les caractères tapés ne sont pas affichés à l'écran. Cette dernière remarque induit l'utilisation de mots de passe, de codes (« secrets »), etc.

Exemple d'application : jeu du pendu, mastermind dans lesquels un joueur doit découvrir une combinaison secrète (qui doit demeurer masquée) :

```
1Ø CODE$ = INPUT$ (1Ø)
2Ø PRINT "LE CODE EST INTRODUIT : A VOUS DE
   JOUER"
3Ø :
```

La ligne 1Ø permet de mettre dans CODE$ une chaîne de 1Ø caractères tapée au clavier (celle-ci ne sera pas affichée à l'écran).

Remarques :

Dans ce dernier exemple, l'utilisateur pourra constater, en faisant tourner le programme, qu'à l'exécution de la ligne 1Ø l'ordinateur attend les 1Ø caractères ; dès qu'ils ont été tapés au clavier, le MO5 reprend l'exécution sans attendre que l'utilisateur tape sur la touche [ENTREE].

C'est le seul cas où la pression de la touche de validation [ENTREE] n'est pas utile.

Contrairement au cas des deux instructions précédentes, pour INPUT$ le nom de la variable dans laquelle va figurer la chaîne de caractères est situé avant le nom de l'instruction, et le signe = est inséré.

INSTRUCTIONS DE SORTIE

Elles sont destinées à extraire de l'unité centrale les résultats obtenus par le calculateur.

*** PRINT ou ? -TAB**

Syntaxe :

```
PRINT      [TAB(variable numérique)]
?          [expression]
```

Remarque : Le mot clé PRINT−115 peut être remplacé par un point d'interrogation. (?)

Exemple : PRINT A% est équivalent à ? A%

Elle permet d'afficher sur l'écran de contrôle des résultats ou des suites de caractères (dont la taille est à préciser grâce à l'instruction

ATTRB, cf. ci-après). Le terme *expression* situé dans la syntaxe est à prendre au sens large ; il signifie en même temps : expression logique, expression chaîne de caractères, expression numérique.

Exemples d'expressions :

1. 1Ø INPUT "MISE" ; MISE
   2Ø PRINT MISE>= Ø

Ce petit programme affiche – 1, ou Ø si la mise est négative.

2. 1Ø INPUT "CARACTERES" ; A$
   2Ø PRINT A$ + A$
3. En tapant
   PRINT (2 * 3 + 5 * 3)/7 [ENTREE] on obtient
   3
   OK

Remarque : les expressions peuvent contenir des fonctions telles que cosinus, sinus, etc. (cf. ci-après en ce qui concerne les fonctions).

La fonction TAB à laquelle il est fait référence (dans la syntaxe) permet d'écrire exactement à l'endroit désiré, sans restriction. Les lignes sur l'écran de télévision comportent 4Ø positions, elles sont notées de Ø à 39. L'insertion de l'instruction TAB(I) où I est une grandeur numérique comprise entre Ø et 39, permettra de positionner le curseur en place I.

Remarques sur les valeurs de I ; si :

— elle n'est pas entière, le MO5 prendra l'entier le plus proche ;
— elle est strictement négative, il la considérera comme nulle ;
— elle dépasse 39, la valeur prise sera le reste de la division entière de I par 4Ø.

Exemple :

PRINT TAB(25) "BONJOUR" [ENTREE]

permettra d'imprimer le mot BONJOUR à partir de la position 25.

En utilisant l'instruction PRINT seule (sans TAB), trois formes d'impression sont possibles, à savoir :

• Imprimer tout à la suite sans espaces. Cette impression s'obtient en séparant les différents éléments à imprimer par des points-virgules.

Exemple :

```
PRINT "BONJOUR" ; "MONSIEUR" [ENTREE]
```

donne

```
BONJOURMONSIEUR
OK
```

Remarque :

Dans un programme, si le dernier élément à imprimer est suivi d'un point-virgule, la prochaine impression se fera à la suite, sans espace, sinon il y aura passage à la ligne.

Exemples :

1.
```
10 PRINT "BONJOUR" ;
20 PRINT "COMMENT ALLEZ-VOUS ?"
30 PRINT "MONSIEUR"
```

A l'exécution le programme donne :

```
BONJOUR COMMENT ALLEZ-VOUS ?
MONSIEUR
OK
```

2.
```
10 PRINT "BONJOUR" ;
20 PRINT "MONSIEUR"
```

L'éxécution donne

```
BONJOURMONSIEUR
OK
```

• Imprimer avec des espaces entre chaque élément. Il faut

utiliser des virgules à la place de points-virgules. Le MO5 fractionne en trois zones les 40 positions d'affichage.

la première va de Ø à 13 (inclus)
la seconde va de 13 à 25 (inclus)
la troisième va de 26 à 38 (inclus)

Avec la présence d'une virgule, lorsque l'impression d'un élément est terminée, il passe au début de la zone suivante pour en imprimer d'autres.

Remarque :

Si dans un programme le dernier élément à imprimer (d'une instruction PRINT) est suivi d'une virgule, l'affichage suivant se fera dans la zone suivante sinon il y a passage à la ligne.

Exemple :

```
1Ø PRINT "AU REVOIR", "MONSIEUR_";
2Ø PRINT "A"
3Ø PRINT "BIENTOT"
```

Ce programme donne à l'exécution :

```
AU REVOIR  MONSIEUR A     BIENTOT
↑          ↑              ↑
Ø          13             26
```

Remarques :

— Le MO5 « passe » à ligne automatiquement lorsqu'il n'y a plus assez de place sur une ligne pour afficher ce qui est demandé.

— Les valeurs numériques ne peuvent pas être affichées sur plus d'une ligne ; elles sont précédées d'un

signe – si elles sont négatives
blanc si elles sont positives

• Imprimer une ligne sans caractère (i.e. faire un saut de ligne). Il suffit de placer (d'introduire) l'instruction PRINT « seule ».

PRINTUSING

Syntaxe :

PRINTUSING [expression chaine];[X](;([Y]))...

X et Y sont des expressions numériques

Elle offre beaucoup de possibilités d'affichage. Il faut introduire dans l'instruction la description de la façon de disposer l'affichage. Cette description est faite au moyen d'une image (qui est constituée par l'expression chaîne cf. syntaxe).

Par exemple le MO5 peut afficher les résultats avec un certain nombre de chiffres après la virgule, ou encore utiliser la notation dite scientifique etc.

Une image est contituée par une chaîne de caractères. Il peut y avoir plusieurs images dans ce cas : elles seront placées les unes à la suite des autres. L'image ou l'ensemble des images (suivant le cas) sera placé entre guillemets.

Elle(s) est(sont) constituée(s) par des caractères à choisir parmi :

et . servent à définir le nombre de chiffres à inscrire avant et après « la virgule » (ceci définit le format de base de l'affichage)

Exemple : PRINT USING "##.#";3.14[ENTREE]
donne
3.1
OK

Remarque : Si le nombre à afficher comporte plus de chiffres avant (resp après) la « virgule » que celui autorisé par le format de base

(ex. : PRINTUSING "#" ; 31 (resp PRINTUSING "#.#" ; 3.14)) le MO5 fait précéder l'affichage du nombre du symbole % (resp. : le MO5 arrondi le résultat).

+ oblige le MO5 à faire précéder l'affichage de tout nombre de son signe.

Exemple : PRINT USING " + #.# ; 3.38 [ENTREE]
donne
+ 3.4
OK

* ordonne à l'ordinateur de compléter par la gauche l'affichage du nombre par des astérisques si celui-ci comporte moins de chiffre avant la « virgule » que celui indiqué dans le format de base

Exemple : PRINTUSING "*###" ; 32 [ENTREE]
donne
*32
OK

Remarque : Les symboles + et * sont à placer avant la description du format de base (cf. ci-dessus)

^ fait passer en notation exponentielle

Exemple : PRINT USING "#.#^" ; 32 [ENTREE]
donne
3.2 E+Ø1
OK

Les images données sont utilisées les unes à la suite des autres. Si elles ont été toutes utilisées et qu'il reste encore des informations à afficher, le MO5 reprend le début de la suite des images jusqu'à épuisement des éléments à imprimer.

Exemple : PRINT USING "#.#" ; 3.14;3.16 [ENTREE]
donne

3.1 3.2
OK

Il y a une image (à savoir #.#) et deux éléments à afficher, le MO5 se sert « deux fois de l'image » ; après l'affichage de 3.14 il reprend l'image.

Remarques :

— Si aucune image n'est fournie l'erreur de syntaxe 2 est détectée.

— Il est possible d'insérer dans les images des chaînes de caractères qui seront considérées comme des commentaires à l'affichage

Exemple : PRINT USING "####.## FRANCS"
1ØØ.1 [ENTREE]
donne
1ØØ.1Ø FRANCS
OK

Les images données sont utilisées les unes à la suite des autres. Si elles ont été toutes utilisées et qu'il reste encore des informations à afficher, le MO5 reprend le début de la suite des images jusqu'à épuisement des éléments à imprimer.

Exemple : PRINT USING "#.#"; 3.14;3.16 [ENTREE]
donne

*** ATTRB**

Syntaxe ATTRB [X], [Y]

Elle permet de définir la taille des caractères affichés sur l'écran de télévision. Les valeurs possibles pour X et Y sont Ø et 1.

X définit la largeur des caractères
(Ø largeur normale ; 1 largeur double)

Y définit la hauteur des caractères
(Ø : hauteur normale ; 1 hauteur double)

Remarque :

— Lors de l'utilisation de caractères de largeur double une ligne ne contient que 2Ø (soit 4Ø/2) caractères au maximum (cf. instruction PRINT).

Exemple :

1Ø ATTRB , 1

Ordonne à l'ordinateur d'imprimer les caractères avec une hauteur double.

*** CLS**

Syntaxe CLS

Elle nettoie la fenêtre de travail (à l'écran) et elle positionne le curseur en haut et à gauche de cette fenêtre.

(Exemple : cf. chapitre 1, jeu du 21).

2.3.5 Commandes de mise au point d'un programme

Dans la suite de ce paragraphe les termes fichier et descripteur de fichier vont être cités assez souvent. Ces termes seront décrits en détail dans le paragraphe « Entrée-Sorties Généralisées ». Pour l'instant on retiendra les définitions suivantes :

— Fichier, ou plus exactement nom de fichier, représente le nom sous lequel est mémorisé (sur cassette, disquette, ...) ou transféré (vers la voie série, imprimante, écran, ...) un programme ou des données.

Un nom de fichier est constitué par une suite de six caractères suivis d'un point et d'un suffixe optionnels indiquant s'il s'agit d'un fichier BASIC (BAS), de Données (DAT) ou Binaire (BIN). (Binaire se rapporte aux programmes écrits en langage machine).

Exemple : Un programme de jeu peut être mémorisé sur cassette sous le nom de fichier « JEUX.BAS ».

— Descripteur de fichier

Il contient le nom de fichier et le nom du type de périphérique (cassette (CASS), imprimante (LPRT), erreur auteur écran (SCRN), etc.) dans lequel est situé ou transféré le programme ou les données. L'ensemble de ces deux noms est placé entre des guillemets.

Exemple : "CASS : JEUX.BAS"

périphérique fichier

*** LIST**

LIST a deux utilisations possibles.

LIST ([numéro de ligne])(−[numéro de ligne])

Celle-ci permet de lister sur l'écran tout ou partie du programme qui se trouve en mémoire centrale

Exemples :

LIST [ENTREE] imprime tout le programme
LIST 4Ø–5Ø liste les lignes 4Ø à 5Ø
LIST –5Ø liste de la ligne 1 à la ligne 5Ø
LIST 5Ø– liste de la ligne 5Ø à la fin
LIST ([descripteur de fichier])([plage de numéros de ligne])

Elle permet d'imprimer sur le périphérique concerné tout ou partie du programme situé en mémoire centrale : celui-ci sera placé dans le fichier indiqué (dans le descripteur).

Remarques :

1. Cette commande est utilisable avec : l'écran, le lecteur-enregistreur de cassettes, l'imprimante parallèle, et la voie série.

2. Si aucun périphérique n'est mentionné l'ordinateur considère d'autorité l'écran.

3. Les lignes à imprimer sont déterminées par [plage de numéro de ligne].

Exemple :

LIST "CASS : ESSAI.ONE", 5Ø–1ØØ

permet d'imprimer sur cassette les lignes 5Ø à 1ØØ du programme situé en mémoire centrale. Le nom de fichier sera ESSAI.ONE.

*** DELETE**

Syntaxe :

DELETE ([numéro de ligne]) – ([numéro de ligne])
[numéro de ligne]

Elle est utilisée en mode calcul uniquement, et elle permet de détruire les lignes de programme mentionnées dans l'instruction.

Exemples :

DELETE – [ENTREE] détruit tout le programme
DELETE 4Ø–5Ø supprime les lignes 4Ø à 5Ø
DELETE 45– supprime toutes les lignes à partir de la ligne 45

*** END**

C'est l'instruction qui arrête l'exécution d'un programme

Remarques :

— Cette instruction ferme les fichiers ouverts en cours de programme.

— Si elle n'est pas toujours nécessaire, il vaut mieux prendre l'habitude de l'utiliser, du moins pour les débuts en programmation

Exemples :

1.
```
10 PRINT "BONJOUR,,
20 END (ici END n'est pas utile)
```
2.
```
100 GOSUB 1000
  :
1000
  :
1100 RETURN
```

Si aucun END n'est placé dans le programme (entre les lignes 0 et 199 cf. exemple 2), l'ordinateur considérera, lors de l'exécution, les lignes 1000 à 1100 comme étant du programme, ce qui conduit le MO5 à afficher :

```
In 1100  (il y a un return sans GOSUB)
Error 3
```

*** STOP**

Cette instruction peut être utilisée au clavier (touche [STOP]) ou dans un programme (instruction STOP).

- Au clavier elle stoppe l'exécution du programme. Pour reprendre l'exécution il suffit de taper sur l'une quelconque des touches à l'exclusion de CNT.
- Dans un programme elle arrête définitivement l'exécution et le message suivant apparaît

Break in [numéro de ligne]

*** CONT**

(Mis pour CONTinue) (Syntaxe CONT)

Cette instruction :

— permet de reprendre l'exécution d'un programme s'il a été stoppé ; dans ce cas elle reprend à partir de l'endroit où le MO5 s'est arrêté.

— n'est pas utilisable en mode programme.

*** REM**

Syntaxe REM [texte]

Cette instruction permet d'introduire du texte dans un programme (le texte est écrit à la suite de REM). Il faut autant de REM qu'il y a de lignes de commentaires.

Exemple : cf. chapitre 1, jeu du 21.

Remarque : REM peut être remplacé par une apostrophe.

*** TRON-TROFF**

Elles permettent de passer en mode Trace, et de le quitter. Le mode Trace se caractérise par le fait que l'ordinateur affiche à l'écran chaque numéro de ligne exécutée. Ce mode est très utile pour la recherche d'erreurs dans un programme. En effet il permet de vérifier que l'ordre d'exécution des instructions est en accord avec celui que désire le programmeur.

Les instructions suivantes sont relatives à la sauvegarde de programmes, au listage de programmes, etc.

*** NEW**

Syntaxe : NEW

Elle vide de son contenu la partie de la mémoire réservée à l'utilisateur. Ainsi le programme et toutes les variables sont détruits.

Remarque : les commandes faites par une instruction CLEAR antécédente ne sont pas oubliées par le MO5. En effet, celles-ci ne sont pas mémorisées dans la zone mémoire réservée à l'utilisateur.

*** MERGE**

Syntaxe : MERGE ([descripteur de fichier](,R))

Dans certains cas il est intéressant de fusionner deux programmes (ex. : un programme est enregistré sur cassette, un autre vient d'être écrit et il utilise le premier comme sous-programme ; dans ce cas il est inutile de taper le premier programme une deuxième fois). En effet avec la commande MERGE le MO5 cherche sur le périphérique indiqué le fichier désiré (ces deux derniers étant ceux mentionnés dans le descripteur de fichier). Lorsqu'il l'a trouvé, il le place en mémoire programme tout en conservant celui qui s'y trouvait déjà, puis lance l'exécution si ,R est ajouté.

Remarques :

1. Les seuls périphériques utilisables avec cette instruction sont
- le lecteur-enregistreur de cassettes,
- L'unité de disquette.

2. Si aucun fichier n'est mentionné dans le descripteur, le MO5 prend le premier (fichier) trouvé (sur le périphérique).

Si aucun périphérique n'est indiqué (dans le descripteur de fichier) l'ordinateur considérera d'autorité le lecteur-enregistreur.

3. Seuls les programmes conservés dans les périphériques sous forme ASCII (cf. instruction SAVE) sont fusionnables ; pour les autres l'erreur 51 est détectée.

4. Si les deux programmes contiennent des lignes de programme ayant les mêmes numéros, le MO5 ne conservera que celles issues du programme stocké sur le périphérique (celles du programme principal sont éliminées).

*** RUN**

RUN a deux utilisations possibles :

1. RUN ([numéro de ligne]) lance l'exécution du programme, chargé en mémoire centrale, à partir de la ligne indiquée par [numéro de ligne] si elle est précisée, sinon à partir de la ligne 1.

2. RUN [descripteur de fichier] (,R) cherche dans le périphérique mentionné le fichier désiré (ces deux derniers sont à préciser

dans le descripteur de fichier). Après avoir trouvé il lance l'exécution.

Remarques :

— cf. remarque 1 commande MERGE (ci-avant)
— cf. remarque 2 commande MERGE (ci-avant)
— la commande RUN ferme tous les fichiers qui étaient déjà ouverts, sauf si ,R est ajouté.

Exemple :

RUN "CASS : ESSAI.ONE"

*** SAVE**

Syntaxe :

SAVE [descripteur de fichier] $\left\{\begin{matrix},(A)\\,(P)\end{matrix}\right\}$

Cette instruction permet de sauvegarder (i.e. : enregistrer) le programme qui se trouve en mémoire centrale sur le périphérique mentionné et sous le nom de fichier indiqué.

Remarques :

1. Les périphériques utilisables avec cette instruction sont : l'écran, le lecteur-enregistreur, l'unité de disquette, l'imprimante parallèle.

2. Cf. remarque 2 commande MERGE.

3. Si le terme optionnel A (resp P) est indiqué, le MO5 conservera le programme sous forme ASCII (resp sous forme protégée). Dans ce dernier cas,

- le listage
- la modification de programme
- l'utilisation des fonctions PEEK et POKE (sur les mémoires concernant le programme)

seront impossibles.

4. Le nom de fichier doit toujours être mentionné dans le descripteur de fichier, sans quoi le MO5 affiche

Error 55

Exemple :

SAVE "CASS" : JEU" ,A

conserve un programme sous forme ASCII sur la cassette, le nom étant JEU.

*** SAVEM**

Syntaxe :

SAVEM "nom de fichier]", [adresse 1], [adresse 2], [adresse d'exécution]

Elle permet de sauvegarder sur cassette un programme binaire ou une image écran.

— Sauvegarde de programme binaire

Le programme est situé dans les mémoires [adresse 1], [adresse 1]+1, ... jusqu'à [adresse 2]

L'adresse d'exécution du programme étant [adresse exécution].

Exemple : 1Ø SAVEM "ESSAI" &H6200, &H7520, &H0000 permet de sauvegarder un programme inscrit dans les mémoires &H6200, ..., &H75A0 du fichier ESSAI, l'adresse d'exécution étant &H0000

— Sauvegarde d'image écran

Se reporter au paragraphe 3.5.3

*** LOAD**

Syntaxe : LOAD "[nom de fichier]" [, R]

Elle permet de chercher, dans le périphérique mentionné, le fichier indiqué (dans le descripteur de fichier). Les fichiers, rencontrés par le MO5, qui ne correspondent pas à celui désiré, sont mentionnés sur l'écran par le message Skip [nom de fichier]. Tant que le fichier n'est pas trouvé l'ordinateur affiche également le terme : Searching. Dès qu'il l'a trouvé, il écrit : FOUND [nom de fichier]. Puis il le charge en mémoire programme, après quoi il indique OK si aucune erreur n'a été rencontrée ; dans le cas contraire il affiche Error 53 il faut alors recommencer l'opération.

Remarques :

1. Cf. remarque 1 commande MERGE
2. Cf. remarque 2 commande MERGE
3. Si le fichier n'est pas trouvé après examen de tout ce qui se trouvait dans le périphérique, il faut réinitialiser le MO5.
4. Cf. remarque 3 commande RUN.

*** LOADM**

Syntaxe :

LODAM ("[nom du fichier]")((,[décalage])(,R))

Elle permet de charger un programme binaire ou une image écran dans la mémoire centrale. Le terme [décalage] indique de combien de cases mémoires le chargement doit être décalé en mémoire. Elle n'est utilisable qu'avec le lecteur-enregistreur.

Exemple :

LOADM "ESSAI", &HØ1ØØ

charge en mémoire centrale le fichier ESSAI et déplace son chargement de 1ØØ cases mémoires (cf. SAVEM).

Remarque : Cf. remarque 3 de RUN.

*** MOTORON-MOTOROFF**

Elles permettent respectivement de faire défiler et d'arrêter le défilement du lecteur-enregistreur. Ce dernier doit être connecté et les touches « lecture » ou « lecture et enregistrement » doivent être enfoncées.

2.4 NOTIONS DE SOUS-PROGRAMME ET FONCTIONS

2.4.1 Sous-programmes :

Malheureusement il est difficile, sinon impossible, de donner une méthode systématique de création de sous-programmes. La meilleure façon de l'apprendre est de s'y entraîner. Cependant des

critères généraux permettent de savoir si la solution obtenue est acceptable. Une énumération non limitative des règles est proposée ci-dessous :

— Le fractionnement du programme en sous-programmes doit être effectué lors du passage de l'organigramme au programme.

— Les tailles des différents sous-programmes ne doivent pas être trop disparates. Sans quoi il faut vérifier :

- si les petits sous-programmes ne réalisent pas des fonctions ridiculement simples.
- si les gros ne peuvent pas être à leur tour fractionnés en plusieurs parties.

— Un sous-programme doit réaliser une seule fonction à la fois.

— La fonction réalisée doit être indépendante de celles du reste du programme.

— Cette fonction doit être relativement simple.

— Il faut éviter qu'un sous-programme utilise des sous-sous-sous programmes etc.

Parmi les sous-programmes qu'il est possible d'insérer dans un programme, l'utilisateur peut en utiliser de trois types :

— Les sous-programmes écrits à la suite du programme principal (cf. instructions GOSUB RETURN).

— Les sous-programmes écrits en langage machine (cf. chapitre 5).

— Les sous-programmes préenregistrés dans le MO5.

2.4.2 Fonctions

Celles-ci sont appelées des sous-programmes préenregistrés dans le MO5. Une liste de ces fonctions est fournie ci-après.

* **ABS(X)**

— calcule la valeur absolue de X

* **ASC(X$)**

— donne le code ASCII du premier caractère de la chaîne X$

* **CHR$(X)**
— convertit le code ASCII en le caractère associé (pour ceci il faut se référer au codage ASCII)

* **COS(X)**
— cosinus (X)

* **CSRLIN**
— donne le numéro de ligne sur lequel se trouve le curseur

* **EOF** (numéro de canal)
— permet de savoir si le fichier qui est connecté par le canal (indiqué par numéro de canal) est vide ou non (cf. §2.5).

* **EXP(X)**
— exponentielle de X

* **FIX(X)**
— permet de supprimer la partie décimale de X ;
exemple : FIX(PI) = 3 où PI = 3.14159

* **FRE (Ø)**
(X$)
— elle donne la taille disponible en octets de la mémoire centrale

* **INKEY$**
— indique quel a été le dernier caractère frappé sans l'afficher

* **INSTR ((I,)X$,Y$)**
— elle permet de repérer la première apparition de la chaîne Y$ dans la chaîne X$ à partir du I^{e} caractère de la chaîne X$ si I est présent ; exemples
INSTR ("Ø122","2") = 3
INSTR (4,"Ø122","2") = 4

* **INT(X)**
  — donne la partie entière de X

* **LEFT$(X$,I)**
  — tronque la chaîne X$ à partir du I^e^ caractère ; exemple : LEFT$ ("ABCD",2) = "AB"

* **LEN(X$)**
  — donne la longueur de la chaîne X$; exemple : LEN("1234") = 4

* **LOG(X)**
  — donne le logarithme népérien de X

* **MID$**
  Syntaxe : MID$ ([variable chaîne de caractères], n(,m))
  n et m sont des expressions numériques
  — Elle permet de ne conserver dans la variable chaîne de caractères
    - que les caractères numéro n, n + 1, n + 2, ...,n + m
    - tous les caractères situés après le nième inclus) au cas où m est omis

  Exemple :

```
1Ø A$ = "HYDROGLISSEUR"
2Ø A$ = MID$ (A$,1,4) + "avion"
3Ø PRINT A$
4Ø PRINT MID$ (A$,2,6)
```

  Ce programme donne

```
HYDRAVION puis
YDRAVI
```

* **POS (([numéro de canal]))**
  — elle indique la position du curseur dans le fichier relié au

MO5 par le canal indiqué (ou le canal Ø si le numéro n'est pas mentionné) cf. § 2.5

* **RIGHT$(X$,I)**
  — extrait de la chaîne X$ les I derniers caractères ; exemple RIGHT$ ("ABCD",3) = "BCD"

* **RND(X)**
  — donne un nombre réel aléatoire entre Ø et 1 exclus

* **SGN(X)**
  — donne le signe de X

* **SIN(X)**
  — donne le sinus de X

  SPC(X)
  — écrit X blancs

* **SQR(X)**
  — racine carrée de X

* **STR$(X)**
  — convertit X en une chaîne de caractères exemple : STR$(15) = "15"

* **TAN(X)**
  — tangente de X

* **VAL(X$)**
  — convertit la chaîne de caractères en une valeur numérique ; exemple : VAL ("36") = 36

Ces fonctions ne sont pas fondamentales. Elles doivent être approfondies au fur et à mesure des besoins que le programmeur pourra ressentir lors de l'écriture de programmes.

2.5 ENTRÉES-SORTIES GÉNÉRALISÉES

Les instructions étudiées précédemment permettent le dialogue entre l'utilisateur et l'ordinateur (en fait entre le clavier et l'unité centrale). Pour disposer de possibilités plus nombreuses, il est nécessaire de pouvoir établir entre l'ordinateur (l'unité centrale) et des périphériques le dialogue indispensable pour enregistrer des programmes, imprimer des messages sur l'écran, utiliser des imprimantes, des lecteurs de disquettes etc. Pour permettre ces dialogues, il faut disposer de voies de communication ou canaux ; il y en a dix-sept sur le MO5.

Parmi ceux-ci :

— le canal Ø est ouvert en permanence et de manière automatique. C'est par son intermédiaire que l'utilisateur peut dialoguer directement avec l'ordinateur et que ce dernier peut dialoguer avec l'écran pour l'affichage.

— Les seize autres canaux s'ouvrent sur demande.

Remarque : les instructions précédemment étudiées utilisent le canal Ø.

2.5.1 Ouverture - fermeture

Pour les autres dialogues possibles entre l'ordinateur et les périphériques il faut que l'utilisateur ouvre des canaux. Un canal par type de dialogue est nécessaire.

Cependant, pour faciliter l'utilisation des périphériques, l'ouverture d'un canal se fait automatiquement pour certaines opérations (la fermeture étant opérée systématiquement dès la fin des opérations). Ce cas se présente pour tout ce qui concerne des *programmes* avec le lecteur-enregistreur (pour sauvegarder ou charger (i.e. enregistrer ou lire) un programme) ; avec l'imprimante (pour lister des programmes) ; etc...

Par contre pour enregistrer ou lire des *données* (en utilisant le lecteur-enregistreur, etc...), il est nécessaire d'ouvrir les canaux et de les refermer sitôt l'opération effectuée en utilisant les instructions OPEN CLOSE.

*** OPEN-CLOSE**

Syntaxes :

OPEN { ("O") / ("I") } ,(#)[numéro de canal],[descripteur de fichier]
CLOSE ((#)[numéro de canal])...

— Pour OPEN

a) Le choix entre O et I se fait de la manière suivante, si le canal doit transmettre des informations :

- de l'ordinateur vers le périphérique, il faut choisir "O" (Output)
- du périphérique vers l'ordinateur, il faut choisir "I" (Input).

b) Le numéro de canal est à choisir entre 1 et 16. Il doit être utilisé autant d'instructions Open qu'il y a de canaux à ouvrir.

c) Le terme [*descripteur de fichier*] contient les informations relatives au périphérique, au fichier, etc... Il doit contenir en général :

- le type de périphérique suivi de deux points, à savoir :

  KYBD : (pour le clavier)
  SCRN : (pour l'écran)
  LPRT : (pour l'imprimante)
  CASS : (pour le lecteur-enregistreur ; qui peut être omis, cf. remarque).

Remarque : si le type de périphérique n'est pas précisé dans le descripteur de fichier, le MO5 considère d'autorité le lecteur-enregistreur.

- Une constante numérique (comprise entre Ø et 255) qui, placée entre parenthèses, détermine le nombre de caractères par ligne. Elle ne doit être introduite que pour une imprimante.
- Le nom de fichier sous lequel vont être rassemblées les informations transmises. Il est constitué par :

— un nom qui comporte de un à huit caractères à l'exception de la virgule, du Ø, du nombre 255 et du point ;

— et un suffixe *optionnel* séparé du nom par un point. Ce suffixe comporte de 1 à 3 caractères répondant aux mêmes restrictions que celles relatives au nom (cf. ci-dessus).

Remarque : si le suffixe n'est pas précisé le MO5 en ajoute un, selon le cas :

DAT s'il s'agit de données
BIN s'il s'agit d'un fichier binaire
BAS s'il s'agit d'un fichier programme

N.B. : le descripteur de fichier est à mettre entre guillemets.

Exemples :

1. Descripteurs de fichier
   "LPRT : (4Ø) ESSAI.ONE"
   "PGM.BON" (ce dernier est équivalent à "CASS PGM.BON")

2. Utilisation de l'instruction Open
   OPEN "I", # 1, "CASS : ESSAI.TWO"

Cette instruction ouvre le canal 1 en I(nput) pour communiquer avec la cassette sur le fichier ESSAI.TWO.

Remarque : si vous essayez d'ouvrir un canal déjà ouvert, le message suivant apparaît :

Error 52

— L'instruction CLOSE ferme les canaux indiqués. Les différents numéros de canaux doivent être séparés les uns des autres par une virgule (ils sont entiers et compris entre 1 et 16).

Exemple :

CLOSE 1, 3, 15

La demande de fermeture d'un canal déjà fermé n'entraîne pas de message d'erreur. Par contre un numéro de canal incorrect provoque l'affichage suivant :

Error 5Ø

Les canaux inutilisés doivent être fermés.

Remarque : L'instruction CLOSE seule ferme tous les canaux ouverts.

2.5.2 Entrée de données

*** Instructions INPUT#, LINEPUT# et INPUT$**

Syntaxes :

INPUT# [numéro de canal] , [variable] (,[variable])
LINEINPUT#[numéro de canal] , [variable chaîne]
INPUT$([A] (,(#) [numéro de canalB]))

Remarque préliminaire : pour ces instructions il faut nécessairement qu'une ouverture de canal ait été effectuée avant de les utiliser ; sans quoi l'erreur suivante est détectée :

Error 57 (In ...)

Cette ouverture doit être faite en Input.

L'étude portera successivement sur les deux premières puis sur la troisième.

*** INPUT# LINE INPUT#** (avec ou sans blanc entre LINE et INPUT)

Elles s'utilisent de la même manière qu'INPUT et LINEINPUT, mais il ne faut pas oublier le symbole # suivi du numéro de canal.

Exemple 1 :

```
1Ø OPEN "I" ,#1, "LYCEE"
2Ø INPUT # 1, NOM$, NOTE, CLASSEMENT
3Ø PRINT NOM$ ; ":" ; NOTE ; ":" ; CLASSEMENT
4Ø CLOSE 1
```

Ce programme permet de lire dans le fichier LYCEE (inscrit sur cassette) une chaîne de caractères, et deux valeurs numériques qui vont respectivement être placées dans NOM$, NOTE et CLASSEMENT (ligne 2Ø).

Exemple 2 :

```
1Ø OPEN "I", #1, "KYBD : NOMS"
2Ø LINE INPUT #1, NOM$
```

```
3Ø PRINT NOM$
4Ø CLOSE 1
```

Ce programme "permet" le dialogue entre le clavier et l'ordinateur (soit entre l'utilisateur et l'ordinateur) par le moyen du canal 1.

Remarques :

- Le symbole # n'est pas obligatoire pour OPEN et CLOSE mais il l'est pour INPUT # et LINEINPUT #
- L'ouverture de canaux entre le lecteur-enregistreur et l'ordinateur nécessite la commutation :

— de la touche lecture pour une ouverture en Input ;

— des touches lecture et enregistrement pour une ouverture en Output.

* **INPUT$** (nombre, canal)

Exemple :

```
1Ø OPEN "I", 3, "NOMS"
2Ø NOM$ = INPUT$ (5,3)
3Ø CLOSE 3
```

La ligne 4Ø ouvre le canal 3 en Input pour accéder au fichier NOMS inscrit sur cassette.

La ligne 2Ø permet de récupérer les cinq premiers caractères figurant dans ce fichier (NOMS ; car le canal 3 est ouvert sur ce fichier ; cf. ci-avant).

2.5.3 Fonction EOF *(End of File)*

Syntaxe :

EOF ([numéro de canal])

Lorsque des canaux sont ouverts *en Input* il est utile de pouvoir déterminer s'il reste des informations dans le fichier avec lequel l'unité centrale dialogue. Ceci permet d'éviter de lire un fichier vide ou vidé (par suite de lectures successives), ce qui conduirait au message d'erreur Error 54. EOF réalise cette détermination.

Elle renvoie la valeur :

— vrai (−1) si la fin du fichier est atteinte.

— faux (Ø) sinon

Cette instruction est toujours utilisée avec une instruction IF − THEN (ELSE)

Exemple :

```
1Ø OPEN "I", 15, "DONNEES"
2Ø IF EOF(15) THEN 6Ø
3Ø INPUT#15, DONNEE
4Ø PRINT "DONNEE"
5Ø END
6Ø PRINT "FICHIER DONNEES VIDE"
7Ø END
```

Ce programme permet de lire dans le fichier DONNEES, situé sur cassette, à travers le canal 15. Si ce fichier contient un élément l'ordinateur l'imprime, sinon il écrit FICHIER DONNEES VIDE.

2.5.4 Sortie des données

PRINT# PRINT#USING

Syntaxes :

```
PRINT# [numéro de canal],([expression])
PRINT# [numéro de canal],USING [chaîne image]
          ;([expression])...
```

La différence entre ces deux instructions et les deux précédentes est la même que celle entre INPUT et INPUT #. Il s'agit de l'utilisation d'un canal différent du canal Ø pour pouvoir par exemple enregistrer des données dans un fichier sur cassette.

Pour ces deux instructions, il ne faut pas oublier :

— d'avoir un canal en Output avant de les utiliser sans quoi l'erreur Error 57 apparaît ;

— de refermer le canal dès qu'il n'est plus utilisé.

Remarque : avec l'utilisation du lecteur-enregistreur, il ne faut pas oublier d'enfoncer simultanément les touches lecture et enregistrement.

Exemple :

Pour enregistrer des données, rentrées au clavier, sur cassette, on peut partir du programme suivant :

```
1Ø OPEN "O", 1, "DONNEES"
2Ø INPUT "VALEUR" ; A
3Ø PRINT # 1, A
4Ø INPUT "VOULEZ-VOUS ENREGISTRER D'AU-
   TRES DONNEES", REPONSE$
5Ø IF REPONSE$ <>"OUI" THEN 6Ø ELSE GOTO 2Ø
6Ø CLOSE 1
7Ø END
```

2.6 LE CRAYON OPTIQUE

2.6.1 Introduction

Le crayon optique permet de dessiner des figures variées directement sur l'écran du téléviseur. Son emploi peut être comparé à celui d'un crayon à papier sur une feuille, la feuille étant l'écran et le crayon à papier le crayon optique.

2.6.2 Principe général

Il peut paraître surprenant au premier abord qu'un crayon apparemment quelconque puisse autoriser une telle opération, la première question venant à l'esprit étant : « Comment le téléviseur peut-il savoir que le crayon se trouve là ? » La réponse est la suivante :

L'unité centrale du système MO5 génère elle-même l'image de télévision. Pour cela, il lui faut allumer un à un chaque point de l'écran de télévision dans un ordre rigoureux. L'ordre d'allumage est de haut en bas et de gauche à droite ; pour que l'œil puisse percevoir ce phénomène, il est nécessaire d'éclairer chaque point 25 fois par seconde.

Un œil suffisamment rapide observerait donc un à un les points de l'écran s'allumer et s'éteindre, il lui serait facile à un instant donné de vérifier qu'un point de son choix est noir ou coloré. Cet

œil existe, c'est l'élément photosensible du crayon lumineux que l'on peut observer au fond de son logement sous l'interrupteur. Le rôle de l'élément photosensible est d'avertir l'unité centrale lorsque le point situé à proximité du crayon optique est allumé. L'unité centrale en déduit alors facilement le numéro du secteur que pointe le crayon optique, il sera ensuite évident de changer la couleur du secteur correspondant par programme.

2.6.3 Les instructions spécifiques au crayon optique

L'utilisation du crayon optique n'est possible que si des instructions permettent de déterminer sa position sur l'écran. Ces dernières sont décrites ci-dessous.

*** INPEN et INPUTPEN**

Ces deux instructions permettent de relever la position du crayon optique sur l'écran sous forme de coordonnées graphiques.

La syntaxe est :

```
    INPEN X, Y
ou  INPUTPEN X, Y
```

La coordonnée horizontale est affectée à la variable X.

La coordonnée verticale est affectée à la variable Y.

Si l'opération se déroule normalement, on a :

$$\emptyset \leqslant X \leqslant 319 \text{ et } \emptyset \leqslant Y \leqslant 199$$

Dans le cas contraire X et Y se voient attribuer la valeur −1. Ce dernier cas provient soit d'un manque de luminosité de l'image soit du fait que le crayon optique se trouve hors de l'écran.

INPEN X, Y provoque l'affectation immédiate des coordonnées aux variables X et Y alors que INPUTPEN X, Y n'effectue cette opération que lorsque l'interrupteur qui se trouve à l'extrémité du crayon est enfoncé.

Exemple 1 :

```
1Ø SCREEN Ø, 7, Ø : CLS
2Ø INPEN A, B
3Ø CLS : PRINT A, B
4Ø FOR I = 1 TO 5ØØ : NEXT I
5Ø GOTO 2Ø
```

Ce programme colore la fenêtre de travail en blanc puis affiche à intervalles de temps réguliers les valeurs de A et B. En déplaçant le crayon lumineux sur l'écran, les valeurs A et B se modifient en fonction des coordonnées. Pour stopper le programme, presser simultanément les touches [CNT] et [C].

Exemple 2 :

```
10 SCREEN 0, 7, 0 : CLS
20 INPUTPEN X, Y
30 CLS : PRINT X, Y
40 GOTO 20
```

Le rôle de ce programme est identique au précédent. On constatera toutefois qu'il faut presser le crayon lumineux contre l'écran de télévision pour changer les valeurs de X et Y.

Ces deux exemples mettent en évidence le rôle des instructions INPEN et INPUTPEN ainsi que leurs différences.

Remarque : En fait, la coordonnée en X n'est pas donnée point par point mais secteur par secteur ainsi, la valeur la plus basse de X sera 3 (en dehors de −1) et X ne prendra que des valeurs de la forme 3 + H * 8 ($0 \leqslant H \leqslant 39$) ; H étant le numéro du secteur sur la ligne Y.

Le programme suivant permet de dessiner sur l'écran à l'aide du crayon optique. Pour sortir du programme il suffit de presser la touche [S].

```
10 SCREEN 0, 7, 0 : CLS
20 IF INKEY$ = "S" THEN GOTO 70
30 INPEN U, V
40 IF (U < 0) or (V < 0) THEN GOTO 20
50 PSET (U,V), -5
60 GOTO 20
70 END
```

*** PTRIG**

La syntaxe est :

```
PTRIG
```

Cette fonction indique l'état de l'interrupteur du crayon optique. Sa valeur est −1 lorsque l'interrupteur est enfoncé et Ø dans le cas contraire.

Exemple :

```
1Ø CLS
2Ø LOCATE Ø, Ø
3Ø PRINT PTRIG
4Ø GOTO 2Ø
```

*** TUNE**

Cette instruction permet de régler le crayon optique. Elle affiche un trait vertical au centre de l'écran en blanc sur fond noir et attend que l'utilisateur y pointe le crayon optique.

Attention : Pour bien régler et utiliser le crayon optique, il peut être utile d'augmenter la luminosité du téléviseur.

2.6.4 Conclusion

Vous possédez maintenant les éléments nécessaires à l'utilisation du crayon optique. Il vous appartient d'exploiter au mieux vos connaissances pour créer des programmes qui n'auront que votre imagination pour limite.

Page blanche

CHAPITRE 3

Le graphisme du MO 5

3.1 INTRODUCTION

Les différentes instructions BASIC décrites jusqu'ici vous permettent de mener à bien des calculs plus ou moins complexes, de traiter des chaînes de caractères ou encore de créer des jeux de réflexion. Cependant, votre ordinateur vous communique ses résultats uniquement sous forme de suites de caractères alphanumériques, à l'exclusion de toute forme d'image ou dessin et encore bien moins sous forme d'animations. Utilisé ainsi, votre écran de télévision couleur ne présente pas beaucoup plus d'intérêt qu'un terminal classique, c'est-à-dire uniquement doté de l'affichage de signes, lettres et chiffres identiques à ceux du clavier.

Le jeu d'instructions décrit dans ce chapitre apporte une dimension nouvelle à vos programmes BASIC sur MO5. En effet, ces instructions vous conduiront dans le domaine passionnant de la micro-informatique vidéo.

Nous allons bien entendu passer en revue toutes les possibilités graphiques de votre MO5, mais auparavant, pour bien utiliser le graphisme, il est nécessaire de comprendre la manière dont une image est organisée et représentée.

3.2 REPRÉSENTATION DES IMAGES SUR MO 5 ET MODE GRAPHIQUE

L'image provenant du MO5 et apparaissant sur l'écran de télévision est formée d'un cadre (dans lequel il est impossible d'écrire mais dont la couleur est modifiable) et de la fenêtre de travail. Par abus de langage, nous appellerons par la suite « Image » cette fenêtre de travail puisqu'elle constitue la seule partie de l'écran nous intéressant réellement.

3.2.1 Généralités

L'image fournie par le MO5 peut être considérée comme formée de 2ØØ lignes, chaque ligne se divisant en 4Ø zones ou secteurs de 8 points consécutifs (voir figure ci-dessous), ceci étant dû au fait que la mémoire d'image est organisée en octets de points.

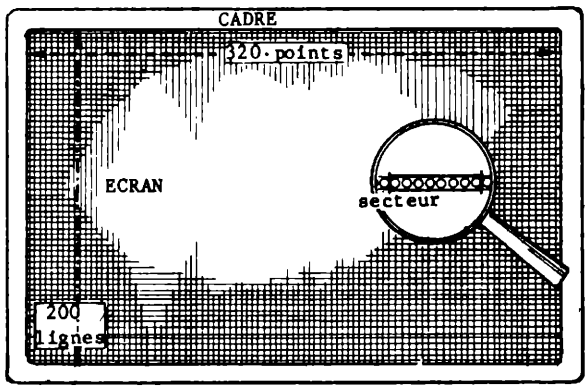


Figure 1 : L'écran de télévision

Nous voyons apparaître à ce stade une restriction essentielle aux machines appartenant à cette gamme de prix. En effet, il n'est pas possible de colorer chaque point d'une image de façon totalement indépendante du secteur de 8 points alignés dont il fait partie

Cependant cette petite restriction, grâce à laquelle le graphisme couleur sur ordinateur peut être mis à la disposition du grand public, ne gêne pas la très grande majorité des programmes que l'on peut être conduit à écrire.

Chaque secteur de 8 points ne peut donc posséder que 2 couleurs : une couleur de fond et une couleur de graphisme. Par contre, la paire de couleurs graphisme-fond d'un secteur quelconque est totalement indépendante des autres secteurs de l'écran.

Ainsi, une image est constituée de 2ØØ lignes comportant chacune 32Ø points. L'ensemble de ces points convenablement colorés forment l'image désirée. C'est pour cette raison que l'on parle à propos du MO5 d'une résolution graphique de 64 ØØØ points ou encore : 2ØØ × 32Ø.

3.2.2 Exemples de possibilités graphiques

Pour faire apparaître sur l'écran une image suivant le seul principe décrit ci-dessus, il faudrait donc attribuer aux 2ØØ × 4Ø = 8 ØØØ secteurs de 8 points une couleur de graphisme et une couleur de fond et, pour chacun des 8 points d'un secteur, préciser s'il doit posséder la couleur du graphisme ou celle du fond. La synthèse d'une telle image, bien que possible sur MO5, serait longue et fastidieuse à mettre en œuvre.

En fait, des instructions spécialement étudiées sont à votre disposition et permettent de réduire la plupart des programmes de dessin à quelques lignes. Par exemple, pour dessiner un rectangle mauve entouré de bleu foncé et écrire à l'intérieur « BONJOUR » en caractères de taille double et de couleur rouge, le tout sur fond noir, le programme suivant suffit :

```
1Ø SCREEN 5, Ø, Ø : CLS
2Ø BOXF (11, 7) - (3Ø, 16) CHR$ (127)
3Ø BOX (1Ø, 6) - (31, 17) CHR$ (127), 4
4Ø ATTRB 1, 1 : COLOR 1, 5 : LOCATE 14, 12
5Ø PRINT "BONJOUR" : ATTRB Ø, Ø : END
```

Après exécution du programme, pour revenir aux couleurs de départ, il faut successivement exécuter :

[RAZ] puis SCREEN 4, 6, 6 et [ENTREE]

3.2.3 Visualisation des secteurs et du cadre

Dans le but de faire visualiser aux lecteurs quelques-uns des 8 ØØØ secteurs que comporte un écran géré par MO5 nous proposons la manipulation suivante :

Tout d'abord, afin de mieux distinguer les segments de droite que forment les secteurs, passons en affichage vert sur fond noir en écrivant :

SCREEN 2, Ø, Ø [ENTREE]

puis appuyons sur la touche [RAZ]. Le curseur que l'on peut voir clignoter constitue lui-même un des 8 ØØØ secteurs de l'image ; mais faisons apparaître d'autres secteurs ; pour cela, entrons :

POKE 4Ø19, 255 [ENTREE]

Un segment de droite vert apparaît au beau milieu de l'écran ; il représente le 4 Ø2Ø ième secteur de l'écran (compté de gauche à droite et ligne de 8 petits points consécutifs. En entrant maintenant : par ligne). Vous pouvez constater qu'il est bien formé

POKE 4659, 255 [ENTREE]

un autre secteur apparaît 2 lignes au-dessus du précédent.

De façon générale, pour allumer le Nième segment, il faut entrer :

POKE N − 1, 255 [255]

avec : N = 4Ø *L + H ; L représentant le nombre de lignes et H l'ordre du secteur dans cette ligne. Cette façon de procéder sera expliquée au paragraphe consacré à la constitution physique de la mémoire d'écran.

Le programme suivant permet de visualiser sur votre écran le cadre de couleur rouge ainsi que les lignes dont les secteurs auront alternativement une couleur noire ou blanche.

```
1Ø SCREEN 2, Ø, 1 : C = Ø : CLS
2Ø FOR J = Ø TO 199
3Ø C = (C = Ø) * 8
4Ø FOR I = Ø TO 319 STEP 8
5Ø PSET (I, J), C : C = (C = Ø) * 8
6Ø NEXT I : NEXT J : END
```

Après avoir exécuté le programme il faudra, pour revenir aux couleurs d'origine, entrer successivement :

[RAZ]
SCREEN 4, 6, 6 [ENTREE]

3.2.4 La couleur

Maintenant que nous commençons à mieux concevoir la façon dont est structurée une image graphique, il est temps de compléter ces premières notions par la plus intéressante : la couleur.

Le système MO5 permet de générer sur l'écran 16 couleurs qui seront codées dans la plupart des instructions par 16 chiffres de Ø à 15. Le tableau suivant donne ces seize couleurs ainsi que le code correspondant à chacune d'entre elles.

couleur	code graphisme ou fond	code fond
NOIR	0	– 1
ROUGE	1	– 2
VERT	2	– 3
JAUNE	3	– 4
BLEU (bleu foncé)	4	– 5
MAGENTA (violet)	5	– 6
CYAN	6	– 7
BLANC	7	– 8
GRIS	8	– 9
ROUGE CLAIR	9	– 10
VERT CLAIR	10	– 11
JAUNE CLAIR	11	– 12
BLEU CLAIR	12	– 13
MAGENTA CLAIR	13	– 14
CYAN CLAIR	14	– 15
ORANGE	15	– 16

La signification de ces codes apparaîtra plus clairement par la suite.

Sans plus attendre, voici un programme permettant d'apprécier les différentes couleurs sur différents fonds. Un rectangle de petite dimension prend successivement les 16 couleurs possibles à l'intérieur d'un rectangle plus grand qui change de couleur seize fois moins rapidement que le petit de façon à ce que toutes les possibilités soient visualisées.

```
1Ø SCREEN 2, Ø Ø : CLS
2Ø FOR I = 1 TO 15
3Ø FOR J = Ø TO 15
4Ø BOXF (84, 56) - (236, 144), I
5Ø BOXF (14Ø, 88) - (18Ø, 112), J
6Ø FOR K = 1 TO 5ØØ : NEXT K
7Ø NEXT J : NEXT I : END
```

Après l'exécution de ce programme, l'écran écrit en caractères verts sur fond noir. Cette présentation est à la fois reposante pour les yeux et pour votre téléviseur ; en effet, seuls les caractères utiles étant allumés, le tube cathodique de l'écran du téléviseur aura une usure plus lente qu'avec la présentation bleu foncé sur fond bleu clair. Pour cette raison, nous conseillons au lecteur de travailler avec un écran ainsi coloré. Le passage dans ce mode de couleur sera effectué par la commande :

```
SCREEN 2, Ø, Ø [ENTREE]
```

3.2.5 Visualisation des couleurs dans un secteur

Nous allons visualiser ici tout ce qui a été dit au sujet des secteurs noirs, dans le domaine de la couleur.

Intéressons-nous au secteur 4 Ø2Ø. Dans un premier temps, nous allons le faire apparaître avec ses quatre premiers points en vert (couleur de graphisme) et les quatre derniers en noir (couleur de fond). Pour cela, il faut effectuer dans l'ordre :

```
SCREEN 2, Ø, Ø si vous ne l'avez déjà fait
[RAZ]
POKE 4Ø19, 24Ø [ENTREE]
```

Comme prévu, seuls les quatre premiers points du secteur sont

visibles. Entrons maintenant le programme suivant (après avoir fait un NEW) :

```
1Ø POKE &HA7 CØ, 1
2Ø POKE 4 Ø19, 24Ø
3Ø POKE &HA7C Ø, Ø
4Ø POKE 4 Ø19, &H14
5Ø END
```

Puis exécutons-le après avoir fait un RAZ. Le secteur 4 Ø2Ø apparaît alors, mais cette fois-ci les quatre premiers points sont de couleur rouge et les quatre derniers de couleur bleu. Jusque là, tout se passe comme prévu.

Essayons maintenant de colorer en vert le premier point du secteur (initialement de couleur rouge). Pour cela, effectuons la commande suivante :

PSET (152, 1ØØ), 2 [ENTREE]

Nous constatons alors que tous les points rouges qui appartenaient dans ce cas au graphisme sont devenus verts. Une constatation s'impose : il est impossible de colorer un secteur à l'aide de plus de deux couleurs ; ceci est normal en vertu de tout ce qui a été énoncé au début du présent chapitre. En changeant le chiffre 2 dans la commande précédente par un chiffre quelconque entre Ø et 15, l'ensemble des quatre premiers points revêtira la couleur correspondante.

Si maintenant, nous essayons de changer le fond en entrant :
soit PSET (158, 1ØØ), – 4 [ENTREE]
soit PSET (158, 1ØØ), – 6 [ENTREE]
soit PSET (158, 1ØØ), – 7 [ENTREE]
nous vérifions qu'il n'est possible de changer que deux couleurs. Cependant, il est important à ce stade d'avoir bien compris que dans un secteur donné, les points ne peuvent prendre que deux couleurs, mais que n'importe quel point peut prendre la couleur du fond ou celle du graphisme.

Pour finir cette illustration, exécutons à nouveau le programme précédent (RUN) après avoir fait un RAZ ; puis entrons :

POKE 4 Ø2Ø, 255 [ENTREE]

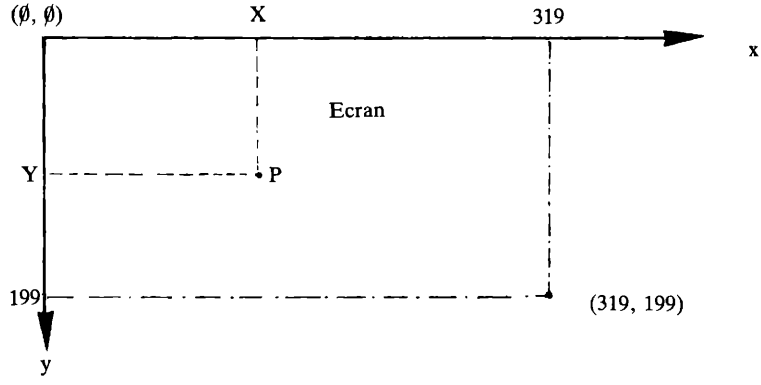
Le secteur 4 Ø21 apparaît à la suite du 4 Ø2Ø, nous voyons que ces deux secteurs consécutifs possèdent des couleurs totalement indépendantes. Ceci confirme que deux secteurs peuvent être colorés de façon totalement indépendante.

3.2.6 Mode graphique et coordonnées d'un point sur l'écran

Lorsque l'on manipule des points sur l'écran, qu'on leur attribue des couleurs, qu'on les regroupe en formes quelconques, on dira que l'on est en mode graphique et que l'on utilise des instructions graphiques. Ceci par opposition au mode caractère que l'on utilise pour inscrire sur l'écran des paquets de points qui représentent des signes connus et utilisés de façon fréquente.

Les instructions graphiques qui permettent de positionner et de colorer les points ou les groupes de points nécessitent bien entendu une désignation précise de la position des points concernés. A cette fin, un système de coordonnées très simple a été standardisé pour tous les BASIC Microsoft.

Un point P quelconque de l'écran est repéré par ses coordonnées (X, Y). Y représentant le nombre de lignes à compter en partant du haut de l'écran (ligne Ø) pour atteindre le point P (Ø ⩽ X ⩽ 199) et X le nombre de points qu'il faut compter sur la ligne Y en partant de la gauche de l'écran (point Ø) pour atteindre P (Ø ⩽ X ⩽ 319). Ainsi, le point situé le plus en haut à gauche de l'écran a pour coordonnées (Ø, Ø) et le plus en bas à droite (319, 119). Remarquons que de Ø à 199, nous avons bien 2ØØ lignes et de Ø à 319 nous comptons bien 32Ø points.



Ces coordonnés serviront à spécifier le point sur lequel on voudra faire agir l'instruction graphique. Par exemple pour positionner un point rouge au centre de l'écran (16Ø, 1ØØ) il suffira d'écrire :

PSET (16Ø, 1ØØ), 1 [ENTREE]

Les coordonnées du point considéré apparaissent clairement dans cette instruction. D'autres instructions nécessitent deux couples de coordonnées. Par exemple, pour tracer une ligne rouge entre le point A (1ØØ, 15Ø) et le point B (3ØØ, 19Ø) il faudra utiliser l'instruction

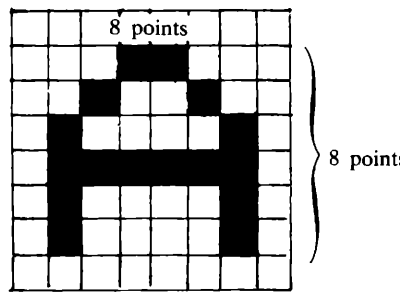
LINE (1ØØ, 15Ø) – (3ØØ, 19Ø), 1 [ENTREE]

Là encore, les coordonnées des points A et B sont présentes dans l'instruction.

3.3 REPRÉSENTATION DES CARACTÈRES SUR MO5 ET MODE CARACTÈRE

3.3.1 Définition d'un caractère

Un caractère est un bloc carré de 64 points (8 × 8). La forme du caractère est définie par l'appartenance de chacun des 64 points au fond ou au graphisme. Par exemple, le caractère « A » est formé par les 16 points ombrés appartenant au graphisme et les 48 points non colorés appartenant au fond dans le dessin ci-dessous :



Bien entendu, il n'est pas utile à chaque fois que l'on veut faire apparaître un caractère sur l'écran de sélectionner les 64 points qui devront appartenir au graphisme ou au fond. Ceci est dû au fait que chaque caractère a été mémorisé par construction dans votre MO5.

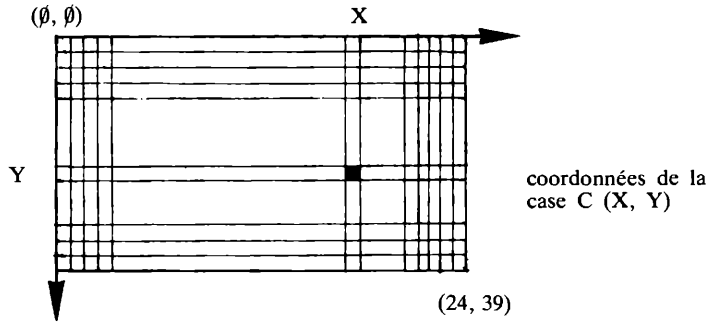
Ainsi, le programme qui gère votre ordinateur prend en charge cette opération ; chaque fois que l'écriture d'un caractère sur l'écran lui est demandée, il affiche les 64 points conformément à la répartition préprogrammée en mémoire.

Afin de faire un lien avec tout ce qui a été vu à propos de la représentation des images sur MO5, il faut retenir qu'un caractère coïncide toujours sur l'écran avec 8 secteurs empilés. Le programme de gestion de l'ordinateur n'affiche donc pas véritablement les 64 points les uns après les autres mais affiche les 8 secteurs correspondants, chaque secteur ayant reçu sa répartition de points fond et graphisme. D'autre part, en raison de cette disposition en secteurs de l'image, les caractères ne peuvent occuper que des cases de 64 points bien définies sur l'écran. C'est ainsi que nous sommes menés à la notion de coordonnées en mode caractère.

3.3.2 Coordonnées en mode caractère

En ce qui concerne le mode caractère, l'écran peut être considéré comme constitué de 25 lignes, chaque ligne étant divisée en 4Ø cases dans lesquelles on pourra inscrire les caractères. Ces cases comportent bien évidemment 64 points en matrice 8 × 8 et l'intérêt du mode caractères est, entre autres choses, d'éviter d'avoir à calculer la position de ces 64 points sur l'écran lorsque l'on veut inscrire un caractère dans une case quelconque.

Tout comme pour le mode graphique, on attribue à chaque case C un couple de coordonnées (X, Y). X désignera le nombre de cases en partant de la gauche de l'écran (case Ø) que l'on devra compter pour atteindre la case C, et Y le numéro de la ligne sur laquelle la case C se situe (la ligne Ø étant en haut de l'écran) (Ø ≤ X ≤ 39) et Ø ≤ Y ≤ 24).



Les cases extrêmes ont pour coordonnées (Ø, Ø) et (24, 39). On pourrait aussi parler de lignes pour Y et de colonnes pour X.

Les coordonnées serviront à spécifier la case sur laquelle on voudra faire agir l'instruction de caractère. Par exemple pour positionner un caractère "A" au milieu de l'écran (colonne 19 et ligne 12) il faudra écrire :

PSET (19, 12) "A" [ENTREE]

Les coordonnées (X, Y) apparaissent clairement dans cette instruction que nous verrons plus en détail par la suite.

Le programme suivant permet de visualiser l'ensemble des 4Ø × 25 = 1 ØØØ cases de 64 points du mode caractère. Le cadre apparaît en rouge et les cases alternativement en vert et en noir. La case de coordonnées (19, 12) s'inscrit en rouge sur l'écran.

```
1Ø SCREEN 2, Ø, 1 : C = 2 : CLS
2Ø FOR J = Ø TO 24
3Ø C = - (C =Ø) * 2
4Ø FOR I = Ø TO 39
5Ø PSET (I, J) CHR$ (127), C
6Ø C = - (C = Ø) * 2
7Ø NEXT I : NEXT J
8Ø PSET (19, 12) CHR$ (127), 1
9Ø COLOR 2 : END
```

Après avoir vérifié que la case rouge a bien pour coordonnées (19, 12), vous reviendrez en mode normal en exécutant les opérations suivantes :

[RAZ]
SCREEN 4, 6, 6 [ENTREE]

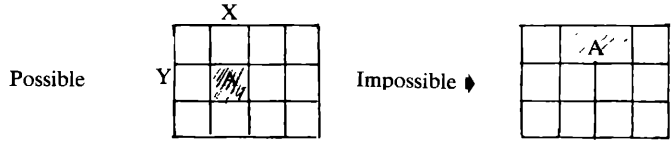
3.3.3 Les caractères ASCII

Physiquement, les informations véhiculées et manipulées par un ordinateur sont une suite de Ø et de 1. Un ensemble de Ø et de 1 ordonnés peut être assimilé à un nombre. C'est la raison pour laquelle, pour manipuler une information quelconque (lettres, signes, mots, dessins, etc.), il est nécessaire d'établir une correspondance rigoureuse entre l'information traitée et un nombre qui la représentera dans la machine. Cette correspondance s'apelle un codage. Ainsi, il existe une multitude de codes qui

peuvent être utilisés pour représenter par exemple l'alphabet ; le plus simple étant A = 1, B = 2, ..., Z = 26.

Dans un ordinateur, il y a beaucoup plus de caractères et autres informations que les seules lettres de l'alphabet, ne serait-ce que tous les caractères que comporte votre clavier. Afin d'ordonner l'informatique dans ce domaine, un Code Standard a été mis au point et adopté par la majorité des concepteurs ; il s'agit du code ASCII (American Standard Code for Information Interchange).

Le code ASCII est utilisé par certaines instructions, il faut savoir l'employer correctement pour profiter pleinement des capacités graphiques de votre MO5. Il est important de savoir que le code ASCII comprend en plus des signes qui apparaissent sur l'écran des caractères de commande tels que : le déplacement du curseur, l'effacement ou le beep ; ces caractères pourront être utilisés dans des programmes d'animation. Bien entendu, tous les caractères du code ASCII qui ne sont pas des caractères de commande sont transformés en matrice 8 × 8 (blocs de 64 points) lors de leur affichage et placés dans la case de coordonnées spécifiées ; comme tout caractère, un caractère ASCII ne peut chevaucher deux ou plusieurs cases de 64 points.



3.3.4 Les caractères programmables

Malgré tous les caractères que possède le MO5 dans sa mémoire ROM (non effaçable), l'utilisateur est souvent amené à utiliser ses propres caractères, surtout dans des programmes de jeu. Une instruction spéciale : DEFGR$ lui permet de les définir et donc, de créer son propre jeu de caractères associé à son propre code.

L'utilisation de l'instruction DEFGR$ sera expliquée en détail au paragraphe 4.5. Cependant, nous pouvons déjà expliquer la façon dont un caractère est défini pour cette instruction.

Comme il a déjà été dit au paragraphe 3. 1, un caractère est formé de 8 secteurs placés les uns au-dessus des autres, le tout formant un bloc carré de 64 points (8 × 8). Chaque secteur étant formé de 8 points, il est possible de le définir à l'aide d'un octet (nombre binaire composé de 8 chiffres binaires (1 ou Ø). Chaque chiffre binaire prenant le nom de bit, un octet est composé de 8 bits. Ces bits sont numérotés de 7 à Ø ; le plus à gauche étant le bit 7 et le plus à droite le bit Ø. La correspondance entre les bits d'un octet et un secteur est très simple : par exemple, si le point 6 du secteur appartient au fond, alors le bit 6 de l'octet vaudra Ø dans le cas contraire, bien entendu, il vaudra 1 (le point 6 étant le deuxième point du secteur en partant de la gauche).

Prenons pour illustrer ceci le secteur suivant où les points sombres représentent les points graphisme.

● ○ ● ● ○ ○ ○ ●

L'octet définissant ce secteur aura pour valeur :

1 Ø 1 1 Ø Ø Ø 1

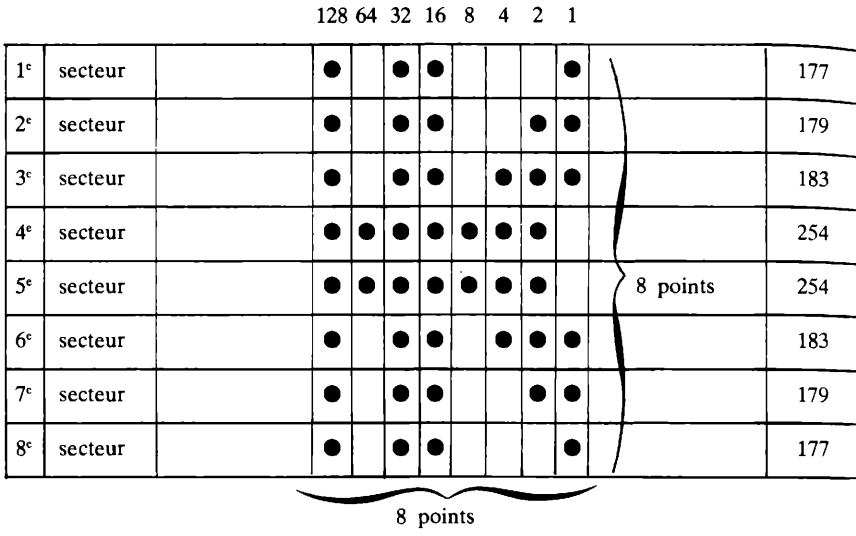
qui est une valeur binaire. Cette valeur peut être convertie en valeur décimale :

$1 \times 2^7 + Ø \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + Ø \times 2^3 + Ø \times 2^2 + Ø \times 2^1 + 1 \times 2^0 = 177$

Ainsi, 177 sera la représentation du secteur décrit.

Un caractère étant formé de 8 secteurs, il sera décrit par une suite de 8 nombres déterminés par la même méthode que précédemment. Cette suite commencera par le nombre représentant le secteur le plus haut du caractère et finira par le secteur le plus bas, les nombres étant séparés par une virgule.

Par exemple, pour décrire le caractère suivant :



on utilise la suite de nombres 177, 179, 183, 254, 254, 183, 179, 177. Un moyen pratique pour calculer la valeur d'un secteur est d'ajouter tous les nombres situés en haut du carré dont le point dans la colonne correspondante appartient au graphisme (car $2^7 = 128$, $2^6 = 64$ etc.). Ainsi le nombre 177 a été déterminé par l'opération 128 + 32 + 16 + 1 + 177.

Ce caractère pourra être visualisé sur l'écran en exécutant le programme :

```
1Ø CLEAR, 1
2Ø DEFGR$ (Ø) = 177, 179, 183, 254, 254, 183, 179, 177
3Ø PRINT GR$ (Ø) : END
```

La suite de nombres définissant le caractère graphique désiré apparaît clairement à la ligne 2Ø du programme.

3.4 CONSTITUTION PHYSIQUE DE LA MÉMOIRE D'ÉCRAN

Ce paragraphe peut être ignoré des débutants en micro-informatique, la compréhension de ce qui suit n'en sera pas affectée. Cependant, il permettra à d'éventuels programmeurs en

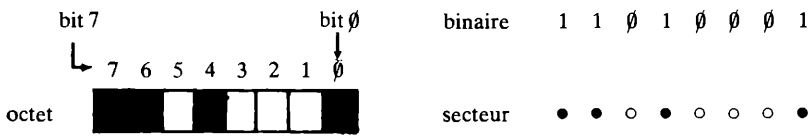
langage d'assemblage d'utiliser leurs connaissances pour accélérer certains jeux et certaines animations. Toutefois, pour ne pas décourager le lecteur désireux d'en savoir plus mais débutant, les explications seront volontairement nombreuses.

La mémoire d'écran se situe de l'adresse &HØØØ à &H1F3F, soit en décimal de la case mémoire Ø à la case mémoire 7 999. Chaque case mémoire contient un octet (nombre binaire formé de huit chiffres) correspondant à un secteur de l'écran.

Il y a une correspondance directe entre les bits (chiffres binaires) de chaque octet et le graphisme du secteur correspondant. Ainsi, chaque bit de l'octet représente un point du secteur ; le bit 7 étant associé au premier point en partant de la gauche du secteur et le bit Ø au dernier point. Si le bit est à 1, le point appartient au graphisme et si le bit est à Ø, le bit appartient au fond.

Exemple :

Si le secteur 39 (première ligne, dernier secteur) est allumé de la façon suivante :



(les points sombres représentent la couleur forme ou graphisme et les points clairs le fond) alors, l'octet mémoire 39 = aura pour valeur binaire $(1\ 1\ Ø\ 1\ Ø\ Ø\ Ø\ 1)_2$ = &HD1 = 209 (à l'adresse &H27).

Il est possible de vérifier ceci en faisant :

POKE 39, 2Ø9 [ENTREE]

ou encore

POKE &H27, &HD1 en notation hexadécimale

Le secteur apparaît sur la droite de l'écran à l'endroit prévu et avec la forme prévue.

Si l'on numérote les secteurs de gauche à droite et de haut en bas,

le premier étant le Øième et le dernier le 7 999ième, l'octet mémoire correspondant au Nième secteur se trouvera à l'adresse N. Nous rappelons qu'il y a 4Ø secteurs par ligne et 2ØØ lignes.

En fait, la mémoire d'image est constituée de deux plans mémoire. Le premier plan mémoire est celui dont il vient d'être question et représente le plan graphisme puisqu'il détermine les points fond et les points graphisme. Son adressage se situe entre Ø et 3 999. Le deuxième plan mémoire est le plan couleur, il comporte également 8 ØØØ octets, chaque octet étant attribué au secteur correspondant sur l'écran exactement de la même manière que pour le premier plan mémoire. De plus, ces deux plans se situent exactement à la même adresse, de sorte que l'octet graphisme et l'octet couleur du Nième segment seront tous deux à l'adresse N. Une question vient alors à l'esprit : Comment deux octets différents peuvent-ils se trouver à la même adresse ? La réponse est simple, il existe en fait un buffer (le point AØ d'un PIA) dont le contenu 1 ou Ø sélectionne respectivement le premier plan mémoire ou le deuxième. Ce PIA se trouve à l'adresse &HA7CØ et est habituellement positionné à 1. Lorsque l'on écrit en mode machine de bureau :

POKE 39, 2Ø9 [ENTREE]

Le buffer est déjà positionné à 1 ce qui explique que l'on ait toujours accès dans ce cas au premier plan mémoire. Si l'on veut positionner le buffer à Ø, il faudra le faire par programme.

Le codage de la couleur graphisme et fond pour un secteur donné se fait par l'intermédiaire de l'octet correspondant dans le deuxième plan mémoire de la façon suivante :

Les bits 7 à 4 contiennent en binaire le code de la couleur graphisme et les bits 3 à Ø le code de la couleur de fond. Le code est celui indiqué au paragraphe 2.4 du présent chapitre.

Exemple :

Nous voulons faire apparaître le secteur 39 en rouge sur fond jaune ; le code du rouge est 1 soit ØØØ1 en binaire ; le code du jaune est 3 soit ØØ11 en binaire. L'octet correspondant (adresse 39) contiendra :

ØØØ1 ØØ11
forme fond

ØØØ1 représente le code graphisme et ØØ11 le code fond. L'octet est $(ØØØ1\ ØØ11)_2$=&H13 = 19.

Si nous prenons la même disposition graphisme-fond que dans l'exemple précédent, le programme vérifiant tout ce qui a été dit s'écrit :

```
1Ø POKE &HA7CØ, 1
2Ø POKE 39, 2Ø9
3Ø POKE &HA7CØ, Ø
4Ø POKE 39, 19
```

— la ligne 1Ø sélectionne le 1er plan mémoire ;
— la ligne 2Ø met dans l'octet graphisme du secteur 30 la bonne configuration ;
— la ligne 3Ø sélectionne le 2e plan mémoire ;
— la ligne 4Ø met dans l'octet couleur du secteur 39 les couleurs voulues.

Après un RUN, on vérifie que le secteur s'est allumé comme prévu.

A ce stade de nos observations, deux vérités énoncées a priori dans les paragraphes précédents s'expliquent :

— La première concerne le fait qu'il est impossible de colorer indépendamment les 8 points d'un même secteur puisque leurs couleurs sont définies par le même octet et que ce dernier ne peut contenir que l'information correspondant à deux couleurs.

— La deuxième explique le codage des couleurs. En effet, nous savons tous qu'une image est constituée de trois couleurs de base : le rouge, le vert et le bleu. Or, nous disposons de $2^4 = 16$ couleurs sur le MO5. Lors de la formation du signal vidéo, il faut reconstituer ces couleurs à l'aide des bits contenus dans la mémoire d'écran. Pour que le décodage du nombre binaire CXYZ soit simple, il suffit de décider que X représente le bleu (1 allumé, 0 éteint, que de même Y représente le vert, Z le rouge et C le fait que la couleur est claire (C : couleur de fond C = Ø). Ainsi sachant que les seize teintes sont obtenues de la façon suivante :

BLANC = ROUGE + VERT + BLEU + FONCE
JAUNE = ROUGE + VERT + FONCE
CYAN= VERT + BLEU + FONCE

MAGENTA = ROUGE + BLEU + FONCE
NOIR = aucune teinte + FONCE
BLEU = BLEU + FONCE
ROUGE = ROUGE + FONCE
VERT = VERT + FONCE
GRIS = NOIR CLAIR
ROUGE CLAIR = ROUGE + CLAIR
VERT CLAIR = VERT + CLAIR
JAUNE CLAIR = ROUGE + VERT + CLAIR
BLEU CLAIR = BLEU + CLAIR
CYAN CLAIR = VERT + BLEU + CLAIR
MAGENTA = ROUGE + BLEU + CLAIR

Le cas du ORANGE est spécial, il se substitue au BLANC CLAIR qui n'est d'aucune utilité (BLANC CLAIR = BLANC !).

On a le codage :

Code	Couleur	lit du «clair»	lit du bleu	lit du vert	lit du rouge
ø	NOIR	ø	ø	ø	ø
1	ROUGE	ø	ø	ø	1
2	VERT	ø	ø	1	ø
3	JAUNE	ø	ø	1	1
4	BLEU	ø	1	ø	ø
5	MAGENTA	ø	1	ø	1
6	CYAN	ø	1	1	ø
7	BLANC	ø	1	1	1
8	GRIS	1	ø	ø	ø
9	ROUGE CLAIR	1	ø	ø	1
10	VERT CLAIR	1	ø	1	ø
11	JAUNE CLAIR	1	ø	1	1
12	BLEU CLAIR	1	1	ø	ø
13	MAGENTA CLAIR	1	1	ø	1
14	CYAN CLAIR	1	1	1	ø
15	ORANGE	1	1	1	1

Sur le tableau ci-dessus, on constate que le monbre binaire constitué par le bit du bleu, le bit du vert et le bit du rouge (CBVR)

correspond au nombre décimal du code couleur. C'est pour cette raison que l'octet de couleur (deuxième plan mémoire) est codé en deux paquets de 4 bits.

En résumé :

Pour allumer le secteur N à notre convenance, il faut :

1) Sélectionner le premier plan mémoire image en faisant

POKE &HA7CØ, 1
(ou l'équivalent assembleur)

2) Décider des points fond et graphisme en attribuant la valeur 1 aux bits correspondant aux points graphisme du secteur N et Ø aux points fond, coder le tout sur un octet que l'on placera à l'adresse N :

POKE N, &HXY
(ou l'équivalent assembleur)

3) Sélectionner le deuxième plan mémoire image en faisant :

POKE &HA7CØ, Ø
(ou l'équivalent assembleur)

4) Décider des couleurs fond et graphisme en mettant dans les bits 7 à 4 le code couleur de graphisme et dans les bits 3 à Ø le code couleur du fond, coder le tout sur un octet que l'on placera à l'adresse N :

POKE N, &HVW
(ou l'équivalent assembleur

3.5 LES INSTRUCTIONS GRAPHIQUES DU BASIC MO5

3.5.1 Les instructions de commande de l'écran

Nous appelons instructions de commande de l'écran les instructions permettant de modifier une partie ou la totalité de l'image.

Dans la suite, les crochets [ ] entourant une expression ou un paramètre indiquent que l'expression ou le paramètre peut être omis(e).

*** CONSOLE**

Cette instruction fait appel à la notion de fenêtre de travail. La fenêtre de travail se délimite par une ligne supérieure et une ligne inférieure. Elle peut posséder une dimension verticale quelconque (entre 1 et 25 lignes de caractères) mais la largeur est toujours celle de l'écran, et en temps normal la fenêtre correspond à l'ensemble de l'image (lignes Ø à 24).

Cette fenêtre de travail est définie de façon à être utilisée par d'autres instructions telles que SCREEN et CLS ; ainsi, lors d'un effacement d'écran (RAZ ou CLS), seul le contenu de la fenêtre sera effacé.

La suite des manipulations décrites ci-dessous vous permettra de visualiser et de mieux comprendre l'intérêt de cette fenêtre.

En définissant une fenêtre de travail entre les lignes 5 et 15 à l'aide de

CONSOLE 5, 15 [ENTREE]

il est possible de changer uniquement la couleur de l'écran située entre les lignes 5 et 15. Pour cela le fait de rentrer

SCREEN 2, Ø [ENTREE]

va colorer le fond de la fenêtre en noir et les caractères en vert. Si maintenant nous inscrivons dans cette fenêtre un message quelconque, le fait d'appuyer sur la touche RAZ effacera seulement le contenu de la fenêtre et le reste de l'écran demeurera inchangé.

Pour revenir à la configuration de départ, il faut tout d'abord restituer à la fenêtre de travail une hauteur de 25 lignes en entrant :

CONSOLE Ø, 24 [ENTREE]

puis, pour le vérifier, appuyer sur la touche RAZ. On constate que la fenêtre occupe alors tout l'écran.

Un

SCREEN 4, 6 [ENTREE]

permet de revenir à la configuration initiale des couleurs.

Maintenant que la notion de la fenêtre de travail est comprise, nous pouvons étudier l'instruction CONSOLE.

La syntaxe générale est :

CONSOLE [A] [, [B] [, [I] [, J]]]

A représente le numéro de la ligne supérieure de la fenêtre et N le numéro de la ligne inférieure. Bien entendu, A doit toujours être inférieur ou égal à B.

I est rarement utilisé : si I vaut Ø les caractères s'inscrivant avec la couleur qui leur a été spécifié et si I vaut 1, la couleur ne changera pas bien que celle-ci reste enregistrée dans la mémoire d'écran et réapparaisse si l'on fait à nouveau I = Ø.

J peut posséder 3 valeurs : Ø, 1, ou 2.

Si J = Ø, le défilement à l'intérieur de la fenêtre se fait à vitesse normale.

Si J = 1, le défilement se fait beaucoup plus lentement.

Si J = 2, lorsque la dernière ligne de la fenêtre est atteinte, le défilement reprend à partir de la première ligne.

Le programme suivant permet d'apprécier les différentes possibilités de défilement associées à la valeur de J.

```
  Ø CONSOLE 5, 15, , Ø:GOTO 1ØØ
  1 CONSOLE 5, 15, , 1:GOTO 1ØØ
  2 CONSOLE 5, 15, , 2:GOTO 1ØØ
1ØØ LOCATE Ø, 5 : CLS
11Ø FOR I = 1 TO 5Ø
12Ø FOR J = 1 TO 2Ø: NEXT J
13Ø PRINT "I VAUT :"; I
14Ø NEXT I : END
```

Après avoir rentré correctement le programme, faites tout d'abord un

RUN Ø [ENTREE]

Le texte défile du bas vers le haut relativement vite et dans la fenêtre de travail uniquement. Ceci correspond à J = Ø (ligne Ø).

Maintenant faites un

RUN 1 [ENTREE]

(après avoir fait un RAZ). Un message identique au précédent défile mais avec une vitesse beaucoup moins rapide. Ceci correspond à J = 1.

Enfin, en effectuant un

RUN 2 [ENTREE]

vous pouvez observer un défilement du haut vers le bas et lorsque la dernière ligne de la fenêtre est atteinte, le défilement reprend à partir du haut. Pour obtenir à nouveau une fenêtre normale exécuter la commande

CONSOLE Ø, 24 [ENTREE]

*** SCREEN**

Cette instruction permet de changer les couleurs de la fenêtre de travail ainsi que du cadre. La syntaxe générale est :

SCREEN [A] [, [B[[, [C] [, [D]]]]

A indique le code de la couleur du graphisme (ou forme).

B indique le code de la couleur du fond.

C indique le code de la couleur du cadre.

(Le code des couleurs se trouve au paragraphe 3.2.4).

D indique lorsqu'il est présent que les couleurs fond et graphique doivent être interverties. D ne peut prendre que les valeurs Ø ou 1. En fait, on a l'équivalence des deux expressions

SCREEN A, B, C, D et SCREEN B, A, C

Les valeurs de A, B, et C sont comprises entre Ø et 15.

Exemples d'utilisation

La commande

SCREEN, , 1

permettra de modifier uniquement la couleur du cadre qui deviendra rouge.

La commande

SCREEN Ø, , 3

changera la couleur des caractères qui deviendront noirs ainsi que celle du cadre qui deviendra jaune.

La commande

SCREEN 2, Ø, Ø

provoquera un changement de toutes les couleurs :

- les caractères deviendront verts ;
- le fond deviendra noir ;
- le cadre deviendra noir.

Définissons-nous maintenant une fenêtre de travail en exécutant

CONSOLE 7, 2Ø [ENTREE]

La commande

SCREEN Ø, 3, 5 [ENTREE]

colore le cadre en mauve et à l'intérieur de la fenêtre de travail le fond sera jaune avec des caractères noirs. On constate que le reste de l'écran demeure inchangé : l'instruction SCREEN n'agit bien que sur la fenêtre de travail.

Grâce à la commande SCREEN l'utilisateur peut choisir la disposition des couleurs qu'il trouve la plus agréable et la plus reposante pour ses yeux. Ceci est important lorsqu'on passe de nombreuses heures devant un écran de téléviseur ; à ce sujet, nous recommandons les caractères verts sur fond noir obtenus par la commande

SCREEN 2, Ø, Ø [ENTREE]

Remarque : lorsque le MO5 est muni de l'extension qui permet de faire de l'incrustation sur l'image du téléviseur, l'instruction SCREEN admet un 5e paramètre qui par sa présence rend les points noirs transparents pour l'image de télévision.

*** CLS**

Cette instruction permet d'effacer le contenu de la fenêtre de travail et de repositionner le curseur en haut à gauche dans cette même fenêtre. Aucun changement de couleur n'est effectué.

La syntaxe est CLS.

*** PSET**

Cette instruction permet de positionner un point à un endroit quelconque de l'écran en mode graphique et une case quelconque en mode caractère.

— Mode graphique

La syntaxe est :

PSET (X,Y) [, C]

X est la coordonnée horizontale du point à placer ($\emptyset \leqslant X \leqslant 31\emptyset$).

Y est la coordonnée verticale ($\emptyset \leqslant Y \leqslant 199$) c'est-à-dire la ligne.

C doit avoir la valeur de la couleur désirée (voir paragraphe 3.2.4).

Si le point doit appartenir au graphisme alors : $\emptyset \leqslant C \leqslant 15$.

Si le point doit appartenir au fond alors : $-16 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs de graphisme ou de fond différentes.

Exemples :

PSET (2ØØ, 1ØØ), 1

met un point rouge graphisme en (2ØØ, 1ØØ).

PSET (2ØØ, 1ØØ), −5

met un point bleu en fond en (2ØØ, 1ØØ).

— Mode caractère

La syntaxe est :

PSET (X, Y) "Caractères" [, [A] [, [B] [, C]]]

X représente la colonne de la case à remplir (en mode caractères) donc Ø ⩽ X ⩽ 39.

Y représente la ligne de la case à remplir (Ø ⩽ Y ⩽ 24).

"Caractère" représente le caractère que l'on désire insérer dans la case de coordonnées (X, Y).

"Caractère" doit obligatoirement être une chaîne de caractères ou une expression chaîne. Seul le premier caractère de l'expression est pris en compte.

"Caractère" peut donc revêtir les formes suivantes :

- "E" ou "EXEMPLE" où E est un caractère du code ASCII.
- A$ qui représente une expression chaîne.
- CHR$(N) où N représente le code ASCII du caractère.
- GR$(N) où N représente le numéro du caractère programmé par l'utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond de la case (X, Y).

C indique lorsqu'il est présent que les couleurs A et B doivent être inversées donc :

PSET (X, Y) "C", A, B, C équivaut à PSET (X, Y) "C", B, A

C ne peut prendre que les valeurs Ø ou 1.

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. Il est d'ailleurs important de signaler qu'après l'exécution d'un programme, les caractères qu'affichera sur l'écran l'utilisateur auront la dernière couleur définie dans le programme si une instruction du type PSET, SCREEN, COLOR etc... a été utilisée. En conséquence, il est important avant de finir un programme de réinitialiser les couleurs. Cependant, à la suite d'une mauvaise manœuvre, il est possible de se trouver par exemple dans une configuration caractères noirs et fond noir. Il est alors impossible de vérifier que les caractères frappés au clavier l'ont été

correctement. Pour échapper à cette situation, il convient d'appuyer sur la touche [ENTREE] de façon à être certain de ramener le curseur (invariable) en début de ligne (ou sur la touche RAZ pour le ramener en début d'écran) puis d'écrire au clavier (sans pouvoir le vérifier)

COLOR 4, 6 [ENTREE]

Exemples :

PSET (3Ø, 1Ø) "+", 2, 1

va provoquer l'apparition d'un signe + de couleur verte sur fond rouge dans la case (3Ø, 1Ø). Si l'instruction PSET (3Ø, 2Ø) "=" suit immédiatement, le signe = apparaîtra en (3Ø, 2Ø) avec les mêmes couleurs que le signe +. Par contre, PSET (3Ø, 2Ø) "=", , 6 aurait fait apparaître un signe = vert mais un fond cyan.

Lors de l'exécution du programme suivant :

```
1Ø A$ = ">"
2Ø SCREEN 6, Ø, Ø: CLS
3Ø PSET (5, 1Ø) A$
4Ø PSET (1Ø, 1Ø) A$, 2, 1 : END
```

Le signe > sera affiché en (5, 1Ø) avec une couleur cyan (ligne 3Ø) puis en (1Ø, 1Ø) avec une couleur verte sur fond rouge. Après exécution du programme, les caractères que pourra écrire l'opérateur seront également verts sur fond rouge car aucune instruction modifiant ces dispositions n'existe avant la fin du programme (END).

L'instruction PSET (3Ø, 1Ø) CHR$(43), 2, 1 a exactement le même effet que PSET (3Ø, 1Ø) "+", 2, 1 puisque 43 est le code ASCII du signe +.

*** DEFGR$**

Cette instruction permet de définir les caractères de son choix et de les utiliser comme les caractères habituels du MO5.

La syntaxe est :

DEFGR$(N) = (liste d'entiers)

N représente le numéro de codage que vous attribuez au nouveau caractère ; N doit être compris entre ∅ et 127.

(liste d'entiers) est une liste de 8 nombres entiers séparés par une virgule. Cette liste décrit le caractère (comme nous l'avons déjà expliqué au paragraphe 3.3.4) et est composée par conséquent de nombres compris entre ∅ et 255. Comme tous les caractères du MO5, la dimension des caractères utilisateurs est 8 × 8 soit 64 points.

La méthode que nous conseillons pour déterminer la liste d'entiers est la suivante :

— Dessiner le caractère désiré dans un carré composé de 64 petits carrés en ombrant les petits carrés (représentant les points) lorsqu'ils doivent appartenir à la couleur du graphisme (figure ci-dessous).

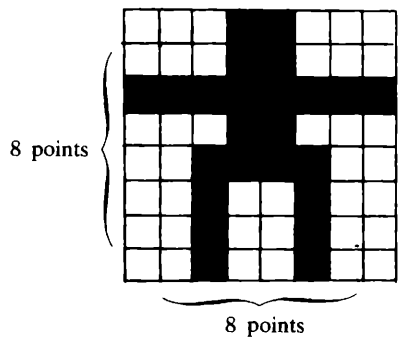


Figure 1

— Inscrire au-dessus de chacune des 8 colonnes du grand carré et de gauche à droite les nombres suivants dans l'ordre : (figure 2)

128, 64, 32, 16, 8, 4, 2, 1

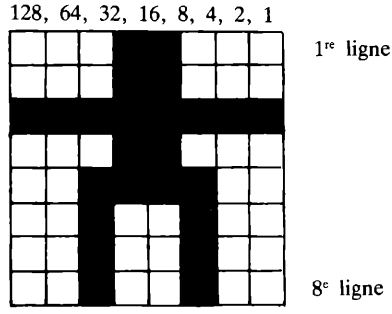


Figure 2

— Les 8 nombres se déterminent alors ligne par ligne et de haut en bas. Pour une ligne donnée ajouter entre eux tous les nombres se trouvant au-dessus des colonnes et dont les points sur cette ligne sont ombrés.

Par exemple, pour la première ligne, les points correspondant aux nombres 16 et 8 sont ombrés, donc le premier nombre de la liste sera 16 + 8 = 24 (figure 3).

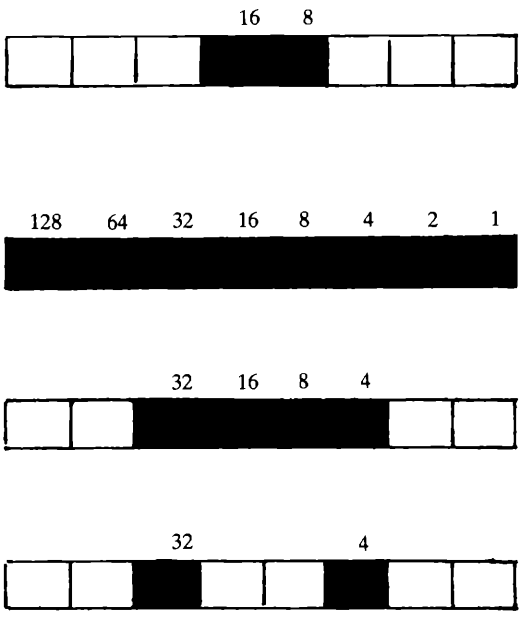


Figure 3

La liste d'entiers de l'exemple de la figure 1 est donc :

24, 24, 255, 24, 60, 36, 36, 36

L'instruction

DEFGR$(Ø) = 24, 24, 255, 24, 6Ø, 36, 36, 36

définira sous le numéro Ø le caractère graphique de la figure 1.

Utilisation de DEFGR$

Un caractère pour pouvoir être défini par DEFGR$ doit au préalable se voir réserver une place mémoire à l'aide de l'instruction CLEAR. Nous rappelons que la syntaxe de l'instruction CLEAR était :

CLEAR [A] [, [B], C]]

C indique le nombre de caractères graphiques utilisateur à réserver pour la suite du programme.

Exemple :

Il vous faut définir 7 caractères utilisateur ; vous devez alors (de préférence en tout début de programme) utiliser l'instruction CLEAR ,, 7 avant de définir vos 7 caractères.

Les caractères une fois définis pourront être désignés soit par GR$(N) où N représente le code du caractère lors de sa déclaration par DEFGR$(N) : (liste d'entiers), soit par CHR$ (128 + N).

Il est possible de créer jusqu'à 128 caractères utilisateur.

Exemples d'utilisation

Le programme suivant :

```
1Ø CLEAR ,, 1
2Ø DEFGR$(Ø) = 24, 24, 255, 24, 6Ø, 36, 36, 36
3Ø SCREEN 2, Ø, Ø : CLS
4Ø PSET (1Ø, 5) GR$(Ø)
5Ø PSET (11, 5) CHR$ (128), 1
6Ø PRINT GR$(Ø)
7Ø COLOR 2 : END
```

réserve à la ligne 1Ø la place mémoire nécessaire à un caractère utilisateur, définit ce caractère à la ligne 2Ø (caractère de la figure 1) puis après avoir effacé l'écran (ligne 3Ø) le dessine en 3 endroits.

A la ligne 4Ø on utilise l'instruction PSET et la désignation GR$(Ø), à la ligne 5Ø la désignation CHR$(128 + Ø) = CHR$ (128) puis un PRINT à la ligne 6Ø. On constate que toutes ces syntaxes peuvent être utilisées de façons équivalentes.

*** LINE**

Cette instruction permet de tracer n'importe quelle droite entre deux points (X_1, Y_1) et (X_2, Y_2) de l'écran ; et ceci aussi bien en mode graphique qu'en mode caractère.

Mode graphique

La syntaxe est :

LINE [(X_1, Y_1)] – (X_2, Y_2), C

X_1 et Y_1 sont les coordonnées du premier point ($Ø \leqslant X_1 \leqslant 319$ et $Ø \leqslant Y_1 \leqslant 199$).

X_2 et Y_2 sont les coordonnées du deuxième point ($Ø \leqslant X_2 \leqslant 319$ et $Ø \leqslant Y_2 \leqslant 199$).

C représente le code couleur de la ligne (voir paragraphe 2.4).

Si les points de la ligne doivent appartenir au graphisme alors $Ø \leqslant C \leqslant 15$.

Si les points de la ligne doivent appartenir au fond alors $-16 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs de graphisme ou de fond différentes ; si une ligne passe dans le même secteur qu'un point ou qu'une autre ligne, il risque d'y avoir des modifications de couleurs indésirables.

Exemples :

LINE (1ØØ, 1ØØ) – (3ØØ, 15Ø), 2

trace une ligne verte entre les points (1ØØ, 1ØØ) et (3ØØ, 15Ø)

LINE – (Ø, 15Ø), 2

trace une ligne verte entre les points (3ØØ, 15Ø) et (Ø, 15Ø) car le premier point par défaut est le dernier point utilisé dans une instruction graphique.

Par contre, après un RAZ effectuez un.

LINE (1ØØ, 1ØØ) – (2ØØ, 15Ø), — 2 [ENTREE]

Contrairement à ce qui pourrait être imaginé en premier lieu, la ligne rouge est en fait devenue un escalier. Tout s'est passé comme si la couleur rouge avait déteint sur l'ensemble des secteurs coupés par la droite.

Ceci s'explique simplement par le fait que lorsqu'un point de la droite est inscrit dans un secteur, il y modifie la couleur du fond qui devient le rouge (−2), et par conséquent l'ensemble du secteur ayant à l'origine un fond bleu ciel aura désormais un fond rouge (si au départ vous étiez en configuration caractères bleu et fond cyan).

Le même type de problème se pose lors de l'intersection de deux droites. Après avoir effacé l'écran (RAZ), entrez successivement :

LINE (1ØØ, 1ØØ) – (2ØØ, 15Ø), 1 [ENTREE]
LINE (1ØØ, 15Ø) – (2ØØ, 1ØØ), 2 [ENTREE]

la première ligne trace une ligne rouge entre les points (1ØØ, 1ØØ) et (2ØØ, 15Ø), la deuxième une ligne verte entre (1ØØ, 15Ø) et (2ØØ, 1ØØ). Observez alors l'intersection des deux droites : une partie de la ligne rouge proche de l'intersection est devenue verte. L'explication est simple : lors du tracé de la ligne verte, les secteurs communs aux deux droites (proches de l'intersection) se sont vu attribuer une couleur de graphisme verte et les points graphiques initialement rouges sont restés des points graphiques mais ont pris la couleur verte.

Ces deux derniers exemples montrent les limitations annoncées au début de ce chapitre ; il conviendra donc à l'utilisateur de prendre des précautions particulières lorsque des points de couleurs différentes risquent de se trouver trop proches les uns des autres.

A ce stade, il peut être intéressant d'observer un phénomène très classique concernant le tracé de droites peu inclinées sur écran à balayage télévision. Traçons une droite rouge entre les points (Ø, 1ØØ) et (3ØØ, 12Ø)

LINE (Ø, 1ØØ) – (3ØØ, 12Ø), 1 [ENTREE]

Nous constatons que la "droite" est en fait formée de paliers. Ceci ne constitue pas un défaut de votre ordinateur mais montre tout simplement une limitation liée aux écrans de télévision (l'algorithme de tracé de droites étant quant à lui tout à fait éprouvé).

Mode caractère

La syntaxe est :

LINE [(X_1, Y_1)] – (X_2, Y_2) "Caractère" [, [A] [, [B] [, C]]]

X_1 et Y_1 sont les coordonnées de la première case (début de la ligne) donc : $\emptyset \leqslant X_1 \leqslant 39$ et $\emptyset \leqslant Y_1 \leqslant 24$.

X_2 et Y_2 sont les coordonnées de la deuxième case (fin de la ligne), on a également : $\emptyset \leqslant X_2 \leqslant 39$ et $\emptyset \leqslant Y_2 \leqslant 24$.

"Caractère" représente le caractère qui s'inscrit dans toutes les cases par lesquelles passe la droite. "Caractère" doit obligatoirement être une chaîne ou une expression chaîne. Seul le premier caractère de l'expression sera pris en compte. "Caractère" peut revêtir les formes suivantes :

- "E" ou "EXEMPLE" où E est un caractère du code ASCII.
- A$ qui représente une expression chaîne.
- CHR$(N) où N représente le code ASCII du caractère.
- GR$(N) où N représente le numéro du caractère programmé par l'utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond des cases qui dessinent la ligne.

C indique par sa présence que les couleurs A et B doivent être interverties.

C ne peut prendre que les valeurs $\emptyset$ et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. A ce stade, la même remarque pour l'instruction PSET en mode caractère est valable : après exécution du programme, les caractères entrés au clavier auront la dernière couleur définie durant le programme par ce type d'instruction, il convient donc être vigilant.

Exemples :

```
LINE (Ø, 1Ø) - (3Ø, 1Ø) "+", 3, Ø
```

va provoquer une ligne horizontale de 31 caractères + de couleur jaune sur fond noir.

```
LINE (2Ø, Ø) - (2Ø, 2Ø) CHR$(127), 5
```

trace une ligne verticale de couleur mauve. CHR$(127) est un caractère qui remplit toute la case (64 points en couleur graphisme).

```
LINE (Ø, Ø) - (1Ø, 2Ø) "$", 1, 7
```

provoque le tracé d'une ligne de caractères $ rouges sur fond blanc. Bien entendu, le défaut constaté en mode graphique est ici amplifié par la diminution de la définition.

Le programme ci-dessous montre un exemple de dessin simple à réaliser à l'aide des instructions SCREEN, PSET et LINE.

```
1Ø SCREEN 4, Ø, Ø : CLS
2Ø LINE (Ø, 1Ø) - (39, 1Ø) CHR$(127
3Ø LINE (Ø, 16) - (39, 16) CHR$(127)
4Ø FOR I = Ø TO 39 STEP 4
5Ø FOR J = Ø TO 1
6Ø PSET (I + J, 13) CHR$(127), 3
7Ø NEXT J : NEXT I
8Ø LOCATE Ø, Ø : INPUT A$
9Ø SCREEN 4, 6, 6 : CLS : END
```

La ligne 1Ø impose le fond (noir) et indique la couleur graphique (bleu) qui va être utilisée aux lignes 2Ø et 3Ø. Les lignes 2Ø et 3Ø tracent deux lignes parallèles de couleur bleue, cette couleur n'est pas indiquée puisque la valeur par défaut est 4 (définie à la ligne 1Ø). Les lignes 4Ø à 7Ø tracent un pointillé jaune entre les deux lignes bleues. A la ligne 6Ø, la couleur jaune (3) est indiquée puisque dans le cas contraire, la valeur par défaut aurait été le bleu. La ligne 8Ø permet de replacer le curseur en haut à gauche de l'écran en attendant que la touche [ENTREE] soit enfoncée ; la valeur A$ ne correspond à rien mais évite une erreur de syntaxe. Lorsque la

touche [ENTREE] est enfoncée, le programme se termine non sans être retourné aux conditions initiales grâce à la ligne 9Ø.

Ce dessin peut représenter une route pour un jeu vidéo ou pour une animation quelconque.

*** BOX et BOXF**

Ces deux instructions sont très utiles et permettent de tracer facilement des rectangles pleins ou "creux", elles présentent l'intérêt de remplacer plusieurs instructions LINE à la suite :

BOXF permet de tracer des rectangles pleins.

BOX ne trace que le périmètre du rectangle.

La syntaxe est très proche de celle de l'instruction LINE.

Mode graphique

La syntaxe est :

BOX [F] [(X_1, Y_1)] – (X_2, Y_2), C

Si F est absent (BOX), le rectangle sera tracé avec la même finesse que les lignes, c'est-à-dire avec une largeur de 1 point. Si F est présent (BOXF), le rectangle aura les mêmes dimensions que celui produit par BOX mais sera plein (toute la surface sera colorée).

(X_1, Y_1) représente les coordonnées du point le plus en haut à gauche du rectangle (figure 1).

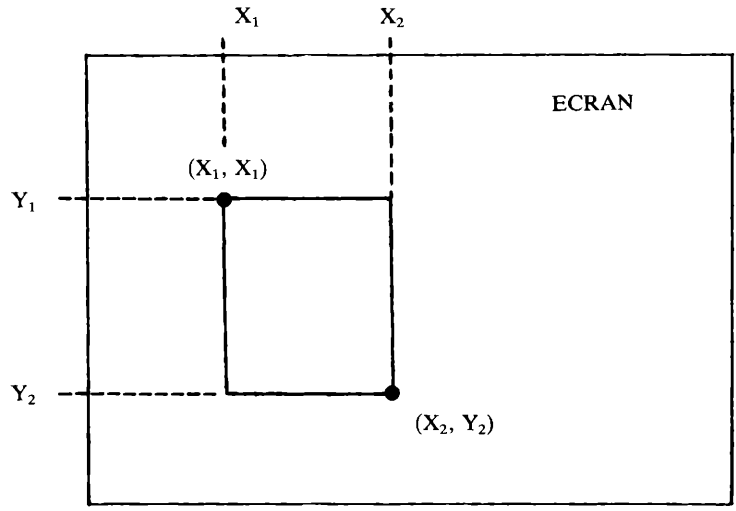
(X_2, Y_2) représente les coordonnées du point le plus en bas à droite du rectangle (figure 1).

En conséquence, les rectangles auront toujours des côtés parallèles aux bords de l'écran de télévision. D'autre part, si (X_1, Y_1) est omis, la valeur par défaut prend la précédente valeur utilisée dans une instruction graphique (ou une valeur nulle si aucune valeur n'a déjà été définie).

C représente le code couleur du rectangle (voir paragraphe 2.4).

Si les points du rectangle doivent appartenir au graphisme, alors : $\emptyset \leqslant C \leqslant 15$. Si les points doivent appartenir au fond alors : $-16 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs graphisme ou fond différentes ; si une partie du rectangle passe dans le même secteur qu'un point, une ligne ou une autre figure, il risque d'y avoir des modifications de couleur indésirables.



Exemple :

BOX (2ØØ, 1Ø) – (28Ø, 19Ø), 1

trace un rectangle rouge ; seul le périmètre est dessiné.

BOXF (2ØØ, 1Ø) – (28Ø, 19Ø), 1

trace un rectangle rouge ; toute la surface du rectangle est colorée.

Mode caractère

La syntaxe est :

BOX [F] ([X_1, Y_1)] – (X_2, Y_2) "Caractère"
[, [A] [, [B] [, C]]]

Si F est écrit (BOXF), le rectangle verra toutes les cases par lesquelles passe sa surface contenir « Caractère ». Si F est absent, seules les cases constituant le périmètre du rectangle contiendront « Caractère ».

(X_1, Y_1) représente les coordonnées de la case la plus en haut à gauche du rectangle ($\emptyset \leqslant X_1 \leqslant 39$, $\emptyset \leqslant Y_1 \leqslant 24$).

(X_2, Y_2) représente les coordonnées de la case la plus en bas à droite du rectangle.

"Caractère" représente le caractère qui s'inscrira dans les cases affectées par l'instruction BOX [F]. "Caractère" doit obligatoirement être une chaîne de caractères ou une expression chaîne. Seul le premier caractère de l'expression est pris en compte. "Caractère" peut revêtir les formes suivantes :

- "E" ou "EXEMPLE" où E est un caractère du code ASCII.
- A$ qui représente une expression chaîne.
- CHR$ (N) où N représente le code ASCII du caractère.
- GR$ (N) où N représente le numéro du caractère graphique utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond des cases qui dessinent le rectangle.

C indique par sa présence que les couleurs A et B doivent être inversées, C ne peut prendre que les valeurs $\emptyset$ et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. Après exécution du programme, les caractères entrés au clavier auront la dernière couleur définie durant le programme par ce type d'instruction.

Exemples :

```
BOX (1Ø, 1Ø) - (3Ø, 3Ø) ">", 5, Ø
```

trace un cadre de caractères > mauves sur fond noir.

```
BOXF (1Ø, 1Ø) - (3Ø, 3Ø) ">", 5, Ø
```

trace un rectangle dont la surface est formée de caractères > mauves sur fond noir.

Le programme suivant :

```
1Ø SCREEN Ø, Ø: CLS
2Ø BOXF (5, 5) – (14, 19) CHR$ (127), 4
3Ø BOXF (15, 5) – (24, 19) CHR$ (127), 7
4Ø BOXF (25, 5) – (34, 19) CHR$ (127), 1
5Ø LOCATE Ø, Ø: INPUT A$
6Ø SCREEN 4, 6, 6 : CLS : END
```

affiche un drapeau bleu, blanc, rouge sur fond noir. Les lignes 50 et 60 servent à revenir aux conditions initiales après exécution du programme.

*** ATTRB**

Cette instruction permet de doubler la largeur et/ou la hauteur des caractères.

La syntaxe est :

ATTRB [L], [H]

L vaut Ø pour une largeur normale et 1 pour une largeur double.

H vaut Ø pour une hauteur normale et 1 pour une hauteur doublée des caractères. Si la hauteur est doublée, le caractère occupe la ligne courante ainsi que la ligne précédente.

Dans le cas où un paramètre est absent, la valeur prise par défaut est celle définie par la dernière instruction de ce type ; à l'initialisation, les valeurs des deux paramètres L et H sont Ø.

Exemple :

Le programme suivant montre différents formats d'affichage obtenus à l'aide de ATTRB.

```
1Ø CLEAR, , 1 : SCREEN 1, Ø, Ø: CLS
2Ø DEFGR$ (Ø) = 177, 179, 183, 254, 254, 183, 179, 177
3Ø ATTRB 1, 1 : LOCATE 7, 2
4Ø PRINT "DEMONSTRATION"
5Ø COLOR 2 : PRINT GR$ (Ø), "A"
6Ø ATTRB, Ø
7Ø PRINT GR$ (Ø), "A"
8Ø LOCATE Ø, 9
9Ø ATTRB Ø, 1
```

```
1ØØ PRINT GR$ (Ø)
11Ø LOCATE 24, 9 : PRINT "A"
12Ø LOCATE Ø, 23 : END
```

La ligne 30 définit un affichage double hauteur, double largeur. Le mot "démonstration" apparaît donc en grands caractères rouges sur l'écran. Ensuite sont affichés un caractère graphique et une lettre pour chacun des trois cas de démonstrations doublées. Le premier cas affiché est hauteur doublée, largeur doublée (ligne 50). Le deuxième cas correspond à hauteur normale et largeur doublée (lignes 60 et 70), on remarque à la ligne 60 que L est omis puisqu'il a déjà été défini correctement à la ligne 30. Le dernier cas correspond à des caractères de largeur normale mais dont la hauteur est doublée (lignes 90 et 110).

* COLOR

Cette instruction permet de changer la couleur graphisme et fond des prochains caractères et points affichés.

La syntaxe est :

COLOR (A) (, B) (, C)

A indique la nouvelle couleur des caractères.

B indique la nouvelle couleur de fond.

C indique par sa présence que les couleurs de A et B doivent être permutées. C ne peut prendre que les valeurs Ø et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées par une instruction de ce type. Nous rappelons que A et B doivent avoir les valeurs suivantes :

Ø	pour le NOIR
1	" " ROUGE
2	" " VERT
3	" " JAUNE
4	" " BLEU
5	" " MAGENTA
6	" " CYAN
7	" " BLANC

8	"	" GRIS
9	"	" ROUGE CLAIR
10	"	" VERT CLAIR
11	"	" JAUNE CLAIR
12	"	" BLEU CLAIR
13	"	" MAGENTA CLAIR
14	"	" CYAN CLAIR
15	"	" ORANGE

Exemples :

COLOR 3, Ø

provoque un affichage de tous les prochains caractères en jaune sur fond noir. Si cette instruction est suivie de :

COLOR, 1

alors les prochains caractères seront jaunes sur fond rouge, la valeur par défaut de A étant 3.

Cette instruction est très utilisée pour modifier la couleur des caractères que l'on frappe au clavier (notamment après un programme n'ayant pas réinitialisé les couleurs correctement). Il faut également signaler que le fait d'appuyer sur la touche RAZ après une commande COLOR provoque une modification des couleurs de la fenêtre de travail qui après s'être effacée prend la couleur du fond définie lors de la commande COLOR A, B.

Exemples : si les caractères sont bleus sur fond cyan.

COLOR 2, Ø [ENTREE]

provoque un changement de couleur pour tous les caractères entrés par la suite au clavier et une action sur la touche RAZ colore la fenêtre de travail en noir. On remarque que la couleur du cadre n'a pas changé.

3.5.2 Les fonctions de contrôle de l'écran

Nous appelons fonctions de contrôle de l'écran les fonctions permettent de déterminer l'état de l'écran ou d'une partie de l'écran à un moment donné. Ces fonctions permettant par exemple de trouver la position du curseur ou encore de savoir quelle est la couleur du point de coordonnées (X, Y).

Elle permettent également d'éviter la mise en mémoire systématique (ailleurs que dans la mémoire d'écran) d'informations concernant l'image créée.

*** SCREEN**

Cette fonction permet de déterminer le code ASCII du caractère se trouvant à l'intérieur d'une case quelconque de l'écran.

La syntaxe est :

SCREEN (X, Y)

X représente le numéro de la colonne où se situe la case.

Y représente le numéro de la ligne où se situe la case.

Nous sommes en mode graphique, on a $\emptyset \leqslant X \leqslant 39$ et $\emptyset \leqslant Y \leqslant 24$.

La valeur obtenue est le code ASCII (voir annexe) du caractère situé en (X, Y). Seuls les caractères du code ASCII sont reconnus. Les caractères graphiques utilisateur ne faisant pas partie du code ASCII ne seront pas identifiés et le code obtenu sera $\emptyset$ comme tous les autres caractères non identifiables.

Le code des minuscules et du ç seront :

128	ç	138	í
129	à	139	î
130	á	140	ï
131	â	141	ò
132	ä	142	ó
133	è	143	ô
134	é	144	ö
135	ê	145	ù
136	ë	146	ú
137	ì	147	û
		148	ü

Exemple :

Le programme suivant :

```
1Ø CLEAR, , 1 : CLS
2Ø DEFGR$ (Ø) = 177, 179, 183, 254, 254, 183, 179, 177
3Ø PSET (1Ø, 1Ø) GR$ (Ø)
4Ø PSET (1Ø, 15) "A"
5Ø LOCATE Ø, Ø
6Ø PRINT SCREEN (1Ø, 1Ø), SCREEN (1Ø, 15)
```

inscrit tout d'abord un caractère graphique utilisateur en (1Ø, 1Ø) puis la lettre A en (1Ø, 15). La ligne 60 affiche en haut de l'écran le code ASCII issu de l'instruction SCREEN de ces deux caractères. On obtient Ø pour le caractère utilisateur puisqu'il ne fait pas partie du code ASCII et 65 pour A.

*** POINT**

Cette fonction permet de déterminer la couleur de n'importe lequel des 64 000 points de l'écran.

La syntaxe est :

POINT (X, Y)

X représente la coordonnée horizontale du point (Ø ≤ X ≤ 319).

Y représente la coordonnée verticale du point (Ø ≤ Y ≤ 199).

On obtient une valeur comprise entre Ø et 15 si le point appartient au graphisme et entre − 1 et − 16 si le point appartient au fond. Le code des couleur étant le code classique du mode graphique, nous avons :

pour le graphisme	pour le fond
Ø noir	— 1 noir
1 rouge	— 2 rouge
2 vert	— 3 vert
3 jaune	— 4 jaune
4 bleu	— 5 bleu
5 magenta	— 6 magenta
6 cyan	— 7 cyan
7 blanc	— 8 blanc
8 gris	— 9 gris
9 rouge	— 10 rouge clair
10 vert clair	— 11 vert clair
11 jaune clair	— 12 jaune clair
12 bleu clair	— 13 bleu clair
13 magenta clair	— 14 magenta clair
14 cyan clair	— 15 cyan clair
15 orange	— 16 orange

Exemple :

```
1Ø PSET (2ØØ, 1ØØ), 1
2Ø PRINT POINT (2ØØ, 1ØØ)
```

Ce programme inscrit un point rouge en (2ØØ, 1ØØ) puis imprime la valeur de la couleur du point (2ØØ, 1ØØ) qui est 2.

Utilisation de la fonction POINT en coordonnée « caractère »

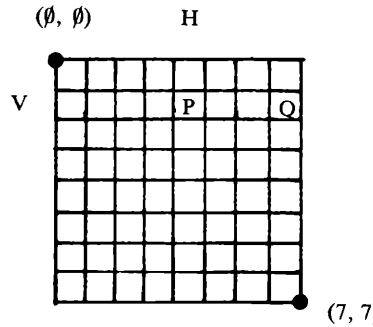
Dans beaucoup de jeux, ainsi que dans des programmes plus « sérieux », il est nécessaire de repérer un caractère graphique par sa couleur. Cette technique s'avère indispensable lorsqu'il s'agit des caractères graphiques utilisateur puisqu'il n'existe aucun autre moyen de les détecter.

Le problème est le suivant. Connaissant les coordonnées graphiques X et Y ($Ø \leqslant X \leqslant 39$ et $Ø \leqslant Y \leqslant 24$) d'une case, comment vérifier ou connaître la couleur du caractère s'y trouvant ?

Un raisonnement assez simple permet de déduire la formule suivante :

Code couleur = POINT (X * 8 + H, Y * 8 + V)

(X, Y) sont les coordonnées de la base scrutée, H et V sont les coordonnées du point scruté à l'intérieur de la case. Ces dernières coordonnées sont définies de la façon suivante ($Ø \leqslant H \leqslant 7$et $Ø \leqslant V \leqslant 7$) :



Le point le plus en haut à gauche de la case a pour coordonnées

(Ø, Ø) et le plus en bas à droite (7, 7). Dans la figure ci-dessus, P a pour coordonnées (3, 3) et Q (5, 3). Par exemple, si la case a pour coordonnées X = 12 et Y = 16 et que l'on veut connaître la couleur du point Q (5, 3) dans cette case, il faudra utiliser la fonction

POINT (12 * 8 + 5, 16 *8 + 3)

soit :

POINT (1Ø1, 131)

*** CSRLIN**

La syntaxe est :

CSRLIN

Cette fonction donne le numéro de la ligne où se trouve le curseur.

Exemple :

Le curseur se trouve sur la ligne 9 ; vous effectuez :

PRINT CSRLIN [ENTREE]

le résultat est 10 car après avoir appuyé sur [ENTREE] le curseur se trouve sur la ligne suivant 9 voit : 1Ø

*** POS**

La syntaxe est :

POS

Cette fonction donne le numéro de la colonne où se trouve le curseur.

Exemple :

Le curseur se trouve, après avoir écrit :

PRINT POS [ENTREE]

dans la colonne Ø puisque l'instruction PRINT fait passer à la ligne (la commande PRINT aussi). Le nombre affiché sera donc Ø.

3.5.3 Entrées-sorties d'images

Nous avons regroupé dans ce paragraphe, les trois instructions permettant d'enregistrer ou de charger des images et (ou) de les imprimer.

*** SAVEM**

D'une façon générale, cette instruction permet de sauvegarder des programmes en binaire. La mémoire d'écran représentant un espace codé en binaire, il est également possible de la sauvegarder à l'aide de cette instruction

La syntaxe générale est :

SAVEM «[nom du programme]», [adresse D], [adresse F], [adresse exéc.]

«[nom du programme]» est du même type que celui employé par la commande SAVE.

[adresse D] est l'adresse de début du programme binaire ; pour l'image on a : [adresse D] = Ø

[adresse F] est l'adresse de la fin du programme binaire ; pour l'image on a : [adresse F] = 7 999

[adresse exéc.] est l'adresse de début d'exécution du programme ; pour l'image on choisira : [adresse exéc.] = Ø

La mémoire d'écran étant composée de 2 plans mémoires situés à la même adresse mais sélectionnés par un buffer se trouvant à l'adresse &HA7CØ (voir paragraphe 4) le chargement couleur se fera en deux étapes.

Le programme-type de sauvegarde s'insèrera dans le programme dont on veut préserver l'image ; le listing sera le suivant :

```
XXXX      POKE &HA7CØ, PEEK (&A7CØ) OR 1
XXXX+10 SAVEM "GRAPH", Ø, 7 999, Ø
XXXX+20 POKE &HA7CØ, PEEK (&A7CØ) AND 254
XXXX+30 SAVEM "COUL", Ø, 7999, Ø
XXXX+40 RETURN (dans le cas d'un sous-programme)
```

La ligne XXXX sélectionne le plan-mémoire graphisme, la ligne XXXX+1Ø le préserve sur bande magnétique. La ligne XXXX+2Ø

sélectionne le plan-mémoire couleur et la ligne XXXX+3Ø le préserve sur bande magnétique.

*** LOADM**

Cette instruction complète la précédente en chargeant une image stockée sur bande magnétique.

La syntaxe est :

LOADM ["nom de programme"], [décalage] [, R]]

En ce qui concerne l'image, décalage doit être nul. "nom du programme" a la même signification que pour l'instruction SAVEM.

Le programme type de chargement d'une image est :

```
XXXX      POKE &HA7CØ, PEEK (&HA7CØ) OR1
XXXX+1Ø LOADM "GRAPH"
XXXX+2Ø POKE &HA7CØ, PEEK (&HA7CØ) AND 254
XXXX+3Ø LOADM "COUL"
XXXX+4Ø RETURN (dans le cas d'un sous-programme)
```

*** SCREENPRINT**

Cette instruction permet de recopier sur imprimante graphique le contenu de l'écran.

La syntaxe est :

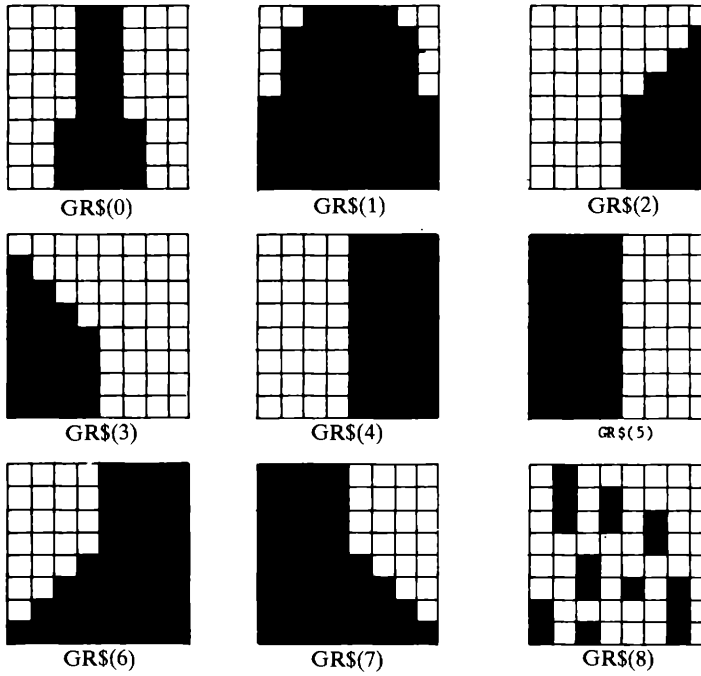
SCREENPRINT

3.5.4 Exemples de programmes utilisant les instructions graphiques

Le programme pris en exemple et dont le listing se trouve ci-dessous a été conçu pour utiliser le plus d'instructions graphiques différentes possibles dans des configurations différentes. Le thème est l'animation du décollage d'une fusée.

Les caractères graphiques utilisateur sont définis aux lignes 20 à

100, les différents graphismes sont les suivants :



La fusée est dessinée à l'aide des 8 premiers caractères. Le dernier GR$ (8) permet de dessiner la traînée de fusée (ligne 220).

Les lignes 200 à 230 assurent le déplacement de la fusée, les lignes 130 à 180 le compte à rebours.

Le sous-programme 1 000 dessine la figure à un emplacement dépendant de K dans le sens vertical (permet le déplacement). Le sous-programme 2 000 dessine la tour de lancement.

La description détaillée de ce programme serait très longue. Cependant le lecteur soucieux d'apprendre et de se perfectionner aura tout intérêt (à titre d'exercice) à comprendre l'utilité de chaque ligne et de chaque instruction. Tous les éléments permettant de comprendre entièrement et de perfectionner ce programme ont été donnés dans ce chapitre.

```
 1Ø CLEAR, , 9 : SCREEN, Ø, Ø: CLS
 2Ø DEFGR$ (Ø) = 24, 24, 24, 24, 24, 6Ø, 6Ø, 6Ø
 3Ø DEFGR$ (1) = 6Ø, 126, 126, 126, 255, 255, 255, 255
 4Ø DEFGR$ (2) = Ø, 1, 3, 7, 15, 15, 15, 15
 5Ø DEFGRS (3) = Ø, 128, 192, 224, 24Ø, 24Ø, 24Ø, 24Ø
 6Ø DEFGR$ (4) = 15, 15, 15, 15, 15, 15, 15, 15
 7Ø DEFGR$ (5) = 24Ø, 24Ø, 24Ø, 24Ø, 24Ø, 24Ø, 24Ø, 24Ø
 8Ø DEFGR$ (6) = 15, 15, 15, 15, 31, 63, 127, 255
 9Ø DEFGR$ (7) = 24Ø, 24Ø, 24Ø, 24Ø, 248, 252, 254, 255
1ØØ DEFGR$ (8) = 64, 8Ø, 84, 4, 32, 42, 13Ø, 162
11Ø GOSUB 2ØØØ
12Ø K = Ø: GOSUB 1ØØØ
13Ø ATTRB 1, 1
14Ø FOR I = 9 TO Ø STEP −1
15Ø LOCATE Ø, 2Ø: PRINT I
16Ø FOR J = 1 TO 4ØØ: NEXT J
17Ø NEXT I
18Ø ATTRB Ø,Ø
2ØØ FOR K = Ø TO − 9 STEP −1
21Ø GOSUB 1ØØØ
22Ø LINE (18, 21 + K) − (2Ø, 21 + K) GR$ (8), 3, 7
23Ø COLOR, Ø: NEXT K
24Ø COLOR 2 : END
1ØØØ 'FUSEE
1Ø1Ø PSET (19, 16 + K) CHR$ (127), 4
1Ø2Ø PSET (19, 11 + K) GR$ (Ø), 1
1Ø3Ø PSET (19, 12 + K) GR$ (1), 1
1Ø4Ø FOR I = Ø TO 2
1Ø5Ø PSET (19, 13 + I + K) CHR$ (127)
1Ø6Ø NEXT I
1Ø7Ø PSET (18, 16 + K) GR$ (2).
1Ø8Ø PSET (2Ø, 16 + K) GR$ (3)
1Ø9Ø FOR I = Ø TO 2
11ØØ PSET (18, 17 + I + K) GR$(4), 1
111Ø PSET (19, 17 + I + K) CHR$(127), 2
112Ø PSET (2Ø, 17 + I + K) GR$(5), 1
113Ø NEXT I
114Ø PSET (18, 2Ø + K) GR$(6)
115Ø PSET (2Ø, 2Ø + K) GR$(7)
116Ø PSET (19, 2Ø + K) CHR$(127)
```

```
117Ø LINE (18, 21) – (2Ø, 21) CHR$(127), 4
118Ø RETURN
2ØØØ FOR I = Ø TO 72 STEP 24
2Ø1Ø LINE (176, 72 + I) – (192, 72 + I), 4
2Ø2Ø LINE (176, 96 + I) – (192, 72 + I), 4
2Ø3Ø LINE (176, 72 + I) – (192, 96 + I), 4
2Ø4Ø NEXT I
2Ø5Ø LINE (176, 72) – (176, 168), 4
2Ø6Ø LINE (192, 72) – (192, 168), 4
2Ø7Ø LINE (16, 21) – (24, 21) CHR$(127), 4
2Ø8Ø RETURN
```

CHAPITRE 4

LES SONS ET LE MO 5

Ce chapitre a pour but d'étudier les possibilités sonores du MO5 et de montrer par quelques exemples comment mettre en œuvre ces possibilités. Après avoir lu les quelques pages qui suivent, vous serez capable de composer de la musique sur votre MO5, de générer des effets sonores spéciaux pour agrémenter vos jeux : tir de mitraillette, sirène...

Un programme de jeu montre en particulier comment faire très rapidement un jeu simple mais attrayant grâce au graphisme et au son.

4.1 LE GÉNÉRATEUR DE SONS DU MO 5

Le MO5 est pourvu d'un générateur de sons très perfectionné qui lui permet de jouer de la musique sur une plage de cinq octaves. Le son est transmis par le haut-parleur du téléviseur.

Les sons produits par le MO5 sont les notes classiques (avec ou

sans dièse et bémol). Le tempo, l'attaque et la durée de chaque note sont réglables.

4.2 LECTURE D'UNE CASSETTE

L'ensemble MO5 + lecteur-enregistreur de programmes peut se comporter comme un lecteur de cassettes normal.

Pour cela, il vous suffit de placer la cassette que vous désirez écouter dans le lecteur-enregistreur de programmes. Faites alors « LOAD » et pressez [ENTREE]. Pressez ensuite la touche PLAY du lecteur-enregistreur de programmes. Vous entendrez alors la musique enregistrée sur la cassette.

L'intérêt principal de cette possibilité n'est pas, bien sûr, d'écouter de la musique pré-enregistrée, la qualité de reproduction n'étant pas excellente... Ce procédé est particulièrement utile pour repérer les programmes enregistrés sur une cassette grâce à une annonce vocale (le nom du programme par exemple) ou musicale. L'annonce doit être enregistrée avec un magnétophone normal.

4.3 INSTRUCTION PLAY

L'instruction PLAY permet de faire « jouer » de la musique par le MO5.

La syntaxe de cette instruction est la suivante :

PLAY [expression chaîne], [expression chaîne]...

Les chaînes de caractères contenues dans expression chaîne doivent comporter un ou plusieurs éléments choisis obligatoirement parmi les cinq éléments suivants :

— note
— octave
— attaque
— tempo
— durée

Ces différents éléments peuvent être séparés les uns des autres par un point-virgule. Cela étant facultatif, le point-virgule sera systématiquement omis dans les exemples qui vont suivre.

Remarque : PLAY n'est pas seulement une instruction, c'est aussi une commande.

Faites

PLAY "DO RE MI FA SO LA SI DO" [ENTREE]

et vous entendrez le MO5 « monter » la gamme de do majeur.

4.3.1 Notes

Le MO5 peut jouer toutes les notes classiques :

DO RE MI FA SOL LA SI.

Attention, la note SOL est codée par SO ! !

Ces notes peuvent comporter un dièse, il suffit pour cela de faire suivre le nom de la note par le symbole #.

Exemple : DO#

Les notes peuvent aussi être avec bémol en faisant suivre leur nom par le symbole b (b minuscule).

Exemple : SI b

Le MO5 peut jouer aussi une note très particulière : le silence, noté P.

4.3.2 Octave

Le générateur de sons du MO5 a une plage de cinq octaves (de 1 à 5). Ø1 est l'octave la plus grave, Ø5 la plus aiguë.

PLAY « *On* » (avec *n* compris entre 1 et 5) sélectionne l'octave n. Tant qu'une autre instruction de même type n'aura pas été effectuée, toute note jouée sera à l'octave n.

Par défaut le MO5 joue à l'octave 4.

Exemples :

```
PLAY "DO RE MI RE DO"
PLAY "O2 DO RE MI RE DO"
PLAY "O5 DO RE MI O4 RE O3 DO"
```

4.3.3 Attaque

L'attaque de la note émise par le générateur de sons est réglable grâce à l'élément An où n est un entier compris entre Ø et 255.

AØ correspond à un son continu. Une valeur élevée pour l'attaque donne un son rapidement amorti. En pratique une attaque supérieure à 118 donne un son inaudible.

Le réglage de l'attaque est particulièrement intéressant pour obtenir des sons très secs. On peut ainsi, par exemple, simuler une rafale de mitraillette par :

```
PLAY "A11Ø T2 DO DO DO DO DO DO DO DO DO
      DO"
```

Ce genre de bruitage est fréquemment employé dans les programmes de jeux.

De même que pour le choix de l'octave, le choix d'une attaque donnée reste variable jusqu'à ce qu'un nouvel élément An soit rencontré.

Exemple :

```
PLAY "A2 DO DO A3Ø DO DO"
```

Par défaut l'attaque est AØ.

4.3.4 Durée

Ln indique la durée relative des notes qui suivent cet élément, n'étant un entier variant entre 1 (durée la plus courte) et 96 (durée la plus longue).

On peut poser comme convention : L24 correspond à une noire.

Cela permet d'établir le tableau suivant :

L 96 = ronde
L 48 = blanche
L 24 = noire
L 12 = croche
L 6 = double croche
L 3 = triple croche
L 72 = blanche pointée
L 36 = noire pointée
L 18 = croche pointée
L 9 = double croche pointée

Bien entendu toutes les durées intermédiaires sont possibles, ce qui permet d'obtenir des triolets, quintolets, etc.

Par défaut, le MO5 prend L 24.

Remarque : P (silence) est une note à part entière. Il est donc possible d'obtenir un soupir (L24 P), un demi-soupir (L12 P), etc.

Exemple :

```
PLAY "T15 AØ L6 DO RE L18 MI L6 SO L18 SO L6 LA
L12 SO MI DO L6 DO RE L12 MI MI RE DO L24 RE L12 P"
```

4.3.5 Tempo

L'élément Tn (où n est un entier compris entre 1 et 255) règle la vitesse absolue de l'exécution en agissant dans les mêmes proportions sur les durées de toutes les notes qui suivent.

T1 est le tempo le plus rapide alors que T255 est le tempo le plus lent. Par défaut le tempo est pris à T5.

Exemple :

```
PLAY "T1 DO MI SO MI DO T1Ø DO MI DO"
```

4.3.6 Exemples de mise en œuvre — Musique

*** Exécution d'un petit air de musique très connu**

Entrez le programme suivant :

```
1Ø A$ = "T1Ø L6 DO RE L12 MI SO L18 SO L6 LA L12
   SO MI DO L6 DO RE L12 MI MI RE"
2Ø B$ = "L12P L24 FA L36 FA L12 LA L36 LA L12 LA
   SO SO MI DO L24 RE L12 P"
3Ø C$ = "DO L24 RE L12P" : D$ = "L12 RE L24 DO
   L12 P"
4Ø FOR I = TO 2
5Ø PLAY A$ + C$ + A$ + D$ : PLAY B$ + A$ + D$
6Ø NEXT I
```

Vous reconnaîtrez certainement (du moins nous l'espérons) l'air : Oh Suzanna.

Ce programme appelle deux remarques :

— Il est préférable, pour programmer une musique, d'écrire d'abord cette musique sur une portée ; cela facilitera grandement la programmation :

Exemple :

L6 DO RE L12 MI SO L18 SO L6 LA L12 SO MI...

— On a tout intérêt à mettre dans des variables chaînes les séquences musicales qui reviennent plusieurs fois au cours d'un morceau. Le programme précédent en est un bon exemple. Il pourra parfois s'avérer nécessaire de réserver de la place mémoire pour la manipulation des chaînes de caractères à l'aide de l'instruction CLEAR (reportez vous au chapitre sur le BASIC du MO5 pour des explications complètes de l'instruction CLEAR). Les chaînes « musicales » sont en effet souvent très longues et l'erreur 14 peut être provoquée par le manque de mémoire nécessaire pour leur manipulation.

*** — Sirène/alarme**

Le fonctionnement d'une sirène est très simple : le son croît et décroît de façon continue et très rapidement.

Il est impossible de faire jouer au MO5 des sons espacés de moins d'un demi-ton. Cela n'est pas gênant dans la mesure où l'expérience montre qu'une gamme chromatique exécutée à grande vitesse produit quand même l'effet de sirène recherché. La vitesse d'exécution « gomme » la différence minimale d'un demi-ton entre les notes, et le son semble varier de façon continue.

Le programme ci-dessus génère quatre bruitages différents que vous pourrez utiliser avec profit dans vos programmes de jeux. Ils recouvrent en effet toute la gamme des bruits galactiques, rayons lasers, OVNI, etc. Le bruitage 4 est classiquement employé dans les jeux où l'on doit incrémenter ou décrémenter un score.

```
10 A$ = "DO DO # RE RE # MI FA SO SO # LA LA# SI"
20 B$ = "SI LA # LA SO # SO FA # FA MI RE # RE DO # DO"
30 CLS : INPUT "1, 2, 3 ou 4?", N
40 N = INT (N) : IF (N - 1) * (N - 4) > 0 THEN 30
50 ON N GOSUB100, 200, 300, 400
60 GOTO 30
100 FOR I = 1 TO 10
110 PLAY "O4 T1A0L6" + A$
120 NEXT I
130 RETURN
200 PLAY "T1 A0L4"
210 FOR I = 1 TO 10
220 PLAY "O4" + A$ + "O5" + A$
230 NEXT I
240 RETURN
300 PLAY "T1 A0L4"
310 FOR I = 1 TO 10
320 PLAY "O3" + A$ + "O4" + A$ + "O5" + A$ + "O4" + B$
330 NEXT I
340 RETURN
400 LOCATE 20, 15 : D = 0: PRINT D : PLAY "L4 A0T1"
410 FOR I = 1 TO 10
420 PLAY "O3" + A$ + "O4" + A$ + "O5" + A$ : D = D + 1
430 LOCATE 20, 15 : PRINT D
440 NEXT I
450 FOR I = 1 TO 10
460 PLAY "O5" + B$ + "O4" + B$ + "O3" + B$ : D = D - 1
470 LOCATE 20, 15 : PRINT D
```

```
48Ø NEXT I
49Ø RETURN
```

*** — Jeu : Missiles**

La principale application des sons concerne les jeux. Bien sonoriser un jeu est fondamental pour rendre celui-ci le plus attrayant possible. Les exemples précédents vous ont donné quelques aperçus de ce que l'on peut attendre du générateur de sons du MO5. Nous vous proposons maintenant un programme complet de jeu : MISSILES qui fait appel aux possibilités sonores du MO5.

Vingt cibles sont affichées en haut de l'écran. Un petit chariot mobile se déplace alternativement de droite à gauche et de gauche à droite dans le bas de l'écran. Quand vous pressez la touche F, le petit chariot s'arrête et tire un missile en direction des cibles. Vous disposez de trente missiles pour descendre le maximum possible de cibles.

```
  5 CLEAR, , 1 : GOSUB 1ØØØØ
 1Ø FOR I = Ø TO 39
 2Ø LOCATE I, 2Ø: PRINT "*"
 3Ø E$ = INKEY$ : IF E$ = "F" THEN GOSUB 1ØØØ
 4Ø LOCATE I, 2Ø: PRINT " "
 5Ø NEXT I
 60 FOR I = 39 TO Ø STEP - 1
 7Ø LOCATE I, 2Ø: PRINT "*"
 8Ø E$ = INKEY$ : IF E$ = "F" THEN GOSUB 1ØØØ
 9Ø LOCATE I, 2Ø: PRINT " "
 1ØØ NEXT I
 11Ø GOTO 1Ø
1ØØØ PLAY "AØL4T104"
1Ø1Ø B = B - 1
1Ø2Ø COLOR 6, Ø: LOCATE 28, 22 : PRINT B :
     COLOR 2, Ø
1Ø4Ø FOR J = 10 TO 1 STEP - 1
1Ø5Ø LOCATE I, J : PRINT " + "
1Ø6Ø PLAY A$ (j)
1Ø7Ø LOCATE I, J : PRINT " "
1Ø8Ø NEXT J
1Ø9Ø IF POINT (4 + 8 * I, 4) <> 1 THEN RETURN
```

```
11ØØ LOCATE I, Ø: PRINT " "
111Ø COLOR 6, Ø
112Ø FOR J = 1 TO 5 : S = S + 1 : LOCATE 7, 22 :
     PRINT S : PLAY B$ + C$ : NEXT J
113Ø COLOR 2, Ø
114Ø RETURN
1ØØØØ CLS : S = Ø : B = 3Ø : DIM A$ (26) : SCREEN 2, Ø,
     Ø
1ØØ1Ø C$ = "DO DO # RE RE # MI FA FA # SO SO#
     LA LA # SI"
1ØØ2Ø B$ = "AØT1L4"
1ØØ3Ø FOR I = 1 TO 21
1ØØ4Ø READ A$(I)
1ØØ5Ø NEXT I
1ØØ6Ø DATA "SO", "FA #", "FA", "MI", "RE#", "RE",
     "DO #", "DO", "O5", "SI", "LA #", "LA",
     "SO #", "SO", "FA #", "FA", "MI", "RE #",
     "RE", "DO #", "DO"
1ØØ7Ø DEFGR$ (Ø) = 6Ø, 126, 219, 255, 255, 153, 153, 153
1ØØ8Ø ATTRB 1, 1 : COLOR 4, Ø
1ØØ9Ø LOCATE 1Ø, 12 : PRINT "MISSILES" : ATTRB Ø,
     Ø : COLOR 2, Ø
1Ø1ØØ LOCATE Ø, 22 : INPUT "PRESSEZ ENTREE
     POUR COMMENCER", Q$ : CLS
1Ø11Ø FOR I = Ø TO 38 STEP 2
1Ø12Ø LOCATE I, Ø, Ø : COLOR 1, Ø : PRINT GR$(Ø)
1Ø13Ø NEXT I
1Ø14Ø COLOR 5, Ø
1Ø15Ø LOCATE Ø, 22 : PRINT "SCORE :" ; S : LOCATE
     2Ø, 22 : PRINT "BALLES :" ; B
1Ø16Ø COLOR 2, Ø
1Ø17Ø RETURN
```

Nous espérons que ce jeu vous plaira et que vous y trouverez des idées pour faire vos propres jeux : bruitage du tir d'un missile, ...

Nous vous laissons le soin de sonoriser vous-même les jeux qui sont donnés dans le chapitre d'initiation au BASIC. Vous trouverez des versions élaborées de ces jeux ainsi que MISSILE dans le livre «JEUX SUR MO5» chez le même éditeur.

4.4 INSTRUCTION BEEP

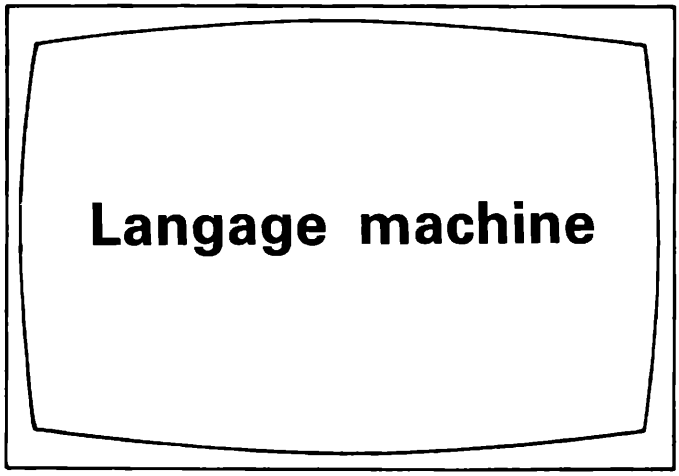
Cette instruction, qui est aussi une commande, produit un son court identique à celui qui est produit à la frappe d'une touche de clavier.

Faire BEEP et PRINT CHR$(7) est équivalent car de façon standard le caractère de code 7 dans le code ASCII représente le bip sonore.

Exemple :

```
1Ø FOR I = 1 TO 1Ø
2Ø R = INT (RND (1) * 1 + 1) : PRINT R
3Ø IF R = 7 THEN BEEP : END
4Ø NEXT I
5Ø PRINT CHR$(7)
6Ø END
```

CHAPITRE **5**



5.1 INTRODUCTION

Pour réaliser des programmes qui nécessitent une grande vitesse d'exécution (les jeux d'actions rapides par exemple), le Basic n'est pas « compétent ». En effet, chaque instruction Basic est traduite lors de son exécution ; si elle est exécutée dix fois, elle est traduite en code machine dix fois. Ceci provoque une perte de temps préjudiciable au programme ; aussi vous serez intéressé rapidement à programmer en langage machine ou en assembleur plus exactement. Le langage machine n'étant qu'une suite de nombres binaires (ou hexadécimaux si on regroupe les chiffres), le langage d'assemblage a été créé pour pouvoir mémoriser plus rapidement les codes. A chaque code machine correspond un mnémonique. Par exemple, il est plus facile de se souvenir de LDA [1Ø,X] que de A6 981Ø. Nous verrons plus loin dans ce chapitre les différents mnémoniques du langage d'assemblage du MO5. Mais avant cette description, nous devons connaître quelques aspects de la structure

matérielle du MO5. Nous donnerons aussi dans ce chapitre de nombreuses « informations » (variables systèmes...) pour programmer en assembleur de façon efficace.

5.2 STRUCTURE MATÉRIELLE DU MO5

5.2.1 Introduction

Un ordinateur personnel tel que le MO5 est avant tout une machine à traiter l'information. Elle reçoit, stocke, traite et communique des données avec le clavier, le téléviseur, ou avec le magnétophone à cassettes.

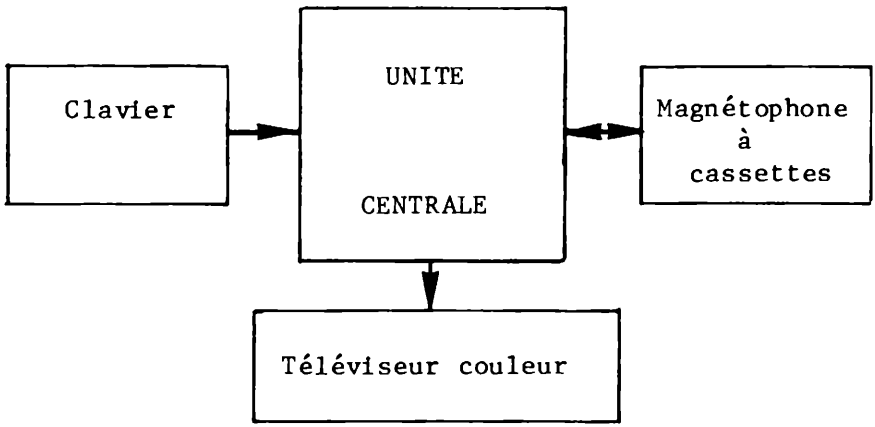


Figure 1 : *Structure du MO5*

Le cœur de votre MO5 est appelé unité centrale (CPU en anglais). Cette unité supervise le fonctionnement du système tout entier. Elle décide quand il faut aller chercher les données au clavier, les envoyer sur l'écran ou réaliser un transfert vers la cassette (ou la disquette). Elle exécute les programmes que *vous* lui demandez. L'unité centrale est organisée autour d'un circuit intégré (« puce électronique ») appelé un microprocesseur. Celui-ci fait les calculs, lit et écrit les données, etc. ; il a besoin pour cela

d'autres circuits appelés des mémoires, qui contiennent le programme et ses données. Les deux types suivants de mémoire existent :

— les mémoires mortes — MEM en abrégé — (Read Only Memory — ROM) dont le contenu est permanent, n'est pas effacé lorsque l'on coupe le courant et qui contiennent notamment le BASIC et les sous-programmes de gestion des entrées-sorties. Vous ne pouvez pas les modifier.

— les mémoires vives — MEV en abrégé — (Random Acces Memory — RAM) dans lesquelles il est possible de lire et écrire de l'information aussi souvent que cela est souhaitable. Elles contiendront par exemple vos programmes et leurs données, mais leur contenu sera effacé par toute coupure de courant.

En plus des mémoires et du microprocesseur, votre MO5 contient des circuits qui réalisent l'interface électronique avec les périphériques que sont le clavier, le téléviseur, le lecteur de programme sur cassettes, le crayon lumineux...

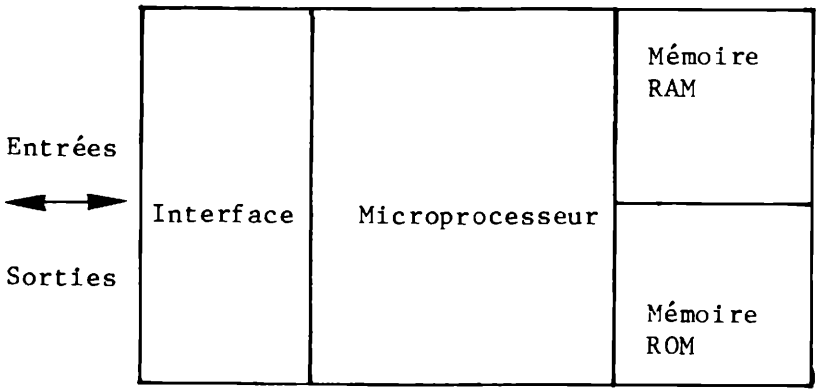


Figure 2 :*Unité centrale du MO5*

Il est possible de gérer d'autres entrées-sorties en ajoutant des extensions sur le connecteur plat situé à l'arrière du MO5

5.2.2 Numérotations binaires et hexadécimales

Toute information dans un ordinateur est codée sous forme d'une suite de bit (binary digit en anglais), c'est-à-dire comme une succession de Ø et de 1 ; ceci résulte des circuits utilisés en électronique pour fabriquer des ordinateurs qui travaillent sur deux tensions (Ø ou 5 volts). Dans le cas des microprocesseurs « 8 bits » comme le Motorola 68Ø9 (celui de votre MO5), les bits sont groupés par huit et ceux-ci sont appelés des octets (1 octet = 8 bits). Les programmes, les données, etc. sont codés sur des suites d'octets.

Par commodité, les groupes de bits ont été codés par des systèmes de numération différents de la base 1Ø. Ainsi, sur trois bits il est possible de coder huits valeurs (Ø à 7) :

binaire	décimal
ØØØ	Ø
ØØ1	1
Ø1Ø	2
Ø11	3
1ØØ	4
1Ø1	5
11Ø	6
111	7

Le système de numérotation adopté est alors la base 8 (Ø, 1, 2, 3, 4, 5, 6, 7, 1Ø, 11, 12,...) appelé encore l'octal. De même, il est possible d'utiliser la base 16 (ou le système hexadécimal) pour coder des groupes de quatre bits

binaire	hexadécimal
ØØØØ	Ø
ØØØ1	1
ØØ1Ø	2
ØØ11	3
Ø1ØØ	4
Ø1Ø1	5
Ø11Ø	6
Ø111	7

binaire	héxadécimal
1ØØØ	8
1ØØ1	9
1Ø1Ø	A
1Ø11	B
11ØØ	C
11Ø1	D
111Ø	E
1111	F

Le système héxadécimal est le plus utilisé tandis que l'octal tend à disparaître. Les lettres A à F de l'héxadécimal correspondent aux nombres 1Ø à 15 en système décimal. Ainsi, on a la correspondance suivante :

base 1Ø : Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18...

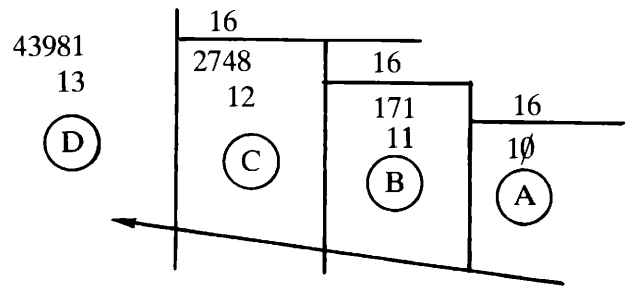
base 16 : Ø 1 2 3 4 5 6 7 8 9 A B C D E F 1Ø 11 12...

Un nombre héxadécimal tel que

&HABCD

vaut en décimal $A \times (16)^3 + B \times (16)^2 + C \times (16)^1 + D$
c'est-à-dire $(1Ø \times 4Ø96) + (11 \times 256) + (12 \times 16) + 13$
ce qui donne 43 981

Le calcul inverse est réalisé de la façon suivante



Le nombre héxadécimal est bien &HABCD. Faites plusieurs opérations de conversion pour vous y habituer.

N.B. Les octets étant des groupes de huit bits, ils sont représentés par deux chiffres héxadécimaux et leur valeur est comprise entre Ø et 255 (&HFF)

5.2.3 Le microprocesseur Motorola 68Ø9

*** Introduction**

C'est l'un des microprocesseurs 8 bits les plus récents. Le terme « 8 bits » signifie qu'il traite en parallèle des données de 8 bits (c'est-à-dire un octet). D'autres microprocesseurs sont de 16 ou 32 bits. La conception relativement récente du 68Ø9 en fait un outil puissant pour créer des ordinateurs familiaux. Nous décrivons ici sa structure interne sans donner d'explications relatives à son interfaçage avec d'autres circuits, vu qu'il n'est pas dans notre but de refaire un MO5.

*** Structure interne du 68Ø9 :**

Comme c'est le cas pour tous les microprocesseurs, le 68Ø9 contient les éléments suivants :

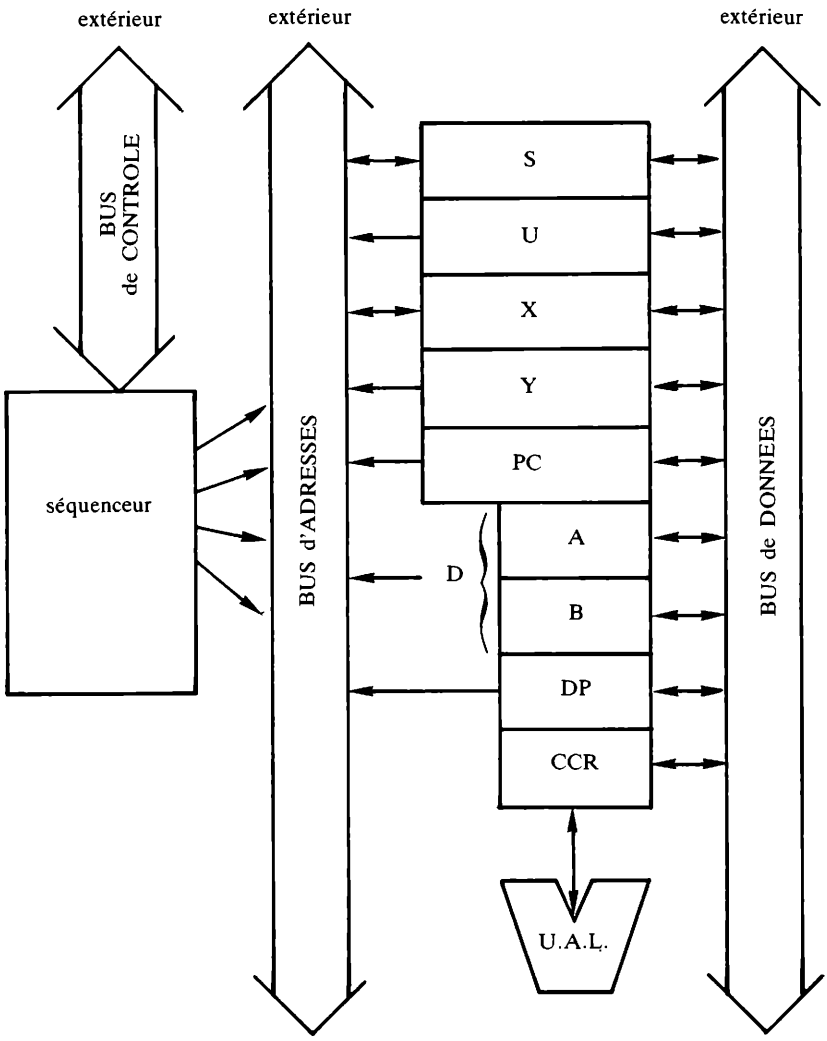
— une unité arithmétique et logique qui fait les calculs ;

— un séquenceur qui ordonne le fonctionnement du microprocesseur et traite les instructions en langage machine ;

— un ensemble de registres qui sont en fait des mémoires de 8 ou 16 bits internes au microprocesseur et d'accès très rapide ;

— des ensembles de fils appelés des BUS qui véhiculent l'information entre les éléments ;

— un ensemble de bits de condition permettant de faire des tests logiques.



La séquenceur communique avec tous les éléments pour leur donner des ordres (prendre ou sortir une donnée, faire tel ou tel calcul...) selon l'instruction en langage machine traitée. Les trois bus (adresses, données, contrôle) permettent de communiquer avec l'extérieur pour aller chercher en mémoire des données et des instructions. Examinons maintenant le rôle des différents registres du 68Ø9.

*** Les registres du 68Ø9**

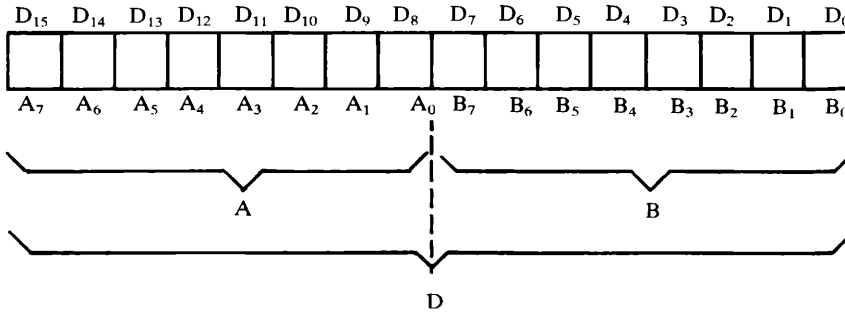
— Le pointeur de programme PC

Il contient l'adresse en mémoire de la prochaine instruction en langage machine à exécuter. Faisons exécuter celle-ci pas à pas en observant ce qui se passe. En premier lieu, le contenu du PC est placé sur le bus d'adresses. La mémoire sélectionnée dépose alors le code de l'instruction en langage machine sur le bus de données. Le 68Ø9 examine alors celui-ci pour générer des signaux de commande. On dit qu'il la décode. Selon l'instruction, tout se déroule ensuite dans le 68Ø9 où des accès à la mémoire sont réalisés, ce qui explique les différences de temps de traitement nécessaire. A chaque accès en mémoire pour aller chercher les octets correspondant au code de l'instruction en langage machine, le PC est incrémenté. De la sorte, le PC contient toujours l'adresse de l'instruction suivante lorsqu'une instruction se termine.

Nous verrons ci-après que le PC pourra être échangé avec les autres registres de 16 bits que sont S, U, X, Y, D.

— Les accumulateurs A, B, D

Ils sont généralement les entrées de l'UAL. Le 68Ø9 contient en fait deux accumulateurs de huit bits A et B ; D est en fait un registre de 16 bits constitué de A et B accolés.

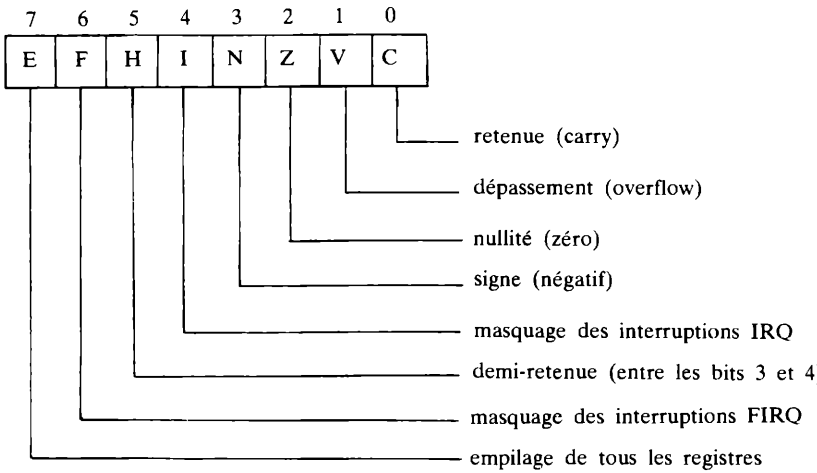


Dans les opérations arithmétiques ou logiques, un des opérandes sera généralement un des accumulateurs et l'autre le contenu d'une case mémoire ; le résultat est ensuite remis dans l'accumulateur

concerné. Toutefois, le 68Ø9 permet de réaliser des opérations sur la mémoire.

— *Le registre de condition CCR*

Il comporte 8 bits d'indicateurs d'états. Chacun des bits est utilisé pour représenter une condition particulière (retenue, égalité, débordement, signe, etc.). Ces bits sont testés lors des instructions de branchement conditionnel (BCC, BCS, BEQ, BGE, BGT, BHI,...) qui sont équivalentes en langage machine à l'instruction IF du Basic.



Ces bits sont classiques sur les microprocesseurs. Nous ne les détaillerons pas plus. Le lecteur ne connaissant pas du tout la programmation en assembleur se reportera avec intérêt à un livre spécifique.

— *Les registres 16 bits U, S, X, Y*

Le 68ØØ prédécesseur du 68Ø9 possédait un registre d'index X et un pointeur de pile S. Le 68Ø9 contient deux registres d'index X et Y et deux pointeurs de pile S et U. Ces quatre registres sont quasiment interchangeables et leurs contenus peuvent être échangés.

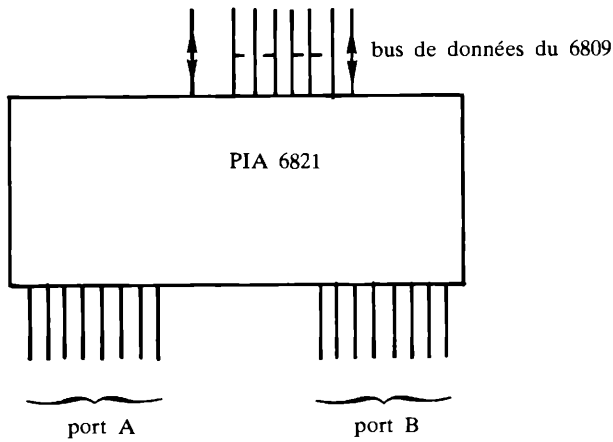
— *Le registre de page DP*

Il permet de faire de l'adressage direct (c'est-à-dire en indiquant l'adresse littérale) sur toute la mémoire (64 K octets) adressable ; à cet effet, il contient l'octet de poids fort de l'adresse. Une fois DP chargé, le mode d'adressage direct permet d'accéder à 256 octets.

Nous décrivons au paragraphe 5.3 les instructions assembleur du 68Ø9 ainsi que les divers modes d'adressage réalisables.

5.2.4 Interfaçage avec les périphériques

L'interface entre le microprocesseur 68Ø9 et les périphériques est réalisé par des circuits intégrés spéciaux : les PIA 6821. Ces boîtiers contiennent chacun deux portes de 8 bits en parallèle.



Chacun des 8 fils de chaque port peut être programmé en entrée ou en sortie et nous verrons au paragraphe 5.7 le rôle de ces bits dans le MO5.

Le MO5 contient un PIA 6821 pour le clavier, l'écran, le son et le crayon lumineux. Les contrôleurs de jeu et de communication en contiennent chacun un supplémentaire, soit trois PIA pour un « MO5 » complet (deux à la fois sont connectés : le PIA interne et l'un des contrôleurs).

5.3 PRÉSENTATION DU LANGAGE D'ASSEMBLAGE DU 68Ø9

5.3.1 Techniques de base de la programmation en assembleur

*** Introduction**

Nous donnons ci-après un certain nombre de renseignements à connaître lorsque l'on souhaite programmer en assembleur. Lorsque l'on programme en Basic, l'interpréteur traduit les instructions Basic en plusieurs instructions assembleur pour faciliter votre travail. Par exemple PRINT « A » met le code ASCII 65 (celui de la lettre A) dans la mémoire d'écran du MO5. En assembleur, c'est à vous de faire le travail. *Vous* décidez la lecture et l'écriture de données entre le microprocesseur et la mémoire (ou les périphériques), vous demandez les opérations élémentaires (+, −, ou, et, non,...). En clair, vous gérez tout. Le lecteur qui ne connaît que le Basic doit savoir les correspondances suivantes.

*** Structure d'un programme en assembleur**

Alors qu'un programme Basic est organisé selon des numéros de lignes, un programme assembleur est basé sur la notion d'adresses. Les divers octets correspondant au programme en langage machine sont rangés en mémoire et le microprocesseur vient les lire.

Exemple :

```
8E Ø3          AFF :   LDX #$3
1Ø 8E 11               LDY #$11
86 3Ø                  LDA #$3Ø
B7 2Ø 36               STA $2Ø36
86 12                  LDA #$12
B7 2Ø 2B               STA $2Ø2B
3F 1Ø                  SWI $1Ø
2Ø ED                  BRA AFF
```

Si l'on place ce programme à l'adresse $ 3ØØØ, le contenu de la mémoire est le suivant

$ 3ØØØ	8E	LDX	#$3	X ← 3
$ 3ØØ1	Ø3			
$ 3ØØ2	1Ø	LDY	#$11	Y ← 17
$ 3ØØ3	8E			
$ 3ØØ4	11			
$ 3ØØ5	86	LDA	#$3Ø	A← 48 (caractère Ø)
$ 3ØØ6	3Ø			
$ 3ØØ7	B7	STA	$2Ø36	caractère à écrire
$ 3ØØ8	2Ø			
$ 3ØØ9	36			
$ 3ØØA	86	LDA	#$12	A ← 12
$ 3ØØB	12			
$ 3ØØC	B7	STA	$2Ø2B	couleurs rouge/vert
$ 3ØØD	2Ø			
$ 3ØØE	2B			
$ 3ØØF	3F	SWI	$1Ø	appel du moniteur
$ 3Ø1Ø	1Ø			
$ 3Ø11	2Ø	BRA	AFF	branchement au début
$ 3Ø12	ED			

Il est possible de remplacer l'instruction BRA AFF par JMP $ 3ØØØ (aller à l'adresse $ 3ØØØ). En Basic, cela correspond à un GOTO n° ligne. Dans la suite de ce chapitre, nous vous indiquons un certain nombre de correspondances Basic — Assembleur.

*** Opérations arithmétiques**

Elles sont réalisées entre les accumulateurs et la mémoire, ou entre les accumulateurs. Les additions 8 et 16 bits, les soustractions 8/16 bits ainsi qu'une multiplication 8 bits sont possibles. Les instructions sont les suivantes :

nom		résultat
ADCA	adresse	A ← A + Mémoire + Carry (retenue)
ADCB	adresse	B ← B + Mémoire + C
ADDA	adresse	A ← A + Mémoire
ADDB	adresse	B ← B + Mémoire
ADDD	adresse	D ← D + (Mémoire, Mémoire + 1)
SBCA	adresse	A ← A − Mémoire − C
SBCB	adresse	B ← B − Mémoire − C
SUBA	adresse	A ← A − Mémoire
SUBB	adresse	B ← B − Mémoire
SUBD	adresse	D ← D − (Mémoire, Mémoire + 1)
MUL		D ← A * B (multiplication non signée)

Remarques :

1) nous utilisons le symbole ← pour indiquer le registre qui reçoit le résultat. Par exemple,

A ← A − Mémoire.

signifie que l'accumulateur A reçoît la différence entre A et la case mémoire indiquée.

2) La retenue indiquée (Carry) est celle présente dans le bit correspondant du registre d'états (CCR). Les instructions ADC et SBC permettent en conséquence de faire des opérations sur des groupes d'octets (16, 24, 32 bits ou plus).

3) L'accumulateur D est, rappelons-le, un groupement de A (poids fort) et de B (poids faible). Une addition/soustraction sur 16 bits modifie donc les accumulateurs A et B.

Si vous désirez maîtriser la programmation en assembleur, il est important que vous maîtrisiez bien l'arithmétique binaire, héxadécimale, DCB, ... Pour cela reportez-vous à des livres présentant en détail l'assembléur du 68Ø9 comme celui de Lance A Leventhal "68Ø9 assembly langage programming".

Deux autres instructions existent pour les opérations arithmétiques :

décrément (−1)			incrément (+1)
DECA		INCA	
DECB		INCB	
DEC	Mémoire	INC Mémoire	

Toutes ces instructions positionnent les bits du registre d'états CCR en fonction du résultat (zéro, signe, retenue, dépassement, etc...). Nous verrons comment utiliser ces bits pour faire des tests. La mémoire adressée par ces instructions peut être indiquée immédiatement (valeur), explicitement (adresse),indirectement, etc. Nous verrons au paragraphe 5.3.2 ces divers modes d'accès qui sont classiquement appelés des modes d'adressage.

*** Opérations logiques : AND, OR, NOT, EOR**

Les instructions correspondantes sont :

Nom		rôle	
ANDA	Mémoire	A ← A > Mémoire	et logique
ANDB	Mémoire	B ← B > Mémoire	
ANDCC	Mémoire	CC ← CC > Mémoire	
COMA		A ← $\bar{A}$	non (complément à 1) logique
COMB		B ← $\bar{B}$	
COM		M ← $\bar{M}$	
EORA	Mémoire	A ← A + Mémoire	ou exclusif
EORB	Mémoire	B ← B + Mémoire	
NEGA		A ← $\bar{A}$ + 1 = −A	moins (complément à 2)
NEGB		B ← $\bar{B}$ + 1 = −B	
NEG		M ← $\bar{M}$ + 1 = −M	
ORA	Mémoire	A← A > Mémoire	ou logique
ORB	Mémoire	B← B > Mémoire	
ORCC	Mémoire	CC ← CC > Mémoire	

Ces opérations se font entre le contenu d'un accumulateur (ou du registre d'états pour AND et OR) et la mémoire, et ce bit-à-bit.

OU

Bit Mémoire \ Bit accumulateur ou CCR	0	1
0	0	1
1	1	1

ET

Bit Mémoire \ Bit accumulateur ou CCR	0	1
0	0	0
1	0	1

ou exclusif

Bit Mémoire \ Bit accumulateur	0	1
0	0	1
1	1	0

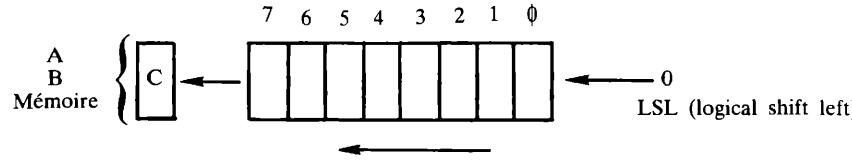
Ces opérations logiques positionnent les bits d'états. Elles rendent inutiles une comparaison avant un branchement conditionnel (cf. ci-après).

*** Décalages et rotations**

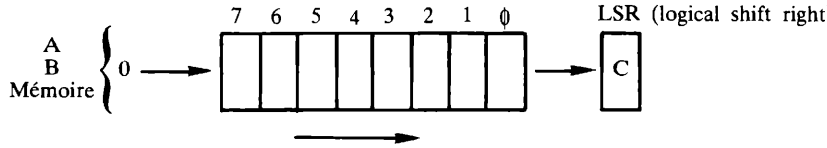
Ces décalages et rotations permettent de faire des multiplications, des divisions, des positionnements de bits (utilisées conjointement avec AND, OR, EOR), des tests de bits (via la retenue).

— *Décalages logiques*

Les instructions LSL et LSR réalisent des décalages logiques respectivement à gauche et à droite sur les accumulateurs A, B ou en mémoire.



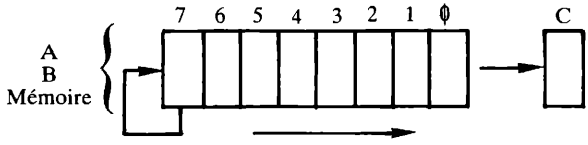
décalage logique
gauche de tous les
bits d'une position
C ← bit 7, bit 7 ← bit 6, ..., bit 1 ← bit ∅, bit ∅ ← ∅



décalage logique
droit de tous les
bits d'une position
bit 7 ← ∅, bit 6 ← bit 7, ..., bit ∅ ← bit, C ← bit ∅

— *Décalages arithmétiques*

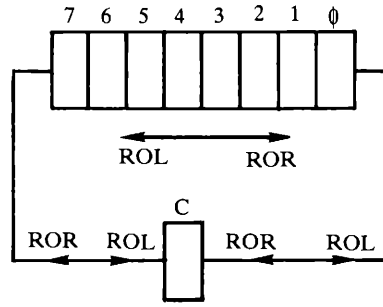
L'instruction ASR réalise un décalage droit arithmétique. Le bit 7 reste inchangé (bit de signe), les autres bits sont affectés comme par l'instruction LSR.



L'instruction ASL est identique à LSL. Un éventuel changement de signe de l'opérande décalé est positionné dans le bit de débordement (overflow) du registre d'états.

— *Rotations*

ROL et ROR réalisent une rotation des bits des accumulateurs A, B ou de la mémoire respectivement vers la gauche et la droite via le Carry (retenue) selon le schéma suivant.



* **Branchements — Tests — Sauts**

Comme en Basic, deux types de branchements existent :

- inconditionnels GOTO, JMP — BRA — LBRA
- conditionnels IF... GOTO, BCC, BCS, BEQ, BGE, BGT,
  IF... THEN, BHI, BHS, BLE, BLO, ...,
  LBCC, LBCS, ...

— Branchements inconditionnels

Les trois branchements suivants existent :

JMP adresse absolue
BRA déplacement relatif < 128
LBRA déplacement relatif ⩾ 128

Dans le premier cas, l'adresse à laquelle le microprocesseur se branche ne dépend pas de l'implantation en mémoire du programme assembleur. Dans le second cas, l'adresse change.

Exemple : A l'adresse $ 22ØØ plaçons le programme suivant :

```
861ØØØ      EX :    LDA     #1ØØØ  ←┐
971ØØ1              STA     $1ØØ1   │
OE22ØØ              JMP     $22ØØ  ─┘
```

Ce programme boucle — Plaçons le maintenant en $ 23ØØ.

```
23ØØ :  861ØØØ     EX :    LDA     #$1ØØØ ←┐
        971ØØ1              STA     $1ØØ1   │
        OE22ØØ              JMP     $22ØØ  ─┘
```

L'instruction JMP ne va plus au début du programme, mais toujours à l'adresse $22ØØ. Les instructions BRA et LBRA ne font pas un branchement à une adresse fixe, mais font un déplacement (ajout/soustraction au PC et non remplacement).

Exemple : Le même programme devient :

```
861ØØØ    EX :    LDA    #$1ØØØ
971ØØ1            STA    $1ØØ1
2Ø F8             BRA    EX
```

L'adresse de EX n'intervient plus. LBRA s'emploie lorsque le déplacement est grand (≥ 128)

— Branchement conditionnels — Comparaison — Test

Pour remplacer les instructions IF... THEN, IF... GOTO, il faut procéder en deux temps. En premier lieu, il faut positionner les bits du registre d'états CCR par une instruction de comparaison s'il y a lieu (ce n'est pas toujours la peine après une opération arithmétique, logique, ...). Ensuite, il faut tester la condition voulue à l'aide des instructions appropriées.

Comparaison :	CMP A	Mémoire
	CMP B	Mémoire
	CMP D	Mémoire, Mémoire + 1
	CMP S	Mémoire, Mémoire + 1
	CMP U	Mémoire, Mémoire + 1
	CMP X	Mémoire, Mémoire + 1
	CMP Y	Mémoire, Mémoire + 1

Cette instruction fait, en interne, la soustraction (Registre − Mémoire) pour positionner les bits du registre d'états CCR, sans modifier les registres.

ranchement :

Instruction		Condition pour branchement	
BCC, LBCC, BHS, LBHS		$\geqslant$ (C = Ø)	
BCS, LBCS, BLO, LBLO		$<$ (C = 1)	
BEQ, LBEQ		= (Z = 1)	
BGE, LBGE		$\geqslant$ Ø (N⊕V = Ø)	vrai positif / faux négatif
BGT, LBGT		$>$ Ø (Z + N⊕V = Ø)	vrai positif strict / faux négatif strict
BHI, LBHI		$>$ (Z + C = Ø)	
BLE, LBLE		$<=$ Ø (Z + (N⊕V) = 1)	vrai négatif / faux positif
BLS, LBLS		$\leqslant$ (Z + C = 1)	
BLT, LBLT		$<$ Ø (N⊕Z = 1)	vrai négatif strict / faux positif strict
BMI, LBMI		$<$ Ø (N = 1)	vrai / faux } négatif
BNE, LBNE		$\neq$ (Z = Ø)	non nul
BPL, LBPL		$\geqslant$ Ø (N = Ø)	vrai / faux } positif
BVC, LBVC	pas d'overflow	(V = Ø)	pas de débordement
BVS, LBVS	overflow	(V = 1)	débordement

Tous ces branchements sont intimement liés à l'arithmétique binaire. Reportez-vous à un livre spécialisé sur le 68Ø9 si vous ne la possédez pas. Comme pour les branchements inconditionnels, la lettre L désigne un branchement long.

Remarques :

1) BGE, BGT, BLE, BLT
BMI, BPL, BVC, BVS
travaillent sur des nombres signés en complément à deux (débordement pris en compte)

{BHS, BHI, BLS, {BLO
{BCC {BCS
travaillent sur des nombres non signés (débordement non considéré)

2) Nous appelons des nombres
vrais positifs } lorsque le signe n'est pas entaché
vrais négatifs } d'une erreur due à un débordement.

Nous verrons au paragraphe 5.3.3 (Notion de pile — Sous-programmes) les autres branchements et sauts existants.

*** Chargement, sauvegarde des registres en mémoire**

Les instructions LD (A, B, D, S, U, X, Y), LEA (S, U, X, Y), ST (A, B, D, S, U, X, Y) permettent respectivement de charger un registre avec la valeur d'une case mémoire, l'adresse d'une case mémoire et de stocker la valeur d'un registre en mémoire. L'instruction EXG permet d'échanger le contenu de 2 registres 8 ou 16 bits (EXG R1, R2).

Ces instructions sont d'emploi très fréquent dans les programmes en assembleur qui sont en pratique une suite de

{ chargement
{ calcul
{ stockage

5.3.2 Modes d'adressage du 68Ø9

Dans les descriptions des instructions faites précédemment, nous avons indiqué un opérande appelé « Mémoire ». Nous allons voir ci-après ce qui se cache derrière ce terme et comment on accède à la donnée. La liste des instructions fournie au paragraphe 5.3.4 précisera quels modes d'adressage sont utilisables pour chaque instruction.

*** Adressage immédiat**

Dans ce mode d'adressage, la valeur à traiter suit immédiatement le code de l'instruction. Il est possible de charger un registre 8 ou 16 bits, mais pas une adresse mémoire directement.

Exemple :

LDA #$Ø3

adressage immédiat — valeur héxadécimale

On peut charger A, B, D, U, S, X, Y par une valeur immédiate. Il est possible d'agiter, de soustraire, ... des données immédiates à des registres.

*** Adressage implicite ou inhérent**

Tout se passe dans ce mode à l'intérieur du 68Ø9 : complémentation, échange de registre... Ces instructions sont les plus rapides à exécuter et leur code opération tient sur un ou deux octets.

Exemple : L'instruction EXG R1, R2

Le premier octet vaut &H1E
Le second octet dépend des registres concernés

R1	R2

Les codes 4 bits pour les registres sont :

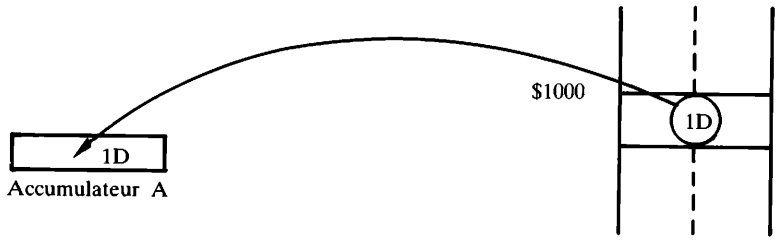
A	&H8	S &H4
B	&H9	U &H3
D	&HØ	X &H1
		Y &H2
DP	&HB	
CCR	&HA	PC&H5

EXG A, B a pour code 1E89

*** Adressage direct étendu**

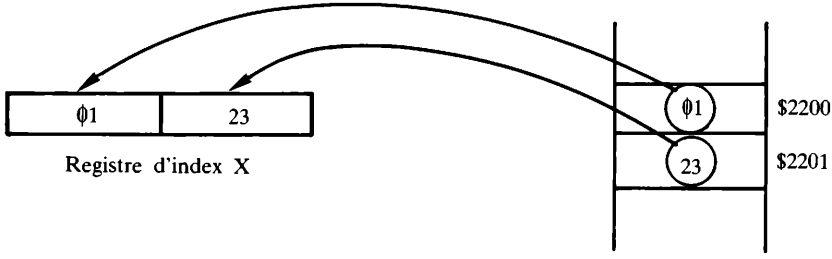
Dans ce mode, la valeur à traiter se trouve dans une case mémoire dont on indique l'adresse.

Exemple 1 : LDA $1ØØØ



Si la case mémoire située à l'adresse $1ØØØ contient la valeur $1D, l'accumulateur A aura cette même valeur après l'exécution de l'instruction.

Exemple 2 : LDX $22ØØ



L'adresse $22ØØ contient le futur octet de poids forts de X, l'adresse $22Ø1 contient les futurs poids faibles.

L'adressage absolu est assez long et occupe de la place (2 octets) dans le code de l'instruction. Dans le cas où l'on a plusieurs accès à réaliser dans une zone mémoire de taille limitée, l'adressage direct paginé est alors meilleur que l'adressage absolu.

*** Adressage direct paginé**

Ce mode d'adressage diffère du précédent par le fait que le poids fort de l'adresse est présent dans le registre de page DP. Seul le poids faible est à préciser dans l'instruction. On peut ainsi accéder à une page mémoire de 256 octets.

Exemple : Chargons DP		LDA	#$2Ø
		EXG	A, DP
	Accès en mémoire	LDB	$Ø1
		LDU	$1E
		LDX	$Ø5

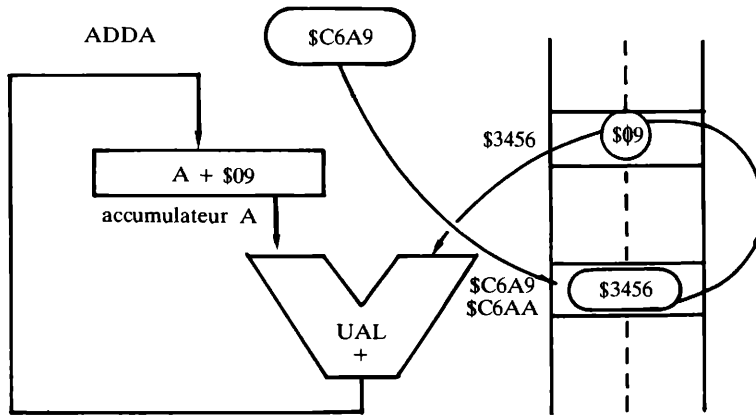
Après cette séquence

B aura comme valeur le contenu de l'adresse $2ØØ1
U des adresses $2Ø1E, $2Ø1F
X des adresses $2ØØ5, $2ØØ6

Remarque : le registre DP n'est chargeable que par un échange avec l'un des accumulateurs A ou B. Par compatibilité logicielle avec le microprocesseur 68ØØ, le 68Ø9 met à Ø le registre DP lors de son initialisation.

*** Adressage indirect étendu**

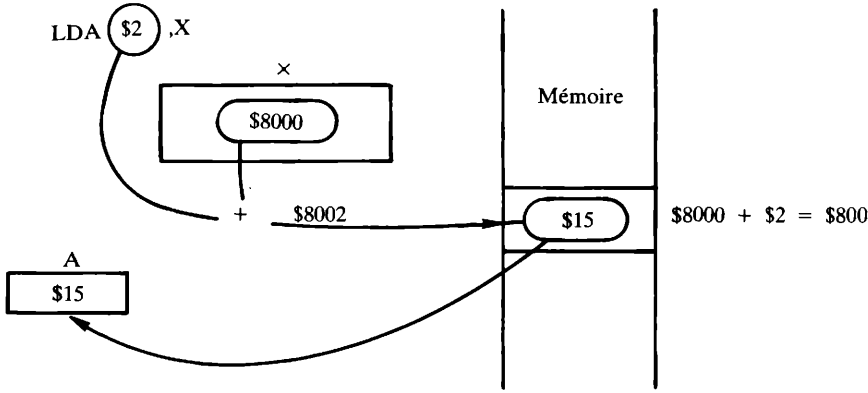
Dans ce mode, l'adresse de la donnée à charger est à une adresse donnée. Dans l'instruction, il faut préciser une adresse sur 16 bits. L'adressage indirect est indiqué par des crochets.



***Adressages indéxés (ou basés)**

Dans ces modes d'adressage, l'un des registres 16 bits que sont les registres X, Y, U, S, PC est utilisé comme index (ou base) pour le calcul de l'adresse où se trouve la valeur à prendre en compte.

Exemple :



Le format général est le suivant :

Mnémonique d, Base

déplacement par rapport à la base.

U, S, X, Y

L'adressage indexé permet par exemple d'accéder aux éléments d'un tableau ou d'une liste. Selon la valeur du déplacement, la structure du code opération varie. Le cas du registre PC comme base étant spécial, nous le verrons séparément.

— *Déplacement nul*

Dans ce cas, la valeur traitée est à l'adresse indiquée par la base.

Exemple : LDA, U

— *Déplacement constant*

Dans ce cas, le déplacement est codé dans le code opération sur 5, 8, ou 16 bits

5 bits : − 16 à + 15 par rapport à la base
8 bits : −128 à +127 par rapport à la base
16 bits : − 32 768 à + 32 767 par rapport à la base

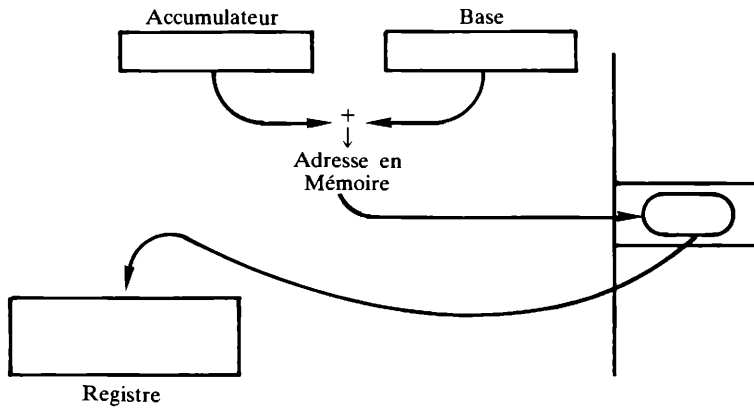
Le code opération tient alors sur 2, 3 ou 4 octets.

— *Déplacement accumulateur*

Le déplacement à ajouter à la base n'est plus fixe, mais inclu dans l'un quelconque des accumulateurs A, B, D.

Exemples :

```
LDB     A,X
LDU     B,Y
LDD     D,U
```



— *Auto — incrémention/décrémentation*

Lors du traitement de tableaux, de chaînes de caractères ou de listes, il est souvent nécessaire de considérer des octets rangés consécutivement en mémoire. Par exemple, pour envoyer le message « Guide du MO5 — Edimicro » sur l'écran, il faut traiter les caractères un par un à la suite (", G, u, i, d, e ...). Il en est de même pour faire une moyenne de nombres, etc...

Le 68Ø9 offre la possibilité d'auto-incrémenter/décrémenter d'une valeur de 1 ou de 2 la base utilisée avant ou après son emploi. La notation est la suivante.

, − − R	décrémentation de 2	avant emploi
, − R	1	avant emploi
, + + R	incrémentation de 2	avant emploi
, + R	1	avant emploi
, R − −	décrémentation de 2	après emploi
, R −	1	après emploi
, R + +	incrémentation de 2	après emploi
, R +	1	après emploi

Lorsque l'incrément/décrément vaut 1, cela permet de travailler sur des tableaux d'octet ; lorsqu'il vaut 2, la prise en compte est faite sur des tableaux de 16 bits (adresses, données...). La possibilité de gérer des « piles » logicielles (cf $ 5.3.3) est ainsi offerte au programmeur 6 8Ø9. Nous verrons dans la suite de ce paragraphe comment utiliser des tableaux d'adresse pour programmer de façon efficace.

Exemples : ADDB , Y + a pour effets successifs

1) B ← la valeur de la case mémoire pointée par Y
2) Y ← Y + 1

Par contre ADDB , + Y provoque

1) Y ← Y + 1
2) B ← la valeur de la case mémoire pointée par le nouveau contenu de Y

Ces efforts de pré — et post— incrémentation/décrémentation se retrouvent dans des langages évolués comme le C.

Exemple : Voici un programme de copie d'une zone de 11 octets de l'adresse $3 ØØØ à adresse $2ØØ0 en permutant les termes (début ⟷ fin)

```
        LDB     # $ØA
DEB :   LDX     # $3 ØØØ
        LDY     # $2ØØB
COP :   LDA     , X +
        STA     , - Y
        DECB
        BNE     COP
FIN :
```

Les valeurs successives sont prises aux adresses $3ØØØ, $3ØØ1, ..., $3ØØA et rangées aux adresses $2ØØA, $2ØØ9, ..., $2ØØØ. Essayez de voir la différence entre pré- et post- in/décrémentation en faisant se dérouler le programme à la main.

— **Utilisation du PC comme base :**

Ce mode d'adressage permet de créer des programmes ne dépendant pas de l'adresse mémoire à laquelle il est implanté. Le déplacement par rapport au PC est codé sur 8 ou 16 bits (plus de mode 5 bit).

Exemple :

ADDA POS, PCR

Le symbole PCR signale au programme assembleur (qui traduit les mnémoniques en code machine) que le déplacement est relatif par rapport au PC. Attention de ne pas modifier ensuite le code machine en ajoutant/supprimant une instruction ! Nous vous conseillons de n'utiliser ce mode que lorsque vous maîtriserez bien le langage machine.

*** Adressage indéxé indirect**

Ce mode combine un adressage indexé suivi d'un adressage direct.

Exemple : LDA [4, X]

L'accumulateur A reçoît la donnée dont l'adresse est placée aux adresses X + 4, X + 5.

L'adressage indexé indirect permet de gérer des tableaux

d'adresses. Il est possible d'utiliser un déplacement fixe, un déplacement dans les accumulateurs, un adressage auto in/décrémenté de ± 2 (on travaille sur des adresses). Nous donnons ci-dessous un exemple d'utilisation de l'adressage indexé indirect avec déplacement dans un des accumulateurs.

Supposons que nous ayions trois ports d'entrée-sortie gérables de la même façon (avec des périphériques de même type). Il serait inutile de faire trois programmes identiques au numéro de port près. Créeons deux tableaux contenant les adresses des ports de données :

PIAA :	@ PIA1 A	adresse du port A du PIA1
	@ PIA2 A	2
	@ PIA3 A	3
PIAB :	@ PIA1 B	adresse du port B du PIA1
	@ PIA2 B	2
	@ PIA3 B	3

Chargeons dans X l'adresse du tableau PIAA (LEAX PIAA)
Y PIAB (LEAY PIAB)
B le numéro du périphérique (1, 2,3)

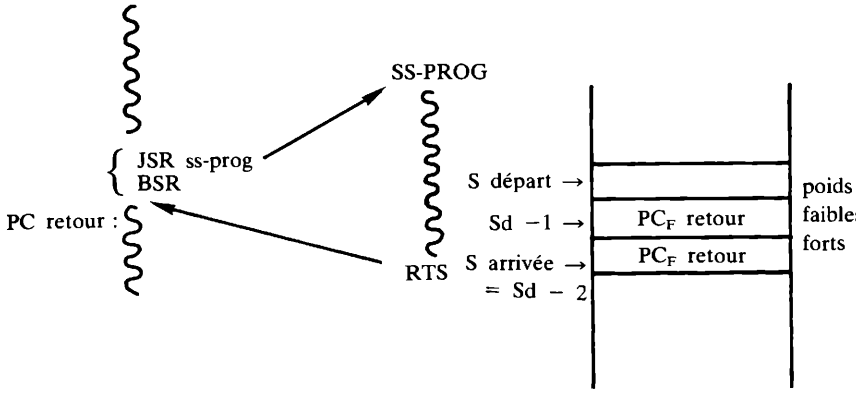
L'accès au port B du périphérique courant est réalisé par l'instruction

LDA [B, Y]

5.3.3 Notion de pile — Sous-programmes

*** Introduction**

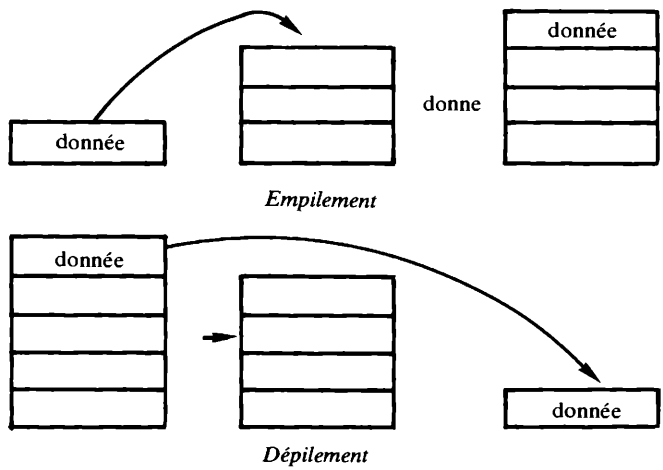
La création de sous-programmes est utile comme en Basic. Aux instructions GOSUB et RETURN correspondent les instructions JSR (saut), BSR (branchement court), LBSR (branchement long), et RTS (retour). Les instructions d'appel de sous-programme placent l'adresse de l'instruction qui suit l'appel par JSR, BSR dans la pile matérielle, c'est-à-dire aux adresses (S−2), (S−1). L'instruction RTS reprend ces valeurs pour redonner la main au programme principal.



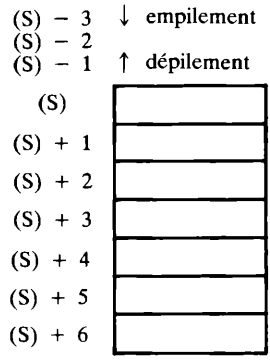
La seule différence entre JMP et JSR, BRA et BSR réside dans le fait que JSR, BSR se souviennent de l'adresse de retour dans le programme principal (en la mettant à l'adresse pointée par S et en ôtant 2 à S).

*** Notion de pile**

Une pile est une structure de données dans laquelle la dernière donnée introduite est la première sortie (LIFO last In Erst out). Son fontionnement est le même que celui d'une pile d'assiettes.



Pour gérer une pile, on utilise un pointeur qui contient l'adresse en mémoire du dernier élément introduit dans la pile, les autres étant placés consécutivement. Le 68Ø9 offre deux registres de pile S et U qui peuvent être utilisés aussi comme registres d'index.



Pointeur de pile

S est employé comme pointeur de pile matériel (hardware, système...), U est employé comme pointeur de pile utilisateur. Lors d'un emplilement, S (ou U) est décrémenté du nombre d'octets empilés, lors d'un dépilement, S (ou U) est incrémenté du nombre d'octets dépilés.

*** Instructions de manipulation de la pile**

Le 68Ø9 fournit deux instructions de manipulation d'une pile :
PSH (PSHS, PSHU) empilement
PUL (PULS, PULU) dépilement
Tous les registres peuvent être empilés/dépilés séparément.

Exemple : PSHS A, B empile les contenus de A et B sur la pile S.
PULU X, Y dépile les contenus de X et Y de la pile U.

Attention : L'ordre d'empilement des registres indiqués n'est pas toujours celui qui est indiqué dans l'instruction. Il est fixé matériellement selon l'ordre suivant :

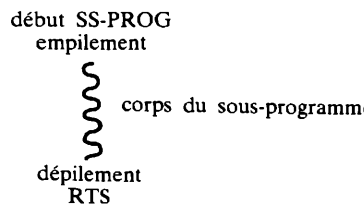
dernier (12) octet empilé	↑	CCR
		A
		B
		DP
		X (Forts)
		X (faibles)
		Y (Forts)
		Y (faibles)
		U/S (Forts)
		U/S (faibles)
1e octet empilé		PC (Forts)
		PC (faibles)

Il est important de bien se rappeler cet ordre pour utiliser efficacement une pile. Celles-ci servent à conserver un contexte (registres d'un programme), à stocker des données locales, des paramètres d'appel et de reour, à créer des sous-programmes réentrants (c'est-à-dire appelables simultanément par 2 taches temps réel par exemple qui prennent le contrôle du 68Ø9 à tour de rôle selon les évènements). Savoir utiliser une pile est très important pour bien programmer en assembleur.

*** Utilisation d'une pile**

— *Sauvegarde d'un contexte*

Il est fréquent qu'un sous-programme doive utiliser les mêmes registres que le programme appelant sans que l'on veuille perdre leurs contenus. Il suffit alors de les empiler au début du sous-programme et de les dépiler à la sortie.



— *Passage de paramètres*

Il existe trois manières principales de passer des paramètres entre un programme et un sous-programme :

- dans les registres si les paramètres ne sont pas nombreux,
- en mémoire,
- dans une pile.

Le passage dans une pile est le mode le plus souple. Il permet de créer des programmes qui ne dépendent pas de leur implantation en mémoire :

structure type	PUSH	{	arguments
	BSR		ssprog
	POP		paramètres de retour.

— *Variables locales*

Une pile peut contenir les variables locales (ou internes) d'un sous-programme. Le pointeur est en général utilisé alors comme registres d'index.

5.3.4 Liste des instructions du 68Ø9

Nous donnons ci-dessous la liste des instructions du 68Ø9 en précisant pour chacune d'entre elles :

- leurs mnémoniques et formats
- les modes d'adressage utilisables
- leurs modifications éventuelles du registre d'états

La liste est faite par ordre alphabétique.

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
ABX X←X+B	ABX	—	—
ADC addition avec retenue	ADCA Mémoire ADCB Mémoire	immédiat, direct, étendu, indexé, indirect, indexé, indirect	H,N,V,C,Z "
ADD addition simple	ADDA Mémoire ADDB Mémoire ADDD Mémoire	immédiat, direct, étendu, indexé, indirect, indexé, indirect	H,N,V,C,Z H,N,V,C,Z N,V,C,Z
AND et logique	ANDA Mémoire ANDB Mémoire ANDCC Mémoire	immédiat, direct, étendu, indexé, indirect, indexé indirect- immédiat	H,N,V,C,Z Tous
ASL décalage gauche arithmétique	ASLA ASLB Mémoire ASL	implicite (registres) direct, étendu, indexé, indirect, indexé indirect	N,V,C,Z
ASR décalage droit arithmétique	ASRA ASRB ASR Mémoire	implicite (registres) direct, étendu, indexé, indirect indexé indirect	N,C,Z
BCC branchement si C= 0 (≥ non signé)	BCC déplacement LBCC déplacement	relatif court relatif long	— —
BCS branchement si C = 1 (< non signé)	BCS déplacement LBCS déplacement	relatif court relatif long	— —
BEQ résultat nul (Z = 1)	BEQ déplacement LBEQ déplacement	relatif court relatif long	— —
BGE ≥ 0 (complément à 2)	BGE déplacement LBGE déplacement	relatif court relatif long	— —
BGT > 0 (complément à 2)	BGT déplacement LBGT déplacement	relatif court relatif long	— —
BHI > 0 (non signé)	BHI déplacement LBHI déplacement	relatif court relatif long	— —
BHS même instruction que BCC	BHS déplacement LBHS déplacement	relatif court relatif long	— —

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
BIT et logique entre l'accumulateur concerné et la mémoire sans toucher à l'accumulateur (bits d'états positionnés)	BITA Mémoire BITB Mémoire	immédiat, direct, indirect, indexé, indexé indirect	N,Z,V
BLE ⩽ 0 (complément à 2)	BLE déplacement LBLE déplacement	relatif court relatif long	— —
BLO même instruction que BCS	BLO déplacement LBLO déplacement	relatif court relatif long	— —
BLS ⩽ 0 (non signé)	BLS déplacement LBLS déplacement	relatif court relatif long	— —
BLT < 0 (complément à 2)	BLT déplacement LBLT déplacement	relatif court relatif long	— —
BMI < 0 (N = 1) pas de test de débordement	BMI déplacement LBMI déplacement	relatif court relatif long	— —
BNE ≠ 0 (Z = 0)	BNE déplacement LBNE déplacement	relatif court relatif long	— —
BPL ⩾ 0 (N = 0)	BPL déplacement LBPL déplacement	relatif court relatif long	— —
BRA branchement inconditionnel	BRA déplacement LBRA déplacement	relatif court relatif long	— —
BRN instruction nulle (~ NOP)	BRN déplacement LBRN déplacement	relatif court relatif long	— —
BSR appel de sous-programme	BSR déplacement LBSR déplacement	relatif court relatif long	— —
BVC pas de débordement	BVC déplacement LBVC déplacement	relatif court relatif long	— —
BVS débordement	BVS déplacement LBVS déplacement	relatif court relatif long	— —

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
CLR Mise à zéro	CLRA CLRB CLR Mémoire	registres (implicite) directs, indexé, indirect, indexé indirect	N,Z,V,C
CMP Comparaison de deux opérandes	CMPA Mémoire CMPB Mémoire CMPD Mémoire CMPS Mémoire CMPU Mémoire CMPX Mémoire CMPY Mémoire	immédiat, direct étendu, direct paginé, indirect, indexé, indexé indirect	N,Z,V,C
COM Complément à 1	COMA COMB COM Mémoire	registres (implicite)	
CWAI attente d'interruption	CWAI valeur	immédiat	C
DAA Ajustement décimal de l'accumulateur A (calcul BCD)	DAA	Registres (implicite)	N,Z,V,C
DEC Décrément de 1	DECA DECB DEC Mémoire	Registres (implite) directs, indexé, indirect, indexé indirect	N,Z,V
EOR ou exclusif	EORA Mémoire EORB Mémoire	immédiat, directs, indirect, indexé indexé indirect	N,Z, V (V ← 0)
EXG échange de registres	EXG R_1, R_2	immédiat (code de registres)	—
INC incrémentation (+ 1)	INCA INCB INC Mémoire	registres (implite) directs, indirect, indexé, indexé indirect	N,Z,V
JMP saut	JMP adresse	directs, indirect, indexé, indexé indirect	—
JSR appel de sous-programme	JSR adresse	directs, indirect, indexé, indexé indirect	—

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
LD Chargement d'un registre par une valeur	LDA Mémoire LDB Mémoire LDD Mémoire LDS Mémoire LDU Mémoire LDX Mémoire LDY Mémoire	immédiat, direct étendu, direct paginé, indirect, indexé, indexé indirect	N,Z V (V←0)
LEA Chargement d'un registre 16 bits par une adresse	LEAS adresse LEAU adresse LEAX adresse LEAY adresse	indexé, indexé indirect	—
LSL décalage logique d'une position vers la gauche	LSLA LSLB LSL Mémoire	registres (implicite) directs, indirect, indexé, indexé indirect	N,Z,V,C,
LSR décalage logique d'une position vers la droite	LSRA LSRB LSR Mémoire		N,Z,V,C
MUL Multiplication 8 bits non signée D ← A*B	MUL	registres (implicite)	Z,C,
NEG Complément à 2	NEGA NEGB NEG Mémoire	registres (implicite) directs, indirect, indexé, indexé indirect	Z,N,V,C
NOP ne fait rien (attente)	NOP	—	—
OR ou logique	ORA Mémoire ORB Mémoire ORCC Mémoire	immédiat, directs, indirect, indexé, indexé indirect immédiat	N,Z,C Tous
PSH empilement	PSHU liste de registres S		—
PUL dépilement	PULU liste de registres S		Tous si CCR est dépilé
ROL Rotation à gauche d'une position	ROLA ROLB ROL Mémoire	Registres (implicite)	

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
ROR rotation à droite d'une position	RORA RORB ROR Mémoire	Registres (implicite)	
RTI Retour d'interruption	RTI		—
RTS retour de sous-programme	RTS		—
SBC soustraction au contenu d'un accumulateur d'une valeur et de la retenue	SBCA Mémoire SBCB Mémoire	immédiat, directs, indirect, indexé, indirect indexé	N,Z,V,C
SEX Transformation d'un nombre 8 bits en complément de 2 contenu dans B en un nombre 16 bits contenu dans D	SEX	registres (implicite)	N,Z
ST sauvegarde en mémoire du contenu des registres	STA Mémoire STB Mémoire STD Mémoire STU Mémoire STS Mémoire STX Mémoire STY Mémoire	directs, indirect, indexé, indexé indirect	N,Z V (V←0)
SUB soustraction simple Accumulateur Mémoire	SUBA SUBB SUB Mémoire	directs, indirect, indexé, indexe indirect	N,Z,V,C
SWI interruption logicielle	SWI SWI2 SWI3	$FFFA-FFFB $FFF4-FFF5 $FFF2-FFF3	vecteur implicite
SYNC Synchronisation avec interruptions	SYNC	—	—

Mnémonique et description	Format assembleur	Modes d'adressages	Registre d'états
TFR Transfert R2 ← R1	TFR R1, R2	immédiat (code de registre)	—
TST	TSTA TSTB TST Mémoire		N,Z, V (V←0)

5.3.5 Mise au point d'un programme assembleur

Pour programmer en assembleur, deux solutions sont possibles :

- coder les instructions assembleur en mémoire (En Basic, grâce aux instructions DATA, READ, POKE) ;
- utiliser un outil d'aide à la mise au point.

Une cartouche « assembleur » réalisée par la firme Microsoft est disponible dans le commerce (ou le sera prochainement). Elle comprend un éditeur plein écran pour le programme assembleur en clair, un assembleur qui traduit les mnémoniques en code machine et un débogeur qui permet de suivre l'exécution du programme. Nous vous conseillons d'utiliser un tel outil de développement.

Si vous êtes débutant, ne commencez pas tout de suite à faire de gros programmes en assembleur. Nous vous conseillons de suivre les étapes suivantes :

- comprendre les instructions et la structure du 68Ø9
- faire de petits exercices
- étudier les primitives (sous-programmes du moniteur) disponibles (il est inutile de reprogrammer ce qui existe)
- faire des sous-programmes assembleurs accessibles en Basic
- explorer le moniteur système du MO5 pour découvrir de nombreux trucs de programmation.

Remarque : 1) Avant de décider quelle partie d'un programme doit être écrite en assembleur plutôt qu'en Basic, faites tourner le programme complet en Basic. Cela permet d'être sûr de son fonctionnement.

2) Il est tout à fait possible de programmer de façon structurée en assembleur. Ne l'oubliez pas ! Sinon, vos programmes deviendront vite illisibles.

5.4 ACCÈS A PARTIR DU BASIC

L'instruction EXEC adresse permet d'appeler le sous-programme assembleur qui se trouve à l'adresse indiquée. Elle joue

le même rôle que les instructions JSR et BSR. Le retour au programme Basic sera donc réalisé par l'instruction RTS.

Exemple type :

```
1Ø REM APPEL TYPE SSPROG ASSEMBLEUR
2Ø CLEAR, Adresse Assembleur - 1
                  §
                  §
1111 EXEC adresse Assembleur
                  §
```

Attention ! Pour prévenir le Basic de ne pas utiliser la zone mémoire occupée par le (ou les) sous-programme(s) assembleur(s), il faut se servir de l'instruction BASIC intitulée CLEAR dont le second paramètre indique l'adresse mémoire accessible par le Basic.

Il existe deux façons de charger le programme assembleur en mémoire :

- insérer les codes hexadécimaux dans des instructions DATA et les recopier en mémoire à l'adresse prévue.
- utiliser les instructions SAVEM/LOADM qui réalisent une sauvegarde/lecture sur cassette d'une zone mémoire.

Avant de donner des exemples d'emploi de sous-programmes assembleurs dans un programme, voici des informations très utiles sur le fonctionnement du MO5 (carte mémoire, variables systèmes, ...). Elles n'ont pas la prétention d'être exhaustives, car faute de renseignements précis sur la ROM du MO5, nous avons dû procéder par tâtonnements pour découvrir petit à petit les trésors cachés du MO5. D'autres informations relatives au graphisme ont été décrites au chapitre 3.

5.5 ORGANISATION DE LA MÉMOIRE DU MO5

La cartographie mémoire est la suivante (Les adresses sont données en héxadécimal) :

ØØØØ	—	1F3F	Mémoire écran (2 fois 8ØØØ octets)
1F4Ø	—	1FFF	non utilisé.
2ØØØ	—	21FF	variables système
22ØØ	—	9FFF	Mémoire utilisateur (31,5K octets)
AØØØ	—	A7BF	Réservés au contrôleur de disquettes
A7CØ	—	A7C3	Ports d'entrée/sortie (voir plus loin)
A7C4	—	A7CB	non utilisé
A7CC	—	A7CF	Ports d'entrée/sortie (voir plus loin)
A7DØ	—	A7DF	Contrôleur de disquettes
A7EØ	—	A7E3	Ports d'entrée/sortie (voir plus loin)
A7E4	—	A7FF	Entrées sortie diverses
A8ØØ	—	AFFF	non utilisé (2K octets)
BØØØ	—	FFFF	Cartouche ROM, moniteur et BASIC (20K octets)

Les places non utilisées sont libres pour l'utilisateur (à priori). On constate que le CPU 68Ø9 du MO5 adresse la totalité de son champ d'adressage (64 k octets). En fait 8 K octets supplémentaires sont adressés pour la mémoire d'écran par l'intermédiaire du port d'un des PIA.

5.6 LES VARIABLES SYSTÈMES

Ces variables en mémoire permettent d'agir sur le fonctionnement du MO5. Il est ainsi possible de supprimer le bip sonore du clavier, d'enlever l'auto-repeat, etc. en Basic (instruction POKE) ou en assembleur.

Nous donnons ici la liste des adresses hexadécimale des variables systèmes utiles que nous avons repérées.

&H2Ø19 : Tous les bits de cette variable sont utilisés. Les principaux sont les suivants :
bit 7 : Ø= majuscule ; 1 = minuscule
bit 3 : Ø = bip sonore à l'enfoncement des touches
1 = supression du bip sonore.

&H2Ø1B : ligne
&H2Ø1C : colonne } du curseur

&H2Ø1E :
&H2Ø1F : } lignes délimitrices de la fenêtre

&H2Ø2Ø :
&H2Ø21 : } colonnes délimitrices de la fenêtre

&H2Ø29 : couleur du caractère à afficher

&H2Ø2B : Couleurs de forme et de fond courantes. Cet octet est codé de la même façon que les octets de la page mémoire de couleur écran : b Ø à b 3 pour le fond et b 4 à b 7 pour la forme (cf. chapitre 3).

&H2Ø2C : Si cette variable est à Ø, on est en mode défilement par pages. Si elle vaut &HFF, on est en mode SCROLL

&H2Ø2D : Lorsque l'on est en mode SCROLL, le SCROLL est normal si cette variable est à Ø et lent si elle vaut &HFF.

&H2Ø36 : code ASCII du caractère à afficher.

&H2Ø37 : donne le code de la dernière touche enfoncée au clavier

&H2Ø61 – 2Ø62 : vecteur d'interruption IRQ

&H2Ø7Ø – 2Ø71 : contient l'adresse de début du générateur de caractères en cours d'utilisation.

&H2Ø73 – 2Ø74 : contient l'adresse du générateur de caractères standard.

&H2Ø76 : cette variable contrôle le temps qui s'écoule entre la première pression d'une touche et le début de l'auto-repeat de cette même touche. La valeur initiale est 7. Si vous prenez une valeur très faible (par exemple 1 ou Ø), le clavier n'a plus d'anti-rebond mais ceci est très intéressant dans les jeux d'action et de réflexe. Si la valeur est trop forte, il n'y aura plus d'auto-repeat.

5.7 ENTRÉES-SORTIES

Nous détaillons ici le rôle des différents PIA du MO5 et l'adresse de ces boîtiers. Le port A est à une adresse paire, le port B à l'adresse du port A + 1.

PIA1 : clavier, écran, crayon, optique, son, cassette

— Port A : (&HA7CØ)
bit Ø : 1 = sélection page mémoire écran caractères
Ø = sélection page mémoire écran couleurs
bit 1 : rouge
bit 2 : vert
bit 3 : bleu
} couleur du cadre (voir code des couleurs)
bit 4 : clair
bit 5 : associé au crayon optique
bit 6 : écriture cassette
bit 7 : lecture cassette

— Port B : (&HA7C1)
bit Ø : sortie son
bit 1-7 : utilisés pour le clavier

PIA2 : manettes de jeu
Port A : (&HA7CC) lecture des manettes de jeu
Port B : inutilisé

PIA3 :
adresse &HA7EØ

Le rôle de ce PIA est la gestion des entrées/sorties parallèles (contrôleur de communication).

Remarque : Pour utiliser ces ports d'entrée-sortie, il faut auparavant les programmer. Le lecteur intéressé se reportera à la notice technique du PIA 6 821 du catalogue de composants de Motorola. Dans le cas où il s'agit de faire un accès classique au périphériques, il est préférable d'utiliser les routines appropriées du moniteur, plutôt que de les reprogrammer ce qui perturberait le fonctionnement de votre MO5. Il est possible grâce à l'instruction SWI de faire des appels de sous-programmes du moniteur tels que :

— la sélection des pages mémoire d'écran couleur/fond-forme,

— l'envoi d'un bip sonore,

— l'affichage de caractères,

— la lecture du clavier,

— la coloration d'un point,

— le positionnement du curseur, des couleurs de fond, forme et cadre,

— la lecture des manettes de jeu.

Ces sous-programmes utilisent les variables systèmes présentées ci-avant. Voici leur description (numéro, paramètres...). Ils sont très utiles en assembleur.

5.8 PRIMITIVES DU MONITEUR MO5

5.8.1 L'instruction assembleur SWI

SWI permet de simuler une interruption (logicielle). Les interruptions sont des événements matériels qui stoppent le programme en cours pour provoquer un débranchement vers un sous-programme spécial de traitement (par exemple pour lire un périphérique).

SWI a pour effet d'incrémenter le PC, d'empiler tous les registres (exceptés S) dans la pile S. Le contrôle est alors passé au sous-programme dont l'adresse est située en $FFFA, $FFFB et les interruptions matérielles sont bloquées. La fin du « programme » de traitement est marquée par l'instruction RTI qui réautorise les interruptions, dépile les registres et rend la main au programme interrompu.

SWI peut être employée pour une grande variété de fonctions.

En particulier comme point d'entrée d'un débogeur, d'un système d'exploitation de disquettes ou d'un groupe de primitives d'un moniteur système. Pour appeler une primitive sur le MO5, il faut initialiser tout d'abord les registres contenant les arguments d'appel, puis exécuter l'instruction :

SWI NBRP où NBRP est le numéro de la primitive.

Nous allons ci-dessous vous donner les numéros et les arguments des primitives que nous connaissons.

5.8.2 Gestion de l'écran

SWI&HØ2 : écriture d'un caractère à l'écran

Le caractère à écrire doit se trouver dans B (code ASCII) (B = registre B du 68Ø9)

Remarque : Selon le type de caractères à afficher, il faut appeler une ou plusieurs fois cette primitive (voir paragraphe 5.9)

SWI &HØ4 : sélection de la page mémoire d'écran couleur

SWI &HØ6 : sélection de la page mémoire d'écran forme et fonde coordonnées X et Y.

SWI &HØE : Tracé d'une droite «graphique»

en entrée : X et Y fournissent les coordonnées (Ø à 319, Ø à 199) du point d'arrivée de la droite
&H2Ø36 = Ø

en sortie : —

Cette primitive trace une droite graphique entre le précédent point allumé et le point de coordonnées X et Y.

SWI &HØE : Tracé d'une droite «caractère»

en entrée : X et Y fournissent les coordonnées (Ø à 39, Ø à 24) du caractère d'arrivée de la droite.
&H2Ø36 = code ASCII du caractère
&H2Ø2B = couleurs forme et fond

en sortie : —

Cette primitive trace une droite caractère entre le précédent point allumé et le point de coordonnées caractères X et Y.

Remarque : SWI &HØE correspond à l'instruction LINE du Basic.

SWI &H1Ø : Coloration d'un point

en entrée : X contient la coordonnée horizontale du point (entre Ø et 319).

Y contient la coordonnée verticale du point (entre Ø et 199). &H2 Ø29 contient la couleur du point (entre 15 et– 16 selon les codes déjà donnés au chapitre concernant le graphisme). &H2 Ø36 = Ø

en sortie : le point est allumé dans la couleur désirée.

SWI &H1Ø : affichage d'un caractère

en entrée : X contient la coordonnée horizontale du caractère (entre Ø et 39)

Y contient la coordonnée verticale (entre Ø et 24)

&H2Ø2B contient les couleurs forme et fond (voir le paragraphe 5.6)

&H2Ø36 contient le code ASCII du caractère.

SWI &H14 : lecture de la couleur d'un point sur l'écran

en entrée : X et Y contiennent les coordonnées du point sur l'écran.

en sortie : B contient la couleur du point.

SWI &H 1A : lecture d'un caractère sur l'écran.

en entrée : X contient la coordonnée horizontale (Ø ≤ × ≤ 39)

A contient la coordonnée verticale (Ø ≤ Y ≤ 24)

en sortie : B contient le code ASCII du caractère positionné en (X, A) sur l'écran.

5.8.3 Gestion du clavier

SWI &HØA : lecture du clavier

Cette routine lit un caractère entré au clavier. Elle tient compte des variables &H2 Ø19 et &H2 Ø76 (autorepeat) vues précédemment. En sortie, B contient le code ASCII du caractère. D'autres codes sont envoyés s'il ne s'agit pas d'un caractère standard du code ASCII (exemple : touche BASIC enfoncée). CC est affecté.

Remarque : La lecture d'une touche requiert trois appels de SWI &HØA dans certains cas.

SWI &HØC : lecture du clavier

Si une touche a été enfoncée, le bit Z de CC est à Ø (1 dans le cas contraire).

B contient le code de la touche.

A indique si une des touches de fonction a été enfoncée.

L'autorepeat (&H2Ø76) n'est pas pris en compte.

5.8.4 Sons :

SWI &HØ8 : envoi un bip sonore si la variable &H2Ø19 est à Ø.

SWI &H1E : émission d'une note

Cette primitive émet une note variant selon les valeurs des adresses &H2Ø39 à &H2Ø3F. Nous n'avons pas eu le temps de voir son comportement plus en détail.

5.8.5 Manettes de jeu :

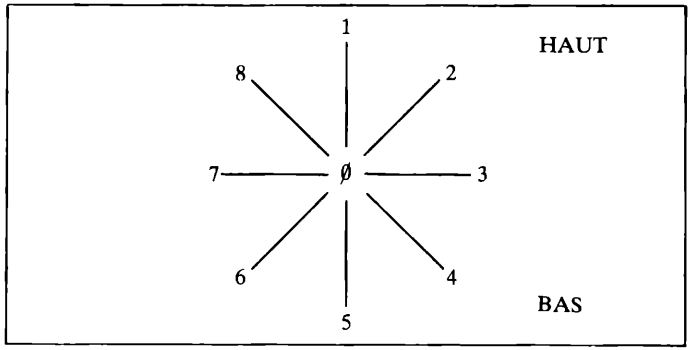
SWI &H1C : scrutation des manettes de jeu.

en entrée : A contient le numéro de la manette de jeu à scruter (Ø ou 1).

en sortie : B contient une valeur comprise entre 1 et 8 selon la position de la manette.

B contient Ø si la manette est au repos.

Les valeurs de B sont les suivantes :



Le bit C (Carry) du CC est à 1 si la touche ACTION de la manette a été enfoncée.

5.8.6 CRAYON OPTIQUE

SWI &H16 : retourne l'état du crayon optique (enfoncé ou non) dans le bit C du CCR du 6809

SWI &H18 : lecture de la position du crayon optique

en entrée : —

en sortie : X
Y — position du crayon sur l'écran

bit C validité de la position

5.9 EXEMPLES D'UTILISATION DES PRIMITIVES

5.9.1 Affichage sur l'écran

Le programme assembleur suivant affiche le caractère "*" à l'écran aux coordonnées (3, 17) en couleur rouge un fond vert :

```
AFF :   LDX  #$3      ; X prend la valeur 3 (horizontale)
        LDY  #$11     ; Y prend la valeur 17 (verticale)
        LDA  #$2A     ; caractère * en ASCII
        STA  $2Ø36    ; initialisation du caractère à écrire
        LDA  #$12     ; rouge = 1 = forme/vert = 2 = fond
        STA  $2Ø2B    ; initialisation couleurs
        SWI           ; appel à la primitive de
        FCB  $10      ; numéro &H1Ø
```

5.9.2 Gestion des caractères spéciaux

Les « caractères » du MO5 ne sont pas tous codés sur un seul octet :

un octet : lettres classiques

deux octets : séquences de commande de l'écran

trois octets : lettres accentuées

Nous nous intéressons ici à l'affichage de ces caractères au moyen de la primitive SWI &HØ2.

— affichage de la lettre A à la position courante du curseur.

```
AFF1 :  LDB  #$41      code ASCII de A
        SWI
        FCB  $Ø2
        RTS
```

— affichage d'une lettre accentuée.

Il est nécessaire de faire les trois appels suivants à SWI &HØ2

- envoi de SS2 (code $16) pour passer en mode accent
- envoi du code de l'accent

$41 accent grave
$42 accent aigu
$43 accent circonflexe
$48 tréma
$4B cédille

• la lettre à « accentuer » :

Le sous-programme suivant affiche un a accent grave.

```
SS2     EQU  $16
AFF2 :  LDB  SS2        envoi de SS2
        SWI
        FCB  $Ø2
        LDB  $41        envoi du code de l'accent grave
        SWI
        FCB  $Ø2
        LDB  $61        lettre a
        SWI
        FCB  $Ø2
        RTS
```

— contrôle de l'écran :

• Gestion des fenêtres

L'envoi successif de

US (code $1F)

$1a

$1b

précise la ligne basse ab de la fenêtre sur l'écran du MO5 (en utilisant SWI &HØ2).

Exemple : L'envoi de la séquence $1F$11$19 positionne la ligne 19 comme le bas de la fenêtre.

De même, l'envoi de $1F $2a $2b précise la ligne haute ab de la fenêtre sur l'écran du MO5.

Exemple :

```
FENETRE :   · LDB  $1F    US
              SWI
              FCB  $Ø2
              LDB  $2Ø    ligne du haut de la fenêtre
```

```
SWI
FCB  $Ø2      = Ø
LDB  $29      + 9
SWI           soit Ø9
FCB  $Ø2
RTS
```

*** Gestion des couleurs et des attributs :**

L'envoi par SWI &HØ2 de ESC (code $13) permet de modifier les couleurs de la forme, du fond et du cadre ainsi que les attributs (taille des caractères, scrolling, vidéo inverse, etc.).

- Modification des couleurs de forme/fond/cadre.

Il faut envoyer ESC (code $1B) suivi de

$4Ø + code couleur pour la forme,

$50 + code couleur pour le fond, et

$60 + code couleur pour le cadre.

Les codes couleurs sont les suivants :

Ø	pour le NOIR
1	" " ROUGE
2	" " VERT
3	" " JAUNE
4	" " BLEU
5	" " MAGENTA
6	" " CYAN
7	" " BLANC
8	" " GRIS
9	" " ROUGE CLAIR
10	" " VERT CLAIR
11	" " JAUNE CLAIR
12	" " BLEU CLAIR
13	" " MAGENTA CLAIR
14	" " CYAN CLAIR
15	" " ORANGE

Ainsi l'envoi par SWI &HØ2 de la séquence $1B $6F donne un cadre de couleur orange.

```
CADROR :    LDB  $1B    ESC
            SWI
            FCB  $Ø2
            LDB  $6F    cadre orange
            SWI
            FCB  $Ø2
            RTS
```

- Modification des attributs :

L'envoi de la séquence ESC ($1B), un code compris entre $7Ø et $7B par la primitive SWI &HØ2 modifie les attributs.

taille des caractères
- $70 taille normale
- $71 double hauteur
- $72 double largeur
- $73 double largeur et double hauteur

mode
- $74 mode caractère/couleur
- $75 mode caractère seul

scrolling
- $78 scrolling normal
- $79 scrolling lent
- $7A mode page

vidéo
- $7B vidéo inverse

Exemple : passage en vidéo inverse.

```
VINF :      LDB  $1B     ESC
            SWI
            FCB  $Ø2
            LDB  $7B     vidéo inverse
            SWI
            FCB  $Ø2
            RTS
```

5.9.3 Lecture des manettes de jeu :

Le sous-programme suivant lit les manettes et range le résultat en &H7ØØØ

```
&H7ØØ1      LDA  #$Ø
            SWI
            FCB  $1C
            STB  $7ØØ1
```

Il est équivalent de faire
```
VAR = STICK(Ø)
ou
EXEC &H7ØØ1
VAR = PEEK (&H7ØØØ)
```

Annexes

A. — Récapitulatif des instructions BASIC

* **ATTRB X, Y**

Définit le mode d'affichage à l'écran : largeur double ou simple, hauteur double ou simple.

* **BEEP**

Produit une impulsion sonore.

* **BOX (X1, Y1) — (X2, Y2), K**

Dessine un rectangle de coins opposés (X1, Y1) et (XL, Y2) en couleur codée par K.

* **BOXF**

Idem BOX avec coloriage intérieur du rectangle.

* **CLEAR**

Permet de réserver de la place mémoire ou de créer des caractères graphiques etc. (cf. chapitre 2).

* **CLOSE I**

Permet de fermer le canal I (et le fichier correspondant).

* **CLS**

Efface la totalité de l'écran.

* **[CNT] [C]**

Arrête l'exécution d'un programme

* **COLOR X, Y, Z**

Change la couleur des caractères et du fond de l'écran.

* **CONSOLE I, J, K, L**

Définit la fenêtre de travail.

* **CONT**

Reprise de l'exécution d'un programme.

* **DATA**

Introduction dans le programme d'une liste de données.

* **DEFGR$**

Permet de définir des caractères graphiques.

* **DEFINT/DEFSNG/DEFSTR**

Permettent de déclarer le type des variables selon la première lettre de leur nom.

* **DELETE**

Détruit tout ou partie d'un programme (cf. chapitre 2).

* **DIM**

Dimensionne les matrices (ex : DIM A(20)).

* **END**

Indique la fin du programme principal.

* **ERL, ERR**

Variables contenant le numéro de la ligne où s'est produite la dernière erreur, ainsi que le code de l'erreur

* **ERROR I**

Permet de simuler l'erreur codée par le numéro I.

* **EXEC** adresse mémoire

Lance l'exécution d'un sous-programme écrit en langage machine.

* **FOR ... TO ... STEP**

:

NEXT

Instruction de boucle (cf. chapitre 2).

* **GOSUB**
:
RETURN

Instruction de branchement vers un sous-programme (écrit en BASIC) (cf. chapitre 2).

* **GOTO**

Instruction de branchement inconditionnel (cf. chapitre 2).

* **IF ... THEN ... ELSE ...**

Instruction de contrôle permettant des branchements conditionnels (cf. chapitre 2).

* **INPEN, INPUTPEN**

Lecture des coordonnées d'un point de l'écran repéré par le crayon optique.

* **INPUT #**

Introduction de données (au clavier) en cours d'exécution d'un programme (cf. chapitre 2).

* **INPUT**

Lecture de données inscrites sur un fichier.

* **LINEINPUT**

Permet d'introduire dans une variable chaîne de caractères une suite comportant plus de 255 caractères.

* **LINEINPUT #**

Permet de lire dans un fichier des suites de caractères et de les affecter à une variable chaîne de caractères.

* **LIST**

Permet de lister ou d'enregistrer tout ou partie d'un programme.

* **LOAD**

Chargement en mémoire centrale d'un programme BASIC.

* **LOADM**

Chargement en mémoire centrale d'un programme binaire.

* **LOCATE I, J, K**

Place le curseur à l'endroit désiré (coordonnées I, J) en le rendant visible ou non suivant la valeur de K.

* **MID$**

Permet des modifications de chaîne de caractères (cf. chapitre 2).

* **MOTORON**

Permet de mettre le moteur du lecteur-enregistreur de cassette en marche.

* **MOTOROFF**

Provoque l'arrêt du moteur.

* **MERGE**

Permet de fusionner deux programmes.

* **NEW**

Efface le contenu de la zone mémoire réservée à l'utilisateur (programme et variables).

* **ON ERROR GOTO**

Permet de ne pas stopper l'exécution d'un programme si une erreur est rencontrée.

* **ON ... GOTO**
ON ... GOSUB

Branchements conditionnels (cf. chapitre 2).

* **OPEN**

Ouverture d'un canal logique (et du fichier correspondant).

* **PLAY**

Permet de jouer de la musique avec le MO5.

* **POKE I, J,**

Place dans la case mémoire I la valeur représentée par J.

* **PRINT (ou ?)**

Affichage à l'écran de valeurs de variables ou de messages.

* **PRINT #**

Permet d'inscrire des données sur un fichier.

* **PRINTUSING**

Affichage à l'écran (comme PRINT), selon un format décrit par l'utilisateur, dans une expression chaîne appelée image (cf. chapitre 2).

* **PRINT # USING**

Permet d'inscrire des données sur un fichier selon un format défini par l'utilisateur.

* **READ A, ...**

Lecture de données inscrites dans un programme (cf. instruction DATA) et de les affecter aux variables A etc.

* **REM (ou)**

Commentaires dans un programme.

* **RESTORE**

Permet de relire un jeu de données (placé après les instructions DATA).

* **RESUME**

Retour au programme principal après traitement d'un sous-programme d'erreur (cf. chapitre 2).

* **RUN**

Lance l'exécution d'un programme : soit celui qui est situé en mémoire centrale, soit un programme situé sur un périphérique (cf. chapitre 2).

* **SAVE [fichier]**

Sauvegarde d'un programme dans un fichier sur un périphérique donné.

* **SAVEM**

Enregistre une image-écran ou un programme écrit en langage machine.

* **SCREEN I, J, K**

Modifie la couleur des caractères, du fond et du cadre (cf. chapitre 3).

* **SCREENPRINT**

Recopie l'écran sur l'imprimante (référencée par PR9Ø-Ø4Ø exclusivement).

* **SKIPF**

Positionne la bande d'une cassette à un endroit voulu (cf annexe D).

* **STOP**

Stoppe l'exécution d'un programme.

* **TRON**

Permet de passer en mode trace (cf. chapitre 2).

* **TROFF**

Permet de quitter le mode trace (cf. chapitre 2).

B. — Explication des messages d'erreur

Il existe 35 messages d'erreur différents. Ils sont explicités, ci-dessous et un par un , dans l'ordre de leur code d'erreur.

Remarque préliminaire : les messages d'erreur apparaissent de deux manières :

— en mode calcul le MO5 affiche le terme Error suivi du numéro de l'erreur détectée

Exemple :

Error 11

— en mode programme, l'ordinateur affiche le même message (qu'il aurait imprimé en mode calcul) suivi du numéro de ligne où s'est produite l'erreur

Exemple :

Error 11 in 1ØØ
(l'erreur est apparu en (in) ligne 1ØØ).

*** Erreur 1**

Elle correspond à une instruction de boucle(s) mal définie. Deux cas se présentent :

1. Le MO5 a rencontré un NEXT se rapportant explicitement ou implicitement à une variable, sans avoir trouvé au préalable une instruction FOR relative à cette même variable (explicite ou implicite).

Exemple :

```
1Ø DIM A(1Ø)
2Ø PRINT A(I)
3Ø NEXT (ou 3Ø NEXT I)
```

2. La suite des variables rencontrées dans la file des instructions FOR *n'est pas retrouvée dans l'ordre inverse* au fur et à mesure de la lecture des instructions NEXT. C'est le cas, par exemple, pour les boucles enchevêtrées.

Exemple :

```
1Ø FOR I = Ø TO 3
2Ø FOR J = Ø TO 3
3Ø INPUT A(I, J)
4Ø NEXT I, J
```

*** Erreur 2**

Elle indique une erreur de syntaxe. Cette erreur est relativement courante. Il faut vérifier la ponctuation, les espacements, les parenthèses, les guillemets, etc...

Exemple :

```
1Ø INPUT "VOTRE NOM S.V.P.", NOM$
2Ø PRINT "BONJOUR", NOM$
```

*** Erreur 3**

Dans un programme le MO5 a rencontré une instruction RETURN sans avoir préalablement trouvé une instruction GO-SUB. Cette erreur est propre aux sous-programmes insérés dans le programme principal au lieu d'être placés à sa suite (après le END final).

Exemple :

```
1Ø PRINT "TAPEZ VOTRE MISE (POSITIVE)"
2Ø INPUT "MISE", MISE
3Ø IF MISE = Ø THEN GOSUB 1Ø
  :
```

Remarque : Il faut utiliser un GOTO à la place du GOSUB ou déplacer le sous-programme après le END final.

*** Erreur 4**

Le MO5 peut recevoir l'ordre de lire des données, dans les lignes de programme repérées par l'instruction DATA, par l'intermédiaire d'une instruction READ. Si les données sont épuisées alors que le MO5 doit poursuivre la lecture, le message d'erreur 4 est révélé.

Exemple :

```
1Ø DIM A(1Ø)
2Ø FOR I = Ø TO 1Ø
3Ø READ A(I)
4Ø NEXT I
1ØØ DATA 1, 3, 5, 7, 9
```

Il y a 5 données et 11 instructions de lecture (lignes 2Ø à 4Ø).

*** Erreur 5**

Cette erreur correspond à une mauvaise utilisation de fonction (au sens large). Elle peut se présenter dans de nombreuses circonstances :

Exemple 1:

1Ø DIM A(5)
2Ø INPUT A(− 2) (l'indice de « localisation » est négatif, ce qui est incorrect)

Exemple 2 :

Avec POKE I, J si : • I est négatif ou supérieur à 65 535 ;
• J est négatif ou supérieur à 255.

*** Erreur 6**

Une variable numérique a dépassé sa valeur limite.

Exemple :

```
A% = 327ØØ [ENTREE]
A% = 2 * A% [ENTREE]
```

*** Erreur 7**

Elle indique que la capacité de la zone mémoire réservée à l'utilisateur a été dépassée (il faut utiliser une extension mémoire, ou simplifier (alléger) si possible le programme ou les jeux de variables).

*** Erreur 8**

Cette erreur apparaît avec l'utilisation des instructions de branchement (GOTO, GOSUB, etc...), lorsque la (ou les) ligne(s) de branchement n'existe(nt) pas.

Exemple :

```
5Ø GOTO 99
 :
9Ø PRINT SCORE
1ØØ INPUT "VOULEZ VOUS REJOUER", A$
 :
```

*** Erreur 9**

Une matrice *dimensionnée* préalablement a été mal utilisée. L'ordinateur a reçu l'ordre de travailler avec une case *inexistante* de la matrice.

Exemple :

```
1Ø DIM A(5, 1Ø)
2Ø INPUT A(5, 11), A(2, 8, 3)
```

En ligne 2Ø l'indice numéro 2, à savoir 11, est trop grand. Il ne doit pas dépasser 1Ø. De plus la matrice A est de dimension 2 (ligne 1Ø) donc il n'y a que *deux* indices.

Il faut vérifier :

— si les indices sont corrects ;

— quelles sont les instructions de DIMensionnement des matrices.

*** Erreur 1Ø**

L'erreur apparaît uniquement lorsqu'un tableau (ou matrice) se trouve déclaré deux fois : implicitement ou explicitement.

Exemple :

```
1Ø DIM A(1Ø)
2Ø DIM A(2Ø)
```

*** Erreur 11**

Cette erreur correspond à la division par zéro.

Exemple :

```
PRINT A/Ø [ENTREE]
```

*** Erreur 12**

Cette erreur n'apparaît qu'en mode calcul. L'instruction utilisée est sans signification en mode calcul.

Exemple :

```
INPUT A [ENTREE]
```

*** Erreur 13**

Cette erreur est relative à un conflit de type. L'ordinateur reçoit l'ordre de mettre une chaîne de caractères dans une mémoire affectée à des valeurs numériques ou l'inverse.

Exemple :

```
1Ø INPUT "NOM", A$
2Ø B = A$ + A$
```

*** Erreur 14**

Le MO5 peut manipuler théoriquement des chaînes de caractères allant jusqu'à 255 caractères. Il convient généralement de le prévenir afin qu'il réserve la place mémoire voulue (pour ces chaînes de caractères). Sans quoi, si par suite d'une ou de plusieurs concaténation(s) une "petite" chaîne grandit, l'ordinateur ne sera pas en mesure de la stocker (cf. instruction CLEAR).

Exemple :

```
1Ø INPUT A$
2Ø FOR I = 1 TO INT (255/LEN(A$) )
3Ø B$ = B$ + A$
4Ø NEXT
```

Remarque : Il est possible qu'avec le programme ci-dessus le message d'erreur n'apparaisse pas à tous les coups mais la probabilité d'apparition reste grande.

*** Erreur 15**

L'ordinateur se trouve avec une chaîne de caractères dépassant 255 caractères. L'utilisateur doit alors fractionner cette chaîne en un nombre suffisant de parties, de sorte que chacune d'elles contienne moins de 255 caractères.

*** Erreur 16**

Une expression sur chaîne trop complexe détermine cette erreur.

*** Erreur 17**

Cette erreur indique que l'exécution d'une suite d'instructions est impossible. Les raisons peuvent être multiples.

Exemple :

L'exécution d'un programme est interrompue par l'instruction [CNT] [C]. Si ce programme est modifié, une tentative de reprise d'exécution n'est pas possible.

*** Erreur 18**

Le MO5 reçoit l'ordre d'utiliser une fonction non définie.

*** Erreur 19**

L'instruction RESUME est absente dans le programme. Elle est nécessaire dès qu'une instruction ON ERROR GOTO est placée dans un programme (cf. chapitre 2).

Exemple :

```
1Ø ON ERROR GOTO 5Ø
2Ø INPUT A, B
3Ø PRINT A/B
4Ø END
```

```
5Ø PRINT "B NE DOIT PAS ETRE NUL"
6Ø END
```

Si, à l'exécution, la valeur introduite pour B est Ø l'erreur 19 sera révélée.

*** Erreur 2Ø**

Si, dans un programme, l'ordinateur rencontre une instruction RESUME sans avoir préalablement détecté une erreur, il s'arrête et affiche l'erreur 2Ø.

Exemple :

```
1Ø ON ERROR GOTO 4Ø
2Ø INPUT "DONNEZ DEUX NOMBRES", A, B
3Ø PRINT A/B
4Ø RESUME
```

Si, à l'exécution, la valeur introduite par B est différente de Ø, l'erreur 2Ø apparaît.

Remarque : en pratique et généralement, les instructions RESUME ne doivent pas faire partie du programme principal mais plutôt du sous-programme (ou du moins être situées après le END final).

Exemple :

```
1Ø ON ERROR GOTO 5Ø
2Ø INPUT "DONNEZ DEUX NOMBRES", A, B
3Ø PRINT A/B
4Ø END
5Ø RESUME
```

*** Erreur 21**

L'instruction ERROR (cf. chapitre 3) a été utilisée avec un code d'erreur incorrect.

Exemple :

```
1Ø ERROR -5 (l'erreur -5 n'existe pas)
```

*** Erreur 22**

Il manque un opérande dans une expression.

Exemple :

```
PRINT 2 +
```

Un nombre manque après l'addition.

*** Erreur 23**

Cette erreur correspond *uniquement* à une instruction de boucle mal définie. Il manque une instruction Next à la fin de la boucle (For ... Next).

Exemple :

```
1Ø DIM A(1Ø, 1Ø)
2Ø FOR I = Ø TO 1Ø
3Ø FOR J = Ø TO 1Ø
4Ø PRINT A(I, J)
5Ø NEXT J
```

Il faut écrire :

```
    5Ø NEXT J, I
OU  5Ø NEXT J
    6Ø NEXT I
```

*** Erreur 50**

Les canaux qui peuvent être ouverts et fermés sont au nombre de 16 (de 1 à 15). Cette erreur correspond à une tentative d'ouverture ou de fermeture d'un canal différent des 16 utilisables.

Exemple :

```
Open "O", 17, "DON" [ENTREE]
```

*** Erreur 51**

Cette erreur indique que le mode d'accès au fichier est incorrect.

Exemple 1 :

```
OPEN "I", 4, "ZØ"
PRINT#4, 89Ø
```

L'ouverture du fichier s'est faite par le canal 4 en Input, soit dans le sens périphérique vers MO5. Or l'instruction PRINT# est réservée aux communications MO5 vers périphérique.

Exemple 2 :

Avec la commande MERGE, le fichier à charger doit être codé sous forme ASCII (sans quoi l'erreur 51 est détectée).

*** Erreur 52**

L'utilisateur a essayé d'ouvrir un canal déjà ouvert.

Exemple :

```
1Ø OPEN "I", 1, "ESSAI ONE"
2Ø OPEN "O", 1, "RESULT"
```

L'erreur 52 est détectée en ligne 2Ø, car le canal 1 se trouve déjà ouvert en ligne 1Ø.

*** Erreur 53**

Elle est significative d'une erreur d'entrée-sortie, (Input, Output). Cela peut apparaître lors de l'utilisation d'un périphérique qui présente un défaut de fonctionnement.

*** Erreur 54**

Cette erreur est relative à la lecture d'un fichier vide. En effet la lecture d'un fichier (à travers un canal) ne peut être effectuée que si celui-ci est non-vide.

Exemple :

```
1Ø OPEN "O", 3, "ZONE"
2Ø CLOSE 3
```
} on ouvre le fichier dans lequel rien n'est mis puis on le referme

```
3Ø OPEN "I", 3, "ZONE"
4Ø INPUT#3, A
5Ø CLOSE 3
```
} on essaie de lire le fichier (vide)

*** Erreur 55**

Le descripteur de fichier est incorrect.

Exemple 1 :

```
SAVE "COMM :"
```

Il manque le nom du fichier dans lequel doit être sauvegardé le programme (cf. chapitre 2).

*** Erreur 56**

Cette erreur indique que l'utilisateur a tenté une commande directe dans un fichier (codé ASCII) qui se trouve en cours de chargement.

*** Erreur 57**

Le MO5 a reçu l'ordre de travailler avec un périphérique (sur un fichier) sans que le canal nécessaire à cette communication ait été ouvert. Il est donc nécessaire d'ouvrir un canal en Input ou Output selon le cas (cf. chapitre 2).

Exemple :

```
1Ø INPUT#3, DONNEE, NOM$
```

Il faut ouvrir le canal 3 en Input.

*** Erreur 58**

L'utilisateur a utilisé incorrectement les instruction READ et

DATA. L'ordinateur a reçu l'ordre de lire une donnée . Le type de la donnée lue (numérique ou chaîne de caractères) n'est pas le même que le type de la donnée attendue.

Exemple :

```
1Ø READ A$, NB
:
1ØØ DATA "JOSEPHINE", "SCORE"
```

A la lecture (ligne 1Ø) le MO5 attend une chaîne de caractères et un réel et il se présente (ligne 1ØØ) deux chaînes de caractères.

Dans ce cas il faut vérifier les lignes « DATA et READ » et vérifier si une instruction RESTORE (cf. chapitre 2) n' a pas été oubliée dans le programme.

*** Erreur 59**

Elle apparaît lors de l'utilisation d'un périphérique si ce dernier est déjà en utilisation.

*** Erreur 60**

L'ordinateur a reçu l'ordre d'utiliser un périphérique mal raccordé ou inexistant. Il faut donc :

— renoncer à utiliser le périphérique,
ou
— vérifier si celui-ci est bien connecté au MO5.

*** Erreur 61**

Lorsque les programmes sont protégés, toute tentative

— de listage
— de modification du programme
— d'utilisation des fonctions PEEK, POKE etc...

se traduit par ce message d'erreur.

*** Notons l'erreur :** ? Redo qui joue un rôle un peu à part.

Cette erreur est révélée lorsque le MO5 attend une donnée de la part de l'utilisateur (cf. instruction INPUT) et que ce dernier opère

une mauvaise Introduction. Il faut alors réintroduire correctement (si possible) la donnée.

Exemple :

MISE ?	3,14 [ENTREE]
cette partie est affichée par le MO5 grâce au programme suivant :	cette partie est tapée par l'utilisateur (elle est incorrecte car la virgule doit être remplacée par un point).

1Ø INPUT «MISE», A

C. — Utilisation d'une cassette

1. *Généralités*

1.1 Le lecteur-enregistreur a deux fonctions :

— enregistrer sur la cassette l'information en provenance de l'ordinateur ;

— lire sur la cassette l'information à destination du MO5.

L'information peut être de plusieurs types :

— un programme écrit en BASIC,
— un programme écrit en langage machine,
— un ensemble de données.

1.2 Raccordement du lecteur-enregistreur

Il se fait au moyen de deux cables à savoir :

— l'alimentation (du lecteur avec le secteur 22Ø V alternatif),

— la liaison avec le MO5 (le câble est terminé par une prise DIN qui est à brancher sur le flanc droit du MO5).

Si une cassette* est introduite et si tout est mis sous tension (MO5 + lecteur) la manœuvre de :

— la touche [▷▷] ou [◁◁] fait défiler la bande selon le sens indiqué ;

(*) En principe toute cassette audio peut convenir, mais une cassette de bonne qualité est cependant recommandée.

— les touches [▷] ou [▷] et [enregistrement] sont sans effet *apparent* sauf s'il y a un ordre provenant du MO5 (cet ordre est tapé par l'utilisateur).

remarque : les syntaxes qui se trouvent dans cette annexe concernent exclusivement l'utilisation de la cassette.

2. *Instructions d'enregistrement* (sur cassette)

Ces instructions supposent que le lecteur-enregistreur se trouve connecté et que les touches lecture et enregistrement ont été enfoncées (le voyant rouge doit se trouver allumé).

2.1 Enregistrement de programme ou de partie de programme

2.1.1 SAVE permet d'enregistrer sous l'une des trois formes suivantes la *totalité* du programme BASIC situé dans la mémoire centrale (du MO5) à savoir :

— sous forme conventionnelle ;
— sous forme codée ;
— sous forme protégée.

Exemples de chacune des trois formes :

— 1 SAVE "JEU"
— 2 SAVE "JEU", A (A pour ASCII)
— 3 SAVE "JEU", P (P pour protégé)

Remarque : dans les trois cas la cassette défile un certain temps puis s'arrête et le message OK, signalant la fin de l'enregistrement, s'affiche à l'écran si aucune erreur n'a été rencontrée. Au cas où une erreur est apparue il faudra recommencer la procédure d'enregistrement.

2.1.2 LIST

Syntaxe :

LIST "CASS : nom de fichier" [, plage de numéro de ligne]

Permet d'enregistrer, sous forme codée ASCII, un ensemble de lignes de programme (indiqué par plage de numéro de ligne). Il est nécessaire de préciser le nom du périphérique soit CASS sans quoi

le MO5 considère d'autorité l'imprimante.

Remarques :

• Si la plage de numéro de ligne est omise dans la commande, cette instruction produira le même effet que :

SAVE "nom de fichier", A

(la totalité du programme est enregistrée)

• Voir la remarque commande SAVE ci-avant.

Exemple :

LIST "CASS : JEU", 5Ø – [ENTREE]

enregistre les lignes 5Ø et suivantes.

2.2 Enregistrement, dans des fichiers binaires, de programmes écrits en langage machine ou d'images écran

Remarque : ce type de fichier sert très peu ou pas du tout lors d'une première approche.

Les adresses mémoires où se trouve le programme dans l'unité centrale doivent être connues ainsi que l'adresse de lancement de ce programme.

Les images écran sont stockées dans les mémoires situées de &HØ à &H7999 (cf. paragraphe 3.2.5.3).

Moyennant la connaissance des adresses (mémoires) l'instruction SAVEM (syntaxe SAVEM adresse de début de programme, adresse de fin de programme, adresse d'exécution) provoque l'enregistrement sur cassette du contenu des mémoires ainsi désignées.

2.3. Enregistrement de données (dans un fichier de données)

Il se fait en trois temps :

a) Ouverture d'un canal en Output et par là même d'un fichier permettant le dialogue MO5 vers lecteur-enregistreur (cf. instruction Open chapitre 2).

Exemple :

```
1Ø OPEN "O", # 1, "DONNEES".
```

La bande de la cassette se met à défiler durant quelques secondes et s'arrête.

Le canal 1 est ouvert en Output vers le fichier DONNEES situé sur cassette.

b) Envoi des données à enregistrer

(cf. instruction PRINT# et PRINT#USING chapitre 2).

Exemple :

Reprendre l'exemple précédent en ajoutant les lignes suivantes :

```
2Ø INPUT "VOULEZ-VOUS RENTRER UNE DON-
   NEE ? OUI — NON"; REPONSE$
3Ø IF REPONSE$ = "NON" THEN GOTO 7Ø
4Ø INPUT "TAPEZ LA"; A$
5Ø PRINT#1, A$
6Ø GOTO 2Ø
7Ø END
```

Remarques :

- Si on désire entrer des valeurs numériques il est nécessaire de remplacer A$ en ligne 4Ø et 5Ø ci-dessus par A ou A%
- Dans cette deuxième phase la cassette peut ne pas avancer mais s'il y a défilement il se fait par saccades (ceci est dû à la présence de mémoire tampon ce qui ne fait pas l'objet de ce développement).

c) Fermeture du canal et du fichier

Cette phase est importante car elle sert à repérer sur la bande de la cassette la fin du fichier.

Exemple :

```
7Ø CLOSE 1
8Ø END
```

Dans cette phase la cassette défile toujours durant quelques instants.

3. *Instructions de lecture* (sur cassette)

Le lecteur-enregistreur doit être connecté et la touche lecture [▷] doit être préalablement enfoncée.

3.1 Lecture d'un programme

3.1.1 LOAD

Syntaxe :

LOAD "nom de fichier"

Permet de charger en mémoire centrale la totalité du programme enregistré dans la cassette sous le nom (nom de fichier) donné.

Exemple :

LOAD "JEUX 3" [ENTREE]

La bande se met à défiler, le message Searching (en recherche) apparaît sur l'écran, puis le MO5 affiche le nom des fichiers rencontrés soit par exemple :

Searching	DAT
Skip JEUX 1	ces termes indiquent si
Searching	le fichier est BASIC
Skip DONNEES	(BAS) ou binaire (BIN)
Searching	ou de données (DAT)
BAS	

Jusqu'à trouver le fichier demandé (JEUX3) et alors le message suivant s'affiche :

FOUND JEUX3 BAS

La bande continue de défiler (le MO5 lit ce qu'il y a dans le fichier). Après un temps plus ou moins long suivant la taille du fichier, le défilement est stoppé et le MO5 affiche

OK

si aucune erreur n'a été rencontrée, sinon le message d'erreur correspondant s'affiche (il faudra alors recommencer la lecture).

Remarques :

- si le nom de fichier est omis, le MO5 charge le premier rencontré ;
- si le fichier indiqué n'a pas été trouvé avant la fin de la cassette, il faut réinitialiser le MO5.

3.1.2 Commande MERGE

Syntaxe :

MERGE "nom du fichier" [,R]

Cette instruction permet :

- de charger le programme situé dans la cassette sous le nom nom de fichier,

et

- de le fusionner au programme éventuellement présent dans le MO5 (cf. chapitre 2).

Cette commande est peu utilisée par les néophytes.

Exemple :

LOAD "PGM1" [ENTREE] (le fichier PGM1 est chargé)
MERGE "PGM2" [ENTREE] (le fichier PGM2 va être fusionné avec PGM1)

Remarques :

- le caractère R (qu'il est possible de rajouter à la suite de la commande MERGE) ordonne au MO5 de lancer l'exécution du programme sitôt la fusion effectuée. De ce fait,

MERGE "nom de fichier", R [ENTREE]

est équivalent à :

MERGE "nom de fichier" [ENTREE]
RUN [ENTREE]

- si le nom du fichier est omis le MO5 prend en compte le premier fichier BASIC rencontré.

3.2 Lecture d'un fichier binaire contenant un programme écrit en langage machine ou d'une image écran

Il faut connaître l'adresse des mémoires dans lesquelles va être placé le contenu du fichier, ainsi que les adresses mémoire dans lesquelles le fichier était placé lors de son enregistrement sur cassette. Si ces adresses mémoires ne se correspondent pas il y a un décalage d'adressage. Moyennant la connaissance de la valeur de ce décalage, la lecture peut être effectuée par l'instruction LOADM (cf. chapitre 2).

Exemples :

1. LOADM "FIGURE" [ENTREE]
2. LOADM "FONCTI", 5Ø [ENTREE]

5Ø représente le décalage d'adressage.

La cassette doit défiler jusqu'à trouver le fichier correspondant puis s'arrêter après le chargement (opéré).

3.3 Lecture de fichier de données

Elle se fait en trois temps :

a) Ouverture d'un canal en *Input*

Exemple :

```
1Ø OPEN "I", # 12, "DONNEES"
```

La bande défile — à l'exécution — jusqu'à trouver le fichier DONNEES après quoi elle s'arrête et il est alors seulement possible de passer à la suite des opérations.

b) Lecture des données inscrites sur le fichier

Cf. instructions INPUT #, INPUT$, LINEINPUT# chapitre 2).

Exemple :

Rajouter à la ligne 1Ø de l'exemple précédent :

```
2Ø IF EOF (12) THEN GOTO 6Ø
3Ø INPUT # 12, A
4Ø PRINT A
```

```
5Ø GOTO 2Ø
6Ø END
```

Remarque : ainsi que pour PRINT# la bande ne défile pas continûment.

c) Fermeture du fichier et du canal ouvert

Cette opération est indispensable pour que le MO5 puisse travailler, par la suite, sur d'autres fichiers (sur cette même cassette) sinon un message d'erreur peut apparaître.

Exemple :

Remplacer dans l'exemple précédent (cf. ci-dessus) la ligne 6Ø par :

```
6Ø CLOSE 12
```

4. *Instructions diverses*

En plus des instructions examinées dans les paragraphes précédents il en existe trois autres.

4.1 SKIPF

Syntaxe SKIPF "nom de fichier"

Elle permet de *positionner* la bande de la cassette juste après le fichier indiqué par nom de fichier. Si ce dernier ne figure pas sur la cassette, la bande défile jusqu'au bout et le MO5 doit être réinitialisé. Cette instruction n'est guère utilisée si l'utilisateur procède avec méthode lors de l'emploi des cassettes (cf. paragraphe 4 ci-après).

Exemple :

```
SKIPF "JEUX 3" [ENTREE]
```

Remarque : si le nom du fichier n'est pas précisé le MO5 place la bande de la cassette après le premier fichier rencontré.

4.2 MOTORON — MOTOROFF

Ces instructions permettent respectivement de mettre en route et d'arrêter le défilement de la bande. Elles sont utilisées pour effacer des fichiers ou des cassettes chargées de fichiers. Pour ce faire il faut procéder comme suit :

a) Positionner la bande de la cassette (avec par exemple les indications de position affichées par le compteur) au moyen des touches [▷▷] ou [◁◁] juste avant la zone à effacer.

b) Enfoncer simultanément les touches lecture [▷] et [enregistrement]. Le voyant rouge s'allume.

c) Taper l'instruction suivante sur le MO5 :

MOTORON [ENTREE]

qui provoque l'effacement du contenu de la cassette.

d) L'arrêt du défilement s'obtient en tapant :

MOTOROFF [ENTREE]

5. *Conseils pratiques pour l'utilisation de la cassette*

* Il est important de bien inscrire sur chaque cassette :
  — le nom de chacun des fichiers enregistrés,
  — le type de leur contenu (fichier programme BASIC, fichier binaire, fichier données),

et

  — le repère de début et, éventuellement, de fin de fichier, indiqué par le compteur du lecteur-enregistreur.

* D'autre part il est utile de réserver un espace pour séparer entre chaque fichier enregistré. Cet espace doit correspondre à un défilement minimum de 5 unités de compteur.

* A la lecture d'un fichier donné, il est préférable de faire défiler la cassette au moyen des touches [◁◁] ou [▷▷] afin de la positionner juste avant le fichier cherché.

D. — Utilisation d'une imprimante

1. *Généralités*

* L'imprimante n'a qu'une seule fonction :

Imprimer (sur du papier) l'information en provenance de l'ordinateur.

L'information peut être de plusieurs types :

— un programme BASIC ;
— une image écran ;
— un ensemble de données.

* Deux imprimantes Thomson sont utilisables*.

— l'imprimante PR 9Ø.Ø4Ø permet d'imprimer jusqu'à 4Ø caractères par ligne ;

— l'imprimante PR 9Ø.Ø8Ø permet d'imprimer. jusqu'à 8Ø caractères par ligne.

Remarque : les imprimantes à interface Centronics sont également utilisables.

* Vérifier que l'imprimante se trouve bien connectée au MO5 avec les liaisons prévues.

2. *Impression de programme BASIC*

Elle se fait au moyen de l'instruction LIST.

Syntaxe :

LIST "LPRT : [(constante)]", plage de numéro de ligne].

La constante est le nombre de caractères par ligne : elle est à choisir parmi les entiers positifs inférieurs à 4Ø ou 8Ø, selon l'imprimante. Si cette constante n'est pas mentionnée le MO5 la considère d'autorité comme égale à 4Ø. L'imprimante liste :

- la partie programme indiquée par plage de numéro de ligne ;
- la totalité du programme si la plage de numéro de ligne n'est pas précisée.

Exemple 1 :

LIST "LPRT : (3Ø)", – 15Ø [ENTREE]

permet de lister les 15Ø premières lignes de programme avec 3Ø caractères par ligne.

Exemple 2 :

LIST "LPRT : (8Ø)" [ENTREE]

liste la totalité du programme avec 8Ø caractères par ligne.

3. *Impression d'une image écran*

Elle se fait au moyen de l'instruction SCREENPRINT.

Ainsi ce qui est affiché sur l'écran se retrouve imprimé sur le listing.

4. *Impression d'un ensemble de données*

Elle se fait en trois temps :

a) Ouverture d'un canal en Output vers l'imprimante pour la communication MO5 vers imprimante.

Exemple :

OPEN "O", # 1, "LPRT : (3Ø)" [ENTREE]

Remarques :

- Aucun nom de fichier ne doit être inscrit dans le descripteur de fichier (lorsque c'est l'imprimante qui est concernée).
- Le canal 1 est ouvert en Output vers l'imprimante, les lignes transmises comportent 3Ø caractères.

b) Impression des données

Elle se fait au moyen des instructions PRINT# et PRINT#USING (cf. chapitre 2).

Exemple :

PRINT#1, "BONJOUR MONSIEUR" [ENTREE]

Le message BONJOUR MONSIEUR doit s'imprimer sur le listing.

c) Fermeture du canal

Elle se fait au moyen de l'instruction CLOSE.

Exemple :

CLOSE 1 [ENTREE]

E. — Tableau des codes ASCII

Code	Caractère	Code	Caractère	Code	Caractère
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	/
007	BEL	050	2	093	]
008	BS	051	3	094	†
009	HT	052	4	095	<
010	LF	053	5	096	‘
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	|
038	&	081	Q	124	|
039	'	082	R	124	|
040	(	083	S	125	~
041	)	084	T	126	DEL
042	*	085	U	127	

ASCII codes are in decimal.
LF = Line Feed, FF = Form Feed, CR = Carriage Return, DEL = Rubout

F. — Analyse du jeu "reverse"

* Les lignes 2Ø à 6Ø servent à faire tourner le générateur aléatoire un certain nombre de fois afin d'obtenir des séquences de nombres différentes à chaque début de jeu.

* Les lignes 7Ø à 11Ø déterminent la suite initiale.

* Les lignes 13Ø à 15Ø permettent au joueur de demander la rotation qu'il veut. Toute rotation supérieure à 9 est refusée. Une rotation nulle signifie l'arrêt du jeu.

* Les lignes 16Ø à 26Ø effectuent la rotation demandée.

* Les lignes 22Ø à 24Ø permettent de vérifier si la suite obtenue après permutation est 1 2 3 4 5 6 7 8 9. Si ce n'est pas le cas la ligne 23Ø renvoie à la ligne 13Ø pour une autre rotation.

* Le sous-programme 1ØØØ gère l'affichage. Le point-virgule après l'instruction PRINT A(I) de la ligne 1Ø2Ø est indispensable pour obtenir la suite sur une ligne.

LA BIBLIOTHÈQUE EDIMICRO

• *Collection « Ordinateurs personnels »*

ATMOS/ORIC 1

Kosniowsky ..... **Nouveaux Jeux sur ATMOS**
Chane-Hune, Darbois ..... **Jeux Graphiques sur ATMOS**
Bayvejiel ..... **Guide de l'Oric**
Chane-Hune, Darbois ..... **Jeux sur ORIC**
Viguier ..... **Premiers pas en programmation sur ORIC**

COMMODORE 64

Fleurier, Meiller ..... **Jeux sur Commodore 64 :** Jeux d'adresse et de hasard
Fleurier, Meiller ..... **Jeux sur Commodore 64 :** Jeux d'action et de réflexion

ELECTRON

Bennani, Chaieb ..... **Graphisme et sons sur Electron**

MO5

Bieber, Perbost, Renucci ... **Tout sur le MO5**
Perbost, Renucci ..... **Jeux sur MO5**

SPECTRUM

Bridge, Carnell ..... **Aventures sur Spectrum**
Hurley ..... **Jeux Graphiques sur Spectrum**

TO7

— **ouvrages**
Bieber, Perbost, Renucci ... **Guide du TO7**
Perbost, Renucci ..... **Jeux sur TO7**
— **logiciels sur cassette**
Perbost, Renucci ..... **4 jeux avec manette pour TO7**
Perbost, Renucci ..... **6 jeux d'action et de réflexion pour TO7**

VG 5000 PHILIPS

Amsler, Bardon ..... **Guide du VG 5000 Philips**
Amsler, Villemaud ..... **Jeux sur VG 5000 Philips**

PHILIPS C7420 VIDEOPAC +

Bardon, de Merly ..... **Jeux sur Philips C7420 Vidéopac +**

Achevé d'imprimer en novembre 1984
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : novembre 1984
N° d'imprimeur : 409034