Les livres sur le BASIC sont nombreux, et le possesseur d'un micro-ordinateur est à même d'y trouver toutes les informations nécessaires pour débuter en programmation. Cependant, la réalisation de programmes demandant une grande vitesse d'exécution l'amène à se heurter rapidement à un problème majeur : la lenteur de l'interpréteur BASIC. Ce livre permet d'aller plus loin en abordant la programmation en langage machine. La façon de programmer l'équivalent des instructions BASIC PRINT, GOTO, GOSUB, FOR/NEXT, etc., est tout d'abord étudiée, puis ces notions sont appliquées à la réalisation d'un jeu d'action.
De nombreux sous-programmes pourront être réutilisés par le lecteur dans ses propres programmes.

0153 0985 78 F
9 782736 101534

STEVE WEB

MSX

PROGRAMMES EN LANGAGE MACHINE

STEVE WEBB

MSX

PROGRAMMATION EN LANGAGE MACHINE

SYBEX

Paris • Berkeley • Düsseldorf • Londres

Traduction française de Michel Laurent

SYBEX n'est lié à aucun constructeur.

Publié en 1985 en Grande-Bretagne par :

Virgin Books Ltd,
61-63 Portobello Road
LONDON W113DD

ISBN Version originale : 0863690742
ISBN Version française : 2-7361-0153-1

S O M M A I R E

NOTE DE L'AUTEUR

Ce livre n'est qu'une introduction à la programmation en langage machine. Un grand nombre de codes machine du microprocesseur Z80 n'ont pas été décrits. Leur signification pourra être trouvée dans l'un des nombreux ouvrages spécialisés consacrés exclusivement à ces codes. Le lecteur commençant à écrire ses propres programmes en langage machine pourra être confronté à certaines difficultés non évoquées dans ce livre d'initiation. L'auteur demeure à sa disposition pour des informations supplémentaires. Il suffit de lui écrire, par l'intermédiaire de l'éditeur, en n'oubliant pas de joindre une enveloppe timbrée et libellée à l'adresse de réexpédition. Le lecteur doit également savoir que les programmes présentés dans cet ouvrage ont un apport essentiellement didactique. Ils n'ont pas été écrits dans un souci de performance ou d'élégance particulière, mais seulement pour être facilement compris.

STEVE WEBB
janvier 1985

INTRODUCTION

Ce livre est une introduction à la programmation en langage machine sur les ordinateurs MSX. Il a été conçu pour permettre un apprentissage progressif, ce qui nécessite une lecture linéaire de l'ouvrage. Il est conseillé de n'aborder un chapitre qu'après s'être assuré d'avoir assimilé le contenu des chapitres précédents.

La programmation en langage machine n'est pas aussi difficile que ce que l'on en dit parfois. Si l'on possède un minimum de pratique du langage BASIC, les concepts utilisés pour la programmation en langage machine peuvent être assimilés très rapidement.

Les premiers chapitres de cet ouvrage traitent de la théorie du langage machine, montrant en particulier comment sont stockés les nombres. La description des équivalents fondamentaux de plusieurs instructions BASIC, et celle de l'organisation interne d'un ordinateur MSX, permettent ensuite d'apporter des éléments de réponse à la question : "Qu'est-ce que le langage machine ?" Le chapitre principal montre comment écrire, dans ce langage, un programme de jeu très simple, en commençant par en établir l'organigramme, dont chaque bloc est ensuite converti en de courts sous-programmes en langage machine.

Les derniers chapitres décrivent quelques sous-programmes utilitaires et montrent comment l'on peut tirer parti des possibilités évoluées des ordinateurs MSX. Tout au long de cet ouvrage, quelques questions permettent au lecteur de faire le point sur les connaissances nouvellement acquises. Les réponses à ces questions sont données à la fin du livre. En cas de réponse erronée, il est conseillé de reprendre la lecture du chapitre correspondant, jusqu'à ce que l'on soit capable de fournir la réponse exacte.

LE STANDARD MSX

Au moment où ce livre est écrit, le standard MSX a été adopté par plus de vingt-cinq constructeurs d'ordinateurs. La définition d'un standard constitue une étape importante dans le développement de la micro-informatique familiale et offre des garanties essentielles pour l'utilisateur comme pour le programmeur. La norme concerne tant la partie matérielle *(hardware)* que la partie logicielle *(software)*.

Supposons qu'un programme écrit en BASIC ou en langage machine ait été sauvegardé sur cassette, sur un ordinateur MSX de marque Sony, par exemple. Le standard permet d'utiliser cette cassette et le

programme qu'elle contient sur n'importe quelle autre marque d'ordinateur MSX. Le programme tournera correctement sans nécessiter la moindre modification, la seule condition étant naturellement qu'il ait été écrit en respectant les spécifications du standard.

Cette portabilité, c'est-à-dire la possibilité d'interchanger un logiciel sur des ordinateurs de différentes marques, suppose également une standardisation maximale au niveau matériel et à celui du *firmware* (possibilité de programmer des cartouches équipées d'EPROM). Les variables système sont en effet toujours stockées aux mêmes emplacements mémoire, d'un ordinateur MSX à l'autre, et les sorties écran sont nécessairement les mêmes. Dans ces conditions, pourquoi choisir une marque d'ordinateur MSX plutôt qu'une autre? Comme en ce qui concerne la haute-fidélité, le choix peut être en partie guidé par des critères plus ou moins subjectifs, tels que le prestige ou l'attachement à une marque ou une certaine esthétique du matériel proposé. Il existe cependant d'autres critères (au moins aussi importants) pouvant motiver le choix. Certains fabricants peuvent offrir un matériel disposant de caractéristiques supplémentaires, non imposées par la norme, telles qu'un interface pour crayon optique par exemple. Il est important de se souvenir que ces caractéristiques supplémentaires ne sont pas nécessairement disponibles sur les autres ordinateurs répondant au standard. Un programme qui a été écrit pour être utilisé avec un crayon optique ne tournera naturellement pas sur un ordinateur MSX ne disposant pas de cette sortie. Il est cependant à noter que le standard définit une normalisation pour la plupart des unités périphériques essentielles, telles que les imprimantes par exemple.

Une chose importante n'est pas définie par le standard : il s'agit de la taille mémoire (il existe en fait une norme, mais celle-ci est tellement basse que la totalité des constructeurs proposent un matériel supérieur à la norme). On peut ainsi trouver des ordinateurs MSX de 32, 48 ou 64 K, selon le modèle et la marque. Un programme écrit sur un ordinateur MSX donné tournera sur tout autre ordinateur disposant d'au moins la même capacité mémoire (il pourra éventuellement tourner sur un ordinateur disposant d'une taille mémoire plus faible, s'il ne dépasse pas ses capacités). On peut penser que 64 K deviendra très rapidement la norme de fait des ordinateurs MSX.

Signalons enfin que MSX est un standard universel. Un programme écrit sur un ordinateur MSX vendu en France tournera également sur un ordinateur du standard vendu au Japon ou en Papouasie-Nouvelle-Guinée.

1

LE LANGAGE MACHINE

L'unité de taille de la mémoire centrale d'un ordinateur est le kilo octet (ou K). Un kilo octet correspond à un petit peu plus de 1 000 octets (1 024 exactement), de sorte qu'un ordinateur 64 K dispose d'un peu plus de 64 000 emplacements mémoire (ou adresses) différents. Chaque emplacement mémoire peut être considéré comme une boîte ayant un numéro compris entre 0 et 65535. Ces numéros sont virtuels et servent uniquement à repérer chacune des adresses. Lorsque l'ordinateur est mis sous tension, seuls un peu plus de 28 000 de ces emplacements mémoire peuvent être utilisés pour des programmes BASIC ou des programmes en langage machine. En effet, 4 000 octets sont réservés pour les variables système. Sur les ordinateurs disposant de 64 K, 32 K octets supplémentaires sont disponibles et s'ajoutent aux 28 K décrits précédemment. Cependant, ces 32 K supplémentaires ne sont accessibles qu'au moyen de techniques évoluées de programmation en langage machine qui dépassent le cadre de cet ouvrage d'initiation. Tous les programmes donnés ici peuvent tourner sur des ordinateurs MSX disposant d'au moins 32 K de mémoire.

Chaque emplacement mémoire peut contenir un nombre compris entre 0 et 255 inclus. Le fait de ne pouvoir stocker un nombre supérieur à 255 constitue naturellement une limitation, de sorte qu'une méthode a été recherchée pour stocker des nombres plus grands. L'exemple ci-dessous illustre la méthode qui a été retenue.

Supposons que le nombre 29 248 doive être stocké en mémoire centrale de l'ordinateur. La première opération à effectuer consiste à diviser ce nombre par 256 et à retenir la partie entière de cette division. Ainsi, 29 248 divisé par 256 donne 114,25; la valeur 114 est donc retenue. Elle est appelée partie de poids fort du nombre à stocker et représente le nombre de multiples entiers de 256 qu'il contient. En multipliant la partie de poids fort du nombre par 256 et en soustrayant le résultat obtenu au nombre à stocker, on obtient le reste de la division entière précédente :

114 X 256 = 29 184
29 248 – 29 184 = 64

64 est appelé partie de poids faible de ce nombre.

Le nombre 29 248 sera stocké en mémoire en chargeant la partie de poids faible (64) à une adresse mémoire choisie et la partie de poids fort (114) à l'adresse suivante. Le stockage d'un nombre supérieur à 255 nécessite donc l'utilisation de deux emplacements mémoire. Ainsi, lorsque l'on lit qu'un nombre supérieur à 255 est stocké par exemple à l'adresse 50000, cela signifie qu'il occupe en fait les adresses 50000 et 50001.

Quelles sont les parties de poids fort et de poids faible du nombre 45 621 ?

Quel est le nombre qui a 31 pour partie de poids faible et 64 pour partie de poids fort ?

Chaque emplacement mémoire susceptible de contenir un nombre compris entre 0 et 255 correspond à un octet. Un programme ayant une taille de 5 000 octets occupe donc 5 000 emplacements mémoire.

La zone de la mémoire centrale où sont stockés les programmes utilisateur (qu'il s'agisse de programmes écrits en BASIC ou en langage machine) est appelée mémoire RAM (*Random Access Memory*/ mémoire à accès sélectif). Le contenu de la mémoire RAM est effacé dès que l'ordinateur n'est plus sous tension. Il existe une autre partie de la mémoire centrale appelée mémoire ROM (*Read Only Memory*/ mémoire à lecture seule). Comme son nom l'indique, il n'est possible que de lire le contenu des adresses correspondantes et non de les modifier. Ce contenu n'est pas effacé lorsque l'ordinateur n'est plus sous tension, ce qui est normal car se trouve en particulier à cet emplacement le système d'exploitation sans lequel l'ordinateur ne serait qu'un bloc de métal inerte. Le système d'exploitation contient un certain nombre de sous-programmes en langage machine qui gèrent les fonctions principales de l'ordinateur; ils permettent par exemple de lire le clavier, d'effectuer les calculs fondamentaux et de vérifier la syntaxe des programmes écrits en BASIC.

Le cœur de l'ordinateur est son microprocesseur (de type Z80). Il n'a pas l'intelligence d'un cerveau. La seule chose qu'il puisse faire, ce sont des opérations arithmétiques tout à fait élémentaires. Sa puissance, comparée à celle du cerveau humain, résulte dans le fait que le microprocesseur effectue un très grand nombre d'opérations élémentaires par seconde, ce qui peut parfois lui donner l'apparence d'une certaine " intelligence ".

Le microprocesseur ne comprend que les instructions en langage machine. Il existe plus de 600 variantes possibles pour ces instructions. Chaque instruction correspond à un nombre ou à une combinaison de nombres. Par exemple, le nombre 198 correspond à l'instruction commandant d'additionner deux nombres, de sorte que, lorsque le microprocesseur rencontre le nombre 198, il sait que l'opération qu'il doit immédiatement effectuer est une addition (la procédure qu'il utilise pour réaliser cette opération sera décrite plus avant dans l'ouvrage).

Le microprocesseur ne comprenant que les instructions en langage machine, comment peut-il intervenir dans l'exécution de programmes écrits en BASIC ? La réponse à cette question tient à l'existence d'un interpréteur, équivalent informatique de l'interprète dans une langue étrangère. De la même manière qu'un interprète a pour fonction de traduire un discours d'une langue dans une autre, le BASIC s'adresse au microprocesseur par l'intermédiaire d'un interpréteur. On conçoit volon-

tiers que le recours à un interprète ou à un interpréteur a pour inconvénient de ralentir le rythme de la conversation entre deux personnes ou entre le programme BASIC et le microprocesseur.

La possibilité de programmer en langage machine permet, en accédant directement au microprocesseur, de se passer de l'interpréteur, ce qui augmente d'autant la vitesse d'exécution d'un programme. En règle générale, un programme écrit en langage machine s'exécute 50 à 100 fois plus vite que le programme équivalent écrit en BASIC.

2

ÉQUIVALENCE DE QUELQUES INSTRUCTIONS BASIC EN LANGAGE MACHINE

Certaines commandes BASIC telles que LIST, NEW, RENUM, DELETE et AUTO n'auraient aucune raison d'être en langage machine et n'ont donc pas d'équivalent. D'autres instructions BASIC telles que RUN, READ, DATA, PRINT, VPOKE et VPEEK, n'ont pas réellement d'équivalent en langage machine, mais nous verrons qu'il est possible de simuler leur action au moyen de certains codes (ou instructions en langage machine).

Les codes présentés ici sont les plus fréquemment utilisés. Leur connaissance suffit à écrire de petits programmes en langage machine.

En BASIC, on est habitué à utiliser des variables telles que A, B, C, ..., X, Y, Z. En langage machine, de telles variables n'existent pas. Leurs équivalents les plus proches sont les registres. Il existe un très petit nombre de registres, et ceux qui sont les plus utilisés sont notés par les lettres :

A, B, C, D, E, H, L

Un autre registre, noté F, sera décrit un peu plus loin. Chaque registre peut être considéré comme représentant un emplacement mémoire; ainsi, ils ne peuvent contenir que des nombres compris entre 0 et 255. Afin de pouvoir utiliser les registres pour y stocker des nombres plus grands, six d'entre eux ont été groupés par paires, de la manière suivante :

HL
BC
DE

Ces regroupements n'interdisent pas d'utiliser chacun de ces registres individuellement.

L'instruction BASIC :

LET A=5

a pour code machine équivalent :

LD A,5

LD est une abréviation du mot LOAD (charger). La signification complète du code ci-dessus est « Charger la valeur 5 dans le registre A ».

Chacun des cinq autres registres peut ainsi recevoir une valeur comprise entre 0 et 255 :

```
LD H,199 (Charger la valeur 199 dans le registre H)
LD D,2   (Charger la valeur 2 dans le registre D)
```


Il a été mentionné précédemment que chaque registre pouvait être considéré comme représentant un emplacement mémoire. Il a été également précisé qu'un nombre tel que 827 avait la valeur 3 (partie entière de la division de 827 par 256) comme partie de poids fort et la valeur 59 comme partie de poids faible. Lorsque le code machine LD HL,827 est exécuté, la partie de poids fort du nombre 827 est chargée dans le registre H et la partie de poids faible dans le registre L (H et L sont d'ailleurs les initiales des mots anglais *high* (haut) et *low* (bas). De même, dans le registre double BC, B est la partie haute alors que dans la paire DE, E est la partie basse :

BASIC : LET A=B
Signification : Donner à la variable A la même valeur que celle contenue dans la variable B.
Code machine : LD A,B
Signification : Charger dans le registre A la même valeur que celle contenue dans le registre B.

Il est ainsi possible de charger dans un registre simple la même valeur que celle contenue dans n'importe quel autre registre simple. On peut ainsi avoir les codes suivants :

```
LD A,H
LD E,A
LD H,C
```

Le code machine LD est probablement le plus utilisé (il est même difficile de concevoir un programme machine sans en faire usage).

BASIC : LET A=A+5
Signification : Ajouter 5 à la valeur courante de A.
Code machine : ADD A,5
Signification : Augmenter de 5 le contenu du registre A.

Le registre A est le seul pour lequel il soit possible d'augmenter ainsi directement la valeur qu'il contient. Les codes machine suivants sont par exemple interdits :

```
ADD B,9
ADD E,3
```

Cette limitation apparente du microprocesseur Z80 peut être facilement contournée, comme décrit plus loin.

BASIC : LET A=A+B
Signification : Ajouter la valeur de B à celle de A et stocker la valeur obtenue dans A.
Code machine : ADD A,B
Signification : Augmenter la valeur courante du registre A de celle contenue dans le registre B.

Il est ainsi possible d'ajouter à la valeur courante du registre A le contenu de l'un quelconque des sept registres précédemment mentionnés :

```
ADD A,B
ADD A,C
ADD A,H
ADD A,L
ADD A,D
ADD A,E
ADD A,A
```

Le dernier code, ADD A,A, a pour effet de doubler la valeur courante du registre A. Toute autre combinaison que celles indiquées ci-dessus est interdite; c'est par exemple le cas des codes suivants :

```
ADD B,H
ADD D,C
```

Les registres doubles peuvent également être additionnés. Cependant, les seules combinaisons permises sont les suivantes :

```
ADD HL,BC
ADD HL,DE
ADD HL,HL
```

Le résultat doit donc toujours être stocké dans HL. Le dernier code, ADD HL,HL, a pour effet de doubler la valeur courante du registre HL. Il n'est pas possible d'ajouter directement la valeur contenue dans un registre simple à celle d'un registre double.
Pour ajouter un nombre au contenu d'un autre registre que le registre A, la procédure à suivre est la suivante, en supposant par exemple que l'on veuille ajouter 9 à la valeur courante 14 du registre B :

Étape 1 : **LD A,B**
Le registre A a alors la même valeur que celle contenue dans le registre B.

Étape 2 : **ADD A,9**
Le contenu du registre A est augmenté de 9.

Étape 3 : **LD B,A**
Le résultat de l'opération est transféré dans le registre B.

Il est ainsi possible, au moyen de trois instructions, d'ajouter un nombre à la valeur contenue dans n'importe lequel des sept registres.

Une autre limitation précédemment mentionnée du microprocesseur Z80 réside dans le fait qu'il est seulement possible d'ajouter au registre A le contenu d'un autre registre. Ainsi, on ne peut écrire :

```
ADD B,H
```


Cette restriction peut être contournée de la manière suivante. Supposons que B ait la valeur 5, H la valeur 7 et que l'on veuille additionner le contenu de ces deux registres et stocker le résultat dans B :

Étape 1 : **LD A,B**
Le registre A a ainsi la même valeur que celle du registre B.

Étape 2 : **ADD A,H**
L'addition est effectuée dans le registre A.

Étape 3 : **LD B,A**
Le résultat est transféré dans le registre B.

De nouveau, trois étapes suffisent pour additionner le contenu de deux registres quelconques. Il est aussi possible d'additionner ainsi la valeur d'un registre à lui-même.

De la même manière, on peut additionner le contenu de deux registres doubles quelconques. Pour ajouter par exemple le contenu du registre BC à celui de DE, le résultat devant être stocké dans BC, les trois étapes suivantes doivent être effectuées :

Étape 1 : **LD H,B**
LD L,C
Le registre double HL prend ainsi la valeur du registre BC.

Étape 2 : **ADD HL,DE**
L'addition est effectuée dans HL.

Étape 3 : **LD B,H**
LD C,L
Le résultat est transféré dans le registre double BC.

La plupart des limites du microprocesseur Z80 peuvent être ainsi aisément contournées, un minimum de pratique permettant de sélectionner la méthode la plus efficace pour répondre à un besoin particulier.

BASIC : LET A=A–5
Signification : Soustraire 5 à la valeur courante de la variable A.
Code machine : SUB A,5
Signification : Soustraire 5 à la valeur contenue dans le registre A.

Comme dans le cas de l'addition, le registre A est le seul auquel on puisse soustraire directement un nombre. Dans le cas des six autres registres, la procédure à employer pour effectuer cette opération est la suivante, en supposant par exemple que la valeur 5 soit à soustraire du registre D :

Étape 1 : **LD A,D**
Le registre A prend ainsi la valeur de D.

Étape 2 : **SUB A,5**
La soustraction est effectuée dans le registre A.

Étape 3 : **LD D,A**
Le résultat de l'opération est transféré dans le registre D.

BASIC : LET A=A–B
Signification : Soustraire B à A et stocker le résultat dans A.
Code machine : SUB B
Signification : Soustraire la valeur du registre B de celle du registre A.

Là encore, le registre A est le seul auquel il soit ainsi possible de soustraire la valeur contenue dans un autre registre.

Pour soustraire la valeur d'un registre à celle d'un registre autre que A, il est nécessaire de passer par trois étapes semblables à celles décrites ci-dessus.

Il est également possible de soustraire le contenu d'un registre double de celui d'un autre. Cependant, les seules combinaisons permises sont les suivantes :

```
SBC HL,BC
SBC HL,DE
SBC HL,HL
```

SBC est une forme particulière de la soustraction. L'instruction SUB ne peut pas être utilisée sur des registres doubles avec le microprocesseur Z80. La signification complète du code SBC sera donnée un peu plus loin. Pour le moment, il suffit de le considérer comme un simple opérateur de soustraction. Une instruction très utilisée, INC, permet d'ajouter (d'INCrémenter) une unité au contenu d'un registre simple ou d'un registre double. Les combinaisons possibles sont les suivantes :

```
INC A
INC B
INC C
INC D
INC E
INC H
INC L
INC HL
INC BC
INC DE
```

De manière semblable, l'instruction DEC permet de soustraire (de DECrémenter) une unité au contenu de n'importe lequel des registres (simples ou doubles).

BASIC : GOTO (numéro de ligne)
Signification : Branchement vers la ligne spécifiée.
Code machine : JP (adresse mémoire)
Signification : Saut à l'adresse mémoire spécifiée.

Le code machine JP a une fonction très proche de l'instruction BASIC "GOTO", l'adresse mémoire remplaçant ici le numéro de ligne. Le chapitre suivant montrera comment les codes machine sont stockés en mémoire. L'utilisation de ceux-ci deviendra alors plus claire.

BASIC : GOSUB (numéro de ligne)
Signification : Branchement vers un sous-programme dont le numéro de la première ligne est spécifié. Le sous-programme doit se terminer par une instruction RETURN.
Code machine : CALL (adresse mémoire)
Signification : Branchement vers un sous-programme commençant à l'adresse spécifiée. Comme les sous-programmes en BASIC, ceux écrits en langage machine doivent se terminer par une instruction RET.
BASIC : IF A=5 THEN (GOTO/GOSUB/LET/etc.)
Signification : Si A=5, alors exécuter l'instruction ou le branchement spécifié par la clause THEN.

En langage machine, l'équivalent des instructions IF...THEN existe, mais celles-ci ne sont pas implantées de la même manière qu'en BASIC. La condition IF se rapporte toujours, en langage machine, au résultat de la dernière opération qui a été effectuée. L'équivalent typique d'une instruction IF...THEN est :

CALL Z, (adresse mémoire spécifiée)

Cette instruction signifie que, si le résultat du dernier calcul est zéro, le sous-programme se trouvant à l'adresse spécifiée doit être appelé.

La liste des codes machine équivalant aux instructions IF...THEN les plus fréquemment utilisées est donnée ci-dessous :

CALL NZ,nn
Si le résultat du dernier calcul effectué est différent de zéro, le sous-programme se trouvant à l'adresse nn est appelé.

CALL M,nn
Si le résultat du dernier calcul effectué est négatif, le sous-programme se trouvant à l'adresse nn est appelé.

CALL P,nn
Si le résultat du dernier calcul effectué est positif, le sous-programme se trouvant à l'adresse nn est appelé.

JP Z,nn
Si le résultat du dernier calcul effectué est zéro, l'exécution du programme est déroutée vers l'adresse nn.

JP NZ,nn
Si le résultat du dernier calcul effectué est différent de zéro, l'exécution du programme est déroutée vers l'adresse nn.

JP M,nn

Si le résultat du dernier calcul effectué est négatif, l'exécution du programme est déroutée vers l'adresse nn.

JP P,nn

Si le résultat du dernier calcul effectué est positif, l'exécution du programme est déroutée vers l'adresse nn.

BASIC : FOR A=1 TO 100
Instruction(s) devant être exécutée(s) 100 fois.
NEXT A
Signification : Effectuer 100 fois la boucle.
Code machine : LD A,100
Instruction(s) devant être exécutée(s) 100 fois.
SUB A,1
JP NZ, adresse de la première instruction de la boucle.

Ainsi, pour simuler la boucle FOR/NEXT indiquée, la valeur 100 doit tout d'abord être chargée dans le registre A. Cette valeur représente le nombre de fois où les instructions se trouvant à l'intérieur de la boucle doivent être exécutées. Après chaque exécution de la série d'instructions, la valeur 1 est soustraite au contenu du registre A. Si le résultat de cette soustraction est NZ (Non Zéro), l'exécution est redirigée vers la première instruction de la boucle. Le processus se poursuit jusqu'à ce que la soustraction d'une unité au contenu du registre A donne pour résultat zéro, ce qui signifie alors que la boucle a été exécutée autant de fois qu'il était spécifié.

Les instructions BASIC "PEEK" et "POKE" sont, dans ce langage évolué, celles qui se rapprochent le plus des codes machine. Elles permettent en effet un accès direct au contenu de la mémoire.

BASIC : LET A=PEEK(40000)
Signification : Assigner à la variable A la valeur contenue à l'adresse mémoire 40000.
Code machine : LD A,(40000)
Signification : Charger la valeur se trouvant à l'adresse 40000 dans le registre A.

Si la valeur 81 est stockée à l'adresse 40000 et que le code LD A,(40000) est exécuté, la valeur 81 sera chargée dans le registre A. Ce registre est le seul registre auquel il soit possible d'assigner ainsi directement une valeur se trouvant à une adresse mémoire. Par exemple, l'instruction suivante est incorrecte :

LD D,(40000)

Par contre, les trois registres doubles peuvent être utilisés de cette manière pour simuler une instruction PEEK.

Code machine : LD HL,(40000)

Cette instruction a pour effet de charger dans le registre L la valeur se trouvant à l'adresse 40000 et dans le registre H la valeur se trouvant à l'adresse 40001. Cela découle naturellement de la manière dont les nombres supérieurs à 255 sont stockés en mémoire centrale de l'ordinateur.

Si les valeurs 5 et 15 sont respectivement stockées aux adresses 40000 et 40001, quelle sera la valeur contenue dans le registre HL après que le code machine LD HL,(40000) aura été exécuté ?

BASIC : POKE(40000),A
Signification : Charger la valeur de la variable A à l'adresse 40000.
Code machine : LD (40000),A
Signification : Charger la valeur se trouvant dans le registre A à l'adresse 40000.

Le registre A est le seul registre simple dont la valeur puisse être chargée directement à un emplacement mémoire quelconque. Par contre, les trois registres doubles peuvent être ainsi utilisés pour simuler l'action d'une instruction POKE.

Code machine : LD (4000),HL
Signification : Charger à l'adresse 40000 la valeur contenue dans le registre L et à l'adresse 40001 la valeur contenue dans le registre H.

Si le registre HL contient la valeur 35621, quel sera le contenu des adresses mémoire 40000 et 40001 après que le code machine LD (40000),HL aura été exécuté ?

BASIC : LET A=5
LET B=40000
POKE(B),A
Signification : Charger la valeur de la variable A à l'adresse mémoire spécifiée par B.
Codes machine : LD A,5
LD HL,40000
LD (HL),A
Signification : Charger la valeur contenue dans le registre A à l'adresse mémoire correspondant à la valeur contenue dans HL.

BASIC : LET B=40000
LET A=PEEK(B)
Signification : Assigner à la variable A la valeur se trouvant à l'adresse spécifiée par B.
Codes machine : LD HL,40000
LD A,(HL)
Signification : Charger dans le registre A la valeur se trouvant à l'adresse correspondant à la valeur contenue dans HL.

Le registre A est le seul registre simple qui puisse être ainsi utilisé avec une adresse spécifiée par le contenu de l'un des trois registres doubles. Les combinaisons permises sont donc les suivantes :

```
LD A,(HL)
LD A,(BC)
LD A,(DE)
```

Les six autres registres simples ne peuvent être ainsi chargés que lorsque l'adresse est spécifiée par le registre double HL, au moyen des combinaisons suivantes :

```
LD B,(HL)
LD C,(HL)
LD D,(HL)
LD E,(HL)
LD H,(HL)
LD L,(HL)
```

Codes machine : PUSH
POP

Ces deux instructions n'ont pas vraiment d'équivalent en BASIC. Elles sont par contre essentielles pour la programmation en langage machine et l'exemple suivant va permettre d'illustrer leur fonction. Supposons qu'un programme utilise les sept registres de la manière suivante :

```
HL
BC
DE
A
```

Si l'on voulait exécuter plusieurs fois cette suite d'instructions en BASIC, on utiliserait une boucle FOR/NEXT. Comme indiqué précédemment, une telle boucle peut être simulée par des instructions en langage machine. Dans l'exemple présent, ces instructions sont les suivantes :

```
                        LD A,8 (nombre de fois où la boucle doit être
                               répétée)
(début du sous-programme) HL
                        BC
                        DE
                        A
                        SUB A,1
                        JP NZ, (début du sous-programme)
                        END
```

En l'état, un tel programme ne tournerait pas. En effet, cela revient à définir une boucle FOR/NEXT telle que FOR A=1 TO 8 et à utiliser la variable A non seulement comme compteur, mais aussi comme paramètre à l'intérieur de la boucle. Dans ces conditions, la valeur de A contrôlant la boucle serait modifiée par son utilisation à l'intérieur de celle-ci. En BASIC, un tel problème est facilement évité car un très grand nombre de variables différentes sont disponibles. Il n'en est pas de même en langage machine et les instructions PUSH et POP sont destinées à surmonter cette difficulté.

Le sous-programme présenté ci-dessous montre comment ces instructions doivent être utilisées pour rendre opérant le sous-programme précédent :

```
                        LD A,8
(début du sous-programme) PUSH A
                        HL
                        BC
                        DE
                        A
                        POP A
                        SUB A,1
                        JP NZ, (début du sous-programme)
```

Après que la valeur 8 a été chargée dans le registre A, l'instruction PUSH A est exécutée. Cela signifie que la valeur courante de ce registre est mise de côté, dans un endroit appelé "pile". Les instructions suivantes peuvent alors être exécutées sans risque de perdre cette valeur particulière. Une fois que le sous-programme a été exécuté, l'instruction POP A permet de réaffecter au registre A sa valeur originale. PUSH est appelée instruction d'"empilage" et POP instruction de "dépilage".

On peut ainsi empiler ou dépiler n'importe lequel des registres doubles. Par contre, ces opérations ne peuvent être effectuées sur les registres simples, contrairement à ce que peut laisser croire l'exemple donné ci-dessus. Celui-ci était uniquement destiné à éviter toute confusion, mais le code machine PUSH A n'existe pas. Par contre, il est possible d'empiler un registre double nommé AF. Le registre F est un registre particulier qui ne peut être utilisé de la même manière que les sept autres registres simples. Il est cependant possible d'écrire des programmes très élaborés en langage machine sans avoir recours à ce registre F.

La fonction première du registre F est de servir au microprocesseur Z80 à indiquer le résultat de différents calculs. En fonction de la valeur contenue dans ce registre, le microprocesseur peut par exemple indiquer si le résultat du dernier calcul effectué était nul, positif ou négatif, ou s'il comportait une retenue. C'est précisément cette retenue (ou report) qui fait la différence entre les codes machine SUB (Soustraction) et SBC (Soustraction avec retenue). Si l'indicateur de report est positionné, la retenue est prise en compte dans la soustraction effectuée par l'instruction SBC. L'utilisation du registre F est assez délicate pour les programmeurs peu familiers du langage machine. Aussi ne sera-t-elle pas davantage évoquée dans la suite de cet ouvrage.

3

STOCKAGE DES CODES MACHINE EN MÉMOIRE

Jusqu'à présent, seule la manière dont les nombres sont stockés en mémoire a été décrite. Il a été cependant mentionné que les codes machine étaient représentés par un nombre unique ou par une combinaison de nombres. En pratique, cela signifie que certains codes machine sont représentés par un seul nombre compris entre 0 et 255 et que d'autres le sont par deux nombres compris entre les mêmes limites.

Les nombres représentant les codes machine sont stockés en mémoire de manière tout à fait identique à celle utilisée pour le stockage de n'importe quel autre nombre. Certains codes machine nécessitent un seul emplacement mémoire (c'est-à-dire un seul octet), d'autres en nécessitent deux.

Si la première instruction d'un programme était INC A (incrémenter le registre A), la première chose à connaître serait la valeur du numéro correspondant à cette instruction. Il s'agit en fait de 60. On aurait ensuite besoin de savoir à quelle adresse mémoire doit commencer ce programme. Supposons que ce soit l'adresse 40000. La valeur 60 doit donc être chargée à l'adresse 40000. (La procédure à suivre pour réaliser cette opération sera expliquée un peu plus loin.) Une fois le premier code machine chargé en mémoire centrale, le numéro correspondant à la seconde instruction, soit par exemple 52 pour le code INC HL, doit être chargé à l'adresse suivante, soit ici 40001.

La totalité du programme est ainsi construite pas à pas en stockant aux adresses consécutives les numéros correspondant aux instructions successives. Les deux codes machine donnés en exemple s'étendent sur un seul octet, c'est-à-dire qu'ils n'occupent chacun qu'un seul emplacement mémoire. Cependant, d'autres instructions telles que ADD A,5 s'étendent sur deux octets. Le premier emplacement mémoire contient le numéro correspondant à l'instruction ADD A, tandis que l'emplacement mémoire suivant contient la valeur qui doit être ajoutée au contenu du registre A. Si ADD A,5 était la première instruction d'un programme commençant à l'adresse 40000, la valeur 198 (qui est le numéro correspondant au code machine ADD A) devrait être chargée à l'adresse 40000, la valeur 5 étant chargée à l'adresse 40001.

Lorsque le programme est lancé, le microprocesseur examine le contenu de la première adresse mémoire, en l'occurrence ici ADD A. Il regarde ensuite, à l'adresse 40001, quelle est la valeur qui doit être ajoutée à celle du registre A. Une fois l'addition effectuée, le microprocesseur examine le code machine de l'instruction se trouvant à l'adresse suivante, soit ici 40002.

Certains codes machine s'étendent eux-mêmes sur deux octets et doivent être suivis d'un ou deux arguments occupant chacun un octet. Ces instructions en langage machine s'étendent donc sur trois ou quatre octets au total. Là encore, ces octets doivent être stockés à des adresses consécutives en mémoire centrale.

Une différence essentielle par rapport au BASIC réside dans le fait que les instructions du langage machine ne sont pas précédées d'un numéro de ligne. Elles sont stockées à des adresses consécutives en mémoire centrale. Le microprocesseur garde en permanence un pointeur sur l'adresse de l'instruction qu'il est en train d'exécuter. Le pointeur se déplace à l'adresse de l'instruction suivante une fois que la première a été exécutée. En dépit de l'absence de numéros de lignes, il est néanmoins possible d'avoir l'équivalent des instructions BASIC de déroutement GOTO et GOSUB. Au lieu d'être dirigée vers un numéro de ligne, l'exécution du programme est déroutée vers une adresse mémoire.

Lorsque l'on écrit un programme en langage machine, les nombres ne sont pas tapés en utilisant le système de notation décimale classique. A la place est utilisée une notation dite " hexadécimale " (HEX en abrégé), c'est-à-dire une notation se rapportant au système à base 16. Les quelques exemples ci-dessous illustrent la correspondance entre les systèmes décimal et hexadécimal :

Décimal	Hexadécimal
0	00
9	09
10	0A
15	0F
16	10
255	FF

Un tableau complet de conversion est donné en annexe. Dans cet ouvrage, afin d'éviter toute confusion, les nombres exprimés en notation décimale sont suivis de la lettre d et ceux exprimés en notation hexadécimale sont suivis de la lettre h :

8d = 8 décimal
12h = 12 hexadécimal

> Quel est l'équivalent décimal de E3h ?
>
> Si FBh est la partie de poids fort d'un nombre et CBh sa partie de poids faible, quel est ce nombre exprimé en décimal ?

Le tableau de conversion décimal/hexadécimal donné à la fin de cet ouvrage peut être utilisé pour répondre à ces questions.

La conversion d'un nombre d'un système en un autre devient très facile lorsque l'on possède un minimum de pratique. Un des avantages du système hexadécimal réside dans sa plus grande facilité d'utilisation en informatique. Deux chiffres hexadécimaux suffisent pour représenter n'importe quel nombre décimal compris entre 0 et 255 (soit des nombres exprimés par un, deux ou trois chiffres décimaux).

Il existe plusieurs méthodes permettant d'entrer des nombres ou des codes machine en mémoire centrale de l'ordinateur. Le programme BASIC présenté ci-dessous peut être utilisé pour entrer et vérifier très rapidement un programme écrit en langage machine. Ce programme BASIC doit être soigneusement tapé et, après avoir vérifié qu'il ne contient pas d'erreur de recopie, être sauvegardé sur cassette au moyen de l'instruction :

SAVE "CAS:ENTHEX"

```
10 CLEAR 200,39999
15 CLS
20 LOCATE 0,0
25 PRINT "Vérouillez les majuscules"
30 LOCATE 0,4
35 PRINT "Tapez la touche E pour entrer un code hexadécimal"
40 LOCATE 0,8
45 PRINT "Tapez la touche C pour vérifier un code hexadécimal"
50 LOCATE 0,12
55 PRINT "Tapez la touche'X pour vérifier tous les codes entrés"
60 LOCATE 0,16
65 PRINT "Tapez la touche Q pour arrêter"
70 A$=INKEY$
75 IF A$="E" THEN 185
80 IF A$="C" THEN 380
85 IF A$="X" THEN 100
90 IF A$="Q" THEN STOP ELSE 70
100 CLS
105 LOCATE 0,0
110 INPUT "Entrez l'adresse de départ";AS
120 LOCATE 0,5
125 INPUT "Entrez la dernière adresse";AE
135 LET D=0
140 FOR I=AS TO AE
145 LET D=D+PEEK(I)
150 NEXT I
155 CLS
160 PRINT "Totalisateur=";D
165 LOCATE 0,20
170 PRINT "Tapez la touche M pour retourner au menu"
175 IF INKEY$ <> "M" THEN 175 ELSE 15
185 CLS
190 LOCATE 0,0
195 PRINT "Vérouillez les majuscules"
200 LOCATE 0,4
205 INPUT "Entrez l'adresse de départ";S
215 IF S<40000 THEN 370
220 LET A$=""
225 LOCATE 0,23
230 LET ET=S
235 IF A$="" THEN INPUT A$
240 LET BAD=0
245 IF A$="M" THEN 15
250 LET E=LEN(A$)-1
260 LET C$=A$
265 FOR D=1 TO E STEP 2
```



```
270 LET B$=LEFT$(C$,2)
275 LET C=VAL("&H"+B$)
280 IF C=0 THEN GOSUB 360
285 LET C$=MID$(C$,3)
290 NEXT D
295 IF BAD=1 THEN 345
300 LOCATE 0,21:PRINT S;" "A$
305 LET B$=LEFT$(A$,2)
310 IF LEN(B$)=1 THEN 345
315 LET C=VAL("&H"+B$)
320 POKE (S),C
325 LET S=S+1
330 LET A$=MID$(A$,3)
335 IF A$="" THEN 230 ELSE 305
345 LOCATE 0,22:PRINT "Entrée incorrecte.Essayez de nouveau"
350 LET S=ET
355 GOTO 220
360 IF B$<>"00" THEN LET BAD=1
365 RETURN
370 PRINT "L'adresse initiale doit être supérieure ou égale à
    40000"
375 GOTO 205
380 LET ND=0
385 CLS
390 LOCATE 0,0
395 INPUT "Entrez l'adresse de départ";AS
405 LOCATE 0,5
410 INPUT "Entrez la dernière adresse";AE
420 CLS
425 IF AS+20>AE THEN 490
430 FOR C=AS TO AS+20
435 IF PEEK(C)<16 THEN 480
440 PRINT C;"";HEX$(PEEK(C))
445 NEXT C
450 IF ND=1 OR C>AE THEN 505
460 PRINT "Tapez M si vous souhaitez continuer"
465 LET AS=C
470 IF INKEY$ <> "M" THEN 470 ELSE 425
480 PRINT C;"0";HEX$(PEEK(C))
485 GOTO 445
490 FOR C=AS TO AE
495 LET ND=1
500 GOTO 435
505 PRINT "Tapez M pour revenir au menu"
510 IF INKEY$ <> "M" THEN 510 ELSE 15
```

Pour charger ce programme à partir de la cassette, il suffit de taper la commande :

LOAD "CAS:",r

Cette commande permet de commencer automatiquement l'exécution du programme une fois celui-ci chargé en mémoire centrale. Le menu suivant est alors affiché :

Tapez la touche E pour entrer un code hexadécimal
Tapez la touche C pour vérifier un code hexadécimal
Tapez la touche X pour vérifier tous les codes entrés
Tapez la touche Q pour arrêter

Pour entrer une instruction en code machine, il suffit donc de taper la touche E. Un message indiquant de verrouiller les majuscules est alors affiché.

Le court programme en langage machine donné ci-dessous permettra au lecteur de tester si le programme BASIC " ENTHEX " a été correctement recopié. Ce programme de démonstration additionne deux nombres et stocke le résultat à l'adresse 40100. La marche à suivre est la suivante (en réponse aux messages affichés à l'écran) :

1. Taper la touche E.
2. Appuyer sur la touche de verrouillage des majuscules.
3. L'adresse de départ à entrer est 40000.
4. Les codes hexadécimaux peuvent maintenant être entrés. Le premier est 3E05 (presser ensuite la touche retour chariot). Cela correspond aux deux octets se trouvant aux adresses 40000 et 40001.
5. Entrer de la même manière les trois autres lignes, en tapant un retour chariot à la fin de chacune d'elles.
6. Après que la dernière ligne a été entrée (et le retour chariot activé), il suffit de taper M (toujours suivi d'un retour chariot) pour revenir au menu.

Adresse de départ **40000**
Adresse finale **40001**
Total hexadécimal **852**

```
3E05      LD A,5
C610      ADD A,16
32A49C    LD (40100),A
C9        RET
```

Il faut ensuite tester que les codes ont été correctement entrés. Pour ce faire, il suffit de taper C (à l'affichage du menu). Il est alors demandé d'entrer l'adresse de départ du code machine qui doit être testé, c'est-à-dire ici 40000. L'adresse finale est ensuite demandée, soit dans ce cas 40007. Les emplacements mémoire et leur contenu sont alors affichés à l'écran. Ceux-ci doivent être soigneusement vérifiés. Si une erreur est décelée, il suffit de revenir au menu et d'entrer à nouveau le code machine correct.

L'activation de la touche X permet, lorsque le menu est affiché, d'effectuer une vérification supplémentaire. De nouveau, les adresses initiale et finale doivent être fournies. Le programme ENTHEX additionne alors le contenu des valeurs se trouvant à toutes ces adresses et affiche le total, soit dans cet exemple 852. Si ce n'est pas le cas, c'est qu'un

code est incorrect ou que, hypothèse moins probable, le programme ENTHEX a été mal tapé. Le problème doit être résolu avant de poursuivre.

Toutes ces vérifications sont essentielles car, contrairement à ce qui se passe en BASIC où l'exécution du programme s'interrompt lorsqu'une erreur est rencontrée, une erreur dans un programme écrit en langage machine entraîne l'ordinateur dans des boucles sans fin dont le seul moyen de sortir est de couper l'alimentation de la machine.

Afin de tester le petit programme en langage machine qui vient d'être écrit, il suffit de sortir du programme ENTHEX en répondant Q au choix proposé par le menu. Les deux lignes suivantes doivent ensuite être écrites :

```
1000 def usr=40000
1005 a=usr (1)
```

Ces deux lignes ont une signification équivalente de celle de l'instruction GOSUB, la seule différence étant qu'ici le sous-programme appelé est écrit en langage machine. On peut d'ailleurs remarquer que la dernière instruction de ce sous-programme est une instruction RET, équivalente de RETURN en BASIC. Lorsque le microprocesseur rencontre cette instruction, il renvoie l'exécution à l'instruction d'appel, c'est-à-dire ici au programme BASIC.

Pour exécuter le sous-programme en langage machine, il suffit de taper, en mode direct, la commande GOTO 1000, suivie de PRINT PEEK (40100). Le nombre 21 doit alors s'afficher ; il correspond au résultat de l'addition de 16 et 5.

La procédure à utiliser pour entrer et vérifier tous les programmes en langage machine se trouvant dans cet ouvrage est identique à celle qui vient d'être détaillée ici. Elle ne sera donc pas expliquée chaque fois. Il est important de toujours noter les adresses initiale et finale de ces programmes, ainsi que la somme des codes hexadécimaux mis en jeu. Ces indications sont données en tête de chaque listing. De même, les tests à effectuer sont toujours identiques à ceux qui viennent d'être décrits. D'autre part, quelques lignes de programme BASIC peuvent également être indiquées en tête de chaque listing ; elles commencent toujours à la ligne 1000 et sont exécutables au moyen de la commande GOTO 1000. Lorsque les tests ont été effectués, le programme ENTHEX peut être réinitialisé au moyen de la commande RUN.

4

L'AFFICHAGE VIDÉO

Les ordinateurs MSX disposent de quatre modes écran différents :

MODE 0 24 LIGNES, 40 CARACTÈRES PAR LIGNE
MODE 1 24 LIGNES, 32 CARACTÈRES PAR LIGNE
MODE 2 MODE GRAPHIQUE HAUTE RÉSOLUTION
MODE 3 MODE MULTICOULEUR

Chacun de ces modes possède des caractéristiques particulières, le choix de l'un d'entre eux dépendant de la nature du programme exécuté.

Dans le reste de cet ouvrage, on supposera, sauf spécification contraire, que le mode 1 a été sélectionné. Un chapitre particulier est cependant consacré à la description des autres modes écran. Le mode 1 est probablement celui qui est le mieux adapté à la réalisation de programmes de jeux.

La question essentielle à laquelle va tenter de répondre ce chapitre est : "Comment l'image affichée à l'écran est-elle construite ?" Les caractères qui composent le texte que l'on souhaite afficher sont stockés en mémoire sous forme de nombres. L'ordinateur examine constamment la zone mémoire contenant ces nombres et effectue les opérations nécessaires pour les transformer en une image vidéo.

Il a été précédemment mentionné que les programmes écrits en BASIC ou en langage machine étaient stockés dans une zone particulière de la mémoire centrale de l'ordinateur, appelée mémoire RAM. Il existe, sur les ordinateurs MSX, une autre zone mémoire destinée au stockage des informations concernant l'affichage écran. Cette zone est appelée mémoire vidéo RAM, ou VRAM. Elle s'étend sur 16 384 octets.

La mémoire VRAM est utilisée pour stocker les données concernant l'image à afficher, ainsi que d'autres informations telles que la taille des caractères et celle des *sprites* (le chapitre suivant est consacré à la description des sprites). En mode écran 1, un tableau de 32×24=768 octets est défini en mémoire VRAM. Chaque élément de ce tableau correspond à une position de caractère à l'écran (32 colonnes × 24 lignes = 768 positions). Le nombre 65 code pour la lettre A ; si le premier élément (c'est-à-dire le premier octet) du tableau contient le numéro 65, la lettre A sera affichée dans le coin supérieur gauche de l'écran. Pour comprendre plus précisément le processus, il convient tout d'abord de connaître l'adresse, en mémoire VRAM, du premier des 768 octets consécutifs du tableau. Pour ce faire, il suffit de taper les deux lignes suivantes :

```
SCREEN 1
PRINT BASE(5)
```

La première instruction sélectionne le mode écran 1. La seconde affiche l'adresse de la base (c'est-à-dire l'adresse du premier octet) de

ce tableau. Cette adresse particulière sera par la suite appelée INITECRAN. Les ordinateurs MSX initialisent cette adresse à 6144. Pour charger la valeur 65 à l'adresse INITECRAN, il suffit de taper l'instruction :

```
VPOKE(INITECRAN),65
```

(Le mot INITECRAN ne doit pas être tapé, mais remplacé par sa valeur, par exemple 6144 si l'adresse de la base de ce tableau n'a pas été modifiée par l'utilisateur). La lettre A apparaîtra alors dans l'angle supérieur gauche de l'écran. Il est maintenant très simple de calculer l'adresse de n'importe quelle autre position d'affichage sur l'écran, sachant qu'un octet du tableau correspond à une position de caractère. Par exemple, l'adresse définissant l'affichage dans l'angle supérieur droit de l'écran est INITECRAN+31. Il est à noter que sur certains téléviseurs l'image n'est pas toujours parfaitement centrée de sorte que les angles de l'image ne sont pas toujours visibles.

N'importe quel nombre compris entre 0 et 255 peut être ainsi chargé, au moyen de l'instruction VPOKE, en l'un quelconque des 768 octets formant le tableau. Le caractère correspondant au nombre sera ainsi affiché à une position définie par l'adresse de l'octet dans le tableau. La mémoire VRAM contient également un autre tableau dans lequel sont stockées les données correspondant aux formes des caractères. Chaque forme est définie sur 8 octets. Comme 256 caractères différents sont disponibles, le tableau de formes s'étend sur 8×256=2 048 octets. L'adresse de la base de ce tableau en mémoire VRAM peut être connue en tapant :

```
PRINT BASE(7)
```

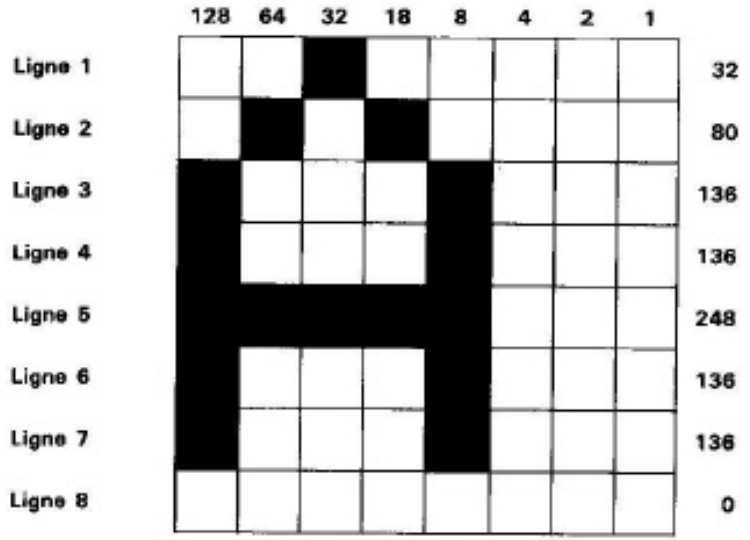
En mode écran 1, les ordinateurs MSX initialisent ce tableau de formes à zéro, qui doit donc être la valeur affichée lorsque l'instruction ci-dessus est exécutée (sauf si l'adresse de la base de ce tableau a été modifiée par l'utilisateur). Cette adresse d'initialisation sera par la suite désignée par CARINIT. L'adresse du premier octet codant pour la forme de l'un quelconque des caractères est donc donnée par la formule :

```
(Numéro du caractère × 8) + CARINIT
```

Comment une forme de caractère est-elle définie et stockée en mémoire VRAM ? Considérons par exemple la lettre A qui a pour numéro 65 et dont le premier octet de forme se trouve donc, en mémoire VRAM, à l'adresse suivante :

```
(65 × 8) + CARINIT = 520
```

Chacune des formes de caractère peut être visualisée sur une grille de 8×8, comme le montre l'exemple ci-dessous, dans le cas de la lettre A.



Chacune des huit lignes horizontales représente un octet. Chaque ligne est convertie en un nombre dont la valeur dépend de la répartition des carrés noirs et blancs qui la composent (voir en annexe le paragraphe consacré à la notation binaire). Ce sont ces huit nombres (32, 80, 136, ..., 0) qui sont stockés en mémoire VRAM et qui définissent la forme de la lettre A. L'exemple suivant montre comment ces données de formes sont stockées :

```
VPOKE(INITECRAN+5),65
```

Cette instruction permet d'afficher la lettre A à l'écran. Supposons que nous voulions modifier la forme de cette lettre. Pour cela, les lignes suivantes peuvent être entrées en mode direct. Leur effet sur la forme du caractère affiché peut être visualisé après activation de la touche retour chariot (CARINIT doit être remplacé par l'adresse d'initialisation de la base du tableau de formes, soit habituellement 0) :

```
VPOKE(CARINIT+520),255
VPOKE(CARINIT+521),129
VPOKE(CARINIT+522),129
VPOKE(CARINIT+523),129
VPOKE(CARINIT+524),129
VPOKE(CARINIT+525),129
VPOKE(CARINIT+526),129
VPOKE(CARINIT+527),255
```


La forme du caractère n° 65 (lettre A) a été ainsi complètement redessinée. La procédure utilisée est exactement la même que celle qui doit être mise en œuvre pour créer un caractère qui n'est pas normalement défini dans le jeu de caractères MSX. En annexe, un programme est donné pour permettre de redessiner très rapidement un jeu complet de caractères et le sauvegarder sur cassette pour une utilisation dans d'autres programmes.

Un troisième tableau, le tableau des couleurs, est initialisé en mémoire VRAM. Il s'étend sur 32 octets, chaque octet codant pour la couleur d'un bloc de huit caractères. Il est impossible de définir différentes couleurs à l'intérieur d'un même bloc, de sorte que, lorsqu'une valeur (correspondant à un numéro de couleur) a été assignée à un octet, les huit caractères du bloc correspondant sont affichés dans cette couleur. L'adresse de la base du tableau des couleurs peut être connue au moyen de l'instruction :

```
PRINT BASE(6)
```

En mode écran 1, l'adresse de la base de ce tableau est initialisée à 8192, en mémoire VRAM. Cette adresse sera désignée par COLINIT. Pour illustrer la manière dont le tableau des couleurs contrôle l'affichage des caractères, les deux lignes suivantes peuvent être tapées :

```
SCREEN 1
VPOKE(INITECRAN+5),65
```

La lettre A s'affiche alors à l'écran. Pour modifier l'octet codant pour la couleur dans laquelle est affiché ce caractère, il faut commencer par calculer l'adresse de cet octet de couleur, soit :

```
COLINIT + (NUMÉRO DU CARACTÈRE / 8)
```

Seule la partie entière du résultat de cette opération est à retenir. Si l'adresse d'initialisation de la base du tableau des couleurs n'a pas été modifiée par l'utilisateur, le résultat est :

```
8192 + (65 / 8) = 8200
```

Dans ces conditions, la couleur d'affichage de la lettre A peut être modifiée au moyen d'une instruction VPOKE :

```
VPOKE(8200),241
```

La lettre A est ainsi affichée en blanc sur fond noir.

```
VPOKE(8200),159
```

La lettre A est alors affichée en rouge sur fond blanc.

Seize couleurs différentes sont disponibles, chaque couleur étant identifiée par un numéro :

0 transparent
1 noir
2 vert
3 vert clair
4 bleu foncé
5 bleu clair
6 rouge foncé
7 cyan
8 rouge
9 rouge clair
10 jaune foncé
11 jaune clair
12 vert foncé
13 magenta
14 gris
15 blanc

La valeur de l'argument à fournir à l'instruction VPOKE pour définir les couleurs choisies peut être calculée de la manière suivante :

- Multiplier par 16 le numéro de la couleur d'avant-plan choisie et ajouter à ce résultat le numéro de la couleur de fond.
- Pour afficher, par exemple, un caractère magenta sur fond blanc, l'argument doit avoir pour valeur :

$$(13 \times 16) + 15 = 223$$

5

LES SPRITES

Les sprites constituent une des particularités des ordinateurs MSX. Supposons qu'un programme de jeu ait été conçu et qu'il nécessite le déplacement d'une figure sur l'écran. En l'absence de sprite, il serait nécessaire d'afficher une forme définie en une position particulière de l'écran, de l'effacer, puis de redéfinir la forme au niveau d'une autre position de l'écran, etc. Cette procédure est non seulement fastidieuse, mais également peu performante au niveau de la vitesse de déplacement qu'il est ainsi possible de programmer.

La possibilité de définir des sprites facilite considérablement la conception de programmes nécessitant le déplacement de motifs graphiques prédéfinis. Imaginons qu'un sprite soit simplement un caractère quelconque; il est possible de définir sa forme et sa couleur. Un sprite peut être affiché en n'importe quelle position de l'écran, pas nécessairement en un emplacement prédéterminé pour un caractère. Les sprites n'ont pas besoin d'être effacés de leur position antérieure pour simuler l'impression de mouvement. Le déplacement d'un sprite, en BASIC comme en langage machine, ne nécessite que la modification du contenu d'un emplacement mémoire.

32 sprites au maximum peuvent être affichés simultanément à l'écran. Il existe cependant une limitation supplémentaire qui sera évoquée plus loin. Les sprites peuvent être affichés dans quatre tailles différentes. Cependant, tous les sprites se trouvant à l'écran à un instant donné doivent nécessairement avoir la même taille. Dans l'exemple ci-dessous, la taille minimale a été choisie, c'est-à-dire une taille correspondant à celle dans laquelle sont affichés les caractères habituels. Les 32 sprites sont numérotés de 0 à 31. Pour positionner la taille de sprite sélectionnée, il suffit de taper :

```
VDP (1) = 224
```

Les autres tailles de sprites disponibles seront décrites dans un prochain chapitre. Il existe, en mémoire VRAM, un tableau dans lequel sont stockées les informations concernant les formes qui ont été définies pour les sprites. L'adresse de la base de ce tableau peut être connue en tapant :

```
PRINT BASE (9)
```

Cette adresse est initialisée à 14336, en mode écran 1 (comme d'ailleurs dans les autres modes écran où les sprites peuvent être utilisés). Cette adresse sera désignée par FORMSPRIT. Comme les caractères habituels, les sprites de cette taille (minimale) sont définis sur huit octets, c'est-à-dire que les données de forme concernant le premier sprite sont stockées entre les adresses 14336 et 14343. Les quelques instructions suivantes montrent comment des valeurs peuvent être

chargées dans ces huit octets, afin que le sprite ainsi défini représente le dessin d'un "envahisseur de l'espace".

```
VPOKE FORMSPRIT + 0,60
VPOKE FORMSPRIT + 1,90
VPOKE FORMSPRIT + 2,255
VPOKE FORMSPRIT + 3,231
VPOKE FORMSPRIT + 4,128
VPOKE FORMSPRIT + 5,36
VPOKE FORMSPRIT + 6,66
VPOKE FORMSPRIT + 7,36
```

Ces instructions ne suffisent pas à afficher à l'écran la forme ainsi définie. Il faut aussi préciser en quel endroit de l'écran le sprite doit être positionné. Un tableau supplémentaire, en mémoire VRAM, contient les informations sur la position et la couleur des sprites. Ce tableau (tableau des attributs de sprites) contient, pour chacun des 32 sprites, quatre informations :

1. Sa position verticale
2. Sa position horizontale
3. Son numéro
4. Sa couleur.

L'adresse de la base du tableau des attributs de sprites peut être connue en tapant :

```
PRINT BASE (8)
```

Les ordinateurs MSX initialisent la base de ce tableau à l'adresse 6912. Cette adresse sera ici désignée par ATSPRIT. En conséquence, l'adresse du premier octet codant pour les quatre attributs correspondant à un sprite donné peut être calculée au moyen de la formule :

```
(NUMÉRO DE SPRITE × 4) + ATSPRITE
```

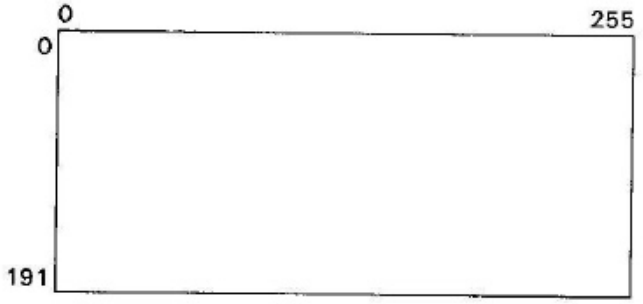
Par exemple, dans l'hypothèse où l'adresse de la base du tableau des attributs de sprites n'a pas été modifiée par l'utilisateur, le premier octet codant pour les attributs du sprite n° 15 se trouve à l'adresse :

```
(15 × 4) + 6912 = 6972
```

Les quatre lignes suivantes permettent de faire apparaître à l'écran le sprite préalablement défini :

```
VPOKE (ATSPRIT + 0) ,100
VPOKE (ATSPRIT + 1) ,50
VPOKE (ATSPRIT + 2) ,0
VPOKE (ATSPRIT + 3) ,10
```

Les paramètres se rapportant aux positions horizontale et verticale correspondent au coin supérieur gauche de chaque sprite. Pour positionner les sprites à l'écran, les repères suivants peuvent être définis :



Il est ainsi possible, en donnant différentes valeurs aux paramètres de position des instructions VPOKE, de déplacer un sprite en différentes parties de l'écran. Si le paramètre correspondant à la position verticale est supérieur à 191, le sprite disparaît progressivement derrière la bordure de l'écran. Il existe d'autre part un registre de collision qui est positionné lorsque deux sprites se superposent, au moins partiellement, à l'écran. Son utilisation sera illustrée dans le programme présenté au chapitre suivant.

Une limite importante existe quant aux possibilités d'affichage des sprites à l'écran. Bien que le nombre total de sprites affichés simultanément soit de 32, seuls quatre d'entre eux peuvent apparaître en même temps sur une même ligne horizontale de l'écran. Il faut toujours avoir cette limitation présente à l'esprit lorsque l'on écrit un programme, particulièrement un programme de jeu. Le diagramme ci-dessous montre la manière dont sont affichés les sprites supplémentaires lorsque cette limite est dépassé :

(La partie hachurée du sprite 5 n'apparaît pas à l'écran.)

La description de l'utilisation des sprites mériterait un ouvrage à elle seule (voir par exemple l'excellent livre de Mike Shaw) (*MSX guide du graphisme*, Sybex, 1985). En annexe est donné un programme permettant de dessiner des formes de sprites et de stocker ces formes sur cassette pour une utilisation dans d'autres programmes.

6

ÉCRITURE D'UN PROGRAMME EN LANGAGE MACHINE : LES ENVAHISSEURS DE L'ESPACE

Un ensemble de sous-programmes écrits en langage machine et dont la fonction est liée aux entrées/sorties du système BASIC (sous-programme BIOS) se trouvent en mémoire ROM des ordinateurs MSX. Ces sous-programmes peuvent être utilisés pour faciliter la conception d'un programme utilisateur écrit en langage machine. Les plus importants d'entre eux seront décrits en détail dans un prochain chapitre. Cependant, le programme proposé ici faisant appel à quelques-uns de ces sous-programmes, les indications indispensables seront déjà données à ce niveau.

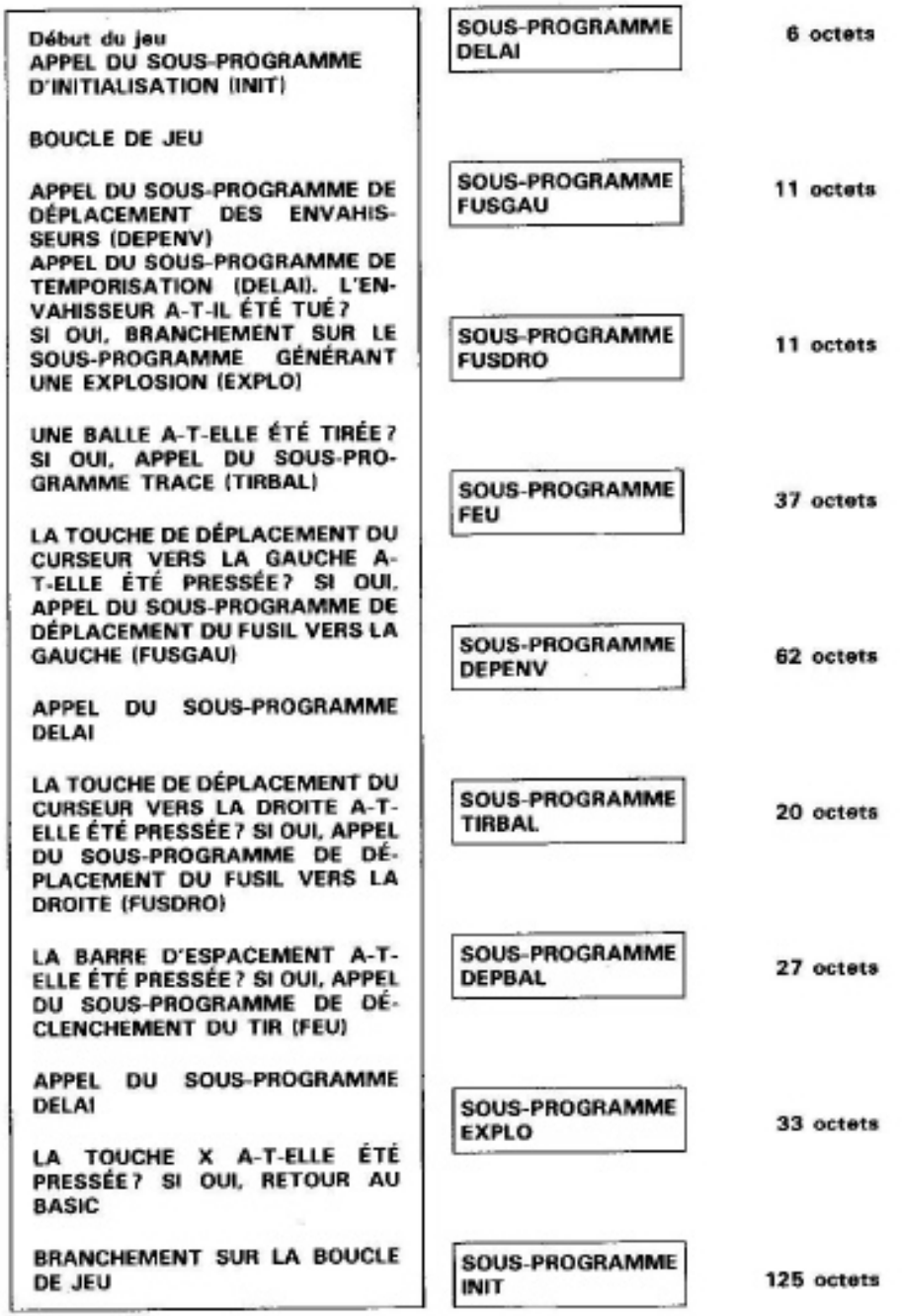
Le programme de jeu proposé ici est simple. Cependant, tous les principes mis en œuvre sont applicables à la réalisation de programmes plus sophistiqués. Un programme peut être décomposé en une succession de blocs, chaque bloc étant ensuite lui-même découpé en une suite de sous-programmes. Selon ce principe, le lecteur doit être capable, à la fin de ce chapitre, d'écrire ses propres programmes en langage machine.

Ce chapitre commence par une introduction à la notion d'organigramme, déjà familière aux lecteurs ayant pratiqué le langage BASIC. Il se poursuit par la description de l'organisation d'une carte mémoire. Cette étape, qui n'est pas nécessaire en BASIC, est indispensable lorsque l'on commence à écrire ses propres programmes en langage machine. Chacun des blocs de l'organigramme sera ensuite détaillé. L'un de ces blocs concerne par exemple le déplacement d'une balle à l'écran. La méthode mise en œuvre pour réaliser ce déplacement sera expliquée et l'organigramme d'un sous-programme interne sera également présenté à cette occasion.

Tous les sous-programmes ont été écrits afin de pouvoir être testés individuellement, ou en liaison avec d'autres sous-programmes qui l'ont déjà été. Le programme complet peut ainsi être examiné étape par étape et l'on peut voir quelle est exactement la fonction de chacun de ses sous-programmes constitutifs.

ORGANIGRAMME

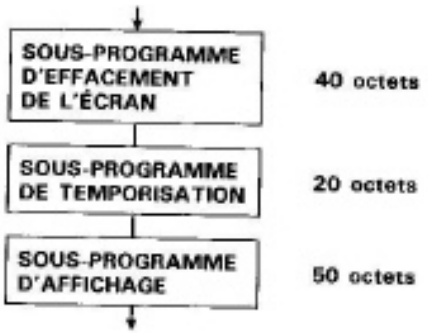
L'organigramme est une description graphique des différentes opérations devant être effectuées par le programme, le schéma devant en principe montrer l'articulation logique entre les différents blocs. L'organigramme présenté ci-dessous est dépouillé de toute symbolisation pour ne laisser apparaître que le découpage en blocs du programme " Les envahisseurs de l'espace ".



LA CARTE MÉMOIRE

Il a déjà été précisé qu'il n'existait pas de numéros de lignes dans un programme écrit en langage machine. Cela signifie que si l'on souhaite ajouter une instruction dans un programme déjà écrit, il est nécessaire de déplacer toutes les instructions se trouvant après celle que l'on veut insérer, ce qui nécessite de modifier les adresses absolues données comme arguments aux instructions telles que CALL et JUMP. Supposons par exemple qu'un sous-programme de temporisation DELAI ait été écrit à partir de l'adresse 40000. Pour accéder à ce sous-programme, le programme principal contient une instruction CALL 40000. Si, pour une raison ou pour une autre, une simple instruction de trois octets doit être ajoutée à la fin du sous-programme précédent se terminant normalement à l'adresse 39999, les adresses 40000, 40001 et 40002 devront être utilisées. Il est donc nécessaire de déplacer le sous-programme DELAI afin qu'il commence à l'adresse 40003. Cela fait, les instructions CALL 40000 se trouvant dans le programme devront être remplacées par CALL 40003. L'utilisation d'une telle méthode pour écrire un programme en langage machine serait extrêmement laborieuse et prendrait un temps considérable.

Ces difficultés peuvent être évitées en créant une carte des emplacements mémoire à partir de l'organigramme du programme. Cela nécessite de connaître le nombre approximatif d'octets occupés par chacun des sous-programmes. Les emplacements mémoire peuvent alors être alloués aux sous-programmes en laissant quelques octets libres entre la fin d'un sous-programme et le début du suivant. Ces quelques octets supplémentaires permettront d'étendre, si nécessaire, un sous-programme sans qu'il y ait besoin pour autant de modifier tout le programme. L'organigramme présenté ci-dessous montre comment créer une carte mémoire (dans le cas d'un programme très simple) :



Dans cet organigramme, la taille de chaque sous-programme est écrite en regard de ceux-ci. Le sous-programme d'effacement de l'écran

peut ainsi être chargé à partir de l'adresse 40000, le sous-programme de temporisation à partir de l'adresse 40050 et le sous-programme d'affichage à partir de l'adresse 40080. Quelques octets sont laissés libres entre deux sous-programmes consécutifs, facilitant ainsi leur extension éventuelle.

Dans cet esprit, la taille approximative de chacun des sous-programmes constitutifs du programme " Les envahisseurs de l'espace " est indiquée sur l'organigramme correspondant. A partir de ces informations, il est possible de construire la carte mémoire présentée ci-dessous. La colonne de gauche contient le nom de chaque sous-programme, celle de droite l'adresse de départ correspondante :

BCLJEU	**40000**
TIRBAL	**40082**
EXPLO	**40133**
FEU	**40157**
DEPBAL	**40205**
DEPENV	**40243**
FUSGAU	**40316**
FUSDRO	**40338**
INIT	**40363**
DELAI	**40474**

Une zone mémoire a également été assignée aux paramètres. La fonction de ceux-ci sera brièvement expliquée.

DÉFINITIONS

L'explication des quelques termes et expressions ci-dessous facilitera la description du programme.

1. **Bouclejeu** (bcljeu) : il s'agit de la boucle principale du programme. Cette boucle contrôle en particulier si les touches correspondant aux déplacements vers la droite ou vers la gauche ou au déclenchement du tir ont été activées. En conséquence, cette boucle effectue des appels répétés au sous-programme gérant le déplacement de l'envahisseur de l'espace. Une fois que toutes les fonctions ont été exécutées, le programme est redirigé sur la première instruction et le cycle recommence (d'où le nom bouclejeu).

2. **Indcol** : Il existe un emplacement particulier de la mémoire centrale de l'ordinateur appelé registre de collisions. Ce registre s'étend sur un seul octet (adresse 41000) et est normalement à zéro. Il est positionné à 1 lorsque la balle a touché l'envahisseur (c'est-à-dire lorsqu'il y a eu collision).

3. **Indicateur de balle tirée** (indbal) : Il s'agit d'un simple emplacement mémoire (un octet) qui est normalement positionné (c'est-à-dire qu'il a pour valeur 1). Lorsqu'une balle a été tirée, l'octet est remis à zéro. Cet indicateur, qui se trouve à l'adresse 41001, est utilisé pour éviter que plusieurs balles apparaissent simultanément à l'écran.

4. **Indicateur de direction de l'envahisseur** (inddir) : cet indicateur, qui s'étend sur un seul octet (adresse 41002), est positionné si l'envahisseur se déplace de droite à gauche. Il est à zéro si l'envahisseur se déplace de gauche à droite.

LE SOUS-PROGRAMME D'INITIALISATION

Ce sous-programme réalise les opérations suivantes :

- il positionne la taille des sprites de sorte que ceux-ci soient agrandis (d'un facteur 2);
- il positionne l'adresse de la base du tableau de formes des sprites (adresse FORMSPRIT) à 14336;
- il positionne l'adresse de la base du tableau des attributs de sprites (adresse ATSPRITE) à 6912;
- il affecte une valeur aux quatre paramètres gouvernant l'affichage des trois sprites qui ont été définis (c'est-à-dire l'envahisseur, la balle et la base de tir);
- il dessine les formes de ces trois sprites.

Les 24 octets nécessaires à la définition des formes de sprites sont chargés aux adresses 14336 à 14359 par le sous-programme d'initialisation.

Adresse initiale 40363
Adresse finale 40463
Total hexadécimal 6910

```
3E01      2050 INIT:    LD    A,1
32AFFC    2060          LD    (64687),A
CD5F00    2070          CALL  95
21EC9D    2080          LD    HL,donnée
110038    2090          LD    DE,14336
011800    2100          LD    BC,24
CD5C00    2110          CALL  92
0E05      2120          LD    C,5
0636      2130          LD    B,54
CD4700    2140          CALL  71
21049E    2150          LD    HL,donné1
11001B    2160          LD    DE,6912
```



```
010C00    2170          LD    BC,12
CD5C00    2180          CALL  92
3E01      2190          LD    A,1
32269E    2200          LD    (indcol),A
32279E    2210          LD    (indbal),A
32289E    2220          LD    (inddir),A
0E01      2230          LD    C,1
06E1      2240          LD    B,225
CD4700    2250          CALL  71
0E02      2260          LD    C,2
0606      2270          LD    B,6
CD4700    2280          CALL  71
C9        2290          RET
18181B7E  2300 Donnée   DEFB  24,24,126
FFFFFF00  2310          DEFB  255,255,255,0
3C7E99FF  2320          DEFB  60,126,153,255
663C4224  2330          DEFB  102,60,66,36
18181818  2340          DEFB  24,24,24,24
18181818  2350          DEFB  24,24,24,24
AA64000F  2360 Donné1   DEFB  170,100,0,15
0000010C  2370          DEFB  0,0,1,12
C8000201  2380          DEFB  200,0,2,1
```

Ce sous-programme d'initialisation peut être testé en tapant les quelques lignes écrites ci-dessous en BASIC :

```
1000 DEF USR=40363
1005 A=USR (1)
1010 GOTO 110
```

Le dessin de l'envahisseur s'affiche en vert dans le coin supérieur gauche de l'écran et celui du fusil est centré au bas de l'écran. Ce dernier apparaît en blanc.

LE SOUS-PROGRAMME DE DÉPLACEMENT DU FUSIL VERS LA GAUCHE ("FUSGAU")

Ce sous-programme est appelé lorsque la touche prévue à cet effet est activée. Il commence par tester si le fusil ne se trouve pas déjà au niveau du bord gauche de l'écran, auquel cas l'exécution est immédiatement retournée vers la boucle de jeu. Dans le cas contraire, le fusil est repositionné vers la gauche. Pour ce faire, la position horizontale courante du fusil est tout d'abord lue. Une unité est ensuite soustraite à cette valeur et le résultat de l'opération est chargé à l'adresse

correspondante en mémoire VRAM, c'est-à-dire celle qui définit la position horizontale du fusil.

Adresse initiale 40316
Adresse finale 40327
Total hexadécimal 1084

```
21011B    1720 FUSGAU  LD   HL,6913
CD4A00    1730         CALL 74
3D        1740         DEC  A
C8        1750         RET  Z
CD4D00    1760         CALL 77
C9        1770         RET
```

Les quelques lignes de programme BASIC ci-dessous permettent de tester ce sous-programme (le sous-programme INIT doit se trouver déjà en mémoire centrale de l'ordinateur) :

```
1000 DEF USR=40363
1005 A=USR (1)
1010 DEF USR=40316
1015 A=USR (1)
1020 FOR B=1 TO 100
1025 NEXT B
1030 GOTO 1015
```

Le fusil se déplace ainsi jusqu'à atteindre le bord gauche de l'écran.

LE SOUS-PROGRAMME DE DÉPLACEMENT DU FUSIL VERS LA DROITE ("FUSDRO")

Ce sous-programme est très semblable au précédent. Le test initial concerne cette fois la présence éventuelle du fusil à l'extrême droite de l'écran. D'autre part, une unité est ajoutée (et non soustraite) à la position horizontale courante.

Adresse initiale 40338
Adresse finale 40352
Total hexadécimal 1916

```
21011B    1880 FUSDRO  LD   HL,6913
CD4A00    1890         CALL 74
D6F0      1900         SUB  240
C8        1910         RET  Z
C6F1      1920         ADD  A,241
CD4D00    1930         CALL 77
C9        1940         RET
```


Ce sous-programme peut être testé au moyen du programme BASIC précédent, en remplaçant la ligne 1010 par :

```
1010 DEF USR = 40338
```

LE SOUS-PROGRAMME DE DÉPLACEMENT DE L'ENVAHISSEUR ("DEPENV")

Ce sous-programme est le plus élaboré de tous. La liste des opérations qu'il réalise successivement est donnée ci-dessous :

1. Si l'indicateur de direction de l'envahisseur est positionné, l'exécution est redirigée vers le sous-programme de déplacement du sprite de la droite vers la gauche (6).
2. L'indicateur de direction étant normalement à zéro, l'envahisseur se déplace de la gauche vers la droite.
3. Si l'envahisseur se trouve à l'extrême droite de l'écran, le programme positionne l'indicateur de direction et retourne à la boucle de jeu.
4. L'envahisseur ne se trouvant pas à l'extrême droite de l'écran, il est déplacé d'une unité vers la droite ; le principe de ce déplacement est tout à fait semblable à celui du fusil : la position horizontale courante est tout d'abord lue et, dans le cas d'un déplacement vers la droite, une unité est ajoutée à cette valeur. La nouvelle valeur est alors chargée à l'emplacement requis, en mémoire VRAM, c'est-à-dire à l'adresse de stockage de la position horizontale de l'envahisseur.
5. Retour à la boucle de jeu.
6. L'indicateur de direction de l'envahisseur est positionné de sorte que celui-ci se déplace de la droite vers la gauche.
7. Si l'envahisseur se trouve à l'extrême gauche de l'écran, le programme remet à zéro l'indicateur de direction et l'exécution est redirigée vers la boucle de jeu.
8. L'envahisseur ne se trouvant pas à l'extrême gauche de l'écran, il est déplacé d'une unité vers la gauche. Pour ce faire, une unité est soustraite de la position horizontale courante de ce sprite.
9. Retour à la boucle de jeu.

Adresse initiale 40243
Adresse finale 40305
Total hexadécimal 5797

Le sous-programme de déplacement de l'envahisseur peut être testé au moyen des quelques lignes suivantes :

```
1000 DEF USR=40363
1005 A=USR (0)
1010 DEF USR=40243
1015 A=USR (0)
1020 GOTO 1015
```

```
3A289E    1350 DEPENV   LD   A,(inddir)
3D        1360          DEC  A
CA569D    1370          JP   Z,ENGAU
21051B    1380          LD   HL,6917
CD4A00    1390          CALL 74
D6F0      1400          SUB  240
C24B9D    1410          JP   NZ,ENDRO1
3E01      1420          LD   A,1
32289E    1430          LD   (inddir),A
C9        1440          RET
21051B    1450 ENDRO1   LD   HL,6917
CD4A00    1460          CALL 74
3C        1470          INC  A
CD4D00    1480          CALL 77
C9        1490          RET
21051B    1500 ENGAU:   LD   HL,6917
CD4A00    1510          CALL 74
D601      1520          SUB  1
C2679D    1530          JP   NZ,ENGAU1
3E00      1540          LD   A,0
32289E    1550          LD   (inddir),A
C9        1560          RET
21051B    1570 ENGAU1   LD   HL,6917
CD4A00    1580          CALL 74
3D        1590          DEC  A
CD4D00    1600          CALL 77
C9        1610          RET
```

LE SOUS-PROGRAMME "FEU"

Ce sous-programme est appelé lorsque l'on appuie sur la barre d'espacement. Sa première fonction est de vérifier qu'il n'y a pas déjà une balle en mouvement sur l'écran. Dans ce cas, l'exécution est immédiatement renvoyée à la boucle de jeu. Ce test est effectué en examinant la valeur de l'indicateur de balle tirée (indbal). S'il est à zéro, c'est qu'une balle est déjà dans le jeu. S'il vaut 1, il n'y a pas de balle à l'écran et le reste du sous-programme est exécuté.

L'étape suivante consiste à positionner horizontalement et verticalement le sprite correspondant au dessin de la balle, de sorte que celle-ci semble sortir du canon du fusil. Le paramètre définissant la position horizontale a la même valeur que celle correspondant à la position du fusil. Le paramètre définissant la position verticale a pour valeur 154–16=138, 154 correspondant à la coordonnée verticale du fusil.

L'indicateur de balle tirée est alors mis à zéro. Cela permet d'une part d'empêcher qu'une autre balle soit tirée et d'autre part que le sous-programme de déplacement de la balle (voir ci-dessous) soit régulièrement appelé lors de l'exécution de la boucle de jeu.

Adresse initiale 40157
Adresse finale 40194
Total hexadécimal 2697

```
3A279E     870 FEU:   LD   A,(indbal)
3C         880        INC  A
3D         890        DEC  A
C8         900        RET  Z
21011B     910        LD   HL,6913
CD4A00     920        CALL 74
21091B     930        LD   HL,6921
CD4D00     940        CALL 77
21001B     950        LD   HL,6912
CD4A00     960        CALL 74
D611       970        SUB  17
21081B     980        LD   HL,6920
CD4D00     990        CALL 77
3E00      1000        LD   A,0
32279E    1010        LD   (indbal),A
C9        1020        RET
```

Les quelques lignes ci-dessous permettent de tester le sous-programme FEU. La balle apparaît à la sortie du fusil.

```
1000 DEF USR=40363
1005 A=USR (0)
1010 DEF USR=40157
1015 A=USR (0)
```

LE SOUS-PROGRAMME "DEPBAL"

Ce sous-programme gère le déplacement à l'écran de la balle qui vient d'être tirée. Ce mouvement est obtenu en diminuant d'une unité, à chaque appel, la valeur de la position verticale courante du sprite. Le sous-programme teste également si la balle a atteint le bord supérieur de l'écran. Dans ce cas, l'indicateur de balle tirée (indbal) est positionné par le programme et le paramètre définissant la position verticale de la balle prend une valeur telle que le sprite disparaisse derrière la bordure de l'écran.

Adresse initiale 40205
Adresse finale 40232
Adresse hexadécimale 2400

```
21081B    1130 DEPBAL   LD    HL,6920
CD4A00    1140          CALL  74
3D        1150          DEC   A
CA1B9D    1160          JP    Z,HAUT
CD4D00    1170          CALL  77
C9        1180          RET
3EC8      1190 HAUT:    LD    A,200
21081B    1200          LD    HL,6920
CD4D00    1210          CALL  77
3E01      1220          LD    A,1
32279E    1230          LD    (indbal),A
C9        1240          RET
```

Les quelques lignes de programme ci-dessous permettent de tester ce sous-programme. La balle se déplace constamment du bas vers le haut de l'écran :

```
1000 DEF USR=40363
1015 A=USR (0)
1020 DEF USR=40157
1025 A=USR (0)
1030 DEF USR=40205
1035 A=USR (0)
1040 GOTO 1035
```

LE SOUS-PROGRAMME "TIRBAL"

L'indicateur de balle tirée (indbal) est testé en permanence lors de l'exécution de la boucle de jeu. S'il est à zéro, le sous-programme TIRBAL est alors appelé. La première fonction de ce sous-programme est d'appeler trois fois de suite le sous-programme de déplacement de la balle (DEPBAL). Celle-ci s'anime alors d'un mouvement très rapide. Le sous-programme TIRBAL teste ensuite si, lors de ce déplacement, la balle a touché l'envahisseur. Pour ce faire, la valeur du registre de collision se trouvant en mémoire VRAM est examinée. Ce registre est positionné lorsque deux sprites se sont superposés, c'est-à-dire, dans ce programme, lorsque la balle a touché l'envahisseur. Au contraire, si ce registre est à zéro, l'indicateur de collision (indcol) n'est pas positionné (il vaut donc zéro). Lorsqu'une collision s'est produite, l'envahisseur a été tué et une explosion doit être simulée par appel du sous-programme EXPLO.

Adresse initiale 40082
Adresse finale 40102
Total hexadécimal 2416


```
CD0D9D    420 TIRBAL    CALL  DEPBAL
CD0D9D    430           CALL  DEPBAL
CD0D9D    440           CALL  DEPBAL
CD3E01    450           CALL  318
CB6F      460           BIT   5,A
C8        470           RET   Z
3E00      480           LD    A,0
32269E    490           LD    (indcol),A
C9        500           RET
```

LE SOUS-PROGRAMME "EXPLO"

La boucle de jeu examine en permanence l'indicateur de collision. Si celui-ci vaut zéro, le sous-programme EXPLO, qui simule visuellement une explosion, est appelé. Cet effet spectaculaire est généré de la manière suivante : l'écran est tout d'abord réinitialisé en mode 3 (mode multicouleur). Les tableaux se trouvant en mémoire VRAM sont alors remplis par une suite de valeurs aléatoires produisant un affichage de carrés de différentes couleurs. Le processus est répété plusieurs fois.

Adresse initiale 40113
Adresse finale 40146
Total hexadécimal 3587

```
3E03      610 EXPLO:    LD    A,3
32AFFC    620           LD    (64687),A
CD5F00    630           CALL  95
3E64      640           LD    A,100
210000    650           LD    HL,0
E5        660 EXPLO1    PUSH  HL
F5        670           PUSH  AF
11000B    680           LD    DE,2048
01E803    690           LD    BC,1000
CD5C00    700           CALL  92
F1        710           POP   AF
E1        720           POP   HL
24        730           INC   H
3D        740           DEC   A
C2BE9C    750           JP    NZ,EXPLO1
C3409C    760           JP    DEBUT
```

Ce sous-programme peut être testé au moyen des lignes suivantes :

```
1000 POKE 40000,201
1005 DEF USR=40113
1010 A=USR (0)
```

LE SOUS-PROGRAMME DE TEMPORISATION ("DELAI")

Ce sous-programme permet de ralentir l'exécution de certaines étapes afin que celles-ci puissent être discernées par le joueur. La boucle de délai est réalisée en chargeant la valeur 255 dans le registre A, puis en décrémentant celui-ci d'une unité jusqu'à atteindre la valeur 0. A ce moment, l'exécution est renvoyée vers la boucle de jeu. Il n'est pas possible de tester ce sous-programme aussi facilement que les précédents. En effet, même avec une valeur maximale de 255, l'exécution est encore trop rapide pour pouvoir être suivie pas à pas. Cependant, à la fin de ce chapitre, quelques indications seront données pour illustrer le fonctionnement de la boucle de délai.

Adresse initiale **40474**
Adresse finale **40480**
Total hexadécimal **959**

```
3EFF    2490 DELAI:  LD   A,255
3D      2500 DEL:    DEC  A
C21C9E  2510         JP   NZ,DEL
C9      2520         RET
```

LA BOUCLE DE JEU

La boucle de jeu réalise les opérations suivantes :

1. Appel du sous-programme de déplacement de l'envahisseur (DEPENV).
2. Examen de l'état de l'indicateur de collision (indcol). Si celui-ci est à zéro, branchement vers le sous-programme EXPLO.
3. Examen de l'état de l'indicateur de balle tirée (indbal). Si celui-ci est à zéro, branchement vers le sous-programme TIRBAL.
4. Si la touche de déplacement du curseur vers la gauche est activée, appel du sous-programme de déplacement du fusil vers la gauche (FUSGAU).
5. Si la touche de déplacement du curseur vers la droite est activée, appel du sous-programme de déplacement du fusil vers la droite (FUSDRO).
6. Si la barre d'espacement est enfoncée, appel du sous-programme FEU.
7. Si la touche X est activée, sortie du programme et retour sous BASIC.

La boucle de jeu appelle également plusieurs fois le sous-programme de temporisation (DELAI). Celui-ci est indispensable pour que le déplacement des sprites puisse être perçu.

La manière dont le programme détecte qu'une touche a été activée mérite quelques commentaires. Le programme ne possède pas d'instruction lui permettant de savoir quelle touche a été activée. Par contre, il peut tester si une touche particulière a été pressée. Un sous-programme du BIOS permet en effet d'accomplir cette fonction. Avant que ce sous-programme puisse être appelé, il est nécessaire de lui fournir certaines informations. Le tableau ci-dessous va permettre d'illustrer la procédure. Ce tableau possède 9 lignes numérotées de 0 à 8, chacune de ces lignes contenant 8 touches :

	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	;	]	[	\	=	-	9	8
2	B	A	£	/	.	,	'	`
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	CODE	CAP	GRAPH	CTRL	SHIFT
7	RETOUR CHARIOT	SEL	ESPACE ARRIÈRE	STOP	TAB	ESC	F5	F4
8	CURSEUR DROITE	CURSEUR BAS	CURSEUR-HAUT	CURSEUR GAUCHE	DEL	INS	HOME	ESPACE-MENT

Supposons que l'on veuille savoir si la touche correspondant au déplacement du curseur vers la gauche a été pressée. La marche à suivre est la suivante :

- **Charger la valeur 8 dans le registre A** (8 correspond, sur le schéma précédent, au numéro de la ligne sur laquelle se trouve la touche à tester).
- **Appeler le sous-programme en 321d (CALL 321)** (il s'agit de l'adresse du sous-programme lisant le numéro de la ligne qui a été spécifiée).

En retour, le sous-programme charge dans le registre A une valeur correspondant au numéro de la touche, sur la ligne spécifiée, qui a été pressée. Le schéma précédent montre en effet que les colonnes sont numérotées de 0 à 7. La touche de déplacement du curseur vers la gauche correspond à la colonne 4. Pour tester si cette touche a été pressée, il suffit d'utiliser l'instruction :

BIT 4,A

l'instruction suivante devant être :

CALL Z, sous-programme de déplacement du fusil vers la gauche

Si le résultat de l'instruction BIT 4,A est zéro (c'est-à-dire si la touche en question a été pressée), le sous-programme en question est appelé. Le listing de la boucle de jeu est donné ci-dessous.

Adresse initiale **40000**
Adresse finale **40071**
Total hexadécimal **8681**

```
CDAB9D    20 DEBUT:  CALL  INIT
CD339D    30 BOUCLE  CALL  DEPENV
CD1A9E    40         CALL  DELAI
3A269E    50         LD    A,(indcol)
3C        60         INC   A
3D        70         DEC   A
CAB19C    80         JP    Z,EXPLO
3A279E    90         LD    A,(indbal)
3C       100         INC   A
3D       110         DEC   A
CC929C   120         CALL  Z,TIREBAL
3E08     130         LD    A,8
CD4101   140         CALL  321
CB67     150         BIT   4,A
CC7C9D   160         CALL  Z,FUSGAU
CD1A9E   170         CALL  DELAI
3E08     180         LD    A,8
CD4101   190         CALL  321
CB7F     200         BIT   7,A
CC929D   210         CALL  Z,FUSDRO
3E08     220         LD    A,8
CD4101   230         CALL  321
CB47     240         BIT   0,A
CCDD9C   250         CALL  Z,FEU
CD1A9E   260         CALL  DELAI
3E05     270         LD    A,5
CD4101   280         CALL  321
CB6F     290         BIT   5,A
C8       300         RET   Z
C3439C   310         JP    BOUCLE
```

Le jeu peut maintenant être testé dans sa totalité en tapant :

```
1000 DEF USR=40000
1005 A=USR (0)
```


Si l'on désire se rendre compte de l'influence de la boucle de temporisation, il suffit, pour la supprimer, d'entrer l'instruction suivante :

```
POKE 40475,1
```

et de relancer le programme. L'adresse 40475 contient normalement la valeur 255, c'est-à-dire la valeur correspondant au délai de temporisation.

7

CARACTÉRISTIQUES DES ORDINATEURS MSX

Ce chapitre traite de quelques sujets importants ne concernant pas directement la programmation en langage machine. Un livre d'initiation tel que celui-ci ne peut prétendre aborder chaque sujet en détail. Des ouvrages spécialisés doivent être consultés pour en savoir davantage sur certains points particuliers. Cependant, pas plus qu'il n'est indispensable de connaître le fonctionnement du moteur à piston pour pouvoir conduire une voiture, il n'est nécessaire de connaître les moindres détails d'une caractéristique particulière d'un ordinateur pour pouvoir commencer à l'utiliser avec profit. Ce chapitre sera donc plus centré sur les possibilités d'utilisation que sur les structures ou les principes de fonctionnement.

LES QUATRE MODES ÉCRAN

Le seul mode écran qui ait été décrit dans les programmes présentés jusqu'à présent est le mode écran 1. Il s'agit probablement du mode le mieux adapté à la réalisation de jeux d'arcade.

Le mode écran 0 dispose de 24 lignes de 40 caractères. Il est initialisé au moyen de l'instruction SCREEN 0. Seules deux couleurs sont utilisables simultanément dans ce mode, une pour le fond et une pour l'avant-plan. La bordure s'affiche nécessairement dans la couleur du fond. Ce mode est essentiellement utilisé pour des programmes ne faisant pas appel au graphisme et pour des applications telles que le traitement de texte. Les sprites ne peuvent pas être utilisés en mode écran 0.

Le mode 2 ressemble beaucoup au mode 1. Un maximum de 768 caractères différents peuvent être affichés simultanément à l'écran. De plus, chacun des huit octets codant pour la forme d'un caractère peut correspondre à une combinaison de deux couleurs quelconques.

Le mode 3, ou mode multicouleur, est d'un intérêt moindre et d'une utilisation plus délicate. Comme en mode 1, l'écran est divisé en 24 lignes de 32 caractères, soit au total 768 positions de caractères. Cependant, au lieu d'afficher des caractères, ce mode affiche des blocs de couleur. Chaque position de caractère est divisée en quatre carrés accolés, comme le montre le schéma ci-dessous :

Chaque carré peut être affiché individuellement dans l'une des seize couleurs disponibles.

LES SPRITES

L'un des modes possibles pour l'affichage des sprites a été décrit dans un chapitre précédent. Les trois autres modes disponibles vont être brièvement évoqués ici. Les quelques lignes suivantes permettent de faire apparaître à l'écran un sprite défini sur 8 octets et s'affichant dans une taille normale (ATSPRITE représente l'adresse de la base du tableau des attributs de sprites et FORMSPRITE celle de la base du tableau de formes des sprites) :

```
SCREEN 1
VDP(1)=224
VPOKE(ATSPRITE+0),100
VPOKE(ATSPRITE+1),50
VPOKE(ATSPRITE+2),0
VPOKE(ATSPRITE+3),10
VPOKE(FORMSPRITE+0),255
VPOKE(FORMSPRITE+1),129
VPOKE(FORMSPRITE+2),129
VPOKE(FORMSPRITE+3),129
VPOKE(FORMSPRITE+4),129
VPOKE(FORMSPRITE+5),129
VPOKE(FORMSPRITE+6),129
VPOKE(FORMSPRITE+7),255
```

Un sprite ayant la forme d'une boite est ainsi affiché au milieu de l'écran. Tapons maintenant :

```
VDP(1)=255
```

Le sprite est alors affiché dans une taille double. Le sprite affiché dans ce deuxième mode peut être déplacé à l'écran en modifiant ses coordonnées horizontale et verticale. Retapons l'instruction VDP(1)=224 pour redonner au sprite sa taille originale. Le troisième mode d'affichage correspond à un sprite géant (défini sur 32 octets) correspondant à l'association de quatre sprites normaux (définis sur 8 octets), en accord avec le schéma suivant :

Lorsque les attributs du premier sprite sont modifiés au moyen de l'instruction VPOKE, ceux des trois autres sprites le sont aussi. Le programme BASIC donné ci-dessous permet de charger en mémoire VRAM les données de formes correspondant à trois sprites supplémentaires :

```
10 FOR A=8 TO 31
20 READ B
30 VPOKE(FORMSPRITE+A),B
40 NEXT A
50 STOP
60 DATA 255, 195, 165, 153, 153, 165, 195, 255
70 DATA 255, 153, 153, 255, 255, 153, 153, 255
80 DATA 231, 165, 255, 36, 36, 255, 165, 231
```

En tapant ensuite VDP(1)=226, on voit s'afficher les quatre sprites individuels sous la forme d'un sprite géant (défini sur 32 octets). Si l'on modifie la position horizontale ou verticale de ce sprite, le motif graphique se déplace dans sa totalité. Il est possible de définir ainsi les formes de 64 sprites géants (chacun d'entre eux occupant 32 octets dans le tableau de formes). L'adresse du premier octet codant pour les attributs d'un sprite défini sur 32 octets peut être connue au moyen de la formule suivante :

(NUMÉRO DU SPRITE × 16) + ATSPRITE

Les sprites affichés sont également numérotés de 0 à 31 dans ce mode.

Le dernier mode d'affichage des sprites permet d'afficher des sprites définis sur 32 octets mais dont l'image est agrandie à l'écran d'un facteur 2. Ce mode peut être obtenu en tapant :

VDP(1)=227

LA MÉMOIRE VIDÉO RAM (OU VRAM)

La mémoire VRAM dispose de 16 384 octets qu'il est possible de configurer de différentes manières. Cette mémoire est contrôlée par le processeur d'affichage vidéo (VDP) qui possède huit registres accessibles en mode écriture, numérotés de 0 à 7. La valeur affectée à certains de ces registres détermine la configuration de la mémoire VRAM. On ne doit pas choisir ces valeurs au hasard sous peine de bloquer le fonctionnement normal de l'ordinateur et de n'avoir pour seule solution que de couper l'alimentation puis de la rallumer (en ayant perdu dans l'opération toutes les informations stockées en mémoire RAM ou VRAM).

Le registre 0 du VDP sert à positionner des fonctions qui ne seront pas décrites ici. Sa valeur d'initialisation ne doit pas être modifiée sous peine de problèmes majeurs.

La fonction principale du registre 1 est de contrôler le type et la taille des sprites. Lorsque ce registre a pour valeur 224, les formes de sprites sont définies sur 8 octets et ne sont pas agrandies à l'écran. La valeur 225 correspond également à des sprites définis sur 8 octets, mais qui sont agrandis d'un facteur 2 lorsqu'ils sont affichés à l'écran. Lorsque la valeur 226 est chargée dans le registre 1 du VDP, les sprites

sont définis sur 32 octets et ne sont pas agrandis à l'écran. Dans ces conditions, ils occupent à l'écran un emplacement correspondant à quatre sprites normaux non agrandis. La valeur 227 correspond également à des sprites définis sur 32 octets, mais cette fois agrandis d'un facteur 2 (dans les deux directions) à l'écran.

La valeur contenue dans le registre 2 du VDP initialise l'adresse de la base du tableau de noms, en mémoire VRAM. Cette adresse a été désignée par INITECRAN dans les chapitres précédents. Il existe seize possibilités différentes, selon la valeur du registre 2, pour initialiser l'adresse de la base de ce tableau :

VDP 2	Adresse de la base du tableau de noms
0	0
1	1024
2	2048
3	3072
4	4096
5	5120
6	6144
7	7168
8	8192
9	9216
10	10240
11	11264
12	12288
13	13312
14	14336
15	15360

La valeur contenue dans le registre 3 du VDP détermine l'adresse de la base du tableau des couleurs. Ce registre peut prendre n'importe quelle valeur décimale comprise entre 0 et 255. L'adresse de la base du tableau des couleurs est obtenue en multipliant cette valeur décimale par 64. Par exemple, si le registre 3 du VDP a pour valeur 21, le premier octet du tableau des couleurs se trouvera à l'adresse 21×64=1344.

La valeur du registre 4 du VDP détermine l'adresse de la base du tableau des formes de caractères. Il n'existe que huit possibilités :

VDP 4	Adresse de la base du tableau des formes de caractères
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

La valeur du registre 5 du VDP détermine l'adresse de la base du tableau des attributs de sprites. Cette valeur peut être comprise entre 0 et 127 inclus. L'adresse correspondante est obtenue en multipliant la valeur contenue dans le registre 5 par 128. Si cette valeur est de 45 par exemple, le premier octet du tableau des attributs de sprites se trouvera à l'adresse 45×128=5760, en mémoire VRAM.

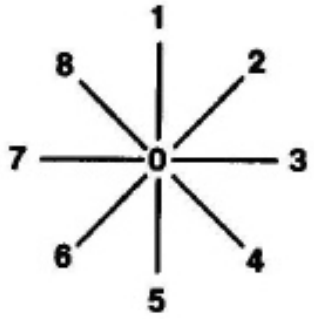
Le registre 5 code pour l'adresse de la base du tableau des formes de sprites. Ce registre peut prendre les valeurs suivantes :

VDP 6	Adresse de la base du tableau des formes de sprites
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

Le registre 7 du VDP présente peu d'intérêt pour le programmeur en langage machine. Sa valeur permet en particulier de définir quelles seront les couleurs (fond et avant-plan) affichées en mode écran 0 une fois qu'une instruction BASIC CLS aura été exécutée.

LES MANETTES DE JEU *(JOYSTICKS)*

Tous les ordinateurs MSX disposent d'une sortie *joystick* à un ou deux connecteurs. L'état du port joystick peut être testé au moyen de deux sous-programmes se trouvant en mémoire ROM. Le premier sous-programme retourne la position du manche à balai sur la manette de jeu spécifiée. Le numéro (1 ou 2) de la manette de jeu doit être chargé dans le registre A et le sous-programme appelé à l'adresse 213. En retour, la valeur correspondant à la position lue se trouve dans le registre A. Le schéma ci-dessous indique les correspondances entre les valeurs retournées et la position du manche à balai. La valeur 0 correspond à la position centrale.



Le sous-programme en langage machine écrit ci-dessous peut être utilisé pour lire la position du manche à balai. La valeur 1 (correspondant au port joystick n° 1) est tout d'abord chargée dans le registre A. Le sous-programme est ensuite appelé à l'adresse 213. La valeur contenue en retour dans le registre A est finalement stockée à l'adresse 50000 :

Adresse initiale **40000**
Adresse finale **40008**
Total hexadécimal **1007**

```
3E01
CDD500
3250C3
C9
```

Les quelques lignes BASIC ci-dessous permettent de tester ce sous-programme :

```
1000 SCREEN 1
1005 DEF USR=40000
1010 A=USR(0)
1015 B=PEEK(50000)
1020 PRINT B
1025 GOTO 1010
```

Lorsque ce programme est exécuté, la valeur correspondant à la position du manche à balai du joystick n° 1 est affichée à l'écran.

Le second sous-programme se trouvant en mémoire ROM sert à tester si le bouton-poussoir de l'une des deux manettes de jeu est enfoncé. Ce sous-programme se trouve à l'adresse 216. Comme dans le cas précédent, la valeur 1 ou 2 doit tout d'abord être chargée dans le registre A, selon le numéro du joystick qui doit être testé. En retour, la valeur 255 est chargée dans le registre A si le bouton-poussoir correspondant était enfoncé. Dans le cas contraire, ce registre contient la valeur 0. Le programme en langage machine donné ci-dessous illustre l'utilisation de ce sous-programme en mémoire ROM. (Ce programme ne doit pas être exécuté si l'on ne dispose pas de manettes de jeu. Le seul moyen de retourner au BASIC consiste en effet à appuyer sur le bouton-poussoir de la manette.)

Adresse initiale **40000**
Adresse finale **40008**
Total hexadécimal **1025**

```
3E01
CDD800
3C
20F8
C9
```

Ce programme peut être testé au moyen des quelques lignes BASIC suivantes :

```
1000 DEF USR=40000
1005 A=USR(0)
1010 CLS
1015 PRINT "Le bouton-poussoir a été enfoncé"
```

LE BIOS (Système d'exploitation des entrées/sorties BASIC)

Le BIOS contient un grand nombre de sous-programmes en langage machine facilitant la réalisation de programmes machine écrits par l'utilisateur. Une liste de ces sous-programmes les plus importants est donnée ci-dessous :

Objet Remplissage de la mémoire VRAM.

Commentaire Lorsqu'il est appelé, ce sous-programme charge des données dans une zone spécifiée de la mémoire VRAM. Par exemple, le caractère n° 32 peut être chargé dans les 768 octets du tableau de noms, en mode écran 1. Cela aura pour effet d'effacer l'écran.

Procédure L'adresse du premier octet du bloc à charger doit être passée au registre HL. La longueur du bloc est chargée dans BC et les données (ici la valeur 32) dans le registre A.

Code d'appel CD 56 00

Objet Déplacement d'un bloc de données de la mémoire VRAM en mémoire RAM.

Commentaire Ce sous-programme peut par exemple être utilisé pour copier en mémoire RAM le contenu de la mémoire écran.

Procédure L'adresse VRAM du premier octet du bloc à déplacer doit être chargée dans le registre HL; celle du premier octet à utiliser en mémoire RAM doit être chargée dans le registre DE et la longueur du bloc est passée comme valeur au registre BC.

Code d'appel CD 59 00

Objet Déplacement d'un bloc de données de la mémoire RAM en mémoire VRAM.

Commentaire Ce sous-programme peut par exemple être utilisé pour recopier, en mémoire VRAM, un nouveau jeu de caractères défini par l'utilisateur et stocké en mémoire RAM.

Procédure L'adresse en mémoire RAM du premier octet du bloc à déplacer doit être chargée dans le registre HL; celle du premier octet à utiliser en mémoire VRAM doit être chargée dans le registre DE et la longueur du bloc à déplacer est passée comme valeur au registre BC.

Code d'appel CD 5C 00

Objet Lecture d'un caractère entré au clavier.

Commentaire Lorsque ce sous-programme est appelé, il attend qu'un caractère soit entré au clavier, puis retourne le numéro de code de ce caractère dans le registre A. Ce sous-programme est très utilisé en mode conversationnel.

Code d'appel CD 9F 00

Objet Positionnement du curseur.

Commentaire Ce sous-programme positionne le curseur à l'endroit spécifié de l'écran. Sa fonction est donc semblable à celle que remplit l'instruction BASIC "LOCATE".

Procédure Le numéro de colonne correspondant à la position définie pour le curseur doit être chargé dans le registre H. En mode écran 1, cette valeur est comprise entre 0 et 31. Le numéro de ligne définissant la position verticale du curseur doit être chargé dans le registre L.

Code d'appel CD C6 00

Objet Suppression de l'affichage à l'écran de la signification des touches de fonction.

Commentaire Lorsque ce sous-programme est exécuté, la ligne consacrée, au bas de l'écran, à l'affichage de la signification des touches de fonction programmables est effacée. Pour autant, ces touches de fonction restent utilisables.

Code d'appel CD CC 00

Objet Rétablissement de l'affichage de la signification des touches de fonction programmables.

Commentaire Lorsque ce sous-programme est exécuté, les premières lettres correspondant aux valeurs affectées à ces touches sont affichées sur la dernière ligne de l'écran.

Code d'appel CD CF 00

Objet Écriture en mémoire VRAM.

Commentaire Ce sous-programme est équivalent de l'instruction BASIC "VPOKE(adresse VRAM),n", où l'adresse VRAM est définie par un nombre compris entre 0 et

16 383, et n (compris entre 0 et 255) est la valeur à charger à cette adresse.

Procédure L'adresse VRAM doit être chargée dans le registre HL et la valeur (comprise entre 0 et 255) à écrire à cette adresse doit être chargée dans le registre A.

Code d'appel CD 4D 00

Objet Lecture du contenu d'un octet en mémoire VRAM.

Commentaire Ce sous-programme est équivalent de l'instruction BASIC "VPEEK(adresse VRAM)". En retour de ce sous-programme, le registre A contient la valeur qui a été lue à l'adresse spécifiée.

Procédure L'adresse VRAM, dont le contenu doit être lu, doit être chargée dans le registre HL.

Code d'appel CD 4A 00

8

SOUS-PROGRAMMES UTILITAIRES

Ce chapitre présente quelques sous-programmes utilitaires écrits en langage machine. Ils démontrent en particulier la puissance de ce langage. Ces sous-programmes s'implantent en différents endroits de la mémoire RAM, de sorte qu'il est possible d'en charger plusieurs simultanément.

Objet Analyse de gauche à droite des caractères contenus dans une ligne de texte.

Commentaire Ce sous-programme permet d'analyser, en partant de la gauche, les caractères se trouvant sur l'une quelconque des 24 lignes disponibles en mode écran 1.

Procédure L'adresse mémoire 50000 doit contenir une valeur comprise entre 0 et 23 correspondant au numéro de la ligne qui doit être analysée.

Adresse initiale **40000**
Adresse finale **40044**
Total hexadécimal **4158**

```
3A50C3   20          LD    A,(50000)
012000   30          LD    BC,32
211F18   40          LD    HL,6175
3C       50          INC   A
3D       60 BOUCLE   DEC   A
CA529C   70          JP    Z,BCLE1
09       80          ADD   HL,BC
C34A9C   90          JP    BOUCLE
CD4A00   100 BCLE1:  CALL  74
F5       110         PUSH  AF
011F00   120         LD    BC,31
2B       130 BCLE2:  DEC   HL
CD4A00   140         CALL  74
23       150         INC   HL
CD4D00   160         CALL  77
2B       170         DEC   HL
0B       180         DEC   BC
79       190         LD    A,C
B0       200         OR    B
C2599C   210         JP    NZ,BCLE2
F1       220         POP   AF
CD4D00   230         CALL  77
C9       240         RET
```


Objet Analyse, de droite à gauche, des caractères contenus dans une ligne de texte.

Commentaire Ce sous-programme permet d'analyser, en partant de la droite, les caractères se trouvant sur l'une quelconque des 24 lignes disponibles en mode écran 1.

Procédure L'adresse mémoire 50001 doit contenir une valeur comprise entre 0 et 23 correspondant au numéro de la ligne qui doit être analysée.

Adresse initiale **40100**
Adresse finale **40144**
Total hexadécimal **4420**

```
3A51C3   20          LD    A,(50001)
012000   30          LD    BC,32
210018   40          LD    HL,6144
3C       50          INC   A
3D       60 BOUCLE   DEC   A
CAB69C   70          JP    Z,BCLE1
09       80          ADD   HL,BC
C3AE9C   90          JP    BOUCLE
CD4A00   100 BCLE1:  CALL  74
F5       110         PUSH  AF
011F00   120         LD    BC,31
23       130 BCLE2:  INC   HL
CD4A00   140         CALL  74
2B       150         DEC   HL
CD4D00   160         CALL  77
23       170         INC   HL
0B       180         DEC   BC
79       190         LD    A,C
B0       200         OR    B
C2BD9C   210         JP    NZ,BCLE2
F1       220         POP   AF
CD4D00   230         CALL  77
C9       240         RET
```

Objet Analyse de bas en haut des caractères contenus dans une colonne de texte.

Commentaire Ce sous-programme permet d'analyser, en partant du bas, les caractères se trouvant sur l'une quelconque des 32 colonnes de texte disponibles en mode écran 1.

Procédure L'adresse mémoire 50002 doit contenir une valeur comprise entre 0 et 31 correspondant au numéro de la colonne qui doit être analysée.

Adresse initiale 40300
Adresse finale 40345
Total hexadécimal 4540

```
3A52C3    20          LD    A,(50002)
3C        30          INC   A
21001B    40          LD    HL,6144
3D        50 BOUCLE   DEC   A
CA7B9D    60          JP    Z,BCLE1
23        70          INC   HL
C3739D    80          JP    BOUCLE
CD4A00    90 BCLE1:   CALL  74
F5       100          PUSH  AF
011700   110          LD    BC,23
112000   120          LD    DE,32
19       130 BCLE2:   ADD   HL,DE
CD4A00   140          CALL  74
ED52     150          SBC   HL,DE
CD4D00   160          CALL  77
19       170          ADD   HL,DE
0B       180          DEC   BC
79       190          LD    A,C
B0       200          OR    B
C2859D   210          JP    NZ,BCLE2
F1       220          POP   AF
CD4D00   225          CALL  77
C9       230          RET
```

Objet Analyse de haut en bas des caractères contenus dans une colonne de texte.

Commentaire Ce sous-programme permet d'analyser, en partant du haut, les caractères se trouvant dans l'une quelconque des 32 colonnes de texte disponibles en mode écran 1.

Procédure L'adresse mémoire 50003 doit contenir une valeur comprise entre 0 et 31 correspondant au numéro de la colonne qui doit être analysée.

Adresse initiale 40200
Adresse finale 40246
Total hexadécimal 4761


```
3A53C3    20          LD    A,(50003)
3C        30          INC   A
21E01A    40          LD    HL,6880
3D        50 BOUCLE   DEC   A
CA179D    60          JP    Z,BCLE1
23        70          INC   HL
C30F9D    80          JP    BOUCLE
CD4A00    90 BCLE1:   CALL  74
F5       100          PUSH  AF
011700   110          LD    BC,23
112000   120          LD    DE,32
ED52     130 BCLE2:   SBC   HL,DE
CD4A00   140          CALL  74
19       150          ADD   HL,DE
CD4D00   160          CALL  77
ED52     170          SBC   HL,DE
0B       180          DEC   BC
79       190          LD    A,C
B0       200          OR    B
C2219D   210          JP    NZ,BCLE2
F1       220          POP   AF
CD4D00   230          CALL  77
C9       240          RET
```

```
1000 SCREEN 1
1005 VDP(2)=6
1010 X=65
1015 FOR A=6144 TO 6880 STEP 32
1020 Y=X
1025 FOR B=0 TO 31
1030 VPOKE(A+B),Y
1035 Y=Y+1
1040 NEXT B
1045 X=X+1
1050 NEXT A
1055 POKE (50000),10
1060 DEF USR=40000
1065 A=USR(1):GOTO 1065
```

Le programme BASIC ci-dessus peut être utilisé pour tester l'un des quatre sous-programmes en langage machine donnés précédemment. Seules les lignes 1055 et 1060 ont à être modifiées en fonction du programme qui doit être testé, et du numéro de la ligne ou de la colonne dont le contenu doit être analysé.

Objet Génération d'un bruit de fusil laser.

Commentaire Le bruit généré par ce programme est typique de ceux produits dans les jeux d'arcade tels que "Les envahisseurs de l'espace".

Adresse initiale 40600
Adresse finale 40667
Total hexadécimal 7249

```
3E08    20          LD   A,8
1E0F    30          LD   E,15
CD9300  40          CALL 147
3E07    50          LD   A,7
1EFE    60          LD   E,254
CD9300  70          CALL 147
3E00    80          LD   A,0
1E6E    90          LD   E,110
CD9300  100         CALL 147
3E01    110         LD   A,1
1E00    120         LD   E,0
CD9300  130         CALL 147
1E6E    140         LD   E,110
3E00    150 BOUCLE  LD   A,0
CD9300  160         CALL 147
3EC8    170         LD   A,200
F5      180 DELAI:  PUSH A,F
3E0A    190         LD   A,10
3D      200 DEL:    DEC  A
C2C09E  210         JP   NZ,DEL
F1      220         POP  AF
3D      230         DEC  A
C2BD9E  240         JP   NZ,DELAI
7B      250         LD   A,E
C606    260         ADD  A,6
CAD49E  270         JP   Z,FIN
3D      280         DEC  A
5F      290         LD   E,A
C3B69E  300         JP   BOUCLE
3E07    310 FIN:    LD   A,7
1EFF    320         LD   E,255
CD9300  330         CALL 147
C9      340         RET
```


Objet Émission d'un bruit simulant l'explosion d'une bombe.

Commentaire Le programme commence par générer le sifflement de la bombe puis le bruit de son explosion.

Adresse initiale 40400
Adresse finale 40514
Total hexadécimal 13909

```
3E07    20          LD   A,7
1EFE    30          LD   E,254
CD9300  40          CALL 147
3E0A    50          LD   A,8
1E0F    60          LD   E,15
CD9300  70          CALL 147
1E28    80          LD   E,40
3E00    90 BOUCLE   LD   A,0
CD9300  100         CALL 147
3E0A    110         LD   A,10
F5      120 DELAI:  PUSH AF
3EFF    130         LD   A,255
3D      140 DEL:    DEC  A
C2EA9D  150         JP   NZ,DEL
F1      160         POP  AF
3D      170         DEC  A
C2E79D  180         JP   NZ,DELAI
7B      190         LD   A,E
D696    200         SUB  150
CAFF9D  210         JP   Z,SUIV
C697    220         ADD  A,151
5F      230         LD   E,A
C3E09D  240         JP   BOUCLE
3E00    250 SUIV:   LD   A,0
1E00    260         LD   E,0
CD9300  270         CALL 147
3E07    280         LD   A,7
1EF7    290         LD   E,247
CD9300  300         CALL 147
3E00    310         LD   A,0
F5      320 SUIV1:  PUSH AF
5F      330         LD   E,A
CD9300  340         CALL 147
3E32    350         LD   A,50
```

```
F5        360 LUCIEN PUSH AF
3EFF      370        LD   A,255
3D        380 PAULO: DEC  A
C2199E    390        JP   NZ,PAULO
F1        400        POP  AF
3D        410        DEC  A
C2169E    420        JP   NZ,LUCIEN
F1        430        POP  AF
D61F      440        SUB  31
CA2D9E    450        JP   Z,LDEL
C620      460        ADD  A,32
C30F9E    470        JP   SUIV1
3E64      480 LDEL:  LD   A,100
F5        490 LDEL1: PUSH AF
3EFF      500        LD   A,255
3D        510 LDEL2: DEC  A
C2329E    520        JP   NZ,LDEL2
F1        530        POP  AF
3D        540        DEC  A
C22F9E    550        JP   NZ,LDEL1
3E07      560        LD   A,7
1EFF      570        LD   E,255
CD9300    580        CALL 147
C9        590        RET
```


ANNEXE 1

CODES MACHINE DU MICROPROCESSEUR Z80

ADC A, (HL)	8E
ADC A, (IX + d)	DD8Ed
ADC A, (IY + d)	FD8Ed
ADC A, A	8F
ADC A, B	88
ADC A, C	89
ADC A, D	8A
ADC A, E	8B
ADC A, H	8C
ADC A, L	8D
ADC A, n	CEn
ADC HL, BC	ED4A
ADC HL, DE	ED5A
ADC HL, HL	ED6A
ADC HL, SP	ED7A
ADD A, (HL)	86
ADD A, (IX + d)	DD86d
ADD A, (IY + d)	FD86d
ADD A, A	87
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, H	84
ADD A, L	85
ADD A, n	C6n
ADD HL, BC	09
ADD HL, DE	19
ADD HL, HL	29
ADD HL, SP	39
ADD IX, BC	DD09
ADD IX, DE	DD19
ADD IX, IX	DD29
ADD IX, SP	DD39
ADD IY, BC	FD09
ADD IY, DE	FD19
ADD IY, IY	FD29
ADD IY, SP	FD39
AND(HL)	A6
AND (IX + d)	DDA6d
AND (IY + d)	FDA6d
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND n	E6n
BIT 0, (HL)	CB46
BIT 0, (IX + D)	DDCBd46
BIT 0, (IY + d)	FDCBd46
BIT 0, A	CB47
BIT 0, B	CB40
BIT 0, C	CB41
BIT 0, D	CB42
BIT 0, E	CB43
BIT 0, H	CB44
BIT 0, L	CB45
BIT 1, (HL)	CB4E
BIT 1, (IX + d)	DDCBd4E
BIT 1, (IY + d)	FDCBd4E
BIT 1, A	CB4F
BIT 1, B	CB48
BIT 1, C	CB49
BIT 1, D	CB4A
BIT 1, E	CB4B
BIT 1, H	CB4C
BIT 1, L	CB4D
BIT 2, (HL)	CB56
BIT 2, (IX + d)	DDCBd56
BIT 2, (IY + d)	FDCBd56
BIT 2, A	CB57
BIT 2, B	CB50
BIT 2, C	CB51
BIT 2, D	CB52
BIT 2, E	CB53
BIT 2, H	CB54
BIT 2, L	CB55
BIT 3, (HL)	CB5E
BIT 3, (IX + d)	DDCBd5E
BIT 3, (IY + d)	FDCBd5E

BIT 3, A	CB5F
BIT 3, B	CB58
BIT 3, C	CB59
BIT 3, D	CB5A
BIT 3, E	CB5B
BIT 3, H	CB5C
BIT 3, L	CB5D
BIT 4, (HL)	CB66
BIT 4, (IX + d)	DDCBd66
BIT 4, (IY + d)	FDCBd66
BIT 4, A	CB67
BIT 4, B	CB60
BIT 4, C	CB61
BIT 4, D	CB62
BIT 4, E	CB63
BIT 4, H	CB64
BIT 4, L	CB65
BIT 5, (HL)	CB6E
BIT 5, (IX + d)	DDCBd6E
BIT 5, (IY + D)	FDCBd6E
BIT 5, A	CB6F
BIT 5, B	CB68
BIT 5, C	CB69
BIT 5, D	CB6A
BIT 5, E	CB6B
BIT 5, H	CB6C
BIT 5, L	CB6D
BIT 6, (HL)	CB76
BIT 6, (IX + d)	DDCBd76
BIT 6, (IY + d)	FDCBd76
BIT 6, A	C877
BIT 6, B	CB70
BIT 6, C	CB71
BIT 6, D	CB72
BIT 6, E	CB73
BIT 6, H	CB74
BIT 6, L	CB75
BIT 7, (HL)	CB7E
BIT 7, (IX + d)	DDCBd7E
BIT 7, (IY + d)	FDCBd7E
BIT 7, A	CB7F
BIT 7, B	CB78
BIT 7, C	CB79
BIT 7, D	CB7A
BIT 7, E	CB7B
BIT 7, H	CB7C
BIT 7, L	CB7D
CALL C, nn	DCnn
CALL M, nn	FCnn
CALL NC, nn	D4nn
CALL nn	CDnn
CALL NZ, nn	C4nn
CALL P, nn	F4nn
CALL PE, nn	ECnn
CALL PO, nn	E4nn
CALL Z, nn	CCnn
CCF	3F
CP (HL)	BE
CP (IX + d)	DDBEd
CP (IY + d)	FDBEd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FEn
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX + d)	DD35d
DEC (IY + d)	FD35d
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ, d	10d
E1	FB
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3
EX AF, AF	08
EX DE, HL	EB
EXX	D9
HALT	76

IM 0	ED46
IM 1	ED56
IM 2	ED5E
IN A, (C)	ED78
IN A, (n)	DBn
IN B, (C)	ED40
IN C, (C)	ED48
IN D, (C)	ED50
IN E, (C)	ED58
IN H, (C)	ED60
IN L, (C)	ED68
INC (HL)	34
INC (IX + d)	DD34d
INC (IY + d)	FD34d
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD23
INC IY	FD23
INC L	2C
INC SP	33
IND	EDAA
INDR	EDBA
INI	EDA2
INIR	EDB2
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9
JP C, nn	DAnn
JP M, nn	FAnn
JP NC, nn	D2nn
JP nn	C3nn
JP NZ, nn	C2nn
JP P, nn	F2nn
JP PE, nn	EAnn
JP PO, nn	E2nn
JP Z, nn	CAnn
JR C, d	38d
JR, d	18d
JR NC, d	30d
JR NZ, d	20d
JR Z, d	28d
LD (BC), A	02
LD (DE), A	12
LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75
LD (HL), n	36n
LD (IX + d), A	DD77d
LD (IX + d), B	DD70d
LD (IX + d), C	DD71d
LD (IX + d), D	DD72d
LD (IX + d), E	DD73d
LD (IX + d), H	DD74d
LD (IX + d), L	DD75d
LD (IX + d), n	DD36dn
LD (IY + d), A	FD77d
LD (IY + d), B	FD70d
LD (IY + d), C	FD71d
LD (IY + d), D	FD72d
LD (IY + d), E	FD73d
LD (IY + d), H	FD74d
LD (IY + d), L	FD75d
LD (IY + d), n	FD36dn
LD (nn), A	32nn
LD (nn), BC	ED43nn
LD (nn), DE	ED53nn
LD (nn), HL	22nn
LD (nn), IX	DD22nn
LD (nn), IY	FD22nn
LD (nn), SP	ED73nn
LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	7E
LD A, (IX + d)	DD7Ed
LD A, (IY + d)	FD7Ed
LD A, (nn)	3Ann
LD A, A	7F
LD A, B	78
LD A, C	79
LD A, D	7A
LD A, E	7B
LD A, H	7C
LD A, I	ED57
LD A, L	7D
LD A, n	3En
LD B, (HL)	46
LD B, (IX + d)	DD46d
LD B, (IY + d)	FD46d
LD B, A	47

LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45
LD B, n	06n
LD B, C (nn)	ED4Bnn
LD BC (nn)	01nn
LD C, (HL)	4E
LD C, (IX + d)	DD4Ed
LD C, (IY + d)	FD4Ed
LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D
LD C, n	0En
LD D, (HL)	56
LD D, (IX + d)	DD56d
LD D, (IY + d)	FD56d
LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55
LD D, n	16n
LD DE, (nn)	ED58nn
LD DE, nn	11nn
LD E, (HL)	5E
LD E, (IX + d)	DD5Ed
LD E, (IY + d)	FD5Ed
LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D
LD E, n	1En
LD H, (HL)	66
LD H, (IX + d)	DD66d
LD H, (IY + d)	FF66d
LD H, A	67
LD H, B	60
LD H, C	61
LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65
LD H, n	26n
LD HL, (nn)	2Ann
LD HL, nn	21nn
LD I, A	ED47
LD IX, (nn)	DD2Ann
LD IX, nn	DD21nn
LD IY, (nn)	FD2Ann
LD IY, nn	FD21nn
LD L, (HL)	6E
LD L, (IX + d)	DD6Ed
LD L, (IY + d)	FD6Ed
LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D
LD L, n	2En
LD SP, (nn)	ED7Bnn
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9
LD SP, nn	31nn
LDD	EDA8
LDDR	EDB8
LDI	EDAO
LDIR	EDB0
NEG	ED44
NOP	00
OR (HL)	B6
OR (IX + d)	DDB6d
OR (IY + d)	FDB6d
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR n	F6n
OTDR	EDBB
OTIR	EDB3
OUT (C), A	ED79
OUT (C), B	ED41
OUT (C), C	ED49

OUT (C), D	ED51
OUT (C), E	ED59
OUT (C), H	ED61
OUT (C), L	ED69
OUT (n), A	D3n
OUTD	EDAB
OUTI	EDA3
POPAF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DDE1
POP IY	FDE1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DDE5
PUSH IY	FDE5
RES 0, (HL)	CB86
RES 0, (IX + d)	DDCBd86
RES 0, (IY + d)	FDCBd86
RES 0, A	CB87
RES 0, B	CB80
RES 0, C	CB81
RES 0, D	CB82
RES 0, E	CB83
RES 0, H	CB84
RES 0, L	CB85
RES 1, (HL)	CB8E
RES 1, (IX + d)	DDCBd8E
RES 1, (IY + d)	FDCBd8E
RES 1, A	CB8F
RES 1, B	CB88
RES 1, C	CB89
RES 1, D	CB8A
RES 1, E	CB8B
RES 1, H	CB8C
RES 1, L	CB8D
RES 2, (HL)	CB96
RES 2, (IX + d)	DDCBd96
RES 2, (IY + d)	FDCBd96
RES 2, A	CB97
RES 2, B	CB90
RES 2, C	CB91
RES 2, D	CB92
RES 2, E	CB93
RES 2, H	CB94
RES 2, L	CB95
RES 3, (HL)	CB9E
RES 3, (IX + d)	DDCBd9E
RES 3, (IY + d)	FDCBd9E
RES 3, A	CB9F
RES 3, B	CB98
RES 3, C	CB99
RES 3, D	CB9A
RES 3, E	CB9B
RES 3, H	CB9C
RES 3, L	CB9D
RES 4, (HL)	CBA6
RES 4, (IX + d)	DDCBdA6
RES 4, (IY + d)	FDCBdA6
RES 4, A	CBA7
RES 4, B	CBA0
RES 4, C	CBA1
RES 4, D	CBA2
RES 4, E	CBA3
RES 4, H	CBA4
RES 4, L	CBA5
RES 5, (HL)	CBAE
RES 5, (IX + d)	DDCBdAE
RES 5, (IY + d)	FDCBdAE
RES 5, A	CBAF
RES 5, B	CBA8
RES 5, C	CBA9
RES 5, D	CBAA
RES 5, E	CBAB
RES 5, H	CBAC
RES 5, L	CBAD
RES 6, (HL)	CBB6
RES 6, (IX + d)	DDCBdB6
RES 6, (IY + d)	FDCBdB6
RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX + d)	DDCBdBE
RES 7, (IY + d)	FDCBdBE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC
RES 7, L	CBBD
RET	C9

RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET P0	E0
RET Z	C8
RETI	ED4D
RETN	ED45
RL (HL)	CB16
RL (IX + d)	DDCBd16
RL (IY + d)	FDCBd16
RL A	CB17
RL B	CB10
RL C	CB11
RL D	CB12
RL E	CB13
RL H	CB14
RL L	CB15
RLA	17
RLC (HL)	CB06
RLC (IX + d)	DDCBd06
RLC (IY + d)	FDCBd06
RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03
RLC H	CB04
RLC L	CB05
RLCA	07
RLD	ED6F
RR (HL)	CB1E
RR (IX + d)	DDCBd1E
RR (IY + d)	FDCBd1E
RR A	CB1F
RR B	CB18
RR C	CB19
RR D	CB1A
RR E	CB1B
RR H	CB1C
RR L	CB1D
RRA	1F
RRC (HL)	CB0E
RRC (IX + d)	DDCBd0E
RRC (IY + d)	FDCBd0E
RRC A	CB0F
RRC B	CB08
RRC C	CB09
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D
RRC A	0F
RRD	ED67
RST 0	C7
RST10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF
RST 8	CF
SBC A, (HL)	9E
SBC A, (IX + d)	DD9Ed
SBC A, (IY + d)	FD9Ed
SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D
SBC A, n	DEn
SBC HL, BC	ED42
SBC HL, DE	ED52
SBC HL, HL	ED62
SBC HL, SP	ED72
SCF	37
SET 0, (HL)	CBC6
SET 0, (IX + d)	DDCBdC6
SET 0, (IY + d)	FDCBdC6
SET 0, A	CBC7
SET 0, B	CBC0
SET 0, C	CBC1
SET 0, D	CBC2
SET 0, E	CBC3
SET 0, H	CBC4
SET 0, L	CBC5
SET 1, (HL)	CBCE
SET 1, (IX + d)	DDCBdCE
SET 1, (IY + d)	FDCBdCE
SET 1, A	CBCF
SET 1, B	CBC8
SET 1, C	CBC9
SET 1, D	CBCA
SET 1, E	CBCB
SET 1, H	CBCC
SET 1, L	CBCD

SET 2, (HL)	CBD6
SET 2, (IX + d)	DDCBdD6
SET 2, (IY + d)	FDCBdD6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1
SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX + d)	DDCBdDE
SET 3, (IY + d)	FDCBdDE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX + d)	DDCbE6
SET 4, (IY + d)	FDCBdE6
SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE2
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX + d)	DDCBdEE
SET 5, (IY + d)	FDCBdEE
SET 5, A	CBEF
SET 5, B	CBE8
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX + d)	DDCBdF6
SET 6, (IY + d)	FDCBdF6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1
SET 6, D	CBF2
SET 6, E	CBF3
SET 6, H	CBF4
SET 6, L	CBF5
SET 7, (HL)	CBFE
SET 7, (IX + d)	DDCBdFE
SET 7, (IY + d)	FDCBdFE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB
SET 7, H	CBFC
SET 7, L	CBFD
SLA (HL)	CB26
SLA (IX + d)	DDCBd26
SLA (IY + d)	FDCBd26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX + d)	DDCBd2E
SRA (IY + d)	FDCBd2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX + d)	DDCBd3E
SRL (IY + d)	FDCBd3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX + d)	DD96d
SUB (IY + d)	FD96d
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

SUB n	D6n	XOR C	A9
XOR (HL)	AE	XOR D	AA
XOR (IX + d)	DDAEd	XOR E	AB
XOR (IY + d)	FDAEd	XOR H	AC
XOR A	AF	XOR L	AD
XOR B	A8	XOR n	EEn

ANNEXE 2

TABLEAU DE CONVERSION HEXADÉCIMAL/DÉCIMAL

	0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E	0F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

ANNEXE 3
LE SYSTÈME BINAIRE

Bien que la connaissance du système binaire ne soit pas indispensable à l'apprentissage de la programmation en langage machine, celle-ci peut se révéler utile à la résolution de certains problèmes particuliers. Le principe de la notation binaire est d'ailleurs relativement facile à comprendre. Dans le chapitre consacré au mode de stockage des nombres, il a été indiqué que chaque emplacement mémoire (c'est-à-dire chaque octet) pouvait recevoir un nombre dont la valeur décimale est comprise entre 0 et 255. Cela résulte de la définition même de l'octet. Un octet est formé de l'association de huit unités élémentaires appelées "bits" (le mot "bit" est la contraction de l'expression anglo-saxonne *binary digit* signifiant "chiffre binaire"). Les bits constituant un octet sont numérotés de la manière suivante :

Huit bits forment un octet	7	6	5	4	3	2	1	0

Ces huit bits peuvent être considérés comme représentant autant d'interrupteurs. Un interrupteur peut être ouvert ou fermé. Quand il est ouvert, le courant ne passe pas et le bit correspondant a pour valeur zéro. Quand il est fermé ou, par analogie, lorsque le courant passe, le bit correspondant a pour valeur 1 (on dit dans ce cas que le bit est "positionné"). Cependant, de même que dans le nombre décimal 33, par exemple, les deux chiffres 3 n'ont pas la même importance (il n'ont pas le même " poids "), le numéro du bit dans un octet détermine son poids dans la valeur donnée à l'octet. Le diagramme ci-dessous donne la valeur décimale d'un bit positionné dans un octet :

Valeur du bit	128	64	32	16	8	4	2	1
Numéro du bit	7	6	5	4	3	2	1	0

Ainsi, lorsque le bit 4 est positionné, il contribue pour 16 à la valeur décimale de l'octet. En d'autres termes, si les bits 1 et 3 d'un octet sont positionnés alors que tous les autres ne le sont pas, cet octet aura pour valeur décimale 10. En effet, le bit 1, lorsqu'il est positionné, correspond à 2 (décimal) et le bit 3 à 8. Lorsque tous les bits d'un octet sont positionnés, celui-ci a pour valeur décimale 128+64+32+16+8+4+2+1=255. Lorsqu'au contraire tous les bits sont à zéro, l'octet vaut naturellement 0. C'est pourquoi il est possible

de donner à un octet toute valeur décimale entière comprise entre 0 et 255. Lorsque l'on donne une telle valeur décimale à un octet, au moyen des instructions PEEK et POKE ou de leur équivalent en langage machine, l'ordinateur effectue lui-même la conversion entre les systèmes décimal et binaire, afin de savoir quels bits de l'octet doivent être positionnés pour représenter la valeur décimale que l'on veut stocker.

Si les bits 1, 4 et 6 d'un octet sont positionnés, quelle est la valeur décimale de cet octet ?

Quels bits doivent être positionnés pour que la valeur décimale d'un octet corresponde à 67 ?

ANNEXE 4

DESSIN DE CARACTÈRES ET DE SPRITES

Le programme ci-dessous permet de dessiner très facilement des caractères ou des sprites, ceux-ci pouvant être ensuite utilisés dans d'autres programmes. Il a été précédemment mentionné que la forme de chaque caractère du jeu MSX était définie sur huit octets. Ce programme permet d'assigner les valeurs de son choix à chacun de ces huit octets de forme. Il est destiné à une utilisation ultérieure en mode écran 1. Le programme peut être entré au moyen de ENTHEX et doit être vérifié comme indiqué précédemment.

Adresse initiale 40000
Adresse finale 40702
Total hexadécimal 85482

```
011800    110 DEBUT:  LD    BC,24
11F803    120         LD    DE,1016
215F9D    130         LD    HL,IXSTR
CD5C00    140         CALL  92
3E01      150         LD    A,1
32E3D6    160         LD    (CARCOR),A
32E8D6    170         LD    (VERTIC),A
32E9D6    180         LD    (HORIZ),A
210218    190         LD    HL,6146
22E6D6    200         LD    (CURS),HL
21A08C    210         LD    HL,POSIT
22E4D6    220         LD    (ADDCAR),HL
3E80      230         LD    A,128
32E2D6    240         LD    (BITCAR),A
21D8D6    250         LD    HL,GRILLE
22E0D6    260         LD    (OCTET),HL
CD849D    270         CALL  CARGRL
CD909D    280         CALL  AFFCAR
3E7F      290         LD    A,127
210218    300         LD    HL,6146
CD4D00    310         CALL  77
3E31      320         LD    A,49
210C18    330         LD    HL,6156
```



```
CD4D00    340         CALL  77
3E30      350         LD    A,48
210B18    360         LD    HL,6155
CD4D00    370         CALL  77
3E30      380         LD    A,48
210A18    390         LD    HL,6154
CD4D00    400         CALL  77
CD779D    410         CALL  RAM2V
3E82      420         LD    A,130
218B18    430         LD    HL,6283
CD4D00    440         CALL  77
00        450 BOUCLE  NOP
3E07      460         LD    A,7
CD4101    470         CALL  321
CB57      480         BIT   2,A
C8        490         RET   Z
3E08      500         LD    A,8
CD4101    510         CALL  321
CB47      520         BIT   0,A
CAEC9D    530         JP    Z,INVERS
CB7F      540         BIT   7,A
CA3D9D    550         JP    Z,DROITE
CB77      560         BIT   6,A
CAFB9C    570         JP    Z,CBAS
CB6F      580         BIT   5,A
CAD89C    590         JP    Z,CHAUT
CB67      600         BIT   4,A
CA1F9D    610         JP    Z,GAUCHE
3E04      620         LD    A,4
CD4101    630         CALL  321
CB6F      640         BIT   5,A
CA8B9E    650         JP    Z,PRECED
CB5F      660         BIT   3,A
CA329E    670         JP    Z,SUIV
C39F9C    680         JP    BOUCLE
3AE8D6    690 CHAUT:  LD    A,(VERTIC)
3D        700         DEC   A
CA9F9C    710         JP    Z,BOUCLE
32E8D6    720         LD    (VERTIC),A
2AE0D6    730         LD    HL,(OCTET)
2B        740         DEC   HL
22E0D6    750         LD    (OCTET),HL
```

```
2AE6D6   760           LD     HL,(CURS)
012000   770           LD     BC,32
A7       780           AND    A
ED42     790           SBC    HL,BC
22E6D6   800           LD     (CURS),HL
CD909D   810           CALL   AFFCAR
C39F9C   820           JP     BOUCLE
3AE8D6   830 C BAS:    LD     A,(VERTIC)
D608     840           SUB    8
CA9F9C   850           JP     Z,BOUCLE
C609     860           ADD    A,9
32E8D6   870           LD     (VERTIC),A
2AE0D6   880           LD     HL,(OCTET)
23       890           INC    HL
22E0D6   900           LD     (OCTET),HL
2AE6D6   910           LD     HL,(CURS)
012000   920           LD     BC,32
09       930           ADD    HL,BC
22E6D6   940           LD     (CURS),HL
CD909D   950           CALL   AFFCAR
C39F9C   960           JP     BOUCLE
3AE9D6   970 GAUCHE    LD     A,(HORIZ)
3D       980           DEC    A
CA9F9C   990           JP     Z,BOUCLE
32E9D6   1000          LD     (HORIZ),A
3AE2D6   1010          LD     A,(BITCAR)
87       1020          ADD    A,A
32E2D6   1030          LD     (BITCAR),A
2AE6D6   1040          LD     HL,(CURS)
2B       1050          DEC    HL
22E6D6   1060          LD     (CURS),HL
CD909D   1070          CALL   AFFCAR
C39F9C   1080          JP     BOUCLE
3AE9D6   1090 DROITE   LD     A,(HORIZ)
D608     1100          SUB    8
CA9F9C   1110          JP     Z,BOUCLE
C609     1120          ADD    A,9
32E9D6   1130          LD     (HORIZ),A
3AE2D6   1140          LD     A,(BITCAR)
CB0F     1150          RRC    A
32E2D6   1160          LD     (BITCAR),A
2AE6D6   1170          LD     HL,(CURS)
```



```
23       1180          INC    HL
22E6D6   1190          LD     (CURS),HL
CD909D   1200          CALL   AFFCAR
C39F9C   1210          JP     BOUCLE
FFC3A599 1220 IXSTR:   DEFB   255,195,165,153
99A5C3FF 1330          DEFB   153,165,195,255
FF81BDBD 1240          DEFB   255,129,189,189
BDBD81FF 1250          DEFB   189,189,129,255
FF818181 1260          DEFB   255,129,129,129
818181FF 1270          DEFB   129,129,129,255
010800   1280 RAM2V:   LD     BC,8
111004   1290          LD     DE,1040
2AE4D6   1300          LD     HL,(ADDCAR)
CD5C00   1310          CALL   92
C9       1320          RET
010800   1330 CARGRL   LD     BC,8
11D8D6   1340          LD     DE,GRILLE
2AE4D6   1350          LD     HL,(ADDCAR)
EDB0     1360          LDIR
C9       1370          RET
210218   1380 AFFCAR   LD     HL,6146
11D8D6   1390          LD     DE,GRILLE
3E08     1400          LD     A,8
F5       1410 AFCAR1   PUSH   AF
1A       1420          LD     A,(DE)
CB7F     1430          BIT    7,A
CDDB9D   1440          CALL   BIT
CB77     1450          BIT    6,A
CDDB9D   1460          CALL   BIT
CB6F     1470          BIT    5,A
CDDB9D   1480          CALL   BIT
CB67     1490          BIT    4,A
CDDB9D   1500          CALL   BIT
CB5F     1510          BIT    3,A
CDDB9D   1520          CALL   BIT
CB57     1530          BIT    2,A
CDDB9D   1540          CALL   BIT
CB4F     1550          BIT    1,A
CDDB9D   1560          CALL   BIT
CB47     1570          BIT    0,A
CDDB9D   1580          CALL   BIT
13       1590          INC    DE
```

387

```
011800  1600          LD      BC, 24
09      1610          ADD     HL, BC
F1      1620          POP     AF
3D      1630          DEC     A
C2989D  1640          JP      NZ, AFCAR1
CD779D  1650          CALL    RAM2V
3E7F    1660          LD      A, 127
2AE6D6  1670          LD      HL, (CURS)
CD4D00  1680          CALL    77
CDE39E  1690          CALL    DELAI
C9      1700          RET
F5      1710 BIT:     PUSH    AF
CAE49D  1720          JP      Z, BIT1
3E80    1730          LD      A, 128
C3E69D  1740          JP      BIT2
3E81    1750 BIT1:    LD      A, 129
CD4D00  1760 BIT2:    CALL    77
F1      1770          POP     AF
23      1780          INC     HL
C9      1790          RET
00      1800 INVERS   NOP
2AE0D6  1810          LD      HL, (OCTET)
7E      1820          LD      A, (HL)
47      1830          LD      B, A
3AE2D6  1840          LD      A, (BITCAR)
4F      1850          LD      C, A
3E01    1860          LD      A, 1
CB79    1870 INV1:    BIT     7, C
C2059E  1880          JP      NZ, INV2
CB00    1890          RLC     B
CB01    1900          RLC     C
3C      1910          INC     A
C3F89D  1920          JP      INV1
CB78    1930 INV2:    BIT     7, B
CA0F9E  1940          JP      Z, INV3
CBB8    1950          RES     7, B
C3119E  1960          JP      INV4
CBFF    1970 INV3:    SET     7, A
3D      1980 INV4:    DEC     A
CA149E  1990          JP      Z, INV5
CB08    2000          RRC     B
C3119E  2010          JP      INV4
```

471


```
78      2020 INV5:    LD      A, B
77      2030          LD      (HL), A
CDF29E  2040          CALL    GRLCHR
CD909D  2050          CALL    AFFCAR
CDE39E  2060 ATTENT   CALL    DELAI
3E08    2070          LD      A, 8
CD4101  2080          CALL    321
CB47    2090          BIT     0, A
CA229E  2100          JP      Z, ATTENT
C39F9C  2110          JP      BOUCLE
00      2120 SUIV:    NOP
3AE3D6  2130          LD      A, (CARCOR)
D67E    2140          SUB     126
CA9F9C  2150          JP      Z, BOUCLE
C67F    2160          ADD     A, 127
32E3D6  2170          LD      (CARCOR), A
2AE4D6  2180          LD      HL, (ADDCAR)
010800  2190          LD      BC, 8
09      2200          ADD     HL, BC
22E4D6  2210          LD      (ADDCAR), HL
CD849D  2220          CALL    CARGRL
CD909D  2230          CALL    AFFCAR
210C18  2240          LD      HL, 6156
CD4A00  2250          CALL    74
D639    2260          SUB     57
CA639E  2270          JP      Z, SUIV1
C63A    2280          ADD     A, 58
CD4D00  2290          CALL    77
C39F9C  2300          JP      BOUCLE
3E30    2310 SUIV1:   LD      A, 48
CD4D00  2320          CALL    77
210B18  2330          LD      HL, 6155
CD4A00  2340          CALL    74
D639    2350          SUB     57
CA7B9E  2360          JP      Z, SUIV2
C63A    2370          ADD     A, 58
CD4D00  2380          CALL    77
C39F9C  2390          JP      BOUCLE
3E30    2400 SUIV2:   LD      A, 48
CD4D00  2410          CALL    77
210A18  2420          LD      HL, 6154
3E31    2430          LD      A, 49
```

```
CD4D00    2440          CALL    77
C39F9C    2450          JP      BOUCLE
00        2460 PRECED   NOP
3AE3D6    2470          LD      A,(CARCOR)
3D        2480          DEC     A
CA9F9C    2490          JP      Z,BOUCLE
32E3D6    2500          LD      (CARCOR),A
2AE4D6    2510          LD      HL,(ADDCAR)
010800    2520          LD      BC,8
A7        2530          AND     A
ED42      2540          SBC     HL,BC
22E4D6    2550          LD      (ADDCAR),HL
CD849D    2560          CALL    CARGRL
CD909D    2570          CALL    AFFCAR
210C18    2580          LD      HL,6156
CD4A00    2590          CALL    74
D630      2600          SUB     48
CABB9E    2610          JP      Z,PREC1
C62F      2620          ADD     A,47
CD4D00    2630          CALL    77
C39F9C    2640          JP      BOUCLE
3E39      2650 PREC1:   LD      A,57
CD4D00    2660          CALL    77
210B18    2670          LD      HL,6155
CD4A00    2680          CALL    74
D630      2690          SUB     48
CAD39E    2700          JP      Z,PREC2
C62F      2710          ADD     A,47
CD4D00    2720          CALL    77
C39F9C    2730          JP      BOUCLE
3E39      2740 PREC2:   LD      A,57
CD4D00    2750          CALL    77
210A18    2760          LD      HL,6154
3E30      2770          LD      A,48
CD4D00    2780          CALL    77
C39F9C    2790          JP      BOUCLE
3E32      2800 DELAI:   LD      A,50
F5        2810 DEL:     PUSH    AF
3EFF      2820          LD      A,255
3D        2830 DEL1:    DEC     A
C2E89E    2840          JP      NZ,DEL1
F1        2850          POP     AF
```



```
3D        2860          DEC     A
C2E59E    2870          JP      NZ,DEL
C9        2880          RET
010800    2890 GRLCAR   LD      BC,8
ED5BE4D6  2900          LD      DE,(ADDCAR)
21D8D6    2910          LD      HL,GRILLE
EDB0      2920          LDIR
C9        2930          RET
```

Une fois entré et vérifié, ce programme peut être sauvegardé sur cassette au moyen de l'instruction :

```
BSAVE "CAS:DESSIN",40000,40800
```

Il sera ensuite testé au moyen des deux lignes BASIC suivantes :

```
1000 DEF USR=40000
1005 A=USR (1)
```

Une grille de 8×8 doit être affichée à l'écran. En haut et à gauche apparaît une lettre X. Cette lettre peut être déplacée sur la grille au moyen des touches de déplacement du curseur. Chacun des 64 carrés composant la grille peut être successivement "allumé" ou "éteint" (voir à ce sujet l'annexe consacrée au système binaire). Pour modifier l'état d'un carré, il suffit de placer le curseur (c'est-à-dire la lettre X) sur celui-ci, puis d'appuyer sur la barre d'espacement. Si ce carré était éteint, il s'allume, et inversement. A droite de la grille apparaît le caractère qui est en train d'être dessiné. Le dessin prend forme à mesure que le curseur est déplacé sur la grille.

Ce programme peut être utilisé pour dessiner 126 caractères. Un numéro est affiché en haut de l'écran; il correspond au caractère qui se trouve sur la grille. Pour passer à un autre caractère, il suffit d'appuyer sur la touche N. Cette touche permet de faire défiler les caractères du numéro 1 au numéro 126 tandis que la touche P permet de les faire défiler dans l'autre sens. Naturellement, la grille est vierge tant que les caractères correspondants n'ont pas été dessinés.

Lorsqu'un certain nombre de caractères ont été ainsi définis et que l'on souhaite les sauvegarder sur cassette, la touche ESC doit être activée. Il suffit ensuite de taper :

```
BSAVE "CAS:CARACT",41000,42007
```

Il est conseillé de sauvegarder deux copies (sur deux cassettes différentes) du nouveau jeu de caractères, afin d'éviter tout problème pouvant résulter de l'altération d'une bande magnétique.

La procédure complète pour créer un nouveau jeu de caractères est la suivante :

1. Taper :

```
CLEAR 200,35999
```

2. Taper :

```
BLOAD "CAS:"
```

Ces instructions permettent de charger en mémoire centrale de l'ordinateur le programme de dessin.

3. A ce stade, il est possible de charger un jeu de caractères que l'on a commencé à définir et qui a été sauvegardé sur cassette; pour ce faire, il suffit de taper :

```
BLOAD "CAS:"
```

4. Taper puis exécuter les deux lignes de programme suivantes :

```
1000 DEF USR=40000
1005 A=USR (1)
```

5. Dessiner les caractères de son choix.

6. Appuyer sur la touche ESC.

7. Sauvegarder sur cassette le nouveau jeu de caractères en tapant :

```
BSAVE "CAS:CARACT",36000,38007
```

Ce jeu de caractères peut maintenant être utilisé dans un programme, de la manière suivante :

1. Taper :

```
CLEAR 200,35999
```

2. Taper :

```
SCREEN 1
```

3. Charger le nouveau jeu de caractères en tapant :

```
BLOAD "CAS:"
```

Les 126 caractères se trouvent ainsi chargés en mémoire RAM. Cependant, pour qu'ils puissent être utilisés, une copie doit être transférée en mémoire VRAM. Le transfert peut être réalisé au moyen des quelques instructions BASIC suivantes :

```
10 FOR A=0 TO 2007
20 VPOKE(CARINIT+A),PEEK(36000+A)
30 NEXT A
40 STOP
```

Ce programme charge le nouveau jeu de caractères en mémoire VRAM de sorte que le premier caractère de ce jeu a pour numéro 126 et que le 126e correspond au numéro 251. Le mode écran ne doit pas être modifié une fois que le nouveau jeu de caractères a été chargé en mémoire

VRAM. Une telle modification risquerait d'effacer ces nouveaux caractères au profit de ceux du jeu MSX standard.

Les quelques lignes suivantes permettent de tester que les nouveaux caractères ont bien été chargés en mémoire VRAM :

```
10 FOR A=126 TO 251
20 VPOKE(INITECRAN+A),A
30 NEXT A
40 STOP
```

Le nouveau jeu de caractères peut aussi bien être utilisé dans des programmes BASIC que dans des programmes écrits en langage machine.

Bien que le programme de dessin proposé ne permette en principe que de définir un jeu de 126 caractères, la procédure ci-dessous peut être utilisée pour créer jusqu'à 252 caractères nouveaux :

1.
```
CLEAR 200,35999
```

2. Charger le programe de dessin au moyen de l'instruction :

```
BLOAD "CAS"
```

3. Entrer et exécuter les deux lignes suivantes :

```
1000 DEF USR=40000
1005 A=USR(1)
```

4. Dessiner 126 caractères.

5. Appuyer sur la touche ESC.

6. Sauvegarder ces caractères sur cassette en tapant :

```
BSAVE "CAS:JEU1",36000,38007
```

7. Taper la commande RUN

8. Dessiner un nouveau jeu de 126 caractères.

9. Appuyer sur la touche ESC.

10. Sauvegarder ces caractères sur cassette en tapant :

```
BSAVE " CAS:JEU2 ",36000,38007
```

Pour pouvoir utiliser les 252 caractères dans un programme, la procédure suivante doit être employée :

1.
```
CLEAR 200,35999
```

2.
```
BLOAD "CAS:JEU1"
```

3. Entrer et exécuter le programme suivant :

```
10 FOR A=0 TO 2007
20 VPOKE(CARINIT+A+1008),PEEK(36000+A)
30 NEXT A
40 STOP
```

Les 126 caractères du premier jeu seront ainsi chargés en mémoire VRAM. Leurs numéros seront compris entre 126 et 251.

4. **BLOAD "CAS:JEU2"**

5. Changer la ligne 20 du programme ci-dessus et la remplacer par :

```
20 VPOKE(CARINIT+A),PEEK(A+36000)
```

6. Taper la commande RUN. Le jeu de caractères n° 2 est ainsi transféré en mémoire VRAM, les caractères correspondants ayant un numéro compris entre 0 et 125.

Pour des questions de clarté de l'affichage, il est préférable, lorsque l'on écrit un programme utilisant un grand nombre de caractères redessinés, de conserver les formes originales tant que l'écriture du programme n'est pas terminée.

DESSIN DE SPRITES

Le programme précédent peut également être utilisé pour dessiner des sprites. La procédure pour créer 32 sprites est la suivante :

1. **CLEAR 200,35999**

2. Charger le programme de dessin en tapant l'instruction :

BLOAD "CAS:"

3. Entrer et exécuter les deux lignes suivantes :

```
1000 DEF USR=40000
1005 A=USR(1)
```

4. Dessiner les formes des 32 sprites en les faisant correspondre aux numéros normalement réservés aux caractères 1 à 32.

5. Appuyer sur la touche ESC.

6. Sauvegarder sur cassette les formes dessinées en tapant :

BSAVE "CAS:SP32",36000,38007

Ces sprites pourront être ensuite utilisés dans un programme de la manière suivante :

1. Taper :

```
CLEAR 200,35999
```

2. Charger les sprites en mémoire centrale en tapant l'instruction :

BLOAD "CAS:SP32"

3. Transférer ces données en mémoire en entrant puis en exécutant le programme BASIC suivant :

```
10 FOR A=0 TO 255
20 VPOKE(FORMSPRITE+A),PEEK(A+36000)
30 NEXT A
40 STOP
```

Il suffit ensuite de définir les paramètres d'affichage (attributs) de quelques sprites et de les charger en mémoire VRAM au moyen de l'instruction VPOKE pour les faire apparaître à l'écran.

RÉPONSES AUX QUESTIONS

1. La partie de poids fort du nombre décimal 45621 est 178, sa partie de poids faible étant 53.

2. Le nombre décimal ayant 64 pour partie de poids fort et 31 pour partie de poids faible, est 16415.

3. Si les octets d'adresses 40000 et 40001 contiennent respectivement les valeurs 5d et 15d, le registre HL contiendra la valeur 3845 après exécution de l'instruction LD HL, (40000).

4. Si le registre HL contient la valeur 35621 et que l'instruction LD (40000) ,HL est exécutée, la valeur 37 est alors chargée à l'adresse 40000 et la valeur 139 à l'adresse 40001.

5. L'équivalent décimal de E3h est 227d.

6. Si FBh est la partie de poids fort d'un nombre et CBh sa partie de poids faible, ce nombre vaut 64459d.

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 429

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost.*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash.
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert.*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano.
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar.
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Gouven Harasgue.
240 pages, Réf. 0099

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bonn.*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar.*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut.*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost.*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut.
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks.
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert.
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer.
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost.
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt.
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Raus.*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Delonghi et Caroline Earhart.
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Levinson.
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut.*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella.
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier.*
256 pages, Réf. 336

IBM PC GUIDE DE L'UTILISATEUR
par Jean Lasselle et Carol Ramsey.
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost.*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford.*
320 pages, Réf. 357

GUIDE DU PC DOS *par Richard A. Kane.*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost.*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly.
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre.*
272 pages, Réf. 0118

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks.*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost.*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw.*
192 pages, Réf. 0132

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw.*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir.
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee.
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw.*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella.
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut.*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly.
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly.*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir.
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost.*
160 pages, Réf. 374

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent.
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut.*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer.*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut.
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Gason Kellog.
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks.
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost.
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann.
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay.*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut.*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost.*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert.*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost.*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert.*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton.*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks.*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks.*
618 pages, Réf. 358

APPLICATIONS DU Z80 *par James W. Coffron.*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks.*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks.*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel-Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088
par François Retoreau,
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredu Tekaia et Michele Bidel,
192 pages, Réf. 0132

La plupart de ces ouvrages existent en version anglaise.

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 203.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

ANGLETERRE
Unit 4 - Bourne Industrial Park
Bourne Road, Crayford
Kent DA1 4BU
Tel. : (0322) 57717

SYBEX

Paris • Berkeley • Düsseldorf • Londres

Achevé d'imprimer le 6 septembre 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 3e trimestre 1985 - Imprimeur n° 1840