

GRAPHISME EN ASSEMBLEUR SUR AMSTRAD CPC



FRANCIS PIEROT

MATÉRIEL

GRAPHISME EN ASSEMBLEUR SUR AMSTRAD CPC

CONNAISSEZ-VOUS TOUTE LA COLLECTION AMSTRAD CHEZ P.S.I. ?

Pour les Amstrad CPC 464, 664 et 6128 :

Initiation :

- La découverte de l'Amstrad - Daniel-Jean David
- Exercices en BASIC pour Amstrad - Maurice Charbit

Programmation BASIC :

- 102 programmes pour Amstrad - Jacques Deconchat
- Super jeux pour Amstrad - Jean-François Sehan
- Amstrad en famille - Jean-François Sehan
- Super générateur de caractères sur Amstrad - Jean-François Sehan
- Photographie sur Amstrad et Apple II - Pierrick Moigneau et Xavier de la Tullaye
- Amstrad en musique - Daniel Lemahieu
- Trois étapes vers l'intelligence artificielle sur Amstrad CPC - René Descamps

Maîtrise du BASIC :

- BASIC Amstrad - 1. Méthodes pratiques - Jacques Boigontier et Bruno Césard
- BASIC Amstrad - 2. Programmes et fichiers - Jacques Boigontier
- BASIC Plus : 80 routines sur Amstrad - Michel Martin
- Périphériques et fichiers sur Amstrad - Daniel-Jean David

Langages :

- Assembleur de l'Amstrad - Marcel Henrot
- Création et animations graphiques sur Amstrad CPC - Gilles Fouchard et Jean-Yves Corre
- Clefs pour dBASE II et III - Michel Keller
- Turbo Pascal sur Amstrad - Pierre Brandeis et Frédéric Blanc

Système :

- Clefs pour Amstrad - 1. Système de base - Daniel Martin
- CP/M Plus sur Amstrad 6128 et 8256 - Yvon Dargery
- Clefs pour Amstrad - 2. Système disque - Daniel Martin et Philippe Jadoul

A paraître :

- Intelligence artificielle : langage et formes sur Amstrad - Thierry Lévy-Abégnolli et Olivier Magnan
- Clefs pour Amstrad 8256 - Eric Baumarti

Pour tout problème rencontré dans les ouvrages P.S.I.
vous pouvez nous contacter au numéro ci-dessous :

Numéro Vert/Appel Gratuit en France

05 21 22 01

(Composer tous les chiffres, même en région parisienne)

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1^{er} de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Éditions du P.S.I. - B.P. 86 - 77402 Lagny/Seine-et-Marne cedex
1986

ISBN 2-86595-340-8

MATÉRIEL

GRAPHISME EN ASSEMBLEUR SUR AMSTRAD CPC

FRANCIS PIEROT



**ÉDITIONS DU P.S.I.
1986**

Francis PIEROT a 23 ans. Il pratique la micro-informatique depuis 1979. Ses études en Informatique et son goût pour l'écriture l'ont aiguillé sur le chemin des magazines spécialisés. Après avoir collaboré à des revues d'informatique, il a rejoint le milieu de l'édition de logiciels. Il a notamment dirigé la réalisation de METRO 2018 sur Amstrad.

Ses passions : la science-fiction, les jeux de rôles, la musique, et la micro-informatique.

Sommaire

Introduction	9
Chapitre 1 Les caractéristiques graphiques de l'Amstrad	11
L'organisation matérielle	12
Instructions graphiques	23
Chapitre 2 Utilisation du langage machine sous Basic	31
Assembleur et langage machine	32
Les outils du programmeur	34
Programmation du Z-80	36
Instructions du Z-80	40
Assembleur et Basic	49
Chapitre 3 Les routines graphiques du système	57
Système d'exploitation	58
Tracé de cercles	60
Tracé d'histogrammes	79
Remplissage des zones	105

Chapitre 4 L'accès direct à la mémoire écran	127
Objets graphiques	128
Structure de la mémoire écran	129
Restitution des objets	131
Routine de restitution	148
Routine de mémorisation	150
Remarques	163
Chapitre 5 Codage des objets graphiques	165
Pourquoi compacter ?	166
Méthodes de compactage	166
Algorithme de compactage	168
Décompactage	174
Chapitre 6 Déplacements par calcul d'adresses	179
Gestion du joystick	180
Déplacements par calcul d'adresses	182
Conséquences de la structure de la mémoire écran	184
Chapitre 7 Gestion des objets sur un décor	189
Problèmes de déplacement	190
Le mode XOR	191
Transparence	195
Problèmes de rapidité	199
Gestion des avant-plans	199
Chapitre 8 Système de coordonnées	207
Quel système de coordonnées ?	208
Territoires interdits et collisions	208
Routine de déplacement automatique	220
Chapitre 9 Création des objets et décors	229
Programme de création de dessins	230
Création de décor	241
Compactage de phases d'animation d'un objet	243
Annexe 1 Les mathématiques de l'informatique	247
Annexe 2 Les adresses systèmes utiles	255
Annexe 3 Couleurs et masques	259
Annexe 4 Carte mémoire Amstrad	262
Annexe 5 Carte mémoire des routines	265
Liste des programmes du livre	265
Emplacement des routines LM du livre	267
Variables globales	268

Annexe 6 Structure écran de l'Amstrad	270
Annexe 7 Le jeu d'instruction du Z-80	275
Annexe 8 Lexique et Index	283
Annexe 9 La disquette d'accompagnement	293
Conseils de lecture	297

Introduction

Tout amateur de micro-informatique, une fois franchie la limite du Basic, se retrouve face à un mur épais qu'il parvient difficilement à percer. Ce mur indestructible sépare le monde des programmeurs spécialistes de celui des amateurs. Ses ingrédients : musique, graphisme, langage machine. Trois obstacles infranchissables pour celui qui ne veut pas passer sa vie à programmer des jeux.

C'est la raison d'être de cet ouvrage. Vous y trouverez, outre un grand nombre d'informations concernant le graphisme sur Amstrad, un ensemble de routines détaillées, de méthodes de travail vous permettant de gérer des graphismes sur Amstrad, à la manière des programmeurs spécialisés.

Le but de ce livre est double. Il aborde tous les problèmes de la gestion du graphisme sous un angle pédagogique. Les étapes de la création des routines sont clairement expliquées, et les routines proprement dites dûment commentées. Il est possible d'utiliser les routines sans se forcer à lire la totalité des explications, mais il est aussi possible de s'initier à leur utilisation en progressant page par page.

Même si la lecture de l'ouvrage ne vous donne pas la connaissance absolue et instantanée de tous les mystères du graphisme sur Amstrad (qui pourrait prétendre les connaître tous ?), il est fort probable qu'elle provoquera le déclic, le : "Ça y est, j'y suis !" qui donne envie d'aller plus loin. Et en micro-informatique, l'envie de progresser est le début de la connaissance.

Une dernière mise en garde s'impose toutefois : ce livre n'est pas destiné à vous enseigner le langage machine, mais il peut vous apprendre à l'appliquer efficacement. Toutefois, l'utilisation des routines ne nécessite pas sa connaissance. C'est pourquoi, un chapitre entier a été consacré au langage machine et à son utilisation sous Basic. Le chapitre en question familiarisera avec l'assembleur ceux qui ne le pratiquent pas encore.

Enfin, bien que l'utilisation d'un programme assembleur soit hautement recommandée, toutes les routines seront fournies sous deux formats : listing assemblé, pour programmer directement en assembleur, et programme Basic avec liste de DATA pour ceux qui ne possèdent pas d'assembleur. Les routines seront donc accessibles à tous.

Pour des raisons de commodité, les annexes 1 et 7 sont tirées de l'ouvrage d'Alain Pinaud *Programmer en Assembleur* paru aux Editions du P.S.I.

LES CARACTÉRISTIQUES GRAPHIQUES DE L'AMSTRAD | 1

L'Amstrad n'est pas exceptionnellement doué pour le graphisme, mais il possède quelques particularités originales et intéressantes.

L'ORGANISATION MATÉRIELLE

Les circuits

Amstrad se distingue au premier abord par son contrôleur vidéo. Tandis que les autres constructeurs emploient des processeurs vidéo, avec tracé de ligne incorporé ou gestion d'une mémoire écran indépendante, Amstrad a choisi une attitude radicalement différente. Le circuit utilisé est un CRTC HD 6845. CRTC signifie Cathode Ray Tube Controller, soit contrôleur de tube cathodique. Son rôle est de transformer le contenu de la mémoire écran en signal pour l'écran. L'utilisation du 6845 est peu banale en micro-informatique familiale : en fait, le micro-ordinateur le plus connu utilisant ce circuit pour l'affichage est l'IBM-PC. On ne peut pas réellement parler d'ordinateur familial.

L'utilisation de ce circuit prend toute sa dimension lorsqu'on constate que, malgré sa rusticité, le contrôleur est capable d'afficher 80 colonnes sans problème de lisibilité, alors que beaucoup de micro-ordinateurs ne peuvent pas dépasser 40 colonnes.

De plus, pour pallier les faiblesses du 6845 (dont le seul travail est de générer des signaux vidéo pour le moniteur) les concepteurs de la machine ont fort intelligemment programmé un circuit annexe, le "Gate Array" (en traduction littérale, zone des portes) qui joue le rôle de bouche-trou : tout ce que les circuits standard de l'Amstrad ne savent pas faire, il le fait. Par exemple, nous devons au Gate Array la possibilité de choisir entre 27 couleurs. Le 6845 en est incapable. Le Gate Array est l'interface entre les choix de l'utilisateur (les couleurs, le nombre de colonnes, la résolution graphique, et ainsi de suite) et le contrôleur 6845 proprement dit (*schéma 1.1*).

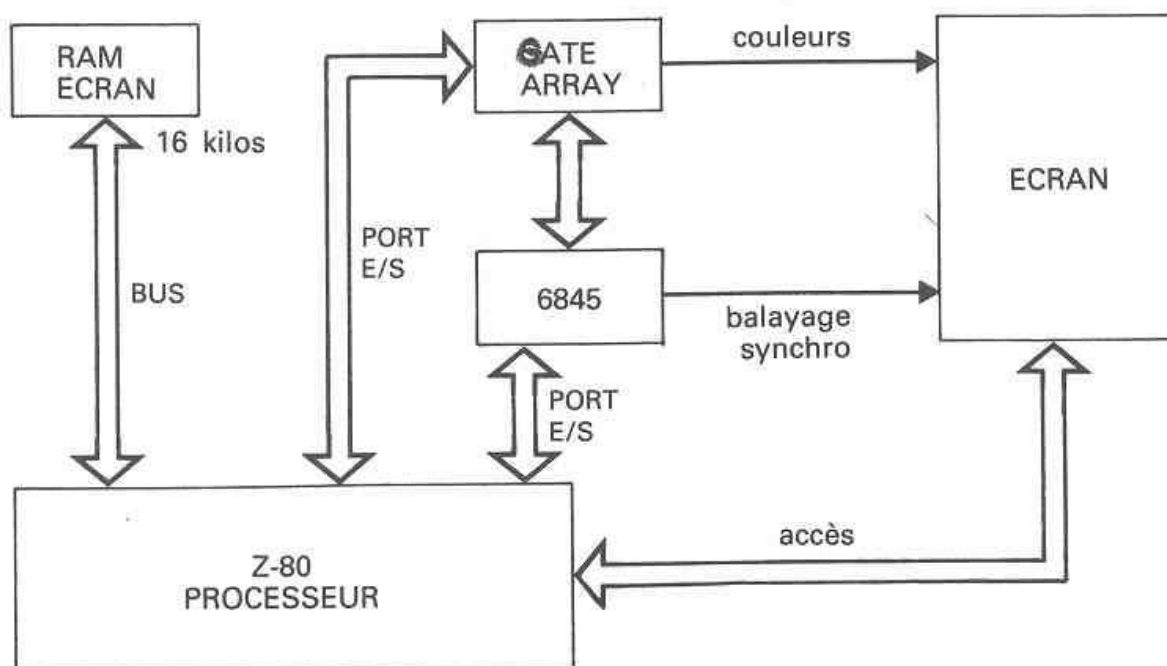


Schéma 1.1

Organisation des éléments visualisateurs.

Bien que les fonctions d'affichage soient toutes disponibles dans le système d'exploitation sous forme de routines, il est des situations où l'on peut vouloir programmer directement le contrôleur vidéo ou le Gate Array. Ces deux circuits sont en effet accessibles au Z-80 par le biais d'instructions OUT.

Le Gate Array (GA) produit les signaux vidéo et RVB pour l'écran. Il s'occupe également des couleurs, des stylos, des modes écran. Il dispose à cet effet de trois registres : un registre "stylo" (recevant les numéros de stylo), un registre "couleur" (acceptant un numéro de couleur) et un registre de commande. Nous appellerons ces registres respectivement STY, COU et COM.

Le port \$7F, ou 127, est dédié au GA. Comme tous les ports sur l'Amstrad, \$7F est la partie haute du numéro de port. Pour y accéder en Basic, on utilisera OUT &7F00, valeur. En langage machine, il faut utiliser la séquence suivante :

```
LD C, valeur
LD B, #7F
OUT (C), C
```

Cette dernière instruction OUT ne doit pas troubler le programmeur : ce n'est pas le registre C qui est utilisé pour le numéro de port mais B, qui est la partie haute de BC.

Les registres du GA ne sont pas accessibles en lecture. On ne peut qu'y écrire. Cela signifie qu'il est par exemple impossible de récupérer le numéro de couleur associé à un stylo si on ne le mémorise pas par ailleurs. Si vous demandez un changement de couleur du stylo 0 par OUT en Basic, cette altération ne va durer qu'une fraction de seconde. En effet, le Basic possède sa propre table de correspondance stylo/couleur, et cette table est régulièrement retransmise au GA. Il est donc inutile de reprogrammer, sous Basic, les couleurs de stylo si l'on ne modifie pas également la table en question.

Suivant la valeur binaire envoyée sur le port \$7F, le GA enverra la donnée sur l'un des trois registres. Deux des huit bits de cette donnée sont, en effet, utilisés pour déterminer le registre visé. Il s'agit des deux bits de gauche :

- une donnée de la forme 00xxxxxx sera envoyée au registre STY ;
- une donnée de la forme 01xxxxxx sera envoyée au registre COU ;
- une donnée de la forme 10xxxxxx sera envoyée au registre COM.

Les six bits restants sont utilisés différemment suivant le registre. Pour le registre STY, ce sont normalement les quatre bits de droite qui forment le numéro de stylo à programmer, à moins que le cinquième bit ne soit à 1, auquel cas c'est la bordure de l'écran qui est visée.

La donnée 0000xxxx placera xxxx dans le registre STY, et permettra alors de programmer, par un autre OUT visant COU, la couleur de ce stylo. L'envoi de 0001xxxx permettra de faire la même chose mais pour la bordure (xxxx est alors ignoré).

Pour le registre COU, seuls les cinq bits de droite sont utilisés afin de connaître le numéro de couleur visé. La couleur ainsi envoyée est associée au stylo dont le numéro se trouve dans le registre STY. On enverra donc la valeur 010xxxxx, xxxxx étant la valeur binaire de la couleur. Rappelons que les couleurs 27 à 31 sont identiques à cinq des autres couleurs.

Le registre COM est plus complexe. Il possède quatre fonctions, pouvant être demandées simultanément (bien qu'elles n'aient pas grand chose à voir entre elles).

Il nous faut, pour examiner ces fonctions, numéroter les bits de la valeur envoyée. Nous utiliserons la notation classique b7 pour le bit 7, le plus à gauche, jusqu'à b0 pour le bit 0, le plus à droite.

Si b4 vaut 1, le compteur de synchronisation verticale est annulé. Ce compteur est utilisé pour générer l'impulsion à la base de toutes les interruptions de l'Amstrad. Le fait de l'annuler retarde cette impulsion, et permet donc de repousser dans le temps la prochaine interruption. b4 est d'une utilisation très puissante, puisqu'on peut grâce à lui intervenir directement sur tout le système d'interruption !

Si b3 vaut 0, les adresses \$C000 à \$FFFF sont connectées sur la ROM du système d'exploitation, sinon elles sont connectées à la RAM écran. Ceci concerne uniquement les accès en lecture, car lors des écritures en mémoire, le Z-80 émet un signal qui connecte automatiquement les RAM.

Si b2 vaut 0, les adresses \$0000 à \$3FFF sont connectées sur la ROM du Basic. Même remarque que pour b3 : ceci ne concerne que les lectures.

Enfin, b1b0 est utilisé pour déterminer la résolution choisie. 00 indique mode 0 (160 points sur 200, 16 stylos), 01 donne le mode 1 (320 sur 200, 4 stylos) et 10 le mode 2 (640 sur 200, 2 stylos).

Cela termine l'exposé des fonctionnalités du GA. Il nous reste le 6845. En l'occurrence, celui-ci peut être utilisé pour de nombreuses tâches peu banales. Le contrôleur 6845 possède 18 registres, et un registre de sélection. La programmation des registres R0 à R15 se fait de la façon suivante :

- sélectionner le registre en envoyant son numéro sur le port \$BC. Ceci programme le numéro dans le registre de sélection. Seuls les cinq derniers bits sont pris en considération, les valeurs 0 à 17 étant les seules effectives ;
- envoyer la donnée sur le port \$BD. Ceci place la donnée dans le registre sélectionné (sauf R16/R17 qui ne peuvent qu'être lus).

Il est également possible de lire le contenu des registres R12 à R17, de la même façon : envoi du numéro de registre sur le port \$BC, lecture de donnée sur le port \$BF et non \$BD.

Examinons en détail les fonctions des registres :

- ☐ **R0** : registre 8 bits. Nombre de caractères par ligne, auquel est ajouté un nombre dépendant de la durée de balayage sur les bords de l'écran et du temps nécessaire pour le retour du faisceau en début de ligne.
- ☐ **R1** : registre 8 bits. Nombre de caractères réellement affichés par ligne. Il s'agit de la valeur de base de ce nombre (voir ci-dessus R0).
- ☐ **R2** : registre 8 bits. Position horizontale. Décale vers la gauche ou la droite l'image écran. La valeur centrale est 46.
- ☐ **R3** : registre 4 bits. Ecartement des impulsions de synchronisation.
- ☐ **R4** : registre 7 bits. Nombre de lignes de grille par image. Permet de régler la synchronisation verticale en fonction de la fréquence (50 Hz ou 60 Hz suivant le pays).
- ☐ **R5** : registre 6 bits. Ajustement du balayage vertical.
- ☐ **R6** : registre 7 bits. Nombre de lignes de caractères affichées (25).
- ☐ **R7** : registre 7 bits. Position verticale. Décale vers le haut ou le bas l'image écran. La valeur centrale est 30.
- ☐ **R8** : registre 2 bits. Saut de ligne. Une valeur de 0 donne l'affichage normal, une valeur de 1 donne un tremblotement.
- ☐ **R9** : registre 5 bits. Nombre de lignes par caractère (7).
- ☐ **R10** : registre 7 bits. Attribut curseur. les bits b0 à b4 indiquent la ligne d'écran où débute le curseur. Les bits b5 et b6 donnent le mode curseur : 00 curseur normal, 01 pas de curseur, 10 curseur clignotant rapide, 11 curseur clignotant lent.
- ☐ **R11** : registre 5 bits. Ligne de fin curseur (voir R10).
- ☐ **R12** : registre 6 bits. Poids fort d'adresse de début de l'écran (on y ajoute \$C000 pour obtenir l'adresse en RAM-écran).
- ☐ **R13** : registre 8 bits. Poids faible de l'adresse de début d'écran.
- ☐ **R14** : registre 6 bits. Poids fort d'adresse curseur (similaire à R12).
- ☐ **R15** : registre 8 bits. Poids faible d'adresse curseur (similaire à R13).
- ☐ **R16** : registre 6 bits. Poids fort adresse écran lorsque le stylo optique émet son signal.
- ☐ **R17** : registre 8 bits. Poids faible adresse écran pour stylo optique (voir R16).

La mémoire écran

Avant toute chose, parlons de la mémoire écran. L'Amstrad, si doué soit-il, ne peut pas afficher une information qu'il ne possède pas en mémoire. Il faut donc que l'écran soit stocké quelque part, si possible à un endroit directement accessible par l'utilisateur. De cette façon, il est possible de modifier le contenu de l'écran en changeant simplement son image en mémoire. Sur certaines machines, la mémoire écran n'est accessible qu'au processeur vidéo, et il faut lui demander explicitement de modifier cette mémoire pour en avoir le droit.

Application concrète : la mémoire écran de l'Amstrad est située aux adresses mémoire de 49152 à 65535. Pour être plus près de la machine, nous pouvons également dire entre C000 et FFFF hexadécimal. Elle est constituée de 16 Ko de mémoire (soit 16384 adresses, 4000 en hexadécimal) qui constituent la totalité de l'image écran. Vous pouvez d'ailleurs le constater très simplement en exécutant le programme 1.1.

```

10 *****
20 ** Programme 1.1 **
30 *****
40 '
50 MODE 1
60 INK 3,20
70 contenu=255
80 FOR adresse=&C000 TO &FFFF
90   POKE adresse,contenu
100 NEXT adresse
110 END

```

La ligne 50 est vitale : nous rencontrerons souvent l'instruction MODE. Elle indique en effet à l'ordinateur (plus exactement au fameux Gate Array) à quel type de résolution correspond le contenu de la mémoire écran. Le mode 1 indique qu'il s'agit du mode 40 colonnes, soit 320 points graphiques en largeur (8 points par colonne).

La ligne 60 range la valeur 255 dans la variable CONTENU. Ce contenu, nous allons le placer dans toute la mémoire écran grâce à l'instruction POKE de la ligne 80 et à la boucle l'encadrant.

Après RUN, vous constatez que tout l'écran se remplit de blanc, point par point. Vous constatez aussi que les lignes successives de tracé ne sont pas dans l'ordre : une ligne est remplie, puis une ligne un peu plus bas, ainsi de suite ; et le processus se reproduit ensuite à partir du haut de l'écran,

décalé d'une ligne. Nous verrons plus loin l'explication de ce phénomène. Pour l'heure, nous avons constaté que, sans utiliser la moindre instruction graphique, nous avons rempli l'écran point par point, en modifiant simplement le contenu de la mémoire écran.

Cette mémoire, nous l'avons vu, occupe 16 Ko. La place n'est ni extensible, ni réductible. Quelle que soit la résolution choisie, l'écran est toujours représenté par ces 16 Ko, ni plus, ni moins. Pourtant, le mode 0 ne permet de disposer que de 32 000 points (160 par ligne et 200 lignes), tandis que le mode 1 et le mode 2 possèdent respectivement 64 000 et 128 000 points.

Mais il faut un certain nombre d'octets pour mémoriser un point. A raison d'un octet par point, il faudrait 32 Ko de mémoire uniquement pour le mode 0, et 128 Ko pour le mode 2. Comment se contenter de ces 16 Ko pour avoir le même nombre de points ? Vous avez pu remarquer, dans la documentation Amstrad, que le nombre de couleurs simultanées à l'écran variait suivant le mode de résolution : 16 en mode 0, 4 en mode 1 et seulement deux en mode 2. Pourquoi ?

Quel que soit le mode, nous venons de constater qu'on ne peut pas réserver un octet par point : il n'y a pas assez de mémoire écran. En revanche, nous pouvons coder plusieurs points par octet, en utilisant les bits (les débutants peuvent consulter l'annexe 8 pour se familiariser avec les octets, bits et autres notions binaires). Il y a en effet 8 bits par octet.

Si nous utilisons des groupes de 4 bits, nous pouvons stocker deux points par octet. Cela nous donne 16 000 octets utilisés qui permettent effectivement de stocker 32 000 points. Mais dans ce cas, chaque information correspondant à un point est formée de 4 bits, ce qui autorise 16 valeurs (2^4). En conclusion, il est possible de donner, dans ce cas, 16 valeurs différentes à un point. Nous avons donc 160*200 points, et 16 couleurs simultanées. Les 16 couleurs sont représentées par une valeur allant de 0 à 15 en mémoire écran.

En mode 1, nous divisons les octets par groupe de 2 bits, de façon à stocker 4 points par octet. Deux bits permettent 4 valeurs différentes, nous ne pourrions donc avoir que 4 couleurs simultanées à l'écran.

Enfin, en mode 2, un point est représenté par un bit unique, ce qui ne permet que 2 valeurs distinctes, donc 2 couleurs.

Le codage des points en mémoire écran

Ces limitations sont plus ou moins gênantes. En effet, il aurait été très agréable de pouvoir disposer de 16 couleurs en 320*200 points, ce qui représente un bon compromis graphique. Mais dans ce cas, il aurait fallu consacrer 32 Ko de mémoire à l'écran, et il ne serait plus resté que 32 Ko pour les programmes au lieu de 48 Ko. L'affichage aurait été plus lent, puisqu'il y aurait eu deux fois plus d'informations à traiter.

Dans une certaine mesure, la limitation à 2 couleurs en mode 80 colonnes n'est pas gênante : pour un jeu, ce mode n'est pas utilisable, et la couleur est inutile pour un traitement de texte. En mode 0, 16 couleurs sont suffisantes, et l'on peut choisir ces 16 couleurs dans l'éventail des 27 disponibles. En jouant sur les demi-teintes et les nuances, on arrive à pallier le manque de points et on obtient tout de même des graphismes honorables. Quant au mode 1, il est presque parfait pour les jeux plus textuels (jeux d'aventure entre autres), et il convient parfaitement aux applications graphiques plus professionnelles (histogrammes, camemberts...) sur lesquelles nous reviendrons.

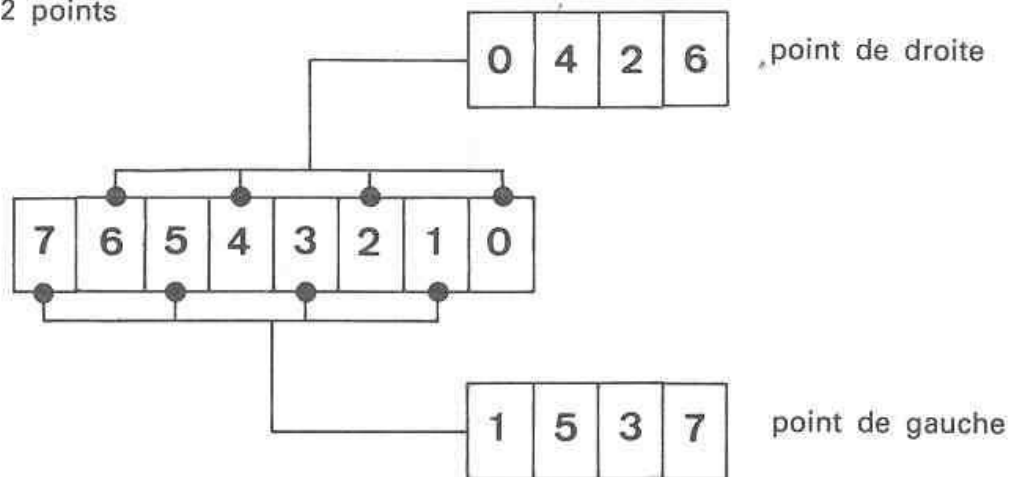
Mais les choses ne sont pas aussi simples. Le circuit 6845 et le Gate Array gèrent 16 Ko de mémoire 50 fois par seconde ! Pour éviter les problèmes, le codage des points en mémoire écran a été optimisé. Vous avez déjà constaté lors de l'exécution du programme 1.1 que les adresses successives en mémoire ne se suivaient pas sur l'écran. En fait, cela est dû à une astuce technique qui permet de synchroniser plus facilement le balayage de l'écran et le rafraîchissement de l'écran. Cette astuce évite, entre autres, la présence de traits noirs parasites lors des modifications de la mémoire écran. Elle donne une qualité graphique et une propreté d'écran remarquables, mais nous verrons vite à quel point elle complique la tâche du programmeur.

Le codage des points en mémoire est, lui aussi, passé à la moulinette de l'optimisation. En effet, les bits propres à chaque point codé dans un octet de la mémoire sont enchevêtrés (techniquement, on dit aussi qu'ils sont entrelacés). Si nous numérotions les bits d'un octet de 7 à 0 (en partant du bit situé le plus à gauche, le plus significatif en terme binaire), voici où sont situés les points, suivant le mode.

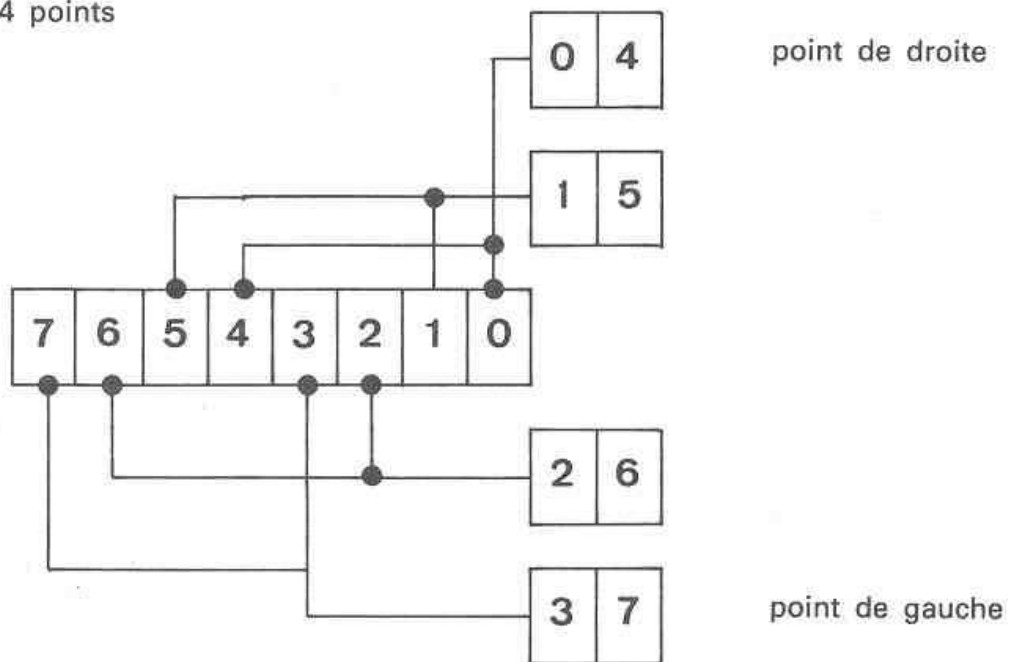
- ☐ **MODE 0** : le point à gauche est stocké dans les bits 1,5,3 et 7 le point à droite dans les bits 0,4,2 et 6.
- ☐ **MODE 1** : le point le plus à gauche dans les bits 3 et 7 puis 2 et 6, puis 1 et 5. Le point le plus à droite dans les bits 0 et 4.
- ☐ **MODE 2** : le point le plus à gauche dans le bit 7, puis dans les bits 6,5,4,3,2,1 et le point le plus à droite dans le bit 0.

Seul le mode 2 suit une certaine logique. Le mode 1 est quant à lui totalement anarchique, en apparence bien entendu (*schéma 1.2. v. p. 19*).

Mode 0 :
1 octet = 2 points



MODE 1 :
1 octet = 4 points



MODE 2 :
1 octet = 8 points

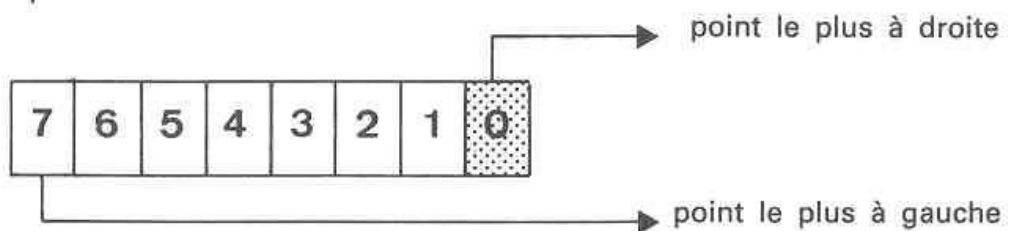


Schéma 1.2

Les huit bits des octets en mémoire écran.

Il est pourtant essentiel de se familiariser avec ces codages. En effet, une grande partie des opérations sur l'écran exécutées dans ce livre nécessite cette connaissance.

Prenons un exemple en mode 1. Les couleurs disponibles dans ce mode sont codées, en mémoire écran, en utilisant les valeurs 0 à 3, soit les valeurs 00, 01, 10 et 11 en binaire.

Si nous voulons allumer les quatre points d'un octet particulier (par exemple celui situé en CA00 hexa, que nous noterons désormais \$CA00) dans les quatre couleurs successives, voici la marche à suivre.

Le premier point est en couleur 00 : les bits 3 et 7 de l'octet devront donc être mis à zéro. Le second point aura la couleur 01 : bit 2 à zéro, bit 6 à un. Ensuite, nous devons mettre 1 au bit 1 et 0 au bit 5 pour avoir la couleur 10 au troisième point, et enfin mettre les bits 0 et 4 à un. Cela se résume en une instruction Basic :

```
POKE &CA00,&X01010011
```

Si vous effectuez cette instruction, un petit amalgame apparaît sur l'écran. Les points sont trop proches pour qu'on puisse véritablement le constater, mais il y a effectivement un point de la couleur 0 (le fond, donc on ne le voit pas !), un autre en couleur 1, puis en couleur 2, puis en couleur 3. Tout cela en un seul octet.

La manipulation, pour le moins complexe, décrite ci-dessus vous donne un aperçu des problèmes de programmation entraînés. Et encore ne s'agissait-il que d'un octet. Le programme 1.2 réalise un remplissage sélectif de l'écran, suivant vos directives.

```
10 *****
20 ** Programme 1.2 **
30 *****
40 '
50 FOR i=0 TO 3:INK i,i*5:NEXT
60 MODE 1:REM mode 320x200, 4 couleurs
70 DIM point(3),puissance(7)
80 'memorisation des puissances de 2.
90 FOR bit=0 TO 7: READ puissance(bit): NEXT bit
100 DATA 1,2,4,8,16,32,64,128
110 '
120 'choix des couleurs des quatres points
130 '
140 FOR p=0 TO 3
150   PRINT "Point numero";p
160   INPUT "Couleur (0 a 3) ";point(p)
170 NEXT p
180 '
```



```

190 'calcul de l'octet representant ces 4 points
200 '
210 FOR p=0 TO 3
220 READ bit1,bit2:REM numero des bits du point
230 'poids fort de la couleur dans le premier bi
    t
240 contenu=contenu+puissance(bit1)*INT(point(p)
    /2)
250 'poids faible de la couleur dans le second b
    it
260 contenu=contenu+puissance(bit2)*((point(p) AN
    D 1)
270 NEXT p
280 DATA 3,7,2,6,1,5,4,0
290 '
300 'remplissage de la memoire par ce contenu
310 '
320 CLS
330 FOR adresse=&C000 TO &FFFF
340     POKE adresse,contenu
350 NEXT adresse

```

Le plus gros du travail est effectué entre les lignes 200 et 270, où le choix de l'utilisateur est transformé en octet. Cette transformation suit les principes expliqués plus haut, concernant le codage en mode 1.

Lorsque nous aurons à gérer ces structures complexes de bits dans des routines en langage machine, il va de soi que nous devrons optimiser leur traitement. En règle générale, les routines travailleront en mode 0, ce qui assure une gestion plus simple qu'en mode 1 (il n'y a que deux points par octet dans ce mode, rappelons-le). Le choix du mode 0 assure, outre une gestion moins lourde des graphismes, un choix de couleurs acceptable.

Couleurs et stylos

Nous l'avons expliqué plus haut, un circuit spécial de l'Amstrad, le Gate Array, est chargé de fournir les 27 nuances de couleurs au 6845. Il est toutefois impossible d'obtenir ces 27 teintes simultanément à l'écran, puisque dans le meilleur des cas (en mode 0), chaque point de l'écran est programmé par une valeur choisie parmi 16 différentes.

Il faut donc, pour utiliser les couleurs voulues, créer une table de correspondance entre chaque code (de 0 à 15 en mode 0, de 0 à 3 en mode 1, et de 0 à 1 en mode 2) et chaque numéro de teinte (de 0 à 26), afin de savoir quelle est la teinte représentée par chaque code de la mémoire écran.

Sous contrôle du Basic, cette table est modifiée par l'instruction INK. Un équivalent existe en langage machine, soit sous la forme d'une instruction du système d'exploitation, soit en programmant directement le Gate Array (plus rapide, mais également plus obscur).

A la mise en marche de l'Amstrad, l'écran est placé en mode 1 et les couleurs suivantes sont assignées :

- code 0 : couleur 1 (bleu foncé) ;
- code 1 : couleur 24 (jaune vif) ;
- code 2 : couleur 20 (bleu pâle vif) ;
- code 3 : couleur 6 (rouge clair).

Si vous tapez alors l'instruction INK 1,0, vous reprogrammez le Gate Array afin qu'il affiche la couleur 0 (noir) pour les points de la mémoire écran contenant le code 1. De jaune, les textes passeront alors immédiatement au noir.

Ce principe de table de correspondance est vital lui aussi. De façon générale, vous pouvez considérer que le code situé en mémoire écran pour un point donné est le numéro d'un stylo avec lequel il a été tracé. Ce stylo, vous pouvez à tout moment en changer la couleur d'encre. Tout ce qui, sur l'écran, a été tracé avec ce stylo subira immédiatement la modification.

Les stylos sont un léger obstacle pour la compréhension, mais ils ont d'énormes avantages. Ils permettent notamment d'utiliser la totalité des 27 couleurs dans un jeu, suivant les tableaux ou les situations représentées, en reprogrammant, au choix, les stylos disponibles pour utiliser des teintes quelconques. Dans tel tableau, le rouge sera utilisé pour le stylo 1, dans un autre ce sera le bleu.

Il est notamment possible d'obtenir des effets de mouvement géométriques facilement grâce aux stylos. Le programme 1.3 est une application de ce procédé.

```

10 '*****
20 '** Programme 1.3 **
30 '*****
40 '
50 MODE 1:DEFINT a-z
60 '
70 'tous les stylos en pale
80 '
90 FOR stylo=1 TO 3: INK stylo,10: NEXT stylo
100 '
110 'trace invisible des ellipses
120 '
130 LOCATE 1,20:PRINT"Patientez SVP..."
140 stylo=1

```

```

150 FOR rayon=10 TO 100 STEP 5
160 DEG
170 MOVE 320+rayon+30,200
180 FOR an=0 TO 360 STEP 8
190   DRAW 320+COS(an)*(rayon+30),200+SIN(an)*ra
      yon,stylo
200   stylo=stylo+1:IF stylo=4 THEN stylo=1
210 NEXT
220 LOCATE 1,24:PRINT INT((100-rayon)/5)
230 NEXT
240 LOCATE 1,24:PRINT "      "
250 LOCATE 1,20:PRINT STRING$(20,32);
260 '
270 'mouvement simule par modification des coule
      urs
280 '
290 INK 1,20:INK 2,10:INK 3,0
300 FOR K=1 TO 300:NEXT
310 INK 1,10:INK 2,0:INK 3,20
320 FOR K=1 TO 300:NEXT
330 INK 1,0:INK 2,20:INK 3,10
340 FOR K=1 TO 300:NEXT
350 GOTO 290

```

INSTRUCTIONS GRAPHIQUES

Organisation Interne

Ce programme introduit les premières instructions graphiques : PLOT et DRAW. A ce niveau, l'Amstrad, malgré deux ou trois omissions regrettables, n'a rien à envier à la plupart des machines.

Une particularité interne singularise l'Amstrad : l'interpréteur Basic, dans la majorité des cas, n'exécute aucune instruction de gestion de la machine. Par exemple, les instructions graphiques sont toutes intégrées au système d'exploitation et non à l'interpréteur. Dans ce cas, le Basic se contente d'appeler le système d'exploitation. Cette façon de procéder diffère radicalement des autres ordinateurs familiaux : elle ressemble plus à un matériel professionnel. Bien que, théoriquement, cela ralentisse légèrement l'exécution des programmes (ce n'est sensible qu'au niveau machine), il en résulte une facilité de programmation en langage machine assez exceptionnelle, puisque toutes les instructions graphiques sont disponibles sous la forme d'un CALL simple. Toutefois, ces instructions seront peu utilisées pour la programmation de jeux rapides, en raison de

leur lenteur. Elles sont cependant très intéressantes pour la réalisation de camemberts, histogrammes. Nous y reviendrons.

Le système de coordonnées

Avant de nous approcher des instructions Basic disponibles, il nous faut examiner le système de coordonnées retenu pour l'écran. A ce niveau encore, les concepteurs de l'Amstrad ont adopté une attitude originale. En effet, quelle que soit la résolution choisie, l'écran est représenté par les coordonnées 0 à 639 (sur l'axe horizontal) et 0 à 399 (pour l'axe vertical). Cela, même si vous n'avez que 160×200 points.

Cette singularité obscurcit sérieusement les premiers pas en Basic : en effet, le point (0,0) est toujours, quel que soit le mode choisi, celui situé dans le coin inférieur gauche de l'écran, et (639,399) dans le coin supérieur droit. En revanche, le point (3,0) est le même que (0,0) en mode 0, il est situé juste à côté de (0,0) en mode 1, et il est situé à trois points de (0,0) en mode 2 !

Pour simplifier cette organisation, nous allons introduire quelques définitions.

Nous appellerons **point logique** un point dont les coordonnées sont représentées par les coordonnées (0,0) à (639,399). En revanche, nous désignerons par **point physique** un point unique sur l'écran, existant réellement. En mode 0, les points physiques auront les coordonnées (0,0) à (159,199). En mode 1, ce sera (0,0) à (319,199) et (0,0) à (639,199) en mode 2.

Il y a 8 points physiques pour un point logique en mode 0, 4 en mode 1, et 2 en mode 2 (*schéma 1.3*)

i multiple de 4, j multiple de 2

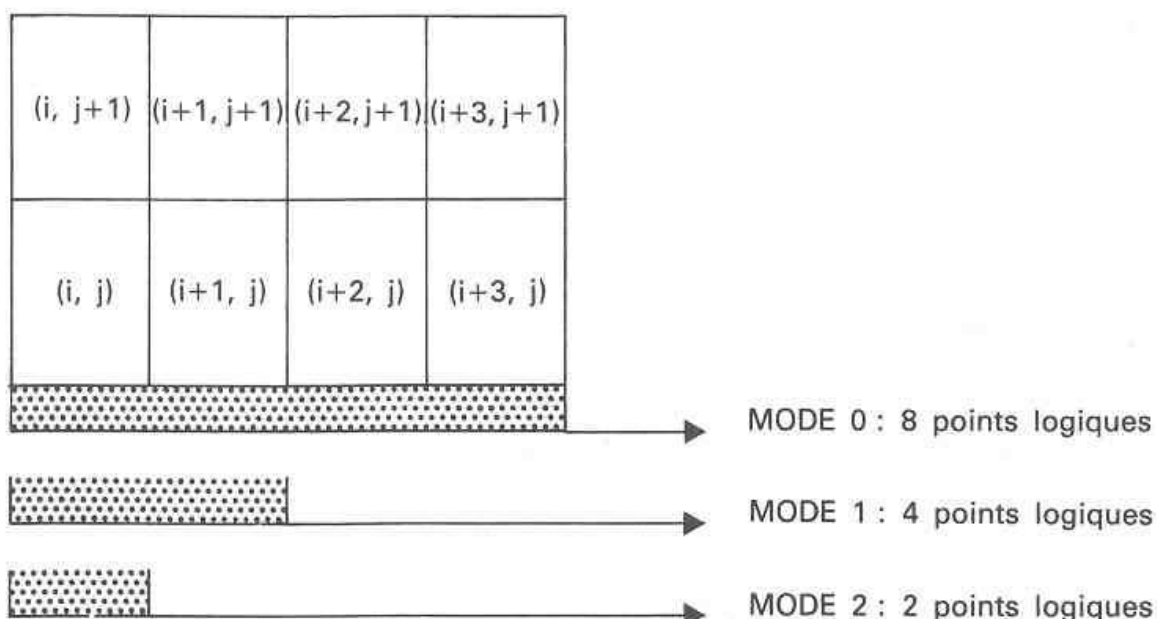


Schéma 1.3

Les points graphiques logiques

Le point physique (0,0) correspond en effet aux points logiques suivants :

- ☐ **MODE 0** : (0,0),(0,1),(1,0),(1,1),(2,0),(2,1),(3,0),(3,1) ;
- ☐ **MODE 1** : (0,0),(0,1),(1,0),(1,1) ;
- ☐ **MODE 2** : (0,0),(0,1).

Les instructions graphiques de l'Amstrad utilisent toutes les coordonnées de points logiques. Certaines doivent disposer des coordonnées des points, d'autres travaillent à partir du nombre de points. Dans ce cas, il faut compter le nombre de points logiques et non physiques.

Les instructions et fonctions graphiques

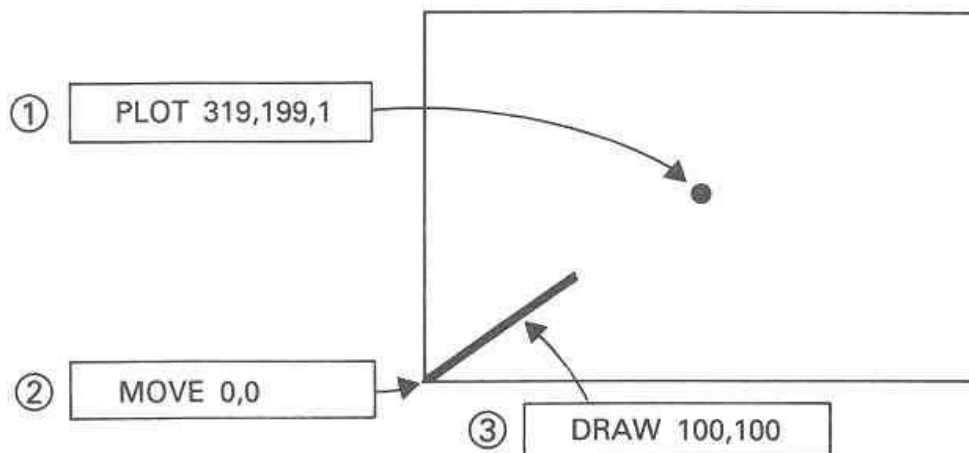
Les plus importantes instructions Basic sont PLOT, DRAW, MOVE ainsi que PLOT, DRAW et MOVER. Il y a également les fonctions XPOS et YPOS ainsi que TEST et TESTR. Nous allons étudier leur fonctionnement.

- ☐ **PLOT** est l'instruction commune à toutes les machines : elle permet de modifier la couleur d'un point logique de l'écran. Par exemple, "PLOT 319,199,1" donne la couleur du stylo 1 au point situé au milieu de l'écran. Petit rappel : quel que soit le mode de résolution choisi, (319,199) est en effet au milieu de l'écran, puisque les coordonnées vont de (0,0) à (639,399). En revanche, si vous effectuez cette instruction dans les trois modes, vous pouvez aisément constater que la taille du point diffère. Pour être précis, le point garde la même hauteur mais il est deux fois moins large en mode 1 qu'en mode 0, et deux fois moins en mode 2 qu'en mode 1.
- ☐ **DRAW** trace une droite dans une couleur donnée jusqu'à un point indiqué, cela à partir du dernier point tracé. Ainsi, après le PLOT ci-dessus, "DRAW 0,0,2" trace une droite du milieu de l'écran au coin inférieur gauche, cela dans la couleur du stylo 2.

L'omission du numéro de stylo provoque l'utilisation du dernier stylo utilisé pour un tracé graphique. Cette particularité discrète permet d'optimiser la taille des instructions graphiques dans les programmes Basic : en effet, il suffit, avant tous les traitements dans une couleur donnée, de positionner le stylo utilisé par un PLOT "bidon", par exemple "PLOT 800,800,stylo". Le point (800,800) n'est pas sur l'écran, il n'est donc pas allumé, mais la couleur est bel et bien sélectionnée. Ensuite, on omettra volontairement le numéro de stylo derrière les instructions graphiques.

- ☐ **MOVE** est un peu l'équivalent de PLOT, mais sans tracé : il permet de positionner le curseur graphique (coordonnées du dernier point tracé) sur n'importe quel point de l'écran. Effectuez par exemple les deux séquences suivantes pour constater la différence (*schéma 1.4 v. p. 26*).

SEQUENCE 1 :



SEQUENCE 2 :

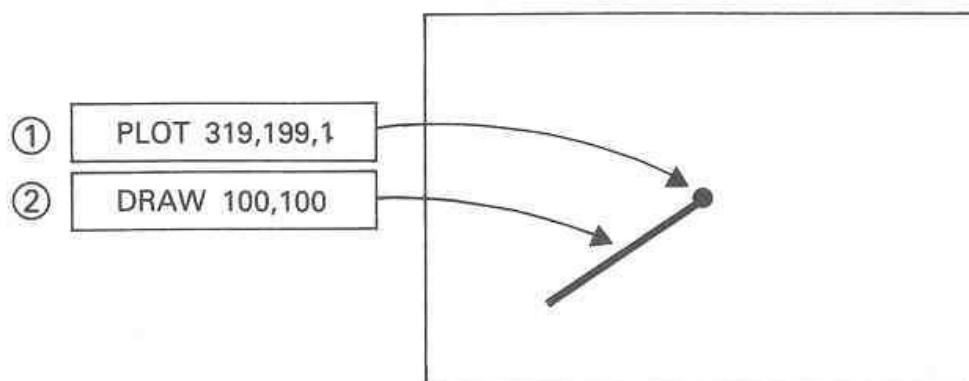


Schéma 1.4

MOVE et DRAW.

Séquence 1
 MODE 1
 PLOT 319,199,1
 MOVE 0,0
 DRAW 100,100

Séquence 2
 MODE 1
 PLOT 319,199,1
 DRAW 100,100

- **PLOTR, DRAWR et MOVER** correspondent exactement à PLOT, DRAW et MOVE, mais au lieu des coordonnées d'un point cible, c'est un déplacement qui est utilisé. Pour l'illustrer, voici à nouveau deux séquences, relativement identiques, d'effet pourtant bien différent.

Séquence 1
 MODE 1
 PLOT 319,199,1
 DRAW 100,100,2

Séquence 2
 MODE 1
 PLOT 319,199,1
 DRAWR 100,100,2

- **XPOS et YPOS** : ces fonctions renvoient la valeur du curseur graphique. Si vous effectuez "PRINT XPOS,YPOS" après les deux séquences précédentes, vous obtiendrez les valeurs suivantes : 100 100 pour la première séquence, 419 299 pour la seconde.
- **TEST (x,y)** : cette fonction renvoie le numéro de stylo correspondant à la couleur du point indiqué.
- **TESTR** fonctionne comme TEST, mais le point à tester est exprimé en nombre de points le séparant de la position actuelle du curseur.

Utilisation des instructions et des fonctions

Dans la pratique, ces fonctions et instructions graphiques ne sont guère utilisées en langage machine, sauf par exemple pour la mise en place de décors particulièrement géométriques. Mais elles sont beaucoup trop lentes pour autoriser une exploitation efficace dans un jeu d'action rapide. En revanche, elles ont une qualité non négligeable : elles ignorent les erreurs sans perturber le fonctionnement des programmes. Mieux, les coordonnées sont toujours prises en compte, même si elles sortent de l'écran. Enfin, la plus grande qualité de ces instructions est leur fonctionnement par système de coordonnées. Les instructions du système d'exploitation qui leur sont associées utilisent le même système, et cela reste essentiel pour réaliser des graphismes à base de motifs géométriques.

Ainsi, vous constaterez aisément que "PLOT 0,0,1" suivie de "DRAW 800,800" trace une droite traversant l'écran, bien que (800,800) soit logiquement en dehors de celui-ci. La droite générée est pourtant celle qui rejoindrait effectivement (800,800) si celui-ci existait (*schéma 1.5*).

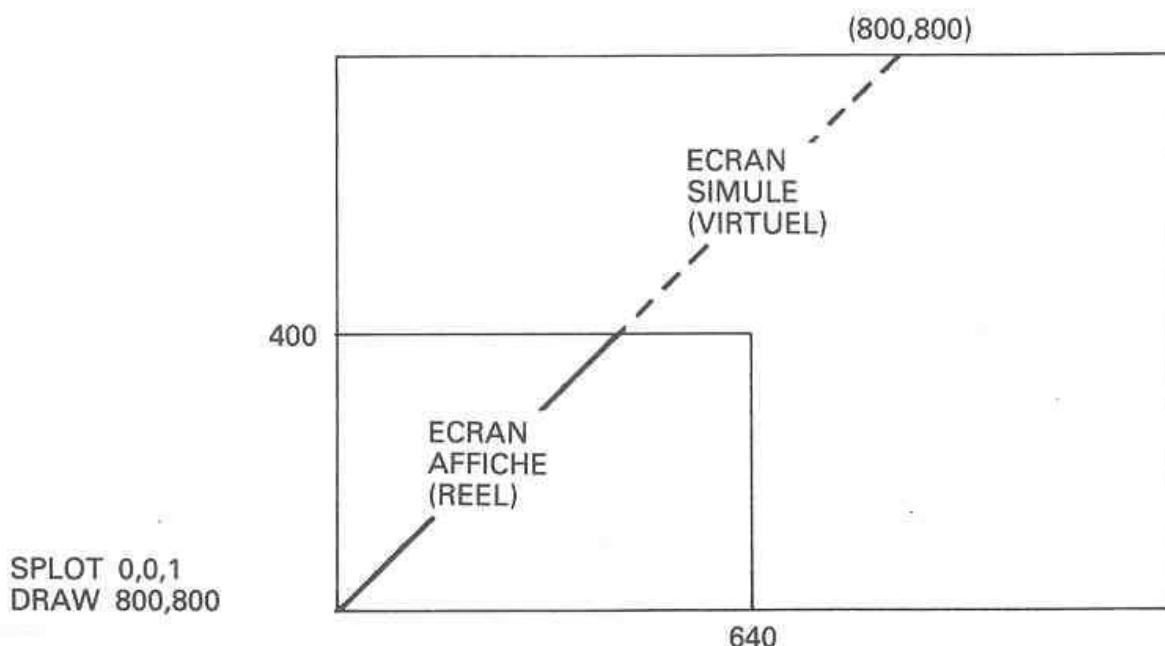


Schéma 1.5

Dépassement de l'écran.

Tout au long de ce chapitre, vous avez pu vous familiariser avec la structure de l'écran et son traitement en Basic, bien que nous soyons volontairement restés proches de la machine. Il est toutefois évident que le Basic ne permet pas la réalisation de jeux rapides, quels qu'ils soient. Pour mettre en évidence la supériorité du langage machine, il suffit d'exécuter chacun des programmes concluant ce chapitre et d'en tirer les conclusions qui s'imposent (programmes 1.4 à 1.7). Nous avons également pris connaissance des instructions graphiques de base, instructions que nous utiliserons plus loin afin de réaliser des utilitaires de tracé de figures. Nous reviendrons sur ces fonctions lors de la réalisation des trois routines dans le chapitre 3.

```

10 '*****
20 '** Programme 1.4 **
30 '*****
40 'duree 42982 (!)
50 'remplissage de l'ecran BASIC par points
60 '
70 MODE 1
80 DEFINT a-y:REM pour aller tres vite
90 PLOT 800,800,1 'selectionne le stylo graphiqu
  e
100 z=TIME
110 FOR y=0 TO 399 STEP 2
120 FOR x=0 TO 639 STEP 2
130 PLOT x,y
140 NEXT x
150 NEXT y
160 PRINT "Duree =";TIME-z

```

```

10 '*****
20 '** Programme 1.5 **
30 '*****
40 'duree 8086
50 'remplissage de l'ecran BASIC par acces memoi
  re ekran
60 '
70 MODE 1
80 DEFINT a-y:REM pour aller tres vite
90 co=&X11110000
100 z=TIME
110 FOR ad=&C000 TO &FFFF
120 POKE ad,co
130 NEXT ad
140 PRINT "Duree =";TIME-z

```

```

10 ;
20 ;remplissage ecran LM par point
30 ;en utilisant PLOT (programme 1.6)
40 ;
BBEA 50 PLOT: EQU #BBEA ;adresse routine PLOT
BBDE 60 INK: EQU #BBDE ;adresse selection stylo
                                graphique
70 ;
4000 80 ORG #4000
4000 3E01 90 LD A,1
4002 CDDEBB 100 CALL INK ;positionne couleur dans AF
110 ;
4005 219001 120 LD HL,400 ;toutes les lignes
130 ;
4008 118002 140 BOUCL1: LD DE,640 ;tous les points de la ligne
400B F5 150 BOUCL2: PUSH AF
400C D5 160 PUSH DE
400D E5 170 PUSH HL
400E CDEABB 180 CALL PLOT
4011 E1 190 POP HL
4012 D1 200 POP DE
4013 F1 210 POP AF
4014 1B 220 DEC DE
4015 1B 230 DEC DE ;x=x-2
4016 F5 240 PUSH AF
4017 D5 250 PUSH DE
4018 E5 260 PUSH HL
4019 CDEABB 270 CALL PLOT
401C E1 280 POP HL
401D D1 290 POP DE
401E 7B 300 LD A,E
401F B2 310 OR D ;teste la fin de la boucle
4020 CA2740 320 JP Z,FIN2 ;fin de la ligne en cours
4023 F1 330 POP AF ;recupere encre
4024 C30B40 340 JP BOUCL2 ;continue la ligne
350 ;
4027 2B 360 FIN2: DEC HL
4028 2B 370 DEC HL ;y=y-2
4029 7D 380 LD A,L
402A B4 390 OR H ;teste fin de boucle
402B CA3240 400 JP Z,FIN1 ;fin du remplissage
402E F1 410 POP AF ;recupere encre
402F C30B40 420 JP BOUCL1
430 ;
4032 F1 440 FIN1: POP AF ;remet pile en etat
4033 C9 450 RET ;fin

```

```

10 *****
20 ** Programme 1.6 **
30 *****
40 'duree 21065
50 'remplissage de l'ecran LM par points
60 '
70 MODE 1
80 MEMORY &3FFF
90 DEFINT a-y:REM pour aller tres vite
100 AD=&4000
110 READ c:IF c=-1 THEN 140
120 POKE ad,c:ad=ad+1:GOTO 110
130 DATA 62,1,205,222,187,33,144,1,17,128,2,245,
      213,229,205,234,187,225,209,241, 27,27,245,21
      3,229,205,234,187,225,209,123,178,202,39,64,2
      41,195,11,64,43,43,125,180,202,50,64,241,195,
      8,64,241,201,-1
140 z=TIME
150 CALL &4000
160 PRINT "Duree =";TIME-z

```

```

10 ;
20 ;remplissage ecran LM par acces memoire ecran
30 ;(programme 1.7)
4000 40      ORG #4000
50 ;
4000 2100C0 60      LD HL,#C000      ;debut memoire ecran
4003 0EF0   70      LD C,%11110000  ;masque pour quatre pts
                                         en couleur 1
80 ;
4005 71     90 LOOP: LD (HL),C      ;remplissage octet
4006 23     100     INC HL          ;octet suivant
4007 7C     110     LD A,H
4008 B5     120     OR L            ;a t'on depasse #FFFF ?
4009 C20540 130     JP NZ,LOOP      ;non:continuer
400C C9     140     RET             ;fin du travail

```

Pass 2 errors: 00

```

10 *****
20 ** Programme 1.7 **
30 *****
40 'duree 57
50 'remplissage de l'ecran LM par acces memoire
   ecran
60 '
70 MODE 1
80 MEMORY &3FFF
90 DEFINT a-y:REM pour aller tres vite
100 AD=&4000
110 READ c:IF c=-1 THEN 140
120 POKE ad,c:ad=ad+1:GOTO 110
130 DATA 33,0,192,14,240,113,35,124,181,194,5,64
      ,201,-1
140 z=TIME
150 CALL &4000
160 PRINT "Duree =";TIME-z

```

UTILISATION DU LANGAGE MACHINE SOUS BASIC | 2

ASSEMBLEUR ET LANGAGE MACHINE

Définitions

Le processeur principal de l'Amstrad est un Z-80. Malgré son âge, ce micro-processeur accomplit brillamment sa mission, gérant notamment 96 Ko de mémoire alors que son champ d'adressage se limite à 64, plus rapidement que sur les machines concurrentes dotées de 64 Ko.

Il va sans dire que pour obtenir de telles performances, les concepteurs de l'Amstrad ont dû recourir à nombre d'astuces. Celles-ci facilitent la tâche du processeur, mais pas toujours celle du programmeur, nous le constaterons souvent à nos dépens.

L'obstacle qui surgit d'emblée lors des premiers pas en assembleur est justement celui de la définition. Quelle est la différence entre l'assembleur et le langage machine ? Il n'y en a pas à proprement parler. Il s'agit bien d'une nuance de définition. Le seul langage compris par un processeur est son langage machine. Ce langage est constitué d'instructions opérant sur des registres, des adresses mémoires, ou des lignes de périphériques. Dans tous les cas, ces instructions sont des nombres. Le stockage de données et des programmes s'effectuant en binaire, il faut donc se résoudre à l'évidence : les seuls programmes exécutables par le processeur sont constitués d'une suite de 0 et de 1. Fort heureusement, les constructeurs de processeurs n'en sont pas restés là. Constatant les difficultés de programmation en binaire, ils ont associé, à chacune des instructions du processeur, un mnémonique, un nom propre qui en facilite la compréhension et la mémorisation. Par exemple, sur le Z-80, l'instruction qui charge le contenu du registre B dans le registre A se matérialise par le nombre \$78 (ou 01111000 en binaire, tel qu'il est effectivement stocké en mémoire) qui est devenu "LD A,B". Toutes les instructions ont ainsi reçu un nom. Nous pouvons maintenant introduire les deux définitions :

- le langage machine est l'ensemble des instructions du processeur sous forme numérique (en binaire, hexadécimal ou décimal) ;
- le langage assembleur est l'ensemble des noms.

Le langage assembleur a une fonction de mise au point : il est beaucoup plus clair de lire "LD A,B" dans un programme que "01111000". Les sources d'erreurs (par inattention surtout) sont d'autant réduites. Mis à part ce point de détail, il est théoriquement équivalent au langage machine. Toutefois, stocker "01111000" dans une case mémoire est facile. Mais que faire de "LD A,B" ?

Avant l'invention des programmes assembleur, la seule solution était la suivante : le programmeur programmat ses routines grâce aux mnémoniques, puis prenait chaque instruction de son programme et la traduisait en langage machine. Ensuite, il rentrait la liste des nombres obtenus en mémoire. Un programme assembleur a pour but d'effectuer ce travail.

Dès lors, le programmeur se retrouve avec un langage facile à mémoriser (en comparaison du langage machine) et à pratiquer. Programmer en langage d'assemblage revient à programmer en langage machine : la frontière qui sépare les deux langages est entièrement abolie par le programme assembleur.

Par abus de langage, le terme assembleur a fini par représenter tout à la fois : le langage machine, le programme assembleur, et même le langage d'assemblage.

Ne vous étonnez donc pas de rencontrer indifféremment les termes langage machine et assembleur dans la suite de l'ouvrage. Ils sont de nos jours synonymes.

Chaque processeur, à l'image d'un interpréteur Basic, possède ses particularités et ses instructions. L'ensemble des instructions forme donc ce que l'on appelle le langage machine. Dans l'essentiel, il s'agit d'instructions plutôt primitives, ne sachant pas faire grand chose. Pourtant, tout ce qui peut être fait sur l'Amstrad l'est dans ce langage. Il faut donc croire que le langage machine, sous son apparente rusticité, représente une réelle puissance.

Puissance et vitesse

La puissance du langage machine vient de deux facteurs. Sa vitesse de traitement, tout d'abord. Vous avez pu constater, au chapitre précédent, qu'un simple programme consistant à remplir 16 Ko de mémoire avec une valeur donnée s'exécutait 142 fois plus vite en langage machine (nous l'appellerons désormais LM) qu'en Basic. Même lorsque le Basic était très proche du LM (lors du remplissage de l'écran point par point), le LM s'exécutait tout de même deux fois plus rapidement.

Deuxième facteur important dans le langage machine : sa compacité. Le programme 1.1, variables comprises, occupait 143 octets. Son équivalent LM (programme 1.7) n'en nécessite que 13.

Enfin, outre ces deux facteurs, il est un troisième avantage plus discret : en programmant en LM on accède simplement à toutes les ressources de la machine. C'est dire qu'en LM, rien n'est impossible, dans la mesure où l'on connaît les limites matérielles du possible. Evidemment, on ne fera jamais afficher 28 couleurs à l'Amstrad : il n'en a que 27 par construction !

Par contre, pour mettre en évidence la puissance du langage machine, il est parfaitement possible d'afficher plus de 16 couleurs différentes sur un même écran, en modifiant les couleurs des stylos en LM d'une certaine façon. En effet, il est parfaitement possible d'exécuter un tel changement de couleur sans que le balayage de l'écran ne soit totalement fini. Le balayage commence alors que le stylo possède une certaine couleur. Arrivé par exemple au milieu de l'écran, si le stylo change de couleur, le

balayage se terminera avec cette nouvelle couleur. Comme vous le constatez, le LM repousse bien loin les limites expliquées dans la documentation Amstrad.

Toutefois, le LM possède un inconvénient énorme : il s'agit, à l'heure actuelle, du langage le plus difficile d'accès, bien qu'il ne soit pas le plus difficile à pratiquer. Alors que tous les autres langages (Basic, Pascal, C, Ada, Forth, etc.), se prêtent bien à un apprentissage en douceur, le LM s'apprend très différemment. En l'occurrence, les débutants apprennent vite à connaître les instructions du Z-80 (ou d'un autre processeur), mais ils se heurtent inévitablement à un mur apparemment infranchissable dès qu'ils tentent de les pratiquer. La persévérance et la patience aidant, au bout d'une période variable, le débutant a l'impression pénible de patiner sur place, et, soudain, le mur s'écroule, et tout devient clair. Le programmeur a alors l'impression que rien ne peut plus lui résister.

La raison de cette sensation est la puissance et la polyvalence même du LM. Comme expliqué plus haut, tout est possible en LM.

LES OUTILS DU PROGRAMMEUR

Pour attaquer le LM, il faut des outils. Comme il en existe beaucoup, chacun fait son choix une bonne fois pour toutes. Une fois ces outils choisis, le programmeur y sera si attaché qu'il ne fera rien pour en avoir de plus puissants ou de plus complets. L'essentiel reste de bien connaître ses outils.

Voici les outils du programmeur LM :

- un livre d'initiation au langage machine Z-80, mettant en évidence les pièges grossiers du processeur, dans lesquels même le plus expérimenté des programmeurs peut tomber par inattention ! Nous vous recommandons particulièrement *Programmer en assembleur* d'Alain Pinaud (publié aux éditions du P.S.I.). Bien qu'assez ancien, ce livre reste le meilleur de sa catégorie, de loin, et possède une table complète des instructions Z-80 ;
- un livre de référence sur le Z-80. Cet achat sera le plus onéreux, mais pas le moins utile. Il en existe quelques-uns, dont la bible, à savoir *Programmation du Z-80* de Rodney Zaks, chez Sybex. Au début, il ne sera toutefois pas utile, mais son besoin se fera sentir lorsque vous en viendrez à calculer le temps d'horloge demandé par une routine pour l'optimiser et accélérer son exécution ;
- un assembleur. Pour l'Amstrad, DEVPAC est sans doute l'un des meilleurs en raison des possibilités d'inclusion de fichiers externes. Mais là aussi, il existe plusieurs assembleurs dignes d'intérêt, sans que l'on puisse affirmer à 100 % lequel est le meilleur. Certains préfèrent DAMS, d'autres ZEN. L'essentiel est de bien connaître les possibilités de son assembleur, le reste est affaire de goût ;

– un debugger/désassembleur. Un tel outil permet par exemple d'exécuter un programme instruction par instruction, ce qui est parfois utile pour trouver une erreur. Les assembleurs récents (ils le sont tous sur la machine qui nous intéresse) possèdent un debugger intégré ou séparé compatible avec l'assembleur ;

– si possible, un lecteur de disquettes. Le besoin ne se fera pas trop sentir pour les routines du livre, toutefois le programmeur réalisant ses propres programmes craquera vite s'il doit recharger l'assembleur à chaque fois que son programme plante l'ordinateur. Il va de soi que l'achat d'un lecteur de disquettes représente un certain investissement, mais de nos jours cet investissement est vite amorti. Si vous possédez un 664 ou un 6128, il va également de soi que notre remarque devient caduque !

– dès que possible, et même avant le lecteur de disquettes, une imprimante. Programmer en assembleur sans imprimante est acceptable si les routines ne dépassent pas deux pages écran. Au-delà, cela frôle le suicide. Ceux qui ne possèdent pas d'imprimante en subissent les conséquences, c'est-à-dire qu'ils sont leur propre imprimante. En clair, lorsqu'il y a une erreur, ils recopient le programme à la main sur papier pour y voir clair ;

– concernant l'Amstrad, il existe trois documentations essentielles. On juge trop souvent les micro-ordinateurs par leurs performances techniques ou les logiciels disponibles. Mais ces éléments, au contraire, sont secondaires. Ce qui compte le plus, c'est la documentation disponible et sa qualité. Hormis le manuel du constructeur, qui ne suffit pas dès que l'on attaque le LM, trois ouvrages très facile à trouver sont à acheter au même titre que le programme assembleur. *CPC 464 FIRMWARE*, publié par Amsoft, résume la totalité des particularités du logiciel interne de l'Amstrad. Il a été écrit par l'un des auteurs de ce logiciel, ce qui est une assurance inégalable. Et non seulement ce manuel dit tout, mais il est extrêmement bien organisé, ce qui ne gâche rien. Si vous rechignez à vous procurer ce gros classeur en langue anglaise, P.S.I. a publié *Clefs pour l'Amstrad* qui est un excellent condensé sur l'Amstrad. Tout y est, bien rangé, facilement accessible, mais attention : ce livre sera surtout utile à ceux qui auront lu le *Firmware*. *Clefs* est au *Firmware* ce que le dictionnaire est à l'encyclopédie ;

– enfin, Micro-application a traduit *la bible du programmeur de l'Amstrad*, qui est un horrible fouillis, mais qui explique en long, en large et en travers tout ce qu'Amstrad n'a pas voulu dire. La programmation directe des circuits y est notamment disséquée, et le livre comporte le listing de la totalité du logiciel interne du CPC 464. Attention, cet ouvrage n'est pas facilement assimilable, il est en tout cas inutile pour ceux qui veulent se familiariser avec le Z-80 ! Mais il sera une aide inégalée pour ceux qui veulent fouiller leur Amstrad en LM, par exemple pour détourner des routines système ou travailler directement avec le 6845 !

PROGRAMMATION DU Z-80

Le processeur

Le Z-80 possède sa propre philosophie. Sa programmation passe principalement par une utilisation appropriée des bons registres aux bons moments. Le nombre de registres du Z-80 est en effet assez élevé, ce qui est un énorme avantage par rapport à la majorité des autres processeurs 8 bits (*schéma 2.1*).

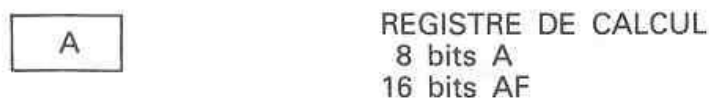
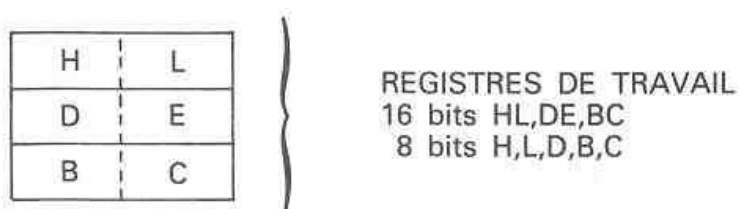




Schéma 2.1

Les registres du processeur Z-80.

En revanche, les possibilités d'accès à la mémoire (que l'on regroupe sous l'appellation de modes d'adressage) sont moins étendues qu'on ne le voudrait. De plus, bien que le nombre d'instructions soit assez impressionnant, il en manque certaines que tout débutant a pourtant tendance à inventer, ce qui conduit inévitablement à une erreur d'assemblage et donc à une correction de l'instruction, pas toujours évidente. Mais dans l'ensemble, l'expérience des défauts vient avec la pratique.

Les registres du Z-80 sont tous de type 8 bits, sauf quatre registres 16 bits utilisés à des fins très particulières. Toutefois, certains des registres 8 bits peuvent être traités par paire pour former des registres 16 bits, par exemple pour traiter des adresses.

Le Z-80 possède d'autre part un second jeu de registres, que l'on peut échanger contre le jeu principal. Cela est utile pour sauver facilement les registres avant d'entamer un travail provisoire. Ces registres sont d'ailleurs utilisés sur l'Amstrad lors des interruptions. Malheureusement, ceci en interdit l'emploi pour la programmation, à moins de détourner les interruptions – ce qui sort largement du cadre de cet ouvrage. La documentation Amstrad *Firmware* indique d'ailleurs comment procéder. Il est dommage que la modification n'ait pas été prévue à l'origine, car les registres secondaires ne sont pas toujours inutiles. Ça ne devrait toutefois pas être un problème pour les débutants, s'ils apprennent le Z-80 en ne les utilisant pas.

Les registres 8 bits

Les registres 8 bits les plus utilisés sont de loin A et F. A est le registre "accumulateur". Il reçoit les résultats des instructions de calcul du Z-80 (addition, soustraction, rotations de bits, opérations logiques, etc.). F est son associé : il résume le résultat de l'opération sous la forme de drapeaux. Il y a par exemple le drapeau Z qui est levé si le résultat est nul, et baissé sinon. F reçoit également sous cette même forme les résultats d'autres opérations (par exemple, les comparaisons entre A et un nombre ou un autre registre) même si le contenu de A ne change pas à cette occasion.

On ne peut pas stocker de valeur particulière dans F. Tout au plus peut-on lever ou baisser certains drapeaux, et sauver ou récupérer ce registre en même temps que A. En revanche, A est extrêmement souple et la totalité des modes d'adressage peut faire appel à A, alors que certains ne sont pas disponibles sur les autres registres.

Deux autres registres 8 bits sont très utilisés : H et L. On les utilise d'ailleurs le plus souvent sous la forme 16 bits (registre HL), afin de stocker des adresses. En effet, HL accède à un grand nombre de modes d'adressage et est donc idéal pour des travaux sur des tables de données, ou pour pointer sur des données.

Les quatre autres registres 8 bits manipulables sont nommés B, C, D et E. Ils sont moins souples que H et L : les registres BC et DE ne possèdent pas tous les modes d'adressage de HL. En revanche, BC et B sont utilisés comme compteur par toutes les instructions Z-80 travaillant à partir d'une boucle. DE possède également ses caractéristiques propres, lors des instructions de recherche ou transfert de chaînes (LDIR, CPIR et les instructions du même style).

Il est à noter que deux registres 8 bits spéciaux sont disponibles : il s'agit de I et de R. Leur utilisation est réservée au processeur. I est le registre d'interruptions, indiquant quel est le type de l'interruption. R servait dans le temps au rafraîchissement de la mémoire. Il est utile pour générer des nombres au hasard, car sa valeur est constamment modifiée et de façon relativement imprévisible.

Les registres 16 bits

Enfin, les registres IX et IY, de 16 bits exclusivement (il ne s'agit pas de registres Y et X regroupés avec le registre I) ont presque toutes les fonctionnalités de HL, plus une : ils permettent un mode d'adressage dit "indexé", que HL ne possède pas. Mais IX et IY sont "en plus" sur le Z-80, et il faut savoir que les instructions s'y rapportant sont beaucoup moins rapides que leurs équivalents travaillant avec HL. On les réservera à l'utilisation de l'adressage indexé, ou bien pour pointer sur des tables comme HL si l'on doit en traiter simultanément plus d'une.

Il existe deux autres registres 16 bits utilisés très peu par le programmeur et très souvent par le processeur : PC et SP. PC est le pointeur de programme : c'est lui qui pointe l'instruction en cours d'exécution. Il y a quelques instructions permettant de modifier son contenu, et un grand nombre le faisant sans en avoir l'air. Entre autres, toutes les instructions JP et JR modifient PC (JP et JR sont exactement équivalentes à un GOTO Basic). Quant à SP, il pointe sur la pile du Z-80.

La pile

La pile est un endroit de longueur limitée mais indéterminée (il faut s'arranger pour que la longueur qu'on lui octroie soit suffisante) où sont rangées toutes les informations temporaires des programmes. Par exemple, lorsque le Z-80 rencontre une instruction CALL (équivalent en Basic : GOSUB), il range l'endroit de l'instruction en cours dans la pile, va exécuter le programme se situant à l'adresse demandée, et revient après le CALL en récupérant l'adresse dans la pile (schéma 2.2).

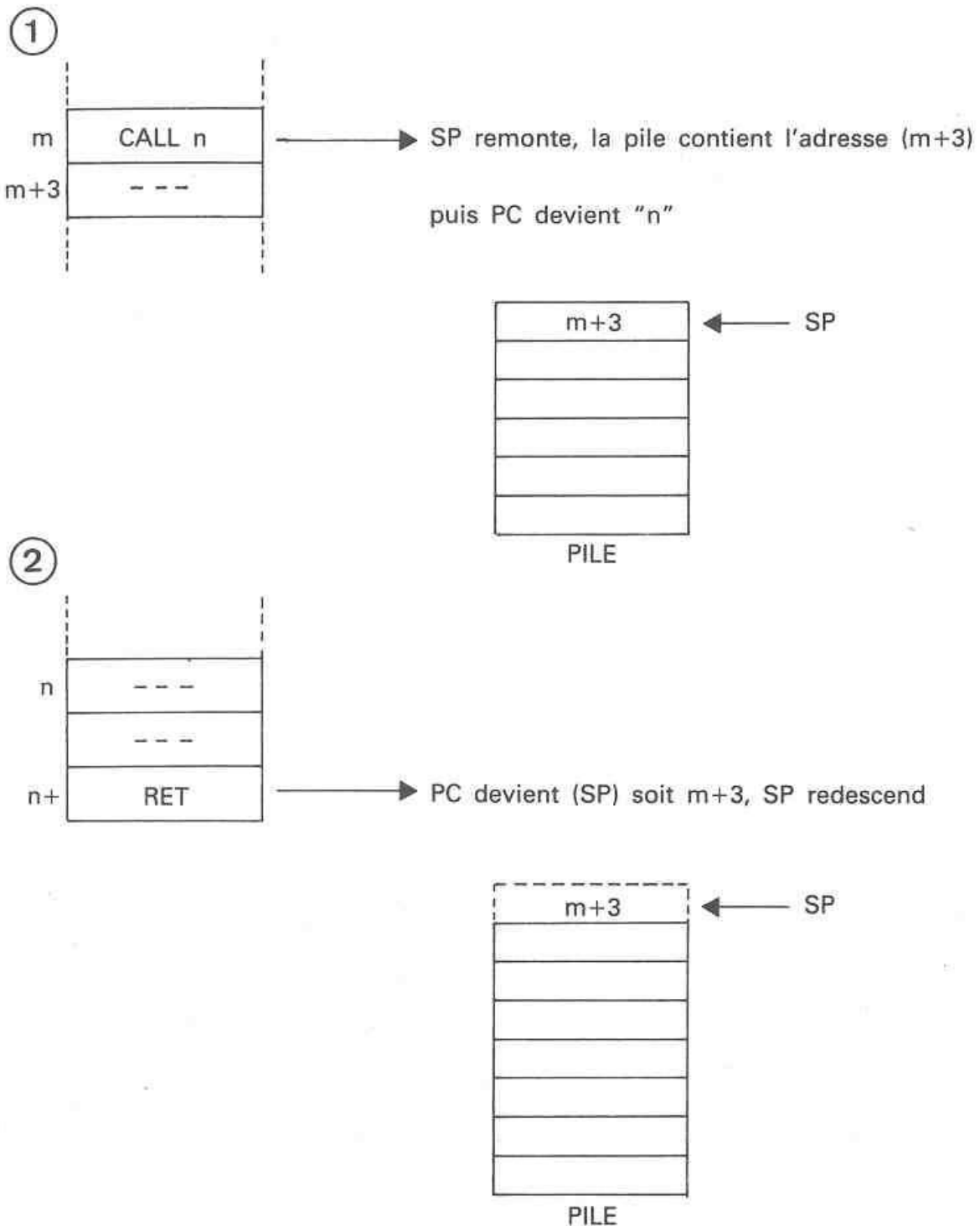


Schéma 2.2

L'utilisateur a la possibilité de travailler avec cette pile, et aussi avec d'autres piles, puisque des instructions permettent de sauver et de récupérer le contenu de SP. Toutefois, si l'utilisateur veut utiliser deux piles, il devra gérer ses deux pointeurs (certains processeurs, comme le 6809, possèdent deux pointeurs de piles, ce n'est hélas pas le cas du Z-80).

La pile est souvent utilisée dans les programmes pour sauver rapidement des registres avant un travail (il serait plus simple de les sauver dans le second jeu de registres, mais nous avons vu plus haut que celui-ci est inutilisable sur l'Amstrad). Elle est très pratique mais aussi très dangereuse. En effet, si le nombre d'instructions rangeant quelque chose dans la pile n'est pas le même que celui des instructions les récupérant, le risque de planter la machine est grand, et celui d'obtenir des erreurs de fonctionnement du programme est énorme. Imaginez une pile d'assiettes en porcelaine : si vous enlevez plus d'assiettes qu'il n'y en a dans la pile, il y a un imprévu. Inversement, si vous laissez une assiette avant de secouer la nappe, il va y avoir de la casse. En Z-80, si une routine appelée par CALL laisse une adresse de trop, le retour va se faire à l'endroit indiqué par celle-ci, qui n'aura rien à voir avec la bonne adresse se trouvant juste dessous. De même, si la routine enlève une adresse de trop, le retour se fera aussi à un endroit n'ayant rien d'intéressant.

Les registres secondaires

Les registres secondaires ne sont pas utilisables sur l'Amstrad. Il est toutefois intéressant de connaître leur existence. Ils sont au nombre de huit : A', F', B', C', D', E', H' et L'. Ils sont exactement équivalents, en temps normal, aux registres A à L. Attention : il n'existe pas de IX', IY', PC' ni SP'.

INSTRUCTIONS DU Z-80

Introduction

Le Z-80 possède de nombreuses instructions. Elles peuvent toutes plus ou moins utiliser les modes d'adressage disponibles. Rappelons qu'un mode d'adressage est une façon d'accéder à une information. Le problème du Z-80 est qu'il n'est pas orthogonal. Un processeur est dit orthogonal quand tous les modes d'adressage sont utilisables sur toutes les instructions, ce qui est loin d'être le cas en Z-80.

Pour simplifier, nous n'allons pas passer en revue les différents modes d'adressage du Z-80. En effet, si ces modes sont facilement identifiables au niveau du code machine (les nombres correspondant aux instructions), il en est autrement au niveau des instructions proprement dites, sous leur forme assembleur comme JP ou ADD. A ce niveau, les modes d'adressage sont moins évidents. Par exemple, "JR adresse" est une instruction qui

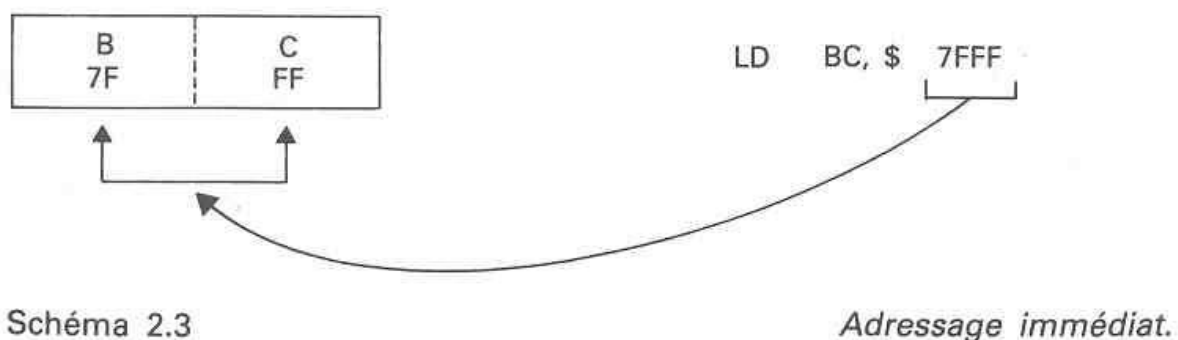
semble utiliser l'adressage absolu, puisqu'une adresse est précisée. En fait, il s'agit d'adressage relatif ! En effet, c'est le programme assembleur qui transforme l'adresse en un autre type de donnée. Ce genre de subtilité est un véritable piège pour le débutant, qui aura bien du mal à faire la différence entre "JR adresse" et "JP adresse, qui font la même chose et se présentent de la même façon. Le Z-80, par contre, fait la différence au niveau codage, encombrement, rapidité et fonctionnement.

Un débutant en Z-80 est vite mis en difficulté par la quantité d'instructions disponibles. Mais au début, seul un certain nombre d'entre elles sont véritablement utiles. C'est la liste de ces instructions vitales que nous allons étudier, ignorant volontairement un bon nombre d'instructions.

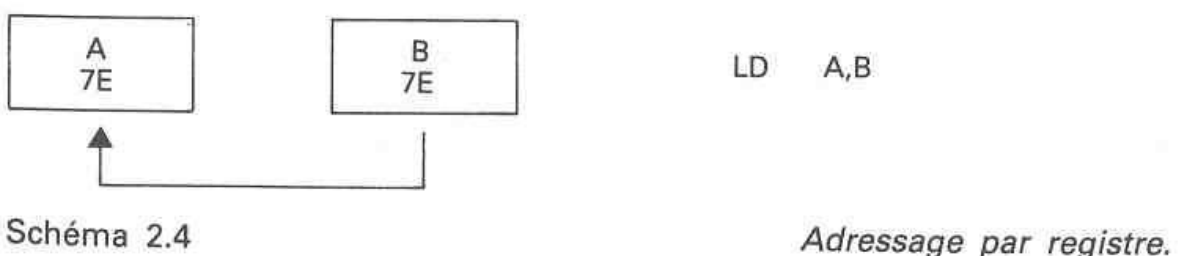
Accès mémoire, chargements des registres

Tout d'abord, car il s'agit sans doute des instructions les plus utilisées, nous allons passer en revue les chargements de registres. Ces instructions LD permettent de stocker une valeur dans chacun des registres du Z-80. Mais il y a plusieurs façons de préciser la donnée en question.

On peut dans tous les cas préciser directement la valeur : il s'agit d'adressage immédiat. Par exemple, "LD A,34" ou "LD BC,32767". Cette façon de charger une valeur dans un registre est réservée aux initialisations (schéma 2.3).



Il est aussi possible, la plupart du temps, de charger un registre avec la valeur contenue dans un autre registre. Par exemple, "LD A,B" charge le contenu de B dans A (schéma 2.4).



Attention : ce type de chargement ne fonctionne que sur les registres 8 bits. Il n'existe pas de "LD BC,HL", par exemple. Pour obtenir la même chose, il faut procéder en deux instructions, en l'occurrence "LD B,H" suivi de "LD C,L". Seul SP fait exception, puisqu'il est possible de charger directement dans ce registre les contenus de HL, IX ou IY. Cela permet de gérer assez facilement plusieurs piles.

Il est possible, pour tous les registres 16 bits (BC, DE et HL mais aussi IX, IY et SP) de charger une donnée 16 bits en indiquant l'adresse où elle se situe. On indique alors cette adresse entre deux parenthèses. Ainsi, "LD BC, (4000)" permet de charger les deux octets situés en 4000 et 4001 dans les registres respectifs C et B (car le premier octet en mémoire est celui de poids faible, c'est-à-dire situé à droite dans un nombre 16 bits). Cette possibilité est aussi offerte au registre A, ainsi qu'à SP (schéma 2.5).

LD BC,(\$4000)

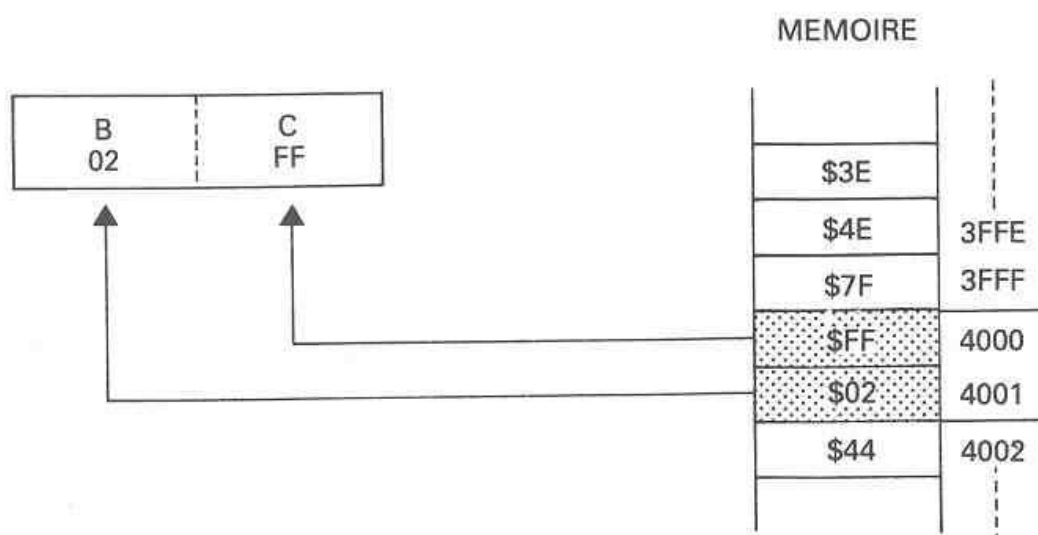


Schéma 2.5

Adressage indirect.

IX et IY permettent un mode particulier de travail : on peut les associer à un déplacement afin de pointer dans une table. Exemple : "LD C,(IX+4)" indique au Z-80 de charger, dans le registre C, le contenu de l'adresse IX+4, c'est-à-dire l'adresse contenue dans IX à laquelle on ajoute 4. C'est ce qu'on appelle l'adressage indexé (schéma 2.6 v. p. 43).

LD C,(IX+4)

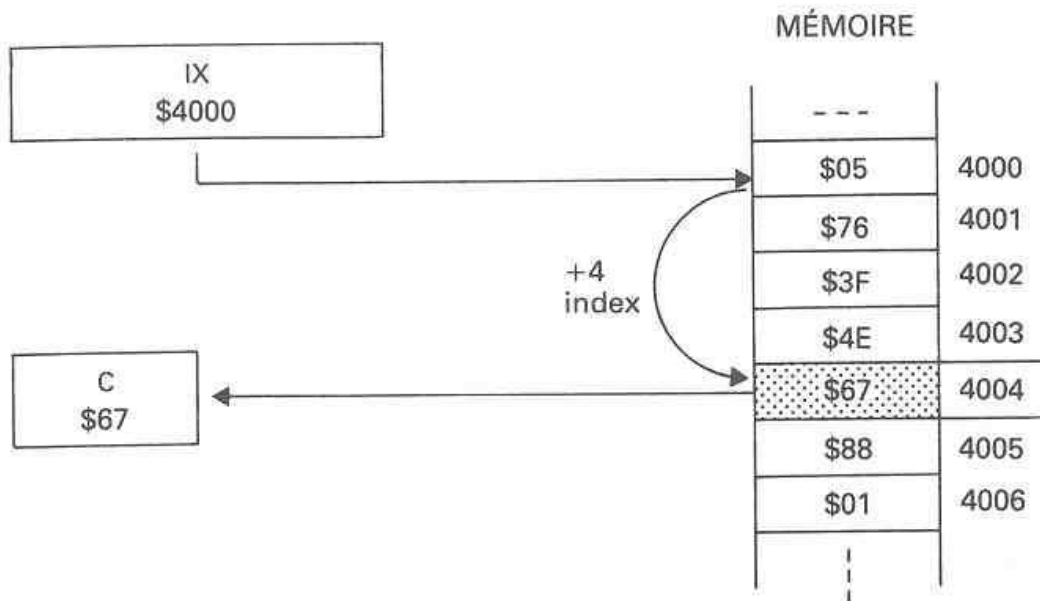


Schéma 2.6

Adressage indexé.

Dans une moindre mesure, ce type de travail peut également être effectué par HL : "LD E, (HL)" permet ainsi de charger dans E le contenu de l'adresse indiquée dans HL. On parle alors d'adressage indirect par registre. (HL), (IX+n) et (IY+n) peuvent ainsi être utilisés avec tous les registres 8 bits communs, et, en guise de bonus, A peut travailler également avec (DE), et avec (BC) (schéma 2.7).

LD C,(HL)

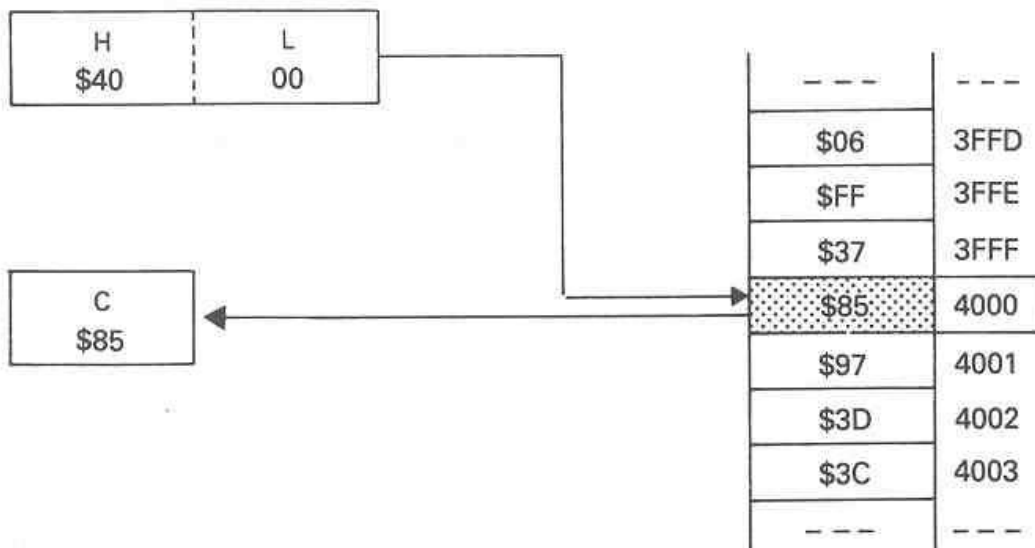


Schéma 2.7

Adressage indirect par registre.

Les chargements en mémoire, qui sont les instructions inverses des chargements de registres, travaillent obligatoirement (sauf trois exceptions : "LD (HL),n", "LD (IX+n),m" et "LD (IY+n),m" qui autorisent le chargement direct d'une donnée à une adresse indiquée par les registres indiqués) avec le contenu d'un registre. Il est par exemple **impossible** d'effectuer un "LD (0),15" pour charger la valeur 15 à l'adresse 0. Les chargements en mémoire sont en nombre beaucoup plus réduits. D'une façon générale, ils s'écrivent "LD représentation d'une adresse, registre" (schéma 2.8).

LD (IX + 3), \$76

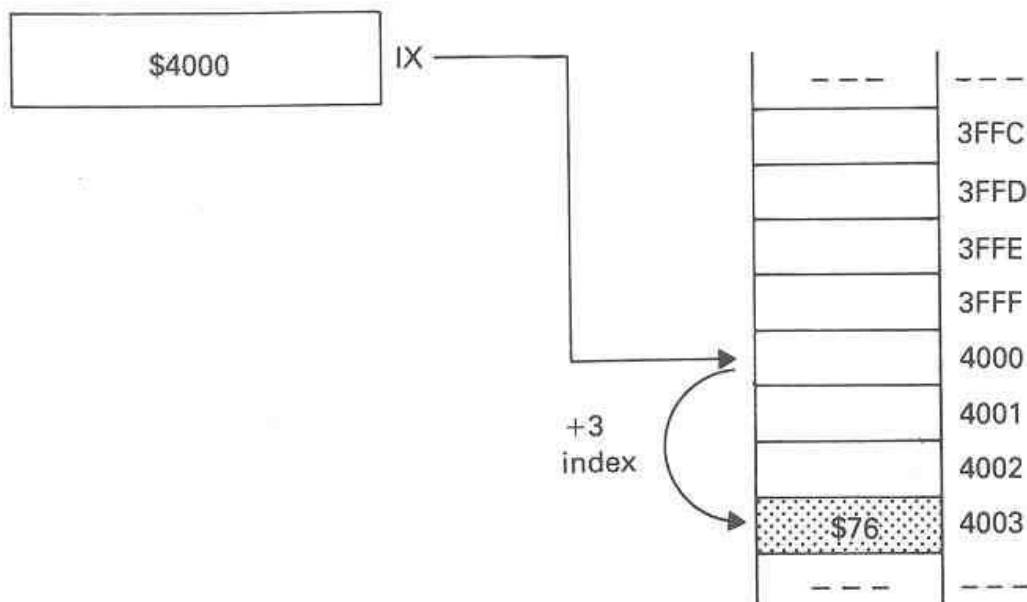


Schéma 2.8

Chargement indexé en mémoire.

On peut effectuer avec le registre A tout ce qui est imaginable à ce niveau : chargement à une adresse indiquée par un registre 16 bits (par exemple "LD (BC), A"), chargement direct à une adresse ("LD (4000),A"), chargement indexé ("LD (IX+n),A"). Les opérations à base du contenu des autres registres 16 bits sont par contre limitées à l'adressage indirect ou indexé. On ne peut charger un registre 8 bits autre que A qu'à une adresse indiquée sous la forme (HL), (IX+n) ou (IY+n).

Enfin, il est possible de sauver à une adresse directement exprimée le contenu des registres 16 bits BC, DE, HL, IX, IY et SP (et, nous l'avons vu ci-dessus, on peut aussi utiliser A pour charger ainsi une valeur 8 bits).

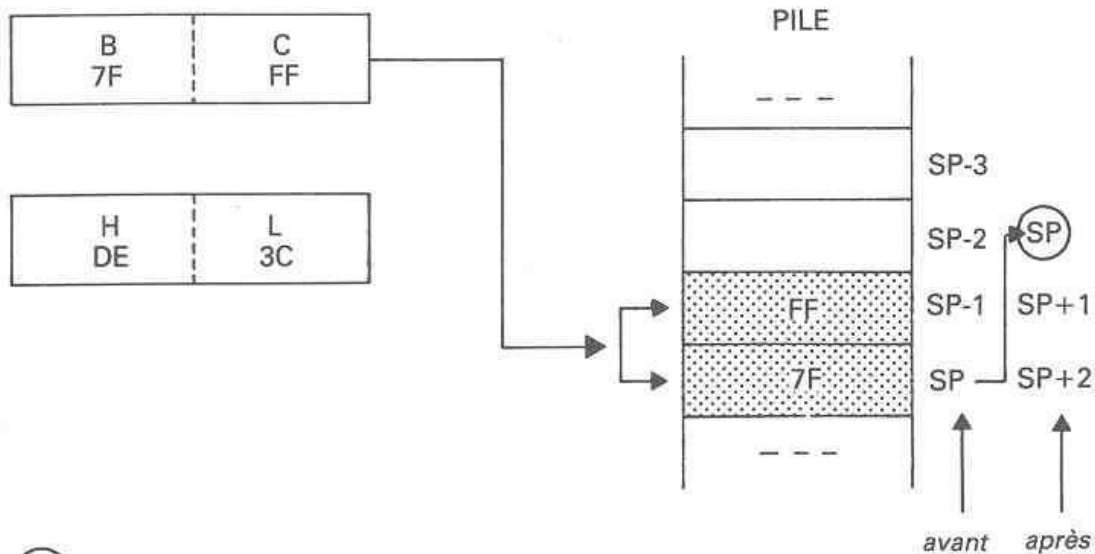
Cela procure une liste non négligeable d'instructions. Dans la pratique, il suffit d'avoir une liste complète des instructions par ordre alphabétique à portée de main pour s'en sortir (voir annexe 1).

La pile et sa gestion

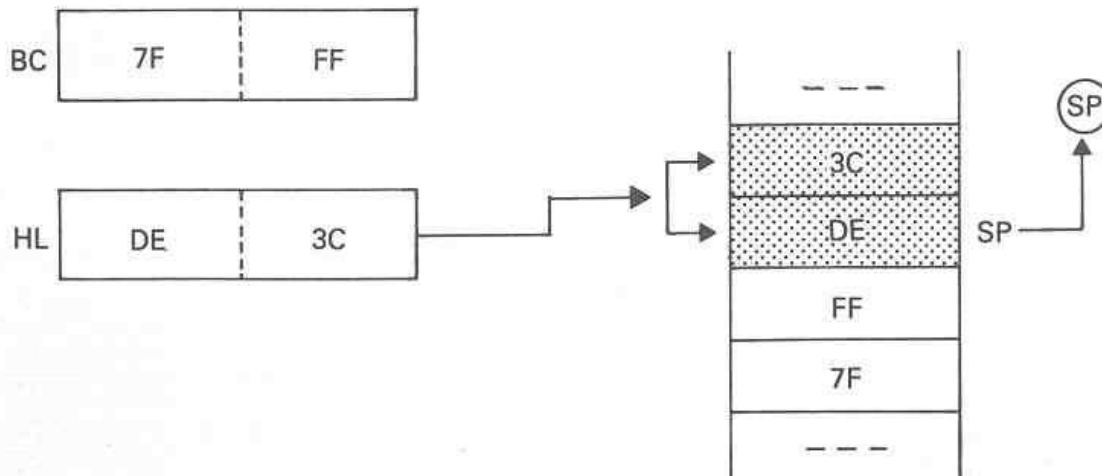
Autre catégorie d'instructions : les instructions de gestion de la pile. Celles-ci sont regroupées sous les dénominations PUSH et POP. PUSH permet de ranger dans la pile les registres 16 bits AF, BC, DE, HL, IX et IY. POP permet de récupérer dans la pile une valeur, en la chargeant dans un de ces registres. Ainsi, la séquence "PUSH BC" suivi de "POP HL" range le contenu de BC dans la pile et le retire immédiatement pour le ranger dans HL. Voilà donc le moyen d'effectuer un simili "LD HL,BC". Cela dit, cette manipulation est peu utilisée car elle est bien plus lente que la combinaison de "LD H,B" et "LD L,C". Par contre, il est possible d'utiliser la pile pour échanger des contenus de registres 16 bits facilement.

Ainsi, on utilise la séquence suivante pour inverser HL et BC (schéma 2.9) :

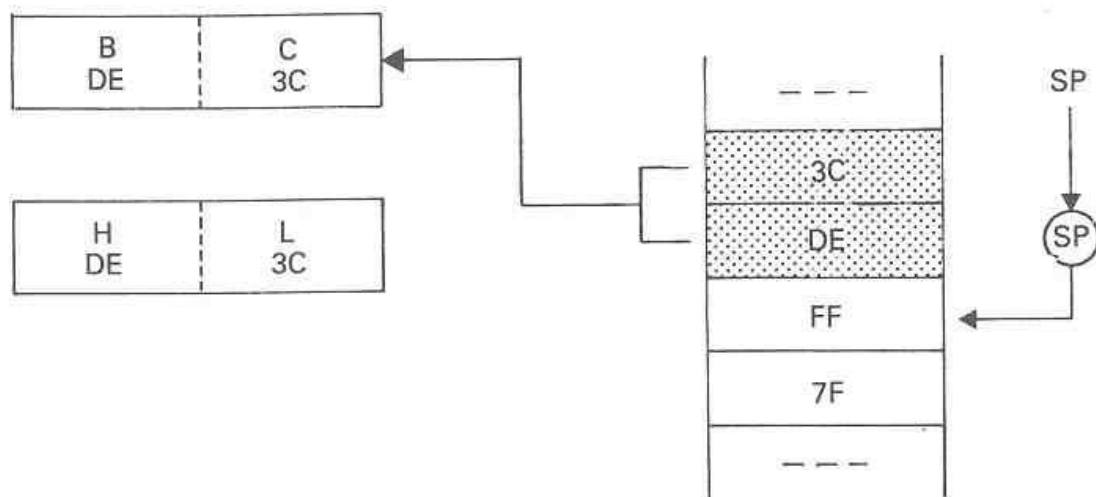
① PUSH BC



② PUSH HL



③ POP BC



④ POP HL

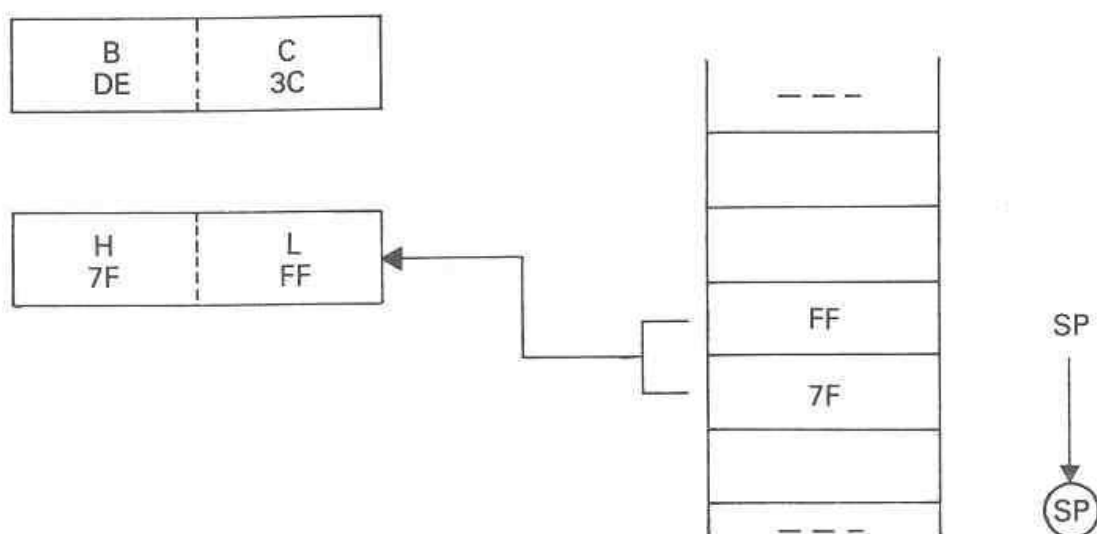


Schéma 2.9

Échange de registres par la pile.

```

PUSH BC
PUSH HL
POP BC
POP HL

```

Enfin, on peut aussi résumer les instructions de gestion du registre SP, le pointeur de pile. Ce registre retient l'emplacement de la pile, plus exactement l'emplacement du haut de la pile. Un PUSH diminue cette valeur de 2, et un POP l'augmente de 2. On peut charger SP avec les valeurs

contenues dans un des registres HL, IX, IY, on peut aussi stocker directement une valeur ("LD SP,adresse") ou indirectement ("LD SP,(adresse)").

Dans le sens inverse, on peut sauver la valeur de SP à une adresse ("LD (adresse),SP"). Il est également possible d'ajouter 1 à SP en utilisant « INC SP », ou d'enlever 1 avec "DEC SP".

Il existe d'autres instructions pouvant utiliser SP, mais elles relèvent du haut professionnalisme !

Les drapeaux/flags

Le Z-80 possède un registre F contenant les drapeaux, nous l'avons vu plus haut. Ces drapeaux sont au nombre de six, utilisant chacun un bit de F (il y a deux bits inutilisés). Le drapeau C, appelé "Carry", indique en général une retenue (lors d'additions), un changement de signe (soustraction) ou un des bits de A après une rotation. On l'utilise, lors des débuts, principalement pour des tests de comparaison.

Ainsi, après une instruction "CP n", le Carry vaut 1 si le contenu du registre A était strictement inférieur à n, et 0 sinon.

Le drapeau N indique, lorsqu'il est mis à 1, que la dernière opération était une soustraction (ce qui permet de savoir si le Carry provient ou non d'une telle opération). Peu de programmes en nécessitent l'emploi, nous conseillons donc vivement d'oublier pour l'instant son existence.

Le drapeau P/V a un double sens. Lors des opérations arithmétiques (addition, soustraction), il indique qu'un débordement a eu lieu. P/V est rarement utilisé dans les programmes, sauf cas vraiment particuliers, comme des calculs en grande précision.

Le drapeau H ne présente aucun intérêt non plus. Par contre, plus intéressants sont les drapeaux Z et S. Ils indiquent en effet respectivement la nullité et le signe de A. Si Z=1, le contenu de A est nul, ou bien (après une comparaison) les nombres comparés sont égaux. Si S=1, le nombre contenu dans A est négatif.

Le débutant (ainsi que la plupart des programmeurs) n'utilisera utilement que C, Z et S. Ce sont en effet les drapeaux qui indiquent l'état du contenu de A : on peut, en les consultant, savoir si A est cohérent, nul, positif ou non.

Les drapeaux ne sont jamais utilisés en tant que tels : ils doivent être consultés et associés à une instruction CALL, RET, JR ou JP. Selon l'association demandée, l'instruction est exécutée si cette association est possible, et ignorée si ce n'est pas le cas.

Les associations sont les suivantes :

- NZ pour savoir si $Z=0$ (A non nul, par exemple) ;
- Z pour savoir si $Z=1$;
- NC pour savoir si $C=0$;
- C pour savoir si $C=1$;
- PO pour savoir si $P/V=1$;
- PE pour savoir si $P/V=0$;
- P pour savoir si A est "P"lus grand que 0, soit positif ($S=0$) ;
- M pour savoir si A est "M"oins grand que 0, soit négatif ($S=1$).

Par exemple, l'instruction "JP P,4000" provoquera un saut en 4000 si le drapeau S est à 0. Elle sera ignorée si S est à 1.

Sauts et branchements conditionnels

Puisque nous en sommes à parler de CALL, JP et JR, autant les passer en revue, d'autant plus que cela terminera notre exposé sur le Z-80. CALL est en LM l'équivalent de GOSUB en Basic : arrivé à "CALL adresse", le programme saute à cette adresse, jusqu'à ce que RET soit rencontré. Ensuite, le programme revient à la suite du CALL. JP provoque la même chose, mais sans retour possible (équivalent de GOTO). Enfin, JR est une version de JP destinée à de petits sauts (dans un endroit pas plus éloigné de l'instruction que par 128 octets). JR est un peu moins rapide que JP, c'est pourquoi nous ne l'utiliserons pas dans nos routines. JR a une seule utilité, qui n'est valable que pour des programmes d'un certain type. En gros, lorsqu'un programme utilise des JP, il ne peut fonctionner que s'il est chargé à une certaine adresse bien précise, celle qui a été choisie lors de l'assemblage. Par contre, si le programme utilise JR, il peut être placé à n'importe quelle adresse. Cette possibilité ne nous intéresse pas ici, car nos routines sont courtes, et nous les placerons toujours à un endroit bien connu. Les débutants n'ont aucun intérêt à utiliser JR. De plus, JR ne peut être associé qu'aux conditions C, NC, Z et NZ. Et JP procure trois facilités intéressantes : il est possible d'utiliser "JP (HL)", "JP (IX)" et "JP (IY)" pour sauter à une adresse calculée. Cela permet de programmer ce que l'on appelle des "vecteurs", dont Amstrad fait un usage important à des fins de compatibilité entre ses différents modèles.

Conclusion

Nous laissons aux lecteurs le soin de consulter des livres spécialisés (par exemple ceux que nous vous recommandons plus haut) afin d'en connaître plus sur les instructions arithmétiques et logiques du Z-80. Il est également intéressant de jeter un œil sur les instructions SET, RES et BIT qui permettent de positionner, baisser ou tester n'importe quel bit de chacun des registres 8 bits, voire (HL) ou (IX+d) et (IY+d).

Toutefois, lorsque nous utiliserons des instructions nouvelles non explorées dans ce chapitre, nous en expliquerons le fonctionnement et l'utilité.

ASSEMBLEUR ET BASIC

Stockage en mémoire

Maintenant que nous connaissons mieux le Z-80, revenons sur terre. Vous ne possédez pas forcément un assembleur, auquel cas les instructions JP ou ADD ne signifient rien. Une seule chose est sûre : votre Amstrad possède un Basic, et fort heureusement ce Basic permet tout de même d'utiliser du langage machine.

Comment se présente un programme en langage machine lorsque nous sommes sous contrôle du Basic ? Rien de plus obscur : une belle suite de nombres apparemment incohérente. Comment le rentrer en mémoire ? Comment l'exécuter ?

Il faut avant tout savoir que POKE permet de modifier le contenu de n'importe quelle case mémoire. Nous l'avons par exemple utilisé lors du remplissage écran. "POKE adresse, valeur 8 bits" est exactement équivalent, en LM, à "LD A,valeur" suivi de "LD (adresse), A". Vous pouvez, sous Basic, préciser l'adresse et la valeur soit en décimal, soit en hexadécimal (adresse ou valeur précédée de '&', par exemple &F23C), soit en binaire (préfixe '&X', par exemple &X11010011).

Cette instruction, associée à READ et DATA, permet de stocker un programme LM en mémoire d'après son listing. Il suffit de taper les nombres correspondant aux instructions dans des lignes DATA, et de les lire l'un après l'autre pour les stocker aux bonnes adresses grâce à READ et POKE.

Cette façon de procéder est la plus simple pour des routines ne risquant pas de "manger" le Basic. Elle n'est pas optimale, on peut faire beaucoup mieux. On peut par exemple stocker directement des routines à l'intérieur même d'une ligne REM Basic. Mais cela nécessite des précautions draconiennes.

Lorsqu'une routine LM est placée en mémoire, il faut en général lui réserver une place pour éviter que le Basic ne vienne la détruire lors de ses travaux. L'instruction MEMORY a ce rôle. "MEMORY adresse-1" indique au Basic qu'il ne devra pas aller travailler au-delà de "adresse". Ainsi, après "MEMORY &3FFF" le Basic ne travaillera jamais au-delà de \$4000 (rappelons que le symbole '\$' signifie hexadécimal, il est noté '&' en Basic, et souvent '#' en assembleur). De cette façon, l'espace libre est disponible pour le LM.

Exécution et paramètres

Enfin, maintenant que nous pouvons stocker un programme LM en mémoire et le protéger du Basic, il faut savoir l'exécuter. Pour cela, le Basic Amstrad possède CALL. "CALL adresse" exécute le programme LM situé à l'adresse indiquée, et revient au Basic lorsque RET est rencontré.

Ces possibilités, tout ordinateur digne de ce nom les offre. Mais l'Amstrad va plus loin : CALL permet également le passage de paramètres.

Qu'entend-on par passage de paramètres ? Tout programme en LM fonctionne avec certaines valeurs. Il y a les valeurs de départ, certes, celles qui doivent être placées dans les registres au début de la routine. Mais il se peut également que la routine puisse travailler à partir de valeurs inconnues. Par exemple, la routine SIN travaille à partir d'un nombre. Ce nombre est un paramètre. CALL permet de passer ainsi des paramètres à une routine, à charge pour celle-ci de les utiliser ou non.

Ainsi, "CALL &4000,14,15" appellera la routine située en \$4000, en lui passant les valeurs 14 et 15. Cela est extrêmement puissant, d'autant plus qu'il est possible d'envoyer une adresse de variable. "CALL &4000,A,B,@C" fournit à la routine le contenu des variables A et B, ainsi que l'adresse où se trouve la variable C. Cela permet par exemple de ranger directement une valeur, par LM, dans une variable avant de revenir au Basic.

Le Basic utilise, pour transmettre ces paramètres, une table provisoire dont l'adresse sera contenue dans IX. Cette table est constituée de données 16 bits exclusivement : on ne peut donc passer que des paramètres de **type entier**.

Le début de la routine appelée par l'instruction CALL peut utiliser ces valeurs 16 bits, rangées dans l'ordre inverse de leur apparition. Ainsi, si nous avons "CALL &4000,Y,Z,@R", nous obtenons la table suivante :

Adresse indiquée par	Contenu
IX+4	poids faible valeur de Y (8 bits de droite)
IX+5	poids fort valeur de Y (8 bits de gauche)
IX+2	poids faible valeur de Z
IX+3	poids fort valeur de Z
IX+0	poids faible adresse de localisation de R
IX+1	poids fort adresse de localisation de R

Cette table est utilisable, rappelons-le, dès le début de la routine, puisque le Basic s'occupe personnellement de transmettre IX.

Nous savons donc déjà comment utiliser des valeurs transmises. Si nous voulons par exemple additionner Y et Z par langage machine, il suffit d'avoir la séquence LM suivante :


```
LD    L,(IX+4)
LD    H,(IX+5)
LD    E,(IX+2)
LD    D,(IX+3)
ADD   HL,DE
```

L'addition aura bel et bien été exécutée. Le problème qui reste est le suivant : comment transmettre le résultat à R ?

Nous avons déjà une partie de la solution, puisque nous avons vu que l'on pouvait passer à notre routine l'adresse de la variable R. Grâce à cette adresse, nous pouvons en effet modifier la valeur de la variable. Pour cela, il nous faut connaître la structure des variables entières, c'est-à-dire ce que nous allons trouver à l'adresse ainsi passée en paramètre.

La meilleure façon de connaître cette structure, c'est sans aucun doute d'utiliser le Basic pour afficher le contenu de la mémoire. Tapez le programme suivant :

```
10 CLEAR:DEFINT A-Z
20 A=&1234
30 FOR I= @A TO @A+1
40 PRINT HEX$(PEEK(I),2);
50 NEXT I
```

A l'exécution, vous obtenez le résultat 3412. Cela nous révèle ce que nous voulions savoir : l'adresse @A indique l'adresse du contenu de A.

Pour modifier la valeur de R dans la routine LM ci-dessus, il suffit de rajouter les instructions suivantes :

```
EX DE,HL    : pour passer le résultat dans DE et disposer de HL ;
LD L,(IX+0) : pour stocker dans HL l'adresse de la variable ;
LD H,(IX+1) : transmise comme dernier paramètre ;
LD (HL),E   : pour stocker le poids faible du résultat ;
INC HL      : dans la variable ;
LD (HL),D   : et son poids fort.
```

Le travail de la routine est alors terminé. Le programme 2.1 résume cette routine et montre son utilisation sous Basic, avec les passages de paramètres.

```
10 ;
20 ;addition 16 bits avec retour du resultat
30 ;dans une variable Basic (prog 2.1)
40 ;
4000 50      ORG #4000
60 ;
4000 DD6E04 70 ADDITI: LD    L,(IX+4)
4003 DD6605 80      LD    H,(IX+5)                ;premier nombre dans HL
```



```

4006 DD5E02      90      LD  E,(IX+2)
4009 DD5603     100      LD  D,(IX+3)      ;deuxieme nombre dans DE
      110 ;
      120 ;ici, l'addition proprement dite
      130 ;
400C 19         140      ADD HL,DE
      150 ;
      160 ;on range le resultat dans la variable
      170 ;
400D EB         180      EX  DE,HL      ;resultat passe dans DE
400E DD6E00     190      LD  L,(IX+0)
4011 DD6601     200      LD  H,(IX+1)      ;adresse du contenu dans HL
4014 73         210      LD  (HL),E
4015 23         220      INC HL
4016 72         230      LD  (HL),D      ;le resultat devient le contenu
4017 C9         240      RET

```

Pass 2 errors: 00

```

1  '*****
2  '** Programme 2.1 **
3  '*****
4  '
10 '
20 'addition 16 bits LM avec modif de variable
30 '
40 MEMORY &3FFF
50 DEFINT a-z
60 ad=&4000
70 READ c:IF c=-1 THEN 140
80 POKE ad,c:ad=ad+1:GOTO 70
90 DATA 221,110,4,221,102,5,221,94,2,221,86,3,25
    ,235,221,110,0,221,102,1,115,35,114,201,-1
140 INPUT "premier nombre ";A
150 INPUT "second nombre ";B
160 C=0:CALL &4000,A,B,@C
170 PRINT "resultat :"
180 PRINT A;"+";B;"=";C
190 END

```

Organisation des routines

Pour souple que soit cette possibilité d'interfaçage entre Basic et LM, elle n'est toutefois pas exempte de défauts. En effet, le temps perdu lors des passages de paramètres est énorme. Les opérations de chargement à base

de IX ou IY sont plus lentes que leurs équivalents avec HL, et le Basic a, de plus, un gros travail à faire lorsqu'il met en place les paramètres dans la table pointée par IX. Vous pouvez constater le peu d'intérêt de notre routine d'addition en utilisant les programmes 2.2 et 2.3.

```

10 '*****
20 '** Programme 2.2 **
30 '*****
40 '
50 'addition chronometree 16 bits LM
60 '
70 MEMORY &3FFF
80 DEFINT a-z
90 ad=&4000
100 READ c:IF c=-1 THEN 130
110 POKE ad,c:ad=ad+1:GOTO 100
120 DATA 221,110,4,221,102,5,221,94,2,221,86,3,2
    5,235,221,110,0,221,102,1,115,35,114,201,-1
130 A=300:B=312
140 z!=TIME
150 FOR I=1 TO 100
160 C=0:CALL &4000,A,B,@C
170 PRINT "resultat :"
180 PRINT A;"+";B;"=";C
190 NEXT
200 PRINT "Temps : "TIME-z!
210 END

```

```

10 '*****
20 '** Programme 2.3 **
30 '*****
40 '
50 'addition chronometree 16 bits basic
60 '
70 DEFINT a-z
80 a=300:b=312
90 z!=TIME
100 FOR I=1 TO 100
110 C=A+B
120 PRINT "resultat :"
130 PRINT A;"+";B;"=";C
140 NEXT
150 PRINT "Temps : "TIME-z!
160 END

```

Alors que dans un cas l'addition est effectuée par LM et dans l'autre en Basic, le temps d'exécution des deux programmes est quasiment le même. Mais le passage de paramètres par CALL a un intérêt : il n'oblige pas à faire de routine en fonction du nombre de paramètres, ni à gérer ceux-ci. De plus, les routines utilisant ce principe peuvent généralement être rendues très indépendantes du Basic en adoptant l'organisation suivante :

□ Appel Basic :

- les paramètres sont rangés à des adresses mémoire bien précises ou dans des registres ;
- CALL routine (voir "Appel LM" ci-dessous) ;
- les résultats sont rangés des variables passées en paramètres directement à partir des registres ou des adresses mémoire mis à jour par la routine ;
- retour au Basic.

□ Appel LM :

- la routine s'effectue et range les résultats éventuels dans des registres ou à des adresses précises.

De cette façon, la routine peut être utilisée de façon interne en LM et non pas uniquement à partir du Basic. Il est à noter que cette organisation a été retenue pour la quasi-totalité des routines du logiciel interne de l'Amstrad (d'où la possibilité de récupérer ces routines pour des logiciels d'application comme le Basic).

Pour illustrer le passage de paramètres, vous pouvez également utiliser le programme 2.4, qui effectue une soustraction 16 bits.

```

10 ;
20 ;soustraction 16 bits avec retour du resultat
30 ;dans une variable Basic (prog 2.4)
40 ;
4000 50      ORG #4000
60 ;
4000 DD6E04 70 SOUSTR: LD  L,(IX+4)
4003 DD6605 80      LD  H,(IX+5)      ;premier nombre dans HL
4006 DD5E02 90      LD  E,(IX+2)
4009 DD5603 100     LD  D,(IX+3)      ;deuxieme nombre dans DE
110 ;
120 ;ici, la soustraction proprement dite
130 ;
400C A7      140     AND  A            ;carry->0 pour SBC
400D ED52    150     SBC  HL,DE       ;car il n'y a pas SUB HL,DE !
160 ;

```

```

170 ;on range le resultat dans la variable
180 ;
400F EB 190 EX DE,HL ;resultat passe dans DE
4010 DD6E00 200 LD L,(IX+0)
4013 DD6601 210 LD H,(IX+1) ;adresse du contenu dans HL
4016 73 220 LD (HL),E
4017 23 230 INC HL
4018 72 240 LD (HL),D ;le resultat devient le
; contenu
4019 C9 250 RET

```

Pass 2 errors: 00

```

1 *****
2 ** Programme 2.4 **
3 *****
4 '
10 '
20 'soustraction 16 bits LM avec modif de variab
le
30 '
40 MEMORY &3FFF
50 DEFINT a-z
60 ad=&4000
70 READ c:IF c=-1 THEN 140
80 POKE ad,c:ad=ad+1:GOTO 70
90 DATA 221,110,4,221,102,5,221,94,2,221,86,3,16
7,237,82,235,221,110,0,221,102,1,115,35,114,20
1,-1
140 INPUT "premier nombre ";A
150 INPUT "second nombre ";B
160 C=0:CALL &4000,A,B,@C
170 PRINT "resultat : "
180 PRINT A;"-";B;"=";C
190 END

```

Il existe également sur l'Amstrad une autre possibilité d'interfaçage des routines LM, il s'agit du RSX. Ce RSX (Resident System eXtension, soit Extension de système résidente) permet de donner un nom aux routines et de les appeler par ce nom. Il permet également de passer des paramètres non entiers. Mais il a d'autres inconvénients. Il rajoute une couche de plus à l'organisation que nous venons d'évoquer, et ralentit encore l'exécution. Nous n'utiliserons donc pas le RSX.

Maintenant que nous savons comment utiliser des routines sous Basic, il nous reste un point important à éclaircir : où allons-nous placer ces routines ?

De toute évidence, nos routines ne devront empiéter ni sur le Basic (encore que ce soit acceptable si celui-ci n'est pas utilisé), ni surtout sur le système d'exploitation. En effet, la machine proprement dite utilise grandement le système d'exploitation et sa zone de mémoire vive.

Il est possible de grignoter une partie de cette mémoire vive, notamment celle qui contient des vecteurs destinés à l'appel des routines de graphisme, de la gestion cassette, etc. Mais dans ce cas, on perd l'accès facile à ces routines (situées en ROM "sous" la RAM, il faut pour les appeler se livrer à une manœuvre des plus délicates). De plus, la place ainsi récupérée n'est pas réellement importante.

Pour faire le point, éteignez votre ordinateur et rallumez-le. Juste après la mise sous tension, tapez `PRINT HEX$(HIMEM)`. Cela vous donnera la plus haute adresse disponible pour le Basic, c'est-à-dire en pratique la plus haute adresse mémoire utilisable sans problème. Au-delà, il y a de grandes chances d'écraser des tables système, une pile ou pire encore.

Sur un Amstrad CPC 464 équipé d'un lecteur de disquettes, la plus haute adresse ainsi accessible est `$A67B`. Nous pouvons considérer, la pratique aidant, que la première adresse rôde aux alentours de `$200`. On peut descendre plus, mais il devient alors difficile de gérer quoi que ce soit sous Basic. Un rapide calcul vous informe du nombre d'octets utilisables :

```
PRINT &A67B-&200+65536
```

Nous devons ajouter 65536 ici pour obtenir un nombre positif : au-delà de `$7FFF`, les nombres hexadécimaux deviennent négatifs, et il faut ajouter 65536 pour obtenir la valeur décimale positive qu'ils représentent réellement. Nous obtenons ainsi 42107, ce qui donne presque 42 Ko disponibles pour les programmes en langage machine.

Pratiquement, dans cet ouvrage, nous ne placerons aucune routine en dessous de `$3000`. En effet, il faut laisser suffisamment de place pour un petit programme Basic (en l'occurrence, cela laisse environ 12 Ko), faute de quoi l'utilisation des routines devient impossible !

Pour éclaircir les choses, vous trouverez en annexe 4 une carte de la mémoire de l'Amstrad.

ROUTINES GRAPHIQUES | **3** DU SYSTÈME

SYSTÈME D'EXPLOITATION

Organisation

Comme nous l'avons laissé entendre plusieurs fois, l'Amstrad se distingue des autres micro-ordinateurs par la conception de son logiciel interne, beaucoup plus évoluée et proche des systèmes d'exploitation d'ordinateurs professionnels comme l'IBM-PC.

Le principe de base de ce logiciel interne est en effet la programmation dans le système d'exploitation de tout ce qui permet de gérer les ressources de l'ordinateur, et non pas seulement de quelques routines. C'est ainsi que les 16 Ko de mémoire morte contenant le système d'exploitation (les autres 16 Ko constituent l'interpréteur Basic) regroupent toutes les routines de gestion des interruptions, des graphismes ou des mémoires de masse (cassette, disquette...).

Ces routines sont situées en mémoire morte : leur fonctionnement ne peut donc être modifié simplement, et par conséquent le Basic serait figé s'il les appelait par leur adresse précise. De plus, à la moindre amélioration matérielle, la compatibilité avec les programmes existants serait perdue. La plupart des constructeurs font leur lot de ce problème, ce qui explique pourquoi des appareils comme l'Apple II ou le TO7, coincés par la nécessité de la compatibilité, n'ont guère évolué depuis leur création.

Le logiciel interne de l'Amstrad, en revanche, est conçu pour évoluer. En effet, il utilise un vecteur pour chaque routine, afin de préserver la compatibilité sans bloquer l'évolutivité du système. Pour mettre en évidence l'avantage de ce principe, il suffit de savoir qu'il est parfaitement possible d'installer sur un Amstrad une extension dotée de mémoire morte qui mettrait en place des graphismes du type 512X256 en 256 couleurs, tout en gardant la compatibilité avec les logiciels existants !

Vecteurs

Qu'est-ce qu'un vecteur, et quel est donc son principe de fonctionnement ? Un vecteur est soit une adresse, soit une instruction de saut à une adresse. Il s'agit d'une information en mémoire qui indique, lorsqu'on la consulte, où trouver l'information que l'on cherche. Ces vecteurs sont situés en mémoire vive. Lors de la mise en route de l'ordinateur, ils sont installés, d'après les indications en ROM, à leur adresse et pointent sur le bon emplacement, celui de l'Amstrad standard. Or, rien n'interdit de modifier ces vecteurs (puisque'ils sont en mémoire vive) afin de substituer une routine à celle visée par le vecteur. De cette façon, on peut par exemple modifier radicalement le fonctionnement de la routine traçant une ligne entre deux points. Il suffit de reprogrammer la routine, et de modifier le vecteur de DRAW pour qu'il pointe sur la nouvelle (*schéma 3.1 v. p. 59*).

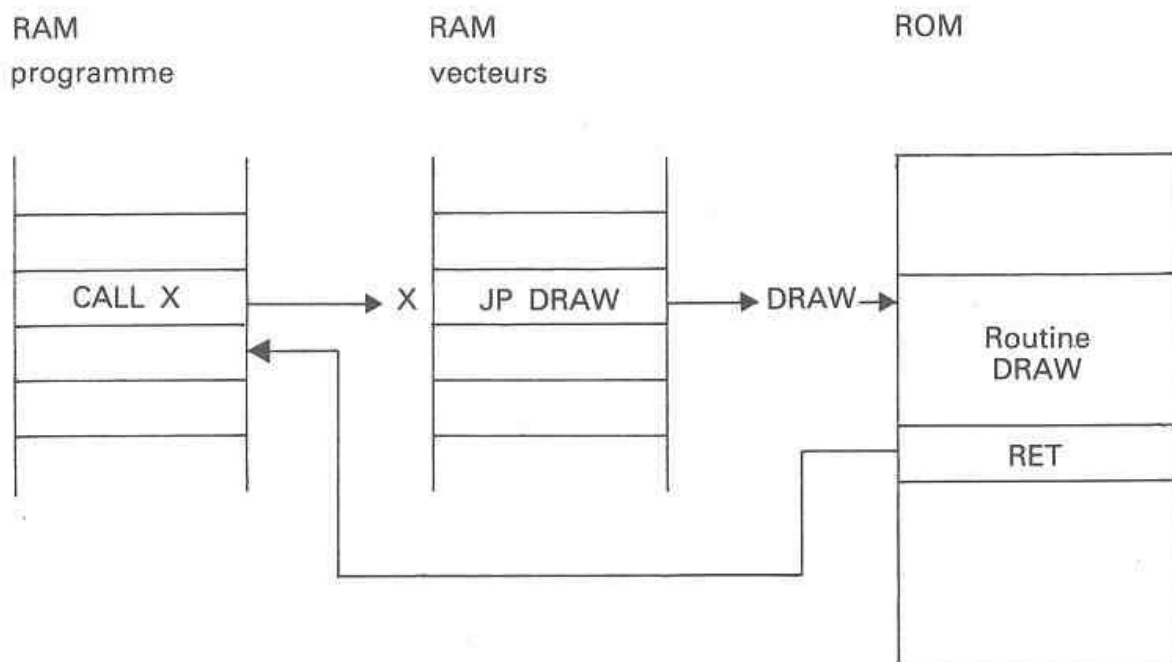


Schéma 3.1

Un vecteur.

Les avantages des vecteurs sont multiples : tout d'abord, ils garantissent la compatibilité d'un matériel à l'autre pour les programmes qui les ont utilisés. Même si le contenu de la ROM change (et donc le contenu des vecteurs), les vecteurs ne changeront pas de place. Un appel à un de ces vecteurs aura le même effet quel que soit le modèle d'Amstrad utilisé. Même si, demain, Amstrad sort un nouvel ordinateur doté de 256 Ko et d'un processeur graphique différent du 6845, rien n'interdit la compatibilité si Amstrad respecte les vecteurs pour adresser les nouvelles routines graphiques. Aucun autre micro-ordinateur ne peut prétendre à la même performance.

Ensuite, il est facile de changer les routines soi-même, en détournant ces vecteurs. On peut par exemple ainsi modifier totalement les routines de chargement et de sauvegarde cassette, sans affecter le moins du monde le fonctionnement du Basic ou des programmes utilisant ces routines par appel de vecteurs.

Enfin, ultime avantage, ils forment un bloc compact regroupant la totalité des fonctionnalités de l'ordinateur. Le programmeur LM se retrouve avec une sorte de langage machine évolué, lui permettant de programmer une grande partie des tâches aussi simplement qu'en Basic.

Le seul défaut des vecteurs est le ralentissement qu'ils provoquent lors de l'exécution des routines système et la place qu'ils occupent en mémoire vive. Mais les avantages procurés en valent largement la peine.

Par contre, mais cela ne vient pas de l'utilisation des vecteurs, on peut reprocher à l'Amstrad une programmation quelque peu fantaisiste de certaines routines système. Ainsi, la quasi-totalité des routines graphiques est d'une lenteur exaspérante (toutes proportions gardées) et modifie le

contenu des registres. Il appartient à l'utilisateur de sauver les registres voulus avant de les utiliser, ce qui est souvent pénible et ralentit encore l'exécution.

Toutefois, l'utilisation du Graphic Manager (terme regroupant les routines graphiques du système accessibles par un vecteur) n'est pas dénuée d'intérêt, nous allons donc nous y pencher.

Outre les routines système, Amstrad a eu l'excellente idée de réserver un bon nombre de vecteurs pour les routines de calcul du Basic. En effet, le système d'exploitation proprement dit ne contient aucune routine de calcul en virgule flottante (nombre dits "réels"), et si l'on ne tient pas compte du Basic, les programmes d'application doivent donc fournir les leurs si besoin est. Sur l'Amstrad, toutes les routines utiles (fonctions logarithmes, tangentes, y compris les routines de conversion "format binaire \leftrightarrow format réel") possèdent un vecteur qui permet de les utiliser comme des instructions du processeur. Il y a certaines restrictions, et surtout beaucoup de règles strictes à respecter lors de l'utilisation de ces routines.

Les vecteurs de l'Amstrad sont situés en mémoire vive, de \$BB00 à \$BE00. Ceux qui nous intéressent ici sont les vecteurs des routines graphiques qui sont regroupés de \$BBBA à \$BBFC (*voir annexe 2*).

TRACÉ DE CERCLES

Méthode de tracé

Pour mettre en application les appels système, nous allons réaliser une routine de tracé de cercles. Cette instruction manque au Basic Amstrad, il faut donc l'implémenter.

Le tracé de cercle peut être extrêmement simple. La preuve la plus évidente est le programme Basic suivant :

```
10 MODE 2
20 DEG
30 FOR angle=0 TO 360
40 PLOT COS(angle)*100+319,SIN(angle)*100+199,1
50 NEXT angle
```

L'exécution de ce programme permet d'obtenir un cercle de 100 points de large centré au milieu de l'écran. Le problème de ce programme est sa lenteur exaspérante. Il est cependant facile de l'améliorer, même en Basic.

En effet, bien qu'un cercle soit défini par l'ensemble de ses points sur 360 degrés, il est tout à fait possible de n'effectuer les calculs de sinus et cosinus que sur 45 degrés, et de tracer tous les points associés par

symétrie. En l'occurrence, si X et Y sont les coordonnées correspondant à un angle donné, compris entre 0 et 45, les points tracés seront les suivants :

$$\begin{array}{l} (X,Y) \ (X,-Y) \ (-X,Y) \ (-X,-Y) \\ (Y,X) \ (Y,-X) \ (-Y,X) \ (-Y,-X) \end{array}$$

Cela suppose que le centre du cercle est placé en $(0,0)$ (schéma 3.2).

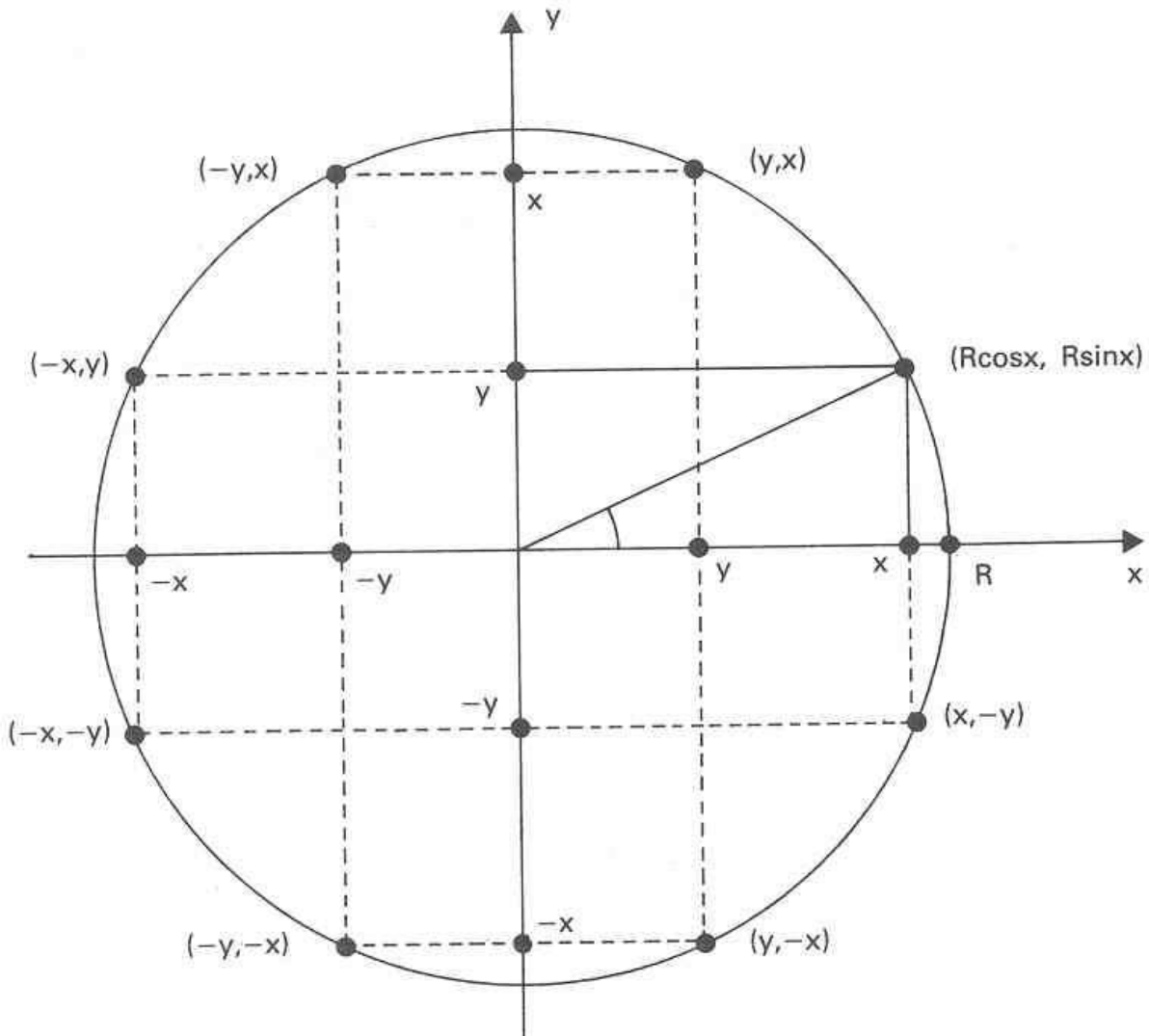


Schéma 3.2

Les points d'un cercle.

Grâce à cette astuce, nous pouvons donc diviser le nombre de calculs par un facteur de 8. De plus, si nous stockons les valeurs des sinus et cosinus dans un tableau, il nous suffira de rappeler ces valeurs pour tracer tous les cercles voulus, sans aucun calcul de sinus ou de cosinus.

Pourquoi se compliquer ainsi la vie ? La raison en est simple : que ce soit en LM ou non, le calcul d'un sinus nécessite une arithmétique en virgule flottante (des nombres réels), donc complexe, et surtout lente.

Parlons encore de l'arithmétique flottante, justement. Qu'il s'agisse d'un jeu d'aventure ou d'action, ce type de calculs en LM doit généralement être évité, ne serait-ce qu'à cause de leur lenteur. Il résulte aussi généralement de ce type d'arithmétique un codage encombrant des nombres (5 octets sur l'Amstrad), et un manque de souplesse pesant (impossible de les gérer par des opérations sur les registres).

Mais ce n'est pas toujours simple. Notamment, comment transformer les valeurs des sinus et des cosinus, qui sont comprises entre -1 et 1 et toutes à virgule, en nombres compris entre 0 et 255 et entiers ?

L'opération est en fait bien plus simple qu'il n'y paraît. En effet, puisque le sinus est compris entre -1 et 1 , il suffit de le multiplier par 256 pour obtenir une valeur comprise entre -256 et 256 ! Ce n'est pas tout, bien sûr. Si nous voulons pouvoir utiliser le sinus sous cette forme, il faudra, au moment des calculs, se souvenir qu'il doit être divisé par 256 . En l'occurrence, la division par 256 est aisée en langage machine.

Résumons-nous. Nous avons trouvé le moyen de tracer huit points d'un coup, et de mémoriser les sinus et cosinus sous la forme de nombres entiers. Nous voici déjà beaucoup plus proches du LM. Le programme 3.1 en Basic le prouve. Le tracé des cercles y est très rapide pour du Basic. Il est en tout cas plus satisfaisant que le programme à base de nombres réels vu plus haut.

```

10 '*****
20 '** Programme 3.1 **
30 '*****
40 '
50 DEFINT a-z
60 DIM si(45),co(45)
70 DEG:FOR i=0 TO 45:co(i)=COS(i)*256:si(i)=SIN(
    i)*256
80 si(i)=si(i)+(si(i)=256):co(i)=co(i)+(co(i)=25
    6)
90 NEXT i
100 INPUT "Rayon ";s
110 MODE 2
120 ORIGIN 319,199
130 FOR r=10 TO 200 STEP s
140 FOR i=0 TO 45
150 x=co(i)*r/256:y=si(i)*r/256
160 PLOT x,y,1
170 PLOT -x,-y,1
180 PLOT x,-y,1
190 PLOT -x,y,1
200 PLOT y,x
210 PLOT -y,-x

```



```

220 PLOT -Y,X
230 PLOT Y,-X
240 NEXT
250 NEXT
260 GOTO 100

```

La boucle des lignes 130 et 250 trace plusieurs cercles concentriques. Vous pouvez à cette occasion constater l'avantage de vitesse procuré par la disparition des calculs de sinus et cosinus. En effet, le tracé de cercle y est effectué grâce aux nombres entiers stockés dans les tableaux SI et CO. Vous remarquerez les calculs de la ligne 150, qui permettent d'obtenir les coordonnées du point de base associé à un angle, en fonction du rayon R. Ce calcul utilise les valeurs de SI et CO, en les divisant par 256. Mais X et Y sont bel et bien des valeurs entières, comme en témoigne la ligne 50 (elle indique que toutes les variables, sauf sur demande, seront entières). L'intérêt de la méthode, outre le gain appréciable de vitesse apporté, réside dans la précision du tracé, tout aussi belle que si des nombres réels avaient été utilisés.

Transposition en assembleur

Il ne reste plus qu'à transposer la méthode en LM. Pour ce faire, il nous faut considérer deux possibilités. Si vous tracez un cercle de rayon 200, vous constatez que les points, très proches, ne le sont toutefois pas assez pour donner un cercle propre. L'idéal serait de relier ces points par des droites. C'était simple avec notre petit programme exemple, où tous les points étaient dessinés l'un après l'autre dans l'ordre. Mais avec notre méthode des symétries, deux points contigus du cercle sont séparés par sept tracés de points !

Il nous faudra donc revoir quelque peu la méthode de tracé pour avoir des cercles propres. La méthode de calcul reste toutefois valable. Nous allons avant tout réaliser la première méthode de tracé (par points), car elle est plus simple et très rapide.

Multiplication en assembleur

Si vous avez un peu l'habitude du langage machine, la ligne 150 du programme vous a immédiatement sauté aux yeux à cause du signe "*" qu'elle contient. La multiplication est la bête noire de ceux qui cherchent à optimiser la vitesse d'exécution des programmes. Supposons que nous voulions multiplier le registre DE par A, et obtenir le résultat dans HL. Cela a déjà une signification : HL se singularise par ses instructions étendues,

c'est pour cela que nous y placerons le résultat. La première idée, puisque l'on multiplie un entier par un entier, est d'additionner A fois le nombre DE dans HL.

Nous obtenons très facilement la routine suivante :

```
MULT: LD  B,A    : stocke compteur dans B pour DJNZ ;
      LD  HL,0    : mise à zéro du résultat ;
```

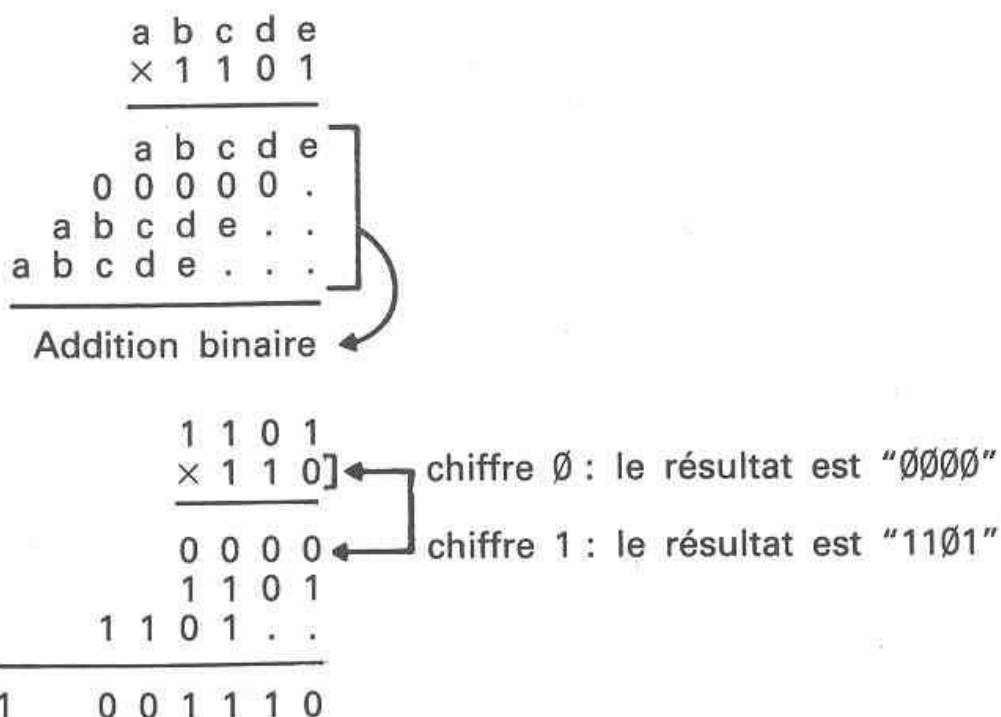
```
ADDI: ADD HL,DE  : additionne DE au résultat ;
      DJNZ ADDI : B fois ;
```

Malheureusement, elle a un énorme défaut. Si le multiplicateur A vaut 3 ou 6, tout va bien. Mais si c'est 200, nous allons effectuer 200 additions pour une simple multiplication ! Pour aller plus loin, il faut se pencher sur le fonctionnement d'une multiplication.

Tout serait simple si nous étions en base dix : nous savons faire une multiplication chiffre par chiffre, en multipliant implicitement le résultat par 1 pour le chiffre des unités, par 10 pour celui des dizaines, et ainsi de suite.

Le fait de savoir cela nous fait déjà progresser d'un grand pas. En effet, la multiplication travaille exactement de la même façon (heureusement) en base 2. On multiplie chiffre par chiffre, en multipliant le résultat de l'unité par 1, puis celui de second chiffre par 2, puis par 4, 8 et ainsi de suite.

C'est même plus simple qu'il n'y paraît : la multiplication d'un nombre par un chiffre, en base deux, n'a que deux résultats : 0 si le chiffre est 0, le nombre lui-même si le chiffre est 1 (*schéma 3.3*).



Voyons comment multiplier le nombre "abcde" par "xyz", en supposant que "abcde" et "xyz" sont binaires (exprimés en base 2, donc avec des 1 et des 0). Tout d'abord, nous devons regarder si z vaut 1. Si oui, nous retenons le résultat "abcde", sinon 0. De la même façon pour Y : si y vaut 1, c'est "abcde0" que nous ajouterons au résultat précédent. Et enfin, pour le cas où x vaut 1, nous ajouterons "abcde00". La somme doit nous donner le bon résultat.

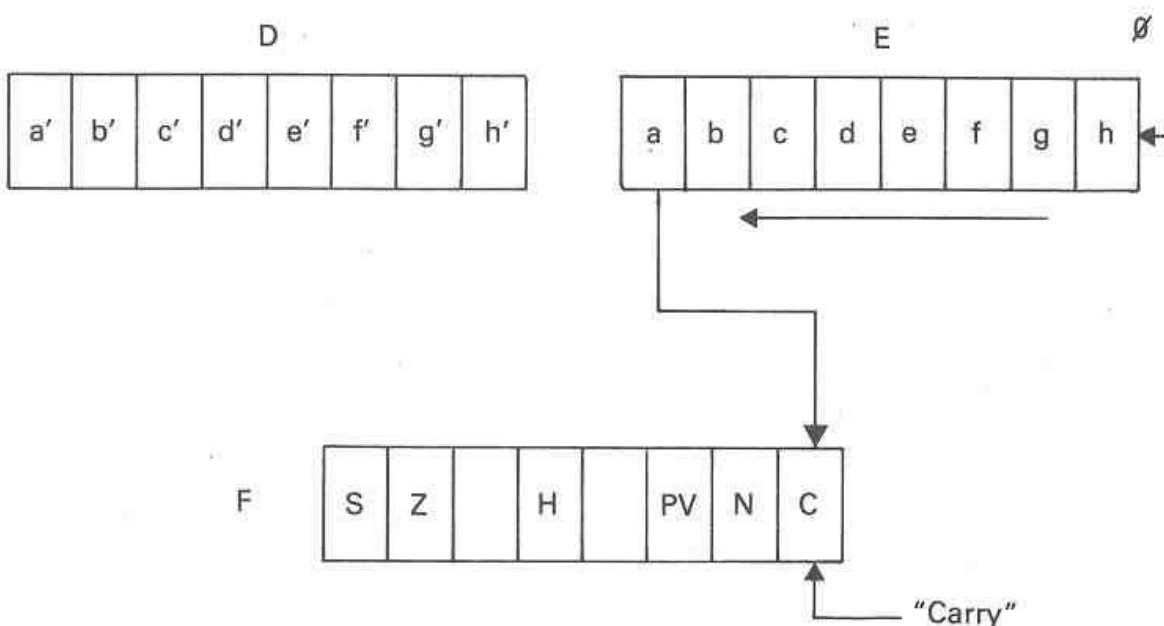
Maintenant, regardons de nouveau nos registres. DE contient l'équivalent de "abcde", mais avec 16 chiffres, et A celui de "xyz" sur 8 chiffres. Nous pouvons utiliser HL pour stocker au fur et à mesure les résultats intermédiaires.

Le reste est simple. En effet, le Z-80 dispose de nombreuses fonctions de rotation et décalage permettant de modifier les contenus des registres. En l'occurrence, il est possible de décaler DE vers la gauche par la suite d'instructions suivantes :

```
SLA E
RL D
```

La première instruction prend le premier bit de E (le bit 7) et le range dans le Carry (c'est le drapeau C du registre F), puis décale les sept autres bits vers la gauche (le bit 6 vient en 7, le bit 5 en 6, etc.) et met un 0 au bit 0. RL D, pour sa part, effectue presque la même chose (décalage à gauche du registre D) mais, au lieu de mettre un zéro au bit 0, il y place le contenu du Carry. Nous avons donc ainsi décalé tout DE vers la gauche (*schéma 3.4*).

① SLA E



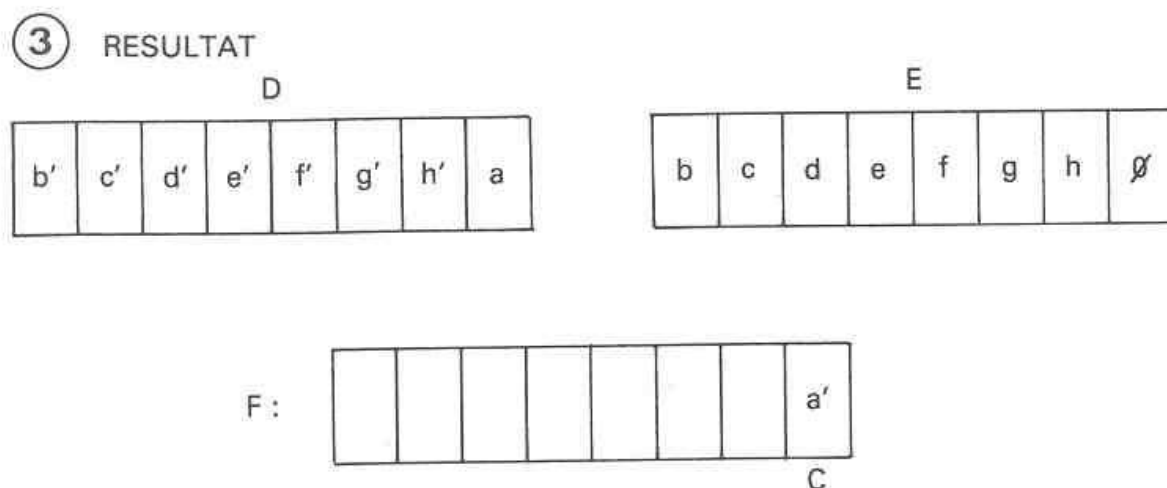
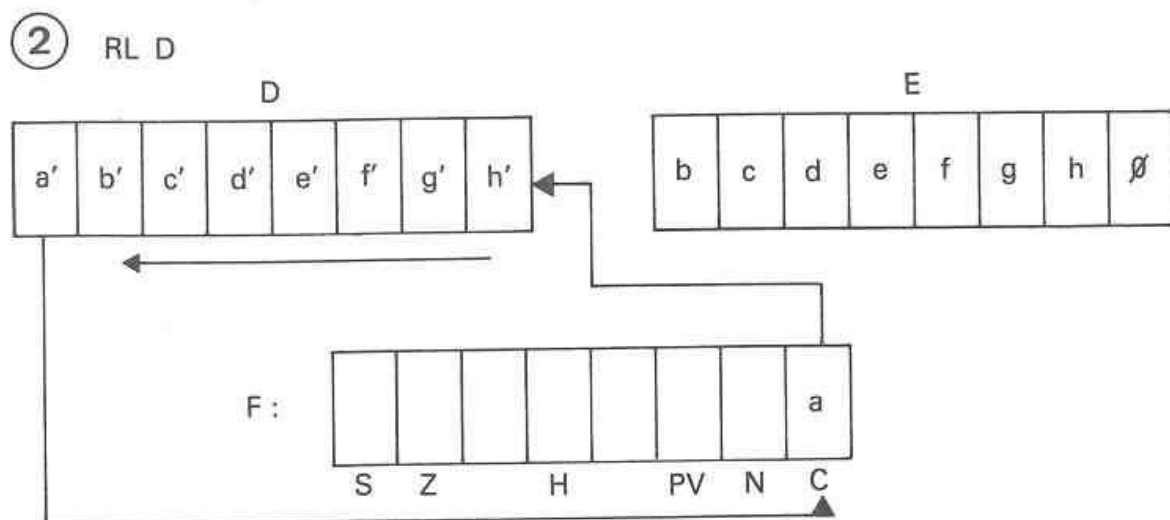
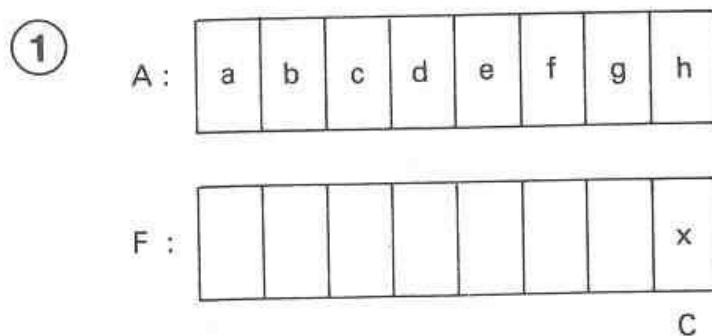


Schéma 3.4

Décalage de registre 16 bits.

Nous avons huit chiffres dans A. RRA permet de décaler ces bits vers la droite, et de récupérer le chiffre ainsi éjecté hors de A dans le Carry, par exemple pour le tester et savoir si l'on doit additionner un résultat dans HL ou non.



② RRA

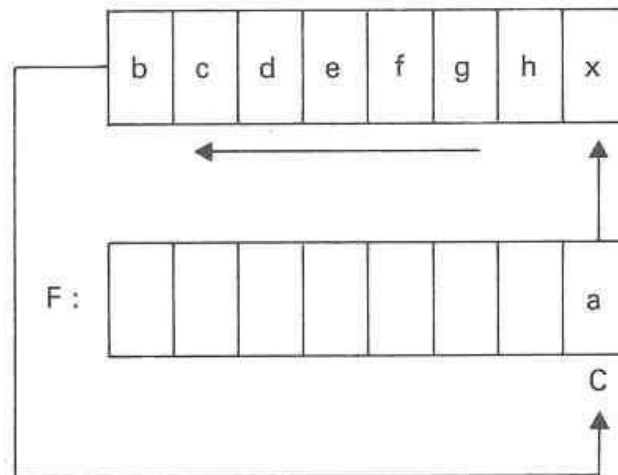


Schéma 3.5

Décalage à droite.

Voici finalement notre routine :

MULT:	LD	B,8	: il y a huit chiffres à examiner dans A, nous allons donc faire huit boucles ;
	LD	HL,0	: nous mettons le stockage du résultat à zéro.
	RRA		: premier décalage : le bit 0 de A dans le Carry.
POIDS:	JP	NC,SUIV	: si le Carry est à zéro, le chiffre était zéro, il n'y a donc aucune addition à faire, simplement passer au chiffre suivant.
	ADD	HL,DE	: le chiffre était 1, nous additionnons donc le résultat intermédiaire dans HL.
SUIV:	SLA	E	
	RL	D	: nous décalons DE vers la gauche en ajoutant un 0 à sa droite. De cette façon, DE représente le résultat intermédiaire pour le prochain chiffre.
	RRA		: nous envoyons par rotation à droite le prochain chiffre dans le Carry.
	DJNZ	POIDS	: suite... jusqu'à ce que les huit chiffres aient été examinés. Remarque : DJNZ ne modifie pas F, et ne touche pas le contenu du Carry.

Si nous nous sommes étendus sur cette routine, c'est qu'elle représente l'esprit même du langage machine. Dans le pire des cas, cette routine procède à 8 additions, au lieu de 255 pour la première version. Bien sûr, son fonctionnement est plus complexe. Mais l'utilisation des décalages et rotations montre jusqu'à quel point on peut optimiser une routine LM. De plus, cette routine pourra toujours servir plus tard.

Le tracé en assembleur

Maintenant que nous avons notre routine de multiplication, le reste est simple. Tout au plus y a-t-il quelques subtilités dues à la méthode de tracé. Notamment, il nous faut disposer de X et Y, mais aussi de $-X$ et $-Y$ pour tracer nos huit points. Or, notre routine est incapable de multiplier par un nombre négatif. De plus, il serait stupide de procéder à une seconde multiplication simplement pour inverser un signe. La solution, HL contenant notre X ou notre Y, est la suivante :

```
LD A,L
CPL
LD L,A
LD H,255
```

Mais attention : ceci convient car nous savons, par définition, que X et Y ne dépassent pas 256 (puisque'ils sont divisés par 256, après la multiplication par le sinus ou le cosinus, eux-mêmes implicitement multipliés par 256). Ensuite, HL contient effectivement $-X$ ou $-Y$.

Enfin, nous arrivons au tracé des points proprement dit. Nous l'avons laissé entendre plus haut, l'Amstrad nous facilite grandement la tâche à ce niveau. En effet, un vecteur situé en \$BBEA pointe la routine PLOT. Il suffit d'effectuer CALL \$BBEA pour allumer le point situé en (DE,HL). Seul défaut de la routine : elle modifie HL et DE (entre autres) et nous oblige donc à récupérer avant chaque tracé de point les valeurs de X, $-X$, Y ou $-Y$ dans HL et DE.

C'est une des particularités gênantes des routines système : une grande partie d'entre elles "mange" ainsi des registres que l'on aimerait conserver. Les solutions : ou bien l'on stocke les valeurs dans la pile et on les récupère par des POP (suivis de PUSH pour les réinstaller) entre chaque appel de routine, ou bien on les place à des adresses bien déterminées. Cette dernière solution est de très loin la plus satisfaisante quant à la rapidité, bien qu'elle nous oblige à grignoter un certain nombre d'octets en plus.

Vous pouvez constater, dans le listing assembleur du programme 3.2 (v.p. 69) qu'à part les points évoqués en détail ci-dessus, la routine est assez simple. Elle travaille exactement comme le programme 3.1 Basic vu plus haut.

```

10 ;
20 ;trace de cercles et d'ellipses par LM
30 ;utilise une table des cosinus et sinus
40 ;(prog 3.2)
50 ;methode point par point
60 ;
BBC9 70 ORIGIN: EQU #BBC9
BBEA 80 PLOT: EQU #BBEA
90 ;
100 ;suppose que la couleur d'écriture graphique
110 ;est déjà sélectionnée
120 ;
4000 130 ORG #4000
140 ;
4000 DD7E00 150 BASIC: LD A,(IX+0) ;rayon demande
4003 32B740 160 LD (RAYON),A
4006 DD6E02 170 LD L,(IX+2)
4009 DD6603 180 LD H,(IX+3) ;Y centre
400C DD5E04 190 LD E,(IX+4)
400F DD5605 200 LD D,(IX+5) ;X centre
4012 CDC9EB 210 CALL ORIGIN
220 ;
4015 062E 230 LM: LD B,46 ;nombre de boucles
4017 11C140 240 LD DE,TABLE ;table des sinus et cosinus
401A C5 250 POINT: PUSH BC ;sauve compteur de boucles
401B 1A 260 LD A,(DE) ;sinus
401C D5 270 PUSH DE
401D ED5BB740 280 LD DE,(RAYON) ;rayon du cercle
4021 CDA340 290 CALL MULT ;multiplie sinus(A) par rayon(DE)
4024 6C 300 LD L,H ;division par 256
4025 2600 310 LD H,0 ;
320 ;
4027 22BB40 330 LD (Y),HL ;stocke y
340 ;
402A 7D 350 LD A,L
402B 2F 360 CPL
402C 6F 370 LD L,A
402D 26FF 380 LD H,255
402F 22BF40 390 LD (YN),HL ;stocke -y
400 ;
4032 D1 410 POP DE
4033 13 420 INC DE ;passe au cosinus dans table
4034 1A 430 LD A,(DE) ;cosinus
4035 13 440 INC DE ;pour la suite de la table
4036 D5 450 PUSH DE
4037 ED5BB740 460 LD DE,(RAYON)
403B CDA340 470 CALL MULT ;multiplie cosinus(A) par rayon(DE)
403E 6C 480 LD L,H ;division par 256
403F 2600 490 LD H,0
500 ;
4041 22B940 510 LD (X),HL ;stocke x
520 ;

```



```

4044 7D      530      LD  A,L
4045 2F      540      CPL
4046 6F      550      LD  L,A
4047 26FF    560      LD  H,255      ;
4049 22BD40  570      LD  (XN),HL      ;stocke -x
                    580 ;
                    590 ;trace effectif d'apres X, Y ,XN et YN
                    600 ;
404C ED5BB940 610      LD  DE,(X)
4050 2AB840  620      LD  HL,(Y)
4053 CDEABB  630      CALL PLOT      ; (x,y)
4056 ED5BB940 640      LD  DE,(X)
405A 2ABF40  650      LD  HL,(YN)
405D CDEABB  660      CALL PLOT      ; (x,-y)
4060 ED5BB040 670      LD  DE,(XN)
4064 2AB840  680      LD  HL,(Y)
4067 CDEABB  690      CALL PLOT      ; (-x,y)
406A ED5BB040 700      LD  DE,(XN)
406E 2ABF40  710      LD  HL,(YN)
4071 CDEABB  720      CALL PLOT      ; (-x,-y)
4074 ED5BB840 730      LD  DE,(Y)
4078 2AB940  740      LD  HL,(X)
407B CDEABB  750      CALL PLOT      ; (y,x)
407E ED5BB840 760      LD  DE,(Y)
4082 2ABD40  770      LD  HL,(XN)
4085 CDEABB  780      CALL PLOT      ; (y,-x)
4088 ED5BBF40 790      LD  DE,(YN)
408C 2AB940  800      LD  HL,(X)
408F CDEABB  810      CALL PLOT      ; (-y,x)
4092 ED5BBF40 820      LD  DE,(YN)
4096 2ABD40  830      LD  HL,(XN)
4099 CDEABB  840      CALL PLOT      ; (-y,-x)
409C D1      850      POP DE      ;recupere table sinus et cosinus
409D C1      860      POP BC      ;recupere compteur
409E 05      870      DEC B
409F C21A40  880      JP  NZ,POINT      ;suite du trace
40A2 C9      890      RET      ;et fin !
                    900 ;
                    910 ;routine multipliant DE par A dans HL
                    920 ;
40A3 0608    930 MULT: LD  B,B      ;il y a huit bits
40A5 210000  940      LD  HL,0
40A8 CB2F    950      SRA A      ;bit 0 dans carry
                    960 ;
40AA D2AE40  970 POIDS: JP  NC,SUIV      ;pas d'addition
40AD 19      980      ADD HL,DE      ;addition de la puissance
40AE CB23    990 SUIV: SLA E      ;multiple DE par deux
40B0 CB12   1000     RL  D      ;idem, recupere carry de SLA E
40B2 CB2F   1010     SRA A      ;decale A, copier bit 0 dans Carry
40B4 10F4   1020     DJNZ POIDS      ;suite multiplication
40B6 C9     1030     RET
                    1040 ;

```

```

40B7 0000    1050 RAYON: DEFW #0000
40B9 0000    1060 X:   DEFW #0000
40BB 0000    1070 Y:   DEFW #0000
40BD 0000    1080 XN:  DEFW #0000
40BF 0000    1090 YN:  DEFW #0000
40C1 00FF04FF 1100 TABLE: DEFB 0,255,4,255,9,255,13,255
40C9 12FF16FF 1110      DEFB 18,255,22,255,27,254,31,254
40D1 24FE28FD 1120      DEFB 36,254,40,253,44,252,49,251
40D9 35FA3AF9 1130      DEFB 53,250,58,249,62,248,66,247
40E1 47F64BF5 1140      DEFB 71,246,75,245,79,243,83,242
40E9 58F15CEF 1150      DEFB 88,241,92,239,96,237,100,236
40F1 68EA6CE8 1160      DEFB 104,234,108,232,112,230,116,228
40F9 78E27CE0 1170      DEFB 120,226,124,224,128,222,132,219
4101 88D98BD7 1180      DEFB 136,217,139,215,143,212,147,210
4109 96CF9ACC 1190      DEFB 150,207,154,204,158,202,161,199
4111 A5C4A8C1 1200      DEFB 165,196,168,193,171,190,175,187
4119 B2B8B5B5 1210      DEFB 178,184,181,181

```

Pass 2 errors: 00

Si vous possédez un assembleur, vous pouvez sauver le programme sur cassette ou disquette et le récupérer par la séquence Basic suivante :

```

MEMORY &3FFF
LOAD "nom",&4000

```

Par la suite, l'utilisation se résumera à un appel "CALL &4000,X,Y,R" où X et Y sont les coordonnées du centre du cercle et R son rayon. Vous pouvez exécuter les lignes 300 à 330 du programme Basic 3.2 pour apprécier le gain de rapidité obtenu !

```

10 *****
20 ** Programme 3.2 **
30 *****
40 '
50 'trace de cercles en LM point par point
60 'avec une table des sinus et cosinus calculés
70 'interface en Basic:
80 'CALL cercle,X,Y,R
90 '
100 MEMORY &3FFF
110 DEFINT a-z
120 ad=&4000:lign=200
130 ctrl=0:READ c$:IF c$="fin" THEN 300
140 FOR i=1 TO LEN(c$) STEP 2
150 c=VAL("&" + MID$(c$,i,2))

```

```

160 POKE ad,c:ad=ad+1:ctrl=ctrl+c
170 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
180 lign=lign+10:GOTO 130
190 '
200 DATA DD7E0032B740DD6E02DD6603DD5E04DD5605CDC
    9BB062E11C140C51AD5ED5BB7, 3800
210 DATA 40CDA3406C260022BB407D2F6F26FF22BF40D11
    31A13D5ED5BB740CDA3406C26, 3431
220 DATA 0022B9407D2F6F26FF22BD40ED5BB9402ABB40C
    DEABBED5BB9402ABF40CDEABB, 4142
230 DATA ED5BBD402ABB40CDEABBED5BBD402ABF40CDEAB
    BED5BBB402AB940CDEABBED5B, 4828
240 DATA BB402ABD40CDEABBED5BBF402AB940CDEABBED5
    BBF402ABD40CDEABBD1C105C2, 4777
250 DATA 1A40C90608210000CB2FD2AE4019CB23CB12CB2
    F10F4C900000000000000000, 2231
260 DATA 0000FF04FF09FF0DFF12FF16FF1BFE1FFE24FE2
    8FD2CFC31FB35FA3AF93EF842, 4327
270 DATA F747F64BF54FF353F258F15CEF60ED64EC68EA6
    CE870E674E478E27CE080DE84, 5400
280 DATA DB88D98BD78FD493D296CF9ACC9ECAA1C7A5C4A
    8C1ABBEAFBBB2B8B5B5000000, 5242
290 DATA fin
300 MODE 2:INK 0,0:INK 1,20:PLOT 800,800,1:REM (
    selection encre graphique)
310 FOR r=10 TO 250 STEP 10
320 CALL &4000,320,200,r
330 NEXT
340 INPUT "Tapez <ENTER> pour un autre dessin ";
    a$
350 '
360 'autre exemple de dessin
370 'semi-animation par modification de couleurs
380 '
390 MODE 1
400 'selection des couleurs
410 FOR i=1 TO 3:INK i,i*4:NEXT
420 'encre de trace=1 au depart
430 i=1
440 DEG
450 'serpentin du haut
460 FOR r=0 TO 150 STEP 2
470 'on trace trois cercles par boucle pour obte
    nir un cercle epais
480 PLOT 800,800,i:x=320+COS(r)*r*4:y=200+r*SIN(
    r)*2:FOR l=r TO r+4 STEP 2:CALL &4000,x,y,l:N
    EXT
490 'passe a l'encre suivante

```

```

500 i=i+1:IF i=4 THEN i=1
510 NEXT
520 'serpentin du bas
530 FOR r=150 TO 0 STEP -2
540 PLOT 800,800,i:x=320-COS(r)*r*4:y=200-r*SIN(
    r)*2:FOR l=r TO r+4 STEP 2:CALL &4000,x,y,l:N
    EXT
550 'passe a l'encre suivante
560 i=i+1:IF i=4 THEN i=1
570 NEXT
580 '
590 'animation des encres
600 '
610 k=1
620 FOR i=1 TO 3:INK 1+((k+i-1) MOD 3),i*4
630 FOR l=1 TO 30:NEXT
640 NEXT
650 k=k+1:IF k=4 THEN k=1
660 GOTO 620

```

Le programme 3.2 comporte des valeurs de vérification des DATA afin d'éviter les erreurs. Toutefois, nous recommandons à ceux qui veulent le taper d'effectuer une sauvegarde avant son exécution. Il en sera de même avec tous les programmes du livre : **n'exécutez jamais un programme LM avant de l'avoir sauvegardé !**

Comme il est expliqué précédemment, une deuxième méthode de tracé existe : nous pouvons tracer un cercle en reliant les points par des lignes. Cela donne un résultat beaucoup plus agréable. En revanche, nous perdons sur deux tableaux : la place occupée par la routine, et la vitesse d'exécution. En effet, puisqu'il faut au minimum effectuer les mêmes opérations que dans le programme 3.2, nous ne pouvons que rajouter des instructions. Et de même, puisqu'il faudra, pour chaque point, ajouter un tracé de ligne, nous allons ralentir de façon sensible l'exécution.

Nous ne pouvons rien faire contre la perte de place, mais nous pouvons, par contre, compenser la perte de vitesse en n'utilisant que la moitié des sinus et des cosinus (22 angles entre 0 et 45 degrés, au lieu de 44). Dans la théorie, le tracé obtenu est un peu plus anguleux, mais la pratique prouve que les lignes reliant les points amortissent les angles et donnent un tracé satisfaisant.

La méthode de tracé par lignes est un peu complexe en théorie : en effet, on ne peut, par la routine DRAW disponible, tracer une droite qu'à partir du dernier point tracé. Or, notre routine ne retrouve un point contigu à un autre que tous les huit tracés !

Pour venir à bout de ce problème, il nous suffit de considérer que les huit tracés correspondant à un angle sont en fait caractérisés par quatre

nombres : X , $-X$, Y et $-Y$. Pour tracer une droite entre deux points contigus, il suffit donc, avant de passer au calcul des X et Y suivants (pour l'angle suivant), de mémoriser les X et Y venant d'être tracés.

Dans la routine 3.3, nous avons appelé $X1$ la variable retenant l'ancienne valeur de X et $Y1$ celle retenant Y . Ces variables sont mises à jour immédiatement avant le calcul des nouveaux X et Y . Puis, on trace la ligne entre $(X1,Y1)$ et (X,Y) , entre $(-X1,Y1)$ et $(-X,Y)$, et ainsi de suite.

```

10 ;
20 ;trace de cercles et d'ellipses par LM
30 ;utilise une table des cosinus et sinus
40 ;(prog 3.3)
50 ;methode par lignes
60 ;
BBC9      70 ORIGIN: EQU #BBC9
BBC0      80 MOVE:  EQU #BBC0
BBF6      90 DRAW:  EQU #BBF6
100 ;
110 ;suppose que la couleur d'écriture graphique
120 ;est déjà sélectionnée
130 ;
4000      140      ORG #4000
150 ;
4000 DD7E00 160 BASIC: LD A,(IX+0)      ;rayon demande
4003 323241 170      LD (RAYON),A
4006 DD6E02 180      LD L,(IX+2)
4009 DD6603 190      LD H,(IX+3)      ;Y centre
400C DD5E04 200      LD E,(IX+4)
400F DD5605 210      LD D,(IX+5)      ;X centre
4012 CDC9BB 220      CALL ORIGIN
230 ;
4015 ED5B3241 240 LM: LD DE,(RAYON)
4019 3EFF      250      LD A,255
401B CD1E41      260      CALL MULT
401E 6C      270      LD L,H
401F 2600      280      LD H,0
4021 223441      290      LD (X),HL
4024 7D      300      LD A,L
4025 2F      310      CPL
4026 6F      320      LD L,A
4027 26FF      330      LD H,255
4029 223841      340      LD (XN),HL      ;initialisation du premier point
402C 210000      350      LD HL,0
402F 223641      360      LD (Y),HL
4032 223A41      370      LD (YN),HL
4035 114441      380      LD DE,TABLE      ;table des sinus et cosinus

```

```

4038 0618      390      LD  B,24
               400 ;
403A C5        410 POINT: PUSH BC          ;sauve compteur de boucles
403B 1A        420      LD  A,(DE)        ;sinus
403C D5        430      PUSH DE
               440 ;
               450 ;transfert des anciennes coordonnees
               460 ;
403D 213441    470      LD  HL,X
4040 113C41    480      LD  DE,X1
4043 010800    490      LD  BC,8
4046 EDB0      500      LDIR              ;transfere anciennes coordonnees
               510 ;
4048 ED5B3241  520      LD  DE,(RAYON)    ;rayon du cercle
404C CD1E41    530      CALL MULT         ;multiplie sinus(A) par rayon(DE)
404F 6C        540      LD  L,H           ;division par 256
4050 2600      550      LD  H,0           ;
4052 223641    560      LD  (Y),HL        ;stocke y
               570 ;
4055 7D        580      LD  A,L
4056 2F        590      CPL
4057 6F        600      LD  L,A
4058 26FF      610      LD  H,255
405A 223A41    620      LD  (YN),HL       ;stocke -y
               630 ;
405D D1        640      POP  DE
405E 13        650      INC  DE           ;passe au cosinus dans table
405F 1A        660      LD  A,(DE)        ;cosinus
4060 13        670      INC  DE           ;pour la suite de la table
4061 D5        680      PUSH DE
4062 ED5B3241  690      LD  DE,(RAYON)
4066 CD1E41    700      CALL MULT         ;multiplie cosinus(A) par rayon(DE)
4069 6C        710      LD  L,H           ;division par 256
406A 2600      720      LD  H,0
               730 ;
406C 223441    740      LD  (X),HL        ;stocke x
               750 ;
406F 7D        760      LD  A,L
4070 2F        770      CPL
4071 6F        780      LD  L,A
4072 26FF      790      LD  H,255        ;
4074 223841    800      LD  (XN),HL       ;stocke -x
               810 ;
               820 ;trace effectif d'apres X, Y ,XN et YN
               830 ;
4077 ED5B3C41  840      LD  DE,(X1)
407B 2A3E41    850      LD  HL,(Y1)
407E CDC0BB    860      CALL MOVE
4081 ED5B3441  870      LD  DE,(X)

```


4085	2A3641	880	LD HL,(Y)	
4088	CDF6BB	890	CALL DRAW	; (x,y)
408B	ED5B3C41	900	LD DE,(X1)	
408F	2A4241	910	LD HL,(YN1)	
4092	CDC0BB	920	CALL MOVE	
4095	ED5B3441	930	LD DE,(X)	
4099	2A3A41	940	LD HL,(YN)	
409C	CDF6BB	950	CALL DRAW	; (x,-y)
409F	ED5B4041	960	LD DE,(XN1)	
40A3	2A3E41	970	LD HL,(Y1)	
40A6	CDC0BB	980	CALL MOVE	
40A9	ED5B3B41	990	LD DE,(XN)	
40AD	2A3641	1000	LD HL,(Y)	
40B0	CDF6BB	1010	CALL DRAW	; (-x,y)
40B3	ED5B4041	1020	LD DE,(XN1)	
40B7	2A4241	1030	LD HL,(YN1)	
40BA	CDC0BB	1040	CALL MOVE	
40BD	ED5B3B41	1050	LD DE,(XN)	
40C1	2A3A41	1060	LD HL,(YN)	
40C4	CDF6BB	1070	CALL DRAW	; (-x,-y)
40C7	ED5B3E41	1080	LD DE,(Y1)	
40CB	2A3C41	1090	LD HL,(X1)	
40CE	CDC0BB	1100	CALL MOVE	
40D1	ED5B3641	1110	LD DE,(Y)	
40D5	2A3441	1120	LD HL,(X)	
40D8	CDF6BB	1130	CALL DRAW	; (y,x)
40DB	ED5B3E41	1140	LD DE,(Y1)	
40DF	2A4041	1150	LD HL,(XN1)	
40E2	CDC0BB	1160	CALL MOVE	
40E5	ED5B3641	1170	LD DE,(Y)	
40E9	2A3B41	1180	LD HL,(XN)	
40EC	CDF6BB	1190	CALL DRAW	; (y,-x)
40EF	ED5B4241	1200	LD DE,(YN1)	
40F3	2A3C41	1210	LD HL,(X1)	
40F6	CDC0BB	1220	CALL MOVE	
40F9	ED5B3A41	1230	LD DE,(YN)	
40FD	2A3441	1240	LD HL,(X)	
4100	CDF6BB	1250	CALL DRAW	; (-y,x)
4103	ED5B4241	1260	LD DE,(YN1)	
4107	2A4041	1270	LD HL,(XN1)	
410A	CDC0BB	1280	CALL MOVE	
410D	ED5B3A41	1290	LD DE,(YN)	
4111	2A3B41	1300	LD HL,(XN)	
4114	CDF6BB	1310	CALL DRAW	; (-y,-x)
4117	D1	1320	POP DE	; recupere table sinus et cosinus
4118	C1	1330	POP BC	; recupere compteur
4119	05	1340	DEC B	
411A	C23A40	1350	JP NZ,POINT	; suite du trace
411D	C9	1360	RET	; et fin !

```

1370 ;
1380 ;routine multipliant DE par A dans HL
1390 ;
411E 0608 1400 MULT: LD B,8 ;il y a huit bits
4120 210000 1410 LD HL,0
4123 CB2F 1420 SRA A ;bit 0 dans carry
1430 ;
4125 D22941 1440 POIDS: JP NC,SUIV ;pas d'addition
4128 19 1450 ADD HL,DE ;addition de la puissance
4129 CB23 1460 SUIV: SLA E ;multiple DE par deux
412B CB12 1470 RL D ;idem, recupere carry de SLA E
412D CB2F 1480 SRA A ;decale A, copier bit 0 dans Carry
412F 10F4 1490 DJNZ POIDS ;suite multiplication
4131 C9 1500 RET
1510 ;
4132 0000 1520 RAYON: DEFW #0000
4134 0000 1530 X: DEFW #0000
4136 0000 1540 Y: DEFW #0000
4138 0000 1550 XN: DEFW #0000
413A 0000 1560 YN: DEFW #0000
413C 0000 1570 X1: DEFW #0000
413E 0000 1580 Y1: DEFW #0000
4140 0000 1590 XN1: DEFW #0000
4142 0000 1600 YN1: DEFW #0000
4144 00FF09FF 1610 TABLE: DEFB 0,255,9,255
4148 12FF1BFE 1620 DEFB 18,255,27,254
414C 24FE2CFC 1630 DEFB 36,254,44,252
4150 35FA3EF8 1640 DEFB 53,250,62,248
4154 47F64FF3 1650 DEFB 71,246,79,243
4158 58F160ED 1660 DEFB 88,241,96,237
415C 68EA70E6 1670 DEFB 104,234,112,230
4160 78E280DE 1680 DEFB 120,226,128,222
4164 88D98FD4 1690 DEFB 136,217,143,212
4168 96CF9ECA 1700 DEFB 150,207,158,202
416C A5C4ABBE 1710 DEFB 165,196,171,190
4170 B2B8B5B5 1720 DEFB 178,184,181,181

```

Si l'on excepte la disparition dans la table sinus/cosinus de la moitié des valeurs, il n'y a aucune autre différence entre les routines 3.2 et 3.3.

Le tracé de cercles concentriques du programme 3.3 (v.p. 78) est beaucoup plus joli que celui du 3.2, il faut le reconnaître. Mais vous pouvez aussi constater qu'il est plus lent, et que la routine est plus encombrante. Pourtant, les différences sont faibles, et nous avons divisé le nombre de calculs par deux.

```

10 '*****
20 '** Programme 3.3 **
30 '*****
40 '
50 'trace de cercles en LM par lignes
60 'avec une table des sinus et cosinus reduite
70 'interface en Basic:
80 'CALL cercle,X,Y,R
90 '
100 MEMORY &3FFF
110 DEFINT a-z
120 ad=&4000:lign=200
130 ctrl=0:READ c$:IF c$="fin" THEN 330
140 FOR i=1 TO LEN(c$) STEP 2
150 c=VAL("&" + MID$(c$,i,2))
160 POKE ad,c:ad=ad+1:ctrl=ctrl+c
170 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
180 lign=lign+10:GOTO 130
190 '
200 DATA DD7E00323241DD6E02DD6603DD5E04DD5605CDC
    9BBED5B32413EFFCD1E416C26, 3601
210 DATA 002234417D2F6F26FF223841210000223641223
    A411144410618C51AD5213441, 1991
220 DATA 113C41010800EDB0ED5B3241CD1E416C2600223
    6417D2F6F26FF223A41D1131A, 2593
230 DATA 13D5ED5B3241CD1E416C26002234417D2F6F26F
    F223841ED5B3C412A3E41CDC0, 3022
240 DATA BBED5B34412A3641CDF6BBED5B3C412A4241CDC
    0BBED5B34412A3A41CDF6BBED, 4126
250 DATA 5B40412A3E41CDC0BBED5B38412A3641CDF6BBE
    D5B40412A4241CDC0BBED5B38, 3819
260 DATA 412A3A41CDF6BBED5B3E412A3C41CDC0BBED5B3
    6412A3441CDF6BBED5B3E412A, 3815
270 DATA 4041CDC0BBED5B36412A3841CDF6BBED5B42412
    A3C41CDC0BBED5B3A412A3441, 3781
280 DATA CDF6BBED5B42412A4041CDC0BBED5B3A412A384
    1CDF6BBD1C105C23A40C90608, 4042
290 DATA 210000CB2FD2294119CB23CB12CB2F10F4C9000
    000000000000000000000000, 1794
300 DATA 0000000000FF09FF12FF1BFE24FE2CFC35FA3EF
    847F64FF358F160ED68EA70E6, 4253
310 DATA 78E280DE88D98FD496CF9ECA5C4ABBE2B8B5B
    5000000000000000000000000000, 3567
320 DATA fin
330 MODE 2
340 FOR r=10 TO 250 STEP 10
350 CALL &4000,320,200,r
360 NEXT

```

Inconvénients des routines systèmes

Alors que nous devions obtenir une routine à peine plus grosse et plus rapide, il se produit le résultat inverse.

Nous allons nous répéter pour l'expliquer. D'une part, les routines systèmes de l'Amstrad sont lentes, très lentes. Elles sont même quasiment inexploitable pour celui qui veut gérer ses graphismes à très grande vitesse. Nous ne reviendrons pas là-dessus : la seule solution reste l'accès direct à la mémoire écran. Toutefois, le tracé d'un point ou d'une droite ne peut guère être très rapide si l'on utilise le système de coordonnées standard. En effet, PLOT et DRAW sont lentes parce qu'elles tiennent compte de tout : les limites de la fenêtre graphique, la couleur sélectionnée, etc.

D'autre part, nous y revenons, les routines sont puissantes et nombreuses, mais elles utilisent sans scrupule les registres disponibles, ce qui oblige le programmeur à de lourdes tâches de sauvegarde. Ces tâches, dans le programme 3.3, occupent à elles seules 112 octets lors des 16 appels de routines système.

Toutefois, si nous avons réalisé ces routines de tracé de cercle, ce n'est pas uniquement pour attirer votre attention sur ces problèmes. En effet, si vous essayez d'imaginer un tracé de cercles ne passant pas par les routines systèmes, vous comprenez immédiatement pourquoi celles-ci ne sont pas inutiles, malgré tous leurs défauts. Certes, la gestion de graphismes figés, comme dans les jeux d'action, se contente facilement de la plus rapide solution : l'accès direct à la mémoire écran. Mais il en va tout autrement dès que l'on parle de graphismes obtenus par calcul : il faut transcrire le résultat des calculs en données pour la mémoire écran, on voit alors surgir l'intérêt de routines systèmes prêtes à l'emploi et d'un système de coordonnées bien gérées.

TRACÉ D'HISTOGRAMMES

Histogramme

Qu'il s'agisse de synthèse d'image, de graphismes utilitaires professionnels ou de figures géométriques, et à condition que la vitesse d'exécution ne soit pas déterminante, le programmeur devra profiter de l'existence de ces routines prêtes à l'emploi. Elles forment un macrolangage graphique.

Nous allons d'ailleurs pouvoir le constater dans ce qui suit. Nous avons évoqué plus haut les graphismes professionnels tels que les camemberts ou les histogrammes. La programmation d'un tracé d'histogrammes ne

manque pas d'intérêt. Les problèmes posés ainsi que les applications possibles justifient pleinement une étude approfondie.

Pour cela, nous allons examiner le problème. Un histogramme est la représentation, par des colonnes, d'une suite de valeurs, la hauteur d'une colonne étant proportionnelle à la valeur qu'elle représente (*schéma 3.6*).

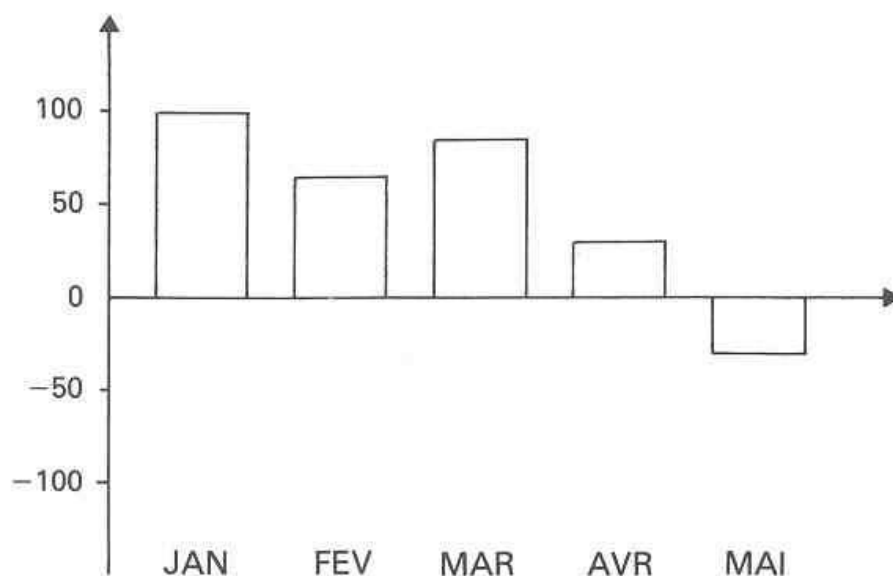


Schéma 3.6

Exemple d'histogramme.

Ces diagrammes sont souvent utilisés en gestion, afin de visualiser, sous forme simple, un tableau de valeurs comme des résultats de ventes sur une année. Il est connu qu'un bon dessin vaut toutes les listes de chiffres du monde. Les histogrammes en sont un vivant exemple, et ils peuvent entrer dans une multitude d'applications.

L'Amstrad ne possède pas d'instruction de tracé automatique d'histogrammes. Nous allons donc y remédier.

Fenêtre de travail

Le principal problème est le suivant : alors que l'écran de notre ordinateur est d'une taille limitée (en mode 1, nous ne disposons que de 320 points de large sur 200 de haut), nous ne connaissons rien des valeurs à représenter. Elles peuvent aussi bien s'échelonner entre 0 et 10 qu'entre -32768 et +32767. Notre routine devra donc effectuer un calcul d'échelle afin de caser au mieux les valeurs sur l'écran.

En faisant un compromis, nous allons volontairement adopter le mode 1 lors du tracé. Si nous prenons le mode 0, la résolution horizontale risque d'être insuffisante. N'oublions pas en effet que, dans ce mode, l'ordinateur ne peut plus afficher que vingt énormes caractères par ligne. Ce sera peu si nous voulons placer quelques commentaires autour du schéma. Et en mode 2, nous n'avons plus que 2 couleurs, c'est-à-dire que les colonnes seront toutes de la même couleur. Le résultat serait d'une affligeante tristesse. Il est désormais admis, en milieu professionnel (gestion, commercial...) que le sérieux d'un sujet ne doit pas pour autant le rendre triste. Nous déciderons donc de colorer un peu l'écran. Une série de colonnes uniformes n'est guère plus enthousiasmante qu'un tableau de chiffres.

L'écran en mode 1 dispose de 320 points sur 200. Mais il va également falloir réduire cet espace. Il est sage de laisser une place libre autour du schéma. Il sera ainsi possible de placer un titre, une échelle des valeurs, des commentaires.

Le choix des espaces libres est purement conventionnel. La logique conduit toutefois à garder de la place à gauche et en dessous du dessin. À gauche, car il est alors possible d'afficher les valeurs correspondant aux colonnes, en dessous, pour écrire un titre ou un libellé sous chaque colonne.

Le rectangle restant, qui recevra les colonnes, sera placé à partir du point physique (50,20). Cela nous laisse 270 points de large et 180 de haut. En compensation, il y a de la place pour six caractères à gauche, et pour deux lignes de caractères en dessous. Le compromis est acceptable (schéma 3.7).

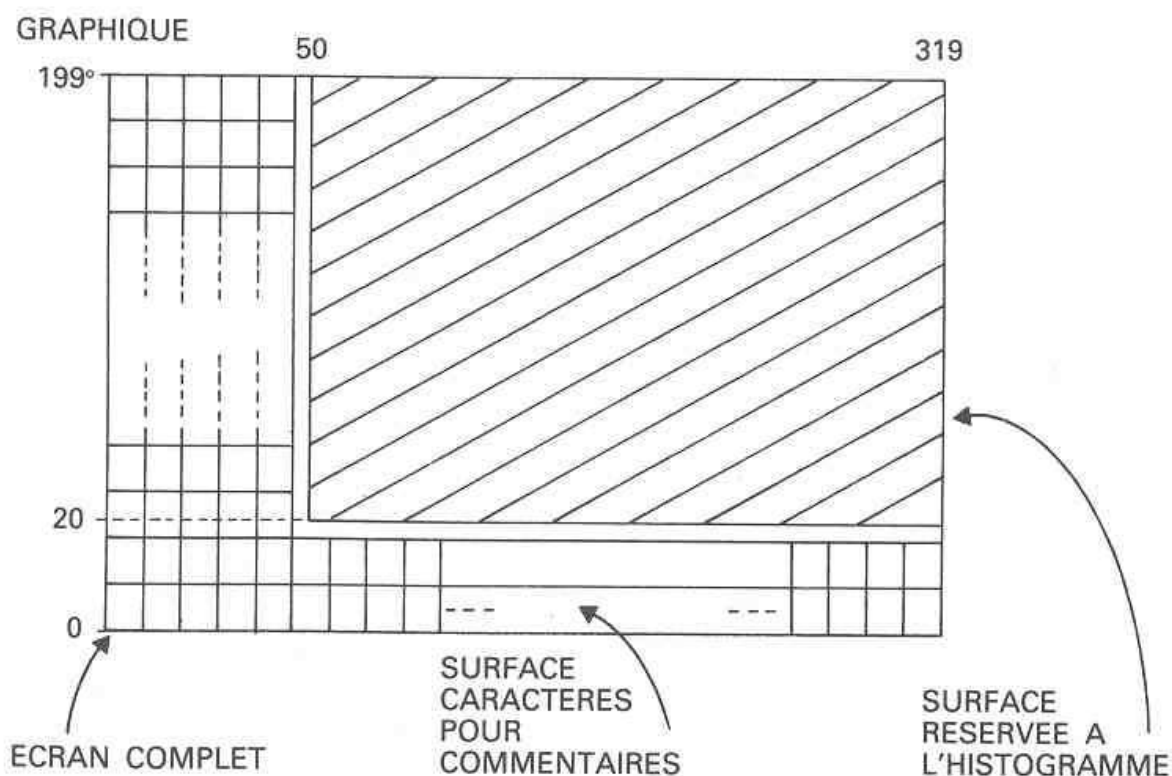


Schéma 3.7

Écran réservé à l'histogramme.

Le programme appelant demandera le tracé de N valeurs quelconques. Pour simplifier le tracé, il devra également fournir une autre donnée. La largeur des colonnes (en nombre de points) est vitale pour la routine. Si nous traçons 10 valeurs, nous pouvons utiliser jusqu'à 180/10 soit 18 points de large pour chaque colonne. Le dessin occupera ainsi la largeur maximale qu'on peut lui accorder. Mais en revanche, s'il faut tracer 30 valeurs, il faut réduire cette largeur de colonne, faute de quoi les vingt dernières valeurs se trouveront à droite, en dehors de l'écran !

En réalité, le calcul automatique de la largeur de tracé est simple et pourrait être intégré à la routine. Mais rien ne dit que l'utilisateur désire vraiment un histogramme occupant tout l'écran graphique. Il pourra donc, en modulant la largeur des colonnes, agir sur l'encombrement horizontal du schéma afin de libérer de l'espace à droite.

Cela pourrait par exemple être utilisé pour incorporer un second schéma à droite de l'histogramme (schéma 3.8).

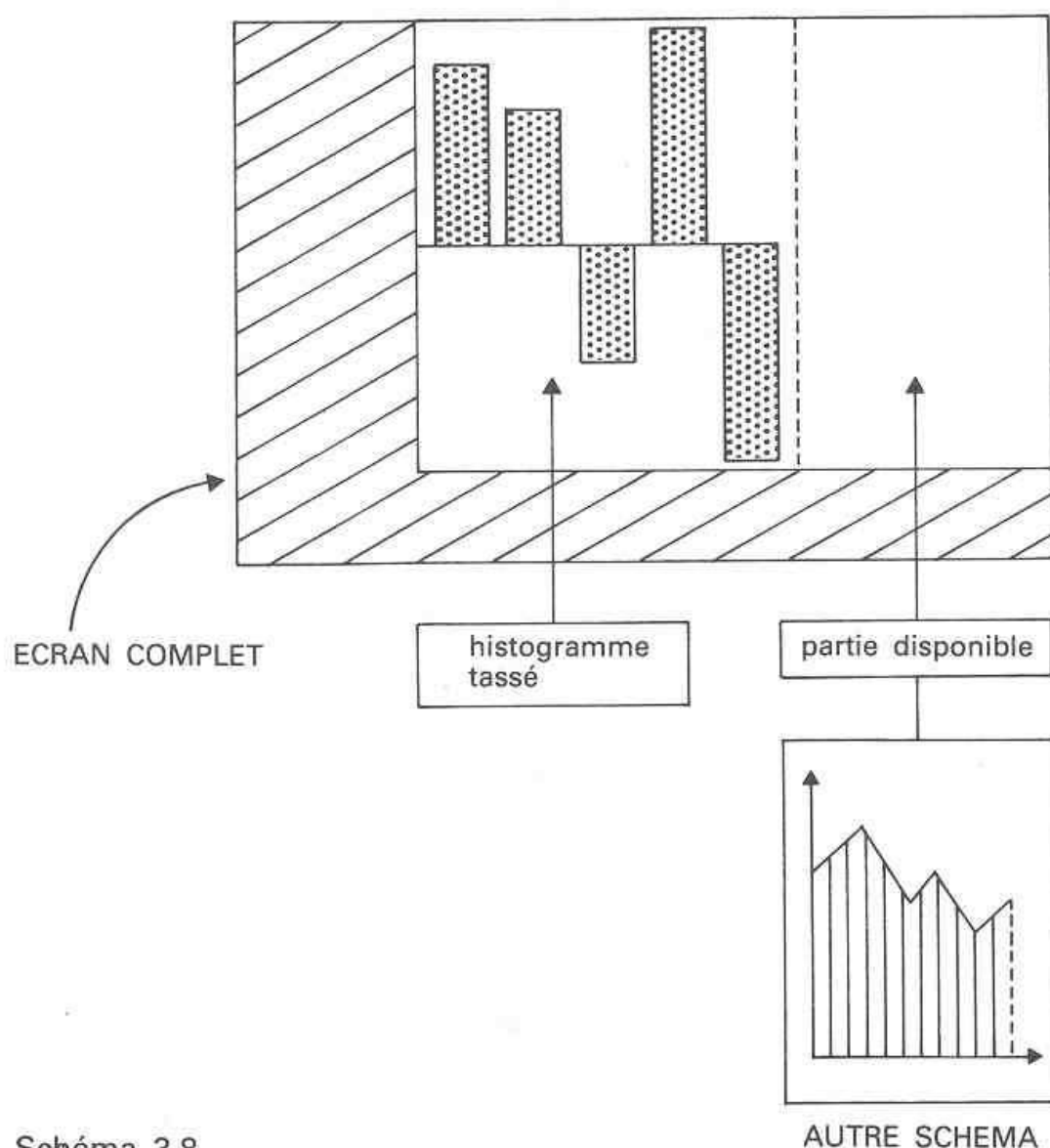


Schéma 3.8

Histogramme tassé pour laisser place à droite (autre schéma).

Enfin, ultime décision influant sur le tracé, la routine espacera chaque colonne par deux points graphiques. Cela permettra d'apporter un peu de clarté, quoique la chose soit discutable. Si les valeurs à représenter sont proches les unes des autres, cette pratique est utile. Si par contre les valeurs sont très différentes, c'est un peu superflu car les différences de hauteur clarifieront automatiquement la situation. Mais comme toujours, le programmeur doit prendre en compte le cas extrême. En l'occurrence, le risque d'un manque de clarté n'est guère admissible. Autant le prendre en considération.

Les fonctionnalités de la routine se précisent. A partir d'un tableau de valeurs, il faudra tracer N colonnes de L points de large, espacées par deux points, et utilisant un axe des Y réduit grâce à un calcul d'échelle.

Calcul d'échelle

Si nous traçons nos colonnes directement à partir des valeurs, le programme ne pourra traiter que des valeurs situées entre 20 et 199. Bien entendu, il est hors de question d'accepter cette contrainte. Il faut un calcul d'échelle qui permette de tracer des valeurs grandes ou petites, positives ou non.

Nous avons déjà mis en garde le lecteur contre les nombres réels, appelés aussi flottants. Nous allons donc nous répéter. En effet, le comble du luxe serait de pouvoir soumettre au programme des nombres à virgule. Malheureusement, plusieurs problèmes s'y opposent. Tout d'abord, le traitement des nombres flottants est lent et compliqué. Le codage d'une valeur ne tient plus dans un seul registre du processeur. De plus, contrairement aux autres routines système de l'Amstrad, les vecteurs des routines mathématiques associées ne sont pas situés aux mêmes adresses sur les différents modèles de l'ordinateur. Cette difficulté peut être contournée lors de l'assemblage, mais elle complique de toute façon la programmation d'une routine compatible avec tous les Amstrad.

La seule contrainte, peut-être gênante de la routine, sera donc la suivante : les valeurs devront être au format entier, ce qui signifie tout de même qu'il est possible de traiter les nombres entiers de -32768 à 32767.

Comment procéder toutefois pour représenter par exemple 10 nombres échelonnés entre 0 et 1 ? Nous avons déjà croisé la solution lors du tracé de cercles. Une valeur comprise entre 0 et 1 peut facilement être multipliée par 100 (ou toute autre valeur assez grande transformant les décimales en nombre entier significatif) pour donner une valeur comprise entre 0 et 100.

Il sera donc à la charge du programme appelant de prévoir le cas de valeurs situées dans un petit intervalle proche de zéro. Il suffira de multiplier toutes les valeurs (par exemple par 100) et de les transformer en entiers avant de les soumettre au traceur d'histogrammes.

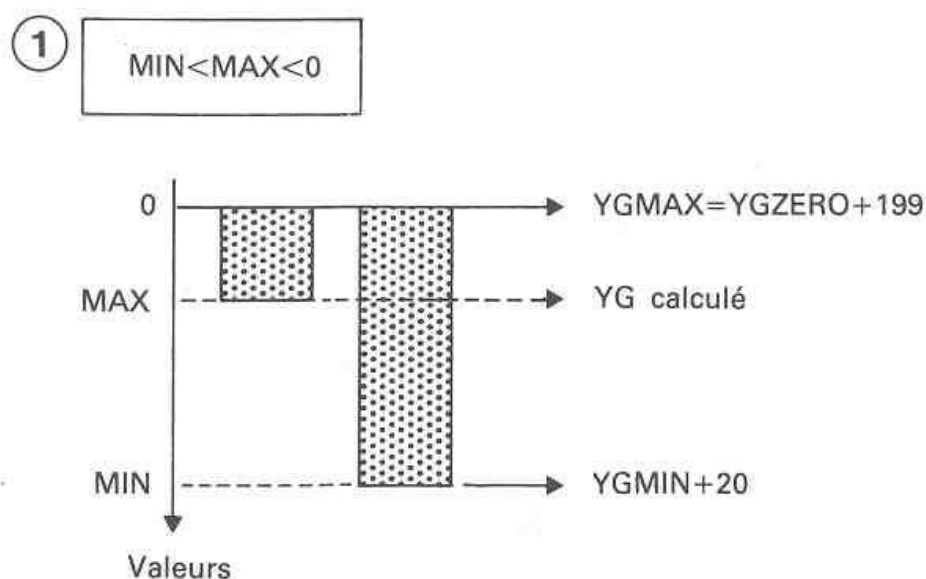
Les choses se précisent encore. Mais il reste un point obscur non négligeable. Comment la routine va-t-elle représenter un tableau de valeurs situées entre 0 et 100, si elle doit aussi être capable de représenter -32000 et +32000 sur le même écran, tout cela en 180 points seulement de hauteur ?

Ce problème donne déjà le premier travail de la routine. En parcourant le tableau de valeurs, elle devra repérer le maximum et le minimum de façon à pouvoir calculer l'échelle de tracé vertical (à savoir le nombre d'unités représentées par un seul point graphique de hauteur).

Malheureusement, une difficulté surgit. Si nous avons des valeurs positives et négatives, il nous faudra calculer la position verticale de l'axe des X, c'est-à-dire la position graphique verticale associée à la valeur zéro. Dans ce cas, la valeur minimale sera associée à l'ordonnée 20 (le bas de l'écran en ce qui concerne la fenêtre fixée par convention), et la plus grande valeur à l'ordonnée 199 (le haut de l'écran). Mais qu'advient-il si les valeurs sont toutes plus grandes que zéro ou toutes négatives ? En effet, une autre convention dans les histogrammes est de toujours représenter la valeur zéro. Si les valeurs se trouvent entre 30000 et 32000, il faudra tout de même représenter le zéro en bas de l'écran, alors que le calcul du minimum nous affirmera "32000 en bas de l'écran".

Dans le cas de nombres négatifs, le zéro sera situé en haut de l'écran. Tout cela se résume en trois cas. Pour poursuivre l'étude, il faut adopter quelques notations. MIN et MAX représenteront les valeurs extrêmes du tableau. YGMIN, YGMAX et YGZERO seront les ordonnées graphiques (entre 20 et 199) associées respectivement à MIN, MAX et 0.

Les trois cas étudiés se résument simplement (*schéma 3.9*).



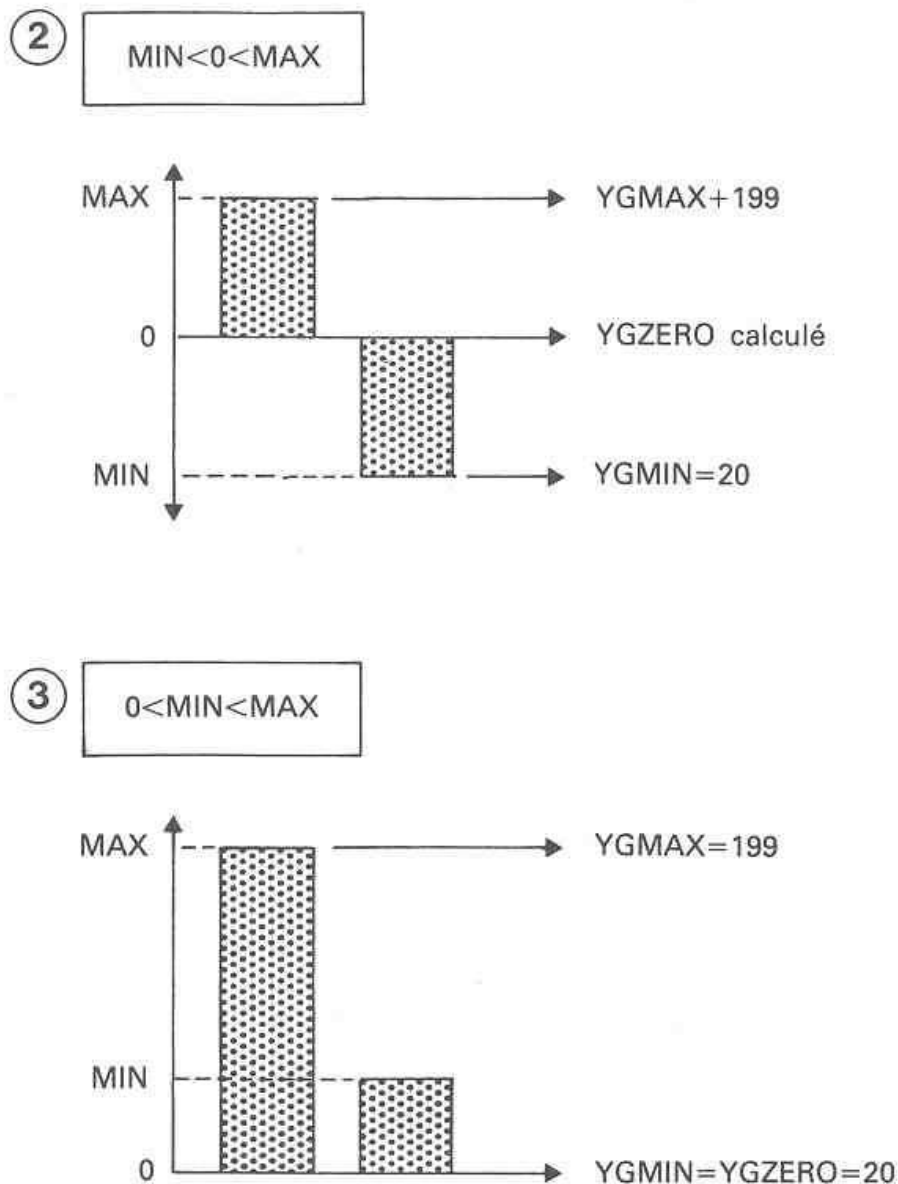


Schéma 3.9

Les trois cas de tracés

- Si $\text{MIN} < \text{MAX} < 0$: toutes les valeurs sont négatives. $\text{YGMIN}=20$, et $\text{YGMAX}=\text{YGZERO}=199$, tout en haut de l'écran.
- Si $\text{MIN} < 0 < \text{MAX}$: il y a des nombres positifs et des négatifs. $\text{YGMIN}=20$, $\text{YGMAX}=199$, et YGZERO doit être calculé. Une règle de trois donne la valeur $\text{YGZERO} = -\text{MIN}/(\text{MAX}-\text{MIN}) * 180 + \text{YGMIN}$.
- si $0 < \text{MIN} < \text{MAX}$: toutes les valeurs sont positives. $\text{YGMIN}=\text{YGZERO}=20$, et $\text{YGMAX}=199$.

Le cas $\text{MAX} < 0 < \text{MIN}$ est évidemment impossible. L'algorithme du début de routine apparaît maintenant :

- calculer MIN et MAX par examen des valeurs du tableau
- $\text{YGMIN}=20$
- $\text{YGMAX}=199$

- si $MIN > 0$, $YGZERO = 20$
 Sinon, si $MAX < 0$, $YGZERO = 199$
 Sinon, $YGZERO = -MIN / (MAX - MIN) * 180 + YGMIN$

Le tracé des valeurs est un peu plus complexe, mais tout est relatif. Il faut néanmoins distinguer le cas des valeurs négatives et positives (schéma 3.10).

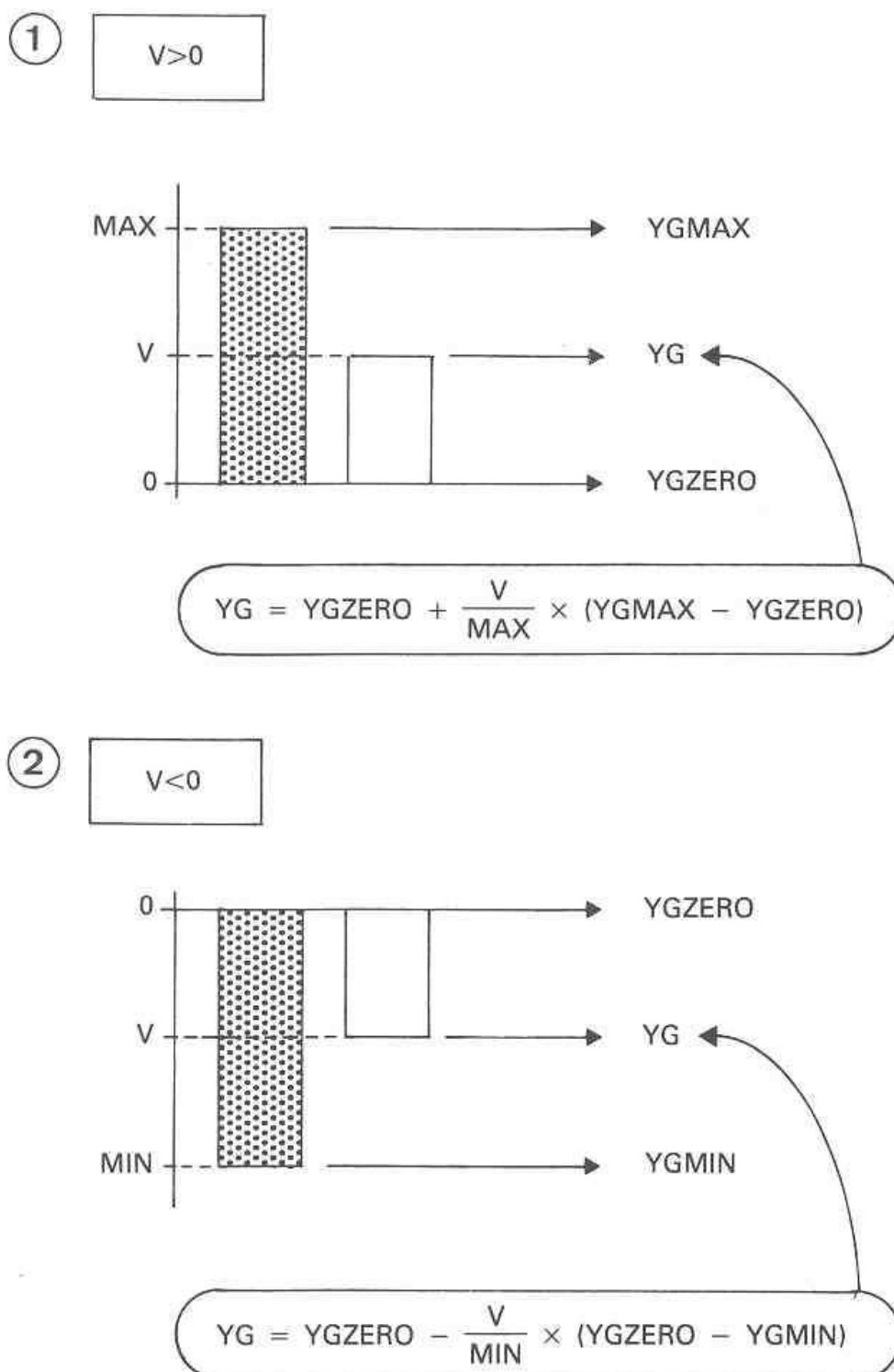


Schéma 3.10

Tracé des valeurs suivant signe.

Pour tracer la colonne associée à une valeur, il faut calculer l'ordonnée graphique qui lui est associée. Cette ordonnée est appelée YG. Le calcul de YG diffère selon le signe de la valeur V en question :

- si V est positive, $YG = YGZERO + V / MAX * (YGMAX - YGZERO)$
- si V est négative, $YG = YGZERO - (-V) / (-MIN) * (YGZERO - YGMIN)$

Multiplication et division en assembleur

Si le reste semble simple, quelques obstacles se dressent encore, dus à la programmation en langage machine. En effet, la routine aura besoin de multiplication et de division. Mais il est question ici de nombres entiers (d'où un problème pour la division). La division est plus complexe que la multiplication. Il est hors de question ici de programmer une division par rotation et décalage. Le mécanisme d'une division est trop éloigné des opérations binaires pour cela. Le programme se contentera d'un algorithme simple qui a fait ses preuves : retrancher le diviseur au dividende jusqu'à obtenir un reste inférieur au diviseur. Cela constitue bien sûr une division entière : il n'est pas question de nombres à virgule.

Malheureusement pour la rapidité du programme, cette contrainte entraîne une impossibilité d'optimisation importante. En effet, les seules divisions à effectuer sont $-MIN/(MAX-MIN)$, $(YGZERO-YGMIN)/(-MIN)$ et $(YGMAX-YGZERO)/MAX$. Or, si le premier calcul ne pose pas de problème évident, il en est autrement des deux autres. Le dividende se situera toujours entre 0 et 180, mais il se peut que le diviseur (MAX ou -MIN) lui soit inférieur. Dans ce cas, puisqu'il est impensable de traiter les décimales du résultat, celui-ci sera zéro. Il s'ensuivra que toutes les valeurs seront assimilées à zéro.

Or, le calcul de YG pour une valeur donnée nécessite ces valeurs. S'il avait été possible de les calculer avant le tracé, de les transformer en constantes pour réduire les calculs de tracé (suppression de la division), la vitesse de tracé aurait été sensiblement améliorée.

Pourtant, il faut se rendre à l'évidence : il est impossible d'optimiser ainsi. Les divisions devront obligatoirement être effectuées pour chaque tracé, car le problème évoqué ci-dessus n'est pas le seul.

La division entière est génératrice d'une erreur de calcul non négligeable. En théorie, $16 \times 10 / 3$ donne le réel 53.33, soit 53 si nous assimilons la division à une division entière. L'erreur de calcul de 0.33 n'est pas visible au niveau des nombres entiers. Par contre, si nous inversons l'ordre des opérations, $16 / 3 \times 10$ qui devrait être identique à la première opération donne le résultat 5.33×10 qui devient 5×10 , soit 50 si la division est entière. Et cette fois-ci, l'erreur est visible : 50 au lieu de 53. Car si l'erreur de calcul provoquée par la division entière est toujours de 0.33 et négligeable, nous l'avons multipliée par dix. Elle est devenue une erreur de 3.33.

La conclusion de cet épisode est simple : quel que soit le calcul à effectuer, la division devra être placée tout au bout des opérations, de façon à obtenir une erreur de calcul toujours inférieure à 1.

On peut compenser cette perte. La division entière donne un quotient et un reste. Il est possible de multiplier ce reste (qui, lui, est entier et inférieur au diviseur) par le diviseur et d'ajouter le nombre ainsi obtenu après la multiplication du quotient. Ce principe donne bien 53 pour notre exemple de petite opération. En effet, $16/3$ donne 5, mais nous retenons un reste de 1, qui donne 3 une fois multiplié par le diviseur. Ensuite, nous l'ajoutons à 5×10 , ce qui conduit à un résultat correct.

Mais il s'agit bien d'un palliatif. Il suffit de remplacer $10 \times 16/3$ par $16 \times 10/3$ pour le constater. La division donne un quotient de 3 et un reste de 1. Multiplié par le diviseur, le reste devient 3. Si nous multiplions, selon la méthode décrite, 16 par le quotient en ajoutant ensuite ce reste compensé, nous obtenons $48 + 3 = 51$, ce qui est mieux que 50, mais toujours faux.

Si nous adoptons cette solution dans le programme, la division effectuée pour chaque valeur sera transformée en addition, et il n'y aura qu'une simple division réalisée avant le tracé des colonnes. Mais la promesse de rapidité qui en découle ne doit pas être prise en compte. En effet, le but d'un traceur d'histogrammes est d'être fidèle. Et cette méthode du reste compensé, valable dans certains cas, ne l'est plus pour cette simple raison. Le risque d'erreur de calcul, et donc d'erreur dans la représentation des valeurs, est éliminatoire. L'histogramme est censé donner une image de la réalité, et doit donc respecter les données fournies, les traiter avec un maximum de précision.

La division aura donc lieu au cours de chaque calcul de YG. Cela nous pose toutefois un dernier problème jusqu'alors ignoré. Puisque nous effectuerons la multiplication de V par $(YGMAX - YGZERO)$ avant la division, nous risquons ce qu'on appelle en langage technique un Overflow, ou encore un dépassement de capacité. La multiplication que nous avons créée pour le tracé de cercles permet de multiplier deux nombres 8 bits, et d'obtenir fort logiquement un nombre 16 bits. Ici, V est un nombre 16 bits et $(YGMAX - YGMIN)$ un nombre 8 bits. Nous allons donc obtenir, dans le pire des cas, un nombre 24 bits, qui ne tiendra jamais dans un des registres du processeur (*schéma 3.11 v. p. 89*).

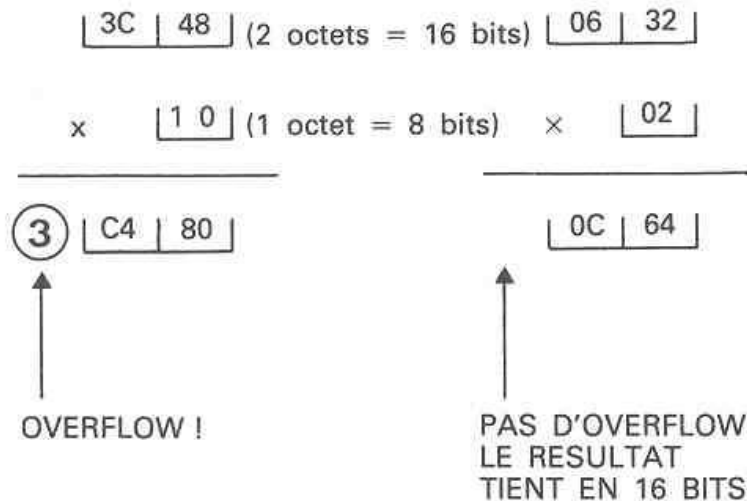


Schéma 3.11

Overflow multiplication.

La solution existe, bien sûr. Elle consiste à ne pas utiliser, pour le stockage des résultats intermédiaires et du résultat total, un registre 16 bits du processeur, mais 3 octets successifs de la mémoire. En traitant ces 3 octets comme des registres 8 bits du processeur, nous obtiendrons un équivalent facilement manipulable (schéma 3.12).

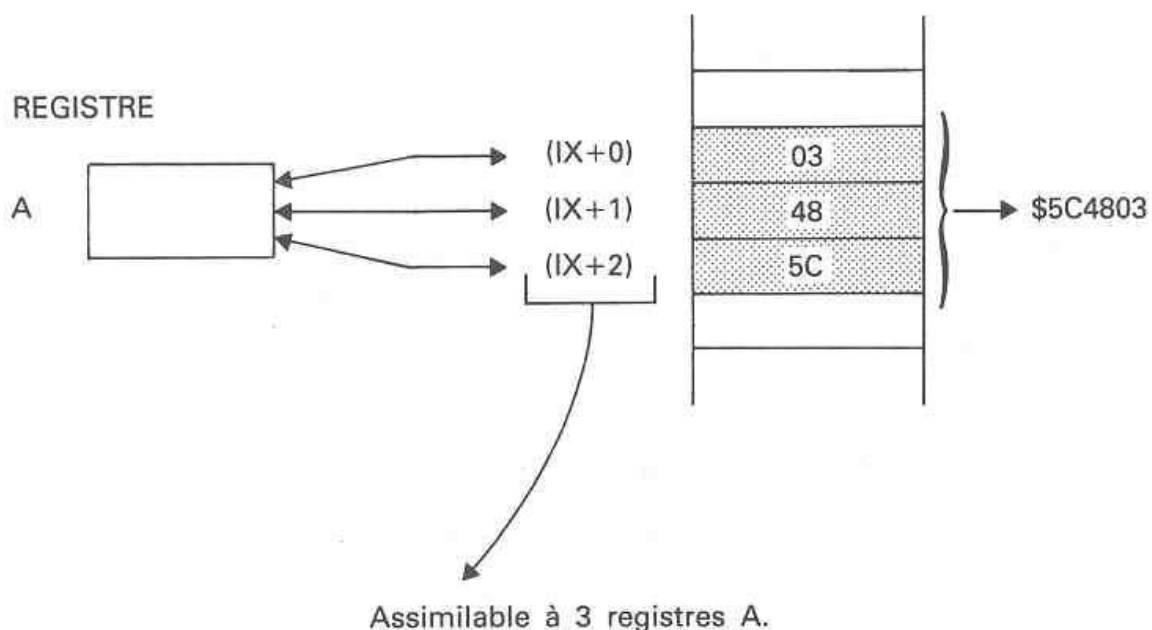


Schéma 3.12

Accumulateur 24 bits.

D'autre part, ce résultat devra être divisé. Là aussi, impossible d'utiliser les registres 16 bits du Z-80 pour soustraire le diviseur (qui est de 16 bits). En réalité, cela sera possible dès que le reste descendra en dessous de

17 bits. La division aura donc lieu en deux étapes : d'abord avec la zone de 3 octets et un registre 16 bits tant que le poids fort du dividende est non nul, puis avec deux registres 16 bits.

Programmation de la méthode

Avant de poursuivre l'étude de la programmation en langage machine, il serait sage de tester l'algorithme en Basic. C'est l'objet du programme 3.4.

```

10 '*****
20 '**  programme 3.4 **
30 '*****
40 '
50 'Trace d'histogrammes en Basic
60 'd'apres un tableau de valeurs aleatoire.
70 '
80 DEFINT a-z: DIM tablo(30): larg=23: nbval=10
90 FOR i=1 TO nbval: tablo(i)=RND*300-150: NEXT
100 MODE 1: FOR i=1 TO 3: INK i, 8*i: NEXT
110 nmax=tablo(1): nmin=nmax
120 FOR i=1 TO nbval: IF nmin>tablo(i) THEN nmin=
    tablo(i): GOTO 140
130 IF nmax<tablo(i) THEN nmax=tablo(i)
140 NEXT
150 ygmin=20: ygmax=199
160 IF nmax<0 THEN ygzero=199: GOTO 210
170 IF nmin>0 THEN ygzero=20: GOTO 210
180 ygzero=ABS(nmin)/(nmax-nmin)*(199-20)
190 MOVE 39*2, ygzero*2: DRAW 639, ygzero*2, 3
200 MOVE 39*2, 40: DRAW 39*2, 399, 3
210 absi=40: inkk=1: FOR i=1 TO nbval
220   IF tablo(i)>0 THEN y1=ygzero: y2=ygzero+tab
    lo(i)/nmax*(ygmax-ygzero) ELSE y1=ygzero-tabl
    o(i)/nmin*(ygzero-ygmin): y2=ygzero
230   FOR x=absi TO absi+larg
240     MOVE x*2, y1*2: DRAW x*2, y2*2, inkk
250   NEXT
260   absi=absi+larg+1: inkk=inkk+1: IF inkk=4 THE
    N inkk=1
270 NEXT
280 LOCATE 1, 1: END

```

Il correspond trait pour trait à ce que nous avons déjà défini. Il trace l'histogramme d'un nombre aléatoire de valeurs... aléatoires elles aussi. On constate que la méthode est bonne. Le tracé des colonnes, bien que réalisé en Basic, reste suffisamment rapide.

Le programme 3.5 est la traduction en langage assembleur de notre algorithme. Son fonctionnement est limpide, il respecte les opérations définies au cours de l'étude. Voyons son organisation en détail.

```

10 ;
20 ;Programme de trace d'histogrammes
30 ;en mode 1 (programme 3.5)
40 ;
50 ;Syntaxe: CALL &4200,@tableau(1),nombre de valeurs,largeur
60 ;
70 ;routines systemes utilisees :
BC62 80 VERLIN: EQU #BC62 ;trace une ligne verticale
BC5F 90 HORLIN: EQU #BC5F ;trace une ligne horizontale
BC2C 100 INKCOD: EQU #BC2C ;codage du numero d'encre
BC0E 110 MODE: EQU #BC0E ;changement de mode
120 ;
4200 130 ORG #4200
140 ;
4200 DD7E00 150 BASIC: LD A,(IX+0)
4203 324F44 160 LD (LARG),A ;memorise largeur des colonnes
4206 DD7E02 170 LD A,(IX+2)
4209 325044 180 LD (NBVAL),A ;memorise le nombre de colonnes
420C DD6E04 190 LD L,(IX+4)
420F DD6605 200 LD H,(IX+5)
4212 225144 210 LD (TABLO),HL ;memorise l'adresse des valeurs
220 ;
4215 3A5044 230 LM: LD A,(NBVAL) ;A utilise comme compteur
4218 FE02 240 CP 2 ;moins de 2 valeurs ?
421A D8 250 RET C ;oui: ne rien faire
421B 2A5144 260 LD HL,(TABLO) ;HL pointe les valeurs
421E 4E 270 LD C,(HL)
421F 23 280 INC HL
4220 46 290 LD B,(HL) ;BC contient la premiere valeur
4221 ED435344 300 LD (MAX),BC
4225 ED435544 310 LD (MIN),BC ;extremas initialises.
4229 23 320 INC HL ;on ignore la premiere valeur
422A 3D 330 DEC A ;et on compte une valeur en moins
340 ;
350 ;recherche des extremas
360 ;
422B 4E 370 LOOP1: LD C,(HL)
422C 23 380 INC HL
422D 46 390 LD B,(HL)
422E 23 400 INC HL ;la valeur est dans BC
422F ED5B5344 410 LD DE,(MAX)
4233 CD8D43 420 CALL CPDEBC ;compare DE et BC signes
4236 D24042 430 JP NC,PAMAX ;maximum >= nombre
4239 ED435344 440 LD (MAX),BC ;nouveau maximum
423D C34E42 450 JP SUITE1 ;suite du travail
460 ;

```

```

4240 ED5B5544 470 PAMAX: LD DE,(MIN)
4244 CD8D43 480 CALL CPDEBC ;compare DE et BC
4247 DA4E42 490 JP C,SUITE1 ;minimum < nombre
424A ED435544 500 LD (MIN),BC ;nouveau minimum
510 ;
424E 3D 520 SUITE1: DEC A ;une valeur en moins
424F C22B42 530 JP NZ,LOOP1 ;suite de la boucle
540 ;
550 ;calcul des YG suivant les trois cas
560 ;
4252 211400 570 LD HL,20
4255 225744 580 LD (YGMIN),HL
4258 21C700 590 LD HL,199
425B 225B44 600 LD (YGMAX),HL
425E 2A5544 610 LD HL,(MIN)
4261 CB7C 620 BIT 7,H ;MIN est-il positif ?
4263 C26F42 630 JP NZ,CAS2 ;non: etudier max
4266 211400 640 LD HL,20 ;Min est positif,
4269 225944 650 LD (YGZERO),HL ;ygzero est en bas
426C C3A142 660 JP CALCUL ;ok
670 ;
426F 2A5344 680 CAS2: LD HL,(MAX)
4272 CB7C 690 BIT 7,H ;MAX est-il negatif ?
4274 CA8042 700 JP Z,CAS3 ;non:cas normal
4277 21C700 710 LD HL,199 ;Max est negatif,
427A 225944 720 LD (YGZERO),HL ;ygzero est en haut
427D C3A142 730 JP CALCUL ;ok
740 ;
4280 ED5B5544 750 CAS3: LD DE,(MIN)
4284 CD4244 760 CALL VALABS ;valeur absolue de DE
4287 3EB4 770 LD A,180
4289 CDB243 780 CALL MULTIP ;!min!*180, resultat (TEMPO1)
428C 2A5344 790 LD HL,(MAX)
428F ED5B5544 800 LD DE,(MIN)
4293 B7 810 OR A
4294 ED52 820 SBC HL,DE ;HL=(max-min)
4296 EB 830 EX DE,HL
4297 CD0744 840 CALL DIV ;divise !min!*180 par DE=(max-min)
429A 211400 850 LD HL,20
429D 09 860 ADD HL,BC ;ajoute offset vertical
429E 225944 870 LD (YGZERO),HL ;fin du calcul de YGZero.
880 ;
890 ;calculs divers pour faciliter tracer rapide
900 ;
42A1 ED5B5544 910 CALCUL: LD DE,(MIN)
42A5 CD4244 920 CALL VALABS
42A8 ED535544 930 LD (MIN),DE ;remplace min par sa valeur absolue
42AC 2A5B44 940 LD HL,(YGMAX)
42AF ED4B5944 950 LD BC,(YGZERO)
42B3 B7 960 OR A
42B4 ED42 970 SBC HL,BC
42B6 7D 980 LD A,L
42B7 326144 990 LD (DPOS),A ;pour >0: (ygmax-ygzero)

```

```

42BA 2A5944 1000 LD HL,(YGZERO)
42BD ED4B5744 1010 LD BC,(YGMIN)
42C1 B7 1020 OR A
42C2 ED42 1030 SBC HL,BC
42C4 7D 1040 LD A,L
42C5 326244 1050 LD (DNEG),A ;pour<0: (ygzero-ygmin)
1060 ;
1070 ;On attaque le trace. D'abord initialisations
1080 ;
42C8 3E01 1090 LD A,1
42CA 326344 1100 LD (INK),A ;encre de tracer de depart
42CD 213200 1110 LD HL,50
42D0 226444 1120 LD (X),HL ;abscisse de depart
42D3 3E01 1130 LD A,1
42D5 CD2CBC 1140 CALL INKCOD
42D8 113200 1150 LD DE,50
42DB 2A5744 1160 LD HL,(YGMIN)
42DE ED4B5B44 1170 LD BC,(YGMAX)
42E2 CD62BC 1180 CALL VERLIN ;axe vertical
42E5 3E01 1190 LD A,1
42E7 CD2CBC 1200 CALL INKCOD
42EA 113200 1210 LD DE,50
42ED 013F01 1220 LD BC,319
42F0 2A5944 1230 LD HL,(YGZERO)
42F3 CD5FBC 1240 CALL HORLIN ;axe horizontal
42F6 3A5044 1250 LD A,(NBVAL)
42F9 47 1260 LD B,A ;compteur du nombre de valeurs
1270 ;
42FA C5 1280 LOOPG: PUSH BC ;sauvegarde compteur valeurs
42FB 2A5144 1290 LD HL,(TABLO)
42FE 5E 1300 LD E,(HL)
42FF 23 1310 INC HL
4300 56 1320 LD D,(HL) ;DE contient la valeur
4301 23 1330 INC HL
4302 225144 1340 LD (TABLO),HL ;mise a jour indice tableau
4305 CB7A 1350 BIT 7,D ;negatif ?
4307 C22743 1360 JP NZ,MOINS ;oui, trace en consequence
1370 ;
1380 ;trace pour un nombre positif
1390 ;
430A 3A6144 1400 LD A,(DPOS) ;ygmax-ygzero
430D CDB243 1410 CALL MULTIP ;valeur*(ygmax-ygzero)
4310 ED5B5344 1420 LD DE,(MAX)
4314 CD0744 1430 CALL DIV ;division par max
4317 2A5944 1440 LD HL,(YGZERO)
431A 09 1450 ADD HL,BC
431B 225D44 1460 LD (YGHUT),HL
431E 2A5944 1470 LD HL,(YGZERO)
4321 225F44 1480 LD (YGBAS),HL
4324 C34643 1490 JP SUIBOU ;suite de la boucle
1500 ;
1510 ;trace pour un nombre negatif
1520 ;

```



```

4327 CD4244 1530 MOINS: CALL VALABS
432A 3A6244 1540 LD A,(DNEG) ;(ygzero-ygmin)
432D CDB243 1550 CALL MULTIP ;!valeur! *(ygzero-ygmin)
4330 ED5B5544 1560 LD DE,(MIN) ;rappel: min=valeur absolue
4334 CD0744 1570 CALL DIV ;division par !min!
4337 2A5944 1580 LD HL,(YGZERO)
433A B7 1590 OR A ;carry mis a zero
433B ED42 1600 SBC HL,BC
433D 225F44 1610 LD (YGBAS),HL
4340 2A5944 1620 LD HL,(YGZERO)
4343 225D44 1630 LD (YGHUT),HL
1640 ;
1650 ;avance dans la boucle
1660 ;
4346 CD6D43 1670 SUIBOU: CALL TRACER
4349 2A6444 1680 LD HL,(X) ;pour mise a jour X
434C 3A4F44 1690 LD A,(LARG) ;on ajoute largeur colonne
434F 4F 1700 LD C,A
4350 0600 1710 LD B,0
4352 09 1720 ADD HL,BC
4353 23 1730 INC HL
4354 23 1740 INC HL ;et un petit espace
4355 226444 1750 LD (X),HL
4358 3A6344 1760 LD A,(INK) ;pour changer de couleur
435B 3C 1770 INC A
435C 326344 1780 LD (INK),A
435F FE04 1790 CP 4 ;couleur=4 ?
4361 DA6943 1800 JP C,OK ;non: pas de probleme
4364 3E01 1810 LD A,1
4366 326344 1820 LD (INK),A ;revenir en couleur 1
1830 ;
4369 C1 1840 OK: POP BC
436A 108E 1850 DJNZ LOOPG ;fin du tracer
1860 ;
436C C9 1870 RET
1880 ;
1890 ;----- routines -----
1900 ;
436D 3A4F44 1910 TRACER: LD A,(LARG) ;nombre de ligne par colonne
4370 47 1920 LD B,A
4371 ED5B6444 1930 LD DE,(X) ;abscisse de depart
1940 ;
4375 C5 1950 LOOPT: PUSH BC ;sauvegarde compteur
4376 D5 1960 PUSH DE ;sauvegarde abscisse
4377 3A6344 1970 LD A,(INK)
437A CD2CBC 1980 CALL INKCOD
437D 2A5F44 1990 LD HL,(YGBAS)
4380 ED4B5D44 2000 LD BC,(YGHUT)
4384 CD62BC 2010 CALL VERLIN
4387 D1 2020 POP DE
4388 13 2030 INC DE ;avancer vers la droite
4389 C1 2040 POP BC
438A 10E9 2050 DJNZ LOOPT

```

```

438C C9      2060      RET
              2070 ;
              2080 ; Comparaison DE et BC
              2090 ; retour : Carry=1 si DE<BC, 0 si DE>=BC
              2100 ; tous registres gardes.
              2110 ;

438D D5      2120 CPDEBC: PUSH DE
438E E5      2130      PUSH HL
438F C5      2140      PUSH BC                ;pour travailler tranquillement
4390 CB78    2150      BIT 7,B                ;BC positif ?
4392 CAA343  2160      JP 7,BCPOS              ;oui, traitement a part
              2170 ;
              2180 ; BC est negatif. et DE ?
              2190 ;

4395 B7      2200      OR A                    ;carry=0 par default
4396 CB7A    2210      BIT 7,D                ;DE positif ?
4398 CA4E43  2220      JP 7,SUITE2              ;oui: donc DE>BC, carry=0
              2230 ; deux nombres negatifs

439B 62      2240      LD H,D
439C 6B      2250      LD L,E                ;transfere DE dans HL
439D B7      2260      OR A                    ;carry a zero
439E ED42    2270      SBC HL,BC              ;soustraction->carry ou non
43A0 C3AE43  2280      JP SUITE2
              2290 ;
              2300 ; BC est positif. Et DE ?
              2310 ;

43A3 37      2320 BCPOS: SCF                  ;carry=1 par default
43A4 CB7A    2330      BIT 7,D                ;DE negatif ?
43A6 C2AE43  2340      JP NZ,SUITE2              ;oui, donc DE<BC, carry=1
              2350 ; deux nombres positifs

43A9 62      2360      LD H,D
43AA 6B      2370      LD L,E
43AB B7      2380      OR A                    ;carry=0
43AC ED42    2390      SBC HL,BC              ;Carry si DE<BC
              2400 ;

43AE C1      2410 SUITE2: POP BC
43AF E1      2420      POP HL
43B0 D1      2430      POP DE
43B1 C9      2440      RET
              2450 ;
              2460 ; Multiplication de DE par A dans buffer TEMP01
              2470 ;

43B2 DD216644 2480 MULTIP: LD IX,TEMP01
43B6 FD216944 2490      LD IY,TEMP02
43BA C5      2500      PUSH BC                ;sauvegarde BC
43BB 0608    2510      LD B,B                ;nombre de bits dans A
43BD DD360000 2520      LD (IX+0),0
43C1 DD360100 2530      LD (IX+1),0
43C5 DD360200 2540      LD (IX+2),0          ;mise a zero du resultat
43C9 FD7300  2550      LD (IY+0),E
43CC FD7201  2560      LD (IY+1),D
43CF FD360200 2570      LD (IY+2),D          ;resultat intermediaire

```

43D3	CB2F	2580	SRA	A	;premier decalage
43D5	D2F543	2590	POIDS:	JP NC,SUIV	;pas d'addition intermediaire
43D8	F5	2600	PUSH	AF	;sauvegarde multiplicateur
43D9	FD7E00	2610	LD	A,(IX+0)	;addition du resultat intermediaire
43DC	DD8600	2620	ADD	A,(IX+0)	
43DF	DD7700	2630	LD	(IX+0),A	
43E2	FD7E01	2640	LD	A,(IX+1)	
43E5	DD8E01	2650	ADC	A,(IX+1)	
43E8	DD7701	2660	LD	(IX+1),A	
43EB	FD7E02	2670	LD	A,(IX+2)	
43EE	DD8E02	2680	ADC	A,(IX+2)	
43F1	DD7702	2690	LD	(IX+2),A	
43F4	F1	2700	POP	AF	;recupere multiplicateur
		2710 ;			
43F5	FDCB0026	2720	SUIV:	SLA (IX+0)	;decalage resultat interm.
43F9	FDCB0116	2730	RL	(IX+1)	
43FD	FDCB0216	2740	RL	(IX+2)	
4401	CB2F	2750	SRA	A	;bit suivant A
4403	10D0	2760	DJNZ	POIDS	;huit fois seulement
4405	C1	2770	POP	BC	;recupere BC
4406	C9	2780	RET		
		2790 ;			
		2800	;division du buffer TEMPO1 par DE dans BC		
		2810 ;			
4407	DD216644	2820	DIV:	LD IX,TEMPO1	
440B	010000	2830	LD	BC,0	;mise a zero resultat
440E	B7	2840	OR	A	;mise a zero carry
		2850 ;			
440F	DD7E02	2860	LOOPS:	LD A,(IX+2)	;poids fort nombre
4412	B7	2870	OR	A	;zero ?
4413	CA2F44	2880	JP	Z,SUB16	;oui, soustraction 16 bits normale
		2890	;soustraction 24 bits		
4416	DD7E00	2900	LD	A,(IX+0)	
4419	93	2910	SUB	E	
441A	DD7700	2920	LD	(IX+0),A	
441D	DD7E01	2930	LD	A,(IX+1)	
4420	9A	2940	SBC	A,D	
4421	DD7701	2950	LD	(IX+1),A	
4424	DD7E02	2960	LD	A,(IX+2)	
4427	DE00	2970	SBC	A,0	
4429	DD7702	2980	LD	(IX+2),A	
442C	C33D44	2990	JP	TESTCA	;test sur le carry pour fin
		3000 ;			
442F	DD7E00	3010	SUB16:	LD A,(IX+0)	
4432	93	3020	SUB	E	
4433	DD7700	3030	LD	(IX+0),A	
4436	DD7E01	3040	LD	A,(IX+1)	
4439	9A	3050	SBC	A,D	
443A	DD7701	3060	LD	(IX+1),A	
		3070 ;			
443D	D8	3080	TESTCA:	RET C	;fin du travail si Carry.
443E	03	3090	INC	BC	;sinon une unite de plus
443F	C30F44	3100	JP	LOOPS	;et continuer soustraction.

```

3110 ;
3120 ;DE devient sa valeur absolue
3130 ;
4442 CB7A 3140 VALABS: BIT 7,D          ;DE negatif ?
4444 C8    3150      RET Z            ;non:pas de travail
4445 F5    3160      PUSH AF          ;sauvegarde A
4446 7B    3170      LD A,E
4447 2F    3180      CPL
4448 5F    3190      LD E,A
4449 7A    3200      LD A,D
444A 2F    3210      CPL
444B 57    3220      LD D,A          ;complement a un
444C 13    3230      INC DE
444D F1    3240      POP AF
444E C9    3250      RET
3260 ;
3270 ;----- zone des variables -----
3280 ;
444F 00    3290 LARG:  DEFB 0          ;largeur des colonnes
4450 00    3300 NBVAL: DEFB 0          ;nombre de valeurs
4451 0000  3310 TABLO: DEFW 0          ;adresse du tableau
4453 0000  3320 MAX:  DEFW 0
4455 0000  3330 MIN:  DEFW 0
4457 0000  3340 YGMIN: DEFW 0
4459 0000  3350 YGZERO: DEFW 0
445B 0000  3360 YGMAX: DEFW 0
445D 0000  3370 YGHAUT: DEFW 0
445F 0000  3380 YGBAS: DEFW 0
4461 00    3390 DPOS:  DEFB 0
4462 00    3400 DNEG:  DEFB 0
4463 00    3410 INK:   DEFB 0
4464 0000  3420 X:     DEFW 0
4466 000000 3430 TEMP01: DEFB 0,0,0    ;buffer de calcul 3 octets
4469 000000 3440 TEMP02: DEFB 0,0,0    ;idem

```

Pass 2 errors: 00

L'exécution débute avec le bloc des lignes 150 à 210. Ces instructions sont chargées de récupérer les paramètres du CALL Basic, et de les placer dans les variables du programme. NBVAL est le nombre de valeurs à tracer, TABLO pointe sur la première de ces valeurs 16 bits, et LARG est la largeur de la colonne tracée, en nombre de points graphiques. Remarquez que NBVAL et LARG sont des valeurs 8 bits : seul leur poids faible est mémorisé. Il est en effet inutile de mémoriser le poids fort. Il faudrait le faire s'il était possible de placer plus de 255 valeurs à l'écran ou de tracer des colonnes de plus de 255 points de large. En l'occurrence, cela est inimaginable, car l'écran fixé ne possède que 290 points de large, et deux points séparent chaque colonne.

Puis, le programme proprement dit commence. Un appel à l'adresse \$4215 (ligne 230 du programme), en langage machine, exécute le tracé en prenant pour données les valeurs des variables vues ci-dessus. On peut donc parfaitement utiliser ce programme en langage machine. Il suffit de placer les données dans les trois variables TABLO, NBVAL et LARG, puis d'effectuer un CALL \$4215.

Avant toute chose, il faut vérifier que l'appelant a demandé le tracé d'un tableau à plusieurs valeurs. En effet, si NBVAL vaut 0 ou 1, le tracé est sans intérêt. Le RET C de la ligne 250 achève l'exécution si $NBVAL < 2$.

Ensuite a lieu la recherche des extrêmes. Ceux-ci sont stockés dans les variables MAX et MIN. On les initialise avec la première valeur du tableau, puis on boucle NBVAL-1 fois à partir de la seconde valeur pour repérer les valeurs extrêmes.

L'instruction CALL CPDEBC est très importante. Elle compare le contenu 16 bits des registres DE et BC et positionne le flag Carry, en conséquence : Carry=1 si $DE < BC$, Carry=0 si $DE \geq BC$. Ce serait facile à programmer s'il s'agissait de nombres de signe positif. En effet, dans ce cas, savoir si $DE < BC$ revient à savoir si $DE - BC < 0$. En soustrayant BC à DE, le Carry serait positionné comme voulu.

Mais hélas, les valeurs à comparer sont considérées comme des nombres signés. Cela signifie que les valeurs \$0000 à \$7FFF sont positives, et \$8000 à \$FFFF négatives. Un problème risque de se poser dans le cas où les nombres sont de signe opposé : il faudra donc procéder autrement. Nous verrons la routine CPDEBC plus loin.

Les extrêmes sont trouvés simplement en comparant chaque valeur à MIN et MAX. Si la valeur est inférieure à MIN (en tenant compte du signe, c'est-à-dire que -18 est inférieur à -3), elle remplace ce dernier. De même, si une valeur est supérieure à MAX, elle le remplace. Et la boucle continue avec la valeur suivante, jusqu'à ce que toutes les valeurs aient été examinées. La fin de la boucle est en ligne 540.

Ensuite, le programme assigne et calcule les valeurs de YGMIN, YGMAX et YGZERO. Il suit, pour ce faire, les calculs vus plus haut. Pour le calcul de YGZERO, il distingue les trois cas rencontrés lors de l'étude :

- si $MIN > 0$, $YGZERO = 199$;
- si $MAX < 0$, $YGZERO = 20$;
- sinon, $YGZERO = (-MIN)/(MAX-MIN)*180 + YGMIN$.

Le sous-programme VALABS utilisé à ce sujet en ligne 760 a un rôle extrêmement important : il change DE en sa valeur absolue. Si DE est positif, rien n'est fait, sinon son signe est inversé par complément à 2 sur 16 bits. Nous détaillerons le fonctionnement de VALABS plus loin.

Une fois arrivés en ligne 910, nous devons initialiser quelques variables pour commencer le tracé. Il s'agit principalement d'optimiser un peu le

tracé des colonnes. MIN est remplacé par sa valeur absolue, afin d'éviter un appel de VALABS lors de chaque boucle. En effet, le programme n'a désormais plus besoin de MIN mais de sa valeur absolue. On calcule également d'avance YGMAX-YGZERO et YGZERO-YGMIN. Leur utilisation lors des tracés en sera simplifiée, une soustraction étant supprimée.

En ligne 1090 débute le tracé. L'encre de tracé est mise à 1, et l'abscisse de départ est fixée à 50. Le tracé des axes (lignes 1130 à 1240) utilise les routines système HORLIN et VERLIN pour placer une ligne verticale de YGLIN à YGMAX à l'abscisse 50, et une ligne horizontale de 50 à 319 à l'ordonnée YGZERO. Une remarque est à formuler concernant ces routines HORLIN et VERLIN. Elles diffèrent de la routine DRAW par d'innombrables aspects. Non seulement leur utilisation demande un masque de couleur (d'où le CALL INKCOD des lignes 1140 et 1200), mais de plus, elles travaillent avec des coordonnées différentes de DRAW, PLOT ou MOVE. Au lieu d'utiliser les points logiques (640×400 quel que soit le mode), HORLIN et VERLIN n'acceptent que les points physiques. Les abscisses vont donc de 0 à 159 en mode 0, de 0 à 319 en mode 1 et de 0 à 639 en mode 2. Quant aux ordonnées, elles vont de 0 à 199.

Cela complique la tâche si le programme doit travailler dans n'importe quel mode, mais en revanche le tracé des lignes par ces routines est beaucoup plus rapide qu'avec DRAW (en fait, DRAW les utilise pour tracer les petits segments constituant une droite).

La boucle de tracé des valeurs commence à la ligne 1280. Comme nous l'avons remarqué lors de l'étude, il faut distinguer le cas des nombres négatifs et celui des nombres positifs. Le calcul de YG a lieu par les formules déterminées. Les variables YGBAS et YGHAUT contiennent les ordonnées extrêmes de la colonne graphique à tracer. Le tracé de cette dernière est effectué par le sous-programme TRACER.

La fin de la boucle (lignes 1670 à 1850) avance l'abscisse de tracé, et modifie la couleur de tracé.

L'ensemble du programme ne pose donc pas de problème particulier, il est juste une application directe des formules de calcul et de l'algorithme déterminés préalablement. Il en est tout autrement des diverses routines restantes. Chacune d'elles nécessite une étude approfondie.

Les sous-programmes

TRACER est la routine de tracé d'une colonne. Son fonctionnement est relativement limpide. Connaissant les données X, YHAUT et YBAS, elle trace la colonne correspondante, de largeur LARG. La seule difficulté est due au fonctionnement de la routine système VERLIN. En effet, celle-ci demande le masque du stylo de tracé, et non son numéro. Heureusement,

il existe une autre routine système qui peut se charger du calcul (CALL INKCOD). Il est également essentiel de sauver les registres utiles avant de tracer la colonne, car VERLIN les modifie tous.

Nous arrivons maintenant à CPDEBC. Comme nous l'avons laissé entendre, la comparaison des registres DE et BC n'est pas simple, car il s'agit de nombres signés. La méthode classique est d'utiliser une soustraction. $DE - BC$ doit nous donner Carry positionné si DE est inférieur à BC. Mais dans ce cas, $\$4000 - \$F000$ donnerait un Carry à 1. Pourtant, $\$4000$ est positif et donc supérieur à $\$F000$, qui lui est négatif. Il s'agit de nombres signés. Il faut donc traiter différemment les cas où DE et BC sont de signe contraire. Les deux cas où DE et BC sont de même signe peuvent se contenter de la soustraction simple. En effet, si ce sont deux nombres positifs, ils sont entre $\$0000$ et $\$7FFF$, et la soustraction donnera le Carry convenu. Si DE et BC sont négatifs également, car le nombre le plus grand de signe négatif est -1 , représenté par $\$FFFF$, et ce dernier est aussi le plus grand 16 bits. La soustraction convient également.

Il reste les deux cas épineux où l'un des deux registres est positif et le second négatif. En réalité, ce n'est pas un problème car un nombre positif est toujours plus grand qu'un négatif. Le tout est de savoir si DE contient le nombre positif ou le négatif. Si DE est positif, alors c'est lui le plus grand des deux, et on met le Carry à zéro, sinon, on le met à un, car BC est le plus grand.

Nous avons donc un algorithme applicable :

- si BC est négatif et DE aussi, faire $DE - BC$;
- si BC est négatif et DE non, $CARRY = 0$;
- si BC est positif et DE aussi, faire $DE - BC$;
- si BC est positif et DE non, $CARRY = 1$.

Il est très facile en Z-80 de savoir si un nombre signé est négatif ou non. Le Z-80 possède une instruction BIT qui permet de savoir si un bit d'un registre quelconque est à 1 ou à 0. Or, le signe du nombre contenu dans DE est indiqué par le bit 7 du registre D : 1 pour un nombre négatif, 0 pour un positif. L'instruction "BIT 7,D" permet de savoir si DE est positif : si oui, le flag Z est mis à 1; un "JP NZ,add" sautera donc à l'adresse indiquée si DE est négatif. On retient l'effet de l'instruction BIT par deux moyens. La définition est la suivante : BIT place, dans le flag Z, l'inverse du bit testé. Si le bit était à 1, Z sera à 0. La deuxième façon de mémoriser cela est de penser "BIT teste si le bit est à 1". Dans ce cas, "JP Z,add" est effectué si la réponse est oui. Par contre, "JP NZ,add" est effectué si c'est non.

Il est important d'assimiler le fonctionnement de BIT. Cette instruction est extrêmement puissante, mais la moindre inattention peut générer une erreur de logique (inversion des branchements à la suite du test).

La routine CPDEBC travaille comme suit :

BIT 7,B	: on teste si BC est négatif ;
JP Z,BCPOS	: non, on va traiter ailleurs ce cas, sachant que BC est positif ;
OR A	: cette petite et anodine opération a pour simple but de positionner le Carry à 0 ;
BIT 7,D	: cette fois-ci, on teste si DE est aussi négatif ;
JP Z,FIN	: non : donc le Carry reste à zéro, $DE > BC$. C'est fini ;
soustrac.	: puisque DE et VC sont positifs, on les soustrait simplement pour avoir directement le Carry ;
BCPOS:SCF	: cette instruction a l'effet inverse de OR A: elle met Carry à 1 ;
BIT 7,D	: DE est-il lui aussi positif ?
JP NZ,FIN	: non : donc $DE < BC$, donc Carry reste à 1. C'est fini ;
soustrac.	: DE et BC sont tous les deux positifs, on les soustrait ;
FIN:	: fin de routine.

Tout cela est bien entendu plus compliqué qu'une simple soustraction. Mais en dehors des particularités de certaines instructions (BIT, OR A, SCF), il n'y a aucune obscurité dans son déroulement.

Nous devons aussi penser à la multiplication. Nous avons vu plus haut que la routine définie pour le tracé de cercles ne convenait pas. Il y a en effet un grand risque de tomber sur un Overflow. Il faut ici multiplier un nombre 16 bits par un 8 bits.

Le mécanisme de la multiplication peut en revanche être récupéré. Nous multiplions toujours par A qui possède huit bits. Mais le travail sur DE est différent. En effet, alors que précédemment, seuls les 8 bits de droite de DE intervenaient, ici les 16 bits peuvent être significatifs. Un simple décalage à gauche nous ferait donc perdre une information, et le résultat intermédiaire serait faux. L'addition finale également, bien sûr.

Pour ne pas perdre les 8 bits du registre D lors des huit décalages successifs. Il faut utiliser un pseudo-registre 24 bits. Le Z-80 ne possède pas de tel registre, il faut le simuler. Pour cela, nous allons utiliser IY, Celui-ci va pointer sur une zone de 3 octets qui sera le fameux pseudo-registre. Notre choix se justifie par une remarque faite au chapitre 2. IY est un registre de pseudo-index : il permet d'assimiler n'importe quelle case mémoire au registre A. Cela signifie qu'on peut effectuer sur une case mémoire pointée par IY (ou IX) n'importe quelle opération, notamment rotation ou décalage.

En ce qui concerne le résultat final, il faut noter que son format est lui aussi susceptible de grimper jusqu'à 24 bits. Il faudra donc également le placer dans un pseudo-registre qui sera pointé par IX.

Si l'on excepte l'utilisation du registre A pour les additions de résultats intermédiaires, la multiplication est alors strictement identique à notre précédente routine. Le décalage du résultat intermédiaire est, entre autres, un bon exemple. Voici les deux séquences :

Multiplication 16 Bits

```
SLA E
RL D
```

Multiplication 24 Bits

```
SLA (IY+0)
RL (IY+1)
RL (IY+2)
```

Il nous reste à réaliser la division. Pas question ici de décalage. Il s'agit de soustractions successives. Mais le problème du format du résultat se pose encore : alors que nous divisons un nombre de 24 bits, le diviseur est de type 16 bits.

La solution a été effleurée plus haut : il faut procéder en deux étapes. La première partie de la division va exécuter une soustraction 24 bits (le troisième octet du diviseur étant assimilé à un zéro). Dès que cela sera possible, la soustraction se fera ensuite sur 16 bits. Le Carry est en effet correctement positionné par la soustraction 16 bits.

Le début de la division est donc le suivant :

- si le dividende est 16 bits (troisième octet=0), on soustrait directement le diviseur. On recommence jusqu'à ce que le Carry soit positionné, après avoir ajouté 1 au résultat. Cela intervient en effet lorsque la dernière soustraction a donné un résultat négatif (dividende inférieur au diviseur) ;
- si par contre le dividende est 24 bits, on soustrait le diviseur au dividende octet par octet, et éventuellement la retenue au troisième de ces octets. Cela est effectué par la séquence suivante :

```
LD A,(IX+0) : premier octet du dividende ;
SUB E       : soustraction simple du premier octet du diviseur.
              Cela positionne le Carry en cas de retenue ;
LD (IX+0),A : mise à jour du premier octet du dividende.
LD A,(IX+1) : deuxième octet.
SBC A,D     : soustraction du deuxième octet diviseur et de la
              retenue.
LD (IX+1),A
LD A,(IX+2) : dernier octet dividende.
SBC A,0     : soustraction de la retenue uniquement.
LD (IX+2),A
```

Remarque : le résultat est remis à jour après le test de fin, et non avant. En effet, on ne détecte la fin du travail que lorsque la soustraction produit un nombre négatif. Cette dernière opération est donc en trop, il ne faut pas la comptabiliser. Pour cette raison, on sort de la routine de division. Sinon, on ajoute un au résultat et on recommence.

Enfin, la dernière routine est celle qui transforme un nombre négatif en sa valeur absolue. Pour cela, il ne suffit pas d'inverser le bit de signe. Il s'agit du format "nombre signé". On obtient le nombre de signe opposé de la façon suivante :

- inverser les 16 bits du nombre, y compris celui de signe ;
- ajouter 1.

Remarquez qu'on obtient par cette opération (qui porte le joli nom de complémentation à deux) l'opposé d'un nombre, même si celui-ci est positif. -12 deviendra 12, mais 365 deviendra -365 (*voir annexe 7*).

Notre routine calcule la valeur absolue. Il ne faut donc pas procéder à la complémentation si le nombre est déjà positif : nous obtiendrions un résultat négatif !

Pour complémenter à deux le registre DE, il nous faut l'instruction CPL. Cette dernière inverse les bits du registre A. La complémentation passe donc par la séquence suivante :

LD A,E	: premier octet de DE
CPL	: inversion de tous les bits
LD E,A	: et remise à jour de E
LD A,D	: même travail sur le second octet de DE
CPL	: inversion des huit bits
LD D,A	: DE est maintenant inversé
INC DE	: et on ajoute 1 pour avoir le complément à deux !

Conclusion

Le programme 3.5 en langage Basic correspond à nos travaux.

```

10 *****
20 ** Programme 3.5 **
30 *****
40 '
50 'Trace d'histogrammes en langage machine
60 '
70 MEMORY &3FFF
80 DEFINT a-z
90 ad=&4200:lign=200
100 ctrl=0:READ c$:IF c$="fin" THEN 550
110 FOR i=1 TO LEN(c$) STEP 2
120 c=VAL("&" + MID$(c$,i,2))
130 POKE ad,c:ad=ad+1:ctrl=ctrl+c
140 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
150 lign=lign+10:GOTO 100
160 '

```



```

170 DATA DD7E00324F44DD7E02325044DD6E04DD, 1647
180 DATA 66052251443A5044FE02D82A51444E23, 1272
190 DATA 46ED435344ED435544233D4E234623ED, 1533
200 DATA 5B5344CD8D43D24042ED435344C34E42, 1789
210 DATA ED5B5544CD8D43DA4E42ED4355443DC2, 1968
220 DATA 2B4221140022574421C700225B442A55, 903
230 DATA 44CB7CC26F42211400225944C3A1422A, 1474
240 DATA 5344CB7CCA804221C700225944C3A142, 1719
250 DATA ED5B5544CD42443EB4CDB2432A5344ED, 1942
260 DATA 5B5544B7ED52EBCD0744211400092259, 1446
270 DATA 44ED5B5544CD4244ED5355442A5B44ED, 1799
280 DATA 4B5944B7ED427D3261442A5944ED4B57, 1656
290 DATA 44B7ED427D3262443E01326344213200, 1258
300 DATA 2264443E01CD2CBC1132002A5744ED4B, 1278
310 DATA 5B44CD62BC3E01CD2CBC113200013F01, 1282
320 DATA 2A5944CD5FBC3A504447C52A51445E23, 1481
330 DATA 5623225144CB7AC227433A6144CDB243, 1602
340 DATA ED5B5344CD07442A594409225D442A59, 1293
350 DATA 44225F44C34643CD42443A6244CDB243, 1610
360 DATA ED5B5544CD07442A5944B7ED42225F44, 1643
370 DATA 2A5944225D44CD6D432A64443A4F444F, 1269
380 DATA 06000923232264443A63443C326344FE, 1043
390 DATA 04DA69433E01326344C1108EC93A4F44, 1431
400 DATA 47ED5B6444C5D53A6344CD2CBC2A5F44, 1844
410 DATA ED4B5D44CD62BCD113C110E9C9D5E5C5, 2474
420 DATA CB78CAA343B7CB7ACAAE43626BB7ED42, 2397
430 DATA C3AE4337CB7AC2AE43626BB7ED42C1E1, 2360
440 DATA D1C9DD216644FD216944C50608DD3600, 1779
450 DATA 00DD360100DD360200FD7300FD7201FD, 1542
460 DATA 360200CB2FD2F543F5FD7E00DD8600DD, 2028
470 DATA 7700FD7E01DD8E01DD7701FD7E02DD8E, 1948
480 DATA 02DD7702F1FDCB0026FDCB0116FDCB02, 2016
490 DATA 16CB2F10D0C1C9DD216644010000B7DD, 1719
500 DATA 7E02B7CA2F44DD7E0093DD7700DD7E01, 1810
510 DATA 9ADD7701DD7E02DE00DD7702C33D44DD, 1953
520 DATA 7E0093DD7700DD7E019ADD7701D803C3, 1870
530 DATA 0F44CB7AC8F57B2F5F7A2F5713F1C900, 1835
540 DATA "fin",0
550 CLEAR:DEFINT a-z:MODE 1:WINDOW #0,1,15,1,2
560 INK 0,0:INK 1,15:INK 2,20:INK 3,26
570 DIM tablo(40)
580 PRINT "SAISIE"
590 i=1:tablo(0)=1
600 WHILE tablo(i-1)<>0 AND i<40
610 PRINT "Valeur";i;
620 INPUT tablo(i):i=i+1
630 WEND:MODE 1
640 larg=(320-50)/(i-2)-2
650 CALL &4200,@tablo(1),i-2,larg
660 GOTO 580

```

Notre programme de tracés d'histogrammes est sensiblement plus rapide que son équivalent Basic, et il est surtout beaucoup plus maniable. En effet, il peut travailler sur n'importe quel ensemble de valeurs. Sa plus grande particularité est son déroulement simple : quelques calculs préliminaires et une boucle unique pour tracer les valeurs. La difficulté ne réside que dans les routines de calcul. La moralité de l'histoire doit paraître évidente. En effet, bien que le programme ait un but uniquement graphique, les problèmes liés au graphisme sont passés quasiment inaperçus. Cet aspect paradoxal doit mettre en évidence l'avantage des routines système de l'Amstrad. En effet, le système de coordonnées retenu pour les routines VERLIN et HORLIN nous a facilité la tâche à un point tel que le tracé des colonnes proprement dit est devenu un jeu d'enfant. S'il avait fallu adresser directement chaque case mémoire de l'écran pour tracer les colonnes, le programme aurait incontestablement été plus compliqué.

REEMPLISSAGE DE ZONES

Introduction

Puisque vous avez constaté la facilité de programmation apportée par les nombreuses routines graphiques de l'Amstrad, vous êtes prêt pour une dernière mise en œuvre de celles-ci.

Jusqu'à présent, les programmes réalisés ne mettaient en œuvre que peu d'appels de routines système. Ceux-ci étaient le plus souvent restreints à un usage bien précis, et généralement unique, au niveau du travail effectué. Dans le tracé de cercles, les routines PLOT et DRAW traçaient uniquement, le contour du cercle, les calculs ne les utilisant pas. Il en allait de même pour le tracé d'histogrammes.

La routine de remplissage de contours que nous allons réaliser sera extrêmement différente. En effet, les routines graphiques vont y jouer un rôle essentiel, tandis que la partie calcul sera d'une grande simplicité. Nous nous proposons de réaliser une routine remplissant avec une certaine couleur n'importe quelle figure fermée placée sur l'écran, connaissant un point situé en son enceinte.

Les possesseurs des modèles 664 et 6128 savent sans doute qu'une instruction de ce type est intégrée à leur interpréteur Basic. Mais cette instruction remplit les figures dont le contour est d'une couleur uniforme. La routine que nous allons réaliser, au contraire, acceptera comme bordure de figure n'importe quelle couleur. Le fonctionnement des deux routines est en fait identique, seuls changent les tests déterminant si un point appartient au bord de la figure ou non. Mais aucun Amstrad ne fournit, à ce jour, d'instruction remplissant une figure de contour multicolore. Nous allons pallier ce manque.

Méthodes de remplissage

Il existe deux façons de traiter le problème. La première conduit à une routine extrêmement simple. Il faut toutefois fixer une contrainte : les figures devront être également simples, concaves (bien que le terme ne soit pas tout à fait exact d'un point de vue mathématique). Un cercle est par exemple une figure convenant à ce point de vue. L'algorithme de remplissage à partir d'un point situé à l'intérieur est alors le suivant :

- se déplacer sur le premier point du bord gauche, sur la ligne actuelle ;
- tracer jusqu'au premier point de la bordure de droite en remplissant les points du fond rencontrés ;
- grimper d'une ligne ;
- recommencer jusqu'à ce qu'aucun espace ne se trouve entre le point de gauche et celui de droite.

Et on recommence la même démarche vers le bas lorsque la partie supérieure de la figure est ainsi remplie (c'est-à-dire celle située au-dessus du point de départ).

Malheureusement, si cette procédure procure le double avantage de la simplicité et la rapidité, elle donne un remplissage tout à fait incorrect si la figure est plus complexe. Le schéma 3.13 représente une figure quelconque (à laquelle nous ferons référence dans la suite de notre étude) et son remplissage par notre méthode simplifiée, à partir du point marqué d'une croix.

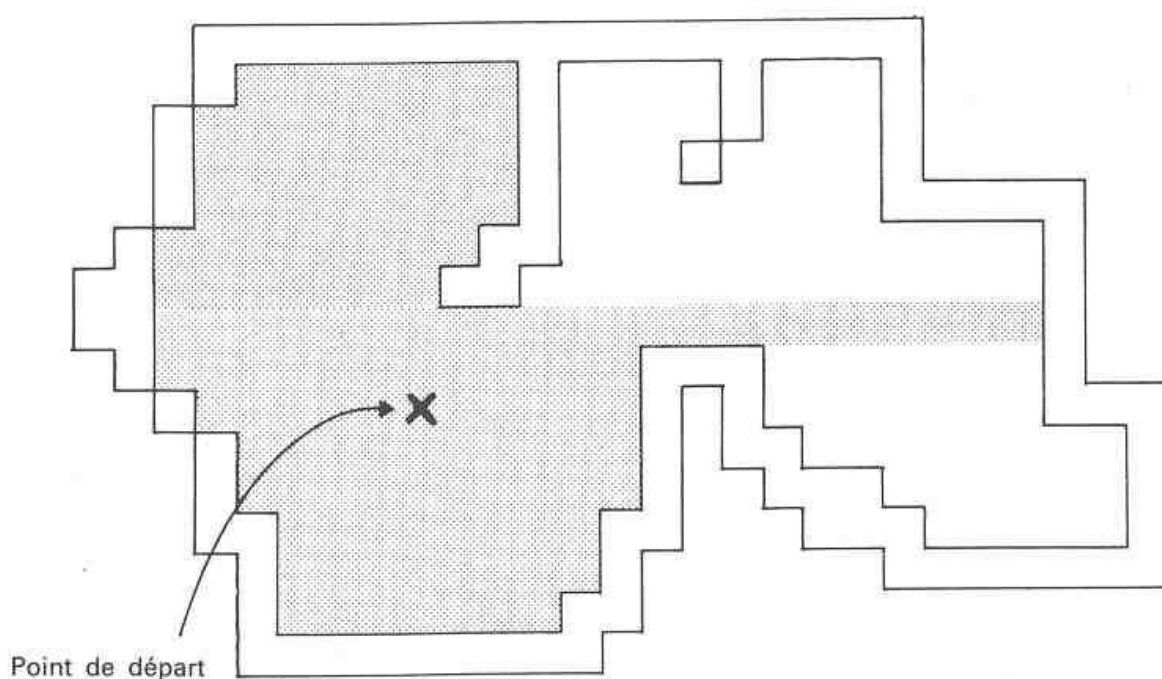


Schéma 3.13

Figure complexe exemple mal remplie.

Vous constatez qu'une grande partie de la figure est restée inexplorée. Vous trouverez pourquoi en suivant la logique du raisonnement à partir de ce point de départ. Lorsque le programme part du point gauche et remplit la figure jusqu'au premier point de bordure rencontré, il ignore si ce point de droite est réellement la bordure de la figure. D'où l'absence de remplissage de la zone qui se trouve éventuellement derrière.

La figure 3.14 représente le même processus appliqué à la même figure, mais à partir d'un point de départ différent.

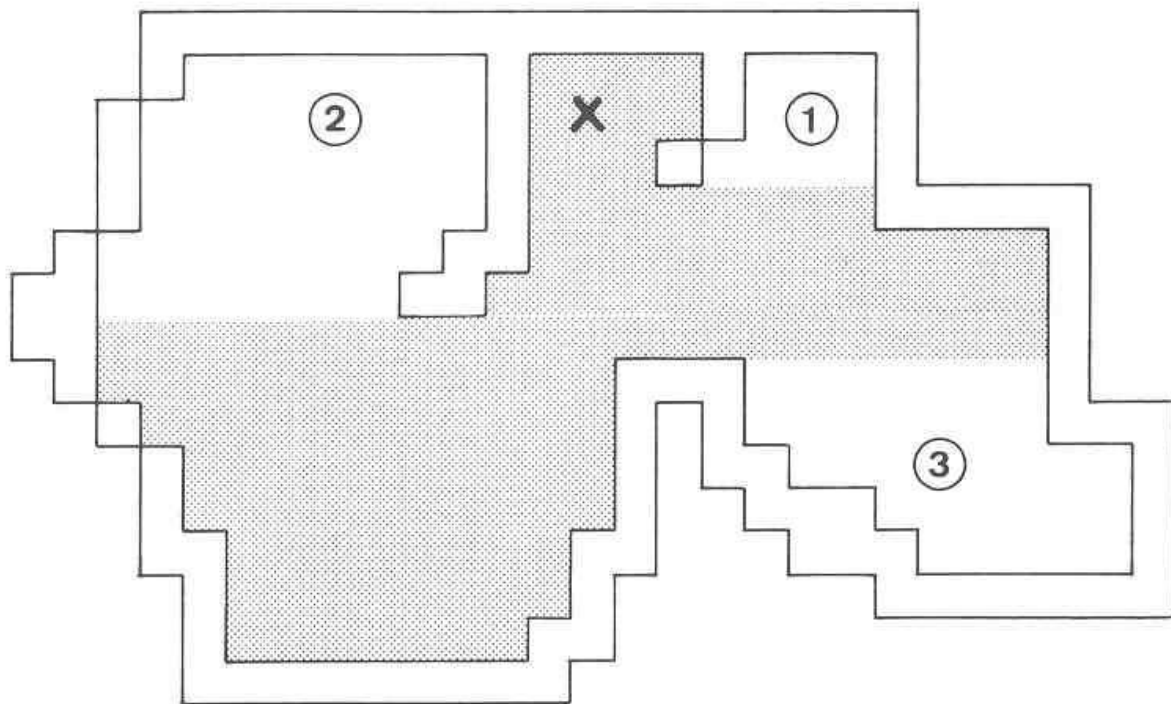


Schéma 3.14

Autre remplissage incorrect.

Le problème n'est pas moindre, bien au contraire. Toutefois, si la figure est connue du programme, il est bien entendu possible d'appeler plusieurs fois la routine en stipulant les différents points d'origine permettant un remplissage complet. Ce sera le cas sur le schéma 3.14 si l'on remplit ensuite à partir des points 1,2 et 3.

En remarquant cela, nous venons de trouver le principe du remplissage véritable : puisque l'algorithme de base ne peut pas s'occuper des zones invisibles, il suffit de le modifier afin qu'il repère les points de départ supplémentaires. De cette façon, nous pourrions repartir automatiquement de ces points pour continuer le remplissage des zones laissées de côté.

Le repérage des points de départ supplémentaires est un problème intéressant. Pour le résoudre, il faut en effet examiner tous les cas de figure possibles. L'algorithme de base, nous l'avons vu, procède de la même façon pour le remplissage vers le haut de la figure et vers le bas de celle-ci.

En ce qui concerne le sens vertical de déplacement, les tests de recherche des points à problème seront exactement les mêmes quel que soit ce sens.

Il nous suffira simplement d'inverser la progression de la routine sur l'axe des ordonnées.

En revanche, nous avons un algorithme qui part toujours du point de gauche pour aller vers la droite. Il faut donc repérer les problèmes entre ces deux points extrêmes.

Supposons que nous progressions vers le haut sur le schéma 3.15 : les deux points extrêmes trouvés provoqueront un remplissage incorrect, la zone problématique se situant au-delà.

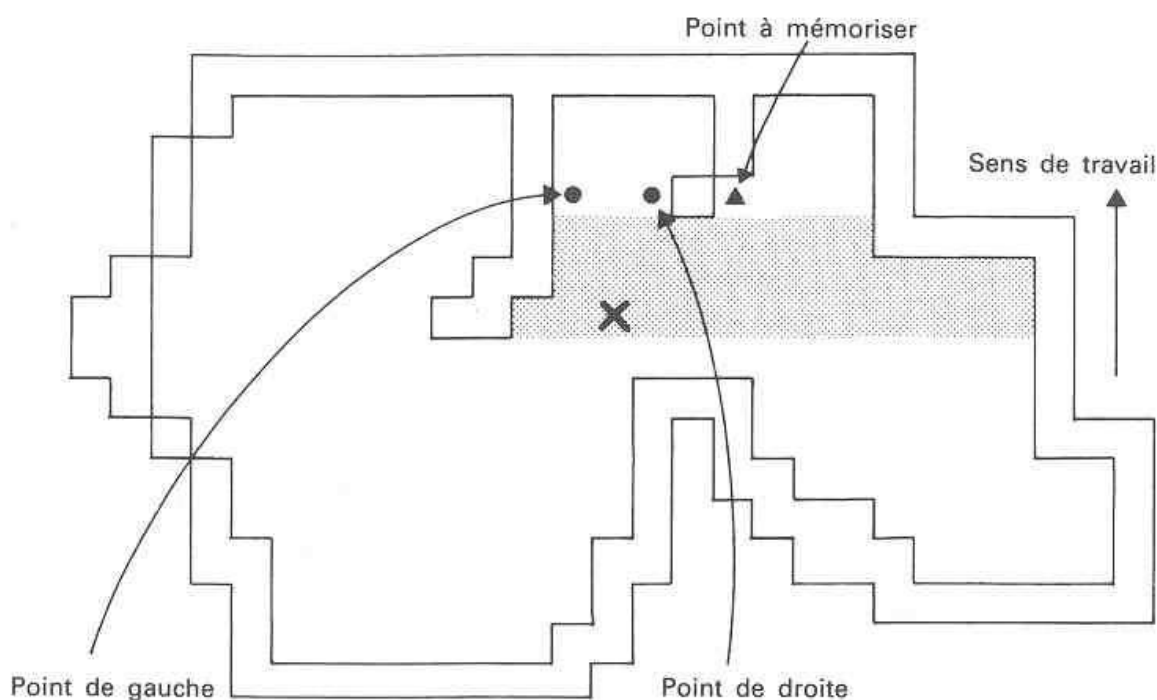


Schéma 3.15

Comment repérer un point spécial.

Par contre, nous pouvons repérer le problème lors du remplissage de la ligne précédente. En effet, si, parallèlement à celui-ci, nous surveillons la ligne du dessus, on va voir défiler dans l'ordre trois points couleur de fond, un point de bordure, et quatre points de fond. Or, nous savons alors pertinemment que le point de bordure situé dans cette liste va poser un problème, puisque les points suivants ne seront pas examinés. Il nous suffit donc de mémoriser, pour traitement ultérieur, le premier point de fond suivant ce point de bordure. Il sera le point de départ pour un nouvel appel de la routine. Ce point est marqué d'un triangle sur le schéma.

Ce que nous venons de remarquer pour un déplacement vers la droite est également valable lorsque nous recherchons le point le plus à gauche.

A propos des extrêmes, un autre problème peut se poser, illustré par la figure 3.16.

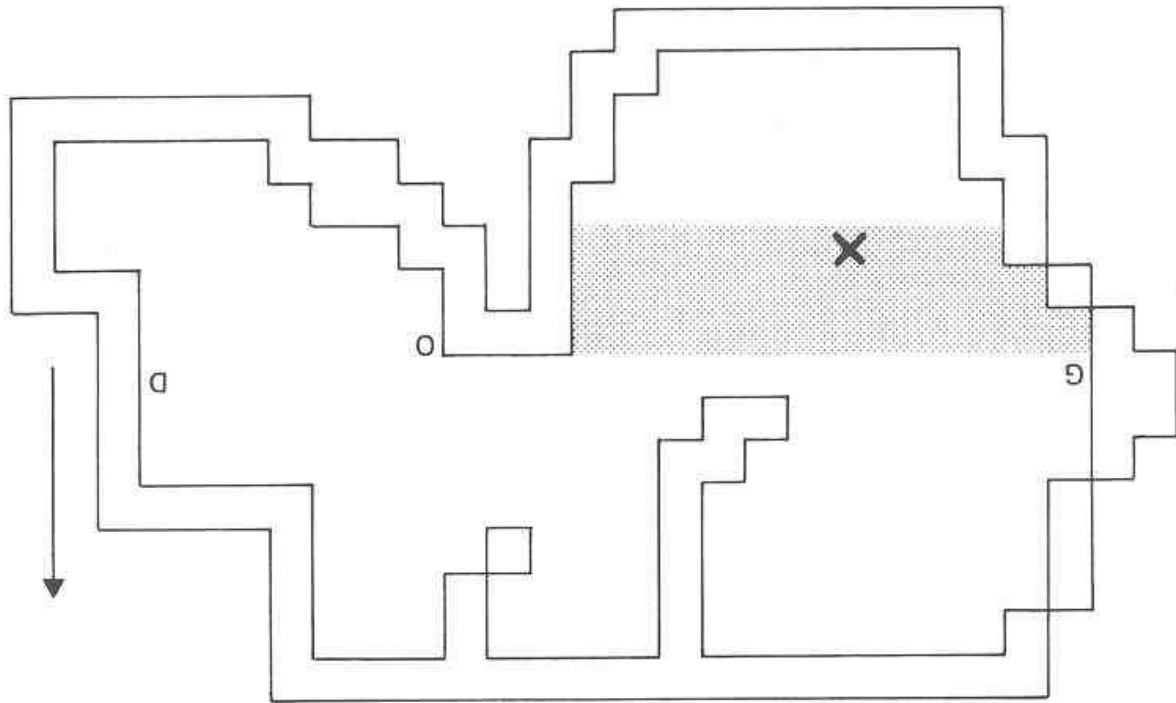


Schéma 3.16

Un autre type de point spécial.

Comment trouver le point gauche extrême (et le droit) lorsque l'on grimpe d'une ligne ? La solution est de partir du point précédent. On grimpe exactement à la verticale. Si le point rencontré est de la couleur du fond, alors cela signifie que la bordure s'est décalée à gauche, et dans ce cas nous poussons notre point dans ce sens jusqu'à toucher la bordure. Il faut bien sûr procéder identiquement pour le point de droite.

Mais la figure met en évidence le problème : le point D, nouvel extrême de droite, nous cache une zone non explorée. Il nous faut donc, lorsque nous déplaçons les points extrêmes avant de remplir la ligne, repérer comme précédemment les points de départ supplémentaires (le point O sur la figure). Il y a donc trois possibilités de découvrir des points à problème : ou bien à gauche et à droite sur la ligne précédente, ou bien à droite sur la ligne actuelle.

Cela dit, nous avons vu que la routine travaillait identiquement quel que soit le sens d'exploration vertical. Or, pour les points à problème, nous connaissons le sens qu'il faudra utiliser pour remplir la zone ignorée. Par exemple, dans notre dernière figure (3.16), il faut prendre le sens inverse (vers le bas). Le remplissage vers le haut sera inutile puisque déjà effectué. Il ne faudra donc pas uniquement mémoriser le point de départ, mais aussi le sens de travail pour la routine lorsqu'elle démarrera le processus à partir de celui-ci.

Algorithme de remplissage

Notre algorithme va se décomposer en plusieurs appels successifs d'une même routine. Celle-ci se chargera du remplissage simple d'une zone, à partir d'un point donné, et dans un seul sens vertical de progression. S'agissant du remplissage simple de notre figure de départ, il faudra donc appeler la routine une fois pour le haut de la figure, et une autre pour le bas. Ensuite, le cas échéant, nous recommencerons avec les éventuels points spéciaux repérés, dans le sens qui sera alors précisé.

La consistance de la routine de remplissage unidirectionnelle est conforme à l'étude précédente :

- à partir du précédent point de gauche (ligne en dessous) trouver le point situé le plus à gauche, cela en contrôlant que la ligne du dessous est conforme, et ne comporte pas de point spécial. Si un tel point est repéré, le mémoriser avec un sens de recherche inverse à celui actuellement utilisé (vers le bas si l'on progresse vers le haut) ;
- procéder à la même opération vers la droite, toujours en repérant un éventuel point particulier, mémorisation identique des points spéciaux (sens inverse de recherche) ;
- remplir la ligne entre ces deux points extrêmes, tout en scrutant la ligne située au-dessus pour repérer les autres points spéciaux. Si un tel point est trouvé, le mémoriser avec un sens de recherche identique ;
- passer à la ligne au-dessus ;
- recommencer tant que le point de gauche est différent de celui de droite.

Le programme 3.6 en Basic est une mise en œuvre de ce procédé.

```

10 *****
20 ** programme 3.6 **
30 *****
40 '
50 'Programme de remplissage de zones en Basic
60 'Programmation volontairement proche du LM
70 ' pour transposition facile.
80 '
90 RANDOMIZE TIME/300: 'initialise generateur ale
   atoire
100 DEFINT a-z: 'toutes variables de type entier
    16 bits
110 DIM xd(320),xg(320),y(320),d(320): ' pile sim
    ulee
120 MODE 1
130 WINDOW #0,1,10,1,25

```



```

140 FOR i=1 TO 20: 'on trace 20 lignes
150   MOVE 640*RND,400*RND: 'au hasard
160   DRAW 640*RND,400*RND,1+INT(RND*3): 'couleur
    au hasard
170 NEXT
180 PLOT 0,0: 'on trace un cadre
190 DRAW 639,0: 'pour eviter tests
200 DRAW 639,399
210 DRAW 0,399
220 DRAW 0,0
230 PLOT 800,800,1: 'positionne le stylo graphiqu
    e
240 MOVE 320,200: 'point de depart
250 c=1: 'stylo de remplissage
260 GOSUB 360: 'remplissage
270 END
280 '=====
    =====
290 '
300 'sous-programme de remplissage
310 '
320 '=====
    =====
330 '
340 'Initialisation de la fausse pile
350 '
360 x=XPOS: 'recupere X-depart
370 y=YPOS: 'recupere Y-depart
380 WHILE TESTR(-2,0)=0: 'on se deplace a gauche
    jusqu'au bord
390 WEND
400 xg=XPOS+2: 'xg=point de gauche=bordure+1
410 MOVE x,y: 'replaces au milieu figure
420 WHILE TESTR(2,0)=0: 'on se deplace a droite j
    usqu'au bord
430 WEND
440 xd=XPOS-2: 'xd=point de droite=bordure-1
450 sp=2: 'faut pointeur pile: deux points a etud
    ier
460 xg(1)=xg: 'premier point pour demi-figure du
    haut
470 xd(1)=xd: 'xg et xd=ceux trouves ci-dessus
480 y(1)=y+2: 'Y=ligne au dessus
490 d(1)=2: 'direction=vers le haut
500 xg(2)=xg: 'second point pour demi-figure du b
    as
510 xd(2)=xd: 'xg et xd=ceux trouves ci-dessus
520 y(2)=y: 'Y=ligne de depart
530 d(2)=-2: 'direction=vers le bas
540 '

```



```

550 'on vide la fausse pile point par point
560 '
570 WHILE sp<>0: 'tant qu'il y a des points a voi
    '
580     xg=xg(sp): 'recuperer xg de depart
590     xd=xd(sp): 'recuperer xd de depart
600     y=y(sp): 'recuperer y de depart
610     s=d(sp): 'recuperer direction de recherche
620     sp=sp-1: 'diminuer la pile de ce point
630 '
640 ' travail sur un point donne XG,XD,Y,S
650 '
660 MOVE xg,y: 'se placer a gauche
670 IF TESTR(0,0)=0 THEN 790: 'le point est dans
    la figure:deplacer
680 '
690 'le point xg n'est pas sur la bordure: on le
    pousse a gauche
700 '
710 WHILE TESTR(2,0)<>0: 'le point est sur bord:
    deplacer a droite
720 WEND
730 xg=XPOS: 'c'est le nouveau XG de cette ligne
740 GOTO 980: 'suite du traitement
750 '
760 'le point xg est sur la bordure :on le pouss
    e a droite
770 'en verifiant qu'il n'y a pas de point speci
    al en dessous
780 '
790 f=0: 'pas de point special repere
800 WHILE TESTR(-2,0)=0: 'on va vers la gauche
810 IF TESTR(0,-s)<>0 THEN 850: 'le point en dess
    ous est de la bordure, ok.
820 f=1: 'c'est un point special,il n'est pas sur
    bordure
830 x2=XPOS: 'retenir cette position d'abscisse
840 y2=YPOS: 'et l'ordonnee
850 MOVER 0,s: 'remonter a la ligne actuelle
860 WEND
870 '
880 xg=XPOS+2: 'c'est le nouvel xg, a droite de 1
    a bordure
890 IF f=0 THEN 980: 'il n'y avait pas de point s
    pecial, ok.
900 sp=sp+1: 'retenir le point detecte
910 xg(sp)=x2-2: 'xg de depart
920 xd(sp)=x2: 'xd de depart
930 y(sp)=y2: 'Y de depart
940 d(sp)=-s: 'direction de recherche=inverse

```

```

950 '
960 'on a fini avec xg, meme travail maintenant
    pour xd mais inverse.
970 '
980 MOVE xd,y: 'se placer a droite sur la ligne
990 IF TESTR(0,0)=0 THEN 1110: 'point dans le vid
    e, pousser a droite
1000 '
1010 'le point xd est sur la bordure, on le pous
    se a gauche
1020 '
1030 WHILE TESTR(-2,0)<>0: 'on pousse le point a
    gauche de la bordure
1040 WEND
1050 xd=XPOS: 'nouvel xd=a gauche de bordure droi
    te
1060 GOTO 1310: 'suite du traitement normale
1070 '
1080 'le point xd n'est pas sur la bordure, on l
    e pousse a droite
1090 'en verifiant la presence de points speciau
    x en dessous
1100 '
1110 f=0: 'pas de point special repere
1120 WHILE TESTR(2,0)=0: 'on va vers la droite
1130 IF TESTR(0,-s)<>0 THEN 1170: 'le point en de
    ssous est normal
1140 f=1: 'ce point est special, le memoriser
1150 x2=XPOS: 'xg de depart
1160 y2=YPOS: 'Y de depart
1170 MOVER 0,s: 'remonter a la ligne actuelle
1180 WEND: 'continuer de pousser a droite
1190 '
1200 xd=XPOS-2: 'nouvel xd=a gauche de bordure dr
    oite
1210 IF f=0 THEN 1310: 'pas de point special vu,
    ok.
1220 sp=sp+1: 'retenir ce point
1230 xg(sp)=x2: 'xg de depart
1240 xd(sp)=x2+2: 'xd de depart
1250 y(sp)=y2: 'Y de depart
1260 d(sp)=-s: 'direction de recherche=inverse
1270 '
1280 'fin du travail sur xg et xd, maintenant co
    mmence le remplissage
1290 'elementaire de la ligne.
1300 '
1310 IF xg>xd THEN 1700: 'c'est la fin du travail
1320 '

```

```

1330 MOVE xg,y: 'se placer a gauche au bord
1340 WHILE TESTR(2,0)=0: 'examiner vers la droite
1350 WEND
1360 '
1370 'on est alle jusqu'a la bordure visible. Il
      faut verifier qu'elle
1380 'correspond a celle trouvee a partir de la
      ligne precedente
1390 '
1400 IF XPOS<xd THEN 1480: 'le bord trouve n'est
      pas le vrai :point special
1410 MOVER -2,0
1420 DRAW xg,y: 'remplissage sans probleme
1430 GOTO 1680: 'et suite de la progression
1440 '
1450 'on a trouve un espace entre les deux xd su
      pposes. Mais peut-etre est-il
1460 'du a un bout de bordure horizontal ?
1470 '
1480 xs=XPOS: 'sauvegarde du XD trouve
1490 MOVER -2,0
1500 DRAW xg,y: 'remplissage
1510 MOVE xs,y: 'revenir a droite au bord trouve
1520 WHILE TESTR(2,0)<>0: 'sauter la bordure
1530 WEND
1540 IF XPOS>=xd THEN 1680: 'c'est bon, la bordur
      e etait horizontale
1550 '
1560 'il y a un espace entre la fin de la bordur
      e horizontale et la vraie
1570 'bordure, donc c'est une zone a traiter plu
      s tard.
1580 '
1590 sp=sp+1: 'il y a une zone a memoriser
1600 xg(sp)=XPOS: 'xg de depart=a droite bordure
      horizontale
1610 xd(sp)=xd: 'xd de depart=celui calcule d'apr
      es ligne prec.
1620 y(sp)=y: 'y de depart=actuel
1630 d(sp)=s: 'direction de recherche=la meme
1640 xd=xs-2: 'recadrer xd pour suite du travail
1650 '
1660 'fin du travail sur la ligne actuelle
1670 '
1680 y=y+s: 'progression verticale de recherche
1690 GOTO 660: 'et suite du travail sur nouvelle
      ligne
1700 WEND: 'finir de vider la pile
1710 RETURN: 'fin du remplissage

```

Il faut noter la partie initialisation de la routine. Elle récupère le point de départ, détermine les points gauche et droit de la ligne de départ, et place les points de départ des deux demi-zones initiales. On utilise à cet effet une fausse pile et un faux pointeur de pile SP. La pile est simulée par quatre tableaux XG, XD, Y et S. S est la direction de recherche : elle est -2 pour aller vers le haut et +2 vers le bas. Remarquez également les procédures de recherche des points extrêmes. Elles sont associées à la recherche des points spéciaux.

Le programme 3.6 est en Basic simplifié. Les instructions ont été rapprochées le plus possible du langage machine afin de simplifier la transposition. Le programme 3.7 qui suit est donc une simple traduction de ce programme.

```

10 ;
20 ;Programme de remplissage de zone
30 ;programme 3.7, transposition LM du prog 3.6
40 ;
BBC6      50 GPOS: EQU #BBC6
BBC0      60 MOVE: EQU #BBC0
BBF6      70 DRAW: EQU #BBF6
BBF3      80 TESTR: EQU #BBF3
BBC3      90 MOVER: EQU #BBC3

100 ;
5000      110 ORG #5000
120 ;
130 ;initialisations diverses
140 ;
5000 FD210853 150 LD IY,STACK
5004 FD360002 160 LD (IY),2
5008 21FF3F 170 LD HL,#3FFF
500B 222253 180 LD (PILE),HL
500E DD210A53 190 LD IX,COUL ;pointe sur couleur
5012 2A0C53 200 LD HL,(MODE) ;deplacement x mode
5015 7D 210 LD A,L
5016 2F 220 CPL
5017 6F 230 LD L,A
5018 7C 240 LD A,H
5019 2F 250 CPL
501A 67 260 LD H,A
501B 23 270 INC HL
501C 220E53 280 LD (NMODE),HL ;fin calcul -mode
290 ;
501F CDC6BB 300 CALL GPOS ;recupere xpos et ypos
5022 ED531053 310 LD (X),DE
5026 221253 320 LD (Y),HL ;stockage pour travaux
330 ;
5029 ED5B0E53 340 L1020: LD DE,(NMODE) ;deplacement vers gauche

```

502D	210000	350	LD HL,0	
5030	CDF3BB	360	CALL TESTR	
5033	B7	370	OR A	;vide ?
5034	CA2950	380	JP Z,L1020	;oui : continuer vers gauche
		390 ;		
5037	CDC6BB	400	CALL GPOS	
503A	2A0C53	410	LD HL,(MODE)	
503D	19	420	ADD HL,DE	
503E	221453	430	LD (XG),HL	;calcul fini pour xg 1ere ligne
5041	ED5B1053	440	LD DE,(X)	
5045	2A1253	450	LD HL,(Y)	
5048	CDC0BB	460	CALL MOVE	;move x,y
		470 ;		
504B	ED5B0C53	480 L1060:	LD DE,(MODE)	
504F	210000	490	LD HL,0	
5052	CDF3BB	500	CALL TESTR	
5055	B7	510	OR A	;vide ?
5056	CA4B50	520	JP Z,L1060	
		530 ;		
5059	CDC6BB	540	CALL GPOS	
505C	2A0E53	550	LD HL,(NMODE)	
505F	19	560	ADD HL,DE	
5060	19	570	ADD HL,DE	
5061	ED531653	580	LD (XD),DE	;fin calcul xd 1ere ligne
		590 ;		
		600	;empilage 1ere demi-zone (haut)	
		610 ;		
5065	2A1053	620	LD HL,(X)	
5068	ED5B0E53	630	LD DE,(NMODE)	
506C	19	640	ADD HL,DE	
506D	EB	650	EX DE,HL	
506E	2A2253	660	LD HL,(PILE)	
5071	CDFE52	670	CALL PUSHDE	
5074	ED5B1053	680	LD DE,(X)	
5078	CDFE52	690	CALL PUSHDE	
507B	ED5B1253	700	LD DE,(Y)	
507F	13	710	INC DE	
5080	13	720	INC DE	
5081	CDFE52	730	CALL PUSHDE	
5084	110200	740	LD DE,2	
5087	CDFE52	750	CALL PUSHDE	
		760 ;		
		770	;empilage 2eme demi-zone (bas)	
		780 ;		
508A	ED5B1453	790	LD DE,(XG)	
508E	CDFE52	800	CALL PUSHDE	
5091	ED5B1653	810	LD DE,(XD)	
5095	CDFE52	820	CALL PUSHDE	

```

5098 ED5B1253 830      LD  DE,(Y)
509C CDFE52   840      CALL PUSHDE
509F 11FEFF   850      LD  DE,-2
50A2 CDFE52   860      CALL PUSHDE
50A5 222253   870      LD  (PILE),HL
                    880 ;
                    890 ;DEBUT BOUCLE PRINCIPALE DE SAUT AU TRAITEMENT
                    900 ;
50A8 3A0853   910 GLOOP: LD  A,(STACK)
50AB B7       920      OR  A                      ;pile vide ?
50AC C8       930      RET Z                      ;fini !
                    940 ;
                    950 ;PROGRAMME DE TRAVAIL. Retourne en GLOOP si fini (pas de RET,
                    960 ;la nouvelle pile est uniquement utilisee pour passer
                    970 ;les parametres (zones a remplir)
                    980 ;
50AD 2A2253   990      LD  HL,(PILE)
50B0 CD0353  1000     CALL POPDE
50B3 ED531853 1010     LD  (DELY),DE
50B7 7B       1020     LD  A,E
50B8 2F       1030     CPL
50B9 5F       1040     LD  E,A
50BA 7A       1050     LD  A,D
50BB 2F       1060     CPL
50BC 57       1070     LD  D,A
50BD 13       1080     INC DE
50BE ED531A53 1090     LD  (NDELY),DE
50C2 CD0353  1100     CALL POPDE
50C5 ED531253 1110     LD  (Y),DE
50C9 CD0353  1120     CALL POPDE
50CC ED531653 1130     LD  (XD),DE
50D0 CD0353  1140     CALL POPDE
50D3 ED531453 1150     LD  (XG),DE
50D7 222253   1160     LD  (PILE),HL
50DA 3A0853   1170     LD  A,(STACK)
50DD 3D       1180     DEC A                      ;raj compteur pile
50DE 320853   1190     LD  (STACK),A
                    1200 ;
                    1210 ;traitement d'une ligne de la zone
                    1220 ;
50E1 ED5B1453 1230 NEWLIN: LD  DE,(XG)
50E5 2A1253   1240     LD  HL,(Y)
50E8 CDC0BB   1250     CALL MOVE                      ;move xg,y
                    1260 ;
50EB 110000   1270     LD  DE,0
50EE 210000   1280     LD  HL,0
50F1 CDF3BB   1290     CALL TESTR                      ;couleur du point ?
50F4 B7       1300     OR  A                      ;vide ?
50F5 CA1051   1310     JP  Z,L2060                      ;oui:pousser a gauche

```



```

1320 ;
1330 ;XG est du bord, pousser a droite
1340 ;
50F8 ED5B0C53 1350 L2020: LD DE,(MODE)
50FC 210000 1360 LD HL,0
50FF CDF3BB 1370 CALL TESTR ;a droite
5102 B7 1380 OR A ;IL Y A UNE COULEUR ?
5103 C2F850 1390 JP NZ,L2020 ;OUI:toujours du bord
5106 CDC6BB 1400 CALL GPOS
5109 ED531453 1410 LD (XG),DE ;nouvel xg
510D C38851 1420 JP L2210 ;suite
1430 ;
1440 ;xg n'est pas du bord, la caler a gauche
1450 ;tout en verifiant la ligne du dessous pour
1460 ;detecter les points critiques
1470 ;
5110 DD360100 1480 L2060: LD (IX+1),0 ;mise a zero flag
5114 ED5B0E53 1490 L2070: LD DE,(NMODE)
5118 210000 1500 LD HL,0
511B CDF3BB 1510 CALL TESTR
511E B7 1520 OR A ;vide ?
511F C24951 1530 JP NZ,L2140 ;non:on est a gauche au max
1540 ;
5122 110000 1550 LD DE,0
5125 2A1A53 1560 LD HL,(NDELY)
5128 CDF3BB 1570 CALL TESTR
512B B7 1580 OR A ;du bord ?
512C C23D51 1590 JP NZ,L2120 ;oui,pas de probleme
1600 ;
512F DD360101 1610 LD (IX+1),1
5133 CDC6BB 1620 CALL GPOS
5136 ED531C53 1630 LD (X2),DE
513A 221E53 1640 LD (Y2),HL
1650 ;
513D 110000 1660 L2120: LD DE,0
5140 2A1853 1670 LD HL,(DELY)
5143 CDC3BB 1680 CALL MOVER
5146 C31451 1690 JP L2070
1700 ;
5149 CDC6BB 1710 L2140: CALL GPOS
514C 2A0C53 1720 LD HL,(MODE)
514F 19 1730 ADD HL,DE
5150 221453 1740 LD (XG),HL ;nouvel xg cale a gauche bord
5153 AF 1750 XOR A'
5154 DDBE01 1760 CP (IX+1) ;flag critique ?
5157 CA8851 1770 JP Z,L2210 ;non,pas de probleme
1780 ;
1790 ;Enregistrer un point critique
1800 ;

```

515A	2A1C53	1810	LD	HL,(X2)	
515D	ED5B0E53	1820	LD	DE,(NMODE)	
5161	19	1830	ADD	HL,DE	
5162	EB	1840	EX	DE,HL	
5163	2A2253	1850	LD	HL,(PILE)	
5166	CDFE52	1860	CALL	PUSHDE	
5169	ED5B1C53	1870	LD	DE,(X2)	
516D	CDFE52	1880	CALL	PUSHDE	;XD
5170	ED5B1E53	1890	LD	DE,(Y2)	
5174	CDFE52	1900	CALL	PUSHDE	;Y
5177	ED5B1A53	1910	LD	DE,(NDELY)	
517B	CDFE52	1920	CALL	PUSHDE	;direction zone
517E	222253	1930	LD	(PILE),HL	
5181	3A0853	1940	LD	A,(STACK)	
5184	3C	1950	INC	A	
5185	320853	1960	LD	(STACK),A	
		1970			;
		1980			;FINI de recadrer xg, passer a xd. Travail identique
		1990			;
5188	ED5B1653	2000	L2210:	LD	DE,(XD)
518C	2A1253	2010		LD	HL,(Y)
518F	CDC0BB	2020		CALL	MOVE
		2030			;move xd,y
					;
5192	110000	2040		LD	DE,0
5195	210000	2050		LD	HL,0
5198	CDF3BB	2060		CALL	TESTR
					;couleur du point ?
519B	B7	2070		OR	A
					;vide ?
519C	CAB751	2080		JP	Z,L2270
		2090			;oui:pousser adroite
					;
		2100			;XD est du bord,pousser a gauche
		2110			;
519F	ED5B0E53	2120	L2230:	LD	DE,(NMODE)
51A3	210000	2130		LD	HL,0
51A6	CDF3BB	2140		CALL	TESTR
51A9	B7	2150		OR	A
					;toujours du bord ?
51AA	C29F51	2160		JP	NZ,L2230
					;toujours du bord
51AD	CDC6BB	2170		CALL	GPOS
51B0	ED531653	2180		LD	(XD),DE
					;nouvel xd
51B4	C32D52	2190		JP	L2430
		2200			;
		2210			;xd n'est pas du bord, le caler a droite
		2220			;en regardant la ligne d'avant pour les
		2230			;points critiques
		2240			;
51B7	DD360100	2250	L2270:	LD	(IX+1),0
					;mise a zero flag
51B8	ED5B0C53	2260	L2280:	LD	DE,(MODE)
51BF	210000	2270		LD	HL,0
51C2	CDF3BB	2280		CALL	TESTR
51C5	B7	2290		OR	A
					;toujours du bord ?

```

51C6 C2F051 2300 JP NZ,L2350 ;oui,a droite max,ok.
2310 ;
51C9 110000 2320 LD DE,0
51CC 2A1A53 2330 LD HL,(NDELY)
51CF CDF3BB 2340 CALL TESTR ;point critique ?
51D2 B7 2350 OR A ;du bord ?
51D3 C2E451 2360 JP NZ,L2330 ;oui,pas de probleme
2370 ;
51D6 DD360101 2380 LD (IX+1),1
51DA CDC6BB 2390 CALL GPOS
51DD ED531C53 2400 LD (X2),DE
51E1 221E53 2410 LD (Y2),HL
2420 ;
51E4 110000 2430 L2330: LD DE,0
51E7 2A1853 2440 LD HL,(DELY)
51EA CDC3BB 2450 CALL MOVER
51ED C3BB51 2460 JP L2280
2470 ;
51F0 CDC6BB 2480 L2350: CALL GPOS
51F3 2A0E53 2490 LD HL,(NMODE)
51F6 19 2500 ADD HL,DE
51F7 221653 2510 LD (XD),HL
51FA AF 2520 XOR A
51FB DDBE01 2530 CP (IX+1)
51FE CA2D52 2540 JP Z,L2430
2550 ;
2560 ;enregistrer point critique
2570 ;
5201 2A2253 2580 LD HL,(PILE)
5204 ED5B1C53 2590 LD DE,(X2)
5208 CDFE52 2600 CALL PUSHDE
520B EB 2610 EX DE,HL
520C ED4B0C53 2620 LD BC,(MODE)
5210 09 2630 ADD HL,BC
5211 EB 2640 EX DE,HL
5212 CDFE52 2650 CALL PUSHDE ;XD
5215 ED5B1E53 2660 LD DE,(Y2)
5219 CDFE52 2670 CALL PUSHDE ;Y
521C ED5B1A53 2680 LD DE,(NDELY)
5220 CDFE52 2690 CALL PUSHDE ;direction zone
5223 222253 2700 LD (PILE),HL
5226 3A0853 2710 LD A,(STACK)
5229 3C 2720 INC A
522A 320853 2730 LD (STACK),A
2740 ;
2750 ;fini de recadrer xd. au travail.
2760 ;
522D 2A1653 2770 L2430: LD HL,(XD)
5230 ED5B1453 2780 LD DE,(XG)

```

5234	B7	2790	OR	A	
5235	ED52	2800	SBC	HL,DE	;calcul XD-XG
5237	CB7C	2810	BIT	7,H	;xg-xd<0 ?
5239	C2A850	2820	JP	NZ,LOOP	;fini XD-XG<0
		2830 ;			
523C	ED5B1453	2840	LD	DE,(XG)	
5240	2A1253	2850	LD	HL,(Y)	
5243	CDC0BB	2860	CALL	MOVE	
5246	ED5B0C53	2870 L2450:	LD	DE,(MODE)	
524A	210000	2880	LD	HL,0	
524D	CDF3BB	2890	CALL	TESTR	
5250	B7	2900	OR	A	
5251	CA4652	2910	JP	Z,L2450	;pas bord:avancer a droite au bord
		2920 ;			
5254	CDC6BB	2930	CALL	GPOS	
5257	2A1653	2940	LD	HL,(XD)	
525A	EB	2950	EX	DE,HL	
525B	B7	2960	OR	A	
525C	ED52	2970	SBC	HL,DE	;calcul XPOS-XD
525E	CB7C	2980	BIT	7,H	;XPOS<XD ?
5260	C27A52	2990	JP	NZ,L2500	;oui, probleme
		3000 ;			
5263	ED5B0E53	3010	LD	DE,(NMODE)	
5267	210000	3020	LD	HL,0	
526A	CDC3BB	3030	CALL	MOVER	
526D	ED5B1453	3040	LD	DE,(XG)	
5271	2A1253	3050	LD	HL,(Y)	
5274	CDF6BB	3060	CALL	DRAW	;on remplit la ligne !
5277	C3F052	3070	JP	L2620	;fini pour cette ligne
		3080 ;			
527A	CDC6BB	3090 L2500:	CALL	GPOS	
527D	ED532053	3100	LD	(XS),DE	;sauvegarde xpos
5281	ED5B0E53	3110	LD	DE,(NMODE)	
5285	210000	3120	LD	HL,0	
5288	CDC3BB	3130	CALL	MOVER	
528B	ED5B1453	3140	LD	DE,(XG)	
528F	2A1253	3150	LD	HL,(Y)	
5292	CDF6BB	3160	CALL	DRAW	;on remplit
5295	ED5B2053	3170	LD	DE,(XS)	
5299	2A1253	3180	LD	HL,(Y)	
529C	CDC0BB	3190	CALL	MOVE	;repositionne
		3200 ;			
529F	ED5B0C53	3210 L2530:	LD	DE,(MODE)	
52A3	210000	3220	LD	HL,0	
52A6	CDF3BB	3230	CALL	TESTR	
52A9	B7	3240	OR	A	
52AA	C29F52	3250	JP	NZ,L2530	;du bord
		3260 ;			

```

52AD CDC6BB 3270 CALL GPOS
52B0 EB 3280 EX DE,HL
52B1 ED5B1653 3290 LD DE,(XD)
52B5 B7 3300 OR A
52B6 ED52 3310 SBC HL,DE ;calcul de xpos-xd
52B8 CB7C 3320 BIT 7,H ;xpos-xd>=0 ?
52BA CAF052 3330 JP Z,L2620 ;oui, pas de probleme
3340 ;
3350 ;point critique
3360 ;
52B0 CDC6BB 3370 CALL GPOS
52C0 2A2253 3380 LD HL,(PILE)
52C3 CDFE52 3390 CALL PUSHDE ;xg=xpos
52C6 ED5B1653 3400 LD DE,(XD)
52CA CDFE52 3410 CALL PUSHDE
52CD ED5B1253 3420 LD DE,(Y)
52D1 CDFE52 3430 CALL PUSHDE
52D4 ED5B1853 3440 LD DE,(DELY)
52D8 CDFE52 3450 CALL PUSHDE
52DB 222253 3460 LD (PILE),HL
52DE 2A2053 3470 LD HL,(XS)
52E1 ED5B0E53 3480 LD DE,(NMODE)
52E5 19 3490 ADD HL,DE
52E6 221653 3500 LD (XD),HL ;abandonner zone critique
52E9 3A0853 3510 LD A,(STACK)
52EC 3C 3520 INC A
52ED 320853 3530 LD (STACK),A
3540 ;
3550 ;FIN DU TRAVAIL SUR LA LIGNE, passer a la suivante
3560 ;
52F0 2A1253 3570 L2620: LD HL,(Y)
52F3 ED5B1853 3580 LD DE,(DELY)
52F7 19 3590 ADD HL,DE
52F8 221253 3600 LD (Y),HL
52FB C3E150 3610 JP NEWLIN
3620 ;
3630 ;SIMULATION PILE
3640 ;
52FE 73 3650 PUSHDE: LD (HL),E
52FF 2B 3660 DEC HL
5300 72 3670 LD (HL),D
5301 2B 3680 DEC HL
5302 C9 3690 RET
3700 ;
5303 23 3710 POPDE: INC HL
5304 56 3720 LD D,(HL)
5305 23 3730 INC HL
5306 5E 3740 LD E,(HL)
5307 C9 3750 RET

```

```

3760 *L+
530B 0000 3770 STACK: DEFW 0
530A 0000 3780 COUL: DEFW 0
530C 0000 3790 MODE: DEFW 0
530E 0000 3800 NMODE: DEFW 0
5310 0000 3810 X: DEFW 0
5312 0000 3820 Y: DEFW 0
5314 0000 3830 XG: DEFW 0
5316 0000 3840 XD: DEFW 0
5318 0000 3850 DELY: DEFW 0
531A 0000 3860 NDELY: DEFW 0
531C 0000 3870 X2: DEFW 0
531E 0000 3880 Y2: DEFW 0
5320 0000 3890 XS: DEFW 0
5322 0000 3900 PILE: DEFW 0

```

Pass 2 errors: 00

```

10 '*****
20 '** Programme 3.7 **
30 '*****
40 '
50 'programme de remplissage de zones en LM
60 'transposition du programme 3.6
70 '
80 MEMORY &2FFF
90 DEFINT a-z
100 ad=&5000:lign=180
110 ctrl=0:READ c$:IF c$="fin" THEN 580
120 FOR i=1 TO LEN(c$) STEP 2
130 c=VAL("&" + MID$(c$,i,2))
140 POKE ad,c:ad=ad+1:ctrl=ctrl+c
150 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
160 lign=lign+10:GOTO 110
170 '
180 DATA FD210853FD36000221FF3F222253DD210A532A0
    C, 1589
190 DATA 537D2F6F7C2F6723220E53CDC6BBED531053221
    2, 1867
200 DATA 53ED5B0E53210000CDF3BBB7CA2950CDC6BB2A0
    C, 2326
210 DATA 5319221453ED5B10532A1253CDC0BBED5B0C532
    1, 1855
220 DATA 0000CDF3BBB7CA4B50CDC6BB2A0E531919ED531
    6, 2296

```



```

230 DATA 532A1053ED5B0E5319EB2A2253CDFE52ED5B105
    3, 2036
240 DATA CDFE52ED5B12531313CDFE52110200CDFE52ED5
    B, 2437
250 DATA 1453CDFE52ED5B1653CDFE52ED5B1253CDFE521
    1, 2605
260 DATA FEFFCDFE522222533A0853B7C82A2253CD0353E
    D, 2420
270 DATA 5318537B2F5F7A2F5713ED531A53CD0353ED531
    2, 1788
280 DATA 53CD0353ED531653CD0353ED5314532222533A0
    8, 1730
290 DATA 533D320853ED5B14532A1253CDC0BB110000210
    0, 1493
300 DATA 00CDF3BBB7CA1051ED5B0C53210000CDF3BBB7C
    2, 2585
310 DATA F850CDC6BBED531453C38851DD360100ED5B0E5
    3, 2454
320 DATA 210000CDF3BBB7C249511100002A1A53CDF3BBB
    7, 2185
330 DATA C23D51DD360101CDC6BBED531C53221E5311000
    0, 1798
340 DATA 2A1853CDC3BBC31451CDC6BB2A0C5319221453A
    F, 2096
350 DATA DDBE01CA88512A1C53ED5B0E5319EB2A2253CDF
    E, 2287
360 DATA 52ED5B1C53CDFE52ED5B1E53CDFE52ED5B1A53C
    D, 2686
370 DATA FE522222533A08533C320853ED5B16532A1253C
    D, 1618
380 DATA C0BB110000210000CDF3BBB7CAB751ED5B0E532
    1, 2171
390 DATA 0000CDF3BBB7C29F51CDC6BBED531653C32D52D
    D, 2810
400 DATA 360100ED5B0C53210000CDF3BBB7C2F05111000
    0, 1861
410 DATA 2A1A53CDF3BBB7C2E451DD360101CDC6BBED531
    C, 2687
420 DATA 53221E531100002A1853CDC3BBC3BB51CDC6BB2
    A, 2078
430 DATA 0E5319221653AFDDBE01CA2D522A2253ED5B1C5
    3, 1775
440 DATA CDFE52EBED4B0C5309EBCDFE52ED5B1E53CDFE5
    2, 2950
450 DATA ED5B1A53CDFE522222533A08533C3208532A165
    3, 1626
460 DATA ED5B1453B7ED52CB7CC2A850ED5B14532A1253C
    D, 2481

```

```

470 DATA C0BBED5B0C53210000CDF3BBB7CA4652CDC6BB2
  A, 2639
480 DATA 1653EBB7ED52CB7CC27A52ED5B0E53210000CDC
  3, 2425
490 DATA BBED5B14532A1253CDF6BBC3F052CDC6BBED532
  0, 2858
500 DATA 53ED5B0E53210000CDC3BBED5B14532A1253CDF
  6, 2153
510 DATA BBED5B20532A1253CDC0BBED5B0C53210000CDF
  3, 2261
520 DATA BBB7C29F52CDC6BBEBED5B1653B7ED52CB7CCAF
  0, 3334
530 DATA 52CDC6BB2A2253CDFE52ED5B1653CDFE52ED5B1
  2, 2692
540 DATA 53CDFE52ED5B1853CDFE522222532A2053ED5B0
  E, 2250
550 DATA 53192216533A08533C3208532A1253ED5B18531
  9, 1200
560 DATA 221253C3E150732B722BC92356235EC9, 1602
570 DATA "fin",0
580 RANDOMIZE TIME/300
590 DEFINT a-z
600 MODE 0
610 FOR i=0 TO 15:INK i,i*2:NEXT
620 FOR i=1 TO 20:MOVE 640*RND,400*RND
630 DRAW 640*RND,400*RND,2+INT(RND*13):NEXT
640 PLOT 0,0:DRAW 639,0:DRAW 639,350:DRAW 0,350:
  DRAW 0,0
650 PLOT 800,800,1:MOVE 300,200
660 c=1
670 GOSUB 700
680 LOCATE 1,1:PRINT"TAPEZ UNE TOUCHE POUR CONTI
  NUER"
690 WHILE INKEY$="":WEND:RUN 590
700 base=&5308:POKE base+2,1:POKE base+4,4:CALL
  &5000
710 RETURN

```

Remarques sur la routine assembleur

Cependant, il convient de remarquer la zone associée au stockage des points particuliers. Pour éviter un écrasement de la pile système, une pile simulée est utilisée, placée à l'adresse \$3FFF. Cette pile est gérée par les routines POPDE et PUSHDE qui placent ou retirent le registre DE dans cette pile, en remettant à jour le pointeur de pile HL. Il faut bien comprendre que

la pile ainsi simulée remonte en mémoire vers le bas : si beaucoup de points spéciaux se présentent dans la pile, celle-ci va se rapprocher de \$3000. C'est pourquoi le programme Basic appelant place un MEMORY &2FFF. Cela réserve un espace de 4 Ko pour cette pile, permettant de stocker ainsi 500 points spéciaux. C'est une prévision pessimiste : il suffirait de prévoir de la place pour une vingtaine de points. Mais encore une fois, il vaut mieux penser au pire, et prévoir large. Notez également qu'il est impossible de prévoir l'encombrement de cette pile. Nous ne connaissons pas le nombre de points spéciaux à l'avance. C'est pourquoi nous ne pouvons pas prendre le risque de la placer au-dessus des routines. La seule solution est de l'installer totalement en dehors de la zone des programmes.

Vous constatez toutefois, dans cette application, l'importance des routines système. Sans elles, les tests seraient un véritable calvaire. Le point que nous avons souligné apparaît maintenant clairement : lorsque l'application nécessite un accès individuel des points et un système de coordonnées, le pack système intégré est plus qu'utile.

L'ACCÈS DIRECT A LA MÉMOIRE ÉCRAN | 4

OBJETS GRAPHIQUES

Nous l'avons constaté de façon spectaculaire lors des précédents chapitres, le traitement des graphismes par un accès direct à la mémoire écran est beaucoup plus rapide que l'utilisation des routines système. Le fait d'allumer un point par POKE ou LD en LM est un facteur de vitesse essentiel. Mais nous avons également pu affronter les difficultés de programmation de ce langage.

Pour obtenir un compromis qui soit assez facile à programmer et tout de même très rapide, nous allons devoir jongler avec les octets et la mémoire écran.

Nous devons avant tout définir une contrainte essentielle : quel que soit le but visé, nous devons traiter les graphismes sous forme d'objets graphiques. Cela suppose que nous devons coder en mémoire ou sur mémoire de masse tous les objets à manipuler avant de pouvoir traiter ceux-ci sur l'écran.

Comment peut-on définir un objet graphique ? Certains constructeurs ont inclus dans leur machine un processeur ou des routines capables de gérer des sprites ou lutins. Ces lutins correspondent exactement à ce que nous appelons un objet graphique. Ils représentent, sous la forme d'un certain nombre de données en mémoire, un dessin à placer sur l'écran selon certaines règles.

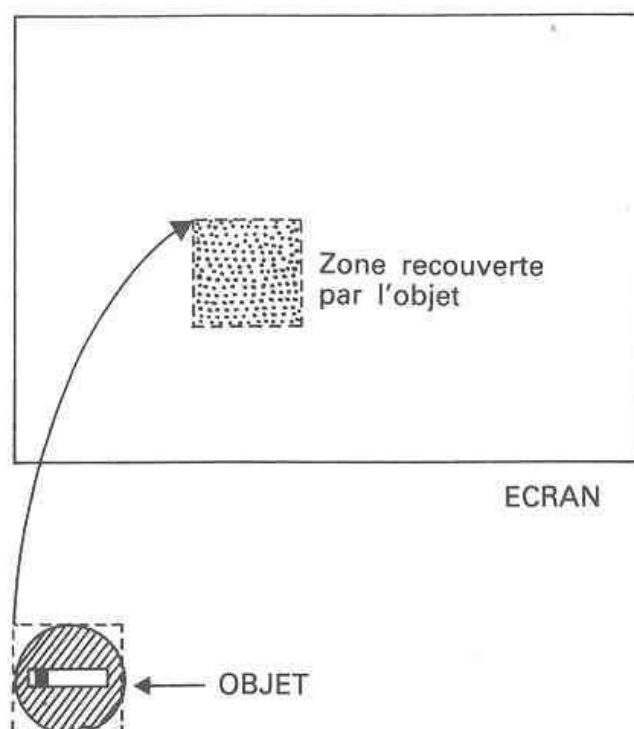


Schéma 4.1

Objet graphique.

Pour notre part, nos objets graphiques devront utiliser un format standard qui leur sera commun. Il faudra également que nous puissions les sauver sur cassette ou disquette ou les traiter facilement.

Si nous considérons par exemple la totalité de l'image écran comme un objet, nous pouvons sauver cet objet par une simple commande Basic :

```
SAVE "ECRAN",B,&C000,&4000
```

Cette commande indique que nous voulons sauver, dans un fichier nommé ECRAN.BIN, le contenu de la mémoire des adresses &C000 à &C000+&4000. Cela sauvegarde dans le fichier la totalité de la mémoire écran. Vous pouvez ensuite recharger l'objet n'importe où en mémoire par une commande comme LOAD"ECRAN". Il est possible, ainsi, de stocker l'écran ailleurs que dans la mémoire écran, donc sans l'afficher tel qu'il a été sauvé. Le problème de cette commande, c'est qu'elle génère un fichier de 16 Ko. Si nous traitons des objets de ce type, nous ne pourrons en stocker que **deux** en mémoire, plus un affiché. Inutile de dire qu'il est inconcevable de traiter des écrans entiers.

La plupart des jeux d'action récents manipulent une cinquantaine, voire une centaine d'objets différents sans aucun chargement de fichier. Tous ces objets sont stockés en mémoire et affichés, effacés, selon les directives du programme. Nous nous pencherons dans les chapitres suivants sur les problèmes de gestion des objets graphiques. Pour l'instant, notre préoccupation est la suivante : comment stocker en mémoire un objet graphique facile à afficher et à mémoriser ?

STRUCTURE DE LA MÉMOIRE ÉCRAN

Si vous avez lu les chapitres précédents, vous savez que la mémoire écran possède une structure relativement déroutante. L'adresse \$C000 représente le coin supérieur gauche de l'écran, \$C001 est la zone située à sa droite, \$C002 encore un peu plus à droite, et ainsi de suite. Tout devient curieux lorsque nous arrivons à \$C04F. En effet, cette adresse représente la dernière zone de la ligne du haut de l'écran. La logique voudrait que l'adresse suivante, \$C050, représente la gauche de la deuxième ligne. Malheureusement, elle représente la gauche de la neuvième ligne.

Nous nous sommes déjà penchés sur cette organisation mystérieuse. Elle est due à une astuce permettant une synchronisation facile du balayage vidéo et des affichages, ce qui donne un aspect net à l'écran. Mais cette astuce matérielle ne facilite pas la programmation : en effet, le traitement des objets graphiques est facile tant qu'on ne change pas de ligne graphique. Il suffit, pour progresser vers la droite sur l'écran, d'ajouter 1 à l'adresse, et de retrancher 1 pour revenir en arrière. En revanche, tout se complique si l'on doit passer à la ligne du dessous ou du dessus. L'adresse n'est pas simple à calculer.

Fort heureusement, plusieurs solutions existent pour vaincre cet obstacle. La plus simple des solutions consiste à utiliser les quatre routines fournies par le logiciel système. En effet, Amstrad a jugé utile de fournir des routines calculant les adresses écran à partir d'une adresse donnée. Ainsi, si l'adresse actuelle est dans le registre 16 bits HL, un CALL #BC26 calculera dans HL l'adresse de la position graphique située une ligne plus bas. Et cela simplifie grandement les routines de traitement.

Un inconvénient : ces appels de routines ralentissent le déroulement des opérations. En effet, leur programmation tient compte d'éventuels décalages en RAM suite à un éventuel scrolling. Cela vient d'une autre astuce matérielle. Elle ne nous intéresse pas ici ; en effet nous ne ferons jamais de scrolling lors de nos programmes. Après un scrolling, l'adresse \$C000 ne représente pas forcément le coin en haut à gauche ! Ce qui ne simplifie pas une tâche déjà ardue pour les programmes.

Donc, ces quatre routines système sont d'un emploi simple (et ne modifient aucun autre registre que HL, ce qui les rend encore plus tentantes), mais lent. Attention, lent ne veut pas dire ici inexploitable. Elles sont lentes dans la mesure où elles font plus que ce qu'on leur demande.

Une autre solution, plus astucieuse, consiste à faire ses propres routines. Cela implique une bonne connaissance de l'arithmétique 16 bits Z-80. Pour information, voici la routine qui permet de descendre d'une ligne dans HL :

```

GOBAS:  PUSH  BC      : sauvegarde du registre BC pour
                                travail ;
        LD    BC,#800 : cela est l'offset dans un cas normal ;
        ADD   HL,BC   : passe à la ligne suivante dans cas
                                normal ;
        JP    NC,FINAL : c'était bien un cas normal ;
        LD    BC,#C050 : offset pour revenir d'un bloc de huit
                                lignes ;
        ADD   HL,BC   : repositionne sur la bonne ligne ;
FINAL:  POP  BC      : récupère BC ;
        RET                : fin et retour.

```

Il est relativement facile de programmer, selon le même principe, une routine remontant d'une ligne.

Il y a une dernière solution très intéressante : elle consiste à utiliser ce qu'on appelle une table d'index. Pour ce faire, on range en mémoire les 200 adresses de débuts de ligne de l'écran. Pour se placer à la ligne 48, 20^e octet, on prend la 48^e adresse de la table, à laquelle on ajoute 20. Rien de plus.

Cette solution est encombrante mais rapide. Elle permet notamment de calculer directement et simplement une adresse écran d'après des coordonnées. Nous ne l'utiliserons pas dans ce livre (sauf dans le programme du chapitre 9).

Pour l'heure, nous allons nous contenter des routines système, qui ont tout de même l'avantage de la clarté.

RESTITUTION DES OBJETS

Notre principal problème est le suivant : alors que la structure de l'écran n'est pas linéaire en mémoire, nous devons gérer des objets graphiques qui seront placés en mémoire dans un bloc compact d'octets. Tout va bien si les objets n'utilisent qu'une seule ligne graphique de l'écran. Mais il va de soi qu'un tel cas est extrêmement rare. La plupart du temps, les objets occuperont un certain nombre de lignes.

La solution est de définir un format standard des objets. En l'occurrence, le format le plus pratique est généralement le rectangle. Si chaque objet est placé dans un rectangle avant sa mémorisation, il sera caractérisé par sa largeur et sa hauteur. Nous saurons alors combien d'octets de l'objet constituent une ligne de l'écran (grâce à la largeur) et combien de lignes il utilise (grâce à la hauteur). L'algorithme de dessin à partir de l'objet sera alors le suivant :

- Pour H (hauteur) lignes :
 Envoyer L (largeur) octets sur l'écran.
 Descendre d'une ligne sur l'écran.
- Suite du dessin.

Bien entendu, cela est très simple à programmer, mais nous impose tout de même plusieurs contraintes. Tout d'abord, puisque nous travaillons à partir d'octets et non de points, nous n'aurons plus accès à chaque point individuellement. De plus, nous devons coder les dessins avant leur mémorisation.

Avant de nous pencher sur la gestion des objets, nous allons prendre une grave décision : seul le MODE 0 sera utilisé. En effet, il est le seul à proposer seize stylos différents. Nous avons besoin de couleurs pour les objets car nous les utiliserons la plupart du temps dans des jeux. De plus, dans ce mode, un octet de la mémoire écran ne comporte que deux points graphiques. L'impossibilité de traiter chaque point sera donc moins contraignante, et de plus, nous le constaterons plus tard, nous pourrons tout de même accéder sans trop de difficulté à chacun des points d'un octet.

Nous devons réaliser deux routines : l'une codera un dessin en mémoire, l'autre dessinera un objet sur l'écran.

En effet, il est beaucoup plus simple de dessiner un objet point par point que de le coder octet par octet. Pour ce travail, il faudrait utiliser une table des masques et calculer chaque octet. Nous pouvons nous faciliter ce travail car le codage des dessins n'a lieu qu'une seule fois : il faut les coder avant de programmer l'application. Une fois les objets codés, nous n'avons plus besoin de leur dessin point par point, seul nous importe l'objet placé en mémoire, directement utilisable par la routine de restitution d'objet.

La routine de codage d'un objet est simple. On commence par dessiner l'objet point par point en haut de l'écran et à gauche, puis on le code ligne après ligne à une adresse fixe de la mémoire. Il faut bien sûr pour cela connaître sa largeur et sa hauteur. Une fois le codage achevé, on peut le sauver sous forme de fichier binaire.

```

10 '
20 'programme de creation de decor
30 'dessine le decor destine aux routines des cha
    pitres 4 a 8
40 'et cree le fichier binaire correspondant
50 'NDA :
60 'evitez les erreurs, tapez ce listing en mode
    2 (80 caracteres par ligne):
70 'chaque data hexa comporte exactement 80 cara
    cteres (le dernier tape arrive
80 'juste en dessous et a gauche du premier).
90 'Bon courage !
100 '
110 MODE 0
120 FOR I=0 TO 15
130 INK I,ASC(MID$("ACLFSPGJOIJXSDZQ",I+1,1))-65
140 NEXT
150 '
160 'affichage haut ecran
170 '
180 ae=49152:lign=460
190 ad=ae
200 FOR o=1 TO 2
210 ctrl=0:READ c$:IF c$="fin" THEN 320
220 FOR i=1 TO LEN(c$) STEP 2
230 c=VAL("&" + MID$(c$,i,2))
240 POKE ad,c:ad=ad+1:ctrl=ctrl+c
250 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
260 lign=lign+10
270 NEXT o
280 ae=ae+&800:IF ae>65535 THEN ae=ae+&C050
290 GOTO 190
300 '
310 '
320 ae=64768
330 ad=ae
340 FOR o=1 TO 2
350 ctrl=0:READ c$:IF c$="fin" THEN 440
360 FOR i=1 TO LEN(c$) STEP 2
370 c=VAL("&" + MID$(c$,i,2))
380 POKE ad,c:ad=ad+1:ctrl=ctrl+c
390 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
400 lign=lign+10
410 NEXT o
420 ae=ae+&800:IF ae>65535 THEN ae=ae+&C050
430 GOTO 330
440 SAVE"image",b,&C000,16384
450 '

```

[illegible]

[illegible]

1280 DATA C3C7CFCCFCFCCFCFCFCFCFCFCFCF8E0C0C0C
0C03030303030303030303030C0C0C0C0CCFCFCBC3C3,
4363

1290 DATA 3C3C3C3C3C3C3C3C3D3F3F3F3F6BC3C3C3C3C
C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3973F3F3F,
5530

1300 DATA C3C3C3C3C3C3C3C3C3C3C3C3CFCFCFCFCFCF8E0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C4DCFCFCB3C3C3C3C3,
5171

1310 DATA 3C3C3C3C3C3C3C3C3D3F3F3F3F3F3F3F3F3FC3
C3C3C3C3C3C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F,
3946

1320 DATA C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3CFCFCFCFC
CFCFCFCFCF0C0C0C0C0C0C0C0C0C4DCFCFCB3C3C3C3C3C3C3,
6529

1330 DATA 3C3C3C3C3C3C3C3C3D3F3F3F3F3F3F3F3F3F3F3F
3F3FC3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F,
3022

1340 DATA 3F3F3F3F3F3F6BC3C3C3C3C3C3C3C3C3C3C3CFCFC
CFCFCFCFCFCFCFCFCFCFCFCFCFCFCB3C3C3C3C3C3C3F3F,
6964

1350 DATA 3E3C3C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F3F
3F,
2498

1360 DATA 3F3F3F3F3F3F3F3F3F3F3FC3C3C3C3C3C3C3C3C3C
C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3973F3F3F,
5908

1370 DATA 3E3C3C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F3F
3F3F3F3F3F3F3F3F3F3F3F3F3F7A3F3F3F3F3F3F3F3F,
2557

1380 DATA 3F3F3F3F3F3F3F3F3F3F3F3F3F3F3FC3C3C3C3C3C
C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3C3973F3F3F3F3F3F3F,
4984

1390 DATA 3E3C3C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F3F
3F3F3F3F3F3F3F3F3F3F3F3FE1F03F3F3F3F3F7AF0,
3073

1400 DATA 3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F6BC3
C3C3C3C3C3C3C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F,
4016

1410 DATA 3F3C3C3C693C3C3D3F3F3F3F3F3F3F3F3F3F3F3F3F
3F3F3F3F3F3F3F3F3F3F3F3FF0C3C3F03F3F7AF0F0F0F0,
3753

1420 DATA F0F03F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F
3F3F3F6BC3C3C3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F3F,
3314

1430 DATA 3F3C3C69C3963C3D3F3F3F3F3F3F3F3F3F3F3F3F3F
7A3F3F3F3F3F3F3F7AF0C3C3C3C3C378F0F0F0F0F0F0,
4535

1440 DATA C378F0F0F0B53F3FB4B53F3F3F3F3F3F3F3F3F3F3F3F
3FF03F,
3770

1450 DATA 3F3E3CC3C3963C3F3F3F3F3F3F3F3F3F3F3F3F3F3F
E1F03F3F3F3F3F3FF0E1C3C3C3C3C378F0F0F0F0F0F0,
5144

1460 DATA C396F0F0F0F0F0F069B53F3F3F3F3F3F3F3F3F3F3F3F
7AC3B53F,
4270

1470 DATA 3FF0C3C3C3C3C3C3F3F3F3F3F3F3F3F3F3F3F3F7A
C3C3B53F7AF0B4F0C3C3C3C3C3C3C3C3963C78F0F0F0F0,
5789

1480 DATA C3C378F0F0F0F0B4C378F0F03F3F3F3F3F3F3F3F3F3F

[illegible]

[illegible]

```

000000000000000000041000000000000000000041,
455
1900 DATA 8200000000000000000000000004100000000000
000000008200000000000000000000C3820000000000,
650
1910 DATA 000000000000000000000000000000000000041
000000000000000000000000000000000000000041,
130
1920 DATA 0000000000000000000000000004100000000000
000000000000000000000000000000041820000000000,
260
1930 DATA 000000000000000000000000000000000000041
00000000000000000000000000000000000000000000,
65
1940 DATA 0000000000000000000000000000000000000000
000000000000000000000000000000041000000000000,
65
1950 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
1960 DATA 0000000000000000000000000000000000000000
000000000000000000000000000000041000000000000,
65
1970 DATA fin
1980 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
1990 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2000 DATA 0000000000000000000000000050A000000000000
00000000000000000000000000000000000000000000,
15
2010 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2020 DATA 0000000000000000000000000000F0F00000000000
00000000000000000000000000000000000000000000,
30
2030 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2040 DATA 000000000000000000000000000050F0F0000000000
00000000000000000000000000000000000000000000,
35
2050 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2060 DATA 000000000000000000000000000050F1B3300000000
00000000000000000000000000000000000000000000,
98
2070 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2080 DATA 000000000000000000000000000051B333300000000
00000000000000000000000000000000000000000000,
134
2090 DATA 0000000000000000000000000000000000000000
00000000000000000000000000000000000000000000,
0
2100 DATA 000000000000000000000000000001B333322000000

```

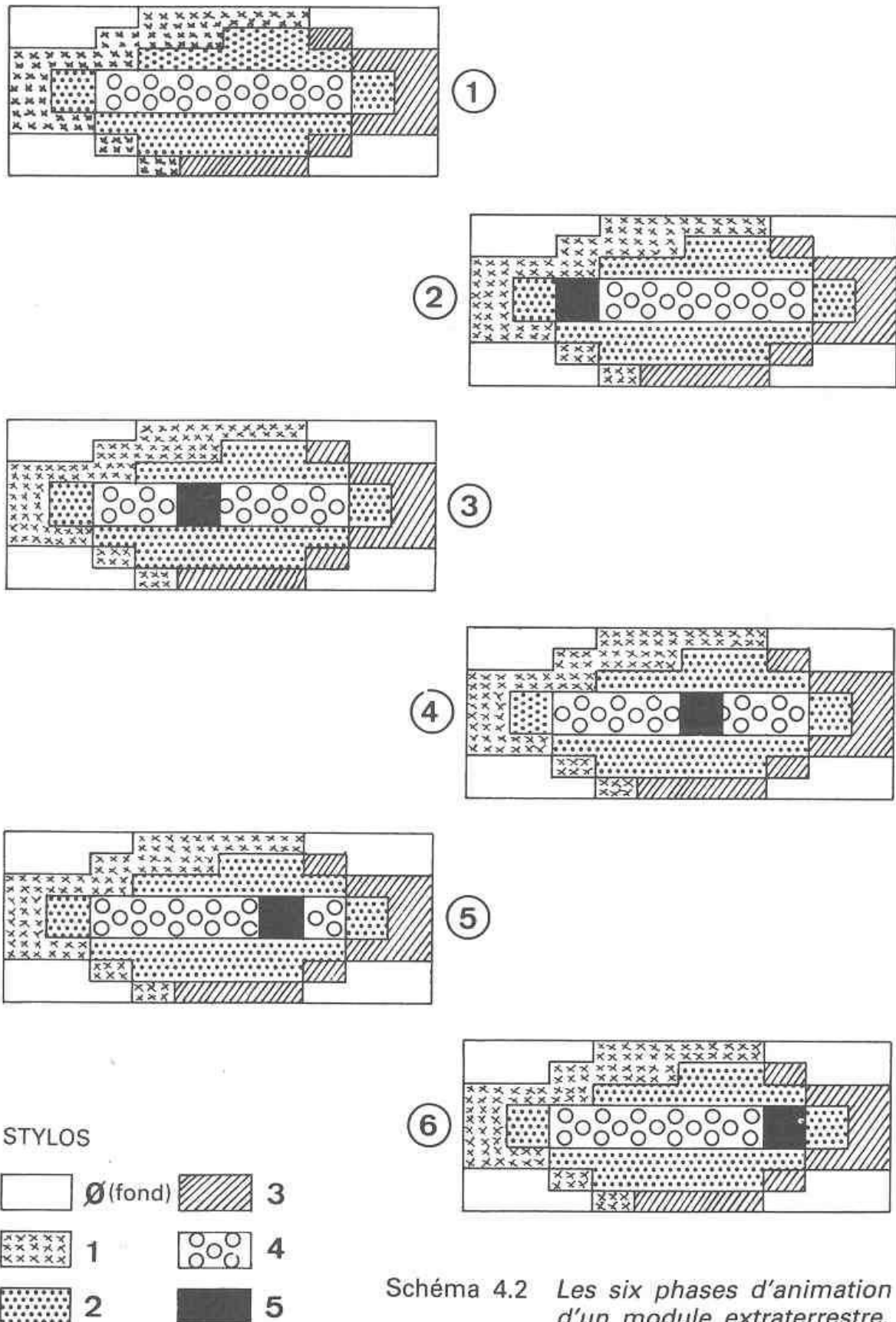

[illegible]

[illegible]

[illegible]

[illegible]

Nous allons prendre un exemple simple. L'objet et ses différentes phases d'animation, ainsi que les couleurs associées, sont résumés sur le schéma 4.2.



ROUTINE DE RESTITUTION

La première routine, que nous allons réaliser, sera celle restituant un dessin déjà codé en mémoire (*schéma 4.3*).

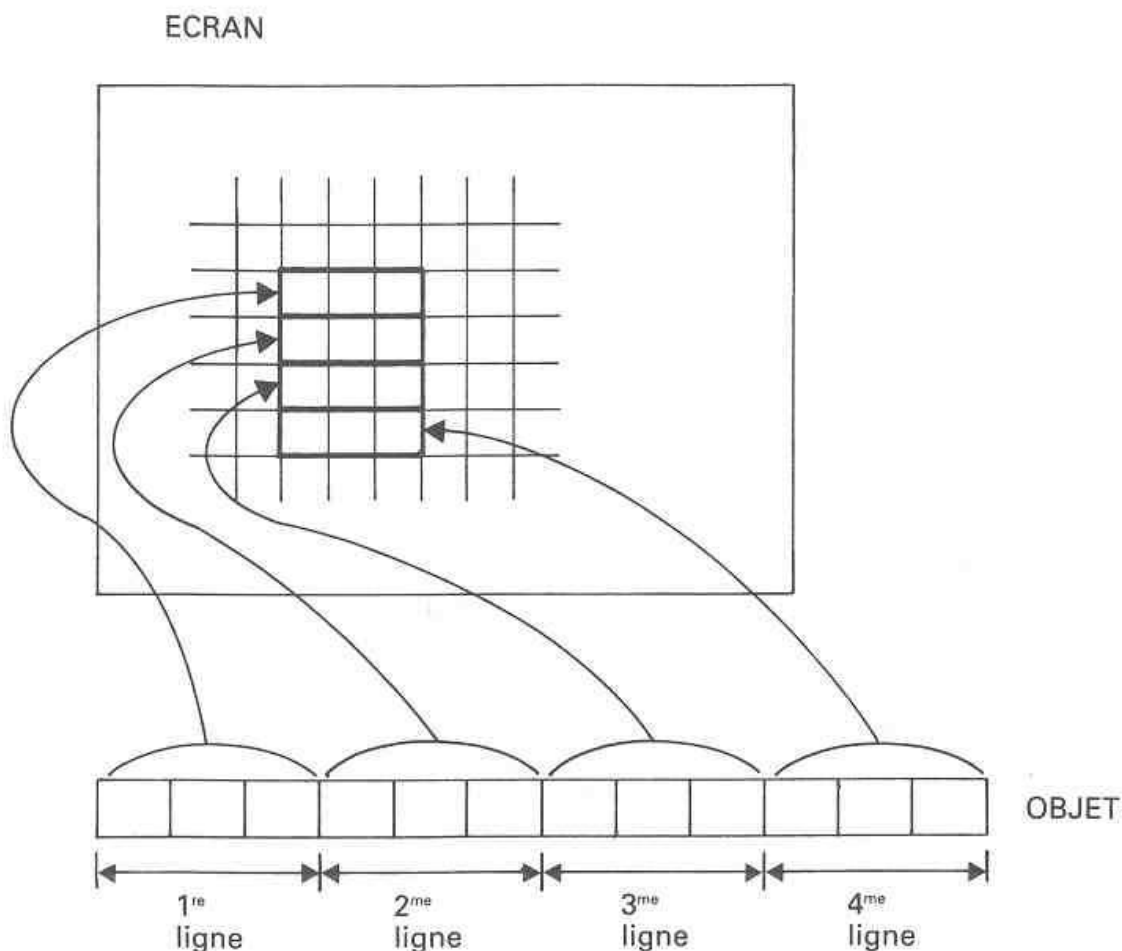


Schéma 4.3

Restitution d'objet.

Comme nous l'avons vu précédemment, elle nécessite quelques données en entrée. Il lui faut connaître l'adresse de localisation du dessin en mémoire, ainsi que sa largeur en octets et sa hauteur en lignes. Il lui faut également l'adresse de l'écran où l'on souhaite dessiner l'objet. Ces renseignements seront fournis dans les variables système respectives, BUF, LAR, HAU et ECRAN. Nous avons utilisé, à cet effet, une zone laissée libre par l'ensemble des routines du livre. Ce sont notamment les mêmes variables que pour les routines suivantes. Le listing assembleur assigne donc un label à chacune de ces variables grâce à la directive EQU (*voir annexe 5*).

Notre utilisation des registres va être la suivante : DE va pointer sur l'écran (c'est la destination), HL sur le dessin (c'est l'origine), B sera le

compteur de lignes. Le travail principal est effectué par une instruction LDIR, qui copie LAR octets du buffer dans l'écran. La routine système #BC26 est utilisée pour descendre d'une ligne sur celui-ci.

La seule difficulté est liée au fonctionnement de cette dernière routine : elle travaille sur HL, alors que nous avons utilisé DE pour stocker l'adresse écran. On s'en sort grâce à l'instruction EX DE,HL qui échange les contenus des registres DE et HL. Puis on appelle #BC26, et on refait un EX DE,HL afin de replacer le pointeur buffer dans HL, et celui d'écran dans DE. Le programme 4.1 résume tout ceci.

```

10 ;
20 ;programme de copie d'objet
30 ;RAM->Ecran
40 ;programme 4.1
50 ;
4700 60      ORG  #4700
70 ;
80 ;ENTREE: (ECRAN) adresse du coin sup.gauche
90 ;      (BUF)  adresse de l'image
100 ;      (LAR)  nombre d'octets par ligne
110 ;      (HAU)  nombre de lignes
120 ;
459C 130 BUF:  EQU  #459C
45A0 140 ECRAN: EQU  #45A0
45A6 150 LAR:  EQU  #45A6
45A7 160 HAU:  EQU  #45A7
170 ;
4700 ED5BA045 180      LD  DE,(ECRAN)
4704 2A9C45 190      LD  HL,(BUF)
4707 3AA745 200      LD  A,(HAU)
470A 47 210      LD  B,A          ;compteur de lignes
220 ;
470B C5 230 NEWLIN: PUSH BC          ;sauvegarde compteur
470C D5 240      PUSH DE          ;sauvegarde adresse ligne
470D 3AA645 250      LD  A,(LAR)
4710 4F 260      LD  C,A
4711 0600 270      LD  B,0          ;BC=nombre d'octets
4713 EDB0 280      LDIR          ;transfert image
4715 EB 290      EX  DE,HL          ;sauvegarde pointeur buffer
4716 E1 300      POP  HL          ;ancien debut ligne
4717 CD26BC 310     CALL #BC26      ;descend d'une ligne
471A EB 320      EX  DE,HL
471B C1 330      POP  BC
471C 10ED 340     DJNZ NEWLIN      ;ligne suivante
471E C9 350      RET

```

ROUTINE DE MÉMORISATION

La seconde routine, qui effectue le travail inverse, est tout aussi simple. Le registre DE est utilisé pour pointer en mémoire (c'est la destination, qui va recevoir le codage du dessin) et HL sur l'écran. De cette façon, le travail est encore une fois effectué grâce à LDIR. Mais #BC26 travaille ici directement sur HL et nous n'avons pas besoin d'échanger celui-ci avec DE. La routine est présentée dans le programme 4.2

```

10 ;
20 ;programme de copie d'objet
30 ;Ecran->RAM
40 ;programme 4.2
50 ;
4730      60      ORG #4730
70 ;
80 ;ENTREE: (ECRAN) adresse du coin sup.gauche
90 ;      (BUF)  adresse du buffer
100 ;      (LAR)  nombre d'octets par ligne
110 ;      (HAU)  nombre de lignes
120 ;
459C      130 BUF:  EQU #459C
45A0      140 ECRAN: EQU #45A0
45A6      150 LAR:  EQU #45A6
45A7      160 HAU:  EQU #45A7
170 ;
4730 ED5B9C45 180      LD  DE, (BUF)
4734 2AA045    190      LD  HL, (ECRAN)
4737 3AA745    200      LD  A, (HAU)          ;nombre de lignes
473A 47        210      LD  B,A              ;utilise comme compteur
220 ;
473B C5        230 NEWLIN: PUSH BC          ;sauve compteur
473C E5        240      PUSH HL            ;sauve add.debut ligne
473D 3AA645    250      LD  A, (LAR)        ;nombre d'octets/ligne
4740 4F        260      LD  C,A
4741 0600      270      LD  B,0            ;transmis a BC pour LDIR
4743 EDB0      280      LDIR              ;transfert
4745 E1        290      POP  HL            ;recupere debut ligne
4746 CD26BC    300      CALL #BC26         ;descend d'une ligne
4749 C1        310      POP  BC
474A 10EF      320      DJNZ NEWLIN        ;suite dessin
474C ED539C45  330      LD  (BUF),DE        ;Raj variable buffer
4750 C9        340      RET

```

Maintenant, il faut mettre les routines en application. Pour cela, nous allons utiliser les graphismes définis plus haut, représentant un petit vaisseau extraterrestre multicolore équipé d'un rayon oscillatoire. Nous avons défini les six phases du dessin. Le mouvement de l'objet est ici très simple, mais le principe des phases d'animation peut bien entendu être appliqué à des mouvements beaucoup plus complexes comme le déplacement d'un personnage. Il faut alors dessiner les différentes phases du mouvement, y compris les intervalles entre chaque position clef du mouvement. Chaque phase est codée en mémoire. Pour la restitution du mouvement, il suffit d'envoyer successivement à l'écran chaque phase.

Le programme 4.3 réalise le codage en mémoire des six phases de mouvement de notre petit module. La routine 4.2 est mise en place en mémoire. Puis, chaque phase est dessinée sur l'écran et codée par un appel de cette routine.

```

10 *****
20 *** Programme 4.3 ***
30 *****
40 '
50 'dessin d'un objet en plusieurs phases
60 'et codage de cet objet en memoire
70 'aux adresses $2fff et suite.
80 'application de la routine 4.2
90 '
100 MEMORY &2FFF
110 DEFINT a-z
120 ad=&4730:lign=200
130 ctrl=0:READ c$:IF c$="fin" THEN 240
140 FOR i=1 TO LEN(c$) STEP 2
150 c=VAL("&"+MID$(c$,i,2))
160 POKE ad,c:ad=ad+1:ctrl=ctrl+c
170 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
180 lign=lign+10:GOTO 130
190 '
200 DATA ED5B9C452AA0453AA74547C5E53AA645, 1908
210 DATA 4F0600EDB0E1CD26BCC110EFED539C45, 2147
220 DATA C9, 201
230 DATA "fin"
240 '
250 INK 0,0:INK 1,21:INK 2,15:INK 3,10:INK 4,6:I
    NK 5,26
260 POKE &459C,&0:POKE &459D,&30:'buffer depart
270 POKE &45A0,&0:POKE &45A1,&C0:'adresse ecran
    du dessin
280 POKE &45A6,7:'          largeur en oc
    tets=5 +2 bords vides
290 POKE &45A7,10:'          hauteur en li
    gnes=8 +2 bords vides
300 FOR phase=1 TO 6
310 buf(phase)=PEEK(&459C)+256*PEEK(&459D):'recu
    pere adresse dessin
320 MODE 0
330 READ larg,haut

```

```

340 FOR h=1 TO haut
350 FOR l=1 TO larg
360 READ colo:PLOT 8+(l-1)*4,396-(h-1)*2,colo
370 NEXT
380 NEXT
390 '
400 CALL &4730
410 NEXT phase:'dessin suivant
420 MODE 2
430 PRINT "DESSINS CODES EN MEMOIRE DE ";HEX$(bu
    f(1),4);" a ";HEX$(PEEK(&459C)+256*PEEK(&459D
    ),4)
440 PRINT:PRINT"Sauvegarde sur fichier DESSINS "
450 SAVE"dessins.bin",b,buf(1),6*7*10+1:'nb de p
    hases*largeur*hauteur+1
460 'phase 1
470 DATA 10,8
480 DATA 0,0,0,10,10,10,10,0,0,0
490 DATA 0,0,10,10,10,1,1,3,0,0
500 DATA 10,10,10,1,1,1,1,1,3,3
510 DATA 10,1,5,5,5,5,5,5,1,3
520 DATA 10,1,5,5,5,5,5,5,1,3
530 DATA 10,10,1,1,1,1,1,1,3,3
540 DATA 0,0,10,1,1,1,1,3,0,0
550 DATA 0,0,0,10,3,3,3,0,0,0
560 'phase 2
570 DATA 10,8
580 DATA 0,0,0,10,10,10,10,0,0,0
590 DATA 0,0,10,10,10,1,1,3,0,0
600 DATA 10,10,10,1,1,1,1,1,3,3
610 DATA 10,1,9,5,5,5,5,5,1,3
620 DATA 10,1,9,5,5,5,5,5,1,3
630 DATA 10,10,1,1,1,1,1,1,3,3
640 DATA 0,0,10,1,1,1,1,3,0,0
650 DATA 0,0,0,10,3,3,3,0,0,0
660 'phase 3
670 DATA 10,8
680 DATA 0,0,0,10,10,10,10,0,0,0
690 DATA 0,0,10,10,10,1,1,3,0,0
700 DATA 10,10,10,1,1,1,1,1,3,3
710 DATA 10,1,5,5,9,5,5,5,1,3
720 DATA 10,1,5,5,9,5,5,5,1,3
730 DATA 10,10,1,1,1,1,1,1,3,3
740 DATA 0,0,10,1,1,1,1,3,0,0
750 DATA 0,0,0,10,3,3,3,0,0,0
760 'phase 4
770 DATA 10,8
780 DATA 0,0,0,10,10,10,10,0,0,0
790 DATA 0,0,10,10,10,1,1,3,0,0
800 DATA 10,10,10,1,1,1,1,1,3,3
810 DATA 10,1,5,5,5,9,5,5,1,3
820 DATA 10,1,5,5,5,9,5,5,1,3
830 DATA 10,10,1,1,1,1,1,1,3,3
840 DATA 0,0,10,1,1,1,1,3,0,0
850 DATA 0,0,0,10,3,3,3,0,0,0
860 'phase 5
870 DATA 10,8
880 DATA 0,0,0,10,10,10,10,0,0,0
890 DATA 0,0,10,10,10,1,1,3,0,0
900 DATA 10,10,10,1,1,1,1,1,3,3
910 DATA 10,1,5,5,5,5,9,5,1,3

```

```

920 DATA 10,1,5,5,5,5,9,5,1,3
930 DATA 10,10,1,1,1,1,1,1,3,3
940 DATA 0,0,10,1,1,1,1,3,0,0
950 DATA 0,0,0,10,3,3,3,0,0,0
960 'phase 6
970 DATA 10,8
980 DATA 0,0,0,10,10,10,10,0,0,0
990 DATA 0,0,10,10,10,1,1,3,0,0
1000 DATA 10,10,10,1,1,1,1,1,3,3
1010 DATA 10,1,5,5,5,5,5,9,1,3
1020 DATA 10,1,5,5,5,5,5,9,1,3
1030 DATA 10,10,1,1,1,1,1,1,3,3
1040 DATA 0,0,10,1,1,1,1,3,0,0
1050 DATA 0,0,0,10,3,3,3,0,0,0

```

```

10 *****
20 ** Programme 4.3b **
30 *****
40 '
50 'creation d'un fichier dessin d'animation
60 'programme 4.3b
70 '
80 MEMORY &2FFF:MODE 2
90 DEFINT a-z
100 lign=240
110 ad=&3000:FOR phase=1 TO 6
120 buf(phase)=ad
130 ctrl=0:READ c$
140 IF c$="fin" THEN lign=lign+20:NEXT:GOTO 2330
150 POKE ad,0:ad=ad+1
160 FOR i=1 TO LEN(c$) STEP 2
170 c=VAL("&"+MID$(c$,i,2))
180 POKE ad,c:ad=ad+1:ctrl=ctrl+c
190 NEXT:POKE ad,0:ad=ad+1:READ teste
200 IF teste<>ctrl THEN PRINT"Erreur DATA ligne"
    lign:END
210 LOCATE 1,14:PRINT lign:lign=lign+10:GOTO 130
220 '
230 'PHASE 1
240 DATA 000000000000000000000000, 0
250 DATA 00000000CE880000000000, 342
260 DATA 00000045CC0C0000000000, 285
270 DATA 000000CE0C488000000000, 418
280 DATA 000000CE48782000000000, 430
290 DATA 000000CE0CD02000000000, 458
300 DATA 000000CE8C482000000000, 450
310 DATA 00000045CC0C8000000000, 413
320 DATA 00000000CC8C8000000000, 472
330 DATA 0000000014B00000000000, 196
340 DATA 00003F0014B00000000000, 259
350 DATA 0000A2C345CA0000000000, 628
360 DATA 0000A200CE8C8000000000, 636
370 DATA 0000A245CE0C8000000000, 577
380 DATA 00007865CC488000000000, 625
390 DATA 000078648C488C48802800, 812
400 DATA 000078648C48CC0CD05000, 936

```



```

410 DATA 00007A64CC0C844C882000, 814
420 DATA 00007865CC0C8000000000, 565
430 DATA 0000F065CE8C8000000000, 815
440 DATA 00000045CF8C8000000000, 544
450 DATA 0000000033330000000000, 102
460 DATA 000000450C0C8000000000, 221
470 DATA 000000CE0C48C000000000, 482
480 DATA 000045CC8C48C080000000, 805
490 DATA 00157EBC3CF0B020000000, 843
500 DATA 51E7CFCC8C48C080000000, 1255
510 DATA E7CFCC8C0C0C48C0000000, 1070
520 DATA E7CECCCC0C0CC0C0000000, 1253
530 DATA 0002000000000010000000, 3
540 DATA 15830000000006B0200000, 261
550 DATA 2F4B020000150F83000000, 291
560 DATA 0000000000000000000000, 0
570 DATA fin
580 'PHASE 2
590 DATA 0000000000000000000000, 0
600 DATA 00000000CE880000000000, 342
610 DATA 00000045CC0C0000000000, 285
620 DATA 000000CE0C488000000000, 418
630 DATA 000000CE48782000000000, 430
640 DATA 000000CE0CD02000000000, 458
650 DATA 000000CE8C482000000000, 450
660 DATA 00000045CC0C8000000000, 413
670 DATA 00000000CC8C8000000000, 472
680 DATA 0000000005830000000000, 136
690 DATA 0000000005830000000000, 136
700 DATA 0000000005830000000000, 136
710 DATA 0000000005830000000000, 136
720 DATA 0000000005830000000000, 136
730 DATA 0000000045CA0000000000, 271
740 DATA 00003F4BCE8C8000000000, 612
750 DATA 0000A245CE0C8000000000, 577
760 DATA 00007865CC488000000000, 625
770 DATA 000078648C488C48802800, 812
780 DATA 000078648C48CC0CD05000, 936
790 DATA 00007A64CC0C844C882000, 814
800 DATA 00007865CC0C8000000000, 565
810 DATA 0000F065CE8C8000000000, 815
820 DATA 00000045CF8C8000000000, 544
830 DATA 0000000033330000000000, 102
840 DATA 003FFC3C78F03000000000, 783
850 DATA 51E7CFCC8C48C080000000, 1255
860 DATA E7CFCC8C0C0C48C0000000, 1070
870 DATA E7CECCCC0C0CC0C0000000, 1253
880 DATA 000200143CF0A100000000, 483
890 DATA 158300CE0C486B02000000, 551
900 DATA 2F4B47CC8C1D0F83000000, 712
910 DATA 0000000000000000000000, 0
920 DATA fin
930 'PHASE 3
940 DATA 0000000000000000000000, 0
950 DATA 0000000000000000000000, 0
960 DATA 0000000000000000000000, 0
970 DATA 0000000000000000000000, 0
980 DATA 0000000000000000000000, 0
990 DATA 0000000000000000000000, 0
1000 DATA 00000000CE880000000000, 342
1010 DATA 00000045CC0C0000000000, 285

```

```

1020 DATA 000000CE0C488000000000, 418
1030 DATA 000000CE48782000000000, 430
1040 DATA 000000CE0CD02000000000, 458
1050 DATA 000000CE8C482000000000, 450
1060 DATA 00000045CC0C8000000000, 413
1070 DATA 00003F00CC8C8000000000, 535
1080 DATA 0000A24B45CA0000000000, 508
1090 DATA 0000A200CE8C8000000000, 636
1100 DATA 0000A245CE0C8000000000, 577
1110 DATA 00007865CC488000000000, 625
1120 DATA 000078648C488C48802800, 812
1130 DATA 000078648C48CC0CD05000, 936
1140 DATA 00007A64CC0C844C882000, 814
1150 DATA 00007865CC0C8000000000, 565
1160 DATA 0000F065CE8C8000000000, 815
1170 DATA 003FFC3C78F03000000000, 783
1180 DATA 51E7CFCC8C48C080000000, 1255
1190 DATA E7CFCC8C0C0C48C0000000, 1070
1200 DATA E7CECCCC0C0CC0C0000000, 1253
1210 DATA 000200003C780100000000, 183
1220 DATA 158300003C786B02000000, 441
1230 DATA 2F4B02450C1D0F83000000, 380
1240 DATA 000000CE0C48C000000000, 482
1250 DATA 000045CC8C48C080000000, 805
1260 DATA 0000000000000000000000, 0
1270 DATA fin
1280 'PHASE 4
1290 DATA 0000000000000000000000, 0
1300 DATA 0000000000000000000000, 0
1310 DATA 0000000000000000000000, 0
1320 DATA 0000000000000000000000, 0
1330 DATA 0000CE8800000000000000, 342
1340 DATA 0045CC0C00000000000000, 285
1350 DATA 00CE0C4880000000000000, 418
1360 DATA 00CE487820000000000000, 430
1370 DATA 00CE0CD020000000000000, 458
1380 DATA 00CE8C4820000000000000, 450
1390 DATA 0045CC0C80000000000000, 413
1400 DATA 0000CC8C80000000000000, 472
1410 DATA 0000058300000000000000, 136
1420 DATA 0000058300000000000000, 136
1430 DATA 000045CA00000000000000, 271
1440 DATA 3F87CE8C80000000000000, 672
1450 DATA A245CE0C80000000000000, 577
1460 DATA 7865CC48C0000000000000, 689
1470 DATA 78648C488C488028000000, 812
1480 DATA 78648C48CC0CD050000000, 936
1490 DATA 7A64CC0C844C8820000000, 814
1500 DATA 7865CC0C80000000000000, 565
1510 DATA F065CE8C80000000000000, 815
1520 DATA 003FFC3C78F03000000000, 783
1530 DATA 51E7CFCC8C48C080000000, 1255
1540 DATA E7CFCC8C0C0C48C0000000, 1070
1550 DATA E7CECCCC0C0CC0C0000000, 1253
1560 DATA 00023C7800000010000000, 183
1570 DATA 15833C78000006B0200000, 441
1580 DATA 2F4B0C0C80150F83000000, 441
1590 DATA 00CE0C48C0000000000000, 482
1600 DATA 45CC8C48C0800000000000, 805
1610 DATA 0000000000000000000000, 0
1620 DATA fin

```

```

1630 'PHASE 5
1640 DATA 000000000000000000000000, 0
1650 DATA 000000000000000000000000, 0
1660 DATA 000000000000000000000000, 0
1670 DATA 000000000000000000000000, 0
1680 DATA 0000CE880000000000000000, 342
1690 DATA 0045CC0C0000000000000000, 285
1700 DATA 00CE0C488000000000000000, 418
1710 DATA 00CE48782000000000000000, 430
1720 DATA 00CE0CD02000000000000000, 458
1730 DATA 00CE8C482000000000000000, 450
1740 DATA 0045CC0C8000000000000000, 413
1750 DATA 0000CC8C8000000000000000, 472
1760 DATA 000005830000000000000000, 136
1770 DATA 3F0005830000000000000000, 199
1780 DATA A24B45CA0000000000000000, 508
1790 DATA A200CE8C8000000000000000, 636
1800 DATA A245CE0C8000000000000000, 577
1810 DATA 7865CC488000000000000000, 625
1820 DATA 78648C488C488028000000, 812
1830 DATA 78648C48CC0CD050000000, 936
1840 DATA 7A64CC0C844C8820000000, 814
1850 DATA 7865CC0C8000000000000000, 565
1860 DATA F065CE8C8000000000000000, 815
1870 DATA 0045CF8C8000000000000000, 544
1880 DATA 000033330000000000000000, 102
1890 DATA 003FFC3C78F03000000000, 783
1900 DATA 51E7CFCC8C48C080000000, 1255
1910 DATA E7CFCC8C0C0C48C0000000, 1070
1920 DATA E7CECCCC0C0CC0C0000000, 1253
1930 DATA 0017FC3C28000100000000, 376
1940 DATA 15830C48C0006B02000000, 537
1950 DATA 2FC30648C0954B83000000, 867
1960 DATA 000000000000000000000000, 0
1970 DATA fin
1980 'PHASE 6
1990 DATA 000000000000000000000000, 0
2000 DATA 000000000000000000000000, 0
2010 DATA 000000000000000000000000, 0
2020 DATA 0000CE880000000000000000, 342
2030 DATA 0045CC0C0000000000000000, 285
2040 DATA 00CE0C488000000000000000, 418
2050 DATA 00CE48782000000000000000, 430
2060 DATA 00CE0CD02000000000000000, 458
2070 DATA 00CE8C482000000000000000, 450
2080 DATA 0045CC0C8000000000000000, 413
2090 DATA 0000CC8C8000000000000000, 472
2100 DATA 000045CA0000000000000000, 271
2110 DATA 3F4BCE8C8000000000000000, 612
2120 DATA A245CE0C8000000000000000, 577
2130 DATA 7865CC488000000000000000, 625
2140 DATA 78648C488C488028000000, 812
2150 DATA 78648C48CC0CD050000000, 936
2160 DATA 7A64CC0C844C8820000000, 814
2170 DATA 7865CC0C8000000000000000, 565
2180 DATA F065CE8C8000000000000000, 815
2190 DATA 0045CF8C8000000000000000, 544
2200 DATA 000033330000000000000000, 102
2210 DATA 00450C0C8000000000000000, 221
2220 DATA 00CE0C48C000000000000000, 482
2230 DATA 45CC8C48C080000000000000, 805

```



```

1000 DATA 0000000044CC8800000000000000CCCC00000000
      , 816
1010 DATA 00000000CC00000000000000000000004488000000
      , 408
1020 DATA 00000044880000000000000000000000CC000000
      , 408
1030 DATA 000000CC00000000000000000000000044880000
      , 408
1040 DATA 0000008800000000000000000000000000880000
      , 272
1050 DATA 0000000000000000000000000000000000000000
      , 0
1060 DATA 0000000000000000000000000000000000000000
      , 0
1070 DATA 0000000000000000000000000000000000000000
      , 0
1080 DATA 0000000000000000000000000000000000000000
      , 0
1090 DATA fin
1100 'phase 5
1110 DATA 0000000000000000000000000000000000000000
      , 0
1120 DATA 0000000000000000000000000000000000000000
      , 0
1130 DATA 0000000000000000000000000000000000000000
      , 0
1140 DATA 0000000000000000000000000000000000000000
      , 0
1150 DATA 0000000000000000000000000000000000000000
      , 0
1160 DATA 0000000000000000000000000000000000000000
      , 0
1170 DATA 0000000000000000A2A200000000000000000000
      , 324
1180 DATA 00000000000000051F3F30000000000000000000
      , 567
1190 DATA 000000000000000515151000000000000000000
      , 243
1200 DATA 000000004488005151510000CC000000000000
      , 651
1210 DATA 000000CCCCCCCC8C0C0CCCCCCCC8800000000
      , 2352
1220 DATA 0000CCCC8844CCC8F0E0CCCC00CCCC88000000
      , 2432
1230 DATA 00CCCC0000000000C0800000000044CC880000
      , 1136
1240 DATA CCCC000000000000000000000000000044CC8800
      , 816
1250 DATA CC000000000000000000000000000000448800
      , 408
1260 DATA 00000000000000000000000000000000000000
      , 0
1270 DATA 00000000000000000000000000000000000000
      , 0
1280 DATA 00000000000000000000000000000000000000
      , 0
1290 DATA 00000000000000000000000000000000000000
      , 0
1300 DATA fin
1310 'phase 6
1320 DATA 00000000000000000000000000000000000000
      , 0

```


La deuxième solution consiste à intégrer un bord vide au dessin lors de son codage. Ainsi, le premier et le dernier octet de chaque ligne sont de la couleur du fond, il n'y a donc aucun scrupule à avoir. Par contre, ceci augmente la taille du codage. Mais la facilité de traitement qui en résulte vaut ce petit sacrifice. Cela ne complique aucunement le dessin préliminaire avant codage. Il suffit simplement de dessiner l'objet sur un écran vide, et de coder ce qui se situe autour de lui. C'est l'explication d'une des particularités du programme 4.3.

En effet, alors que le dessin occupe 5 octets de large et 8 lignes, on place dans les variables LAR et HAU les valeurs 7 et 10. Il s'agit simplement de coder un octet en plus dans chaque direction autour de l'objet. Celui-ci est dessiné par PLOT en haut à gauche de l'écran, en laissant une ligne vide au-dessus et deux points à gauche (un octet contient deux points). Une fois toutes les phases codées en mémoire, le programme écrit celles-ci dans un fichier binaire DESSINS.

Le programme 4.4. récupère ce fichier et affiche les phases l'une après l'autre, grâce à la seconde routine.

```

10 *****
20 ** Programme 4.4 **
30 *****
40 '
50 'dessin d'un objet en plusieurs phases
60 'd'après le codage effectuée par programme 4.3
70 'aux adresses $2fff et suite.
80 'application de la routine 4.1
90 '
100 MEMORY &2FFF
110 ad=&4700:lign=200
120 ctrl=0:READ c$:IF c$="fin" THEN 220
130 FOR i=1 TO LEN(c$) STEP 2
140 c=VAL("&"+MID$(c$,i,2))
150 POKE ad,c:ad=ad+1:ctrl=ctrl+c
160 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er-
    reur DATA ligne"lign:END
170 lign=lign+10:GOTO 120
180 '
190 DATA ED5BA0452A9C453AA74547C5D53AA645, 1892
200 DATA 4F0600EDB0EBE1CD26BCEBC110EDC9, 2271
210 DATA "fin"
220 '
230 MODE 0
240 LOAD"image.bin",&C000:'chargement du decor,
    optionnel
250 FOR I=0 TO 15

```

```

260 INK I,ASC(MID$("ACLFSPGJOLJXSDZQ",I+1,1))-65
270 'INK I,ASC(MID$("AMTQSVSJRLJXSRVQ",I+1,1))-6
    5 pour monochrome
280 NEXT
290 ecr=63120
300 POKE &45A6,7: '          largeur en oc
    tets
310 POKE &45A7,10: '          hauteur en n
    ombre de lignes
320 LOAD"dessins",&3000
330 FOR phase=1 TO 6
340 buf(phase)=%3000+(phase-1)*(7*10)
350 NEXT
360 '
370 FOR phase=1 TO 6
380 POKE &459C,buf(phase)-256*INT(buf(phase)/256
    ):POKE &459D,INT(buf(phase)/256)
390 ecr=ecr+1:POKE &45A0,ecr-256*INT(ecr/256):PO
    KE &45A1,INT(ecr/256)
400 CALL &4700:CALL &4700
410 FOR i=1 TO 70:NEXT
420 NEXT
430 GOTO 370

```

Vous remarquerez la boucle d'attente de la ligne 390, et le double appel de la routine en 380. Cela n'est destiné qu'à obtenir un déplacement souple de l'objet pour rendre le mouvement agréable à l'œil. La suppression de la boucle d'attente entraîne un mouvement saccadé, les affichages se succédant plus rapidement que l'œil ne les fixe.

REMARQUES

Les deux routines de ce chapitre sont utiles à plus d'un titre. Elles constituent en quelque sorte le noyau de l'ensemble des routines du livre. Bien qu'elles soient courtes et simples, elles procurent des fonctionnalités extrêmement pratiques :

- elles peuvent traiter des objets de taille quelconque ;
- elles travaillent à partir de données élémentaires 16 bits, faciles à traiter par un programme extérieur ;
- elles produisent un résultat brut indépendant de toute autre considération liée au programme appelant.

En revanche, elles ont deux défauts : elles n'autorisent pas de mouvement d'objet sur un fond de décor, et ne traitent que des objets codés

simplement, parfois très encombrants. Nous allons remédier à ces deux problèmes dans les chapitres suivants.

Les programmes 4.3b et 4.4b ont respectivement les mêmes fonctions que les 4.3 et 4.4, mais ici l'objet animé est un robot. Sa particularité est de posséder six phases d'animation plus complexes que pour le module, permettant ainsi de constater l'efficacité du principe. Les programmes 4.3c et 4.4c ont également les mêmes fonctions, mais le personnage animé est une puce extraterrestre. Vous pouvez sans problème créer 4.3c et 4.4c à partir de 4.3b et 4.4b. Seuls les DATA des dessins et quelques POKE sont modifiés.

Modifications de 4.4 pour 4.3b

```
290 ecr=59104
300 POKE &45A6,21: '      largeur en o
      ctets
310 POKE &45A7,19: '      hauteur en n
      ombre de lignes
320 LOAD"dessinsC",&3000
340 buf(phase)=&3000+(phase-1)*(21*19)
```

Modifications de 4.4 pour 4.4b

```
290 ecr=64768
300 POKE &45A6,13: '      largeur en o
      ctets
310 POKE &45A7,33: '      hauteur en n
      ombre de lignes
320 LOAD"dessinsB",&3000
340 buf(phase)=&3000+(phase-1)*(13*33)
```

Les fichiers "dessins", "dessinsb" et "dessinsc" créés par les programmes 4.3 sont utilisés dans les chapitres suivants afin de vous épargner de longues listes de DATA. Ne perdez donc pas la cassette ou disquette où vous les installez.

CODAGE DES OBJETS GRAPHIQUES | **5**

POURQUOI COMPACTER ?

Nous avons réalisé dans le chapitre précédent deux routines nous permettant de traiter facilement des objets. Elles permettent de gérer la plupart des petits objets sans difficulté ni inconvénient. Il en va autrement si les objets à dessiner sont de grande taille. Un simple dessin de 40 points sur 20 lignes va nécessiter $22 \times 22 = 484$ octets (y compris les bords vides). Nous pouvons probablement réduire cet encombrement.

La solution passe par un compacteur graphique. Si nous trouvons un moyen de réduire l'encombrement du dessin en le codant différemment, nous aurons plus de place pour les dessins.

Il existe de nombreuses façons de réduire un dessin. La plus simple est de réaliser un interpréteur graphique, et de transformer le dessin en programme. Par exemple, un grand rectangle vide sera une suite de commandes du type "PLOT premier coin, DRAW deuxième coin, DRAW troisième coin, DRAW quatrième coin, DRAW premier coin, REMPLIT point intérieur". Mais ce procédé a un gros inconvénient : sa lenteur. Par là même, il restreint son intérêt aux dessins géométriques et non animés.

Une autre solution est celle du compacteur graphique. Nous allons le réaliser.

MÉTHODES DE COMPACTAGE

Il faut définir un principe de codage. Nous allons compacter chaque ligne de dessin avant de la mémoriser. Lors de la restitution, nous décompacterons avant de dessiner sur l'écran.

Une façon simple de compacter est la suivante : au lieu de stocker chaque octet de la ligne, on stocke les parcelles d'octets identiques sous la forme "Nombre d'octets à dessiner, Contenu de ces octets". Ainsi, si la ligne comporte dix octets à \$00 de suite, ils deviendront deux octets : \$0A, et \$00, soit dix et zéro. Si le dessin comporte alors de nombreuses lignes intégrant elles-mêmes de nombreuses suites d'octets identiques, le compactage sera extrêmement avantageux. Par contre, il en sera autrement si chaque ligne ne contient que des octets différents ou non regroupés : dans ce cas, le nouveau codage est deux fois plus encombrant que l'ancien ! Il faut donc être prudent : le compactage n'est pas universellement intéressant.

Remarquez également que nous pourrions compacter par colonne au lieu de nous intéresser aux lignes : les problèmes et les avantages sont exactement les mêmes.

La solution, telle que nous l'avons envisagée, limite l'intérêt du compactage aux dessins formés de nombreuses lignes contenant des octets

groupés. Nous pouvons, moyennant une légère consommation supplémentaire, en élargir le champ. Pour cela, il suffit d'ajouter un octet préliminaire à chaque ligne. Cet octet permettra de représenter deux cas : ou bien le compactage n'était pas intéressant, auquel cas la ligne n'est pas codée et doit être recopiée telle quelle à l'écran, ou bien il faut la décompacter lors de l'affichage car le compactage était économique.

Cela raffine le procédé. Il se peut qu'une certaine zone du dessin possède de nombreux groupes d'octets, et que le reste du dessin soit anarchique. Dans ce cas, la distinction au niveau de chaque ligne nous permettra de gagner tout de même des octets.

Nous venons de définir le format d'un objet compacté en mémoire. En effet, nous savons que :

- si le premier octet a une certaine valeur, la ligne n'est pas compactée. Il a donc LAR octets à recopier simplement (par exemple par LDIR comme au chapitre précédent) ;
- si, par contre, il a une autre valeur donnée, la ligne est compactée. Au lieu de copier les octets un par un, nous devons lire un certain nombre de couples d'octets et les transformer à l'écran selon leur signification.

En ce qui concerne la valeur de cet octet préliminaire, vous remarquerez que nous n'avons besoin que de deux valeurs. Pourquoi ne pas utiliser un simple bit, signifiant alors "compacté" s'il est à 1 et "non compacté" s'il est à zéro. Dans ce cas, il nous reste 7 bits de l'octet. Nous pouvons alors en profiter pour simplifier le décompactage des lignes ou leur copie non compactée. En effet, si la ligne est compactée, nous avons un nombre inconnu de couples d'octets à lire. Ce nombre dépend de la ligne compactée et du nombre de groupes d'octets s'y trouvant. Il est possible de coder ce nombre de couples dans l'octet préliminaire, dans les 7 bits restants. Cela nous autorise à placer jusqu'à 127 couples d'octets dans une ligne, ce qui n'aura jamais lieu : au pire, la ligne occupe toute une ligne d'écran, ce qui représente 80 octets au maximum. Nous aurions alors 80 couples (et encore ne s'agit-il que d'une vue de l'esprit : dans un tel cas, la ligne ne serait pas compactée !).

Par soucis d'homogénéité, nous pouvons placer LAR dans l'octet préliminaire si la ligne n'est pas compactée. Cela est presque inutile puisque nous connaissons a priori LAR. Le dessin sera donc mémorisé sous la forme suivante :

☐ **Pour chaque ligne :**

- si bit 7 du premier octet=1, ligne compactée : la valeur représentée par les 7 bits de droite est le nombre de couples d'octets à décompacter. Une fois ces couples sautés, la ligne suivante commence ;
- sinon bit7=0, il s'agit d'une ligne non compactée. Copier les LAR octets suivant directement sur l'écran. Après eux commence la ligne suivante.

☐ **Ligne suivante.**

ALGORITHME DE COMPACTAGE

Il nous faut maintenant définir la première routine plus précisément. Elle se chargera de compacter un dessin de l'écran en mémoire. En entrée, elle doit posséder les mêmes renseignements que son homologue du chapitre précédent : BUF, adresse de destination de l'objet, LAR, nombre d'octets de largeur, HAU, nombre de lignes, et ECRAN, adresse du coin en haut à gauche du dessin sur l'écran. En sortie, BUF sera positionné sur le premier octet suivant l'objet.

L'algorithme est le suivant :

- initialiser le pointeur de destination de l'objet ;
- initialiser le pointeur du dessin original ;
- pour HAU lignes :
 - positionner le bit 7 de l'octet préliminaire actuel (par défaut, la ligne est compactée) ;
 - positionner le pointeur destination sur le deuxième octet (c'est-à-dire le début des couples de données, immédiatement après l'octet préliminaire) ;
 - compteur de couples=0 ;
 - compteur d'encombrement=0 ;
 - octet de référence=premier octet du dessin ;
 - compteur de répétition=0.
 - pour LAR octets :
 - incrémenter le compteur d'encombrement.
 - si l'octet écran=octet de référence :
 - incrémenter le compteur de répétition.
 - passer à l'octet écran suivant de la ligne ;
 - si cet octet diffère d'octet de référence :
 - stocker compteur répétition dans la destination ;
 - stocker octet référence derrière ;
 - avancer le pointeur destination ;
 - octet référence=nouvel octet écran ;
 - incrémenter le compteur de lectures ;
 - compteur de répétition=0 ;
 - incrémenter le compteur d'encombrement.
 - fin boucle sur LAR.
 - mettre compteur de lectures dans les sept bits de droite de l'octet préliminaire.
 - se replacer sur l'octet préliminaire en destination sans perdre la valeur finale de la boucle LAR, qui représente le début de la ligne suivante en destination si la ligne actuelle est compactée.
 - si compteur encombrement>=LAR :
 - enlever le bit7 de l'octet préliminaire, la ligne ne doit pas être compactée ;
 - mettre LAR dans les sept autres bits de cet octet ;
 - copier directement la ligne écran dans la destination (LAR octets) ;
 - remettre à jour le pointeur destination : ancien+LAR+1 ;

- sinon, la destination est déjà au point, il faut juste remettre à jour le pointeur destination.
 - passer à la ligne suivante de l'écran.
- ☐ fin boucle sur HAU.
- ☐ fin de la routine, remettre à jour BUF d'après la valeur actuelle du pointeur destination.

Nous devons également faire une liste des variables nécessaires :

- **BUFAC** sera notre pointeur destination. Au départ, il prendra la valeur de BUF et nous travaillerons avec BUFAC ; remettons BUF à jour après le traitement de chaque ligne. C'est une variable 16 bits.
- **BUF** est le pointeur initial de destination. Comme BUFAC, il s'agit d'une variable 16 bits.
- **LIGNE** va être l'équivalent de BUFAC pour l'écran : nous y rangerons l'adresse écran de la ligne en cours de traitement.
- **ECRAN**, enfin, (dernière variable 16 bits) est la donnée de départ indiquant l'adresse du dessin.

Nous aurons aussi comme variables 8 bits :

- **COUNT**, compteur d'encombrement ;
- **REPEAT**, compteur de répétition ;
- **OCTREF**, octet de référence ;
- **LECTUR**, compteur de lecture ;
- **LAR et HAU**, données initiales largeur et hauteur du dessin.

Notons l'usage qui sera fait de certains registres lors des travaux : IX pointe sur COUNT, ce qui permet d'accéder aux autres variables simplement par adressage indexé. Pour incrémenter le compteur de lecture, on utilisera par exemple une simple instruction "INC (IX+3)" au lieu de la séquence classique "LD A, (LECTUR)"—"INC A"—"LD (LECTUR),A" qui est plus lente, encombrante, et nous ferait perdre le contenu du registre A. DE sera le pointeur écran de travail sur la ligne en cours de traitement. BC sera utilisé à chaque fois que l'on aura besoin d'un compteur. L'ancien BC sera bien entendu sauvegardé auparavant s'il ne doit pas être perdu.

Le programme 5.1 est la traduction en LM de l'algorithme. Largement commenté, il vous suffira de le suivre comparativement à l'algorithme pour comprendre son fonctionnement (listing assembleur 5.1).

```

10 ;
20 ;programme de compactage graphique
30 ;Entrees:variables ECRAN,HAU,LAR et BUF
40 ;Compacte l'image designee dans le buffer
50 ;(programme 5.1)
60 ;
4500 70      ORG  #4500
80 ;
```

4500	2AA045	90	ENTREE: LD	HL, (ECRAN)	;adresse du dessin sur ecran
4503	229E45	100		LD (LIGNE), HL	;debut de ligne
		110	;		
		120	;mise en place boucle pour une ligne ecran		
		130	;		
4506	3AA745	140	LD	A, (HAU)	
4509	47	150	LD	B, A	;nombre de lignes
450A	DD21A245	160	LD	IX, COUNT	;debut table variables 8 bits
		170	;		
450E	C5	180	NEWLIN: PUSH	BC	;sauve compteur lignes
450F	2A9C45	190	LD	HL, (BUF)	
4512	23	200	INC	HL	;HL pointe destination des donnees
4513	AF	210	XOR	A	
4514	32A345	220	LD	(REPEAT), A	;compteur REPEAT
4517	32A545	230	LD	(LECTUR), A	;compteur lectures
451A	32A245	240	LD	(COUNT), A	;compteur COUNT
451D	ED5B9E45	250	LD	DE, (LIGNE)	
4521	1A	260	LD	A, (DE)	;1er octet de la ligne/dessin
4522	32A445	270	LD	(OCTREF), A	;=octet de reference
4525	3AA645	280	LD	A, (LAR)	
4528	47	290	LD	B, A	;nombre d'octets sur ligne
		300	;		
		310	;etude de l'octet actuel		
		320	;		
4529		330	NEWOCT:		
4529	1A	340	LD	A, (DE)	;octet ecran
452A	DDBE02	350	CP	(IX+2)	;=octet reference ?
452D	C23345	360	JP	NZ, OCTSUI	;non:octet suivant.
4530	DD3401	370	INC	(IX+1)	;repeat=repeat+1
4533	13	380	OCTSUI: INC	DE	;avance dans ligne ecran
4534	78	390	LD	A, B	;compteur octets
4535	FE01	400	CP	1	;dernier octet ?
4537	CA4145	410	JP	Z, COMPAC	;oui:compacter
453A	1A	420	LD	A, (DE)	;prend l'octet dessin
453B	DDBE02	430	CP	(IX+2)	;=octet ref ?
453E	CA5C45	440	JP	Z, FINOCT	;oui:continuer boucle
		450	;		
		460	;compactage des octets		
		470	;		
4541	3AA345	480	COMPAC: LD	A, (REPEAT)	;repeat
4544	77	490	LD	(HL), A	;place dans buffer
4545	23	500	INC	HL	;avance buffer
4546	3AA445	510	LD	A, (OCTREF)	;octet ref
4549	77	520	LD	(HL), A	;place dans buffer
454A	23	530	INC	HL	;avance buffer
454B	1A	540	LD	A, (DE)	;octet ecran
454C	32A445	550	LD	(OCTREF), A	;nouvelle reference
454F	DD3403	560	INC	(IX+3)	;lectures=lectures+1


```

4552 DD3400      570      INC (IX+0)
4555 DD3400      580      INC (IX+0)           ;count=count+2
4558 AF          590      XOR A               ;A=0
4559 32A345      600      LD (REPEAT),A       ;repeat=0
                        610 ;
455C 10CB        620 FINOCT: DJNZ NEWOCT      ;suite etude ligne
                        630 ;
                        640 ;fin de la ligne. RAJ compteurs et pointeurs
                        650 ;
455E 229A45      660      LD (BUFAC),HL       ;sauve valeur fin BUFAC
4561 2A9C45      670      LD HL,(BUF)        ;debut bufac
4564 3AA545      680      LD A,(LECTUR)      ;nombre de lectures
4567 CBFF        690      SET 7,A            ;compacte, par defaut
4569 77          700      LD (HL),A          ;dans codage de ligne
456A 3AA245      710      LD A,(COUNT)      ;longueur compactage
456D DBE04       720      CP (IX+4)          ;compare a LAR
4570 DA8545      730      JP C,FINLIN        ;ok pour compactage
                        740 ;
                        750 ;ne pas compacter, ce n'est pas interressant.
                        760 ;
4573 3AA645      770      LD A,(LAR)         ;nombre d'octets
4576 77          780      LD (HL),A          ;b7=0,non compacte.
4577 23          790      INC HL
4578 EB          800      EX DE,HL           ;DE pointe datas dans buffer
4579 2A9E45      810      LD HL,(LIGNE)      ;debut de la ligne
457C 4F          820      LD C,A
457D 0600        830      LD B,0            ;BC=LAR=nombre d'octets
457F EDB0        840      LDIR              ;transfert dans buffer
4581 ED539A45    850      LD (BUFAC),DE      ;fin de ligne en buffer
                        860 ;
4585 2A9A45      870 FINLIN: LD HL,(BUFAC)
4588 229C45      880      LD (BUF),HL        ;RAJ pointeur buffer
458B 2A9E45      890      LD HL,(LIGNE)
458E CD26BC      900      CALL #BC26         ;passe ligne plus bas
4591 229E45      910      LD (LIGNE),HL
4594 C1          920      POP BC             ;recupere compteur lignes
4595 05          930      DEC B              ;NEWLIN est trop eloigne pour
                                           djnz
4596 C20E45      940      JP NZ,NEWLIN       ;suite dessin
4599 C9          950      RET
                        960 ;
                        970 ;variables
                        980 ;
459A 0000        990 BUFAC: DEFW 0
459C 0000      1000 BUF: DEFW 0
459E 0000      1010 LIGNE: DEFW 0
45A0 0000      1020 ECRAN: DEFW 0
45A2 00        1030 COUNT: DEFB 0
45A3 00        1040 REPEAT: DEFB 0

```



```

45A4 00      1050 OCTREF: DEFB 0
45A5 00      1060 LECTUR: DEFB 0
45A6 00      1070 LAR:   DEFB 0
45A7 00      1080 HAU:   DEFB 0

```

Pass 2 errors: 00

Le programme Basic 5.1 met en application la routine.

```

10 *****
20 ** Programme 5.1 **
30 *****
40 '
50 'compactage d'un objet graphique
60 'programme 5.1
70 '
80 MEMORY &2FFF
90 DEFINT a-z
100 ad=&4500:lign=200
110 ctrl=0:READ c$:IF c$="fin" THEN 300
120 FOR i=1 TO LEN(c$) STEP 2
130 c=VAL("&" + MID$(c$,i,2))
140 POKE ad,c:ad=ad+1:ctrl=ctrl+c
150 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
160 lign=lign+10:GOTO 110
170 '
180 DATA 2AA045229E453AA74547DD21A245C52A, 1621
190 DATA 9C4523AF32A34532A54532A245ED5B9E, 1768
200 DATA 451A32A4453AA645471ADDBE02C23345, 1495
210 DATA DD34011378FE01CA41451ADDBE02CA5C, 1737
220 DATA 453AA34577233AA44577231A32A445DD, 1488
230 DATA 3403DD3400DD3400AF32A34510CB229A, 1465
240 DATA 452A9C453AA545CBFF773AA245DDBE04, 1909
250 DATA DA85453AA6457723EB2A9E454F0600ED, 1693
260 DATA B0ED539A452A9A45229C452A9E45CD26, 1755
270 DATA BC229E45C105C20E45C9, 1125
280 DATA "fin"
290 '
300 MODE 0:INK 0,0:INK 1,10:INK 2,15:INK 3,20:IN
    K 4,17:INK 5,26
310 READ larg,haut
320 FOR h=1 TO haut
330 FOR l=1 TO larg
340 READ colo:PLOT 8+(l-1)*4,396-(h-1)*2,colo
350 NEXT
360 NEXT
370 '
380 DATA 24,26
390 DATA 0,0,0,0,0,0,0,0,0,0,0,1,1,2,2,0,0,0,0,
    0,0,0,0
400 DATA 0,0,0,0,0,0,0,0,0,0,0,1,1,2,2,2,2,0,0,0,
    0,0,0,0

```

```

410 DATA 0,0,0,0,0,0,0,0,0,0,1,1,2,2,2,2,2,2,0,0,0
    ,0,0,0,0
420 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,2,0,0,0
    ,0,0,0,0
430 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,2,2,0,0
    ,0,0,0,0
440 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,2,2,2,2,0,0
    ,0,0,0,0
450 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,2,2,2,0
    ,0,0,0,0
460 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,2,2,2,0
    ,0,0,0,0
470 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,2,2,2,2,0
    ,0,0,0,0
480 DATA 0,0,0,0,3,3,1,1,1,1,2,2,2,2,2,2,2,2,2,3
    ,0,0,0,0
490 DATA 0,0,3,3,3,3,1,1,1,1,2,2,2,2,2,2,2,2,2,3
    ,3,3,0,0
500 DATA 0,3,3,3,4,4,1,1,1,1,2,2,2,2,2,2,2,2,2,4
    ,4,3,3,0
510 DATA 3,3,3,4,4,0,1,1,1,2,2,2,2,2,2,2,2,2,2,0
    ,4,4,3,3
520 DATA 3,3,4,4,4,0,1,1,1,2,2,2,2,2,2,2,2,2,2,0
    ,0,4,4,3
530 DATA 3,4,4,4,0,0,1,1,1,2,2,2,2,2,2,2,2,2,2,0
    ,4,4,4,3
540 DATA 3,3,4,4,4,0,1,1,1,2,2,2,2,2,2,2,2,2,2,0
    ,4,4,4,3
550 DATA 0,3,3,4,4,4,1,1,1,1,2,2,2,2,2,2,2,2,2,4
    ,4,4,3,3
560 DATA 0,3,3,3,4,4,4,1,1,1,2,2,2,2,2,2,2,2,2,4,4
    ,4,3,3,3
570 DATA 0,0,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
    ,3,3,3,0
580 DATA 0,0,0,0,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,3,3
    ,3,3,0,0
590 DATA 0,0,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3
    ,3,0,0,0
600 DATA 0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,0,0
    ,0,0,0,0
610 DATA 0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,2,2,0,0
    ,0,0,0,0
620 DATA 0,0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,0,0,0
    ,0,0,0,0
630 DATA 0,0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,0,0,0,0
    ,0,0,0,0
640 DATA 0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,2,2,0,0,0,0,0
    ,0,0,0,0
650 '
660 LOCATE 10,10
670 hau=28:lar=14
680 POKE &45A0,0:POKE &45A1,&C0:'param. adresse
    ecran
690 POKE &45A7,hau:' Param hauteur
700 POKE &45A6,lar:' param largeur
710 POKE &459C,0:POKE &459D,&30:'adresse buffer
    dest.
720 CALL &4500:' compactage
730 PRINT"Une touche pour":PRINT"continuer ..."
740 WHILE INKEY$="":WEND
750 '

```

```

760 'Affichage buffer compacte
770 '
780 MODE 2:INK 1,24:ad=&3000: '      debut buf
    fer
790 FOR i=1 TO hau: '              nombre d
    e lignes
800 PRINT HEX$(PEEK(ad),2)";";: '    octet de
    codage
810 IF (PEEK(ad)AND 128)=128 THEN 910: 'si oui,co
    mpacte
820 '
830 'affichage ligne non compactee
840 '
850 AD=AD+1:FOR j=1 TO PEEK(ad-1): '    nombre d'
    octets
860 PRINT HEX$(PEEK(ad),2) "-";:ad=ad+1
870 NEXT:PRINT:GOTO 940
880 '
890 'affichage ligne compactee
900 '
910 ad=ad+1:FOR j=1 TO PEEK(ad-1)-128: nombre d'
    octets
920 PRINT HEX$(PEEK(ad),2) "+"HEX$(PEEK(ad+1),2) "
    ";
930 ad=ad+2:NEXT:PRINT
940 NEXT:PRINT "BUFFER COMPACTE : "AD-&3000:PRINT
    "DESSIN ORIGINAL : "hau*lar
950 SAVE"dessins.cmp",b,&3000,ad-&3000

```

Il dessine une planète dotée d'un anneau, et compacte cet objet en mémoire. Les données affichées ensuite sont celles de l'objet compacté. Le premier octet de chaque ligne est l'octet préliminaire. S'il est supérieur à \$80 compris, la ligne est compactée ; sont alors affichés les couples d'octets de la ligne. Sinon, tous les octets sont présentés en chaîne, la ligne n'étant pas compactée. Enfin, le programme sauve l'objet compacté dans le fichier DESSINS.CMP et affiche le gain d'octets obtenu.

DÉCOMPACTAGE

Il nous faut maintenant réaliser la routine inverse, qui affichera un objet d'après son image compactée en mémoire.

La logique de la procédure est plus simple que la précédente. En effet, au début de la ligne, nous savons si celle-ci est compactée ou non. Ce n'était pas le cas : nous ne savions pas, lors du codage, si le compactage d'une ligne était intéressant ou non, et il fallait le tester. Ici, la démarche à suivre est très compréhensible :

- ☐ initialiser le pointeur écran (destination) ;
- ☐ initialiser le pointeur objet (origine) ;

□ pour HAU lignes :

- si bit 7 de l'octet préliminaire =1 :
 - remettre ce bit à zéro pour voir le nombre de couples à lire ;
 - pour ce nombre de couples :
 - nombre d'octets=1^{er} octet du couple ;
 - contenu=2^e octet du couple ;
 - copier N fois le contenu dans l'écran ;
 - passer au couple suivant de l'origine ;
 - fin boucle sur nombre de couples.
- sinon, bit7=0, la ligne n'est pas compactée :
 - nombre d'octets=octet préliminaire ;
 - transférer directement ces octets à l'écran ;
- passer à la ligne suivante de l'écran.

□ fin boucle sur les lignes.

Le programme assembleur 5.2 est la traduction en langage machine de cet algorithme simple. Sa compréhension ne pose pas de problème particulier.

```

10 ;
20 ;Programme de decompactage graphique
30 ;utilise le format genere par programme compacteur
40 ;programme 5.2
50 ;
4600      60      ORG #4600
70 ;
80 ; ENTREES :
90 ; TABLE = table a afficher
100 ; ECRAN = debut ou l'on doit afficher
110 ; HAU = nombre de lignes du dessin
120 ;
4600 2A4146 130 ENTREE: LD HL,(ECRAN)
4603 224346 140      LD (LIGNE),HL
4606 ED5B4546 150      LD DE,(TABLE)
160 ;
170 ;mise en place boucle ligne
180 ;
460A 3A4746 190      LD A,(HAU)
460D 47      200      LD B,A ;nombre de lignes
210 ;
460E C5      220 BOU1: PUSH BC ;sauve compteur de lignes
460F 1A      230      LD A,(DE)
4610 CB7F    240      BIT 7,A ;compactee ?
4612 C22046 250      JP NZ,DECOMP ;il faut decompacter la ligne
260 ;
270 ; si on arrive ici 1er octet <> 00 donc table non compactee
280 ; on fait donc un transfert direct d'octets
290 ;

```

```

4615          300 NOCOMP:
4615 4F        310          LD  C,A          ;nombre octets a copier
4616 0600      320          LD  B,0          ;passe dans BC
                                     330 ;HL pointe deja sur l'ecran
4618 13        340          INC  DE          ;DE sur les donnees de la ligne
4619 EB        350          EX  DE,HL        ;pour utiliser LDIR
461A EDB0      360          LDIR             ;transfert du bloc de donnees
461C EB        370          EX  DE,HL        ;RAJ DE=pointeur buffer
461D C33446    380          JP   FINLIN      ;suite boucle lignes
                                     390 ;
                                     400 ; sous programme traitant les donnees compactees
                                     410 ;
4620 CBBF      420 DECOMP: RES 7,A          ;nombre de couple de valeurs
4622 47        430          LD  B,A          ;compteur DJNZ
4623 13        440          INC  DE          ;avancer sur bloc de donnees
4624          450 BOU2:
4624 C5        460          PUSH BC          ;sauver compteur lectures
4625 1A        470          LD  A,(DE)       ;repeat
4626 47        480          LD  B,A          ;dans B pour djnz
4627 13        490          INC  DE          ;
4628 1A        500          LD  A,(DE)       ;valeur de l'octet
4629 13        510          INC  DE          ;avance pour donnee suivante
462A 77        520 COPIE: LD  (HL),A        ;copie sur l'ecran
462B 23        530          INC  HL
462C 10FC      540          DJNZ COPIE
462E C1        550          POP  BC          ;recupere compteur lectures
462F 10F3      560          DJNZ BOU2        ;suite de la ligne
4631 C33446    570          JP   FINLIN      ;fin du travail
                                     580 ;
                                     590 ; sous prog de sortie qui recupere les parametres et qui rend
                                     600 ; la main a l'appelant
                                     610 ;
4634          620 FINLIN:
4634 C1        630          POP  BC          ;recupere compteur de lignes
4635 2A4346    640          LD  HL,(LIGNE)   ;retour debut ligne ecran
463B CD26BC    650          CALL #BC26      ;passe ligne en dessous
463B 224346    660          LD  (LIGNE),HL
463E 10CE      670          DJNZ BOU1        ;ligne suivante
                                     680 ;
4640 C9        690          RET
                                     700 ;
                                     710 ;variables du programme
                                     720 ;
4641 0000      730 ECRAN:  DEFW 0
4643 0000      740 LIGNE:  DEFW 0
4645 0000      750 TABLE: DEFW 0
4647 00        760 HAU:    DEFB 0

```

Seule l'instruction "RES 7,A" n'a pas encore été décrite. Son rôle est de remettre à zéro un bit. Ici, il s'agit du bit 7 (le plus à gauche) du registre A.

Enfin, ultime remarque, vous pouvez constater que la routine utilise très peu de variables système. Cela est dû à la simplicité de l'algorithme : il y a suffisamment peu de données à manipuler pour qu'aucun conflit de registre ne se pose.

Le programme Basic 5.2 propose la mise en œuvre de la routine. Il utilise le fichier DESSINS.CMP créé par le programme 5.1. Ce fichier contient l'image compactée de la planète.

```

10 '*****
20 '** Programme 5.2 **
30 '*****
40 '
50 'decompactage d'un objet graphique
60 'programme 5.2
70 '
80 MEMORY &2FFF
90 ad=&4600:lign=200
100 ctrl=0:READ c$:IF c$="fin" THEN 240
110 FOR i=1 TO LEN(c$) STEP 2
120 c=VAL("&" + MID$(c$,i,2))
130 POKE ad,c:ad=ad+1:ctrl=ctrl+c
140 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
150 lign=lign+10:GOTO 100
160 '
170 DATA 2A4146224346ED5B45463A474647C51A, 1308
180 DATA CB7FC220464F060013EBEDB0EBC33446, 1930
190 DATA CBBF4713C51A47131A13772310FCC110, 1473
200 DATA F3C33446C12A4346CD26BC22434610CE, 1756
210 DATA C9, 201
220 DATA "fin"
230 '
240 '
250 MODE 0:MEMORY &2FFF:LOAD"dessins.cmp",&3000
260 hau=28
270 INK 0,0:INK 1,10:INK 2,15:INK 3,20:INK 4,7
280 POKE &4645,0:POKE &4646,&30:'buffer origine
290 POKE &4647,hau:'          nombre de ligne
    S
300 i=0:ad=49232:WHILE i<80:ad=ad-1:i=i+1
310 POKE &4641,ad-256*INT(ad/256):POKE &4642,INT
    (ad/256)

```



```
320 FOR k=1 TO 20:NEXT
330 CALL &4600: '      affichage decom
      pacte
340 WEND
350 LOCATE 1,15
```

Bien évidemment, la routine 5.2 ainsi réalisée ne peut traiter que des images compactées par la routine 5.1. Une remarque s'impose : son efficacité est proportionnelle à la quantité de motifs horizontaux uniformes présents dans le dessin. Si celui-ci comporte principalement des motifs verticaux, il vaut mieux le coder simplement (avec les routines du chapitre 4), ou, mieux, réaliser un compacteur travaillant par colonne plutôt que par ligne. Ce n'est pas excessivement compliqué si vous avez correctement assimilé le fonctionnement de ce chapitre : seule la progression sur l'écran change, les principes de codage et de traitement restant valables. Cela constitue un excellent exercice que nous vous laisserons, pour une fois, affronter.

DÉPLACEMENTS PAR CALCUL D'ADRESSES | **6**

GESTION DU JOYSTICK

Lors des deux chapitres précédents, vous avez pu remarquer que le déplacement des objets s'obtenait facilement en modifiant l'adresse de localisation sur l'écran. En l'occurrence, en ajoutant 1, il se produisait un mouvement vers la droite, et vers la gauche pour -1.

Les déplacements horizontaux sont simples, mais tout change lorsque nous nous penchons sur le problème des mouvements verticaux. En effet, le déplacement d'une ligne vers le haut ou vers le bas n'est pas immédiat. La structure de la mémoire écran s'oppose à tout calcul simple. Nous allons donc réaliser une routine qui, suivant l'état du joystick, modifiera correctement l'adresse de destination en mémoire écran.

Examinons tout d'abord la gestion du joystick. Celle-ci est grandement facilitée par la présence d'une routine système dont le vecteur est placé en \$BB24. L'appel de cette routine range l'état du Joystick 0 dans le registre A, de la façon suivante : bit 3 positionné à 1 si action du joueur vers la droite, bit 2 si vers la gauche, bit 1 si vers le bas, et bit 0 si vers le haut. Notons également les bits 4 et 5 restituant de façon similaire l'état des deux boutons de tir.

Les combinaisons de bits sont bien sûr possibles : si le mouvement choisi est en diagonale vers la gauche en haut, les bits 0 et 2 seront tous deux positionnés. On obtient donc 16 valeurs grâce à ces quatre bits, provoquant un mouvement bien précis. Le tableau ci-dessous résume ces valeurs.

b3b2b1b0	Valeur DEC/HEX	Mouvement correspondant
0 0 0 0	0/\$00	--
0 0 0 1	1/\$01	vers haut
0 0 1 0	2/\$02	vers bas
0 0 1 1	3/\$03 *	--
0 1 0 0	4/\$04	vers gauche
0 1 0 1	5/\$05	vers gauche+haut (DIAG1)
0 1 1 0	6/\$06	vers gauche+bas (DIAG2)
0 1 1 1	7/\$07 *	vers gauche
1 0 0 0	8/\$08	vers droite
1 0 0 1	9/\$09	vers droite+haut (DIAG3)
1 0 1 0	10/\$0A	vers droite+bas (DIAG4)
1 0 1 1	11/\$0B *	vers droite
1 1 0 0	12/\$0C *	--
1 1 0 1	13/\$0D *	vers haut
1 1 1 0	14/\$0E *	vers bas
1 1 1 1	15/\$0F *	--

Toutefois, certaines valeurs ne correspondent à aucune réalité physique : pour produire par exemple la valeur 7, il faudrait placer le manche à la fois vers la gauche, le haut et le bas. Les valeurs ainsi inutiles sont marquées d'une étoile dans le tableau.

Il est cependant plus simple de traiter l'état du joystick par l'intermédiaire d'une table contenant ces 16 valeurs. En effet, on peut placer dans une table une adresse de routine associée à chaque code joystick. Les routines seraient les suivantes :

Valeur	Routine	Mouvement correspondant
0	NOP	routine ne faisant rien du tout
1	HAUT	déplacement vers le haut
2	BAS	déplacement vers le bas
3	NOP	rien du tout
4	GAUCHE	vers la gauche
5	DIAG1	vers la gauche et en haut
6	DIAG2	vers la gauche et en bas
7	GAUCHE	vers la gauche
8	DROITE	vers la droite
9	DIAG3	vers la droite et en haut
10	DIAG4	vers la droite et en bas
11	DROITE	vers la droite
12	NOP	rien
13	GAUCHE	vers la gauche
14	DROITE	vers la droite
15	NOP	rien

Si nous plaçons les adresses de chacune de ces routines dans la table (2 octets par adresse), l'adresse écran originale dans HL (à modifier si le joystick indique un déplacement) et le code joystick dans A, nous pouvons sauter à la routine voulue par la séquence suivante :

AND \$OF : ceci permet de ne garder que les quatre bits de droite du code joystick, et d'obtenir la valeur entre 0 et 15 ;

RLA : ceci décale A vers la gauche d'un bit, recopiant le Carry dans le bit 0. Ce dernier vaut zéro car le AND précédent positionne toujours le Carry à zéro. RLA est donc alors équivalent à une multiplication par 2 du registre A. Ceci donne la position relative de l'adresse de notre routine par rapport au premier octet de la table. Si le code joystick était 0, nous obtenons 0. Si c'était 3, le résultat est 6 (l'adresse à récupérer occupe alors les 6^e et 7^e octets de la table) ;

LD	E,A	
LD	D,0	: ceci place le résultat précédent dans le registre DE afin de pouvoir l'additionner facilement à l'adresse de début de la table ;
LD	IX,TABLE	: on place dans le registre 16 bits IX l'adresse de début de la table ;
ADD	IX,DE	: on obtient ici l'adresse où se situe l'adresse de la routine voulue. Il faut encore la récupérer ;
LD	E,(IX+0)	
LD	D,(IX+1)	: ce qui est fait : l'adresse de la routine se trouve dans DE ;
PUSH	DE	
POP	IX	: on la passe à IX (car l'instruction JP (DE) n'existe pas) ;
JP	(IX)	: et on saute enfin à la routine, l'adresse écran originale étant toujours dans HL.

DÉPLACEMENTS PAR CALCUL D'ADRESSES

Il nous reste enfin à programmer chacune des routines de mouvement. Les diagonales sont les plus simples, par exemple, DIAG1 sera constituée comme suit :

```
DIAG1: CALL GAUCHE
      JP    HAUT
```

Il s'agit d'une simple décomposition en deux mouvements linéaires.

Les déplacements horizontaux ne posent pas de problèmes non plus, excepté celui des bords de l'écran. En effet, un déplacement vers la gauche à partir du bord gauche place la nouvelle position à droite, 8 lignes plus haut. Il n'existe malheureusement pas de moyen simple d'éviter cela, sinon en mémorisant la position horizontale en plus de la position écran. Il faut alors vérifier que la position reste entre 0 et 79 (ou \$00 et \$4F). Si ce n'est pas le cas, on peut ou bien inhiber le mouvement (les bords horizontaux sont infranchissables, ou bien intervertir la position (les bords sont cycliques).

Ce test est à effectuer en plus dans les routines GAUCHE et DROITE. Nous n'allons pas les effectuer. En revanche, nous reviendrons sur le problème lors du chapitre 8, lorsque nous mettrons en place un système de coordonnées. GAUCHE et DROITE sont donc simplement constitués d'une instruction DEC HL ou INC HL suivie d'un RET.

Les choses perdent leur aspect simpliste pour HAUT et BAS. La solution simple est d'appeler les routines systèmes \$BC26 et \$BC29. Une autre solution, plus rapide et plus élégante, est d'étudier la structure écran pour trouver un équivalent. Un déplacement vers le bas est assez simple, nous l'avons d'ailleurs déjà évoqué. Il suffit en effet d'ajouter \$800 à l'adresse. Si l'addition 16 bits provoque un Carry, alors il faut rajouter \$C050. La programmation est la suivante, HL contenant l'adresse :

```
HAUT : LD  DE, $800
      ADD HL,DE : première addition
      RET NC   : pas de Carry : le travail est fini
      LD  DE,$C050
      ADD HL,DE : seconde addition
      RET
```

L'explication de ce calcul est plus simple qu'il n'y paraît. Le positionnement du Carry se produit lorsque le résultat de l'addition dépasse \$FFFF, c'est-à-dire le bout de la mémoire écran. C'est justement ce qui permet de distinguer les sauts de ligne spéciaux des autres. Et dans ce cas, il suffit d'ajouter \$C050 pour obtenir la bonne adresse.

Pour un déplacement vers le bas, la logique est la même, bien que la programmation soit un petit peu moins simple. Il suffit de retrancher \$800. Si l'adresse obtenue dépasse le début de la mémoire écran (c'est-à-dire qu'elle se situe avant celui-ci), alors on retranche également \$C050. C'est exactement l'opération inverse de la précédente. Mais un problème se pose : le début de l'écran se situe en \$C000. Pour savoir si l'adresse obtenue se situe avant ou après, il est donc impossible de tester le Carry. Celui-ci ne sera pas spécialement positionné si l'addition produit un nombre entre \$0000 et \$BFFF. Par contre, nous pouvons tester le poids fort de l'adresse. S'il est compris entre \$C0 et \$FF (bornes incluses), alors l'adresse est toujours dans l'écran. Sinon, il faut retrancher \$C050.

Par commodité, nous n'allons pas utiliser la soustraction 16 bits. En effet, seul SBC existe, et il faut donc remettre le Carry à zéro avant chaque soustraction. Nous pouvons par contre ajouter le complément à 1, ce qui revient au même. Voici le programme correspondant :

```
BAS : LD  DE,$F800
      ADD HL,DE : équivaut à soustraire $800
      LD  A,H   : on place le poids fort dans A
      CP  $C0   : comparer
      RET NC   : le poids fort est supérieur ou égal à $C0 et
                  inférieur à $FF, donc l'adresse est bien dans
                  l'écran, le travail est fini.
      LD  DE,$BFO
      ADD HL,DE : équivaut à soustraire $C050.
      RET
```


CONSÉQUENCES DE LA STRUCTURE DE LA MÉMOIRE ÉCRAN

Nous possédons tous les éléments pour réaliser la routine complète. Le programme 6.1 en assembleur les résume. Le programme 6.2 en Basic est une application de cette routine, basée sur le programme 4.4 et permettant le déplacement, grâce au joystick, du petit module extraterrestre du chapitre 4. Tous les mouvements sont autorisés, et cela sans gestion complexe de POKes au niveau du Basic. Notez que ce programme utilise le fichier DESSINS.BIN créé au chapitre 4 (par le programme 4.3), ainsi que la routine 4.1 de restitution d'objet.

```

10 ;
20 ;Programme de deplacement par joystick
30 ;programme 6.1
40 ;Entree:
50 ;      la variable ECRAN contient l'adresse ecran
60 ;      a recalculer en fonction de l'etat du joystick
70 ;
80 ECRAN: EQU #45A0
90 ;
100      ORG #4C00
110      CALL #BB24          ;get joystick state
120      LD HL, (ECRAN)
130      CALL DEPLAC
140      LD (ECRAN), HL
150      RET
160 ;
170 DEPLAC: AND #0F          ;garde 4 bits de droite
180      RLA                ;multiplie par deux
190      LD E, A
200      LD D, 0            ;transfert de l'offset 16 bits
                           ;dans DE
210      LD IX, TABLE
220      ADD IX, DE          ;calcul localisation de
                           ;l'adresse de saut
230      LD E, (IX+0)
240      LD D, (IX+1)        ;recupere adresse de saut
                           ;dans DE
250      PUSH DE
260      POP IX              ;transfert dans IX
270      JP (IX)            ;et saut a la routine
280 ;
290 ;la table de saut
300 ;

```

4C24	774C	310	TABLE:	DEFW	NOP	
4C26	4E4C	320		DEFW	HAUT	
4C28	444C	330		DEFW	BAS	
4C2A	774C	340		DEFW	NOP	
4C2C	5B4C	350		DEFW	GAUCHE	
4C2E	5F4C	360		DEFW	DIAG1	
4C30	654C	370		DEFW	DIAG2	
4C32	5B4C	380		DEFW	GAUCHE	
4C34	5D4C	390		DEFW	DROITE	
4C36	6B4C	400		DEFW	DIAG3	
4C38	714C	410		DEFW	DIAG4	
4C3A	5D4C	420		DEFW	DROITE	
4C3C	774C	430		DEFW	NOP	
4C3E	4E4C	440		DEFW	HAUT	
4C40	444C	450		DEFW	BAS	
4C42	774C	460		DEFW	NOP	
		470	;			
		480	;	les quatres	deplacements elementaires	
		490	;			
4C44	11000B	500	BAS:	LD	DE,#800	
4C47	19	510		ADD	HL,DE	
4C48	D0	520		RET	NC	;toujours dans l'ecran
4C49	1150C0	530		LD	DE,#C050	
4C4C	19	540		ADD	HL,DE	
4C4D	C9	550		RET		
		560	;			
4C4E	1100F8	570	HAUT:	LD	DE,#F800	
4C51	19	580		ADD	HL,DE	
4C52	7C	590		LD	A,H	
4C53	FEC0	600		CP	#C0	
4C55	D0	610		RET	NC	;toujours dans l'ecran
4C56	11B03F	620		LD	DE,#3FB0	
4C59	19	630		ADD	HL,DE	
4C5A	C9	640		RET		
		650	;			
4C5B	2B	660	GAUCHE:	DEC	HL	
4C5C	C9	670		RET		
		680	;			
4C5D	23	690	DROITE:	INC	HL	
4C5E	C9	700		RET		
		710	;			
		720	;	les quatres	diagonales	
		730	;			
4C5F	CD5B4C	740	DIAG1:	CALL	GAUCHE	
4C62	C34E4C	750		JP	HAUT	
		760	;			
4C65	CD5B4C	770	DIAG2:	CALL	GAUCHE	
4C68	C3444C	780		JP	BAS	
		790	;			

```

4C6B CD5D4C 800 DIAG3: CALL DROITE
4C6E C34E4C 810 JP HAUT
      820 ;
4C71 CD5D4C 830 DIAG4: CALL DROITE
4C74 C3444C 840 JP BAS
      850 ;
      860 ;RIEN !
      870 ;
4C77 C9      880 NOP: RET

```

Pass 2 errors: 00

```

10 *****
20 ** Programme 6.2 **
30 *****
40 '
50 'deplacement d'un objet en plusieurs phases
60 'd'apres le codage effectue par programme 4.3
70 'aux adresses $2fff et suite.
80 'application des routines 6.1 et 4.1
90 '
100 MEMORY &2FFF
110 ad=&4700:lign=200
120 ctrl=0:READ c$:IF c$="fin" THEN 230
130 FOR i=1 TO LEN(c$) STEP 2
140 c=VAL("&" + MID$(c$,i,2))
150 POKE ad,c:ad=ad+1:ctrl=ctrl+c
160 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
170 lign=lign+10:GOTO 120
180 '
190 DATA EDSBA0452A9C453AA74547C5D53AA645, 1892
200 DATA 4F0600EDB0EBE1CD26BCEBC110EDC9, 2271
210 DATA "fin"
220 '
230 ad=&4C00:lign=310
240 ctrl=0:READ c$:IF c$="fin" THEN 400
250 FOR i=1 TO LEN(c$) STEP 2
260 c=VAL("&" + MID$(c$,i,2))
270 POKE ad,c:ad=ad+1:ctrl=ctrl+c
280 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
290 lign=lign+10:GOTO 240
300 '
310 DATA CD24BB2AA045CD0D4C22A045C9E60F17, 1725
320 DATA 5F1600DD21244CDD19DD5E00DD5601D5, 1565

```

```

330 DATA DDE1DDE97774C4E4C444C7774C5B4C5F4C, 1926
340 DATA 654C5B4C5D4C6B4C714C5D4C774C4E4C, 1403
350 DATA 444C774C11000819D01150C019C91100, 1129
360 DATA F8197CFEC0D011B03F19C92BC923C9CD, 2218
370 DATA 5B4CC34E4CCD5B4CC3444CCD5D4CC34E, 1874
380 DATA 4CCD5D4CC3444CC9, 990
390 DATA "fin"
400 MODE 0
410 LOAD"image.bin",&C000:'chargement du decor,
    optionnel
420 FOR I=0 TO 15
430 INK I,ASC(MID$("ACLFSPGJOIJXSDZQ",I+1,1))-65
440 'INK I,ASC(MID$("AMTQSVSJRIJXSRVQ",I+1,1))-6
    5 pour monochrome
450 NEXT
460 ecr=50032
470 POKE &45A0,ecr-256*INT(ecr/256):POKE &45A1,I
    NT(ecr/256)
480 POKE &45A6,7:'          largeur en oc
    tets
490 POKE &45A7,10:'          hauteur en n
    ombre de lignes
500 LOAD"dessins",&3000
510 FOR phase=1 TO 6
520 buf(phase)=&3000+(phase-1)*(7*10)
530 NEXT
540 '
550 FOR phase=1 TO 6
560 POKE &459C,buf(phase)-256*INT(buf(phase)/256
    ):POKE &459D,INT(buf(phase)/256)
570 FOR k=1 TO 4
580 CALL &4C00
590 CALL &4700
600 FOR i=1 TO 10:NEXT
610 NEXT
620 NEXT
630 GOTO 550

```

Modifications de 6.2 pour 6.2b

```

460 ecr=50032
480 POKE &45A6,13:'          largeur en o
    ctets
490 POKE &45A7,33:'          hauteur en n
    ombre de lignes
500 LOAD"dessinsB",&3000
520 buf(phase)=&3000+(phase-1)*(13*33)

```

Modifications de 6.2 pour 6.2c

```

460 ecr=59104
480 POKE &45A6,21: '          largeur en o
      ctets
490 POKE &45A7,19: '          hauteur en n
      ombre de lignes
500 LOAD"dessinsC",&3000
520 buf(phase)=&3000+(phase-1)*(21*19)

```

Vous remarquerez un défaut du programme : lorsque le module disparaît ou apparaît par le haut de l'écran, il laisse une trace de son passage. D'autre part, les deux lignes extrêmes (le haut de l'écran et le bas) ne semblent pas correspondre. Ceci est dû aux 384 octets inutilisés de la mémoire écran. En effet, la dernière ligne de l'écran est située à l'adresse \$FF80. Si nous descendons d'une ligne par notre routine de calcul, nous obtenons \$FF80+\$0800 soit \$0780, avec positionnement du Carry. L'addition supplémentaire de \$C050 donne finalement \$C7D0. Or, ce n'est pas du tout l'adresse du début de la première ligne. Pour l'expliquer il faut se pencher sur les octets inemployés de la mémoire écran. En effet, celle-ci contient 200 lignes de 80 octets. Nous avons donc 16 000 octets utilisés pour l'écran. Mais 16 Ko forment 16 384 octets. Il reste donc 384 octets inutilisés. Si l'on respecte la logique de l'entrelacement des lignes, ces 384 octets correspondent, par blocs de 48 octets, à une 201^e, 202^e et jusqu'à une 208^e ligne situées hors de l'écran. Mais ces pseudo-lignes ne contenant que 48 octets faussent les calculs si l'on s'y aventure. Une vraie ligne contient 80 octets. Voilà pourquoi un objet partant vers le bas de l'écran disparaît sur 8 lignes imaginaires, avant de réapparaître en haut de l'écran, un peu plus à gauche. C'est aussi la raison pour laquelle il laisse un trait lors de son passage.

Ces défauts ne sont pas faciles à corriger pour l'instant. Nous en viendrons facilement à bout dans le chapitre 8 en utilisant un système de coordonnées et en interdisant les sorties d'écran.

GESTION DES OBJETS | 7 SUR UN DÉCOR

PROBLÈMES DE DÉPLACEMENT

Toutes les routines que nous avons réalisées jusqu'à présent permettent d'afficher des objets sans tenir compte de ce qui se trouve déjà sur l'écran. Cela interdit la gestion d'un décor intégré à l'action. Il faut donc revoir intégralement le principe d'affichage si nous voulons implémenter une telle possibilité.

Plusieurs solutions s'offrent au programmeur désirant intégrer un décor. Nous verrons dans le chapitre 8 comment interdire aux objets mobiles l'accès à une zone de l'écran. On peut utiliser cette solution pour placer un décor sur l'écran et interdire tout déplacement sur la surface qu'il occupe. De cette façon, tout objet pourra se déplacer simplement (avec les routines du chapitre 4). Mais la partie opérationnelle de l'écran devra être intégralement couleur de fond. Les collisions entre objets risquent également de poser quelques problèmes. La seule solution semble donc d'autoriser les déplacements sur le fond de l'écran.

Le plus gros problème posé est alors le suivant : sachant que l'objet occupe un rectangle sur l'écran et qu'il va écraser totalement le contenu de ce rectangle, comment faire pour le restituer lorsque l'objet se déplacera à nouveau ?

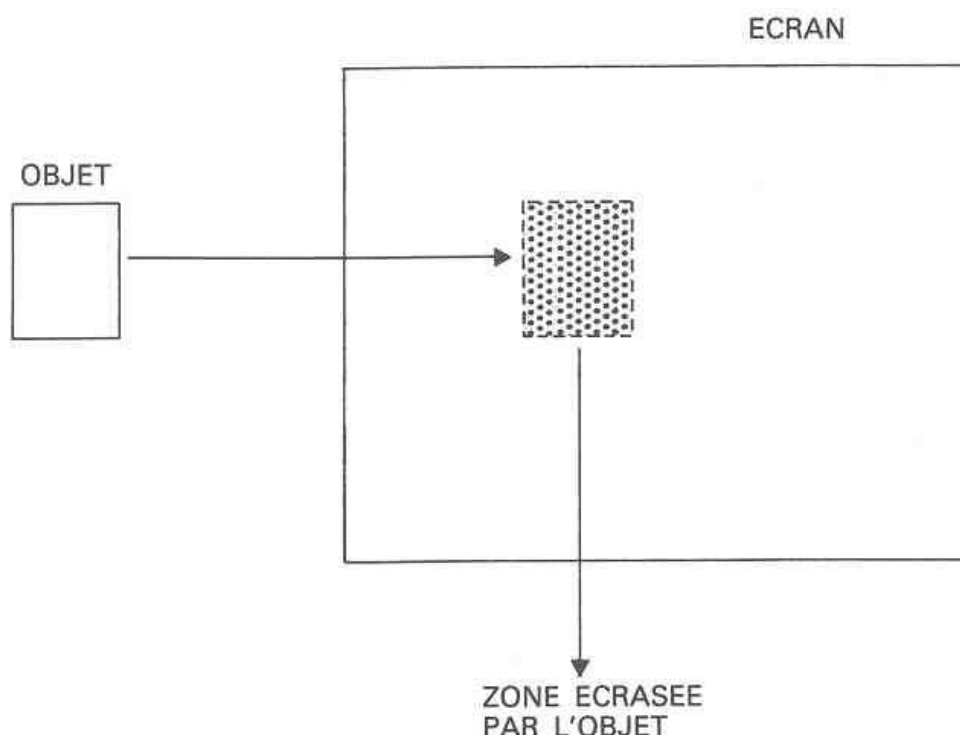


Schéma 7.1

Problème d'écrasement de décor.

Il existe un grand nombre de solutions, et nous allons en étudier deux : le mode XOR et la gestion de transparence.

LE MODE XOR

XOR binaire

Le mode XOR est une astuce très utilisée depuis quelque temps dans les jeux. Il allie une grande facilité d'utilisation à une rapidité de traitement inégalable. Mais, bien évidemment, il possède un gros défaut : il ne conserve pas les couleurs originales de l'objet affiché et du décor lorsque ceux-ci se superposent.

Le principe de fonctionnement du mode XOR est basé sur l'opération binaire du même nom. Le résultat d'un XOR sur deux bits est le suivant : si les deux bits sont identiques, on obtient 0. S'ils sont différents, le résultat est 1. Cette opération a une particularité : $(A \text{ XOR } B) \text{ XOR } B$ donne A . En effet, si le bit original est 0, XOR 0 ne change pas son état et XOR 1 l'inverse. Un second XOR redonne 0. Si le bit original est 1, deux XOR successifs donnent également 1.

Cette particularité est le fondement du mode graphique XOR. Chaque octet de l'objet à placer est composé auparavant avec le contenu de l'écran qu'il va écraser. Puis, lorsqu'un déplacement intervient, on redessine de nouveau l'objet de la même manière avant de le déplacer. Cela constitue, sur 8 bits, l'opération $(\text{ECRAN XOR OBJET}) \text{ XOR OBJET}$, dont le résultat est ECRAN. Sans aucune sauvegarde de l'écran, nous récupérons donc le contenu initial de celui-ci, simplement en dessinant deux fois l'objet à son emplacement.

La routine de dessin en mode XOR sera quasiment identique à celle du chapitre 4, excepté la copie d'une ligne qui se fera octet par octet, en composant chacun d'entre eux avec le contenu de l'écran par l'opération XOR.

Restitution d'objet en mode XOR

Le programme assembleur 7.1 est donc une imitation du 4.1 ; seules quelques lignes changent.

```

10 ;
20 ;programme de copie d'objet
30 ;RAM->Ecran, mode XOR
40 ;programme 7.1
50 ;
4800 60      ORG #4800
70 ;

```

```

80 ;ENTREE: (ECRAN) adresse du coin sup.gauche
90 ;      (BUF)  adresse de l'image
100 ;      (LAR) nombre d'octets par ligne
110 ;      (HAU) nombre de lignes
120 ;
459C 130 BUF: EQU #459C
45A0 140 ECRAN: EQU #45A0
45A6 150 LAR: EQU #45A6
45A7 160 HAU: EQU #45A7
170 ;
4800 2AA045 180 LD HL,(ECRAN)
4803 ED5B9C45 190 LD DE,(BUF)
4807 3AA745 200 LD A,(HAU)
480A 47 210 LD B,A ;compteur de lignes
220 ;
480B C5 230 NEWLIN: PUSH BC ;sauvegarde compteur
480C E5 240 PUSH HL ;sauvegarde adresse ligne
480D 3AA645 250 LD A,(LAR)
4810 47 260 LD B,A ;compteur du nombre d'octets
4811 1A 270 NEWOCT: LD A,(DE) ;recupere octet dessin
4812 AE 280 XOR (HL) ;intersection avec ecran
4813 77 290 LD (HL),A ;et positionnement sur l'ecran
4814 23 300 INC HL
4815 13 310 INC DE
4816 10F9 320 DJNZ NEWOCT
4818 E1 330 POP HL ;ancien debut ligne
4819 CD26BC 340 CALL #BC26 ;descend d'une ligne
481C C1 350 POP BC
481D 10EC 360 DJNZ NEWLIN ;ligne suivante
481F C9 370 RET

```

Pass 2 errors: 00

L'instruction LDIR a notamment disparu au profit d'une boucle s'effectuant LAR fois. Notez également l'inversion des registres DE et HL pour le travail : en effet, avec LDIR, DE pointe la destination et il était donc intéressant de placer le pointeur écran dans ce registre. Ici, il n'en est plus question, nous pouvons donc inverser les deux pointeurs : HL pour l'écran et DE pour l'objet. Cela nous permet de réaliser le XOR de la façon suivante :

```

LD A, (DE) : chargement de l'octet dessin dans le registre A ;
XOR (HL) : XOR avec l'octet de l'écran ;
LD (HL),A : et remise en place du résultat sur l'écran.

```

Nous pouvions bien entendu garder l'attribution originale des registres : dans ce cas, seul "LD (HL),A" se transformait en "LD (DE),A" pour envoyer l'octet à l'écran. Mais cela aurait posé un problème pour le passage à la

ligne suivante par la routine système #BC26. Cette routine travaille en effet sur HL et non DE ; il aurait donc fallu recourir à un "EX DE,HL" après cet appel de routine, exactement comme au chapitre 4. Utiliser HL pour le pointeur écran nous permet d'éviter cette instruction perturbatrice.

Le programme Basic 7.1 est une illustration de la routine. Il déplace notre petit module extraterrestre en mode XOR.

```

10 '*****
20 '** Programme 7.1 **
30 '*****
40 '
50 'déplacement d'un objet en plusieurs phases e
   n mode XOR
60 '
70 MEMORY &2FFF:LOAD"prog6.1ob":'voir annexe 6
80 ad=&4800:lign=160
90 ctrl=0:READ c$:IF c$="fin" THEN 200
100 FOR i=1 TO LEN(c$) STEP 2
110 c=VAL("&" + MID$(c$,i,2))
120 POKE ad,c:ad=ad+1:ctrl=ctrl+c
130 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
140 lign=lign+10:GOTO 90
150 '
160 DATA 2AA045ED5B9C453AA74547C5E53AA645471AAE7
   7, 2298
170 DATA 231310F9E1CD26BCC110ECC9, 1621
180 DATA "fin"
190 '
200 MODE 0
210 LOAD"image.bin",&C000:'chargement du decor,
   optionnel
220 FOR I=0 TO 15
230 INK I,ASC(MID$("ACLFSPGJO[JXSDZQ",I+1,1))-65
240 'INK I,ASC(MID$("AMTQSVSJRL[JXSR YQ",I+1,1))-6
   5 pour monochrome
250 NEXT
260 ecr=50032
270 POKE &45A0,ecr-256*INT(ecr/256):POKE &45A1,I
   NT(ecr/256)
280 POKE &45A6,7:'
   tets
290 POKE &45A7,10:'
   ombre de lignes
300 LOAD"dessins",&3000
310 FOR phase=1 TO 6

```

```

320 buf(phase)=&3000+(phase-1)*(7*10)
330 NEXT
340 '
350 FOR phase=1 TO 6
360 POKE &459C,buf(phase)-256*INT(buf(phase)/256
    ):POKE &459D,INT(buf(phase)/256)
370 FOR k=1 TO 4
380 CALL &4C00
390 CALL &4800
400 FOR i=1 TO 1:NEXT
410 CALL &4800
420 NEXT
430 NEXT
440 GOTO 350

```

Modifications de 7.1 pour 7.1b

```

260 ecr=50032
280 POKE &45A6,13: '          largeur en o
    ctets
290 POKE &45A7,33: '          hauteur en n
    ombre de lignes
300 LOAD"dessinsB",&3000
320 buf(phase)=&3000+(phase-1)*(13*33)

```

Modifications de 7.1 pour 7.1c

```

260 ecr=50032
280 POKE &45A6,21: '          largeur en o
    ctets
290 POKE &45A7,19: '          hauteur en n
    ombre de lignes
300 LOAD"dessinsC",&3000
320 buf(phase)=&3000+(phase-1)*(21*19)

```

Vous constatez que le mode XOR ne touche effectivement pas au décor, au contraire de la routine du chapitre 4. Mais, si vous êtes attentif au déplacement, vous remarquez le défaut évoqué plus haut : lorsqu'une partie de l'objet entre en contact avec le décor, les couleurs du dessin sont étrangement modifiées. Par contre, aucune altération n'a lieu sur les parties couleur de fond, qu'il s'agisse du fond de l'écran ou des parties vides de l'objet (rappelons que celui-ci est inscrit dans un rectangle et comporte donc un certain nombre de zones couleur de fond). L'explication du phénomène est simple : alors que $A \text{ XOR } 0$ donne toujours A , $A \text{ XOR } 1$

donne le bit inverse de A. Nous composons le décor avec l'objet. Si l'un des deux est couleur de fond, tout se passe bien. Par contre, chaque bit à 1 de l'objet va inverser, lors du XOR, le bit correspondant du décor. Le contenu d'un octet étant constitué de masques de couleur, la composition de deux masques non nuls va produire un masque totalement différent. Par exemple, considérons ce qui arrive si nous effectuons une telle opération.

	11111100	(masque indiquant deux points en stylo 7 sur l'objet)
XOR	11000011	(deux points en stylo 9 sur l'écran)
	00111100	ce résultat est placé dans l'écran. Or, il correspond à deux points en stylo 6.

Le mode XOR n'a heureusement que ce seul défaut. Ses avantages le rendent en effet extrêmement souple d'emploi. La plupart des jeux d'action connus et reconnus comme excellents (y compris le plus beau de tous *Sorcery/Sorcery+*) utilise le mode XOR. Il suffit de limiter (grâce aux techniques du chapitre 8) les zones de décor accessibles pour obtenir un résultat acceptable. Toutefois, si les couleurs sont mal choisies, il se peut que le mode XOR produise des horreurs lors des contacts objets/décor ou objet/objet. Par exemple, si un ensemble de points colorés en stylo 7 rencontre un ensemble de points en stylo 9, et que le stylo 6 est associé à l'encre 0 (noir), l'ensemble va disparaître de l'écran ! Ce ne sera que provisoire bien sûr, car un nouveau XOR redonnera le stylo 9.

La précaution qui s'impose alors semble évidente : le stylo 0 doit toujours être pris comme stylo de fond. Il est le seul à ne pas perturber les dessins en mode XOR. De plus, aucun autre stylo ne doit être associé à la même couleur que le stylo 0, afin d'éviter le problème décrit ci-dessus.

Malgré tout cela, vous pouvez retenir la leçon suivante : le mode XOR possède de grandes qualités, le tout est de connaître son défaut et d'en tenir compte, en sachant par avance qu'il faudra limiter les zones de recouvrement avec les décors ou les autres objets.

TRANSPARENCE

Pour les puristes qui ne veulent pas d'interférences XOR, il y a bien entendu une autre solution : la transparence du fond. Elle consiste à traiter l'objet uniquement et non plus la totalité du rectangle qui le contient. Ceci afin d'avoir l'objet uniquement sur le décor. Bien entendu, le problème de la restitution du décor se pose de nouveau. Mais il est facile à résoudre. Si nous supposons que l'objet subit un mouvement, voici la démarche à suivre :

- d'abord replacer le décor là où l'objet se situe pour l'instant ;
- puis mémoriser le décor de l'endroit où va se placer l'objet ;
- enfin, dessiner l'objet à son nouvel emplacement.

La transposition de ce principe en programme suit exactement ce déroulement. La restitution du décor se fait simplement à l'aide de la routine 4.1 du chapitre 4, permettant de restituer un objet. Il faut bien entendu que ce décor ait préalablement été sauvegardé avant le premier affichage. La mémorisation se fait justement avec la routine 4.2. Enfin, il nous reste la dernière partie, qui s'inspire de la routine XOR 7.1 (laquelle provenait du chapitre 4). La légère modification qui doit être incluse est la suivante : si un octet de l'objet est de la couleur du fond, il ne doit pas être affiché. De cette façon, seul l'objet sera placé sur l'écran, parfaitement intégré au décor.

La routine 7.2 suit donc le travail désormais classique des routines 4.1 et 7.1, excepté ce point de détail vite résolu ne nécessitant pas de commentaire supplémentaire.

```

10 ;
20 ;programme de copie d'objet apres deplacement
30 ;RAM->Ecran, FOND TRANSPARENT
40 ;programme 7.2
50 ;
4830 60      ORG #4830
70 ;
80 ;ENTREE: (ECRAN) adresse du coin sup.gauche
90 ;      (BUF)  adresse de l'image
100 ;      (LAR)  nombre d'octets par ligne
110 ;      (HAU)  nombre de lignes
120 ;
459C 130 BUF: EQU #459C
45A0 140 ECRAN: EQU #45A0
45A6 150 LAR: EQU #45A6
45A7 160 HAU: EQU #45A7
170 ;
4830 2AA045 180      LD HL,(ECRAN)
4833 ED5B9C45 190      LD DE,(BUF)
4837 3AA745 200      LD A,(HAU)
483A 47 210      LD B,A ;compteur de lignes
220 ;
483B C5 230 NEWLIN: PUSH BC
483C E5 240      PUSH HL
483D 3AA645 250      LD A,(LAR)
4840 47 260      LD B,A ;nombre d'octets
270 ;
4841 1A 280 NEWOCT: LD A,(DE) ;octet de l'objet
4842 B7 290      OR A ;fond ?
4843 CA4748 300      JP Z,OK ;oui:pas de transfert
4846 77 310      LD (HL),A ;transfere a l'ecran
320 ;

```

4847	23	330	OK:	INC	HL	
4848	13	340		INC	DE	
4849	10F6	350		DJNZ	NEWOC	
484B	E1	360		POP	HL	
484C	CD26BC	370		CALL	#BC26	;ligne plus bas ecran
484F	C1	380		POP	BC	
4850	10E9	390		DJNZ	NEWLIN	
4852	C9	400		RET		

Pass 2 errors: 00

Le programme 7.3 montre l'application de cette routine à nos objets. Malgré la simplicité des routines mises en œuvre, le listing Basic présente une allure inquiétante.

```

10 *****
20 ** programme 7.3 **
30 *****
40 '
50 'Programme illustrant la restitution avec dec
   or garde.
60 '
70 MEMORY &2FFF
80 LOAD"prog6.1ob":LOAD"prog4.1ob":LOAD"prog4.2
   ob":'voir annexe 6
90 ad=&4830:lign=170
100 ctrl=0:READ c$:IF c$="fin" THEN 200
110 FOR i=1 TO LEN(c$) STEP 2
120 c=VAL("&" + MID$(c$,i,2))
130 POKE ad,c:ad=ad+1:ctrl=ctrl+c
140 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
150 lign=lign+10:GOTO 100
160 '
170 DATA 2AA045ED5B9C453AA74547C5E53AA645471AB7C
   A, 2390
180 DATA 474877231310F6E1CD26BCC110E9C9, 1877
190 DATA "fin"
200 MODE 0
210 LOAD"image.bin",&C000:'chargement du decor,
   optionnel
220 FOR I=0 TO 15
230 INK I,ASC(MID$("ACLFSPGJOLJXSDZQ",I+1,1))-65
240 'INK I,ASC(MID$("AMTQSVSJRLJXSRYQ",I+1,1))-6
   5 pour monochrome

```

```

250 NEXT
260 LOAD "dessins",&3000
270 FOR i=1 TO 6
280 c=&3000+(i-1)*(7*10)
290 buf(i,0)=c-256*INT(c/256):buf(i,1)=INT(c/256)
300 NEXT
310 POKE &45A6,7
320 POKE &45A7,10
330 '
340 ad=51369:POKE &45A0,ad-256*INT(ad/256):POKE
    &45A1,INT(ad/256)
350 POKE &459C,0:POKE &459D,&80:CALL &4730:'init
    . sauvegarde decor
360 '
370 FOR p=1 TO 6
380 POKE &459C,0:POKE &459D,&80:CALL &4700
390 CALL &4C00:'deplacement
400 CALL &4770
410 POKE &459C,buf(p,0):POKE &459D,buf(p,1)
420 CALL &4830:FOR i=1 TO 10:NEXT:CALL &BD19
430 NEXT
440 GOTO 370

```

Modifications 7.3 pour 7.3 b

```

260 LOAD "dessinsB",&3000
280 c=&3000+(i-1)*(13*33)
310 POKE &45A6,13
320 POKE &45A7,33
340 ad=51369:POKE &45A0,ad-256*INT(ad/256):POKE
    &45A1,INT(ad/256)
350 POKE &459C,0:POKE &459D,&80:CALL &4730:'init
    . sauvegarde decor
380 POKE &459C,0:POKE &459D,&80:CALL &4700

```

Modifications 7.3 pour 7.3 c

```

260 LOAD "dessinsC",&3000
280 c=&3000+(i-1)*(21*19)
310 POKE &45A6,21
320 POKE &45A7,19
340 ad=51369:POKE &45A0,ad-256*INT(ad/256):POKE
    &45A1,INT(ad/256)
350 POKE &459C,0:POKE &459D,&80:CALL &4730:'init
    . sauvegarde decor
380 POKE &459C,0:POKE &459D,&80:CALL &4700

```

Les POKEs correspondent aux opérations suivantes :

- initialiser LAR et HAU ;
- initialiser ECRAN sur la première position du module ;
- initialiser BUF sur la zone mémoire réservée au stockage du décor (cette zone sera de même taille que l'objet) ;
- CALL routine 4.2 pour sauvegarder le décor.

Puis on entre à l'intérieur de la boucle modifiant l'adresse écran de l'objet :

- initialiser BUF sur la zone de sauvegarde de décor ;
- CALL routine 4.1 pour restituer le décor ;
- initialiser ECRAN sur AD actuelle ;
- CALL routine 4.2 pour sauver le nouveau décor ;
- initialiser BUF sur l'objet ;
- CALL routine 7.2 pour afficher l'objet sur le décor ;
- continuer en changeant l'adresse.

PROBLÈMES DE RAPIDITÉ

Vous constatez néanmoins aisément que l'animation n'est pas très belle. Certes, l'objet se déplace réellement sur le décor. Mais le déplacement le fait flasher légèrement, comme s'il était transparent. Cela est dû à la méthode utilisée. Entre deux affichages consécutifs de l'objet, celui-ci est effacé. La persistance de la vision nous empêche de voir le vide en résultant. Ce vide se mêle aux images de l'objet et produit une sorte d'effet de transparence. La solution de ce problème consiste à laisser beaucoup plus longtemps l'objet affiché qu'effacé. Il existe aussi une autre solution plus astucieuse, plus souple, et surtout plus propre : au lieu d'appliquer le principe en trois étapes vu ci-dessus, on le programme pour chaque octet de l'objet. Il faut alors gérer simultanément deux adresses écran (l'ancienne et la nouvelle) et deux pointeurs en mémoire (celui de l'objet, et celui de la zone de sauvegarde du décor). L'avantage de cette méthode est la disparition du vide entre deux images de l'objet, puisque celui-ci est effacé et réaffiché, octet par octet, et non plus intégralement. En revanche, une routine utilisant quatre pointeurs pose des problèmes de programmation non négligeables sur l'Amstrad, les registres secondaires n'étant pas disponibles. Mais rien n'en interdit la programmation. IX et IY peuvent être utilisés, ainsi que HL et DE.

GESTION DES AVANT-PLANS

Nous savons maintenant comment déplacer un objet sur un décor. Que diriez-vous de le faire passer derrière ? Nous pourrions ainsi simuler une sorte de profondeur de décor.

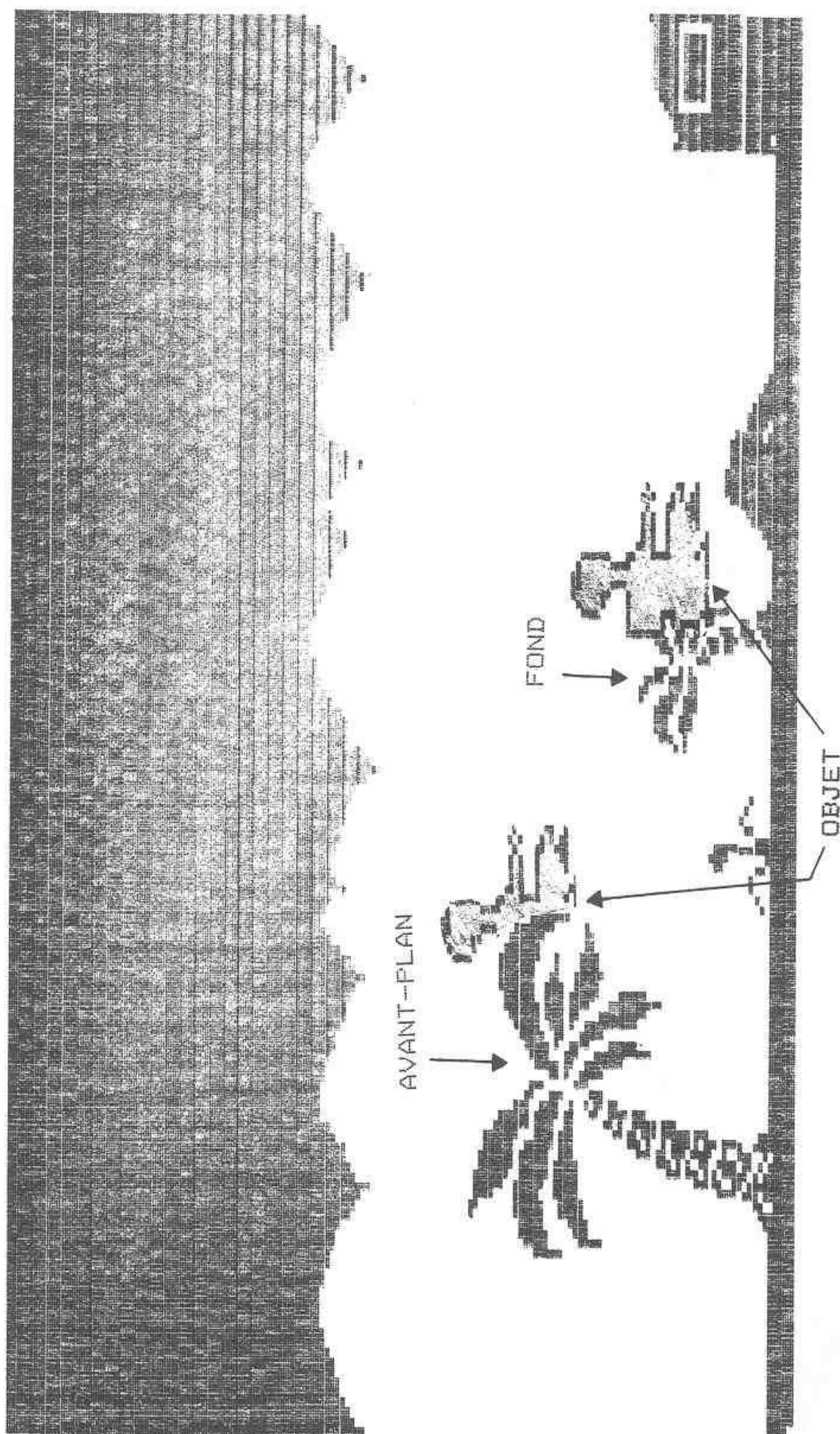


Schéma 7.2

Avant plan et décor.

La gestion de graphismes en trois dimensions est un sujet plus large que notre ouvrage ne peut traiter. Mais nous pouvons, sans trop de difficulté, gérer ce que l'on appelle des "AVANT-PLANS". En clair, nous pourrions placer sur l'écran un certain nombre d'objets, éléments de décor qui se situeront logiquement devant les objets mobiles, et non derrière. Conséquence : lorsque les avant-plans et les objets se rencontrent sur l'écran, seuls les premiers restent visibles. Il s'agit en quelque sorte de l'inverse du fond (schéma 7.2 v. p. 200).

Cette dernière réflexion est d'ailleurs moins innocente qu'il n'y paraît. Elle nous donne la logique de base qui va nous permettre de gérer quelques avant-plans simples. En effet, il nous suffit pour cela d'inverser le principe de la couleur de fond : si un octet de couleur de fond rencontre un autre octet, c'est ce dernier qui l'emporte. Un objet pourra être d'avant-plan s'il est coloré en stylo 15, par exemple. Dans ce cas, le principe sera le suivant : lorsqu'un octet de la couleur 15 rencontre un octet, il l'emporte.

Pour appliquer le principe des avant-plans de cette manière, il suffit de choisir 8 stylos pour les objets et décors, et 8 pour les avant-plans. Pour obtenir un bel effet, on peut associer ces 8 stylos aux mêmes couleurs respectives. Les objets et avant-plans seront alors indissociables tant qu'il n'y aura pas contact entre eux.

Enfin, notre routine de transparence travaille sur un octet complet, soit deux points. Pour traiter chaque point individuellement, il faut compliquer le travail, utiliser les masques de points pour isoler chacun des deux et les traiter séparément, et les masques de stylos pour savoir s'ils correspondent au point, cela pour chaque octet bien entendu. C'est ce qui différencie la routine 7.4 des 7.2, 7.1 et 4.1 Son fonctionnement est un peu plus complexe.

```

10 ;
20 ;programme de copie d'objet
30 ;RAM->Ecran, FOND TRANSPARENT,couleurs 8 a 15 en avant-plan
40 ;programme 7.4
50 ;
4760 60      ORG  #4760
70 ;
80 ;ENTREE: (ECRAN) adresse du coin sup.gauche
90 ;      (BUF)  adresse de l'image
100 ;      (LAR)  nombre d'octets par ligne
110 ;      (HAU)  nombre de lignes
120 ;
459C 130 BUF:   EQU  #459C
45A0 140 ECRAN: EQU  #45A0
45A6 150 LAR:   EQU  #45A6
45A7 160 HAU:   EQU  #45A7

```


		170 ;			
4760	2AA045	180	LD	HL, (ECRAN)	
4763	ED5B9C45	190	LD	DE, (BUF)	
4767	3AA745	200	LD	A, (HAU)	
476A	47	210	LD	B, A	;compteur de lignes
		220 ;			
476B	C5	230	NEWLIN:	PUSH BC	;sauvegarde compteur
476C	E5	240		PUSH HL	;sauvegarde adresse ligne
476D	3AA645	250	LD	A, (LAR)	
4770	47	260	LD	B, A	;compteur du nombre d'octets
4771	7E	270	NEWOCT:	LD A, (HL)	;octet ecran
4772	0F	280		RRCA	;decale a droite
4773	CDA847	290	CALL	MASK	;transforme en numero stylo
4776	CB5F	300	BIT	3, A	;couleur d'avant-plan ?
4778	C28747	310	JP	NZ, POINT2	;oui: rien a faire, traiter point droit
477B	1A	320	LD	A, (DE)	;octet objet
477C	E6AA	330	AND	%10101010	;garde point de gauche
477E	CA8747	340	JP	Z, POINT2	;couleur fond: pas d'affichage
4781	4F	350	LD	C, A	;sinon, garder dans C
4782	7E	360	LD	A, (HL)	
4783	E655	370	AND	%01010101	;garder seulement point droit
4785	B1	380	OR	C	;ajouter point gauche objet
4786	77	390	LD	(HL), A	;remise en place nouvel octet ecran
		400 ;			
4787	7E	410	POINT2:	LD A, (HL)	;prend octet ecran
4788	CDA847	420	CALL	MASK	;transforme en no stylo
478B	CB5F	430	BIT	3, A	;couleur d'avant-plan ?
478D	C29C47	440	JP	NZ, OK	;oui:rien a transferer
4790	1A	450	LD	A, (DE)	;octet objet
4791	E655	460	AND	%01010101	;garde point droite
4793	CA9C47	470	JP	Z, OK	;fond:pas de transfert
4796	4F	480	LD	C, A	;garde pour or.
4797	7E	490	LD	A, (HL)	;octet ecran
4798	E6AA	500	AND	%10101010	;garde point gauche
479A	B1	510	OR	C	;ajoute point droit
479B	77	520	LD	(HL), A	;remise a jour ecran
		530 ;			
		540 ;			
479C	23	550	OK:	INC HL	
479D	13	560		INC DE	
479E	10D1	570		DJNZ NEWOCT	
47A0	E1	580		POP HL	;ancien debut ligne
47A1	CD26BC	590		CALL #BC26	;descend d'une ligne
47A4	C1	600		POP BC	
47A5	10C4	610		DJNZ NEWLIN	;ligne suivante
47A7	C9	620		RET	
		630 ;			

```

640 ;transforme le masque donne dans A
650 ; en numero de stylo
660 ; (masque suppose du point de droite,
670 ; donc de la forme 0x0x0x0x)
680 ;
47A8 4F      690 MASK: LD C,A           ;passe masque dans C
47A9 3E00    700      LD A,0         ;numero mis a zero
47AB CB29    710      SRA C          ;bit 1 dans Carry
47AD D2B247  720      JP NC,JUM1     ;pas de bit1 positionne
47B0 3E08    730      LD A,8         ;poids de ce bit dans le
                                   ;numero stylo
47B2 CB29    740 JUM1: SRA C
47B4 CB29    750      SRA C          ;bit 3 dans carry
47B6 D2BB47  760      JP NC,JUM2     ;bit 3 pas positionne
47B9 C602    770      ADD A,2        ;poids du bit 3
47BB CB29    780 JUM2: SRA C
47BD CB29    790      SRA C          ;bit 5 dans carry
47BF D2C447  800      JP NC,JUM3     ;pas positionne
47C2 C604    810      ADD A,4        ;poids du bit 5
47C4 CB29    820 JUM3: SRA C
47C6 CB29    830      SRA C          ;bit 7 dans carry
47C8 D0      840      RET NC         ;pas positionne:fin du travail
47C9 3C      850      INC A          ;poids=1
47CA C9      860      RET

```

Pass 2 errors: 00

Le registre HL pointe sur l'écran, DE sur l'objet à restituer. NEWOCT est le début du traitement d'un octet de l'objet. Tout d'abord, l'instruction RRCA décale cet octet à droite de façon à récupérer le numéro de stylo du point gauche. CALL MASK effectue le calcul. Celui-ci additionne 1, 2, 4 et 8 dans le registre A en fonction des bits positionnés du point. Reportez-vous à l'annexe 3 pour avoir une liste des masques de points.

Une fois le numéro de stylo calculé (compris entre 0 et 15), un test "BIT 3,A" permet de savoir si celui-ci est supérieur à 7. Si le bit est positionné, le stylo est compris entre 8 et 15. Il s'agit d'une couleur d'avant-plan. Dans ce cas, la routine saute au traitement du point de droite : l'objet n'est pas restitué, il disparaît donc derrière l'avant-plan.

Par contre si le stylo est inférieur à 8, il faut donc restituer l'octet de l'objet. Mais il faut également traiter la transparence. Pour cela, l'octet de l'objet passe par un masque 10101010 binaire. Seuls les bits du point de gauche sont gardés. Si le résultat est 0, alors le point est de la couleur du fond, il ne doit donc pas écraser le décor. Aucune modification n'est effectuée sur l'écran.

Enfin, dernier cas, nous plaçons ce point sur l'écran sans effacer le point de droite de l'écran de la façon suivante :

```
LD C,A      : le registre C reçoit le masque du point de gauche
              (après AND avec 10101010, le registre A contient
              x0x0x0x0, soit l'état du point de gauche) ;
LD A,(HL)    : A contient maintenant l'octet de l'écran ;
AND 01010101 : ceci efface l'état du point gauche de l'écran ;
OR C         : et on ajoute le nouveau point de gauche calculé ;
LD (HL),A    : on remet en place cet octet dans l'écran.
```

Ensuite, on effectue la même manipulation sur le point de droite. L'apparente complexité des manœuvres est trompeuse : la routine est aussi rapide (d'un point de vue visuel) que son équivalent ne traitant pas les points mais les octets. En effet, les opérations ajoutées sont pour la plupart des AND et des décalages. Ces opérations sont extrêmement rapides.

```
10 *****
20 ** programme 7.4 **
30 *****
40 '
50 'Programme illustrant la restitution avec dec
   or garde.
60 '
70 MEMORY &2FFF
80 LOAD"prog6.lob":LOAD"prog4.lob":LOAD"prog4.2
   ob":'voir annexe 6
90 ad=&4760:lign=170
100 ctrl=0:READ c$:IF c$="fin" THEN 240
110 FOR i=1 TO LEN(c$) STEP 2
120 c=VAL("&"+MID$(c$,i,2))
130 POKE ad,c:ad=ad+1:ctrl=ctrl+c
140 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
150 lign=lign+10:GOTO 100
160 '
170 DATA 2AA045ED5B9C453AA74547C5E53AA645477E0FC
   D, 2325
180 DATA AB47CB5FC287471AE6AACA87474F7EE655B1777
   E, 2713
190 DATA CDA847CB5FC29C471AE655CA9C474F7EE6AAB17
   7, 2834
200 DATA 231310D1E1CD26BCC110C4C94F3E00CB29D2B24
   7, 2385
210 DATA 3E08CB29CB29D2BB47C602CB29CB29D2C447C60
   4, 2393
```


Modifications de 7.6 pour 7.6c

```

30.4 0.4
300 LOAD"dessinsC",&3000
320 c=&3000+(i-1)*(19*21)
340 NEXT
350 POKE &45A6,21
360 POKE &45A7,19
380 ad=51369:POKE &45A0,ad-256*INT(ad/256):POKE
    &45A1,INT(ad/256)
390 POKE &459C,0:POKE &459D,&80:CALL &4730:'init
    . sauvegarde decor
420 POKE &459C,0:POKE &459D,&80:CALL &4700

```

Enfin, bien que les points soient désormais traités individuellement, l'objet ne peut toujours se déplacer que d'un octet au minimum, soit deux points. Pour obtenir un déplacement point par point, il faut choisir l'une des deux solutions suivantes :

- soit doubler le nombre de dessins pour un objet, afin d'obtenir des phases décalées par des points et non des ensembles de 2 points ;
- soit refaire les routines pour traiter les différents cas de figure (dessin sur un point de gauche d'un octet, et dessin sur un point de droite).

Toutefois, la majorité des travaux se contentent d'un traitement par octets et non par points.

SYSTÈME DE COORDONNÉES | 8

QUEL SYSTÈME DE COORDONNÉES ?

Les chapitres précédents nous ont permis de maîtriser la restitution visuelle des objets graphiques. Le chapitre 6 nous a même appris à modifier l'emplacement d'un objet à l'écran en fonction du mouvement demandé par le joystick. Mais le plus gros de la gestion des objets reste à faire. En effet, nous n'avons pour l'instant aucun moyen simple de connaître l'endroit de l'écran où se situe l'objet, par exemple par rapport à un mur. La seule solution semble passer par son adresse de visualisation. Étant donné la structure de l'écran, les calculs risquent de devenir cauchemardesques.

Il faut donc recourir à un système de coordonnées fictives, c'est-à-dire que la position de l'objet ne sera plus mémorisée simplement par son adresse en mémoire écran, mais également par deux coordonnées X et Y, celles-ci étant modifiées lors des déplacements. Ces coordonnées ne joueront aucun rôle dans la restitution de l'objet à l'écran. En revanche, elles nous permettront de détecter les collisions avec des éléments de décor, des murs, d'autres objets, cela en comparant simplement les coordonnées de l'objet à déplacer et celles des obstacles possibles. Nous pourrons alors, en fonction du résultat des tests, décider ou non d'inhiber le déplacement souhaité (*schéma 8.1 v. p. 209*).

Le système de coordonnées utilisé est directement lié à l'écran utilisé. Si celui-ci est intégralement pris, nous pourrons par exemple opter pour ce qui suit :

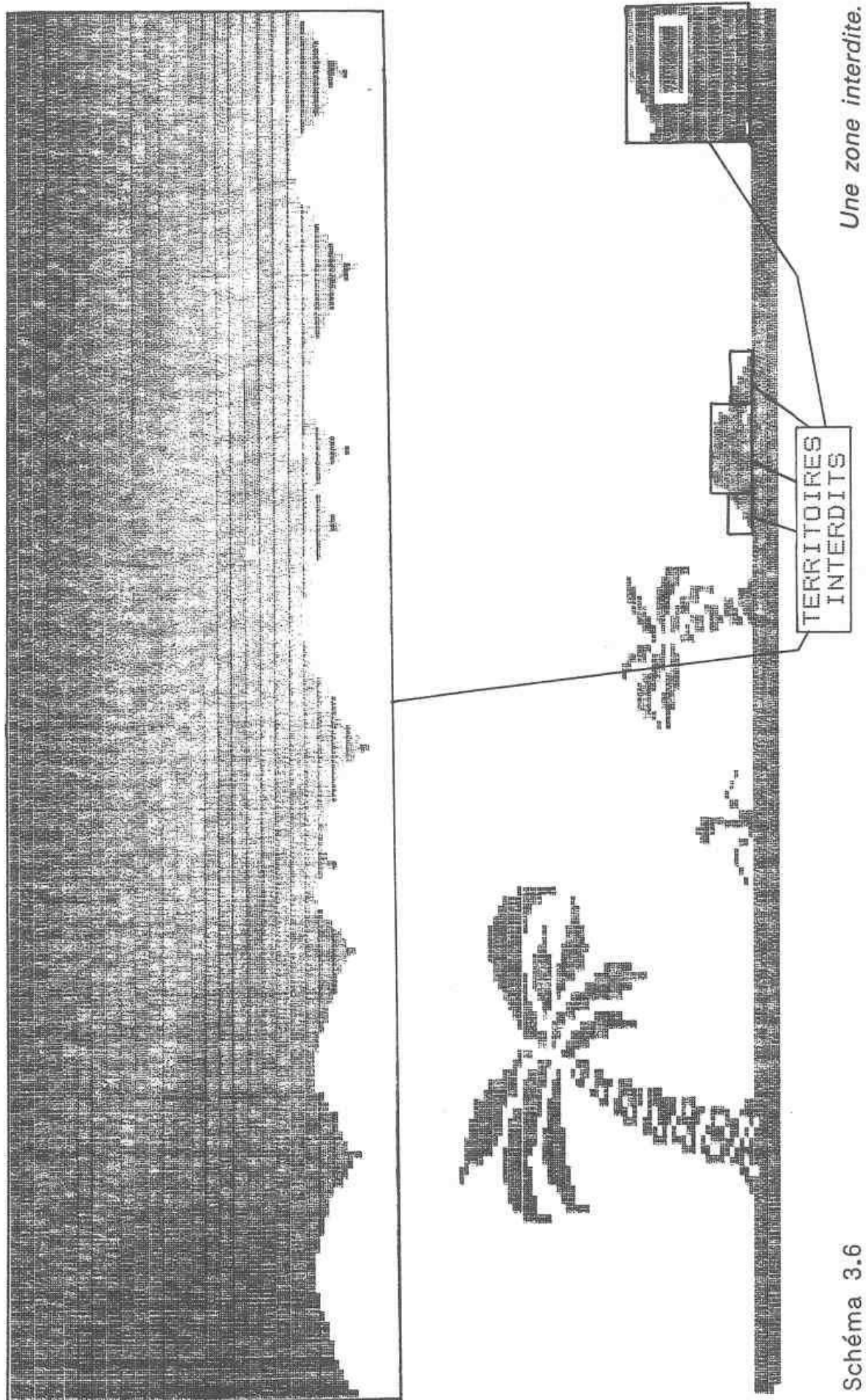
- les abscisses iront de 0 (à gauche, premier octet sur une ligne) à 79 (dernier octet, à l'extrême droite) ;
- Les ordonnées iront de 0 (en haut de l'écran) à 199 (en bas).

Dans ce cas, il sera simple, par exemple, d'interdire une sortie de l'écran, grâce à la séquence suivante :

- retenir les anciennes coordonnées X et Y ;
- calculer les nouvelles d'après le déplacement souhaité ;
- si $x < 0$ ou $x > 79$ ou $y < 0$ ou $y > 199$, restituer les anciens X et Y et fin du travail ;
- sinon, mémoriser ces nouvelles coordonnées et afficher l'objet à son nouvel emplacement en effaçant l'ancien.

TERRITOIRES INTERDITS ET COLLISIONS

On peut également interdire certains endroits de l'écran par ce même principe. Le schéma 8.1 montre par exemple une zone interdite. La zone en question peut aisément être codée dans une table qui résume les points



interdits. Par une suite de tests (vérifiant qu'aucun des points n'est identique à la nouvelle position), on peut savoir si l'objet cherche à empiéter sur la zone interdite.

Ce procédé a un inconvénient : il est lent et encombrant. Une série de tests risque de ralentir les calculs de façon significative, et il faudra 2 octets de codage pour chaque point interdit. Il existe une façon plus simple de procéder : on découpe la zone en rectangles, et on ne retient que les quatre données caractérisant chaque rectangle : le X de gauche et la largeur, le Y du haut et la hauteur.

Déplacement par coordonnées

Bien entendu, la gestion de coordonnées suppose une modification de la routine de calcul d'adresse écran. Lorsque nous effectuons un déplacement, il faut remettre à jour X et Y. Le programme en assembleur 8.1 est donc une mise à jour du programme 6.1. Il ajoute uniquement une modification de X et Y dans les sous-routines HAUT, BAS, GAUCHE et DROITE de déplacement élémentaire.

```

10 ;
20 ;Programme de deplacement par joystick
30 ;Entree:
40 ;      la variable ECRAN contient l'adresse ecran
50 ;      a recalculer en fonction de l'etat du joystick
60 ;      et X et Y les coordonnees de l'objet.
70 ;programme 8.1, remake 6.1
80 ;
45A0      90 ECRAN: EQU #45A0
45AA      100 X:   EQU #45AA
45AB      110 Y:   EQU #45AB
120 ;
489C      130      ORG #489C
489C CD24BB  140      CALL #BB24                ;get joystick state
489F 2AA045  150      LD HL,(ECRAN)
48A2 CDA948  160      CALL DEPLAC
48A5 22A045  170      LD (ECRAN),HL
48AB C9      180      RET
190 ;
48A9 E60F    200 DEPLAC: AND #0F                ;garde 4 bits de droite
48AB 17      210      RLA                        ;multiplie par deux
48AC 5F      220      LD E,A
48AD 1600    230      LD D,0                    ;transfert de l'offset 16 bits
                                           dans DE
48AF DD21C048 240      LD IX,TABLE

```

48B3	DD19	250	ADD IX,DE	;calcul localisation de l'adresse de saut
48B5	DD5E00	260	LD E,(IX+0)	
48B8	DD5601	270	LD D,(IX+1)	;recupere adresse de saut dans DE
48BB	D5	280	PUSH DE	
48BC	DDE1	290	POP IX	;transfert dans IX
48BE	DDE9	300	JP (IX)	;et saut a la routine
		310 ;		
		320 ;	la table de saut	
		330 ;		
48C0	2F49	340	TABLE: DEFW NOP	
48C2	F148	350	DEFW HAUT	
48C4	E048	360	DEFW BAS	
48C6	2F49	370	DEFW NOP	
48C8	0549	380	DEFW GAUCHE	
48CA	1749	390	DEFW DIAG1	
48CC	1D49	400	DEFW DIAG2	
48CE	0549	410	DEFW GAUCHE	
48D0	0E49	420	DEFW DROITE	
48D2	2349	430	DEFW DIAG3	
48D4	2949	440	DEFW DIAG4	
48D6	0E49	450	DEFW DROITE	
48D8	2F49	460	DEFW NOP	
48DA	F148	470	DEFW HAUT	
48DC	E048	480	DEFW BAS	
48DE	2F49	490	DEFW NOP	
		500 ;		
		510 ;	les quatres déplacements elementaires	
		520 ;		
48E0	3AAB45	530	BAS: LD A,(Y)	
48E3	3C	540	INC A	
48E4	32AB45	550	LD (Y),A	
48E7	110008	560	LD DE,#800	
48EA	19	570	ADD HL,DE	
48EB	D0	580	RET NC	;toujours dans l'ecran
48EC	1150C0	590	LD DE,#C050	
48EF	19	600	ADD HL,DE	
48F0	C9	610	RET	
		620 ;		
48F1	3AAB45	630	HAUT: LD A,(Y)	
48F4	3D	640	DEC A	
48F5	32AB45	650	LD (Y),A	
48F8	1100F8	660	LD DE,#F800	
48FB	19	670	ADD HL,DE	
48FC	7C	680	LD A,H	
48FD	FEC0	690	CP #C0	
48FF	D0	700	RET NC	;toujours dans l'ecran
4900	11B03F	710	LD DE,#3FB0	

```

4903 19      720      ADD HL,DE
4904 C9      730      RET
              740 ;
4905 2B      750 GAUCHE: DEC HL
4906 3AAA45  760      LD A,(X)
4909 3D      770      DEC A
490A 32AA45  780      LD (X),A
490D C9      790      RET
              800 ;
490E 23      810 DROITE: INC HL
490F 3AAA45  820      LD A,(X)
4912 3C      830      INC A
4913 32AA45  840      LD (X),A
4916 C9      850      RET
              860 ;
              870 ;les quatres diagonales
              880 ;
4917 CD0549  890 DIAG1: CALL GAUCHE
491A C3F148  900      JP HAUT
              910 ;
491D CD0549  920 DIAG2: CALL GAUCHE
4920 C3E048  930      JP BAS
              940 ;
4923 CD0E49  950 DIAG3: CALL DROITE
4926 C3F148  960      JP HAUT
              970 ;
4929 CD0E49  980 DIAG4: CALL DROITE
492C C3E048  990      JP BAS
              1000 ;
              1010 ;RIEN !
              1020 ;
492F C9      1030 NOP:  RET

```

Pass 2 errors: 00

Le mode d'emploi en est le même que la routine 6.1 : on envoie dans A un code de déplacement (le plus souvent, l'état du joystick), et dans le registre HL l'adresse originale d'affichage de l'objet. Ensuite, on appelle DEPLAC (adresse \$48A9). Le retour se fait avec les variables X, Y et HL remises à jour. Le petit programme au début du listing (adresses \$489C à \$48A8) est destiné à l'utilisation sous Basic de la routine : il appelle l'état de joystick, place l'adresse écran dans HL, et exécute la routine proprement dite. Ensuite, il remet à jour la variable ECRAN. Vous pouvez utiliser le programme 8.1 Basic pour observer la remise à jour des coordonnées (il s'agit bien entendu du programme 6.1 modifié pour la circonstance).

```

10 '*****
20 '** Programme 8.1 **
30 '*****
40 '
50 'deplacement d'un objet en plusieurs phases
60 'd'apres le codage effectuee par programme 4.3
70 'aux adresses $2fff et suite.
80 '
90 MEMORY &2FFF
100 LOAD"prog4.lob":'voir annexe 6
110 '
120 ad=&489C:lign=200
130 ctrl=0:READ c$:IF c$="fin" THEN 290
140 FOR i=1 TO LEN(c$) STEP 2
150 c=VAL("&" + MID$(c$,i,2))
160 POKE ad,c:ad=ad+1:ctrl=ctrl+c
170 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
    reur DATA ligne"lign:END
180 lign=lign+10:GOTO 130
190 '
200 DATA CD24BB2AA045CDA94822A045C9E60F175F1600D
    D, 2215
210 DATA 21C048DD19DD5E00DD5601D5DDE1DDE92F49F14
    8, 2712
220 DATA E0482F49054917491D4905490E49234929490E4
    9, 1166
230 DATA 2F49F148E0482F493AAB453C32AB4511000819D
    0, 1755
240 DATA 1150C019C93AAB453D32AB451100F8197CFEC0D
    0, 2232
250 DATA 11B03F19C92B3AAA453D32AA45C9233AAA453C3
    2, 1815
260 DATA AA45C9CD0549C3F148CD0549C3E048CD0E49C3F
    1, 2733
270 DATA 48CD0E49C3E048C9000000000000000000000000
    0, 1056
280 DATA "fin"
290 MODE 0
300 LOAD"image.bin",&C000:'chargement du decor,
    optionnel
310 FOR I=0 TO 15
320 INK I,ASC(MID$("ACLFSP6JOLJXSDZQ",I+1,1))-65
330 'INK I,ASC(MID$("AMTQSVSJRLJXSRYP",I+1,1))-6
    5 pour monochrome
340 NEXT
350 ecr=51328:x=&30:y=7
360 POKE &45A0,ecr-256*INT(ecr/256):POKE &45A1,I
    NT(ecr/256)
370 POKE &45AA,x:POKE &45AB,y

```



```

380 POKE &45A6,7: '          largeur en oc
      tets
390 POKE &45A7,10: '         hauteur en n
      ombre de lignes
400 LOAD"dessins",&3000
410 FOR phase=1 TO 6
420 buf(phase)=&3000+(phase-1)*(7*10)
430 NEXT
440 '
450 FOR phase=1 TO 6
460 POKE &459C,buf(phase)-256*INT(buf(phase)/256
      ):POKE &459D,INT(buf(phase)/256)
470 FOR k=1 TO 4
480 CALL &489C
490 LOCATE 1,24:PRINT"X :";PEEK(&45AA);" - Y :";
      PEEK(&45AB)
500 CALL &4700
510 NEXT
520 NEXT
530 GOTO 450

```

Modifications de 8.1 pour 8.1b

```

350 ecr=51328:x=&30:y=7
380 POKE &45A6,13: '         largeur en o
      ctets
390 POKE &45A7,33: '        hauteur en n
      ombre de lignes
400 LOAD"dessinsB",&3000
420 buf(phase)=&3000+(phase-1)*(13*33)

```

Modifications de 8.1 pour 8.1c

```

350 ecr=51328:x=&30:y=7
370 POKE &45AA,x:POKE &45AB,y
380 POKE &45A6,21: '         largeur en o
      ctets
390 POKE &45A7,19: '        hauteur en n
      ombre de lignes
400 LOAD"dessinsC",&3000
420 buf(phase)=&3000+(phase-1)*(21*19)

```

La gestion simple de coordonnées est donc la suivante :

- ☐ initialisation de l'adresse écran et des coordonnées de départ ;
- ☐ mémorisation du décor de l'emplacement initial.
- (1) mise en place des variables de travail X,Y,LAR,HAU,ECRAN,BUF.
 - Si déplacement demandé :
 - appel de la routine 8.1 DEPLAC ;
 - test des territoires interdits (bords de l'écran compris) ;
 - si déplacement valide :
 - remise en place du décor ;
 - remise à jour réelle de X,Y et ECRAN ;
 - mémorisation du décor du nouvel emplacement ;
 - affichage de l'objet sur le décor (gestion des avant-plans par la routine 7.3) ;
- ☐ continuer programme ;
- ☐ lorsque le reste du travail est terminé, on recommence en (1).

La gestion des décors écrasés et des différents graphismes d'affichage de l'objet doit être indépendante des déplacements. En effet, il faut une zone de sauvegarde de décor par objet en mouvement, et il faut remettre à jour les variables ECRAN, BUF, BUFFER, X, Y, LAR et HAU pour chaque objet géré ainsi. Il s'agit simplement de chargements en mémoire, ce n'est guère compliqué. Il nous reste à programmer la routine s'occupant des territoires interdits et des collisions entre objets.

Collisions

Chaque objet est codé dans un rectangle dont nous connaissons les caractéristiques. Une collision entre deux objets correspond donc à une intersection non vide des deux rectangles. La procédure à suivre, pour savoir si cette intersection est vide, est simple. Elle est constituée de quatre tests. S'ils sont tous vérifiés, il y a recouvrement, si l'un des quatre n'est pas vérifié, les rectangles ne sont pas en contact. Ces quatre tests fonctionnent quels que soient les cas de figure (*v. schéma p. 216*).

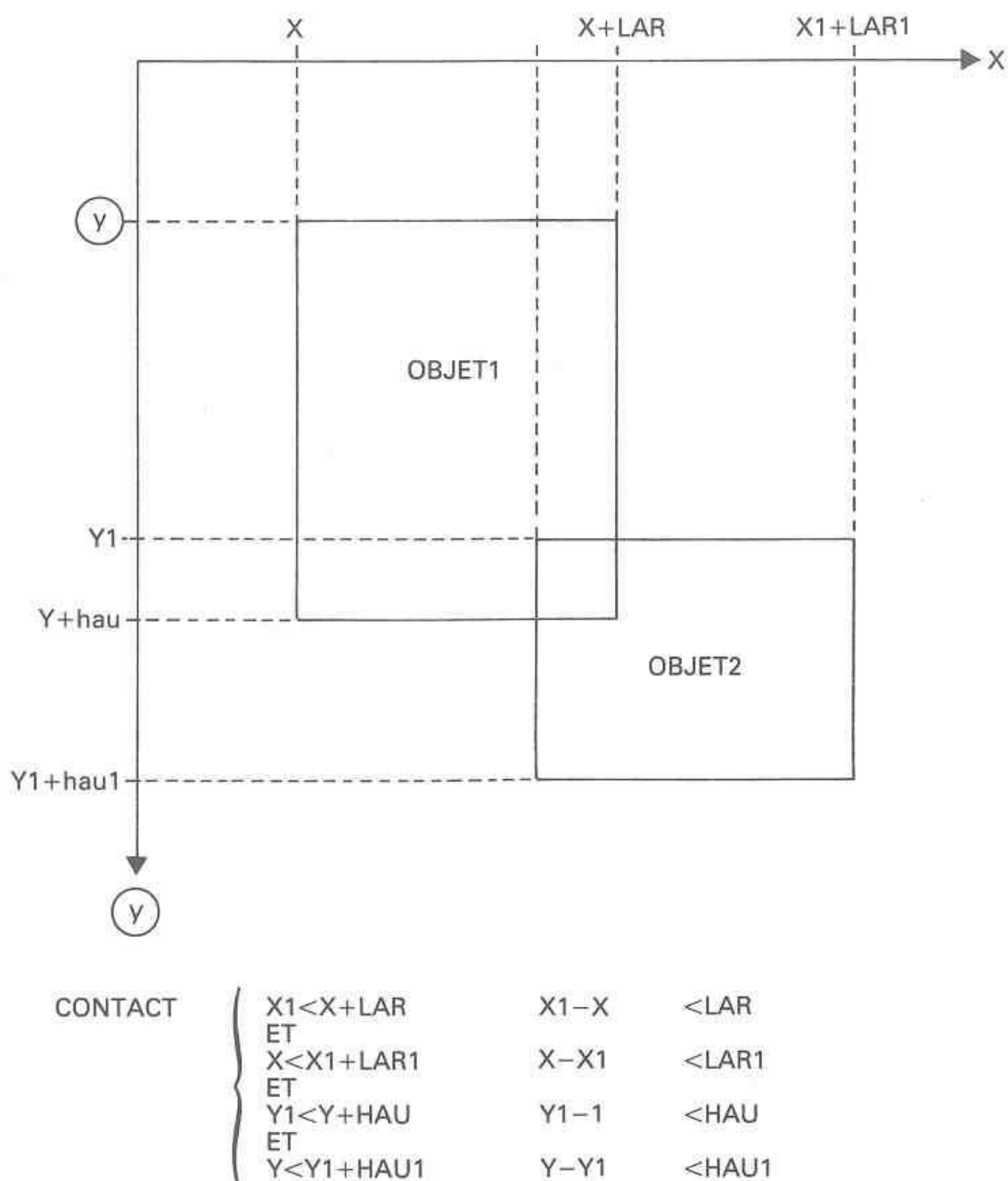


Schéma 8.2

Collision entre objets.

La programmation de la routine est simple si on utilise une zone de variables particulière. En l'occurrence, les variables X, Y, LAR, HAU, X1, Y1, LAR1 et HAU1 sont placées à partir de \$45A8, dans des octets successifs. Le registre IX contenant \$45A8, on utilise l'adressage pseudo-indexé pour accéder facilement à chacune des huit variables. Cette routine est l'occasion de constater la puissance de ce type d'adressage. Chaque case mémoire accessible par (IX+d) peut être assimilée à un registre, ce qui permet de l'ajouter à A, de le modifier et de le comparer à A très simplement.

```

10 ;
20 ;routine de detection de collision entre objets
30 ;programme 8.2
40 ;
50 ;ENTREES: les variables X, Y, LAR et HAU definissent le premier
60 ;                                objet graphique.
70 ;                                X1,Y1,LAR1,HAU1 definissent le second.
80 ;
45A8 90 X: EQU #45A8
45A9 100 Y: EQU #45A9
45AA 110 LAR: EQU #45AA
45AB 120 HAU: EQU #45AB
45AC 130 X1: EQU #45AC
45AD 140 Y1: EQU #45AD
45AE 150 LAR1: EQU #45AE
45AF 160 HAU1: EQU #45AF
45B0 170 COLLI: EQU #45B0
180 ;
4950 190 ORG #4950
4950 DD21A845 200 LD IX,X
4954 DD360800 210 LD (IX+8),0 ;par defaut, pas de contact
220 ;
230 ;1er test X1<X+LAR ?
240 ;
4958 DD7E00 250 LD A,(IX+0) ;A=X
495B DD8602 260 ADD A,(IX+2) ;A=X+LAR
495E 4F 270 LD C,A
495F DD7E04 280 LD A,(IX+4) ;par rapport a X1 ?
4962 B9 290 CP C
4963 D0 300 RET NC ;pas de contact, X+LAR<=X1.
310 ;
320 ;2e test X<X1+LAR1
330 ;
4964 DD7E04 340 LD A,(IX+4) ;A=X1
4967 DD8606 350 ADD A,(IX+6) ;A=X1+LAR1
496A 4F 360 LD C,A
496B DD7E00 370 LD A,(IX+0)
496E B9 380 CP C ;compare a X ?
496F D0 390 RET NC ;pas de contact, X1+LAR1<=X
400 ;
410 ;1er test Y1<Y+HAU
420 ;
4970 DD7E01 430 LD A,(IX+1) ;A=Y
4973 DD8603 440 ADD A,(IX+3) ;A=Y+HAU
4976 4F 450 LD C,A
4977 DD7E05 460 LD A,(IX+5)
497A B9 470 CP C ;Y+HAU<Y1 ?
497B D0 480 RET NC ;oui, pas de contact
490 ;

```

```

                    500 ;2e test Y<Y1+HAU1
                    510 ;
497C DD7E05        520      LD  A,(IX+5)          ;A=Y1
497F DD8607        530      ADD A,(IX+7)          ;A=Y1+HAU1
4982 4F           540      LD  C,A
4983 DD7E01        550      LD  A,(IX+1)
4986 B9           560      CP  C                  ;Y1+HAU1<Y ?
4987 D0           570      RET  NC                 ;oui:pas de contact
                    580 ;
                    590 ;il y a collision, les quatres tests sont verifiees.
                    600 ;
498B DD3608FF      610      LD  (IX+8),#FF        ;positionne COLLI
498C C9           620      RET

```

Pass 2 errors: 00

Cette routine compare les deux rectangles caractérisés par les 8 variables. Nous allons maintenant l'utiliser pour savoir si un objet se place sur un territoire interdit ou non. Pour cela, nous allons supposer que le programme appelant fournit deux types de données :

- ☐ celles de l'objet dans les nouvelles variables X,Y, LAR et HAU ;
- ☐ l'adresse de la table des territoires interdits, dont le contenu est le suivant :

- X territoire1
- Y territoire1
- largeur territoire 1
- hauteur territoire 1
- X territoire 2
- Y territoire 2
- largeur territoire 2
- hauteur territoire 2
- ... ainsi de suite pour chaque territoire interdit ...
- \$FF pour indiquer la fin de la table.

La routine 8.3 s'occupe des tests. En fin de travail, la variable COLLI contient \$FF si un contact a été trouvé, et 0 sinon.

```

                    10 ;
                    20 ;programme d'interdiction de zones
                    30 ;programme 8.3
                    40 ;utilise la routine 8.2 afin de tester les zones
                    50 ;
45A8                60 X:      EQU  #45A8
45A9                70 Y:      EQU  #45A9
45AA                80 LAR:    EQU  #45AA

```

45AB		90 HAU:	EQU #45AB	
45AC		100 X1:	EQU #45AC	
45AD		110 Y1:	EQU #45AD	
45AE		120 LAR1:	EQU #45AE	
45AF		130 HAU1:	EQU #45AF	
45B0		140 COLLI:	EQU #45B0	
45B1		150 TABLE:	EQU #45B1	;adresse table des zones interdites
		160 ;		
49A0		170	ORG #49A0	
		180 ;		
49A0	2AB145	190	LD HL,(TABLE)	;adresse table dans HL
49A3	AF	200	XOR A	;A=0
49A4	32B045	210	LD (COLLI),A	;mise a 0 flag de collision
		220 ;		
49A7	7E	230 LOOP:	LD A,(HL)	;1er octet
49A8	FEFF	240	CP #FF	;fin de table ?
49AA	C8	250	RET Z	;oui: termine avec succes, COLLI reste a zero
49AB	32AC45	260	LD (X1),A	
49AE	23	270	INC HL	
49AF	7E	280	LD A,(HL)	
49B0	32AD45	290	LD (Y1),A	
49B3	23	300	INC HL	
49B4	7E	310	LD A,(HL)	
49B5	32AE45	320	LD (LAR1),A	
49B8	23	330	INC HL	
49B9	7E	340	LD A,(HL)	
49BA	32AF45	350	LD (HAU1),A	
49BD	23	360	INC HL	;donnees transmises en tant qu'objet 2
49BE	E5	370	PUSH HL	;sauve pointeur table
49BF	CD5049	380	CALL #4950	;test de collision (prog 8.2)
49C2	E1	390	POP HL	;recupere pointeur
49C3	3AB045	400	LD A,(COLLI)	; test de colli
49C6	B7	410	OR A	;positionne ou non ?
49C7	C0	420	RET NZ	;oui: fin du travail
49C8	C3A749	430	JP LOOP	;tester autres territoires de la table.

Pass 2 errors: 00

ROUTINE DE DÉPLACEMENT AUTOMATIQUE

La mise en œuvre de la routine ne pose pas de problème particulier ; elle suit l'algorithme décrit plus haut. Le programme Basic 8.3 montre ce déroulement des opérations.

```

10 '*****
20 '** Programme 8.3 **
30 '*****
40 '
50 'deplacement d'un objet avec coordonnees et t
   erritoires interdits.
60 '
70 MEMORY &2FFF
80 LOAD"prog8.1ob":LOAD"prog4.1ob":'voir annexe
   6
90 LOAD"prog4.2ob":LOAD"prog7.4ob":'voir annexe
   6
100 ad=&4950:lign=180
110 ctrl=0:READ c$:IF c$="fin" THEN 240
120 FOR i=1 TO LEN(c$) STEP 2
130 c=VAL("&" +MID$(c$,i,2))
140 POKE ad,c:ad=ad+1:ctrl=ctrl+c
150 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
160 lign=lign+10:GOTO 110
170 '
180 DATA DD21A845DD360800DD7E00DD86024FDD7E04B9D
   0, 2301
190 DATA DD7E04DD86064FDD7E00B9D0DD7E01DD86034FD
   D, 2537
200 DATA 7E05B9D0DD7E05DD86074FDD7E01B9D0DD3608F
   F, 2596
210 DATA C9, 201
220 DATA fin
230 '
240 ad=&49A0:lign=320
250 ctrl=0:READ c$:IF c$="fin" THEN 370
260 FOR i=1 TO LEN(c$) STEP 2
270 c=VAL("&" +MID$(c$,i,2))
280 POKE ad,c:ad=ad+1:ctrl=ctrl+c
290 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
300 lign=lign+10:GOTO 250
310 '

```

```

320 DATA 2AB145AF32B0457EFEFFFC832AC45237E32AD452
    3, 2372
330 DATA 7E32AE45237E32AF4523E5CD5049E13AB045B7C
    0, 2399
340 DATA C3A749, 435
350 DATA fin
360 '
370 MODE 0
380 LOAD"image.bin",&C000:'chargement du decor,
    optionnel
390 FOR I=0 TO 15
400 'INK I,ASC(MID$("ACLFSPGJOLJXSDZQ",I+1,1))-6
    5:'pour couleur
410 INK I,ASC(MID$("AMTQSVSJRLJXSRVQ",I+1,1))-65
    : ' pour monochrome
420 NEXT
430 LOAD"dessins",&3000
440 FOR phase=1 TO 6
450 buf(phase)=%3000+(phase-1)*(7*10)
460 NEXT
470 '
480 'initialisations variables
490 '
500 x=0:y=170:ecran=5280
510 lar=7:hau=10
520 POKE &45A6,lar:POKE &45A7,hau
530 POKE &45A0,ecran-256*INT(ecran/256):POKE &45
    A1,INT(ecran/256)
540 POKE &459C,0:POKE &459D,&37
550 CALL &4730:'memorise premier decor
560 colli=%45B0:table=%37F0
570 POKE &45B1,&F0:POKE &45B2,&37
580 POKE table,0:POKE table+1,0
590 POKE table+2,80:POKE table+3,74
600 POKE table+4,65:POKE table+5,165
610 POKE table+6,15:POKE table+7,35
620 POKE table+8,&FF:'fin table des territoires
    interdits
630 '
640 FOR phase=1 TO 6
650 ecran2=ecran
660 POKE &45AA,x:POKE &45AB,y
670 CALL &489C:'deplacement demande
680 IF PEEK(&45AA)=x AND PEEK(&45AB)=y THEN 800:
    'pas de deplacement
690 IF PEEK(&45AA)>80-lar THEN 800:'sortie ecran
    interdite
700 IF (PEEK(&45AB)>199) OR (y=0 AND PEEK(&45AB)
    >127) THEN 800:'sortie ecran

```

```

710 POKE &45A8,PEEK(&45AA)
720 POKE &45A9,PEEK(&45AB)
730 POKE &45AA,lar
740 POKE &45AB,hau
750 CALL &49A0:'test territoires interdits
760 IF PEEK(colli)<>0 THEN 800:'interdit
770 'deplacement valide
780 x=PEEK(&45A8):y=PEEK(&45A9)
790 ecran2=PEEK(&45A0)+256*PEEK(&45A1)
800 POKE &45A0,ecran-256*INT(ecran/256):POKE &45
    A1,INT(ecran/256)
810 POKE &459C,0:POKE &459D,&37
820 CALL &4700:'remise en place decor
830 ecran=ecran2
840 POKE &45A0,ecran-256*INT(ecran/256):POKE &45
    A1,INT(ecran/256)
850 CALL &4730:'memorisation nouveau decor
860 c=buf(phase):POKE &459C,c-256*INT(c/256):POK
    E &459D,INT(c/256)
870 CALL &4760:'dessin objet, avant-plan et fond
880 '
890 POKE &45A0,ecran-256*INT(ecran/256):POKE &45
    A1,INT(ecran/256)
900 NEXT
910 GOTO 640

```

Modifications de 8.3 pour 8.3b

```

430 LOAD"dessinsB",&3000
450 buf(phase)=&3000+(phase-1)*(13*33)
500 x=0:y=170:ecran=52880
510 lar=13:hau=33
540 POKE &459C,0:POKE &459D,&80
560 colli=&45B0:table=&8200
570 POKE &45B1,&0:POKE &45B2,&82
810 POKE &459C,0:POKE &459D,&80

```

Modifications de 8.3 pour 8.3c

```

430 LOAD"dessinsC",&3000
450 buf(phase)=&3000+(phase-1)*(21*19)
500 x=0:y=170:ecran=52880
510 lar=21:hau=19
540 POKE &459C,0:POKE &459D,&80
560 colli=&45B0:table=&8200
570 POKE &45B1,&0:POKE &45B2,&82
810 POKE &459C,0:POKE &459D,&80

```

Le joystick déplace le module à l'intérieur de l'écran. Vous pouvez également remarquer que l'arbre situé sur le sol est intégré au décor (le module passe devant), tandis que celui situé plus bas est un avant-plan : le module passe derrière. Cela est dû à la routine de restitution d'objets avec avant-plans. Il suffit simplement de tracer l'arbre avec les stylos 8 à 15 pour qu'il fasse partie du premier plan.

Nous sommes sur le chemin d'une gestion automatique des objets et des décors. Mais, malgré tout, le programme Basic 8.3 est un condensé de POKE, PEEK et CALL. De plus, il est extrêmement lent. Pour remédier à cela, il nous suffit de transposer la partie finale de ce programme en assembleur. C'est le but de la routine 8.4.

```

10 ;
20 ;programme de deplacement
30 ;programme 8.4
40 ;
45B3 50 X0: EQU #45B3
45B4 60 Y0: EQU #45B4
45A6 70 LAR0: EQU #45A6
45A7 80 HAU0: EQU #45A7
459C 90 BUF: EQU #459C
45B9 100 BUFFER: EQU #45B9
45BB 110 DESSIN: EQU #45BB
45BD 120 JOYST: EQU #45BD
45A0 130 ECRAN: EQU #45A0
45B5 140 ECRAN0: EQU #45B5
45B7 150 ECRAN2: EQU #45B7
45A8 160 X: EQU #45A8
45A9 170 Y: EQU #45A9
45AA 180 LAR: EQU #45AA
45AB 190 HAU: EQU #45AB
45AC 200 X1: EQU #45AC
45AD 210 Y1: EQU #45AD
45AE 220 LAR1: EQU #45AE
45AF 230 HAU1: EQU #45AF
45B0 240 COLLI: EQU #45B0
250 ;
4A00 260 ORG #4A00
4A00 2A045 270 LD HL, (ECRAN)
4A03 22B545 280 LD (ECRAN0), HL ;retenir ancienne pos ecran
4A06 22B745 290 LD (ECRAN2), HL ;nouvelle pos par defaut=la
                                     meme
4A09 3AB345 300 LD A, (X0)
4A0C 32AA45 310 LD (LAR), A
4A0F 3AB445 320 LD A, (Y0)
4A12 32AB45 330 LD (HAU), A

```

4A15	3ABD45	340	LD	A, (JOYST)	
4A18	B7	350	OR	A	
4A19	C2224A	360	JP	NZ, PASJOY	;pas deplace par joystick
4A1C	CD9C48	370	CALL	#489C	;deplacement d'apres joystick
4A1F	C32B4A	380	JP	SUITE1	
4A22	2AA045	390	PASJOY: LD	HL, (ECRAN)	
4A25	CDA948	400	CALL	#48A9	;deplacement automatique
4A28	22A045	410	LD	(ECRAN), HL	
		420	;		
4A2B	3AAA45	430	SUITE1: LD	A, (LAR)	
4A2E	4F	440	LD	C, A	
4A2F	3AB345	450	LD	A, (X0)	
4A32	B9	460	CP	C	;deplacement sur X ?
4A33	C2414A	470	JP	NZ, SUITE2	;oui: ok pour suite
4A36	3AAB45	480	LD	A, (HAU)	
4A39	4F	490	LD	C, A	
4A3A	3AB445	500	LD	A, (Y0)	
4A3D	B9	510	CP	C	;deplacement sur y ?
4A3E	CAA54A	520	JP	Z, SUITE4	;non: pas de travail a faire
		530	;		
4A41	3AAA45	540	SUITE2: LD	A, (LAR)	;nouvel x
4A44	4F	550	LD	C, A	;transfere dans C
4A45	3AA645	560	LD	A, (LARD)	;largeur
4A48	81	570	ADD	A, C	;additionnee
4A49	FE50	580	CP	80	;bord droit ecran ?
4A4B	D2A04A	590	JP	NC, SUITE3	;oui: pas autorise.
4A4E	79	600	LD	A, C	;x
4A4F	FEFF	610	CP	255	;trop a gauche ?
4A51	CAA04A	620	JP	Z, SUITE3	
4A54	3AAB45	630	LD	A, (HAU)	;nouvel y
4A57	4F	640	LD	C, A	
4A58	3AA745	650	LD	A, (HAUD)	
4A5B	81	660	ADD	A, C	
4A5C	FEC8	670	CP	200	;trop en bas ?
4A5E	D2A04A	680	JP	NC, SUITE3	
4A61	3AAB45	690	LD	A, (HAU)	
4A64	FEFF	700	CP	255	;trop en haut ?
4A66	D2A04A	710	JP	NC, SUITE3	
4A69	3AAA45	720	LD	A, (LAR)	;nouvel x
4A6C	32A845	730	LD	(X), A	
4A6F	3AAB45	740	LD	A, (HAU)	
4A72	32A945	750	LD	(Y), A	;nouvel y
4A75	3AA645	760	LD	A, (LARD)	
4A78	32AA45	770	LD	(LAR), A	
4A7B	3AA745	780	LD	A, (HAUD)	
4A7E	32AB45	790	LD	(HAU), A	
4A81	CDA049	800	CALL	#49A0	;test territoires
4A84	3AB045	810	LD	A, (COLLI)	
4A87	B7	820	OR	A	

4A8B	C2A04A	830	JP	NZ,SUITE3	;il y a eu collision
		840 ;			
4A8B	3AA845	850	LD	A,(X)	
4ABE	32B345	860	LD	(X0),A	;memorise nouvel x
4A91	3AA945	870	LD	A,(Y)	
4A94	32B445	880	LD	(Y0),A	;nouvel y
4A97	2AA045	890	LD	HL,(ECRAN)	;ecran recalculé
4A9A	22B745	900	LD	(ECRAN2),HL	
4A9D	C3A54A	910	JP	SUITE4	
4AA0	3E01	920	SUITE3: LD	A,1	
4AA2	32B045	930	LD	(COLLI),A	
4AA5	2AB545	940	SUITE4: LD	HL,(ECRAN0)	
4AAB	22A045	950	LD	(ECRAN),HL	
4AAB	2AB945	960	LD	HL,(BUFFER)	
4AAE	229C45	970	LD	(BUF),HL	
4AB1	CD0047	980	CALL	#4700	;restitution ancien decor
4AB4	2AB745	990	LD	HL,(ECRAN2)	
4AB7	22B545	1000	LD	(ECRAN0),HL	;ok nouvelle pos ecran
4ABA	22A045	1010	LD	(ECRAN),HL	
4ABD	CD3047	1020	CALL	#4730	;memorise nouveau decor
4AC0	2AB845	1030	LD	HL,(DESSIN)	
4AC3	229C45	1040	LD	(BUF),HL	
4AC6	CD6047	1050	CALL	#4760	;dessin de l'objet
		1060 ;			
4AC9	2AB545	1070	COLLIS: LD	HL,(ECRAN0)	
4ACC	22A045	1080	LD	(ECRAN),HL	
4ACF	C9	1090	RET		

Pass 2 errors: 00

Il faut fournir à cette routine les coordonnées de l'objet, sa largeur, sa hauteur, son adresse et son adresse écran initiale. Nous pourrions alors demander une gestion automatique du joystick répondant aux critères définis plus haut. Une facilité a été ajoutée au programme : si la variable JOYST contient une valeur différente de 0, celle-ci est prise en compte à la place de l'état du joystick, comme si elle en provenait. Il s'agit donc d'une valeur de 0 à 15, indiquant un déplacement selon les règles examinées au chapitre 6. Cette facilité permet de mouvoir un objet indépendamment du joystick, en continu ou non. Elle est essentielle car notre routine de déplacement automatique peut ainsi être utilisée pour des objets n'appartenant pas au joueur, gérés par le programme.

Il convient de noter les variables que le programme appelant doit fournir :

- **XO** est l'abscisse de l'objet lors de l'appel ;
- **YO** est l'ordonnée ;
- **LARD** et **HAUO** sont respectivement la largeur et la hauteur de l'objet ;

- **BUFFER** est l'adresse de la zone de sauvegarde de décor associée à l'objet. Chaque objet doit en effet posséder une zone décor qui lui est propre (même largeur, même hauteur) ;
- **DESSIN** est l'adresse où se situe le codage du dessin. Cette adresse doit être modifiée soit lorsque l'on change d'objet, soit lorsque celui-ci entre dans une nouvelle phase d'animation. Son dessin change alors, indépendamment de toute autre considération ;
- **JOYST** concerne les demandes de déplacement automatique (voir ci-dessus).

Ceci est illustré par le programme Basic 8.4.

```

10 '*****
20 '** Programme 8.4 **
30 '*****
40 '
50 'deplacement d'un objet avec coordonnees et t
   erritoires interdits.
60 '
70 MEMORY &2FFF
80 LOAD"prog8.1ob":LOAD"prog4.1ob":'voir annexe
   6
90 LOAD"prog4.2ob":LOAD"prog7.4ob":'voir annexe
   6
100 LOAD"prog8.2ob":LOAD"prog8.3ob":'voir annexe
   6
110 ad=&4A00:lign=190
120 ctrl=0:READ c$:IF c$="fin" THEN 330
130 FOR i=1 TO LEN(c$) STEP 2
140 c=VAL("&" +MID$(c$,i,2))
150 POKE ad,c:ad=ad+1:ctrl=ctrl+c
160 NEXT:READ teste:IF teste<>ctrl THEN PRINT"Er
   reur DATA ligne"lign:END
170 lign=lign+10:GOTO 120
180 '
190 DATA 2AA04522B54522B7453AB34532AA453AB44532A
   B, 1964
200 DATA 453ABD45B7C2224ACD9C48C32B4A2AA045CDA94
   8, 2332
210 DATA 22A0453AAA454F3AB345B9C2414A3AAB454F3AB
   4, 2078
220 DATA 45B9CAA54A3AAA454F3AA645B1FE50D2A04A79F
   E, 2646
230 DATA FFCAA04A3AAB454F3AA745B1FEC8D2A04A3AAB4
   5, 2687
240 DATA FEFFD2A04A3AAA4532AB453AAB4532A9453AA64
   5, 2416
250 DATA 32AA453AA74532AB45CDA0493AB045B7C2A04A3
   A, 2283

```

```

260 DATA A84532B3453AA94532B4452AA04522B745C3A54
    A, 2121
270 DATA 3E0132B0452AB54522A0452AB945229C45CD004
    7, 1744
280 DATA 2AB74522B54522A045CD30472ABB45229C45CD6
    0, 2023
290 DATA 472AB54522A045C9, 827
300 '
310 DATA fin
320 '
330 MODE 0
340 LOAD"image.bin",&C000:'chargement du decor,
    optionnel
350 FOR I=0 TO 15
360 'INK I,ASC(MID$("ACLFSPGJOLJXSDZQ",I+1,1))-6
    5:'pour couleur
370 INK I,ASC(MID$("AMTQSVSJRLJXSRVQ",I+1,1))-65
    : ' pour monochrome
380 NEXT
390 LOAD"dessins",&3000
400 FOR phase=1 TO 6
410 c=&3000+(phase-1)*(7*10)
420 buf(phase,0)=c-256*INT(c/256):buf(phase,1)=I
    NT(c/256)
430 NEXT
440 '
450 'initialisations variables
460 '
470 x=0:y=170:ecran=52880
480 lar=7:hau=10
490 POKE &45A6,lar:POKE &45A7,hau
500 POKE &45A0,ecran-256*INT(ecran/256):POKE &45
    A1,INT(ecran/256)
510 POKE &459C,0:POKE &459D,&37
520 CALL &4730:'memorise premier decor
530 colli=&45B0:table=&37F0
540 POKE &45B1,&F0:POKE &45B2,&37
550 POKE table,0:POKE table+1,0
560 POKE table+2,80:POKE table+3,74
570 POKE table+4,65:POKE table+5,175
580 POKE table+6,15:POKE table+7,25
590 POKE table+8,&FF:'fin table des territoires
    interdits
600 '
610 '
620 POKE &45B9,0:POKE &45BA,&37
630 POKE &45B3,x:POKE &45B4,y
640 POKE &45BD,0
650 '

```

```

660 FOR phase=1 TO 6
670 FOR I=1 TO 3
680 POKE &45BB,buf(phase,0):POKE &45BC,buf(phase
    ,1)
690 IF PEEK(&45B4)> 50 THEN CALL &BD19
700 CALL &4A00
710 IF PEEK(&45B4)<50 THEN CALL &BD19
720 NEXT
730 NEXT
740 GOTO 660

```

Modifications de 8.4 pour 8.4b

```

390 LOAD"dessinsB",&3000
410 c=&3000+(phase-1)*(13*33)
470 x=0:y=170:ecran=52880
480 lar=13:hau=33
510 POKE &459C,0:POKE &459D,&80
530 colli=&45B0:table=&8200
540 POKE &45B1,&0:POKE &45B2,&82
620 POKE &45B9,0:POKE &45BA,&80

```

Modifications de 8.4 pour 8.4c

```

390 LOAD"dessinsC",&3000
410 c=&3000+(phase-1)*(21*19)
470 x=0:y=170:ecran=52880
480 lar=21:hau=19
510 POKE &459C,0:POKE &459D,&80
530 colli=&45B0:table=&8200
540 POKE &45B1,&0:POKE &45B2,&82
620 POKE &45B9,0:POKE &45BA,&80

```

Vous constatez la quasi-disparition des instructions POKE. Il ne reste en effet que les initialisations (mémorisation préliminaire du décor, mise en place des valeurs initiales des variables) et le changement de phase d'animation (modification de la variable DESSIN). Notez également le test associé à l'instruction CALL &BD19, permettant de synchroniser l'affichage de l'objet avec le balayage de l'écran. Celui-ci dépend de la position verticale de l'objet.

Ce programme nous permet de constater deux choses : d'abord, et c'était le but à atteindre, les déplacements s'effectuent désormais à une vitesse correcte. Mais, surtout, le programme est beaucoup plus simple. Le déplacement est intégralement géré par un simple CALL : restitution et mémorisation du décor, blocage des territoires interdits, préservation des avant-plans. Nous disposons maintenant d'une routine extrêmement puissante.

CRÉATION DES OBJETS ET DÉCORS | 9

La création des objets graphiques et des images de décor nécessite généralement un programme approprié, à moins de recourir au codage manuel de chaque octet de la mémoire écran. Nous proposons donc, dans ce chapitre, un ensemble de programmes utilitaires à cet effet. Nous n'en détaillerons pas le fonctionnement. Ils vous permettront de créer une bibliothèque de dessins et d'objets graphiques utilisables dans vos programmes d'application.

Une dernière remarque s'impose au sujet des programmes de ce chapitre. Afin de mettre en place un véritable système de gestion des objets et décors, les programmes utilisent une structure commune de noms de fichiers. Par exemple, un fichier d'extension IME est une image créée par l'utilitaire de dessin. La majorité des travaux s'effectuant à partir de ces fichiers, nous avons supposé la présence d'un lecteur de disquettes. Les travaux seraient d'ailleurs très fastidieux sans un tel accessoire.

PROGRAMME DE CRÉATION DE DESSINS

Mise en œuvre et utilisation

Le programme Basic charge le fichier binaire SCROLLS.BIN contenant ses routines assembleur. Il faut donc créer ce fichier avant d'utiliser le programme. Pour cela, entrez le source assembleur ci-joint et assemblez-le, puis sortez-le sous forme de fichier. Autre solution : rentrer les codes hexadécimaux en mémoire un par un, puis sauver la zone mémoire ainsi occupée par une commande SAVE du Basic, sous le nom SCROLLS.BIN.

```

10 OPENOUT"d":MEMORY &1BFF:LOAD"scrolls":CLOSEOU
   T: CLEAR:PRINT CHR$(23)CHR$(0)
20 MODE 0:PEN 14:KEY 138,"pen 1:ink 1,15:mode 2"
   +CHR$(13)
30 WINDOW #0,1,20,1,3:WINDOW #1,1,20,4,5
40 DEFINT a-z:DIM col(16),h(16)
50 droite=&81FC:gauche=&819E:haut=&8110:bas=&815
   7:param=&8397:pa1=&1C00+160*79:pa2=pa1+79:pa3=
   &1C00:full=&83AA:clr=&8399:wide=&83E0:GOSUB 20
   000
60 FOR i=0 TO 15:col(i)=i:INK i,i:NEXT:col(16)=-
   1:CALL clr
70 FOR i=0 TO 159 STEP 2:MOVE i*4,0:DRAW i*4,159
   *2,15:MOVE 0,i*2:DRAW 159*4,i*2,15:NEXT
80 SYMBOL 255,&FF,&81,&81,&81,&81,&81,&81,&FF
90 PEN#1,14:PAPER#0,0:CLS#1
100 FOR i=0 TO 15:IF i=enc THEN PRINT#1,CHR$(24)
   +CHR$(col(i)+65)+CHR$(24); ELSE PRINT#1,CHR$(
   col(i)+65);
110 NEXT:PRINT#1
120 FOR i=0 TO 15
130 PAPER#1,i:PRINT#1,CHR$(255);
140 NEXT:PAPER#1,0:PEN#1,15:PRINT#1,CHR$(24)+"@"
   CHR$(24);
150 IF mo=0 THEN 310 ELSE IF mo=2 THEN 520
160 CLS:flag=0:PRINT"SELECTION COULEURS:":IF enc
   =16 THEN GOSUB 10020:enc=15:GOSUB 10030
170 PRINT CHR$(240)+" "+CHR$(241)+":COULEUR":PRI
   NT CHR$(242)+" "+CHR$(243)+":STYLO";
180 A$=INKEY$:IF (A$<>CHR$(13)) AND (a$<>"^") AN
   D (A$<CHR$(240) OR A$>CHR$(243)) THEN 180
190 IF a$<>"^" THEN 220
200 IF flag THEN c=pa3:CALL wide:flag=0:GOTO 180
210 CALL full:flag=1:GOTO 180
220 IF A$=CHR$(13) THEN mo=2:GOTO 520
230 ON ASC(a$)-239 GOSUB 240,260,280,290:GOTO 30
   0
240 IF col(enc)<1 THEN RETURN:REM ELSE IF enc>0
   THEN IF col(enc)=col(enc-1) THEN RETURN
250 col(enc)=col(enc)-1:INK enc,col(enc):GOTO 10
   030

```

```

260 IF col(enc)>25 THEN RETURN:REM ELSE IF enc<1
    5 THEN IF col(enc)=col(enc+1) THEN RETURN
270 col(enc)=col(enc)+1:INK enc,col(enc):GOTO 10
    030
280 IF enc=0 THEN RETURN ELSE GOSUB 10020:enc=enc-1:GOTO 10030
290 IF enc=15 THEN RETURN ELSE GOSUB 10020:enc=enc+1:GOTO 10030
300 GOTO 180
310 '
320 CLS:PRINT"CURSEUR-TRACE"
330 cou=PEEK(&1C00+yg*160+xg):PLOT 4*(xgb*2+1),2
    *(ygb*2+1),16+NOT(cou):LOCATE cou+1,3:PRINT C
    HR$(241);
340 a$=UPPER$(INKEY$):PLOT 4*(xgb*2+1),2*(ygb*2+
    1),cou:LOCATE cou+1,3:PRINT " ";
350 GOSUB 10010
360 IF a$=CHR$(13) THEN mo=1:GOTO 150:REM select
    couleurs
370 IF a$<"@" OR a$>"[" THEN 400
380 c=ASC(a$)-65:FOR l=0 TO 16:IF col(l)=c THEN
    GOSUB 10020:enc=l:GOSUB 10030:l=17
390 NEXT:GOSUB 10020:GOSUB 10030:GOTO 330
400 IF a$<CHR$(240) OR a$>CHR$(243) THEN 330
410 IF enc<>16 THEN POKE &1C00+yg*160+xg,enc:PLO
    T 4*(xgb*2+1),2*(ygb*2+1),enc
420 ON ASC(a$)-239 GOSUB 450,430,470,490:GOTO 51
    0
430 IF ygb=0 THEN IF yg=0 THEN RETURN ELSE pa1=U
    NT(pa1-160):pa2=UNT(pa2-160):pa3=UNT(pa3-160)
    :c=pa3:GOSUB 10000:CALL haut
440 yg=yg+(yg>0):ygb=ygb+(ygb>0):RETURN
450 IF ygb=79 THEN IF yg=159 THEN RETURN ELSE pa
    1=UNT(pa1+160):pa2=UNT(pa2+160):pa3=UNT(pa3+1
    60):c=pa1:GOSUB 10000:CALL bas
460 yg=yg-(yg<159):ygb=ygb-(ygb<79):RETURN
470 IF xgb=0 THEN IF xg=0 THEN RETURN ELSE pa1=U
    NT(pa1-1):pa2=UNT(pa2-1):pa3=UNT(pa3-1):c=pa1
    :GOSUB 10000:CALL droite
480 xg=xg+(xg>0):xgb=xgb+(xgb>0):RETURN
490 IF xgb=79 THEN IF xg=159 THEN RETURN ELSE pa
    1=UNT(pa1+1):pa2=UNT(pa2+1):pa3=UNT(pa3+1):c=
    pa2:GOSUB 10000:CALL gauche
500 xg=xg-(xg<159):xgb=xgb-(xgb<79):RETURN
510 GOTO 330
520 '
530 CLS:PRINT"COM. "+CHR$(24)+"S"+CHR$(24)+"-SAV
    E ECRAN":PRINT CHR$(24)+"L"+CHR$(24)+"-LOAD,"
    +CHR$(24)+"^"+CHR$(24)+"-IMAGE":PRINT CHR$(24)
    +"+D"+CHR$(24)+"-FICHER OBJET";
540 A$=UPPER$(INKEY$)
550 IF A$=CHR$(13) THEN MO=0:GOTO 150
560 IF A$="S" THEN 610:REM save
570 IF A$="L" THEN 640:REM LOAD
580 GOSUB 10010
590 IF A$="D" THEN 7000:REM DUMP SUR PRINTER
600 GOTO 540
610 CLS:PRINT"NOM FICHER":INPUT A$:CLS:PRINT"SA
    VE EN COURS"

```



```

620 SAVE a$+".ime",b,&1C00,&6400:OPENOUT a$+".pae"
    :PRINT#9,xg,yg,pa1,pa2,pa3,xgb,ygb:FOR i=0
    TO 15:PRINT#9,col(i):NEXT:CLOSEOUT:a$="*.bak"
    :!ERA,@a$
630 GOTO 530
640 CLEAR:DEFINT a-z:DIM col(16),h(16):GOSUB 200
    00:CLS:PRINT"NOI FICHIER":INPUT A$:CLS:PRINT"
    LOAD EN COURS":droite=&81FC:gauche=&819E:haut
    =&8110:bas=&8157:param=&8397:full=&83AA:clr=&
    8399:wide=&83E0
650 LOAD a$+".ime",&1C00:OPENIN a$+".pae":INPUT#
    9,xg,yg,pa1,pa2,pa3,xgb,ygb:FOR i=0 TO 15:INP
    UT#9,col(i):INK i,col(i):NEXT:col(16)=-1:CLOS
    EIN:c=pa3:GOSUB 10000:CALL wide:mo=0:GOTO 80
7000 CALL full:x1=0:y1=0:CLS:PRINT"POINT HAUT GA
    UCHE ?":PRINT CHR$(23);CHR$(0);
7010 a=TEST(x1,y1):PLOT x1,y1,0:A$=INKEY$:PLOT X
    1,Y1,15:PLOT x1,y1,a:IF A$<CHR$(240) OR A$>C
    HR$(243) THEN IF A$<>CHR$(13) THEN 7010
7020 IF A$=CHR$(13) THEN 7050
7030 IF A$=CHR$(240) THEN Y1=Y1+2 ELSE IF a$=CHR
    $(241) THEN y1=y1-2 ELSE IF a$=CHR$(242) THE
    N x1=x1-4 ELSE IF a$=CHR$(243) THEN x1=x1+4
7040 GOTO 7010
7050 CLS:x2=X1:y2=Y1:PRINT CHR$(23);CHR$(1);
7060 LOCATE 1,1:PRINT"LAR":INT((X2-X1+1)/8):PRIN
    T"HAU":INT((Y2-Y1)/2):MOVE X1,Y1:DRAW X1,Y2,
    15:DRAW X2,Y2:DRAW X2,Y1:DRAW X1,Y1:A$=INKEY
    $:MOVE X1,Y1:DRAW X1,Y2:DRAW X2,Y2:DRAW X2,Y
    1:DRAW X1,Y1
7070 IF A$=CHR$(13) THEN 7100
7080 IF A$=CHR$(240) THEN Y2=Y2+2 ELSE IF a$=CHR
    $(241) THEN y2=y2-2 ELSE IF a$=CHR$(242) THE
    N x2=x2-4 ELSE IF a$=CHR$(243) THEN x2=x2+4
7090 GOTO 7060
7100 CLS:INPUT"SAVE ADD. ";ad:a1=ad:FOR Y=Y1 TO
    Y2 STEP -2:A$="":FOR X=X1 TO X2 STEP 8:POKE
    ad,h(TEST(x,y))*2+h(TEST(x+4,y)):ad=ad+1:NEX
    T:NEXT:PRINT CHR$(23);CHR$(0);
7110 INPUT "Nom ";f$:SAVE f$+".imo",b,a1,ad-a1+1
    :c=pa3:OPENOUT f$+".pao":PRINT#9,INT((X2-X1)
    /8)+1,INT((Y1-Y2)/2):FOR i=0 TO 15:PRINT#9,c
    ol(i):NEXT:CLOSEOUT:a$="*.bak":!ERA,@a$:CALL
    wide:GOTO 520
10000 POKE param,VAL("&"+RIGHT$(HEX$(c,4),2)):PO
    KE param+1,VAL("&"+LEFT$(HEX$(c,4),2)):RETU
    RN
10010 IF UPPER$(A$)="^" THEN CALL full:WHILE INK
    EY$="":WEND:c=pa3:GOSUB 10000:CALL wide:RET
    URN ELSE RETURN
10020 LOCATE#1,enc+1,1:PRINT#1,CHR$(col(enc)+65)
    ;:RETURN
10030 LOCATE#1,enc+1,1:PRINT#1,CHR$(24)+CHR$(col
    (enc)+65)+CHR$(24);:RETURN
10040 RETURN
20000 RESTORE 20000:FOR i=0 TO 15:READ h$:H(I)=V
    AL("&"+H$):NEXT:RETURN
20010 DATA 00,40,04,44,10,50,14,54,01,41,05,45,1
    1,51,15,55

```

```

10 ;
20 ;programme utilitaire -
30 ;partie assembleur gestion des scrollings et affichages
40 ;

8100      50      ORG #8100
8100 00      60      DEFB 0
8101 40      70      DEFB 64
8102 04      80      DEFB 4
8103 44      90      DEFB 68
8104 10      100     DEFB 16
8105 50      110     DEFB 80
8106 14      120     DEFB 20
8107 54      130     DEFB 84
8108 01      140     DEFB 1
8109 41      150     DEFB 65
810A 05      160     DEFB 5
810B 45      170     DEFB 69
810C 11      180     DEFB 17
810D 51      190     DEFB 81
810E 15      200     DEFB 21
810F 55      210     DEFB 85

220 ;SCROLLING VERS LE HAUT DE DEUX LIGNES
8110 DD215582 230 HAUT: LD IX, TABLE ;IX pointe sur 3e ligne ecran
                                           graphique

8114 DD5E00 240 LOOP: LD E, (IX+0)
8117 DD5601 250      LD D, (IX+1) ;DE=adresse ligne du haut
811A DD6E04 260      LD L, (IX+4)
811D DD6605 270      LD H, (IX+5) ;HL=adresse 2 lignes +bas
8120 015000 280      LD BC, 80 ;80 octets a tradater
8123 EDB0 290      LDIR ;transfert
8125 DD23 300      INC IX
8127 DD23 310      INC IX ;ligne suivante
8129 DD7E04 320      LD A, (IX+4)
812C DDB605 330      OR (IX+5) ;fin table ?
812F C21481 340      JP NZ, LOOP

350 ;remplissage des deux dernieres lignes
360 ;pour quadrillage
8132 DD2A9783 370      LD IX, (PARAM) ;adresse de la ligne dans
                                           l'image
8136 2180F7 380      LD HL, #F780 ;avant derniere
8139 0650 390      LD B, 80 ;80 octets a remplir
813B 0EAA 400      LD C, 170 ;point de gauche allume
813D 1681 410      LD D, #81 ;poids fort table masques

420 ;
813F DD7E00 430 PP1: LD A, (IX) ;couleur du point
8142 5F 440      LD E, A
8143 1A 450      LD A, (DE) ;masque du point
8144 B1 460      OR C ;ajoute point gauche
8145 DD23 470      INC IX ;en avant dans la ligne image
    
```

8147	77	480	LD	(HL),A	
8148	23	490	INC	HL	
8149	10F4	500	DJNZ	PP1	
		510 ;			
814B	2180FF	520	LD	HL,#FF80	;derniere
814E	0650	530	LD	B,80	
8150	3EFF	540	LD	A,255	
8152	77	550 PP2:	LD	(HL),A	
8153	23	560	INC	HL	
8154	10FC	570	DJNZ	PP2	
		580 ;			
8156	C9	590	RET		
		600 ;	SCROLLING VERS LE BAS DE DEUX LIGNES		
8157	DD218F83	610 BAS:	LD	IX,TABFIN	
815B	DD5E04	620 LAAP:	LD	E,(IX+4)	;2 lignes ecran +bas
815E	DD5605	630	LD	D,(IX+5)	
8161	DD6E00	640	LD	L,(IX+0)	;cette ligne
8164	DD6601	650	LD	H,(IX+1)	
8167	015000	660	LD	BC,80	;80 octets a transferer
816A	EDB0	670	LDIR		;copie de la ligne
816C	DD2B	680	DEC	IX	;descend dans la table
816E	DD2B	690	DEC	IX	
8170	DD7E00	700	LD	A,(IX+0)	;octet
8173	DDB601	710	OR	(IX+1)	;fin de la table ?
8176	C25B81	720	JP	NZ,LAAP	;non:continuer
		730 ;	remplissage quadrillage en haut		
8179	2190C1	740	LD	HL,#C190	;ligne a dessiner
817C	0650	750	LD	B,80	;80 octets a remplir
817E	0EAA	760	LD	C,170	;point de gauche allume
8180	1681	770	LD	D,#81	;poids fort table masques
8182	DD2A97B3	780	LD	IX,(PARAM)	;adresse ligne dans image
		790 ;			
8186	DD7E00	800 PLAP1:	LD	A,(IX+0)	;couleur point
8189	5F	810	LD	E,A	
818A	1A	820	LD	A,(DE)	;masque point
818B	B1	830	OR	C	;les deux points positionnes
818C	77	840	LD	(HL),A	
818D	23	850	INC	HL	
818E	DD23	860	INC	IX	
8190	10F4	870	DJNZ	PLAP1	
		880 ;			
8192	2190C9	890	LD	HL,#C990	
8195	0650	900	LD	B,80	
8197	3EFF	910	LD	A,255	
8199	77	920 PLAP2:	LD	(HL),A	
819A	23	930	INC	HL	
819B	10FC	940	DJNZ	PLAP2	
		950 ;			
819D	C9	960	RET		

```

770 ;SCROLLING VERS LA GAUCHE DE 2 POINTS
819E DD215582 980 GAUCHE: LD IX, TABLE
81A2 DD5E00 990 LUUP: LD E, (IX+0)
81A5 DD5601 1000 LD D, (IX+1)
81A8 6B 1010 LD L, E
81A9 62 1020 LD H, D
81AA 23 1030 INC HL
81AB 014F00 1040 LD BC, 79
81AE EDB0 1050 LDIR
81B0 DD23 1060 INC IX
81B2 DD23 1070 INC IX
81B4 DD7E00 1080 LD A, (IX+0)
81B7 DDB601 1090 OR (IX+1)
81BA C2A281 1100 JP NZ, LUUP
1110 ;remplissage quadrillage a droite
81BD FD2A9783 1120 LD IY, (PARAM)
81C1 DD215582 1130 LD IX, TABLE
81C5 1681 1140 LD D, #81 ;poids fort adresse masques
1150 ;
81C7 DD6E00 1160 PLUP1: LD L, (IX+0)
81CA DD6601 1170 LD H, (IX+1)
81CD 014F00 1180 LD BC, 79
81D0 09 1190 ADD HL, BC
81D1 FD7E00 1200 LD A, (IY) ;couleur point
81D4 5F 1210 LD E, A
81D5 1A 1220 LD A, (DE) ;masque point
81D6 F6AA 1230 OR 170 ;ajoute point gauche
81D8 77 1240 LD (HL), A
81D9 0160FF 1250 LD BC, -160
81DC FD09 1260 ADD IY, BC ;point du dessus dans image
81DE DD23 1270 INC IX
81E0 DD23 1280 INC IX
81E2 DD6E00 1290 LD L, (IX+0)
81E5 DD6601 1300 LD H, (IX+1)
81EB 014F00 1310 LD BC, 79
81EB 09 1320 ADD HL, BC
81EC 36FF 1330 LD (HL), 255
81EE DD23 1340 INC IX
81F0 DD23 1350 INC IX
81F2 DD7E00 1360 LD A, (IX+0)
81F5 DDB601 1370 OR (IX+1)
81F8 C2C781 1380 JP NZ, PLUP1
1390 ;
81FB C9 1400 RET
1410 ;SCROLLING VERS LA DROITE DE 2 POINTS
81FC DD215582 1420 DROITE: LD IX, TABLE
8200 DD6E00 1430 LEEP: LD L, (IX+0)
8203 DD6601 1440 LD H, (IX+1)
8206 014F00 1450 LD BC, 79
8209 09 1460 ADD HL, BC
    
```

820A	E5	1470	PUSH HL	
820B	D1	1480	POP DE	
820C	2B	1490	DEC HL	
820D	EDB8	1500	LDDR	
820F	DD23	1510	INC IX	
8211	DD23	1520	INC IX	
8213	DD7E00	1530	LD A,(IX+0)	
8216	DDB601	1540	OR (IX+1)	
8219	C20082	1550	JP NZ,LEEP	
		1560	;remplissage quadrillage a gauche	
821C	FD2A9783	1570	LD IY,(PARAM)	
8220	DD215582	1580	LD IX,TABLE	
8224	1681	1590	LD D,#81	
8226	DD6E00	1600	PLEP1: LD L,(IX+0)	
8229	DD6601	1610	LD H,(IX+1)	
822C	FD7E00	1620	LD A,(IY)	;couleur point
822F	5F	1630	LD E,A	
8230	1A	1640	LD A,(DE)	;masque point
8231	F6AA	1650	OR 170	;ajoute point gauche
8233	77	1660	LD (HL),A	;ok
8234	0160FF	1670	LD BC,-160	
8237	FD09	1680	ADD IY,BC	
8239	DD23	1690	INC IX	
823B	DD23	1700	INC IX	
823D	DD6E00	1710	LD L,(IX+0)	
8240	DD6601	1720	LD H,(IX+1)	
8243	36FF	1730	LD (HL),255	
8245	DD23	1740	INC IX	
8247	DD23	1750	INC IX	
8249	DD7E00	1760	LD A,(IX+0)	
824C	DDB601	1770	OR (IX+1)	
824F	C22682	1780	JP NZ,PLEP1	
		1790 ;		
8252	C9	1800	RET	
8253	0000	1810	DEFW #0000	
8255	90C190C9	1820	TABLE: DEFW #C190,#C990,#D190,#D990,#E190,#E990,#F190,#F990	
8265	E0C1E0C9	1830	DEFW #C1E0,#C9E0,#D1E0,#D9E0,#E1E0,#E9E0,#F1E0,#F9E0	
8275	30C230CA	1840	DEFW #C230,#CA30,#D230,#DA30,#E230,#EA30,#F230,#FA30	
8285	80C280CA	1850	DEFW #C280,#CA80,#D280,#DA80,#E280,#EA80,#F280,#FA80	
8295	D0C2D0CA	1860	DEFW #C2D0,#CAD0,#D2D0,#DAD0,#E2D0,#EAD0,#F2D0,#FAD0	
82A5	20C320CB	1870	DEFW #C320,#CB20,#D320,#DB20,#E320,#EB20,#F320,#FB20	
82B5	70C370CB	1880	DEFW #C370,#CB70,#D370,#DB70,#E370,#EB70,#F370,#FB70	
82C5	C0C3C0CB	1890	DEFW #C3C0,#CBC0,#D3C0,#DBC0,#E3C0,#EBC0,#F3C0,#FBC0	
82D5	10C410CC	1900	DEFW #C410,#CC10,#D410,#DC10,#E410,#EC10,#F410,#FC10	
82E5	60C460CC	1910	DEFW #C460,#CC60,#D460,#DC60,#E460,#EC60,#F460,#FC60	
82F5	B0C4B0CC	1920	DEFW #C4B0,#CCB0,#D4B0,#DCB0,#E4B0,#ECB0,#F4B0,#FCB0	
8305	00C500CD	1930	DEFW #C500,#CD00,#D500,#DD00,#E500,#ED00,#F500,#FD00	
8315	50C550CD	1940	DEFW #C550,#CD50,#D550,#DD50,#E550,#ED50,#F550,#FD50	
8325	A0C5A0CD	1950	DEFW #C5A0,#CDA0,#D5A0,#DDA0,#E5A0,#EDA0,#F5A0,#FDA0	

8335	F0C5F0CD	1960	DEFW	#C5F0,#CDF0,#D5F0,#DDF0,#E5F0,#EDF0,#F5F0,#FDF0	
8345	40C640CE	1970	DEFW	#C640,#CE40,#D640,#DE40,#E640,#EE40,#F640,#FE40	
8355	90C690CE	1980	DEFW	#C690,#CE90,#D690,#DE90,#E690,#EE90,#F690,#FE90	
8365	E0C6E0CE	1990	DEFW	#C6E0,#CEE0,#D6E0,#DEE0,#E6E0,#EEE0,#F6E0,#FEE0	
8375	30C730CF	2000	DEFW	#C730,#CF30,#D730,#DF30,#E730,#EF30,#F730,#FF30	
8385	80C780CF	2010	DEFW	#C780,#CF80,#D780,#DF80,#E780	
838F	80EF80F7	2020	TABFIN:	DEFW	#EF80,#F780,#FF80
8395	0000	2030	DEFW	#0000	
8397	0000	2040	PARAM:	DEFW	0
		2050			;
		2060			;EFFACEMENT IMAGE
		2070			;
8399	21001C	2080	CLS:	LD	HL,#1C00
839C	1600	2090		LD	D,0
839E	010064	2100		LD	BC,160*160
83A1	72	2110	POPOP:	LD	(HL),D
83A2	23	2120		INC	HL
83A3	0B	2130		DEC	BC
83A4	78	2140		LD	A,B
83A5	B1	2150		OR	C
83A6	C2A183	2160	JP	NZ,POPOP	
83A9	C9	2170	RET		
		2180			;
		2190			;copie ecran de l'image
		2200			;
83AA	DD219383	2210	FULL:	LD	IX,TABFIN+4
83AE	FD21001C	2220		LD	IY,#1C00
83B2	2681	2230		LD	H,#81
		2240			;
83B4	DD5E00	2250	PF:	LD	E,(IX+0)
83B7	DD5601	2260		LD	D,(IX+1)
83BA	0650	2270		LD	B,80
		2280			;
83BC	FD7E00	2290	PF2:	LD	A,(IY+0)
83BF	6F	2300		LD	L,A
83C0	7E	2310		LD	A,(HL)
83C1	CB27	2320	SLA	A	
83C3	4F	2330		LD	C,A
		2340			;
83C4	FD23	2350	INC	IY	
83C6	FD7E00	2360		LD	A,(IY+0)
83C9	6F	2370		LD	L,A
83CA	7E	2380		LD	A,(HL)
83CB	B1	2390	OR	C	
83CC	12	2400		LD	(DE),A
		2410			;
83CD	FD23	2420	INC	IY	
83CF	13	2430	INC	DE	
83D0	10EA	2440	DJNZ	PF2	

;derniere adresse=bas ecran ptr
 ;debut image
 ;poids fort table masques
 ;adresse ecran
 ;80 octets a inscrire
 ;couleur point gauche
 ;masque du point
 ;decalage a gauche
 ;point suivant=a droite
 ;couleur point droite
 ;masque point
 ;ajoute point deja inscrit
 ;ajoute point droite a l'ecran


```

                2450 ;
83D2 DD2B      2460      DEC IX
83D4 DD2B      2470      DEC IX                ;continue dans la table en
                                                remontant

83D6 DD7E00    2480      LD  A,(IX+0)
83D9 DDB601    2490      OR  (IX+1)            ;fin table ?
83DC C2B483    2500      JP  NZ,PF
83DF C9        2510      RET

                2520 ;
                2530 ;copie image dans grille
                2540 ;

83E0 DD219383  2550 WIDE: LD  IX,TABFIN+4      ;table ecran
83E4 FD2A9783  2560      LD  IY,(PARAM)        ;adresse debut fenetre image
83E8 2681      2570      LD  H,#81              ;poids forts masques

                2580 ;
83EA DD5E00    2590 PV:  LD  E,(IX+0)
83ED DD5601    2600      LD  D,(IX+1)            ;adresse ecran reelle
83F0 0650      2610      LD  B,80              ;40 octets a ecrire
83F2 3EFF      2620      LD  A,255

                2630 ;
83F4 12        2640 PV2: LD  (DE),A
83F5 13        2650      INC DE
83F6 10FC      2660      DJNZ PV2

                2670 ;
83F8 DD2B      2680      DEC IX
83FA DD2B      2690      DEC IX
83FC DD5E00    2700      LD  E,(IX+0)
83FF DD5601    2710      LD  D,(IX+1)
8402 0650      2720      LD  B,80

                2730 ;
8404 FD7E00    2740 PV3: LD  A,(IY+0)          ;couleur point
8407 6F        2750      LD  L,A
8408 7E        2760      LD  A,(HL)            ;masque point
8409 F6AA      2770      OR  170              ;pour grille
840B 12        2780      LD  (DE),A          ;stockage ecran
840C 13        2790      INC DE              ;avance ecran
840D FD23      2800      INC IY              ;avance image
840F 10F3      2810      DJNZ PV3            ;continue copie ligne

                2820 ;
8411 DD2B      2830      DEC IX
8413 DD2B      2840      DEC IX
8415 115000    2850      LD  DE,80
8418 FD19      2860      ADD IY,DE            ;sauter a la ligne du dessus !
841A DD7E00    2870      LD  A,(IX+0)
841D DDB601    2880      OR  (IX+1)
8420 C2EA83    2890      JP  NZ,PV
8423 C9        2900      RET

```

Mode d'emploi

Le programme comporte trois modes, la touche ENTER ou RETURN permettant le passage du 1 au 2, du 2 au 3, ou du 3 au 1.

Le mode 1 est "CURSEUR TRACE". Dans ce mode, le curseur clignote dans la grille. Vous pouvez alors le déplacer avec les touches flèches du clavier. La couleur de tracé est indiquée au-dessus du dessin par une lettre en vidéo inverse. Il y a 16 lettres, représentant les 16 stylos (la lettre en elle-même indique la couleur du stylo : A pour 0, B pour 1, jusqu'à "]" pour 26). Le symbole suivant, " ", représente un stylo fictif transparent.

Pour sélectionner un stylo de tracé dans le mode 1, il suffit de taper la lettre qu'il représente. Le dernier stylo ("@") permet de déplacer le curseur sans effacer le dessin.

La flèche qui se trouve au-dessus d'une des lettres indique en quel stylo le point sous le curseur est tracé. Enfin, la touche "↑" (à côté de CLR, et non sur le pavé des flèches curseur) permet de visualiser l'écran en taille normale. Un deuxième appui de cette touche ramène l'écran en zoom 4/1.

Le mode 2 permet de modifier les couleurs associées aux stylos, les flèches curseur horizontales choisissent le stylo, et les flèches verticales la couleur. La touche "↑" (à gauche de CLR) affiche l'écran en entier et taille normale, permettant ainsi de choisir les couleurs plus facilement. Il faut appuyer une deuxième fois sur la touche avant de passer en mode 1 ou 3.

Le mode 3 autorise quatre opérations. Les deux premières, "SAUVE ECRAN" et "LOAD", effectuent une sauvegarde ou un chargement de tout l'écran (avec tout ce qui est dessiné, y compris les zones non affichées à cause du mode Zoom) et des couleurs de stylos. On peut ainsi sauver d'un bloc tout le travail en cours et le reprendre ensuite. Attention : les fichiers IME et PAE créés par cette option ne sont pas directement récupérables. Le fichier IME contenant l'image utilise en effet un format de stockage particulier (26 Ko de mémoire) afin d'optimiser les travaux de dessin. Pour transformer ce fichier en image écran utilisable, il suffit d'utiliser l'utilitaire destiné à cet effet (voir partie II).

La fonction "↑" (touche à côté de CLR) permet d'avoir une vision globale de l'image, comme dans les autres modes. Et enfin, "D" permet de sauvegarder un objet dans un fichier binaire. Pour cela, il faut indiquer le point supérieur gauche de l'objet, puis inférieur droit. Le programme affiche la largeur et la hauteur en nombre de points, il faut les noter. Ensuite, le programme demande à quelle adresse il doit placer le codage binaire de l'objet. Cette adresse doit être choisie de façon à ne pas écraser les routines LM du programme. On peut le placer n'importe où entre \$1C00 et \$7000, par exemple. Il convient dans ce cas de sauver préalablement l'écran (avec l'option "S") car l'objet créé à partir de l'écran peut écraser celui-ci sur une petite partie (pour faciliter les traitements, le programme utilise en effet une image logique de l'écran complet à partir de l'adresse 1C00).

Le programme affiche, tant que l'objet n'est pas défini, les valeurs de LAR et HAU afin de vous aider à leur attribuer une valeur précise. Cela permet par exemple de créer sans erreur différentes phases d'animation d'un objet.

Les fichiers IME et PAE

L'option commande "S" du programme envoie dans un fichier d'extension IME ("IMage Ecran") le contenu de l'image. Ce fichier peut être rechargé ou par le programme de dessin (afin de continuer le travail interrompu) ou par l'utilitaire de création d'image, afin de le transformer en véritable image écran, laquelle pourra être chargée en mémoire écran par une commande LOAD en Basic. Les variables de travail associées à cette image dans le programme de dessin (c'est-à-dire l'endroit du curseur de tracé, de la fenêtre d'affichage, etc.) sont sauvegardées dans un fichier de même nom que IME mais d'extension PAE "PAramètres Ecran"). Les couleurs associées aux 16 stylos dans ce dessin sont également sauvées dans ce fichier.

Il ne faut pas effacer le fichier PAE : en effet, les utilitaires suivants l'utilisent pour récupérer les couleurs de stylos.

Les fichiers IMO et PAO

L'option "D" du programme crée les fichiers IMO et PAO de façon similaire à "S". Les fichiers d'extension IMO ("IMage Objet") représentent le contenu exact d'un objet graphique ; il est donc possible de les récupérer sans formalités par une commande LOAD en Basic. Le fichier PAO ("PAramètres Objet") de même nom contient les données caractéristiques de cet objet : largeur (variable LAR), hauteur (variable HAU), et couleurs des 16 stylos. Si le programme d'application s'assigne de lui-même ces valeurs, le fichier PAO peut être effacé. Néanmoins, l'utilitaire de compactage d'objets livré dans les pages suivantes l'utilise pour vérifier l'homogénéité des données. Les données sont stockées dans le fichier de la façon suivante :

```
LAR , HAU
couleur stylo 0
couleur stylo 1
... etc ..
couleur stylo 2
fin du fichier.
```

On peut les récupérer par la séquence suivante :

```
OPENIN "nom.PAO"
INPUT #9,lar,hau
FOR i=0 TO 15
  INPUT#9,c
  INK i,c
NEXT i
CLOSEIN
```

Lorsque vous utilisez le programme pour créer différentes phases d'animation d'un même objet, vous devez prendre garde aux points suivants :

- toutes les phases sauvées sous forme d'objet par la commande "D" doivent avoir des valeurs LAR et HAU identiques ;
- les couleurs des 16 stylos doivent être identiques pour chaque phase ainsi sauvegardée. Il convient donc de les choisir avant tout dessin.

CRÉATION DE DÉCOR

Le fichier décor

Cet utilitaire permet la création d'une image écran véritable à partir d'un fichier IME créé par le programme précédent.

Le programme affiche avant tout la liste des fichiers IME disponibles. L'utilisateur indique alors son choix. Le programme crée, à partir du fichier choisi, un fichier de même nom et d'extension SCR ("SCReen", c'est-à-dire Ecran en anglais) qui correspond directement au contenu de la mémoire écran. Ensuite, l'utilisateur a la possibilité d'effacer le fichier IME original. Ceci est recommandé lors de la création de décors, afin de récupérer de la place sur la disquette (le fichier IME occupe 26 Ko). Néanmoins, dans certains cas, vous pouvez ressentir le besoin de garder ce fichier. Vous pouvez par exemple récupérer une image et la retravailler ultérieurement, ce qui ne sera pas possible si vous effacez le fichier IME.

```
10 MODE 2:INK 0,0:INK 1,15:BORDER 0:PRINT"**** C
  REATION DE FICHIER ECRAN ****"
20 PRINT:PRINT"LISTE DES FICHIERS DISPONIBLES:":
  AS="*.IME":DIR,AS
30 LOCATE 1,VPOS(10)-2:INPUT"Votre choix ";nf$:I
  F INSTR(nf$,".") THEN nf$=LEFT$(nf$,INSTR(nf$,
  ".")-1)
40 MEMORY &1BFF:LOAD"scrolls":LOAD nf$+"*.ime",&1
  C000
```

```

50 OPENIN nf$+".pae":INPUT#9,a,a,a,a,a,a,a:FOR i
  =0 TO 15:INPUT#9,c:INK i,c:NEXT:CLOSEIN
60 MODE 0:CALL &83AA:SAVE nf$+".scr",b,&C000,163
  84:a$="*.bak":!ERA,@a$
70 INK 1,15
80 MODE 2:PRINT "Vous pouvez maintenant effacer
  ";
  r$:IF UPPER$(a$)="OUI" THEN a$=nf$+".ime":!ERA
    ,@a$
90 MODE 0:INK 0,0:INK 1,10:CALL &83AA:WINDOW #0,
  1,20,1,5
100 PRINT "SAISIE TERRITOIRES"
110 ad=&1C00:PRINT"-X- fin de saisie":x1=0:y1=0
120 x=x1:y=y1:h$=CHR$(240):b$=CHR$(241):g$=CHR$(
  242):d$=CHR$(243):LOCATE 1,3:PRINT"HAUT GAUCH
  E (" +g$+h$+d$+b$+"");
130 z$=INKEY$:IF z$="x" OR z$="X" THEN 320:'fin
140 IF z$=CHR$(13) THEN 210:'droite
150 IF z$=h$ THEN y=y-1:IF y<0 THEN y=0:GOTO 190
  ELSE 190
160 IF z$=b$ THEN y=y+1:IF y>199 THEN y=199:GOTO
  190 ELSE 190
170 IF z$=g$ THEN x=x-1:IF x<0 THEN x=0:GOTO 190
  ELSE 190
180 IF z$=d$ THEN x=x+1:IF x>159 THEN x=159:GOTO
  190 ELSE 190
190 PRINT CHR$(23)CHR$(1);:PLOT x*4,399-y*2,15:P
  LOT x*4,399-y*2,15
200 GOTO 130
210 POKE ad,INT(x/2):POKE ad+1,y:LOCATE 1,3:PRIN
  T"BAS DROITE (" +g$+h$+d$+b$+"");:x1=x:y1=y
220 z$=INKEY$:IF z$="x" OR z$="X" THEN 320:'fin
230 IF z$=CHR$(13) THEN 300:'fin
240 IF z$=h$ THEN y=y-1:IF y<0 THEN y=0:GOTO 280
  ELSE 280
250 IF z$=b$ THEN y=y+1:IF y>199 THEN y=199:GOTO
  280 ELSE 280
260 IF z$=g$ THEN x=x-1:IF x<0 THEN x=0:GOTO 280
  ELSE 280
270 IF z$=d$ THEN x=x+1:IF x>159 THEN x=159:GOTO
  280 ELSE 280
280 PRINT CHR$(23)CHR$(1);:FOR i=1 TO 2:PLOT x1*
  4,399-2*y1:DRAW x1*4,399-y*2:DRAW x*4,399-y*2
  :DRAW x*4,399-y1*2:DRAW x1*4,399-2*y1:NEXT
290 GOTO 220
300 POKE ad+2,INT((x-x1+1)/2):POKE ad+3,y-y1+1:a
  d=ad+4:PRINT CHR$(23)CHR$(0);:FOR x2=4*x1 TO
  4*x
310 PLOT x2,399-2*y1,15:DRAW x2,399-2*y:NEXT:x1=
  x:y1=y:GOTO 120
320 POKE ad,255:SAVE nf$+".ter",b,&1C00,ad-&1C00
  +1:a$="*.bak":!ERA,@a$:RUN

```


Les territoires interdits

Une fois le fichier SCR créé, le programme affiche l'image et vous propose de rentrer les territoires interdits un à un. Pour cela, il suffit de déplacer le curseur sur le point situé en haut à gauche d'un territoire, de frapper ENTER ou RETURN, puis de même pour le point inférieur droit de la zone. Le territoire correspondant est rempli par le programme afin de le distinguer du reste du décor. Vous pouvez ainsi mémoriser plusieurs territoires interdits. Une fois le travail achevé, une pression sur la touche "X" du clavier permet la création du fichier d'extension TER contenant les données des territoires ainsi stockées. Ce fichier est directement utilisable sous Basic : il contient l'image exacte de la table des territoires interdits, selon la structure définie au chapitre 8. Un programme d'application utilisant la routine 8.4 pourra donc récupérer la table par une instruction LOAD "nom.TER", adresse suivie de deux POKes destinés à indiquer à la routine l'adresse de la table (voir le programme d'application 8.4).

COMPACTAGE DE PHASES D'ANIMATION D'UN OBJET

Mode d'emploi

Ce programme permet de compacter en mémoire les différentes phases d'animation d'un même objet. Il suppose avant tout que ces phases aient été créées par le programme de dessin (option commande "D"). Les fichiers PAO doivent également être présents.

Avant toute chose, le programme vous propose d'effacer les fichiers PAO et IMO de chaque phase au fur et à mesure de leur compactage en mémoire. Pour l'accepter, il faut entrer OUI en toutes lettres. Cette option n'est utile que dans le cas où vous êtes bien certain d'avoir fini de travailler sur ces phases. De même que pour le programme de création de décor, elle permet surtout de récupérer de la place sur une disquette. Mais si vous avez gardé le fichier IME contenant les phases, il vous sera facile de recréer les fichiers IMO et PAO correspondants à partir de celui-ci, par l'intermédiaire du programme de dessin.

Ensuite, le programme affiche la liste des fichiers image objets dont il dispose sur la disquette. L'utilisateur peut entrer, un par un, les noms des fichiers contenant les phases. Cela fait, il choisit un nom pour le fichier final (celui qui contiendra les phases compactées en mémoire).


```

10 DIM co(15)
20 MODE 2:INK 0,0:INK 1,10:BORDER 0
30 PRINT"**** CREATEUR DE FICHIERS DESSIN ****"
40 MEMORY &2FFF
50 PRINT:PRINT:INPUT "Dois-je effacer les fichiers originaux de chaque phase (OUI/NON)";a$:flg=(UPPER$(a$)="OUI")
60 a$="*.imo":!DIR,@a$
70 PRINT "Entrez les noms des fichiers contenant "
80 PRINT"les phases de l'objet un par un, puis
90 PRINT"tapez ENTER seul pour finir ":PRINT
100 fi$="." :no=0:ad=&3000
110 WHILE fi$<>" "
120 no=no+1
130 PRINT "Fichier No";no;:INPUT "Nom ";fi$:IF INSTR(fi$,".") THEN fi$=LEFT$(fi$,INSTR(fi$,".")-1)
140 IF fi$="" THEN 200
150 OPENIN fi$+".pao":INPUT #9,lar,hau:IF no>1 THEN IF lar<>lar1 OR hau<>hau1 THEN PRINT"CONFLIT DE TAILLE !":GOTO 130 ELSE ELSE hau1=hau:lar1=lar
160 FOR i=0 TO 15:INPUT#9,c:IF no>1 THEN IF c<>c0(i) THEN PRINT"CONFLIT DE COULEUR":GOTO 130 ELSE ELSE co(i)=c
170 NEXT
180 LOAD fi$+".imo",ad:ad=ad+lar*hau
190 IF flg THEN a$=fi$+".imo":!ERA,@a$:a$=fi$+".pao":!ERA,@a$
200 WEND
210 INPUT "Sous quel nom dois-je sauver l'ensemble des images ";fi$:IF INSTR(fi$,".") THEN fi$=LEFT$(fi$,INSTR(fi$,".")-1)
220 SAVE fi$+".obj",b,&3000,ad-&3000+1:OPENOUT fi$+".par":PRINT#9,lar1,hau1,no-1:FOR i=0 TO 15:PRINT#9,co(i):NEXT:PRINT#9,ad+1:CLOSEOUT
230 RUN

```

Les fichiers OBJ et PAR

Le programme crée deux fichiers : le premier, d'extension OBJ ("OBJet") est la réplique exacte de la mémoire, de la première phase à la dernière. Il est possible de charger ce fichier par une commande LOAD "nom.OBJ",adresse. Le second fichier, d'extension PAR, contient les

paramètres associés à ce fichier Obj. On les récupère par la séquence Basic suivante :

```

OPENIN "nom.PAR"
INPUT#9,lar,hau,nombre de phases
FOR i=0 TO 15
  INPUT#9,c
  INK i,c
NEXT i
CLOSEIN

```

Les adresses de début de chaque phase de l'objet peuvent alors être calculées facilement : si le fichier "nom.OBJ" a été chargé à l'adresse ADD, les adresses de chaque phase sont les suivantes :

```

ADD
ADD + lar * hau
ADD + lar * hau * 2
ADD + lar * hau * 3
et ainsi de suite.

```

Annexe 1

Les mathématiques de l'informatique

L'homme a dix doigts et, depuis ce temps-là, il compte habituellement en utilisant la base 10.

Lorsque l'homme de Cromagnon, du haut de son rocher, voulait dire à son collègue se trouvant non loin de là, qu'il pouvait dénombrer 58 aurochs dans la plaine, il devait d'abord lever 8 doigts (unité), puis un instant plus tard, 5 doigts (dizaines, nombre de mains pleines) pour terminer par un geste démonstratif signifiant quelque chose comme "miam... miam !"

Il faut dire qu'ils avaient la vue perçante en ce temps-là, mais tout de même, il y avait des limites, et lorsque la distance était trop importante, il fallait utiliser une autre méthode : les deux bras, par exemple.

Mais si, avec les deux mains, on pouvait compter jusqu'à 10 (base 10), avec les deux bras, on ne peut compter que jusqu'à 2 (base 2). De plus, il ne faut pas oublier que le nombre correspondant à la base n'est jamais utilisé : 10 aurochs auraient été codés avec les mains :

- temps 1 : 0 doigt levé (unité)
- temps 2 : 1 doigt levé (dizaine)
- temps 3 : miam... miam !

De même, 58 aurochs seront codés comme suit :

temps 1 : 8 unités ---> 8
 temps 2 : 5 fois la base = $5 * 10 \text{ --->}$ 50
 temps 3 : miam... miam !

Et s'il y en a 1 253 :

temps 1 : 3 unités		=	3
temps 2 : 5 fois la base	= $5*10$	=	50
temps 3 : 2 fois la base fois la base	= $2*10*10$	=	200
temps 4 : 1 fois...	= $1*10*10*10$	=	1 000
temps 5 : miam...miam !			
			<hr/> 1 253

Avec les bras (base 2), 58 aurochs devront être codés :

temps : 1 2 3 4 5 6

C'est-à-dire :

temps 1 : 0 unité		0
temps 2 : 1 fois la base	= $1*2$	2
temps 3 : 0 fois la base fois la base	= $0*2*2$	0
temps 4 : 1 fois	= $1*2*2*2$	8
temps 5 : 1 fois	= $1*2*2*2*2$	16
temps 6 : 1 fois	= $1*2*2*2*2*2$	32
temps 7 : miam... miam !		<hr/> 58

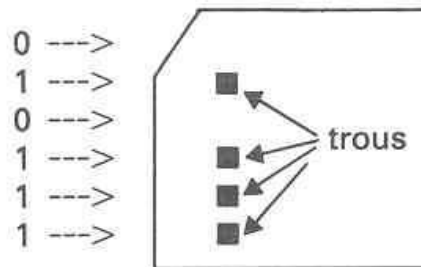
L'avantage de la méthode était évident ; elle laissait à l'homme un bras libre, ce qui lui permettait d'en changer lorsqu'il y avait beaucoup d'aurochs, car dans ce cas-là, la "transmission" était longue et fastidieuse...

Ainsi, comme vous le voyez, 58 en base 10 et 111010 en base 2 sont deux représentations d'un même nombre (il y en a autant que de bases, c'est-à-dire un nombre infini). L'une est la représentation décimale, l'autre la représentation binaire.

Comme le bras de l'homme de Cromagnon qui ne pouvait prendre que deux états (bras levé ou bras baissé), les informations traitées par l'ordinateur seront codées sur le même principe :

- **Etat 0** : pas de circulation de courant, pas de trou sur une carte ou un ruban perforé, pas de magnétisation sur une cassette ou une disquette.
- **Etat 1** : circulation d'un courant, trou dans une carte ou un ruban perforé, magnétisation sur un support magnétique.

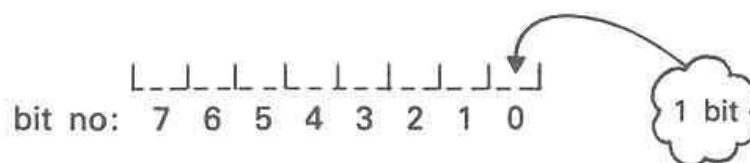
Le nombre 58 pourra alors être représenté en binaire sur une carte perforée :



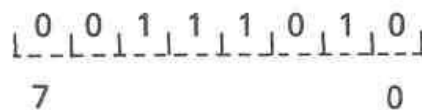
La mémoire centrale est formée d'une mosaïque de cellules appelées *bits*, chacun d'eux pouvant prendre l'un des deux états 0 ou 1 (bascule).

Habituellement, ces bits sont regroupés par huit pour former un octet, et la plupart des microprocesseurs utilisent l'adressage octet, ce qui signifie que l'on accède simultanément à 8 bits de la mémoire, en lecture ou en écriture.

L'octet peut être représenté ainsi :



et le nombre 58 pourra être codé :



avec le bit 0 représentant le poids faible du nombre, et le bit 7 représentant le poids fort, la notion de poids dépendant essentiellement de la valeur de la puissance 2 du rang occupé par chaque bit :

bit 0	--->	2^0	=	1
bit 1	--->	2^1	=	2
bit 2	--->	2^2	=	4
bit 3	--->	2^3	=	8
bit 4	--->	2^4	=	16
bit 5	--->	2^5	=	32
bit 6	--->	2^6	=	64
bit 7	--->	2^7	=	128

Plus le rang est élevé, plus le poids est lourd. Notre nombre 58, dans sa représentation binaire, a les bits 1, 3, 4 et 5 positionnés à 1. Il est donc facile de retrouver sa valeur décimale :

bit 1 --->	2
bit 3 --->	8
bit 4 --->	16
bit 5 --->	32
	<hr/>
	58

A l'inverse, un nombre décimal devra être divisé par les puissances successives de 2 (la base) afin de trouver sa configuration binaire.

Exemple : $133 = 2^7 + 2^2 + 2^0 = 128 + 4 + 1$

correspondant à :

1	0	0	0	0	1	0	1
bits : 7				2		0	

Mais cette représentation est lourde et encombrante pour l'homme. Il a donc été décidé de couper l'octet en 2 quartets (4 bits) :

1	0	0	0	0	1	0	1

et de faire correspondre à chaque quartet 16 symboles différents et uniques :

0000	correspondra à	0
0001		1
0010		2
0011		3
0100		4
0101		5
0110		6
0111		7
1000		8
1001		9
1010	correspondra à :?

Les symboles numériques sont épuisés ? Qu'à cela ne tienne. Employons les alphabétiques :

1010	correspondra à :	A
1011		B
1100		C
1101		D
1110		E
1111		F

Voilà ! Toutes les combinaisons possibles du quartet ont été écrites, et il faut s'arrêter là.

Par cette opération, nous venons, sans le savoir (vraiment ?) de passer de la base 2 (binaire) à la base 16 (hexadécimale), avec l'avantage d'une légère contraction d'écriture (15 devient F). Tous ces systèmes de numération ne sont, rappelons-le, que différents modes de représentation des mêmes nombres. Le tout est de connaître la base utilisée.

Un petit conseil : apprenez par cœur le tableau de correspondance binaire/hexadécimal. Allons... ce n'est pas si difficile !

Le nombre 58 (base 10) pourra être représenté par 3A en base 16, ce qui est tout de même plus agréable que 00111010..., enfin, chacun ses goûts !

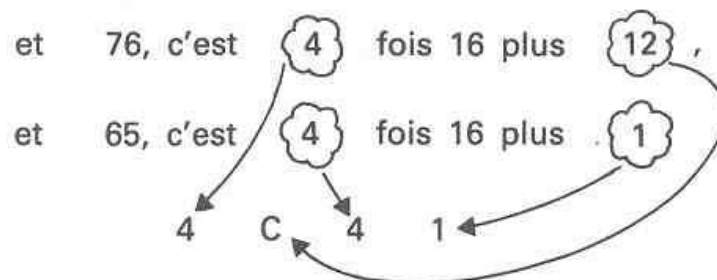
Bien entendu, au-delà d'un octet, le processus continue :

0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	1
14				11 10				6				0			

correspondra à 4C41 en base 16 (ça, c'est facile), et à :

$$2^{14} + 2^{11} + 2^{10} + 2^6 + 1 \text{ en base 10, soit } 19521.$$

Mais 19521... c'est 76 fois 256 plus 65,



Nous venons de repasser en base 16 ! Il est ensuite plus facile de basculer en base 2 si l'on connaît ses classiques :

4	C	4	1
↓	↓	↓	↓
0100	1100	0100	0001

Puisque nous en sommes au binaire, essayons d'additionner deux nombres : $1 + 1 = 2... ?$

Non. En base 2, le 2 n'existe pas. Il faut donc ajouter une tranche supplémentaire au résultat (comme pour $5 + 5$ en base 10) :

$$\begin{array}{r} \textcircled{1} \\ 1 \\ 1 \\ 0 \\ + \\ \hline 1 \textcircled{0} \end{array}$$

Essayons encore :

$$\begin{array}{r} \textcircled{1} \textcircled{1} \textcircled{1} \textcircled{1} \\ 1 \textcircled{0} \textcircled{1} \textcircled{1} \textcircled{0} \textcircled{1} \textcircled{0} \textcircled{0} \\ + \quad 0 \textcircled{0} \textcircled{1} \textcircled{0} \textcircled{1} \textcircled{1} \textcircled{1} \textcircled{0} \\ \hline 1 \textcircled{1} \textcircled{1} \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{1} \textcircled{0} \end{array}$$

Le microprocesseur ne procède pas différemment pour effectuer une addition. En hexadécimal, c'est un peu moins simple, sauf si l'on sait compter sur ses doigts...

$$\begin{array}{r} \textcircled{1} \\ 1 \textcircled{A} \textcircled{6} \textcircled{5} \\ + \quad 0 \textcircled{C} \textcircled{3} \textcircled{5} \\ \hline 2 \textcircled{6} \textcircled{9} \textcircled{A} \end{array}$$

On compte comme en base 10, mais au-delà de 9, on continue par A. Il n'y a une retenue qu'au-delà de F.

Un autre type de représentation est quelquefois utilisé en informatique : c'est la représentation BCD, qui signifie Décimal Codé Binaire (de l'anglais Binary Coded Decimal).

On l'emploie en général dans des programmes exécutant des opérations directement en décimal, sans passer par le binaire.

Les chiffres décimaux prennent les valeurs 0 à 9, on fait correspondre, à chaque chiffre du nombre, un quartet codé ainsi :

0000	--->	0
0001	--->	1
0010	--->	2
....		
1001	--->	9

La codification est la même que pour l'hexadécimal, mais cette fois on s'arrête à 9. Ainsi, le nombre 1794 sera représenté :

1	7	9	4
↙	↙	↙	↙
0001	0111	1001	0100

Pour terminer, nous allons dire un mot au sujet des opérateurs logiques.

Il existe 4 fonctions logiques fréquemment utilisées sur les nombres binaires : le NON (NOT), le ET (AND), le OU (OR) et le OU EXCLUSIF (XOR).

Le NON, nous l'avons déjà vu : c'est le complément à 1. Le ET (ou intersection logique) de deux nombres binaires s'effectue selon la règle suivante :

0 et 0	--->	0
0 et 1	--->	0
1 et 0	--->	0
1 et 1	--->	1

appliquée à chaque couple de bits de deux nombres à intersecter.

Exemple :

	1	0	1	0	1	1	0	1
ET	0	1	1	1	1	0	1	1
<hr/>								
	0	0	1	0	1	0	0	1

Sous forme hexadécimale, nous écrirons que : AD AND 7B = 29. Quoi de plus simple ?

Le bit résultant sera à 1 si le bit du premier nombre ET le bit du second nombre sont à 1.

Le OU (ou réunion logique) obéit à la règle :

0 et 0	--->	0
0 et 1	--->	1
1 et 0	--->	1
1 et 1	--->	1

Exemple :

	1	0	1	0	1	1	0	1
OU	1	1	0	0	0	1	0	1
<hr/>								
	1	1	1	0	1	1	0	1

que l'on peut aussi écrire : AD OR C5 = ED en hexadécimal. Le bit résultant sera à un si le bit du premier nombre OU le bit du second nombre est à 1.

Le OU exclusif (ou disjonction logique) obéit à la même règle que le précédent, sauf pour la dernière condition :

0	et	0	---	>	0
0	et	1	---	>	1
1	et	0	---	>	1
1	et	1	---	>	0

Pour que le bit résultant soit à 1, il faut donc que le bit du premier nombre OU le bit du second nombre soit à 1, mais pas les deux en même temps.

Exemple :

		0	1	0	1	1	1	0	1
OU EX		0	1	1	0	1	0	0	0
		<hr/>							
		0	0	1	1	0	1	0	1

Annexe 2

Les adresses systèmes utiles

Sont regroupés ci-dessous les appels de routines systèmes concernant la gestion des graphismes et des entrées utilisateur (clavier, joystick...). Toutes ne sont pas présentes, pour une raison d'encombrement et de simplification. Il s'agit des routines principales. Elles sont mises en œuvre par un simple CALL à l'adresse indiquée, après avoir éventuellement rempli les registres nécessaires. L'ouvrage *Clefs pour l'Amstrad* de Daniel Martin (Éditions du P.S.I.) contient la liste complète des routines systèmes et leurs entrées et sorties. Le manuel *Firmware* d'Amsoft est également disponible, et fournit une explication plus détaillée des routines et du système, mais en anglais.

- ☐ #**BB06** : attend un caractère au clavier ;
SORTIE : le registre A contient le code ASCII du caractère tapé ;
le flag Carry vaut 1 ;
les flags sont modifiés.
- ☐ #**BB09** : regarde si caractère est disponible au clavier ;
SORTIE : si Carry=1, A contient le code du caractère ;
si Carry=0, il n'y avait pas de caractère ;
les flags sont modifiés.

- **#BB18** : attend qu'une touche soit appuyée au clavier.
 SORTIE : Carry=1
 A contient le caractère associé à la touche ;
 flags modifiés.
- **#BB1B** : regarde si une touche est appuyée ;
 SORTIE : si Carry=1, A contient le caractère associé ;
 si Carry=0, il n'y avait pas de touche ;
 flags modifiés.
- **#BB1E** : teste si une touche donnée est appuyée ;
 ENTREE : numéro de la touche ;
 SORTIE : flag Z=0 si la touche était appuyée, 1 sinon ;
 A et HL modifiés ;
 flags modifiés.
- **#BB24** : renvoie l'état du joystick.
 SORTIE : H et A = joystick 0
 L = joystick 1
 Les bits de l'octet état indiquent :
 b0=1 si direction haut
 b1=1 si direction bas
 b2=1 si direction gauche
 b3=1 si direction droite
 b4=1 si bouton tir 2
 b5=1 si bouton tir 1
 Les flags sont modifiés.
- **#BBC0** : MOVE se positionne sur un point graphique ;
 ENTREE : DE contient la coordonnée x sur 16 bits
 HL contient y ;
 SORTIE : AF, BC, DE, HL modifiés (flags modifiés).
- **#BBC3** : MOVER se déplace graphiquement ;
 ENTREE : DE contient le déplacement sur x en nombre de points ;
 HL contient le déplacement sur y ;
 SORTIE : AF, BC, DE, HL modifiés (flags aussi).
- **#BBC6** : renvoie le point de localisation actuel ;
 SORTIE : DE contient x ;
 HL contient y ;
 AF modifié (flags aussi).
- **#BBDB** : efface la fenêtre graphique.
 SORTIE : AF,BC,DE,HL modifiés (et flags) ;
 le point actuel est placé sur l'origine (0,0).
- **#BBDE** : sélectionne le stylo utilisé pour les tracés graphiques ;
 ENTREE : A contient le numéro de stylo ;
 SORTIE : AF et flags modifiés.

- ☐ **#BBE1** : renvoie le numéro du stylo graphique ;
 SORTIE : A contient le numéro ;
 flags modifiés.
- ☐ **#BBE4** : sélectionne le numéro de stylo utilisé pour le fond graphique ;
 ENTREE : A contient le numéro de stylo ;
 SORTIE : A et flags modifiés.
- ☐ **#BBEA** : PLOT colorie un point ;
 ENTREE : DE contient x ;
 HL contient y ;
 SORTIE : AF,BC,DE,HL et flags modifiés.
- ☐ **#BBED** : PLOTR colorie un point après déplacement ;
 ENTREE : DE contient déplacement sur x ;
 HL déplacement sur y ;
 SORTIE : AF,BC,DE,HL et flags modifiés.
- ☐ **#BBFO** : TEST renvoie le numéro de stylo d'un point ;
 ENTREE : DE contient x ;
 HL contient y ;
 SORTIE : A contient le numéro de stylo ;
 BC,DE,HL et flags modifiés.
- ☐ **#BBF3** : TESTR renvoie stylo d'un point après déplacement ;
 ENTREE : DE déplacement sur x ;
 HL déplacement sur y ;
 SORTIE : A contient numéro de stylo ;
 BC,DE,HL et flags modifiés.
- ☐ **#BBF6** : DRAW trace une ligne du point actuel à un point spécifié ;
 ENTREE : DE contient x du point visé ;
 HL contient y du point visé ;
 SORTIE : AF,BC,DE,HL et flags modifiés.
- ☐ **#BBF9** : DRAWR déplacement et tracé jusqu'à un point ;
 ENTREE : DE déplacement sur x ;
 HL déplacement sur y ;
 SORTIE : AF,BC,DE,HL et flags modifiés.
- ☐ **#BCOE** : MODE positionne l'écran dans un mode de résolution ;
 ENTREE : A contient le mode voulu (0,1 ou 2) ;
 SORTIE : AF,BC,DE,HL et flags perdus ;
 les couleurs associées aux stylos ne sont pas modifiées.
- ☐ **#BC11** : renvoie le mode actuel de résolution ;
 SORTIE : A contient numéro de mode ;
 Carry=1 si mode=0, Carry=0 si mode 1 ou 2 ;
 Z=1 si mode=1, Z=0 si mode 0 ou 2 ;
 Flags modifiés.

- ☐ **#BC20** : calcule l'adresse écran placée à droite de l'actuelle, compte tenu des scrollings décalant l'adresse de début ;
 ENTREE : HL contient l'adresse écran originale ;
 SORTIE : HL recalculé ;
 AF est modifié.

- ☐ **#BC23** : idem BC20, mais décale vers la gauche. Mêmes entrée et sortie que BC20.

- ☐ **#BC26** : idem, mais calcule l'adresse de l'octet en dessous.

- ☐ **#BC29** : idem pour l'octet au-dessus.

- ☐ **#BC32** : modification des couleurs d'un stylo ;
 ENTREES : A contient le numéro de stylo ;
 B contient le numéro de la première couleur ;
 C contient le numéro de la seconde couleur ;
 SORTIES : les registres AF,BC,DE,HL sont modifiés.

- ☐ **#BC38** : modification des couleurs du bord de l'écran ;
 ENTREES : comme BC32 sauf A, pas pris en compte ;
 SORTIE : idem BC32.

- ☐ **#BC5F** : HORLIN ; trace une ligne horizontale ; les coordonnées sont celles de points physiques et non logiques ; le masque d'encre peut être obtenu par appel de INKCOD (#BC2C).
 ENTREES : A masque de l'encre ;
 BC abscisse droite du segment ;
 DE abscisse gauche du segment ;
 HL ordonnée du segment ;
 SORTIE : AF,BC,DE et HL sont modifiés.

- ☐ **#BC62** : VERLIN ; idem HORLIN, mais trace une ligne verticale ;
 ENTREE : A masque de l'encre ;
 BC ordonnée haute du segment ;
 DE abscisse du segment ;
 HL ordonnée basse du segment ;
 SORTIES : AF,BC,DE et HL modifiés.

- ☐ **#BC2C** : INKCOD ; change un numéro de stylo en son masque ;
 ENTREE : A numéro de stylo ;
 SORTIE : A masque du stylo ;
 AF est modifié.

Annexe 3

Couleurs et masques

Chaque stylo peut être associé à une des 27 couleurs disponibles. Cette association est totalement indépendante du contenu de l'écran : celui-ci ne change pas, même si un stylo est modifié. Seul l'écran témoigne de la nouvelle couleur choisie, tous les points tracés avec ce stylo l'adoptant. Il suffit, après avoir réinitialisé l'Amstrad, d'effectuer une instruction "INK 0,0" pour le constater. Le fond de l'écran devient noir au lieu du bleu initial. C'est la seule modification réelle (le contenu de la mémoire écran ne change pas).

En RAM-écran, les points sont associés à un numéro de stylo. Le chapitre 1 explique comment les bits de chaque point sont entrelacés dans un octet, suivant le mode de résolution.

Les entrelacements, d'apparence si compliquée, ne le sont pas tant que cela. En effet, si l'on isole chaque point d'un octet, on constate que la répartition des bits est la même ; seule la position dans l'octet varie. Ainsi, pour le mode 1, si le point considéré est tracé en stylo 3 (binaire 11), on obtient le contenu d'octet 00010001 si le point est celui placé à droite ; 00100010, s'il est un peu plus à gauche, et ainsi de suite. Ces combinaisons élémentaires de bits, où un seul point est allumé, sont appelées des MASQUES. Ce nom leur vient de l'utilité qu'elles représentent. En effet, en prenant le contenu d'un octet "AND" avec ce masque, on récupère la

couleur du seul point correspondant. Supposons par exemple que nous devions connaître, en mode 1, quel est le stylo du point situé le plus à gauche.

```

11010101 (contenu de l'écran, quelconque a priori)
AND 10001000 (masque du point de gauche en mode 1)
10000000 (masque associé au stylo 1 en mode 1)

```

Cet exemple nous amène à dissocier deux types de masques. Les masques de points sont les masques cités ci-dessus : tous les bits du point sont positionnés à 1, les autres à zéro. Les masques de couleur répondent à la même logique : seuls les bits 1 du stylo sont positionnés à 1 dans le masque.

Le tableau suivant résume, pour chaque mode, les masques de points et de stylos en binaire, en hexadécimal et en décimal. Ils sont donnés pour le point de droite uniquement. Pour chaque décalage à gauche du point voulu, il suffit de multiplier par deux la valeur donnée. Par exemple, le masque du point de droite en mode 2 est 1. Pour avoir celui du point de gauche (sept points plus loin), il suffit de multiplier 7 fois par 2, ce qui donne 128.

Il faut également noter que le stylo 0 se caractérise par un masque égal à zéro, quels que soient le mode et le point concernés. D'autre part, le masque de point est exactement le même que celui du dernier stylo disponible.

MASQUE	MODE 0	MODE 1	MODE 2
point	bin 01010101 hex 55 dec 85	bin 00010001 hex 11 dec 17	bin 00000001 hex 01 dec 1
stylo 0 (fond)	bin 00000000 hex 00 dec 0	bin 00000000 hex 00 dec 0	bin 00000000 hex 00 dec 0
stylo 1	bin 01000000 hex 40 dec 64	bin 00010000 hex 10 dec 16	bin 00000001 hex 01 dec 1
stylo 2	bin 00000100 hex 04 dec 4	bin 00000001 hex 01 dec 1	
stylo 3	bin 01000100 hex 44 dec 68	bin 00010001 hex 11 dec 17	
stylo 4	bin 00010000 hex 10 dec 16		

stylo 5	bin 01010000 hex 50 dec 80		
stylo 6	bin 00010100 hex 14 dec 20		
stylo 7	bin 01010100 hex 54 dec 84		
stylo 8	bin 00000001 hex 01 dec 1		
stylo 9	bin 01000001 hex 41 dec 65		
stylo 10	bin 00000101 hex 05 dec 5		
stylo 11	bin 01000101 hex 45 dec 69		
stylo 12	bin 00010001 hex 11 dec 17		
stylo 13	bin 01010001 hex 51 dec 81		
stylo 14	bin 00010101 hex 15 dec 21		
stylo 15	bin 01010101 hex 55 dec 85		

Annexe 4

Carte mémoire Amstrad

Les Amstrad, quel que soit leur modèle, utilisent tous la même carte mémoire (en tout cas en ce qui concerne les modèles 464, 664 et 6128). Leurs données système, blocs de vecteurs et autres limites de la mémoire utilisateur sont les mêmes. Cela signifie que la capacité mémoire des trois modèles est identique, que l'écran se trouve au même emplacement, etc. Il s'agit déjà d'une grande garantie de compatibilité.

Il convient de noter que l'Amstrad possède une organisation en ROM supérieures qui peut également s'étendre à de la RAM (c'est ce qui a été fait sur le 6128). Ces ROM ou RAM parallèles sont des blocs de 16 Ko situés en #C000. Il y a deux contraintes à respecter pour les utiliser : on ne peut accéder qu'à un seul bloc à la fois (quoi qu'il arrive, le Z-80 ne peut adresser que 64 Ko, pas un de plus, ni 16). Et surtout, les blocs de données ou de contrôle situés dans les 48 Ko restants ne doivent en aucun cas être écrasés si l'on veut gérer ces ROM/RAM.

Un bon exemple de ces contraintes : l'addition d'un lecteur de disquettes à un CPC 464 provoque une diminution de la RAM disponible. En effet, les routines de gestion du lecteur sont situées dans une ROM parallèle, et un bloc de données supplémentaires destiné à sa gestion vient prendre place dans la RAM utilisateur, réduisant celle-ci d'autant. Sur 664 et 6128, cette opération est effectuée d'office, le lecteur étant intégré, ainsi que sa ROM de contrôle. On peut de nouveau constater le phénomène sur 6128 en

mettant en place le programme "Bank Manager" livré sur la disquette système, programme dont le rôle est d'assurer la gestion des 64 Ko supplémentaires. Les routines se placent elles aussi en RAM, et la zone utilisateur diminue donc (les 64 Ko parallèles ne sont pas programmables tels quels, mais on peut les utiliser comme un "lecteur de disquettes" supplémentaire).

L'Amstrad possède également une autre ROM parallèle, située de #0000 à #2FFF. Cette ROM "basse" contient le système d'exploitation et ses routines, et possède un bloc de données en RAM. Ce bloc est de loin le plus important et ne doit en aucun cas être perdu, sauf si, évidemment, l'application envisagée fournit ses propres routines systèmes.

Les extensions "RSX" brièvement évoquées dans l'ouvrage prennent place en RAM, ou du moins leurs blocs de contrôle.

CARTE DE LA MÉMOIRE VIVE 64 Ko

\$0000:	vecteurs de contrôle du système (interruptions, connexions des ROM ou RAM parallèles, etc.). NE DOIT EN AUCUN CAS ÊTRE PERDU !!!!!
\$003F	
\$0040:	variables systèmes de l'interpréteur Basic. Peut être écrasé si le programme ne nécessite ni retour au Basic ni utilisation d'une quelconque des routines Basic (mathématiques, par exemple).
\$016F	
\$0170:	DEBUT DE LA RAM UTILISATEUR. Peuvent également être placées ici des variables de contrôle de ROM supérieures. L'adresse \$0170 est alors repoussée par le programme d'initialisation de ces éventuelles ROM. Une application peut ainsi placer ces variables soit en début de zone libre, soit en fin, au choix.
\$A67B:	FIN DE LA RAM UTILISATEUR sur un appareil avec lecteur de disquettes. La zone de cette adresse à \$AB7F peut être utilisée sur un 464 sans lecteur. La connexion de ROM supplémentaires ou d'extensions RSX diminue cette adresse en fonction de leurs besoins.
\$AB7F	
\$AB80:	zone de variables systèmes Basic. Peut être écrasé si le programme ne nécessite ni retour au Basic ni utilisation d'une de ses routines.
\$B100	

\$B101:	zone de variables systèmes utilisées par le système d'exploitation. Peut être écrasé si le programme se passe de celui-ci.
\$B8FF	
\$B900	bloc de vecteurs systèmes. Permettent de sélectionner les ROM parallèles, la ROM basse, etc. ÉVITER D'Y TOUCHER.
\$BAFF	
\$BB00:	bloc des vecteurs. S'y trouvent tous les vecteurs des routines systèmes, mathématiques, etc. Ce bloc fournit ainsi un appel simple pour toute la gestion des ressources (graphismes, sons, interruptions...) et on évitera de le modifier. Chaque vecteur pointe sur une routine d'une ROM. Il est possible de modifier un vecteur afin qu'il pointe sur une autre routine. Il n'est pas non plus interdit d'écraser purement et simplement ce bloc. Dans ce cas, le programme devra soit fournir ses propres routines systèmes (gestion des graphismes, du son...), soit appeler directement les routines voulues en ROM grâce au bloc \$B900-\$BAFF.
\$BDFF	
\$BE00:	zone réservée à la pile.
\$BFFF	
\$C000:	RAM-Ecran. 384 octets de cette RAM ne sont pas visualisés et peuvent donc être récupérés si besoin est. Mais ils sont effacés au premier CLS et déplacés en cas de scrolling.
\$FFFF	

Annexe 5

Carte mémoire des routines

Les routines en assembleur du livre sont placées de façon à ne pas se chevaucher. Il est donc possible de les charger en mémoire simultanément. C'est de plus nécessaire pour certains programmes d'application.

La totalité des routines est disponible sous forme de DATA Basic dans un programme, au moins une fois dans le livre. Une fois le programme en question exécuté, il est donc possible de sauvegarder la routine de la façon suivante :

SAVE "nom de la routine",b,adresse départ,adresse fin-adresse départ+1.

Cette annexe résume donc les adresses de début et de fin de chaque routine du livre.

LISTE DES PROGRAMMES DU LIVRE

Chaque programme comporte une référence indiquant son type :

- PROGn.m : il s'agit du programme Basic "m" du chapitre "n".
- PROGn.mAS : il s'agit du programme assembleur "m" du chapitre "n".

- 1.1 : accès direct à la mémoire écran ;
- 1.2 : calcul d'un octet de la mémoire écran ;
- 1.3 : mouvement simulé par modification de couleurs ;
- 1.4 : remplissage de l'écran en Basic par points ;
- 1.5 : remplissage de l'écran en Basic par accès mémoire ;
- 1.6 : remplissage en LM par points ;
- 1.6as : routine du programme 1.6 ;
- 1.7 : remplissage en LM par accès mémoire ;
- 1.7as : routine du programme 1.7 ;

- 2.1 : addition en LM interfacée avec le Basic ;
- 2.1as : routine du programme 2.1 ;
- 2.2 : addition en LM chronométrée ;
- 2.3 : addition en Basic chronométrée ;
- 2.4 : soustraction en LM interfacée avec le Basic ;
- 2.4as : routine du programme 2.4 ;

- 3.1 : tracé de cercles optimisé en Basic par points ;
- 3.2 : tracé de cercles optimisé en LM par points ;
- 3.2as : routine du programme 3.2 ;
- 3.3 : tracé de cercles optimisé en LM par lignes ;
- 3.3as : routine du programme 3.3 ;
- 3.4 : tracé d'histogrammes en Basic ;
- 3.5 : tracé d'histogrammes en LM ;
- 3.5as : routine du programme 3.5 ;
- 3.6 : remplissage de zone quelconque en Basic ;
- 3.7 : remplissage de zone quelconque en LM ;
- 3.7as : routine du programme 3.7 ;

- image : création de l'image écran servant de décor aux routines ;
- 4.1as : restitution simple d'un objet à l'écran ;
- 4.2as : stockage simple d'une zone écran en mémoire ;
- 4.3 : codage en mémoire d'un module (utilise 4.2as) par dessin ;
- 4.3b : codage en mémoire d'un robot par POKE ;
- 4.3c : codage en mémoire d'une bête par POKE ;
- 4.4 : animation du module (utilise 4.1as) ;
- 4.4b : animation du robot (utilise 4.1as) (modifications de 4.4) ;
- 4.4c : animation de la bête (utilise 4.1as) (modifications de 4.4) ;

- 5.1 : compactage d'un objet graphique ;
- 5.1as : routine du programme 5.1 ;
- 5.2 : restitution écran d'un objet compacté ;
- 5.2as : routine du programme 5.2 ;

- 6.1as : déplacement par joystick d'un objet ;
- 6.2 : application du programme 6.1as ;
- 6.2b : idem 6.2 ;
- 6.2c : idem 6.2 ;

- 7.1 : restitution d'un objet en mode XOR ;
- 7.1b : idem 7.1 ;

- 7.1c : idem 7.1 ;
- 7.1as : routine du programme 7.1 ;
- 7.2as : restitution d'un objet sur décor ;
- 7.3 : application du programme 7.2as ;
- 7.3b : idem 7.3 ;
- 7.3c : idem 7.3 ;
- 7.4 : restitution d'un objet sur décor et sous avant-plans ;
- 7.4b : idem 7.4 ;
- 7.4c : idem 7.4 ;
- 7.4as : routine du programme 7.4 ;
- 8.1 : déplacement d'un objet par coordonnées et joystick ;
- 8.1b : idem 8.1 ;
- 8.1c : idem 8.1 ;
- 8.1as : routine du programme 8.1 ;
- 8.2as : détection de collision entre deux objets ;
- 8.3 : déplacement complet ;
- 8.3b : idem 8.3 ;
- 8.3c : idem 8.3 ;
- 8.3as : gestion des territoires interdits ;
- 8.4 : déplacement automatique du module (décor, avant-plans, terri-
toires) ;
- 8.4b : idem 8.4 ;
- 8.4c : idem 8.4 ;
- 8.4as : routine du programme 8.4.

EMPLACEMENT DES ROUTINES LM DU LIVRE

Si l'on excepte les routines des premiers chapitres, tous les programmes assembleur du livre sont compatibles entre eux et peuvent être chargés ensemble en mémoire. Les adresses ci-dessous permettent, une fois un programme d'application exécuté, de sauvegarder les routines dans un fichier binaire (grâce à l'instruction SAVE décrite plus haut).

Quelques-uns des programmes d'application de ce livre (notamment dans les derniers chapitres) nécessitent le chargement de routines par le biais de fichiers binaires. Par exemple, l'instruction suivante :

```
LOAD "prog4.1ob"
```

suppose que vous avez préalablement sauvegardé le programme 4.1 sous ce nom et à partir des adresses de début et de fin indiquées ci-dessous en hexadécimal.

Les routines ainsi réutilisées dans des programmes sont signalées ci-dessous par une étoile.

Routine numéro	Adresse début	Adresse fin
1.6	\$4000	\$4033
1.7	\$4000	\$400C
2.1	\$4000	\$4017
2.4	\$4000	\$4019
3.2	\$4000	\$411D
3.3	\$4000	\$4173
3.5	\$4200	\$446B
3.7	\$5000	\$5308
4.1 (*)	\$4700	\$471E
4.2 (*)	\$4730	\$4750
5.1	\$4500	\$4599
5.2	\$4600	\$4647
6.1 (*)	\$4C00	\$4C77
7.1	\$4800	\$481F
7.2	\$4830	\$4852
7.4 (*)	\$4760	\$47CA
8.1 (*)	\$489C	\$492F
8.2 (*)	\$4950	\$498C
8.3 (*)	\$49A0	\$49C8
8.4	\$4A00	\$4ACF

VARIABLES GLOBALES

Les routines des chapitres 4 à 8 utilisent une zone de variables commune. Ces variables sont résumées ci-dessous avec leur nom courant. Certaines possèdent deux noms, suivant la routine. Toutes les adresses ci-dessous signalées par "—" sont utilisées comme poids fort de la variable au-dessus, celle-ci étant alors au format 16 bits.

Adresse hexa.	Variable
459A	: BUFAC
459B	—
459C	: BUF / BUFFER
459D	—
459E	: LIGNE
459F	—
45A0	: ECRAN
45A1	—
45A2	: COUNT
45A3	: REPEAT

45A4	: OCTREF
45A5	: LECTUR
45A6	: LAR
45A7	: HAU
45A8	: X
45A9	: Y
45AA	: LAR / X
45AB	: HAU / Y
45AC	: X1
45AD	: Y1
45AE	: LAR1
45AF	: HAU1
45B0	: COLLI
45B1	: TABLE
45B2	—
45B3	: XO
45B4	: YO
45B5	: ECRAN0
45B6	—
45B7	: ECRAN2
45B8	—
45B9	: BUFFER
45BA	—
45BB	: DESSIN
45BC	—
45BD	: JOYST

Annexe 6

Structure écran de l'Amstrad

L'écran est composé de 200 lignes. Chaque ligne est codée avec 80 octets successifs, quel que soit le mode. L'adresse de départ de l'écran est \$C000 (sauf après un scrolling de texte, que nous supposons nul). Il s'y trouve le 1^{er} octet (gauche) de la première ligne (haut). Ensuite, les adresses des premiers octets de chaque ligne sont placés aux adresses indiquées ci-dessous, exprimées en hexadécimal. Les deux numéros placés à gauche des adresses permettent de trouver l'adresse d'une ligne, que le haut de l'écran soit noté 1 ou 200.

Ligne	-	adresse
200	1	C000
199	2	C800
198	3	D000
197	4	D800
196	5	E000
195	6	E800
194	7	F000
193	8	F800
192	9	C050
191	10	C850
190	11	D050
189	12	D850

188	13	E050
187	14	E850
186	15	F050
185	16	F850
184	17	C0A0
183	18	C8A0
182	19	D0A0
181	20	D8A0
180	21	E0A0
179	22	E8A0
178	23	F0A0
177	24	F8A0
176	25	C0F0
175	26	C8F0
174	27	D0F0
173	28	D8F0
172	29	E0F0
171	30	E8F0
170	31	F0F0
169	32	F8F0
168	33	C140
167	34	C940
166	35	D140
165	36	D940
164	37	E140
163	38	E940
162	39	F140
161	40	F940
160	41	C190
159	42	C990
158	43	D190
157	44	D990
156	45	E190
155	46	E990
154	47	F190
153	48	F990
152	49	C1E0
151	50	C9E0
150	51	D1E0
149	52	D9E0
148	53	E1E0
147	54	E9E0
146	55	F1E0
145	56	F9E0
144	57	C230
143	58	CA30
142	59	D230
141	60	DA30

140	61	E230
139	62	EA30
138	63	F230
137	64	FA30
136	65	C280
135	66	CA80
134	67	D280
133	68	DA80
132	69	E280
131	70	EA80
130	71	F280
129	72	FA80
128	73	C2D0
127	74	CAD0
126	75	D2D0
125	76	DAD0
124	77	E2D0
123	78	EAD0
122	79	F2D0
121	80	FAD0
120	81	C320
119	82	CB20
118	83	D320
117	84	DB20
116	85	E320
115	86	EB20
114	87	F320
113	88	FB20
112	89	C370
111	90	CB70
110	91	D370
109	92	DB70
108	93	E370
107	94	EB70
106	95	F370
105	96	FB70
104	97	C3C0
103	98	CBC0
102	99	D3C0
101	100	DBC0
100	101	E3C0
99	102	EBC0
98	103	F3C0
97	104	FBC0
96	105	C410
95	106	CC10
94	107	D410
93	108	DC10

92	109	E410
91	110	EC10
90	111	F410
89	112	FC10
88	113	C460
87	114	CC60
86	115	D460
85	116	DC60
84	117	E460
83	118	EC60
82	119	F460
81	120	FC60
80	121	C4B0
79	122	CCB0
78	123	D4B0
77	124	DCB0
76	125	E4B0
75	126	ECB0
74	127	F4B0
73	128	FCB0
72	129	C500
71	130	CD00
70	131	D500
69	132	DD00
68	133	E500
67	134	ED00
66	135	F500
65	136	FD00
64	137	C550
63	138	CD50
62	139	D550
61	140	DD50
60	141	E550
59	142	ED50
58	143	F550
57	144	FD50
56	145	C5A0
55	146	CDA0
54	147	D5A0
53	148	DDA0
52	149	E5A0
51	150	EDA0
50	151	F5A0
49	152	FDA0
48	153	C5F0
47	154	CDF0
46	155	D5F0
45	156	DDF0
44	157	E5F0

43	158	EDF0
42	159	F5F0
41	160	FDF0
40	161	C640
39	162	CE40
38	163	D640
37	164	DE40
36	165	E640
35	166	EE40
34	167	F640
33	168	FE40
32	169	C690
31	170	CE90
30	171	D690
29	172	DE90
28	173	E690
27	174	EE90
26	175	F690
25	176	FE90
24	177	C6E0
23	178	CEE0
22	179	D6E0
21	180	DEE0
20	181	E6E0
19	182	EEE0
18	183	F6E0
17	184	FEE0
16	185	C730
15	186	CF30
14	187	D730
13	188	DF30
12	189	E730
11	190	EF30
10	191	F730
9	192	FF30
8	193	C780
7	194	CF80
6	195	D780
5	196	DF80
4	197	E780
3	198	EF80
2	199	F780
1	200	FF80

Annexe 7

Le jeu d'instruction du Z-80

Pour la petite histoire, voici comment cette liste a été créée.

- a. Un programme BASIC, partant de la racine de chaque mnémonique, va produire la famille correspondante.

Exemple : LD <SR>,<SR>

(SR représente les simples registres), va créer tous les codes : LD A,A, LD A,B ... LD L,L.

Le fichier ainsi constitué est placé sur disque.

- b. Tri du fichier par ordre croissant.
- c. Numérotation des instructions (programme BASIC).
- d. Assemblage du fichier pour générer le code machine (listing sur fichier disque).
- e. Suppressions des numéros de lignes (devenus inutiles) par programme BASIC.
- f. Altération du code machine (dd, nn, etc.) par un logiciel de traitement de texte.
- g. Disposition du texte en deux colonnes de 60 lignes par page (programme BASIC).
- h. Édition de la liste (voir ci-après).

Les symboles employés ont la signification suivante :

dd	: valeur de déplacement signée sur 8 bits,
N	: valeur de 8 bits de l'opérande,
nn	: valeur de 8 bits dans le code machine,
ADR	: label d'adresse en opérande,
aaaa	: valeur d'adresse sur 16 bits dans le code machine (poids faibles + poids forts),
DEPL	: déplacement dans l'opérande des instructions en mode relatif,
NN	: valeur de 16 bits ou adresse dans l'opérande,
nnnn	: valeur de 16 bits dans le code machine (poids faibles + poids forts).

0000	8E	ADC A,(HL)	*	0066	DDCBdd4E	BIT 1,(IX+dd)
0001	DD8Edd	ADC A,(IX+dd)	*	006A	FDCBdd4E	BIT 1,(IY+dd)
0004	FD8Edd	ADC A,(IY+dd)	*	006E	CB4F	BIT 1,A
0007	8F	ADC A,A	*	0070	CB48	BIT 1,B
0008	88	ADC A,B	*	0072	CB49	BIT 1,C
0009	89	ADC A,C	*	0074	CB4A	BIT 1,D
000A	8A	ADC A,D	*	0076	CB4B	BIT 1,E
000B	8B	ADC A,E	*	0078	CB4C	BIT 1,H
000C	8C	ADC A,H	*	007A	CB4D	BIT 1,L
000D	8D	ADC A,L	*	007C	CB56	BIT 2,(HL)
000E	CEnn	ADC A,N	*	007E	DDCBdd56	BIT 2,(IX+dd)
0010	ED4A	ADC HL,BC	*	0082	FDCBdd56	BIT 2,(IY+dd)
0012	ED5A	ADC HL,DE	*	0086	CB57	BIT 2,A
0014	ED6A	ADC HL,HL	*	0088	CB50	BIT 2,B
0016	ED7A	ADC HL,SP	*	008A	CB51	BIT 2,C
0018	86	ADD A,(HL)	*	008C	CB52	BIT 2,D
0019	DD86dd	ADD A,(IX+dd)	*	008E	CB53	BIT 2,E
001C	FD86dd	ADD A,(IY+dd)	*	0090	CB54	BIT 2,H
001F	87	ADD A,A	*	0092	CB55	BIT 2,L
0020	80	ADD A,B	*	0094	CB5E	BIT 3,(HL)
0021	81	ADD A,C	*	0096	DDCBdd5E	BIT 3,(IX+dd)
0022	82	ADD A,D	*	009A	FDCBdd5E	BIT 3,(IY+dd)
0023	83	ADD A,E	*	009E	CB5F	BIT 3,A
0024	84	ADD A,H	*	00A0	CB58	BIT 3,B
0025	85	ADD A,L	*	00A2	CB59	BIT 3,C
0026	C6nn	ADD A,N	*	00A4	CB5A	BIT 3,D
0028	09	ADD HL,BC	*	00A6	CB5B	BIT 3,E
0029	19	ADD HL,DE	*	00A8	CB5C	BIT 3,H
002A	29	ADD HL,HL	*	00AA	CB5D	BIT 3,L
002B	39	ADD HL,SP	*	00AC	CB66	BIT 4,(HL)
002C	DD09	ADD IX,BC	*	00AE	DDCBdd66	BIT 4,(IX+dd)
002E	DD19	ADD IX,DE	*	00B2	FDCBdd66	BIT 4,(IY+dd)
0030	DD29	ADD IX,IX	*	00B6	CB67	BIT 4,A
0032	DD39	ADD IX,SP	*	00B8	CB60	BIT 4,B
0034	FD09	ADD IY,BC	*	00BA	CB61	BIT 4,C
0036	FD19	ADD IY,DE	*	00BC	CB62	BIT 4,D
0038	FD29	ADD IY,IY	*	00BE	CB63	BIT 4,E
003A	FD39	ADD IY,SP	*	00C0	CB64	BIT 4,H
003C	A6	AND (HL)	*	00C2	CB65	BIT 4,L
003D	DDA6dd	AND (IX+dd)	*	00C4	CB6E	BIT 5,(HL)
0040	FDA6dd	AND (IY+dd)	*	00C6	DDCBdd6E	BIT 5,(IX+dd)
0043	A7	AND A	*	00CA	FDCBdd6E	BIT 5,(IY+dd)

0044 A0	AND B	*	00CE CB6F	BIT 5,A
0045 A1	AND C	*	00D0 CB68	BIT 5,B
0046 A2	AND D	*	00D2 CB69	BIT 5,C
0047 A3	AND E	*	00D4 CB6A	BIT 5,D
0048 A4	AND H	*	00D6 CB6B	BIT 5,E
0049 A5	AND L	*	00D8 CB6C	BIT 5,H
004A E6nn	AND N	*	00DA CB6D	BIT 5,L
004C CB46	BIT 0,(HL)	*	00DC CB76	BIT 6,(HL)
004E DDCBdd46	BIT 0,(IX+dd)	*	00DE DDCBdd76	BIT 6,(IX+dd)
0052 FDCBdd46	BIT 0,(IY+dd)	*	00E2 FDCBdd76	BIT 6,(IY+dd)
0056 CB47	BIT 0,A	*	00E6 CB77	BIT 6,A
0058 CB40	BIT 0,B	*	00E8 CB70	BIT 6,B
005A CB41	BIT 0,C	*	00EA CB71	BIT 6,C
005C CB42	BIT 0,D	*	00EC CB72	BIT 6,D
005E CB43	BIT 0,E	*	00EE CB73	BIT 6,E
0060 CB44	BIT 0,H	*	00F0 CB74	BIT 6,H
0062 CB45	BIT 0,L	*	00F2 CB75	BIT 6,L
0064 CB4E	BIT 1,(HL)	*	00F4 CB7E	BIT 7,(HL)
00F6 DDCBdd7E	BIT 7,(IX+dd)	*	0163 D9	EXX
00FA FDCBdd7E	BIT 7,(IY+dd)	*	0164 76	HALT
00FE CB7F	BIT 7,A	*	0165 ED46	IM 0
0100 CB78	BIT 7,B	*	0167 ED56	IM 1
0102 CB79	BIT 7,C	*	0169 ED5E	IM 2
0104 CB7A	BIT 7,D	*	016B ED78	IN A,(C)
0106 CB7B	BIT 7,E	*	016D DBnn	IN A,(N)
0108 CB7C	BIT 7,H	*	016F ED40	IN B,(C)
010A CB7D	BIT 7,L	*	0171 ED48	IN C,(C)
010C DCaaaa	CALL C,ADR	*	0173 ED50	IN D,(C)
010F FCaaaa	CALL M,ADR	*	0175 ED58	IN E,(C)
0112 D4aaaa	CALL NC,ADR	*	0177 ED60	IN H,(C)
0115 CDaaaa	CALL ADR	*	0179 ED68	IN L,(C)
0118 C4aaaa	CALL NZ,ADR	*	017B 34	INC (HL)
011B F4aaaa	CALL P,ADR	*	017C DD34dd	INC (IX+dd)
011E ECaaaa	CALL PE,ADR	*	017F FD34dd	INC (IY+dd)
0121 E4aaaa	CALL PO,ADR	*	0182 3C	INC A
0124 CCaaaa	CALL Z,ADR	*	0183 04	INC B
0127 3F	CCF	*	0184 03	INC BC
0128 BE	CP (HL)	*	0185 0C	INC C
0129 DDBEdD	CP (IX+dd)	*	0186 14	INC D
012C FDBEdD	CP (IY+dd)	*	0187 13	INC DE
012F BF	CP A	*	0188 1C	INC E
0130 B8	CP B	*	0189 24	INC H
0131 B9	CP C	*	018A 23	INC HL
0132 BA	CP D	*	018B DD23	INC IX
0133 BB	CP E	*	018D FD23	INC IY
0134 BC	CP H	*	018F 2C	INC L
0135 BD	CP L	*	0190 33	INC SP
0136 FEnn	CP N	*	0191 EDAA	IND
0138 EDA9	CPD	*	0193 ED8A	INDR
013A EDB9	CPDR	*	0195 EDA2	INI
013C EDA1	CPI	*	0197 EDB2	INIR
013E EDB1	CPIR	*	0199 E9	JP (HL)
0140 2F	CPL	*	019A DDE9	JP (IX)
0141 27	DAA	*	019C FDE9	JP (IY)
0142 35	DEC (HL)	*	019E DAaaaa	JP C,ADR
0143 DD35dd	DEC (IX+dd)	*	01A1 FAaaaa	JP M,ADR
0146 FD35dd	DEC (IY+dd)	*	01A4 D2aaaa	JP NC,ADR
0149 3D	DEC A	*	01A7 C3aaaa	JP ADR
014A 05	DEC B	*	01AA C2aaaa	JP NZ,ADR

014B 0B	DEC BC	*	01AD F2aaaa	JP P,ADR
014C 0D	DEC C	*	01B0 EAaaaa	JP PE,ADR
014D 15	DEC D	*	01B3 E2aaaa	JP PO,ADR
014E 1B	DEC DE	*	01B6 CAaaaa	JP Z,ADR
014F 1D	DEC E	*	01B9 38dd	JR C,DEPL
0150 25	DEC H	*	01BB 18dd	JR DEPL
0151 2B	DEC HL	*	01BD 30dd	JR NC,DEPL
0152 DD2B	DEC IX	*	01BF 20dd	JR NZ,DEPL
0154 FD2B	DEC IY	*	01C1 28dd	JR Z,DEPL
0156 2D	DEC L	*	01C3 02	LD (BC),A
0157 3B	DEC SP	*	01C4 12	LD (DE),A
0158 F3	DI	*	01C5 77	LD (HL),A
0159 10dd	DJNZ DEPL	*	01C6 70	LD (HL),B
015B FB	EI	*	01C7 71	LD (HL),C
015C E3	EX (SP),HL	*	01C8 72	LD (HL),D
015D DDE3	EX (SP),IX	*	01C9 73	LD (HL),E
015F FDE3	EX (SP),IY	*	01CA 74	LD (HL),H
0161 08	EX AF,AF'	*	01CB 75	LD (HL),L
0162 EB	EX DE,HL	*	01CC 36nn	LD (HL),N
01CE DD77dd	LD (IX+dd),A	*	0256 4C	LD C,H
01D1 DD70dd	LD (IX+dd),B	*	0257 4D	LD C,L
01D4 DD71dd	LD (IX+dd),C	*	0258 0Enn	LD C,N
01D7 DD72dd	LD (IX+dd),D	*	025A 56	LD D,(HL)
01DA DD73dd	LD (IX+dd),E	*	025B DD56dd	LD D,(IX+dd)
01DD DD74dd	LD (IX+dd),H	*	025E FD56dd	LD D,(IY+dd)
01E0 DD75dd	LD (IX+dd),L	*	0261 57	LD D,A
01E3 DD36ddnn	LD (IX+dd),N	*	0262 50	LD D,B
01E7 FD77dd	LD (IY+dd),A	*	0263 51	LD D,C
01EA FD70dd	LD (IY+dd),B	*	0264 52	LD D,D
01ED FD71dd	LD (IY+dd),C	*	0265 53	LD D,E
01F0 FD72dd	LD (IY+dd),D	*	0266 54	LD D,H
01F3 FD73dd	LD (IY+dd),E	*	0267 55	LD D,L
01F6 FD74dd	LD (IY+dd),H	*	0268 16nn	LD D,N
01F9 FD75dd	LD (IY+dd),L	*	026A ED5Bnnnn	LD DE,(NN)
01FC FD36ddnn	LD (IY+dd),N	*	026E 11nnnn	LD DE,NN
0200 32nnnn	LD (NN),A	*	0271 5E	LD E,(HL)
0203 ED43nnnn	LD (NN),BC	*	0272 DD5Edd	LD E,(IX+dd)
0207 ED53nnnn	LD (NN),DE	*	0275 FD5Edd	LD E,(IY+dd)
020B 22nnnn	LD (NN),HL	*	0278 5F	LD E,A
020E DD22nnnn	LD (NN),IX	*	0279 58	LD E,B
0212 FD22nnnn	LD (NN),IY	*	027A 59	LD E,C
0216 ED73nnnn	LD (NN),SP	*	027B 5A	LD E,D
021A 0A	LD A,(BC)	*	027C 5B	LD E,E
021B 1A	LD A,(DE)	*	027D 5C	LD E,H
021C 7E	LD A,(HL)	*	027E 5D	LD E,L
021D DD7Edd	LD A,(IX+dd)	*	027F 1Enn	LD E,N
0220 FD7Edd	LD A,(IY+dd)	*	0281 66	LD H,(HL)
0223 3Annnn	LD A,(NN)	*	0282 DD66dd	LD H,(IX+dd)
0226 7F	LD A,A	*	0285 FD66dd	LD H,(IY+dd)
0227 78	LD A,B	*	0288 67	LD H,A
0228 79	LD A,C	*	0289 60	LD H,B
0229 7A	LD A,D	*	028A 61	LD H,C
022A 7B	LD A,E	*	028B 62	LD H,D
022B 7C	LD A,H	*	028C 63	LD H,E
022C ED57	LD A,I	*	028D 64	LD H,H
022E 7D	LD A,L	*	028E 65	LD H,L
022F 3Enn	LD A,N	*	028F 26nn	LD H,N
0231 ED5F	LD A,R	*	0291 2Annnn	LD HL,(NN)
0233 46	LD B,(HL)	*	0294 21nnnn	LD HL,NN

0234	DD46dd	LD B,(IX+dd)	*	0297	ED47	LD I,A
0237	FD46dd	LD B,(IY+dd)	*	0299	DD2Annnn	LD IX,(NN)
023A	47	LD B,A	*	029D	DD21nnnn	LD IX,NN
023B	40	LD B,B	*	02A1	FD2Annnn	LD IY,(NN)
023C	41	LD B,C	*	02A5	FD21nnnn	LD IY,NN
023D	42	LD B,D	*	02A9	6E	LD L,(HL)
023E	43	LD B,E	*	02AA	DD6Edd	LD L,(IX+dd)
023F	44	LD B,H	*	02AD	FD6Edd	LD L,(IY+dd)
0240	45	LD B,L	*	02B0	6F	LD L,A
0241	06nn	LD B,N	*	02B1	68	LD L,B
0243	ED4Bnnnn	LD BC,(NN)	*	02B2	69	LD L,C
0247	01nnnn	LD BC,NN	*	02B3	6A	LD L,D
024A	4E	LD C,(HL)	*	02B4	6B	LD L,E
024B	DD4Edd	LD C,(IX+dd)	*	02B5	6C	LD L,H
024E	FD4Edd	LD C,(IY+dd)	*	02B6	6D	LD L,L
0251	4F	LD C,A	*	02B7	2Enn	LD L,N
0252	48	LD C,B	*	02B9	ED4F	LD R,A
0253	49	LD C,C	*	02BB	ED7Bnnnn	LD SP,(NN)
0254	4A	LD C,D	*	02BF	F9	LD SP,HL
0255	4B	LD C,E	*	02C0	DDF9	LD SP,IX
02C2	FDF9	LD SP,IY	*	0334	CB8B	RES 1,E
02C4	31nnnn	LD SP,NN	*	0336	CB8C	RES 1,H
02C7	EDA8	LDD	*	0338	CB8D	RES 1,L
02C9	EDB8	LDDR	*	033A	CB96	RES 2,(HL)
02CB	EDA0	LDI	*	033C	DDCBdd96	RES 2,(IX+dd)
02CD	EDB0	LDIR	*	0340	FDCBdd96	RES 2,(IY+dd)
02CF	ED44	NEG	*	0344	CB97	RES 2,A
02D1	00	NOP	*	0346	CB90	RES 2,B
02D2	B6	OR (HL)	*	0348	CB91	RES 2,C
02D3	DDB6dd	OR (IX+dd)	*	034A	CB92	RES 2,D
02D6	FDB6dd	OR (IY+dd)	*	034C	CB93	RES 2,E
02D9	B7	OR A	*	034E	CB94	RES 2,H
02DA	B0	OR B	*	0350	CB95	RES 2,L
02DB	B1	OR C	*	0352	CB9E	RES 3,(HL)
02DC	B2	OR D	*	0354	DDCBdd9E	RES 3,(IX+dd)
02DD	B3	OR E	*	0358	FDCBdd9E	RES 3,(IY+dd)
02DE	B4	OR H	*	035C	CB9F	RES 3,A
02DF	B5	OR L	*	035E	CB98	RES 3,B
02E0	F6nn	OR N	*	0360	CB99	RES 3,C
02E2	EDB8	OTDR	*	0362	CB9A	RES 3,D
02E4	EDB3	OTIR	*	0364	CB9B	RES 3,E
02E6	ED79	OUT (C),A	*	0366	CB9C	RES 3,H
02E8	ED41	OUT (C),B	*	0368	CB9D	RES 3,L
02EA	ED49	OUT (C),C	*	036A	CBA6	RES 4,(HL)
02EC	ED51	OUT (C),D	*	036C	DDCBddA6	RES 4,(IX+dd)
02EE	ED59	OUT (C),E	*	0370	FDCBddA6	RES 4,(IY+dd)
02F0	ED61	OUT (C),H	*	0374	CBA7	RES 4,A
02F2	ED69	OUT (C),L	*	0376	CBA0	RES 4,B
02F4	D3nn	OUT (N),A	*	0378	CBA1	RES 4,C
02F6	EDAB	OUTD	*	037A	CBA2	RES 4,D
02F8	EDA3	OUTI	*	037C	CBA3	RES 4,E
02FA	F1	POP AF	*	037E	CBA4	RES 4,H
02FB	C1	POP BC	*	0380	CBA5	RES 4,L
02FC	D1	POP DE	*	0382	CBAE	RES 5,(HL)
02FD	E1	POP HL	*	0384	DDCBddAE	RES 5,(IX+dd)
02FE	DDE1	POP IX	*	0388	FDCBddAE	RES 5,(IY+dd)
0300	FDE1	POP IY	*	038C	CBAF	RES 5,A
0302	F5	PUSH AF	*	038E	CBA8	RES 5,B
0303	C5	PUSH BC	*	0390	CBA9	RES 5,C

0304	D5	PUSH DE	*	0392	CBAA	RES 5,D
0305	E5	PUSH HL	*	0394	CBAB	RES 5,E
0306	DDE5	PUSH IX	*	0396	CBAC	RES 5,H
0308	FDE5	PUSH IY	*	0398	CBAD	RES 5,L
030A	CB86	RES 0,(HL)	*	039A	CBB6	RES 6,(HL)
030C	DDCBdd86	RES 0,(IX+dd)	*	039C	DDCBddB6	RES 6,(IX+dd)
0310	FDCBdd86	RES 0,(IY+dd)	*	03A0	FDCBddB6	RES 6,(IY+dd)
0314	CB87	RES 0,A	*	03A4	CBB7	RES 6,A
0316	CB80	RES 0,B	*	03A6	CBB0	RES 6,B
0318	CB81	RES 0,C	*	03A8	CBB1	RES 6,C
031A	CB82	RES 0,D	*	03AA	CBB2	RES 6,D
031C	CB83	RES 0,E	*	03AC	CBB3	RES 6,E
031E	CB84	RES 0,H	*	03AE	CBB4	RES 6,H
0320	CB85	RES 0,L	*	03B0	CBB5	RES 6,L
0322	CB8E	RES 1,(HL)	*	03B2	CBBE	RES 7,(HL)
0324	DDCBdd8E	RES 1,(IX+dd)	*	03B4	DDCBddBE	RES 7,(IX+dd)
0328	FDCBdd8E	RES 1,(IY+dd)	*	03B8	FDCBddBE	RES 7,(IY+dd)
032C	CB8F	RES 1,A	*	03BC	CBBF	RES 7,A
032E	CB88	RES 1,B	*	03BE	CBB8	RES 7,B
0330	CB89	RES 1,C	*	03C0	CBB9	RES 7,C
0332	CB8A	RES 1,D	*	03C2	CBBA	RES 7,D
03C4	CBBB	RES 7,E	*	043F	C7	RST 00
03C6	CBBC	RES 7,H	*	0440	CF	RST 08
03C8	CBBD	RES 7,L	*	0441	D7	RST 10H
03CA	C9	RET	*	0442	DF	RST 18H
03CB	D8	RET C	*	0443	E7	RST 20H
03CC	F8	RET M	*	0444	EF	RST 28H
03CD	D0	RET NC	*	0445	F7	RST 30H
03CE	C0	RET NZ	*	0446	FF	RST 38H
03CF	F0	RET P	*	0447	9E	SBC A,(HL)
03D0	E8	RET PE	*	0448	DD9Edd	SBC A,(IX+dd)
03D1	E0	RET P0	*	044B	FD9Edd	SBC A,(IY+dd)
03D2	C8	RET Z	*	044E	9F	SBC A,A
03D3	ED4D	RETI	*	044F	98	SBC A,B
03D5	ED45	RETN	*	0450	99	SBC A,C
03D7	CB16	RL (HL)	*	0451	9A	SBC A,D
03D9	DDCBdd16	RL (IX+dd)	*	0452	9B	SBC A,E
03DD	FDCBdd16	RL (IY+dd)	*	0453	9C	SBC A,H
03E1	CB17	RL A	*	0454	9D	SBC A,L
03E3	CB10	RL B	*	0455	DEnn	SBC A,N
03E5	CB11	RL C	*	0457	ED42	SBC HL,BC
03E7	CB12	RL D	*	0459	ED52	SBC HL,DE
03E9	CB13	RL E	*	045B	ED62	SBC HL,HL
03EB	CB14	RL H	*	045D	ED72	SBC HL,SP
03ED	CB15	RL L	*	045F	37	SCF
03EF	17	RLA	*	0460	CBC6	SET 0,(HL)
03F0	CB06	RLC (HL)	*	0462	DDCBddC6	SET 0,(IX+dd)
03F2	DDCBdd06	RLC (IX+dd)	*	0466	FDCBddC6	SET 0,(IY+dd)
03F6	FDCBdd06	RLC (IY+dd)	*	046A	CBC7	SET 0,A
03FA	CB07	RLC A	*	046C	CBC0	SET 0,B
03FC	CB00	RLC B	*	046E	CBC1	SET 0,C
03FE	CB01	RLC C	*	0470	CBC2	SET 0,D
0400	CB02	RLC D	*	0472	CBC3	SET 0,E
0402	CB03	RLC E	*	0474	CBC4	SET 0,H
0404	CB04	RLC H	*	0476	CBC5	SET 0,L
0406	CB05	RLC L	*	0478	CBCE	SET 1,(HL)
0408	D7	RLCA	*	047A	DDCBddCE	SET 1,(IX+dd)
0409	ED6F	RLD	*	047E	FDCBddCE	SET 1,(IY+dd)
040B	CB1E	RR (HL)	*	0482	CBCF	SET 1,A

040D	DDCBdd1E	RR (IX+dd)	*	0484	CBC8	SET 1,B
0411	FDCBdd1E	RR (IY+dd)	*	0486	CBC9	SET 1,C
0415	CB1F	RR A	*	0488	CBCA	SET 1,D
0417	CB18	RR B	*	048A	CBCB	SET 1,E
0419	CB19	RR C	*	048C	CBCC	SET 1,H
041B	CB1A	RR D	*	048E	CBCD	SET 1,L
041D	CB1B	RR E	*	0490	CBD6	SET 2,(HL)
041F	CB1C	RR H	*	0492	DDCBddD6	SET 2,(IX+dd)
0421	CB1D	RR L	*	0496	FDCBddD6	SET 2,(IY+dd)
0423	1F	RRA	*	049A	CBD7	SET 2,A
0424	CBOE	RRC (HL)	*	049C	CBDO	SET 2,B
0426	DDCBddOE	RRC (IX+dd)	*	049E	CBD1	SET 2,C
042A	FDCBddOE	RRC (IY+dd)	*	04A0	CBD2	SET 2,D
042E	CBOF	RRC A	*	04A2	CBD3	SET 2,E
0430	CB08	RRC B	*	04A4	CBD4	SET 2,H
0432	CB09	RRC C	*	04A6	CBD5	SET 2,L
0434	CB0A	RRC D	*	04A8	CBDE	SET 3,(HL)
0436	CB0B	RRC E	*	04AA	DDCBddDE	SET 3,(IX+dd)
0438	CB0C	RRC H	*	04AE	FDCBddDE	SET 3,(IY+dd)
043A	CB0D	RRC L	*	04B2	CBDF	SET 3,A
043C	OF	RRCA	*	04B4	CBD8	SET 3,B
043D	ED67	RRD	*	04B6	CBD9	SET 3,C
04B8	CBDA	SET 3,D	*	052C	CB20	SLA B
04BA	CBDB	SET 3,E	*	052E	CB21	SLA C
04BC	CBDC	SET 3,H	*	0530	CB22	SLA D
04BE	CBDD	SET 3,L	*	0532	CB23	SLA E
04C0	CBE6	SET 4,(HL)	*	0534	CB24	SLA H
04C2	DDCBddE6	SET 4,(IX+dd)	*	0536	CB25	SLA L
04C6	FDCBddE6	SET 4,(IY+dd)	*	0538	CB2E	SRA (HL)
04CA	CBE7	SET 4,A	*	053A	DDCBdd2E	SRA (IX+dd)
04CC	CBE0	SET 4,B	*	053E	FDCBdd2E	SRA (IY+dd)
04CE	CBE1	SET 4,C	*	0542	CB2F	SRA A
04D0	CBE2	SET 4,D	*	0544	CB28	SRA B
04D2	CBE3	SET 4,E	*	0546	CB29	SRA C
04D4	CBE4	SET 4,H	*	0548	CB2A	SRA D
04D6	CBE5	SET 4,L	*	054A	CB2B	SRA E
04D8	CBEE	SET 5,(HL)	*	054C	CB2C	SRA H
04DA	DDCBddEE	SET 5,(IX+dd)	*	054E	CB2D	SRA L
04DE	FDCBddEE	SET 5,(IY+dd)	*	0550	CB3E	SRL (HL)
04E2	CBEF	SET 5,A	*	0552	DDCBdd3E	SRL (IX+dd)
04E4	CBE8	SET 5,B	*	0556	FDCBdd3E	SRL (IY+dd)
04E6	CBE9	SET 5,C	*	055A	CB3F	SRL A
04E8	CBEA	SET 5,D	*	055C	CB38	SRL B
04EA	CBEB	SET 5,E	*	055E	CB39	SRL C
04EC	CBEC	SET 5,H	*	0560	CB3A	SRL D
04EE	CBED	SET 5,L	*	0562	CB3B	SRL E
04F0	CBF6	SET 6,(HL)	*	0564	CB3C	SRL H
04F2	DDCBddF6	SET 6,(IX+dd)	*	0566	CB3D	SRL L
04F6	FDCBddF6	SET 6,(IY+dd)	*	0568	96	SUB (HL)
04FA	CBF7	SET 6,A	*	0569	DD96dd	SUB (IX+dd)
04FC	CBF0	SET 6,B	*	056C	FD96dd	SUB (IY+dd)
04FE	CBF1	SET 6,C	*	056F	97	SUB A
0500	CBF2	SET 6,D	*	0570	90	SUB B
0502	CBF3	SET 6,E	*	0571	91	SUB C
0504	CBF4	SET 6,H	*	0572	92	SUB D
0506	CBF5	SET 6,L	*	0573	93	SUB E
0508	CBFE	SET 7,(HL)	*	0574	94	SUB H
050A	DDCBddFE	SET 7,(IX+dd)	*	0575	95	SUB L
050E	FDCBddFE	SET 7,(IY+dd)	*	0576	D6nn	SUB N

0512	CBFF	SET 7,A	*	0578	AE	XOR (HL)
0514	CBF8	SET 7,B	*	0579	DDAEdd	XOR (IX+dd)
0516	CBF9	SET 7,C	*	057C	FDAEdd	XOR (IY+dd)
0518	CBFA	SET 7,D	*	057F	AF	XOR A
051A	CBFB	SET 7,E	*	0580	A8	XOR B
051C	CBFC	SET 7,H	*	0581	A9	XOR C
051E	CBFD	SET 7,L	*	0582	AA	XOR D
0520	CB26	SLA (HL)	*	0583	AB	XOR E
0522	DDCBdd26	SLA (IX+dd)	*	0584	AC	XOR H
0526	FDCBdd26	SLA (IY+dd)	*	0585	AD	XOR L
052A	CB27	SLA A	*	0586	EEnn	XOR N

Lexique et index

☐ **Adressage :**

accès à une information. Il existe des modes d'adressage : ceux-ci correspondent à la façon dont on y accède. Par exemple, on parle d'adressage indirect lorsqu'au lieu d'utiliser explicitement une donnée, on passe par un intermédiaire la contenant. Cet intermédiaire peut être soit une case mémoire soit un registre.

☐ **Adresse :**

nombre de 16 bits (sur le Z-80), ou plus, permettant de repérer une case mémoire. Chaque case mémoire est en effet numérotée. La première est 0, la dernière sur le Z-80 est 65535. Sur un microprocesseur 8 bits, une adresse est codée sur 16 bits de façon à accéder à 64 Ko. Sur l'Amstrad, il existe parfois des adresses de 24 bits, indiquant les ROM ou RAM parallèles disponibles (par ce moyen, le Z-80 peut accéder à plus de 64 Ko de mémoire, ce qui n'est théoriquement pas possible). Des circuits spécialisés connectent les RAM ou ROM visées, et l'on peut ensuite accéder, dans les 64 Ko ainsi "visibles" du Z-80, à n'importe laquelle des 65536 cases.

Par abus de langage, on appelle également adresse n'importe quel nombre de 16 bits, même s'il n'est pas destiné à représenter une case mémoire. On peut dans ce cas parler de mot afin d'éviter les confusions.

☐ **Appel :**

en langage machine, un appel provoque l'exécution d'une routine. L'appel en Z-80 se nomme CALL, en Basic GOSUB (ou CALL également pour un appel de routine en langage machine). Une fois l'exécution de la routine achevée, celle du programme appelant se poursuit à l'endroit suivant le CALL.

☐ **Attribution de registres :**

étape de la programmation "Top-Down" en langage machine. Cela consiste à attribuer aux différents registres disponibles un rôle particulier lors des calculs. Cette étape évite les erreurs d'étourderie.

☐ **Binaire :**

mode de calcul en base 2. Les deux chiffres sont 0 et 1. Le binaire constitue le langage de compréhension du processeur : la plupart des opérations travaillent en binaire.

☐ **Bit :**

unité de stockage élémentaire. Un bit permet de stocker un seul chiffre binaire ; il possède donc la valeur 0 ou 1. Le Z-80, ainsi que la majorité des micro-ordinateurs, travaillent avec des groupements de huit bits, appelés octets. Le Z-80 possède également le nécessaire pour travailler avec des valeurs de 16 bits. Un bit permet de mémoriser un nombre compris entre 0 et 1, huit bits permettent de stocker entre 00000000 et 11111111 exprimés en binaire, soit 0 à 255 en décimal. Seize bits permettent de stocker les valeurs 0 à 65535. En anglais, bit signifie "Binary Digit", soit chiffre binaire.

☐ **Boucle :**

structure de programmation. Consiste à répéter une même routine, tâche ou instruction un certain nombre de fois. Ce nombre de répétitions peut dépendre du principe utilisé pour stopper la boucle. On peut boucler six fois, on peut également boucler jusqu'à ce qu'une certaine condition soit remplie. Pour mettre en place une boucle en langage machine, on utilise en Z-80 l'instruction DJNZ ou des tests suivis de sauts conditionnels.

☐ **Buffer :**

zone mémoire réservée au stockage intermédiaire de données avant un traitement. On détermine la taille d'un buffer par le besoin maximal prévu. Ainsi si l'on prévoit d'avoir de 10 à 120 octets à mémoriser, il faudra réserver 120 octets pour un buffer. Et cela même si dans 99 % des cas on ne dépassera pas 50 octets.

☐ **Carry :**

flag particulier présent sur tout processeur. Il s'agit d'un bit du registre d'état (registre F sur le Z-80) qui indique les dépassements de capacité. Le Carry est positionné par exemple lorsque l'addition d'un nombre au registre A conduit à un nombre de plus de huit bits. Ce nombre ne peut pas être stocké dans A (qui ne possède que huit bits), et le "1" dépassant est envoyé dans le Carry.

- ☐ **Compactage :**
 procédure visant à réduire l'encombrement d'une table de données. On peut par exemple recourir à un compactage lorsqu'on stocke cent nombres dont les valeurs s'échelonnent entre 0 et 15 : au lieu d'utiliser cent octets pour stocker ces nombres, on peut en utiliser cinquante en stockant deux nombres par octet.
- ☐ **Décalage :**
 opération binaire consistant à déplacer les chiffres binaires d'un nombre vers la gauche ou la droite, en ajoutant un autre chiffre à l'espace ainsi créé. On utilise généralement les décalages dans les opérations arithmétiques. Un décalage à gauche multiplie par 2, un décalage à droite divise par deux.
- ☐ **Décrémentation :** soustraction d'une unité (voir incrémentation).
- ☐ **Entrelacement :**
 organisation particulière du contenu de la RAM-Ecran, permettant de simplifier la génération du signal vidéo par le contrôleur. Les bits correspondant aux points ne sont pas regroupés séquentiellement.
- ☐ **Flag :**
 information binaire (0 ou 1) reflétant le résultat d'une opération. Le Z-80 possède six flags, regroupés dans le registre F (deux bits y sont inutilisés). Ces flags sont "positionnés" (mis à 1) ou non (mis à 0) suivant les travaux antérieurs. Par exemple, "CP n" provoque le positionnement du flag C (voir "Carry") si le contenu du registre A est strictement inférieur au nombre n, et sa mise à zéro dans le cas contraire.
- ☐ **Fond (couleur) :**
 le fond de l'écran est généralement représenté par le stylo 0. La couleur peut être quelconque. On convient donc d'utiliser une, trois ou quinze couleurs suivant le mode de résolution choisi, plus une pour le fond. Il vaut mieux dissocier le fond et ne pas le considérer comme une couleur, mais plutôt comme l'absence de couleur. Les traitements graphiques de transparence en sont facilités.
- ☐ **Graphe hiérarchique :**
 schéma résumant l'organisation des routines d'un programme. Ce graphe permet de programmer modulairement. Il est généralement utilisé avec les langages structurés comme Pascal, mais rien ne s'oppose à la programmation structurée en langage machine : la structuration concerne l'organisation du programme et non sa programmation proprement dite.
- ☐ **Hexadécimal :**
 mode de numération en base 16. Les chiffres de cette base vont de 0 à 9, suivis de A à F (qui représentent 10 à 15, exprimés en décimal). La base 16 est très utilisée en langage machine pour son aspect pratique. En effet, un octet se représente par deux chiffres (valeurs 00 à FF), une

adresse par quatre chiffres. Un chiffre hexadécimal (0 à F) représente quatre bits, soit les valeurs 0000 à 1111. C'est exactement un demi-octet.

☐ **Incrémentation :**

addition d'une unité. On utilise souvent cette opération pour progresser facilement dans les tables de données. Si le registre HL pointe sur les données, il suffit d'utiliser INC HL pour progresser, sans perdre le contenu de A.

☐ **Indexation :**

adressage particulier. Du strict point de vue de la définition, le Z-80 ne possède pas de vraie indexation. Mais il dispose de deux registres 16 bits IX et IY qui permettent un semi-adressage indexé. Si IX contient l'adresse d'une table de données, on pourra incrémenter le quatrième octet par une instruction INC (IX+3).

☐ **Interface utilisateur :**

ensemble des routines chargées de prendre en compte les actions de l'utilisateur (clavier, joystick, souris...) et de les fournir au programme dans un format pratique. On peut également considérer que les routines chargées de fournir les résultats à l'utilisateur en font partie.

☐ **Interruption :**

événement externe au programme provoquant la suspension de son déroulement, l'exécution d'une routine de traitement de l'interruption, puis la reprise du programme. Il existe les interruptions matérielles et les logicielles. Les interruptions matérielles sont invisibles, elles ne concernent que le système. En revanche, on peut mettre en place des interruptions logicielles de façon à traiter des événements réguliers indépendants du programme. C'est par exemple le cas lorsqu'on fait clignoter un stylo avec deux couleurs : le stylo n'a en fait qu'une seule couleur. Mais une interruption logicielle est mise en place pour modifier celle-ci régulièrement.

☐ **Kilo-octet :**

unité de mesure de capacité mémoire. Un kilo représente 1 024 octets. C'est bien de 1 024 qu'il s'agit et non 1 000. A l'origine de cette étrange unité de mesure se trouve un aspect pratique. En effet, 1 024 se représente par 400 en hexadécimal. C'est un nombre pratique pour les calculs, puisque la plupart du temps les calculs ont lieu en base 16. L'idée originale était de prendre le nombre hexadécimal le plus proche de 1 000 (Kilo) et le plus pratique. Ce fut 400, soit 1 024 ! le "K" s'applique également aux bits. Ainsi, les circuits intégrés mémoire donnent-ils accès à des bits, généralement exprimés en Kbits (1 024 bits). Les 4 116 sont des circuits 16 Kbits (il en faut huit en parallèle pour obtenir 16 Kilo-octets), les 4 164 possèdent 64 Kbits (huit circuits 4 164 constituent la mémoire de l'Amstrad CPC 464), et les plus récents, les 256 Kbits, se nomment 4 256 (voir aussi Méga-octets).

☐ **Lutin :** voir objet graphique.

☐ **Méga-octet :**

unité de mesure de capacité mémoire. Cette unité, principalement utilisée sur les ordinateurs à processeur 32 bits (mini-ordinateurs) ou sur certains micro-ordinateurs 16 bits, représente 1 024 Kilo-octets, soit très exactement 1 048 576 octets. En hexadécimal, cela devient 100 000 octets. Encore une fois (cf. Kilo-octets), l'aspect pratique de programmation a prévalu. On obtient un Méga de mémoire vive en juxtaposant seize Amstrad CPC 464 ou 664 ! Sur les ordinateurs vraiment gros, on parle en Giga-octets, soit 1 024 Méga !

☐ **Mémoire morte :**

on appelle ainsi un circuit mémoire non modifiable. Son contenu est fixé par le fabricant, sur la demande d'un constructeur de matériel. Il est inamovible. La mémoire morte est indispensable au moins lors de la mise sous tension : le processeur doit avoir un programme, et il va chercher celui-ci dans le premier bout de mémoire qu'il connaît. L'Amstrad possède deux blocs de mémoire morte de 16 Ko chacun. L'un contient l'interpréteur Basic, le second le système d'exploitation. Ces deux blocs occupent les mêmes adresses que deux blocs de 16 K de mémoire vive. Il va de soi que chaque bloc d'adresses n'est connecté qu'à un seul des blocs mémoire à la fois. Il est possible de connecter ou déconnecter, en langage machine, chacun de ces blocs.

☐ **Mémoire vive :**

au contraire de la morte, cette mémoire est modifiable. Elle reçoit les informations du système, les programmes, ou les données. On peut réellement en faire n'importe quoi. Toutefois, une zone de la mémoire vive est réservée à l'écran, et quelques autres zones sont utilisées par le système dans sa gestion interne.

☐ **Mips :**

unité de mesure de rapidité de calcul. Mips signifie Million d'instructions par seconde. Sur un micro-ordinateur 8 bits, on parle peu de cette unité de mesure, car il s'agit plus de milliers d'instructions par seconde. On se base plus sur le processeur utilisé et la fréquence de son horloge pour estimer sa rapidité. Un processeur Z-80 à 4 MHz représente une très bonne performance à ce niveau. Mais beaucoup d'autres éléments sont à prendre en compte (optimisation du logiciel interne, organisation matérielle du système, capacité graphiques, etc) pour une bonne évaluation.

☐ **Mode graphique :** voir résolution.

☐ **Mode transparent :**

ce mode graphique de travail n'existe pas au niveau matériel. Mais il est facilement simulable par logiciel. Il consiste à ne pas utiliser la couleur de fond, et à agir comme si celle-ci était transparente. En d'autres termes, un point tracé avec le stylo 0 (par convention, le stylo 0 est associé au fond) est considéré comme inexistant : on ne le trace pas. L'avantage de ce procédé est de permettre un dessin de personnage sur

un décor sans effacer celui-ci. Il existe d'autres variantes du mode transparent : le mode XOR est le plus pratique, mais n'est pas toujours beau.

☐ **Objet graphique :**

dessin élémentaire (personnage, partie de décor...) manipulable. Sur Amstrad, les objets graphiques doivent être gérés par programme. On les inscrit généralement dans un rectangle, de façon à accélérer leur traitement. Certaines machines possèdent des circuits capables de gérer de tels objets (Texas 99/4A, Commodore 64, MSX ...). L'avantage essentiel est de faciliter les traitements, mais on y perd parfois en souplesse. Sur l'Amstrad, il faut programmer la gestion complète des objets graphiques, ce qui autorise une définition "sur mesure".

☐ **Octet :**

huit bits. L'octet est la plus petite unité mémoire accessible à un processeur 8 bits. Le Z-80 possède toutefois certaines instructions de travail sur les bits. Mais le plus souvent, un programme travaille à partir d'octets. Un octet permet de mémoriser les nombres 0 à 255 (00 à FF en hexadécimal). Chaque octet de la mémoire utilisable est caractérisé par son numéro, appelé adresse.

☐ **Pointeur :**

variable ou registre contenant l'adresse d'une information (voir indexation, adressage, incrémentation).

☐ **Port d'entrée/sortie :**

élément du processeur destiné à la communication avec les circuits secondaires et périphériques. Ainsi, le contrôleur vidéo et le gate-array sont programmables grâce aux ports d'E/S qui leur sont attribués. Il en va de même avec la majorité des autres circuits du processeur. Le Z-80 possède des instructions pour envoyer des données sur les ports ou en recevoir.

☐ **RAM :**

signifie "mémoire à accès quelconque" en anglais. En français, on parle de mémoire vive (on peut lire ou écrire des informations) ou de MEV. Il existe d'autres versions : CRAM, notamment, qui indique une mémoire vive constante. Ces CRAM gardent leurs informations lorsque le courant est coupé (voir aussi mémoire vive).

☐ **RAM-Ecran :**

partie de la mémoire vive où est mémorisé l'écran. Cette RAM n'est guère utilisable pour y stocker un programme, encore que cela soit imaginable sur l'Amstrad, car 384 octets de cette RAM ne sont pas pris en compte.

☐ **Registre :**

élément de mémoire du processeur. On peut assimiler les registres à des cases mémoires, à ceci près qu'ils sont intégrés au processeur. Ils ont pour but de mémoriser les arguments des instructions Z-80. Ainsi,

l'instruction "LD A, (HL)" permet de ranger dans le registre A la donnée se situant à la case mémoire dont l'adresse est contenue dans HL. HL et A permettent dans ce cas précis de récupérer, dans le processeur, une information située en RAM externe.

☐ **Résolution :**

capacité de l'écran, en terme d'informations. L'Amstrad possède trois résolutions au choix de l'utilisateur, suivant le nombre de points graphiques, de caractères ou de couleurs qu'il désire.

☐ **Rotation :**

opération binaire consistant à décaler un octet à gauche ou à droite d'un bit en récupérant le bit éjecté de l'autre côté.

☐ **Saut :**

opération consistant à sauter d'un endroit du programme à un autre. Sur le Z-80, l'instruction de saut se nomme JP. Il existe aussi JR, un peu différent par son encombrement mémoire, sa rapidité et son fonctionnement. Les sauts peuvent être conditionnels. Cela signifie qu'un test leur est associé, concernant un des flags du Z-80. Si le test conduit à un résultat vrai, le saut est effectué, sinon il est ignoré.

☐ **Signé (nombre) :**

les opérations arithmétiques du langage machine nécessitent parfois l'existence de nombres négatifs. Or, ceux-ci ne peuvent pas être codés en binaire : un bit peut mémoriser un chiffre 0 ou 1, mais pas un signe "-". Il existe donc une convention sur laquelle l'information se base : celle des nombres signés. Elle indique une façon de considérer un nombre binaire positif. Pour une donnée "n" bits (n est 8 ou 16 en Z-80), "n-1" bits contiennent la valeur du nombre, le dernier bit étant utilisé comme signe. En l'occurrence, c'est toujours le bit le plus à gauche du nombre qui est pris comme signe si le nombre doit être considéré comme signé. Un bit à 1 indique un nombre négatif (voir l'annexe 1).

☐ **Sprite :** voir lutin, objet graphique.

☐ **Stylo :**

élément imaginaire associé à une couleur. Chaque point de l'écran est associé, dans la RAM-Ecran, à une valeur numérique. Cette valeur numérique est le numéro de stylo. Selon la résolution, l'Amstrad possède 2, 4 ou 16 stylos, autorisant donc 2, 4 ou 16 couleurs différentes sur l'écran. Le stylo 0 est généralement considéré comme celui du fond. Le stylo n'a aucun rapport avec sa couleur : rien n'empêche de programmer la même couleur pour deux stylos différents.

☐ **Système d'exploitation :**

ensemble des routines systèmes. Ces routines donnent accès aux fonctionnalités de la machine (à savoir les points graphiques, les ROM complémentaires, les périphériques, etc). Le système d'exploitation de l'Amstrad remplit 16 K de ROM, et est accessible grâce aux blocs de vecteurs, ou bien par appel utilisant une adresse de 24 bits. Bien que le

système soit indépendant du Basic (ce qui n'est pas le cas sur 99 % des micro-ordinateurs), certaines fonctionnalités du Basic sont aussi intégrées dans les blocs de vecteurs, en particulier les routines de calcul en nombres réels. Mais le système d'exploitation lui-même ne les utilise pas.

☐ **Top-down :**

méthode de travail. Consiste à définir le programme en raffinant ses tâches par étapes. On parle aussi de niveau : au niveau 0, l'analyse comporte les données à fournir, le travail effectué, et ce qui doit en sortir. Puis on prend chacun de ces éléments et on les raffine un peu, et ainsi de suite, jusqu'à obtenir un ensemble de tâches indépendantes, qu'on nomme modules. On peut alors définir les variables et structures de données nécessaires, et enfin programmer les modules dans l'ordre inverse (du plus bas vers le plus haut). Par cette méthode, on ne passe à un niveau plus haut que lorsque les routines d'un niveau sont toutes au point. La méthode TOP-DOWN vient en partie des romanciers (!) et des travaux de Kathleen Jensen et Niklaus Wirth, inventeurs du langage Pascal et de la programmation structurée. Elle est adoptée de façon quasi unanime en informatique à la place des organigrammes depuis cinq ou six ans. Le principe de base de ce type de travail est le suivant : dès qu'apparaît une tâche donnée pour laquelle on aimerait disposer d'une instruction jouant le même rôle, on remplace l'énoncé de cette tâche par l'appel d'un module, lequel est analysé par ailleurs. L'un des principes les plus importants est "analyse du haut vers le bas, programmation du bas vers le haut".

☐ **Variable 8/16 bits :**

emplacement fixé de la mémoire où l'on place une donnée de travail. On peut avoir une variable 8 bits pour stocker un octet (il suffit alors de réserver une adresse mémoire à cet effet) ou 16 bits pour stocker une adresse, un pointeur ou un simple nombre 16 bits (il faut dans ce cas réserver deux adresses, si possible successives).

☐ **Vecteur :**

instruction de saut placée à une adresse fixée et connue. Les routines systèmes situées en ROM sont toutes accessibles grâce à un bloc de vecteurs placés en RAM lors de la mise sous tension de l'Amstrad. L'emplacement de ces vecteurs ne change pas quel que soit le modèle. L'appel d'une routine système du 464 par le biais de son vecteur aura donc exactement le même effet sur le 6128, même si la routine ne se situe absolument pas au même endroit en ROM. Une illustration souvent utilisée pour expliquer ce fonctionnement est la parabole d'Alfred, Barnabé et Christian. Alfred cherche Christian, qui peut être en trois endroits différents. Or, Barnabé sait TOUJOURS où se trouve Christian. Il est donc évident que Alfred va trouver Christian en passant par Barnabé. Dans l'Amstrad, Alfred devient le programme, Barnabé le vecteur et Christian la routine système.

CALL basic	50
CALL Z-80	48
CRTC 6845	12
DRAW	25
EQU	148
GATE-ARRAY	12
LD	41
MEMORY	49
MOVE	25
PLOT	25-68
POP	45
PUSH	45
TESTE	27
assembleur	32
avant plan	199
bits	18-180-A7
carry	47
cercle	60
collision d'objets	208
complémentation	68-103
compactage	166
coordonnées (système de...)	24-208
division	87
drapeau	cf flag.
entrelacement	18
flags	37-47
fond	195
histogramme	79
joystick	180
masque	A3
mémoire écran	13-A6
mode XOR	191
multiplication	63-88
nombre flottant	62
nombre signé	A7
objet graphique	128
pile	39
point physique	24
point logique	24
registre	37
reg A	37
reg B	38
reg C	38
reg D	38
reg E	38
reg F	37
reg H	38
reg L	38
reg I	38
reg R	38

reg PC	38
reg SP	38
reg IX	38
reg IY	38
remplissage de zone	105
résolution	17
stylo	22
système d'exploitation	23-58-AZ
territoire interdit	208
transparence	195
vecteur	58
zéro (drapeau)	48

Annexe 9

La disquette d'accompagnement

Vous pouvez utiliser le bon de commande ci-joint (ou une photocopie) afin de vous procurer la disquette correspondant au livre. Cette disquette vous évitera les pénibles tâches de saisie des programmes. En voici le contenu détaillé par face.

- **FACE A :** 1) Les 18 programmes Basic des chapitres 1 à 3 sont contenus tels quels sur cette face de la disquette.
- 2) La première face contient également les fichiers sources des 20 programmes en assembleur des chapitres 1 à 8. Ces fichiers sont au format ASCII et peuvent donc être utilisés par un assembleur quelconque.
- 3) Un utilitaire de listage des programmes sources assembleur est également fourni, vous permettant de lister sur écran ou imprimante les fichiers voulus.
- 4) Une routine supplémentaire et son programme Basic de mise en œuvre (ainsi que le fichier source de la routine) sont également proposés. Cette routine, non présente dans le livre, permet le défilement souple d'un message de 26 caractères sur l'écran. La vitesse de défilement est réglable. Un message exemple est fourni dans un fichier.
- 5) Un programme menu permet de choisir parmi les programmes de cette face l'application désirée, sans avoir à se souvenir du nom de fichier correspondant. Il suffit de demander "RUN" .

- **FACE B :**
- 1) Les programmes DESSIN, IMAGECR, IMAGOBJ sont livrés sur cette face. Ils correspondent aux trois utilitaires du chapitre 9. Le programme assembleur du programme de dessin est également présent ainsi que son source, toujours au format ASCII.
 - 2) Les 11 programmes d'application des chapitres 4 à 8 sont proposés sous une nouvelle forme. Il y est tenu compte de la structure de fichiers définie au chapitre 9. Il est donc possible d'utiliser chacun de ces programmes avec des objets et décors quelconques. Il suffit pour cela de définir les fichiers objet et décor utilisés avec le programme SETSYS.
 - 3) Le programme listeur de sources assembleur est également présent sur cette face.
 - 4) Un grand nombre de fichiers images sont présents sur la face B :
 - 3 fichiers sont destinés au programme dessin. Ils contiennent quelques objets, et l'image utilisée dans le livre (chapitres 4 à 8) assortie avec des couleurs pour les deux types de moniteurs.
 - 3 fichiers objets créés par DESSIN sont destinés au programme IMAGOBJ. Ils reprennent trois phases d'animation d'une puce.
 - 1 fichier résume ces trois phases sous le nom PUCE.
 - 1 fichier PARAM.SYS contenant le nom de l'objet et du décor utilisé (remise à jour par le programme SETSYS).
 - 5) Le fichier SYSTEM.BIN contient les 12 routines LM des chapitres 4 à 8, routines permettant la gestion des objets.
 - 6) Enfin, un programme menu permet l'exécution de tout programme situé sur cette face (RUN" ").

Sur chaque face de la disquette se trouve également un programme explicatif fournissant des renseignements complémentaires. Pour obtenir ses indications, il suffit de demander RUN"EXPLIQUE".

Conseils de lecture

Pour approfondir vos connaissances en BASIC, mieux connaître le système des CPC 464, 664 et 6128, et maîtriser le graphisme sur Amstrad, P.S.I. vous propose une palette d'ouvrages utiles.

Pour maîtriser le BASIC Amstrad :

- ☐ **BASIC Amstrad 1 – méthodes pratiques** – Jacques Boigontier et Bruno Césard (Éditions du P.S.I.)
Pour ceux qui ont déjà pratiqué un BASIC, voici un ouvrage de perfectionnement au BASIC Amstrad. Un chapitre sur le CP/M 2.2 et le CP/M Plus donne les principales commandes systèmes.
- ☐ **BASIC Amstrad 2 – Programmes et fichiers** – Jacques Boigontier (Éditions du P.S.I.)
Pour pratiquer le BASIC Amstrad, cet ouvrage donne de nombreux programmes de gestion, d'éducation et de jeux où le rôle des fichiers est expliqué et largement commenté.
- ☐ **BASIC plus – 80 routines sur Amstrad** – Michel Martin (Éditions du P.S.I.)
Pour pousser votre Amstrad au maximum de ses capacités : 80 routines de simulation d'instructions qui n'existent pas en BASIC Amstrad.

Pour mieux connaître le système des CPC :

- ☐ **Clefs pour Amstrad 1 – système de base** – Daniel Martin (Éditions du P.S.I.)
Mémento présentant synthétiquement le jeu d'instructions du Z-80, les points d'entrée des routines système, les connecteurs et brochages, etc. Le livre de chevet du programmeur sur Amstrad.

- **Clefs pour Amstrad 2 – système disque** – Daniel Martin et Philippe Jadoul (Éditions du P.S.I.).

Ce deuxième tome consacré au système disque présente les points d'entrée des routines disque, les blocs de contrôle, la programmation et les brochages des circuits spécialisés... La deuxième partie du livre est aussi destinée aux possesseurs d'Amstrad 8256.

- **CP/M plus sur Amstrad** – Yvon Dargery (Éditions du P.S.I.). Toutes les commandes CP/M et CP/M plus pour maîtriser le système des 6128 et 8256 : un ouvrage de référence illustré par de nombreux programmes.

- **Le livre de l'Amstrad – Tome 1** – Daniel Martin et Philippe Jadoul (BCM – diffusé par P.S.I.).

Ce livre, destiné aux programmeurs des CPC 464 et 664, donne une étude complète de tous les circuits internes, et analyse la structure interne du BASIC. Vous y trouverez, en outre, une étude complète des RXS, et des programmes de scrolling, de traçage de rectangles, de coloriage de surface et de manipulation vectorielle.

Pour concevoir et améliorer vos graphismes :

- **Mathématiques et graphismes** – Gérald Grandpierre et Richard Cotté (Éditions du P.S.I.).

De très beaux graphismes sont générés par des équations mathématiques. L'univers des fractals, les déformations et les enveloppes, les surfaces en Z2 sont étudiées dans ce livre très pédagogique et de haut niveau. Tous les programmes, écrits en BASIC standard, sont facilement adaptables au BASIC Amstrad.

- **Création et animations graphiques sur Amstrad CPC** – Gilles Fouchard et Jean-Yves Corre (Éditions du P.S.I.).

Dessiner avec la souris ou le joystick et apprendre à faire des scrolling, à fabriquer une gomme, à inverser une image ou à l'éclater en une myriade de points, tel est l'objectif de ce livre écrit en assembleur Amstrad.

GRAPHISME EN ASSEMBLEUR SUR AMSTRAD CPC

Cet ouvrage est destiné aux possesseurs de CPC 464, 664 et 6128 qui souhaitent programmer des applications graphiques en assembleur.

Les débutants en assembleur Z80 trouveront dans ce livre de nombreuses façons de progresser, grâce à des routines prêtes à l'emploi et compatibles entre elles. Ces routines sont toutes livrées sous la double forme d'un programme Basic et d'un listing assemblé : vous pourrez ainsi les utiliser avec ou sans programme Assembleur.

Graphisme en assembleur vous propose de créer des graphismes très variés sur votre Amstrad CPC : tracé d'histogrammes, création d'une corne d'abondance, dessin d'un paysage avec zones réservées, et animation d'un module dans ce paysage n'auront plus de secret pour vous. Ces techniques graphiques vous permettront d'illustrer vos jeux d'aventures ou de rôle, et de maîtriser parfaitement toutes les possibilités graphiques de votre machine.



ÉDITIONS DU P.S.I.
BP 86 - 77402 LAGNY S/MARNE CEDEX - FRANCE

ISBN : 2 86595-340 8

145 FF