AMSTRAD

TURBO PASCAL

TECHNIQUES DE PROGRAMMATION

BERTRAND NELSON

SYBEX

Bernard NELSON

TECHNIQUES DE PROGRAMMATION EN TURBO PASCAL SUR AMSTRAD

Paris • San Francisco • Düsseldorf • Londres

ISBN 2-7361-0300-9

SOMMAIRE

INTRODUCTION

En étant proposé pour l'Amstrad CPC, le système TURBO PASCAL ouvre une dimension professionnelle que le BASIC d'origine ne pouvait offrir. En effet, ce langage associé au système d'exploitation CP/M permet de développer rapidement toutes sortes d'applications, notamment en bureautique. De plus, les programmes peuvent sans problème être adaptés sur d'autres machines CP/M tel l'Amstrad PCW ou même sous d'autres systèmes comme MS-DOS avec les compatibles IBM. Sur les Amstrad 464 et 664, seul CP/M 2.2 est disponible, sur le CPC 6128, CP/M 3.0 (ou CP/M Plus) plus performant peut être utilisé.

Tant que l'on respecte la syntaxe du TURBO PASCAL, la portabilité des programmes est assurée mais il est tentant d'utiliser toutes les possibilités de la machine hôte dans ses applications si l'on désire se limiter à une gamme restreinte de matériels.

Sur l'Amstrad, le TURBO PASCAL est livré en version standard et en version graphique avec de nombreuses extensions propres aux CPC. Les programmes du commerce actuels ont une interface utilisateur performante et font appel à des représentations graphiques et autres touches de fonction, souris ou joystick. Le langage TURBO PASCAL n'est pas suffisant pour créer de telles fonctionnalités et il sera nécessaire d'accéder directement à la mémoire ou d'utiliser le système Amstrad, le 'Firmware' en ROM, pour tirer pleinement parti de la machine.

Cet ouvrage s'adresse aux programmeurs qui veulent utiliser pleinement les capacités des systèmes CPC avec TURBO PASCAL, les explications et les programmes présentés sont pour les systèmes Amstrad CPC mais peuvent être adaptés à d'autres machines.

CHAPITRE 1
LES OUTILS DE BASE

Si l'on en a la possibilité, il est beaucoup plus avantageux de travailler sous CP/M Plus car la taille mémoire disponible pour programmer est d'environ 30 Ko au lieu de 7 à 8 Ko sous CP/M 2.2, les manipulations des disques sont plus aisées et l'on peut travailler avec des disques au format Data.

Par la suite, il suffit de revoir légèrement le programme pour le rendre exécutable sous CP/M 2.2. Il sera sans doute nécessaire de sectionner le texte source pour que celui-ci puisse se loger en mémoire, on utilisera alors des fichiers Include. Mais le plus gros problème viendra des nombreuses incompatibilités des deux systèmes au niveau du jeu de caractères, des caractères de contrôle et surtout de l'utilisation de sections en langage machine.

Il est donc primordial de connaître l'organisation des deux systèmes CP/M, les possibilités du TURBO PASCAL dans ce domaine et la gestion de l'accès à la mémoire et aux ressources de la machine. Le but des prochains chapitres est de vous donner toutes les informations nécessaires et des exemples de programmes pouvant être inclus dans vos applications. Certains de ceux-ci peuvent être utilisés sur d'autres systèmes avec un minimum de modifications.

LES INSTRUCTIONS UTILES DU TURBO PASCAL

Pour bien utiliser le code du Z 80, il est très important de bien connaître toutes les possibilités du TURBO PASCAL. Voici donc un rappel de certaines instructions qui seront souvent utilisées dans les programmes. Même si vous connaissez déjà parfaitement ces instructions, ne sautez pas ce chapitre car la plupart des exemples cités contiennent des informations utiles qui dépassent le cadre de chaque instruction.

Tableau MEM

C'est l'équivalent de PEEK et POKE en BASIC. MEM est assimilé à un tableau prédéfini d'index 0 à 65 535.

Une affectation à un élément équivaut à une écriture en mémoire :

MEM [$5C] :=10 écrit 10 à l'adresse mémoire $5C, c'est l'équivalent de POKE &5C,10 en BASIC.

Une lecture d'un élément revient à une lecture de la mémoire :

(Data est une variable de type Byte)
Data :=MEM [$5C] place dans Data le contenu de la case mémoire $5C, c'est l'équivalent de Data=PEEK (&5C) en BASIC.

Instruction INLINE

Elle permet d'inclure à n'importe quel endroit du programme des codes du microprocesseur Z 80. L'instruction se compose de plusieurs éléments séparés du signe '/' et le tout est entre parenthèses. Chaque élément est une expression composée d'une constante entière, d'un identificateur de variable, de procédure ou de fonction ou d'une référence au compteur d'emplacement avec les opérateurs '+' ou '-'. Quand il est fait référence à un identificateur, c'est l'adresse de la variable, procédure ou fonction qui est prise en compte.

Si un élément a une valeur comprise dans l'intervalle 0 à 255, un seul octet sera codé. Si l'élément a une valeur supérieure, 2 octets seront codés selon le format du Z 80, c'est-à-dire l'octet de poids faible d'abord puis celui de poids fort. Les signes '>' et '<' permettent de forcer le nombre d'octets qui seront codés.

Le signe '>' force le codage d'un mot de 2 octets même si la valeur ne tient que sur un seul octet.

Exemple :

```
Inline ($BE9B)  stockera en mémoire $9B puis $BE.
Inline (>$BE9B) fera de même.
Inline ($0005)  stockera seulement $05.
```

```
Inline (>$0005) stockera $05 puis $00 ce qui représente bien
l'adresse
                $0005 par exemple.
```

Le signe '<' force le codage de l'octet faible de l'expression seulement. Cela est utile par exemple pour coder des valeurs négatives dans les sauts relatifs.

Exemple :

```
Inline (
 $19/      {L00 : Add  HL,DE }
 $10/<-3        { Djnz L00    }
)
```

'<-3' donnera le code $FD qui correspond bien à -3 en représentation complémentaire à 2. Si l'on avait écrit seulement -3, il aurait été codé $FF puis $FD, c'est-à-dire le mot $FFFD.

Par souci de clarté, il est recommandé de faire précéder toute expression sur 2 octets par le signe '>' même si cela est explicite car cela facilitera le comptage des octets dans le cas de calculs pour les sauts (*jumps*).

La référence au compteur d'emplacement se fait avec le signe '*', celui-ci prend la valeur de l'adresse mémoire à l'endroit exact où il est placé. On l'utilisera dans les calculs pour les sauts absolus.

Exemple :

```
Inline (
 $C3/>*+3/      { Jp   Fin }
 $37/           { Scf      }
 $C9       {Fin : Ret      }
)
```

Le compteur est à l'adresse immédiatement après $C3 (Jp), l'adresse doit être codée sur 2 octets (Af et AF, adresse faible et adresse forte).

```
Décalage :       *+0 *+1 *+2  *+3
Octets :   $C3   Af  AF  $37  $C9
                 ^             ^
                 position      destination
                 compteur
```

Fonction ADDR

Elle renvoie sous la forme d'un entier l'adresse du premier octet occupé par l'argument, celui-ci peut être une variable, une procédure ou une fonction. Si l'argument est un tableau, il peut être indexé, si l'argument est un enregistrement (*record*), on peut choisir un champ particulier. Cette instruction est très intéressante car elle permet de travailler directement sur le contenu d'une variable en mémoire et aussi d'utiliser des variables en arguments des fonctions d'appel du BDOS.

Exemple :

Utilisation de la fonction BDOS 9 (impression d'une chaîne de caractères).

Le registre DE doit contenir l'adresse de la chaîne, celle-ci doit être terminée par le signe $. Soit STR une chaîne de caractères terminée par $, le premier octet occupé en mémoire contient la longueur de la chaîne, les octets suivants contiennent les codes ASCII des caractères. L'argument à transmettre est donc l'adresse du second octet (le premier caractère).

L'impression se fait avec BDOS (9,Addr(STR)+1).

Instruction ABSOLUTE

Elle permet de fixer l'emplacement d'une variable en mémoire. L'argument peut être une constante entière ou le nom d'une variable, auquel cas l'adresse sera celle du premier octet occupé par cette variable. Le TURBO PASCAL gère lui-même les allocations mémoire des constituants du programme, il est évident qu'il ne faut pas placer inconsidérément une variable en un quelconque endroit de la mémoire sous peine d'altération grave du programme ou du système. Dans la pratique, il y a deux principales utilisations de cette instruction.

1. *Recouvrir une autre variable*

   On peut déclarer une variable qui 'recouvre' tout ou partie d'une autre variable. La nouvelle variable ne prend pas de place en mémoire car celle-ci est déjà utilisée par la première variable.

Exemple :

```
PROGRAM Essai;
VAR Chaine1 : String [8];
    Chaine2 : String [2] ABSOLUTE Chaine1;

{
 CHAINE2 contiendra toujours les 2 premiers caract}res de CHAINE1,
sans qu'une affectation soit nécessaire.
}

Begin
 Write ('Entrer votre numéro de téléphone (8 chiffres)');
 Readln (Chaine1);
 Writeln ('L''indicatif est :',Chaine2)
End.
```

2. *Placer une variable en dehors de la TPA*

   On peut déclarer une variable à une adresse absolue en dehors de la zone réservée au programme (c.-à-d. la TPA).

Cela sera surtout utilisé pour 'recouvrir' une variable système du CP/M. Pour plus d'informations sur l'utilisation de la mémoire par le système, voir le chapitre consacré au CP/M.

Exemples :

Le FCB par défaut est situé à partir de l'adresse $5C et utilise 36 octets, on pourra manipuler plus aisément le FCB en déclarant une variable absolue à cette adresse.

```
VAR FCB : Array [1..36] Of Byte ABSOLUTE $5C;
```

En CP/M Plus, les 6 octets de $FC5D à $FC62 ne sont pas utilisés par le système et il peut s'avérer utile de s'en servir car comme nous le verrons plus tard, ils font partie de la banque de mémoire non commutée.

```
VAR Truc : Array [1..6] Of Byte ABSOLUTE $FC5D;
```

Dans ces deux exemples, la variable absolue n'utilise pas de mémoire programme.

Instruction EXTERNAL

Cette instruction permet de définir des procédures ou des fonctions écrites en langage machine. Celles-ci peuvent être déjà existantes en mémoire (routines CP/M) ou chargées par le programme pendant l'exécution (routines créées avec un assembleur par exemple). Une procédure ou fonction de ce type n'a pas de bloc d'instructions BEGIN..END et se déclare comme suit :

```
PROCEDURE UserF EXTERNAL $FC5A;
```

L'appel de cette procédure équivaut à une CALL $FC5A en langage machine. En fait, cela permet de nommer toute routine en langage machine qui est ajoutée au programme en Pascal. La routine doit se terminer par l'instruction RET du Z 80 (code $C9).

De plus, on peut passer des paramètres de la même manière que pour une procédure ou une fonction classique. Néanmoins, c'est la routine en langage machine qui devra dépiler elle-même les paramètres situés dans la pile du Z 80 (voir le paragraphe suivant : 'Utilisation des éléments du TURBO PASCAL en langage machine').

Exemple :

```
PROCEDURE Test (x,y : Integer) EXTERNAL $C000;
```

Instruction BLOCKREAD

Cette instruction sera utilisée dans cet ouvrage pour la lecture de fichiers binaires, par exemple des images écran ou des fichiers de codes machine. Ceux-ci seront déclarés comme des fichiers sans type (*file*) et la lecture se fera avec BlockRead : elle se fait par blocs de 128 octets, soit un enregistrement CP/M, et on peut se positionner sur un enregistrement particulier avec l'instruction SEEK.

La syntaxe est :

```
BLOCKREAD (Variable-fichier, Variable, Nombre de blocs <,blocs lus> ).
```

La lecture se fait dans le fichier désigné par 'Variable-Fichier' à partir de la position courante, les octets se placent à partir du premier octet occupé par 'Variable', le 'nombre de blocs' indiquant le nombre d'enregistrements de 128 octets à lire.

Le paramètre facultatif 'blocs lus' indique le nombre de blocs réellement lus, ce nombre pouvant être inférieur au nombre demandés en cas d'erreur (fin de fichier ou erreur lecture). Si ce paramètre est absent et qu'il y a moins de blocs à lire que demandés, une erreur se produit, il vaut mieux l'ajouter si l'on ne connaît pas la taille exacte du fichier.

Exemple :

```
{ n est une variable de type Integer }
BlockRead (Fic,Var,255,n);
```

lit le fichier FIC même s'il fait moins de 255 enregistrements de longueur. Voir Chapitre 4 : ('Manipulation des fichiers binaires' pour les exemples d'utilisation.)

UTILISATION DES ELEMENTS DU TURBO PASCAL EN LANGAGE MACHINE

Pour tirer pleinement profit du langage machine, il est nécessaire de pouvoir utiliser les variables, les procédures et les fonctions dans les parties en code machine. Lorsqu'un identificateur du TURBO PASCAL (variable, procédure ou fonction) est utilisé dans une instruction INLINE, c'est l'adresse en mémoire de l'identificateur qui est codée (2 octets), grâce à cette possibilité nous allons pouvoir 'interfacer' le TURBO PASCAL et le code machine.

Dans le cas des variables, le stockage en mémoire est différent selon le type de la variable (voir le manuel du TURBO PASCAL pour tous les détails). Le cas des constantes est particulier car seules les constantes avec type sont stockées de manière similaire aux variables, les constantes sans type n'ont pas de localisation précise en mémoire car elles sont insérées littéralement dans le programme à chaque utilisation.

Exemple :

(Var est une variable ou une constante d'un type quelconque)

```
Inline ($21/>Var); { Ld HL,Var }
```

charge HL avec l'adresse de Var, c.-à-d. l'adresse du premier octet occupé par VAR.

```
Inline ($3A/>Var); { Ld A,(Var) }
```

charge A avec le contenu du premier octet occupé par VAR.

Si VAR est du type Byte, A contient la valeur de VAR. Si VAR est du type String, A contient la longueur de la chaîne. Si VAR est du type Boolean, A contient 0 pour False et 1 pour True.

```
Inline ($3A/>Var+6); { Ld A,(Var+6) }
```

Si VAR est du type ARRAY [d..f] OF BYTE, A contient la valeur de VAR [d+6].

Si VAR est du type String, A contient le code ASCII du 6e caractère.

Pour les procédures, il faut distinguer les procédures sans ou avec paramètres. Une procédure peut être considérée comme un sous-programme en langage machine. L'adresse qui est fournie par le TURBO PASCAL est l'adresse du premier octet occupé par la procédure et le END final correspond à un RET ($C9) en langage machine. Si la procédure ne nécessite pas de paramètre, il suffit d'exécuter un CALL en langage machine.

Exemple :

```
PROCEDURE Reponse;
Begin
 {...}  { Instructions quelconques }
End;

PROCEDURE Principale;
Begin
 Inline ($CD/>Reponse) { Call Reponse }
 {...}
End;
```

L'effet est le même si l'on avait simplement appelé REPONSE en utilisant de façon standard le mot REPONSE :

```
PROCEDURE Principale;
Begin
 Reponse;
 {...}
End;
```

On peut aussi simplement passer des paramètres directement par les registres du Z 80 :

```
PROCEDURE Reponse;
{
 Le contenu de l'accumulateur est le paramètre.
}
Var Param : Byte;
Begin
 Inline ($32/>Param);  { Ld   (Param),A ;Param reçoit la valeur de A
}
 If Param=0 Then {...} {Note : La procédure peut être uniquement en
code}
            Else {...}       {machine}
End;

PROCEDURE Principale;
Begin
 {...}
 Inline (
  $3E/$01/             { Ld   A,$01    }
  $CD/>Reponse         { Call Reponse  ;A=1}
 )
 {...}
 Inline (
  $3E/00/              { Ld   A,$00    }
  $CD/>Reponse         { Call Reponse  ;A=0}
 )
End;
```

Dans le cas où la procédure ou la fonction nécessite des paramètres, il faut les empiler avant l'appel dans la pile du Z 80. Les paramètres par valeur doivent être empilés selon un format dépendant de leur type ; pour les paramètres par variable (VAR), il faut empiler l'adresse de la variable. Le TURBO PASCAL se charge

au début de la procédure de dépiler les paramètres et de transmettre les valeurs dans les variables désignées.

Dans le cas de procédures ou fonctions EXTERNAL, les paramètres ne sont pas dépilés automatiquement et c'est à la routine de manipuler la pile pour les récupérer. Le premier élément de la pile étant toujours l'adresse de retour, il faut d'abord la dépiler (POP) pour accéder aux paramètres puis la replacer sur la pile pour que le RET final se fasse correctement. Les fonctions EXTERNAL doivent rendre leurs résultats dans les registres ou la pile du Z 80. (Pour tous les détails concernant le passage et le format des paramètres, voir votre manuel du TURBO PASCAL.)

Exemples :

Cas d'un paramètre de type entier : la valeur (2 octets) doit simplement être placée sur la pile. Un entier est stocké sur 2 octets en mémoire selon le format binaire du Z 80 (octet faible suivi de l'octet fort).

```
PROCEDURE Couleur (n : Integer); { N est un paramètre par valeur }
Begin
 { Instructions quelconques }
 { Si on écrit :
  Inline ($2A/>n)       { Ld   HL,(n) ;Charge HL avec le contenu de N}
  le registre HL contiendra la valeur réelle de N.
 }
End;

PROCEDURE Principale;
Var numero : Integer;
Begin
 Write ('Numéro de la couleur ?');
 Readln (numéro);
 Inline (
  $2A/>nn/       { Ld   HL,(nn)   ;Charge HL avec la valeur de numéro}
  $E5/           { Push HL        ;place en paramètre}
  $CD/>Couleur   { Call Couleur   ;Appel }
 );              { le code RET est inutile}
End;
```

L'instruction Inline a le même effet que : Couleur (nn).

Si Couleur était une procédure EXTERNAL :

```
PROCEDURE Couleur (n : Integer) EXTERNAL $F000;
```

La routine en $F000 devrait commencer ainsi :

```
POP  HL    ;récupère adresse de retour
POP  DE    ;récupère paramètre N dans DE
PUSH HL    ;replace adresse de retour
...        ;instructions de la procédure.
RET        ;retour
```

Si la procédure nécessite un paramètre par variable :

```
PROCEDURE Couleur (VAR n : Integer);
Begin
 { Instructions quelconques }
 { Si on {crit :
  Inline ( $2A/>n )    { Ld   HL,(n) ;charge HL avec le contenu de N}
```

Le registre HL contiendra l'ADRESSE de N, pour charger la valeur réelle de N, il faudra faire ensuite par exemple :

```
  Inline (
   $5E/          { Ld   E,(HL)    ;charge octet faible}
   $23/          { Inc  HL        ;puis l'octet fort}
   $56           { Ld   D,(HL)    ;DE est chargé avec la valeur de n}
  )
 }
End;

PROCEDURE Principale;
Var numero : Integer;
Begin
 Write ('Numéro de la couleur ?');
 Readln (numéro);
 Inline (
  $21/>nn/       { Ld   HL,nn     ;Charge HL avec l'ADRESSE de nn}
  $E5/           { Push HL        ;place en paramètre}
  $CD/>Couleur   { Call Couleur   ;Appel }
 );              { le code RET est inutile}
End;
```

CHAPITRE 2
CP/M

Créée en 1981, la version 2.2 de CP/M ne peut gérer plus de 64 Ko de mémoire, les systèmes CPC 464 et 664 ne peuvent utiliser que cette version. Pour le CPC 6128, la version 3.0 (appelée aussi CP/M Plus) est fournie avec la machine, elle permet de gérer plus de 64 Ko de mémoire et intègre de nombreuses fonctions nouvelles. Il ne faut pas oublier qu'en plus de la mémoire vive, le système Amstrad gère aussi 3 ROM de 16 Ko dont deux sont utilisées sous CP/M, ce qui porte l'étendue de mémoire à 112 Ko et 176 Ko pour CP/M 6128. Le micro processeur Z 80 ne peut physiquement adresser un espace supérieur à 64 Ko, de l'adresse 0 à $FFFF, et c'est en plaçant plusieurs circuits mémoire aux mêmes adresses physiques et par le jeu d'habiles commutations de ces circuits que le système retrouve ses petits. Il faut noter qu'une instruction d'écriture se fera toujours dans la RAM quel que soit le circuit connecté.

Dans ce chapitre, nous allons décrire l'organisation générale des deux systèmes CP/M puis ses particularités sur Amstrad qui nous ouvriront les portes pour utiliser le système dans sa totalité.

TERMES UTILISES

Pendant l'exécution d'un programme sous CP/M, la mémoire de l'adresse 0 à $FFFF se trouve dans une configuration particulière, c'est l'ESPACE DE TRAVAIL directement adressable par le microprocesseur, il contient principalement le programme d'application. Le reste de la mémoire ne peut être accessible que par commutation, il sera appelé ESPACE COMMUTE.

Le FIRMWARE est le nom donné aux routines de gestion du système Amstrad qui sont contenues dans la ROM de l'adresse $0000 à $3FFF. Cette ROM est utilisée par le BASIC, CP/M ainsi que tous les programmes en Assembleur. Les routines de la ROM Firmware ne peuvent être appelées directement depuis un programme en RAM car la ROM est déconnectée en permanence.

La ROM BIOS contient le système d'exploitation AMSDOS ainsi qu'une partie du BIOS, c'est la ROM qui est associé au lecteur de disquettes. Elle est située aux adresses $C000 à $FFFF.

La ROM BASIC contient tout l'interpréteur du BASIC, elle est inutilisée sous CP/M. Elle est située aux adresses $C000 à $FFFF.

La RAM VIDEO contient l'espace de 16 Ko adressé par le contrôleur vidéo pour former l'image d'écran. Selon la version du CP/M utilisée, elle se trouve à des adresses différentes mais les routines système gèrent parfaitement ce décalage. On peut lire et écrire directement dans la mémoire vidéo pour réaliser des animations par exemple.

ORGANISATION MEMOIRE DE CP/M

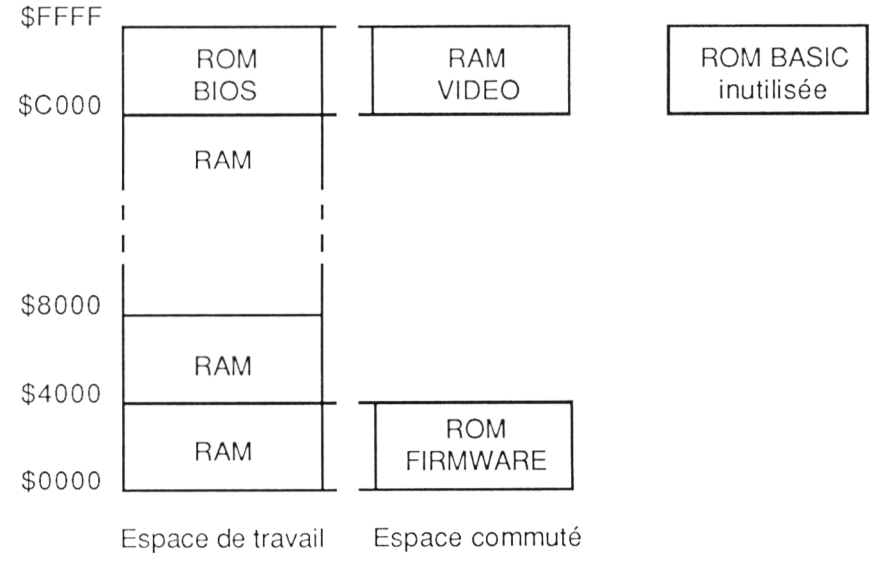


Figure 2.1 : *Carte mémoire du CP/M 2.2.*

Une lecture de la mémoire de l'adresse $C000 à $FFFF donnera le contenu de la ROM BIOS. Pour lire la mémoire vidéo, il faut donc d'abord déconnecter la ROM BIOS. Une écriture dans cette même zone se fera toujours dans la mémoire vidéo.

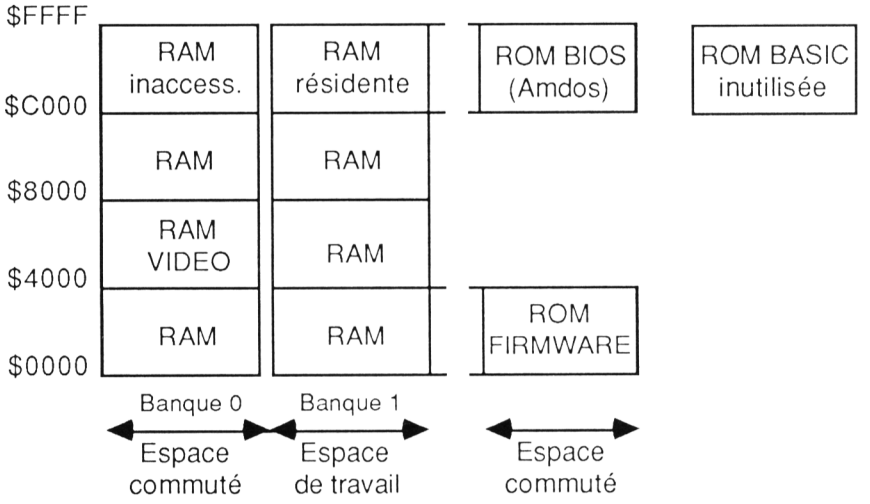


Figure 2.2 : *Carte mémoire du CP/M Plus.*

Pendant l'exécution d'un programme CP/M Plus, c'est la banque 1 qui est en ligne de l'adresse $0000 à $FFFF. Cette banque correspond aux 64 Ko de RAM supplémentaires du CPC 6128. La banque 0 correspond à la mémoire RAM de 64 Ko normalement utilisée avec le BASIC. Les accès aux ROM et à la banque 0 se font par des commutations de mémoire.

LES CONSTITUANTS DE CP/M

Il y a trois parties principales, elles sont communes aux deux CP/M.

- Le CCP *(Console Command Processor)* interprète les commandes entrées au clavier.
- Le BDOS *(Basic Disc Operating System)* traite la gestion logique des disques, sans tenir compte des données matérielles. Il s'occupe par exemple de la gestion du catalogue et de l'espace libre sur le disque.
- Le BIOS (Basic Input Output System) gère les relations entre le logiciel et le matériel (circuits d'entrées/sorties, vidéo, clavier, etc.). Le BIOS est spécifique à chaque ordinateur car il dépend du matériel utilisé.

Le CCP et le BDOS sont communs à toutes les machines CP/M. Le BIOS et une partie du BDOS se situent dans la ROM AMSDOS.

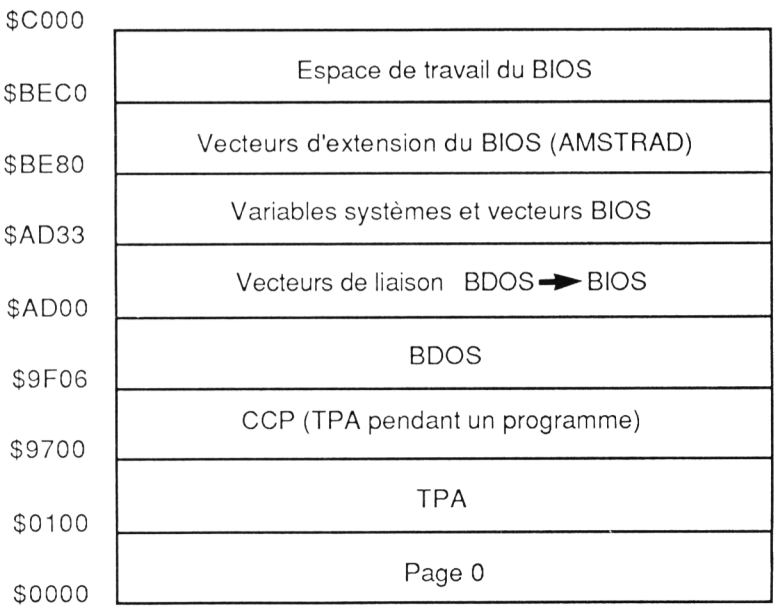


Figure 2.3 : *Détail de la RAM de travail du CP/M 2.2.*

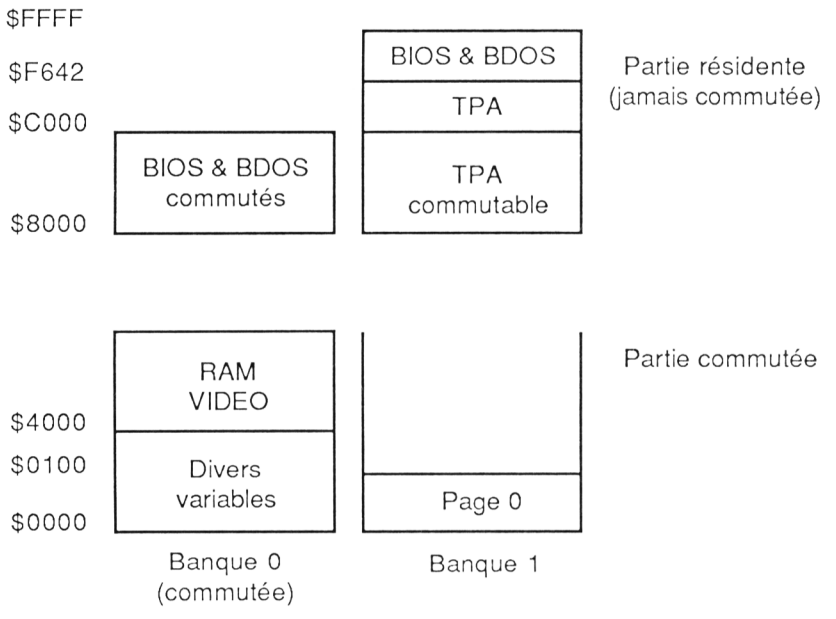


Figure 2.4 : *Détail de la RAM de travail de CP/M Plus.*

En fait, le BIOS est tout simplement le Firmware Amstrad, le BIOS dans la ROM AMSDOS se contente d'effectuer un prétraitement et d'appeler la ROM Firmware. Le BDOS et le BIOS non résidents ainsi que le CCP sont sur les deux premières pistes du disque système sous CP/M 2.2 ou dans le fichier C10CPM3.EMS sous CP/M Plus. Ils doivent donc être préalablement chargés depuis le disque au démarrage du CP/M.

La TPA (*Transient Program Area*) est la zone de mémoire vive qui reçoit les programmes, elle commence en $0100. La mémoire située juste avant, de $0000 à $00FF s'appelle la PAGE 0 et contient des vecteurs d'initialisation ainsi que divers variables et buffers du CP/M.

En CP/M 2.2, pendant l'exécution d'un programme, la TPA recouvre le CCP qui est alors inutile, offrant ainsi un surcroît de mé-

moire. Après l'exécution, le système cherche à recharger le CCP à partir de la disquette système qui doit alors se trouver dans le lecteur. Sous CP/M Plus, le CCP n'est jamais 'écrasé', on peut alors travailler avec des formats de disques différents.

Il faut bien comprendre le problème de la commutation :

Supposons que la mémoire vive soit constituée de deux banques indivisibles de 64 Ko, une seule banque étant en ligne à la fois. Le programme se déroule dans la banque 1 et la programmation d'un certain circuit (en l'occurrence le 'Gate-Array') par des OUT en code machine permet de choisir la banque 0 ou 1.

Si le programme veut utiliser une routine en banque 0 (du Firmware par exemple) à une adresse particulière, il devra 'brancher' la banque 0, appeler la routine à l'adresse choisie puis rebrancher la banque 1. Or, à partir du moment où la banque 0 est choisie par un OUT adéquat, le microprocesseur continue tout simplement l'exécution du programme en banque 0, à l'adresse suivante, ce qui se traduit irrémédiablement par la perte de contrôle du programme.

Pour remédier à cela, une partie de la banque 1, de $C000 à $FFFF, reste branchée en permanence ('résidente'), elle contient en particulier des programmes de commutations entre les banques, elle est toujours accessible au microprocesseur quelle que soit la banque commutée. Seuls les 48 Ko inférieurs, de $0000 à $BFFF, des deux banques sont commutés. La zone de $C000 à $FFFF de la banque 0 n'est plus accessible.

Contenu de la banque 0

Le CCP et le principal du BIOS et du BDOS se trouvent dans cette banque. La mémoire vidéo, contrairement au CP/M 2.2 (et au BASIC d'ailleurs), s'étend de $4000 à $7FFF. La zone de $0000 à $3FFF contient une série de tables et de buffers, en particulier les matrices du jeu de caractères.

Contenu de la banque 1 :

La TPA s'étend de $0100 à $F642. L'espace réservé de $F642 à $FFFF contient les vecteurs standard du BIOS et du BDOS résidents, des routines de transfert et de commutation entre les banques. La TPA située de $C000 à $F642 n'étant jamais commutée, elle pourra être utilisée pour implanter des programmes utilisant la banque 0, pour écrire dans la mémoire vidéo par exemple (voir 'Accès à la mémoire vidéo sous CP/M Plus').

ALLER PLUS LOIN...

A première vue, le système CP/M avec ses fonctions standardisées ne permet pas de faire plus que la norme ne le prévoit. Néanmoins, il est prévu de nombreuses portes vers l'extérieur et nous allons voir les possibilités d'accès aux nombreuses routines de la ROM de l'Amstrad (Firmware et BIOS) ainsi qu'à la mémoire vidéo.

Pendant l'utilisation en BASIC et en Assembleur, la ROM Firmware de l'Amstrad n'est jamais connectée dans l'espace adressable du microprocesseur. Pour accéder à ses routines, et aussi pour permettre la compatibilité des programmes avec les différentes versions de la ROM, une série de vecteurs fort connus des utilisateurs de l'Amstrad, sont installés en mémoire vive dans une zone fixe commune à tous les Amstrad. Ces vecteurs sont constitués dans leur majorité par une instruction RST-1 suivie d'une adresse 'inline' en paramètre sur 2 octets dont les 2 bits supérieurs donnent une information pour la commutation des blocs mémoire. L'instruction RST-1 se comporte comme un JUMP et l'adresse inline est la véritable adresse de la routine en ROM. Pour utiliser ces vecteurs, il est inutile de se préoccuper de cela, il suffit de préparer les paramètres et d'effectuer un CALL à l'adresse du vecteur.

Sous CP/M, ces vecteurs ainsi que les variables système du Firmware sont toujours présents en RAM, il serait donc possible de s'en servir aussi facilement qu'en Assembleur ou en BASIC. Nous pourrons ainsi inclure dans les programmes TURBO PASCAL des sections en langage machine utilisant toutes les possibilités de l'Amstrad, en graphisme par exemple. Il faut noter que les vecteurs

concernant les routines disque ne doivent pas être utilisés car CP/M possède sa propre logique de gestion des disques. D'ailleurs, sous CP/M ces vecteurs sont initialisés pour l'utilisation du lecteur de cassettes.

Il y a cependant des précautions à prendre que nous allons étudier maintenant.

PARTICULARITES DU CP/M 2.2

Les vecteurs d'extension du BIOS

Sur Amstrad, le BIOS possède des vecteurs supplémentaires qui n'existent pas sur les autres machines. Ils permettent d'appeler des routines présentes dans la ROM AMSDOS et aussi d'utiliser les vecteurs d'appels du Firmware. Les 9 premiers vecteurs correspondent aux commandes 1 à 9 de l'AMSDOS accessibles normalement en Assembleur (voir documentation Amstrad).

Adresse	**Fonction**
$BE80	Commande 1. Autoriser/inhiber messages d'erreur du disque.
$BE83	Commande 2. Spécifier les données du disque.
$BE86	Commande 3. Déterminer le format du disque.
$BE89	Commande 4. Lire un secteur.
$BE8C	Commande 5. Ecrire un secteur.
$BE8F	Commande 6. Formater une piste.
$BE92	Commande 7. Chercher une piste.
$BE95	Commande 8. Lire l'état des disques.
$BE98	Commande 9. Fixer le nombre de tentatives de lecture/écriture.
$BE9B	Utiliser le Firmware (Entrer-Firmware).
$BE9E	Fixer la sauvegarde des registres auxiliaires et de IY.
$BEA1 {à} $BEBC	Gestion de l'interface série, non développée dans cet ouvrage.

Accès au Firmware (Entrer-Firmware)

Pendant l'exécution d'un programme d'application, il ne faut pas changer les valeurs des registres auxiliaires (BC', DE', HL', AF') et du registre IY car celles-ci sont utilisées par CP/M. Les routines du Firmware peuvent modifier ces registres, en particulier RST-1 qui est utilisée pour les vecteurs d'appel du Firmware, il faut donc les sauver avant l'appel et les restituer à la fin de la routine.

Le vecteur d'extension du BIOS 'ENTRER-FIRMWARE' en $BE9B effectue cette opération, il reçoit en paramètre Inline l'adresse de la routine à appeler et se comporte comme un JUMP. Selon le contenu de l'accumulateur, le vecteur en $BE9E permet d'activer (A=0) ou d'inhiber (A=$FF) le mécanisme de sauvegarde celui-ci étant actif par défaut, il est fortement conseillé de ne pas changer cet état.

Entrer-Firmware DOIT être utilisé pour appeler les vecteurs du Firmware ou toute routine utilisant les registres auxiliaires ou le registre IY. Il ne faut JAMAIS utiliser Entrer-Firmware pour appeler l'une des routines suivantes :

- Entrer-Firmware elle-même.
- Fixer la sauvegarde des registres ($BE9E).
- Les routines standard du BIOS.

Exemple :

Pour fixer le mode d'écran 0, on utilise le vecteur $BC0E (Scr-Set-Mode) en plaçant dans l'accumulateur le numéro du mode désiré.

En Assembleur il suffit de faire :

```
$3E/$01/        { Ld   A,$01    ;Mode 1}
$CD/>$BC0E      { Call Scr-Set-Mode }
```

mais avec CP/M 2.2, il faut faire :

```
Inline (
 $3E/$01/        { Ld   A,$01     ;Mode 1}
 $CD/>$BE9B/     { Call Entrer-Firmware }
 $BC0E         {DW Scr-Set-Mode   }
)
```

L'utilisation des autres vecteurs d'extension du BIOS doit être faite avec précaution car leurs actions peuvent interférer avec CP/M.

Dans l'exemple suivant, la procédure LogDrive permet de sélectionner le lecteur de travail, elle teste d'abord l'état du lecteur choisi avec le vecteur d'extension n° 8, 'Lire état des disques'. Si il n'y a pas de lecteur ou si celui-ci n'est pas prêt (*ready*), la routine ne fait rien, sinon le lecteur est sélectionné avec la fonction BDOS 14. Le vecteur d'extension N° 8 est situé en $BE95 et reçoit dans A le numéro du lecteur choisi (A :0, B :1). En résultat, si CARRY=1, A contient l'état du disque ou si CARRY=0, HL contient l'adresse d'un buffer dont le deuxième octet contient l'état du disque. Le bit 5 de l'octet d'état est à 1 si le disque est prêt, si oui, on appelle la fonction BDOS 14 pour sélectionner le disque par CP/M.

```
PROCEDURE LogDrive (Lecteur :Byte);
Var Etat : Byte;
Begin
 Inline (
  $3A/>Lecteur/ { Ld   A,(Lecteur)}
  $CD/>$BE9B/   { Call Entrer-Firmware }
  $BE95/      {DW LireEtat       }
  $38/2/        { Jr   C,L00     ;si CY=1, A=résultat}
  $23/          { Inc  HL        ;sinon (HL+1)=résultat}
  $7E/          { Ld   A,(HL)    }
  $32/>Etat  {L00 :Ld   (Etat),A ;resultat dans Etat}
 );
 If (Etat and 32)<>0 Then          {si lecteur prt...}
  Begin
   Bdos (13);                      {Réinitialise disques}
   Bdos (14,Lecteur)               {et sélection du disque choisi}
  End                              {si lecteur pas prt, ne fait rien}
End;
```

Accès à la mémoire vidéo

Comme nous l'avons vu pour l'organisation mémoire du CP/M 2.2 la mémoire vidéo occupe la zone d'adresses de $C000 à $FFFF mais c'est la ROM BIOS qui se trouve normalement commutée à cette adresse. Néanmoins, pour des raisons matérielles, l'écriture dans cette zone se fera toujours dans la mémoire vidéo.

Exemple :

La procédure suivante remplit l'écran avec la valeur de MOTIF.

```
PROCEDURE RemplitEcran (Motif : Byte);
Var adresse : Integer;
Begin
 For adresse :=$C000 To $FFFF Do Mem [adresse] :=Motif
End;
```

Pour lire la mémoire vidéo, il faut déconnecter la ROM BIOS, lire la mémoire vidéo et reconnecter la ROM. Il y a plusieurs méthodes possibles, mais la ROM ne peut être déconnectée pendant l'exécution d'instructions du TURBO PASCAL, il faut faire la lecture des octets en langage machine.

Exemples :

Dans cet exemple, la fonction LectureVideo effectue la déconnection de la ROM de $C000 à $FFFF, sauve le précédent état de la ROM, effectue la lecture d'un octet de la RAM vidéo puis restaure l'ancien état de la ROM. La procédure est appelée en utilisant Entrer-Firmware.

```
FUNCTION LectureVideo (Vadr : Integer) : Byte;
Var Octet : Byte; { contiendra le résultat }

PROCEDURE Lecture;
```

```
Begin
 Inline (
  $CD/$B903/     { Call KL-U-ROM-Disable;déconnecte ROM de $C000 à $FFFF}
  $F5/           { Push AF               ;sauve ancien état des ROM}
  $2A/>Vadr/     { Ld   HL,(Vadr)        ;HL=adresse dans RAM vidéo}
  $11/>Octet/    { Ld   DE,Octet         ;DE=adresse variable OCTET}
  $ED/$A0/       { Ldi                   ;transfert RAM vid{o--> OCTET}
  $F1/           { Pop  AF               ;récupère état}
  $CD/$B90C      { Call KL-ROM-Restore   ;restaure ancien état}
 )
End;

Begin
 Inline (
  $CD/$BE9B/     { Call Entrer-Firmware ;Pour appeler routine }
  >Lecture     {DW Lecture              ;Adresse de la routine }
 );
 LectureVideo :=Octet
End;
```

On peut aussi utiliser la routine Firmware KERNEL-LDIR qui transfère plusieurs octets avec toutes les ROM déconnectées de la même façon que l'instruction LDIR du Z 80.

```
PROCEDURE Transfert (Depart,Dest,Nombre : Integer);
Begin
 Inline (
  $2A/>Depart/      { Ld   HL,(depart) ;Adresse départ}
  $ED/$5B/>Dest/    { Ld   DE,(dest)   ;Adresse destination}
  $ED/$4B/>Nombre/  { Ld   BC,(Nombre) ;Nombre d'octets}
  $CD/$BE9B/        { Call Entrer-Firmware }
  $B91B           {DW KL-LDIR          }
 )
End;
```

Cette procédure transfère NOMBRE octets de l'adresse DEPART à l'adresse DEST, la lecture se fait toujours dans la RAM, on peut travailler dans les 64 Ko disponibles. Cette procédure peut, par exemple, être utilisée pour sauver temporairement une partie de l'écran dans un buffer et la restituer ensuite.

Dans l'exemple suivant, la procédure SauveLigne1 stocke la première ligne de texte dans la variable Buffer et la procédure RemetLigne1 replace le contenu de Buffer sur l'écran. Les deux routines ne fonctionnent correctement que si l'écran n'a jamais été scrollé auparavant.

```
{
 Une ligne de texte est large de 80 octets et haute de 8.
 BUFFER est une variable Array [1..8,1..80] Of Byte qui contiendra les
 octets correspondants de la mémoire vidéo.
}

PROCEDURE SauveLigne1;
Var adresse : Integer;
    i       : Byte;
Begin
 adresse :=$C000;         {Adresse de l'octet le plus à gauche de la ...}
                         {... 1ère ligne vidéo.}
 For i :=1 to 8 do        {Pour 8 lignes vidéo}
  Begin
   Transfert (adresse,Addr(Buffer[i,1]),80); {transfère 80 octets}
   Adresse :=Adresse+$800                    {descend d'une ligne vidéo}
  End
End;

PROCEDURE RemetLigne1;
Var adresse : Integer;
    i       : Byte;
Begin
 adresse :=$C000;         {Adresse de l'octet le plus à gauche de la ...}
                         {... 1ère ligne vidéo.}
 For i :=1 to 8 do        {Pour 8 lignes vidéo}
  Begin
   Transfert (Addr(Buffer[i,1]),adresse,80); {transfère 80 octets}
   Adresse :=Adresse+$800                    {descend d'une ligne vidéo}
  End
End;
```

Nous pourrions aussi agir sur le contenu du Buffer avant de restaurer la ligne sur l'écran, par exemple en la mettant en vidéo

inverse (inverser les couleurs de chaque point), il suffit d'inverser tous les bits si l'écran est en 80 colonnes :

```
PROCEDURE InverseLigne1;
VAR i,j : Byte;
Begin
 For i :=1 to 8 Do
  For j :=1 to 80 Do Buffer [i,j] :=Not Buffer [i,j]
End;
```

PARTICULARITES DU CP/M PLUS

Accès au BIOS

Les vecteurs standard du BIOS se trouvent dans la zone jamais commutée de la RAM mais la plupart de ceux-ci ne peuvent fonctionner qu'avec la banque 0 activée, leur utilisation à partir d'un programme en banque 1 se traduit donc généralement par la perte de contrôle de l'ordinateur. Pour accéder au BIOS, il faut utiliser la fonction BDOS n° 50 (Appel direct du BIOS) qui effectue le passage des paramètres et la commutation des RAM. Les vecteurs d'extension du CP/M 2.2 ne sont pas installés sous CP/M Plus.

Accès au Firmware (UserF)

Les vecteurs du Firmware Amstrad se trouvent en banque 0, pour y accéder depuis un programme en banque 1, il faut utiliser la BIOS N° 30, UserF, située en $FC5A qui reçoit en argument Inline l'adresse de la routine en banque 0 à appeler. Son utilisation est donc similaire à Entrer-Firmware du CP/M 2.2, seule l'adresse de la fonction change.

UserF connecte la banque 0, appelle la routine à l'adresse en paramètre et reconnecte la banque 1.

Exemple :

En reprenant l'exemple du CP/M 2.2, voici comment utiliser le vecteur du Firmware Scr-Set-Mode :

```
Inline (
 $3E/$01/     { Ld   A,$01    ;Mode 1}
 $CD/>$FC5A/ { Call UserF    }
 $BC0E      {DW Scr-Set-Mode }
)
```

Accès à la mémoire vidéo et à la banque 0

Du fait de la structure de la mémoire, on ne peut à partir de la banque 1 accéder directement à la mémoire vidéo comme sous CP/M 2.2. Néanmoins, le haut de la TPA à partir de $C000 étant commun aux deux banques de RAM, on peut y implanter des routines qui seront appelées à partir de la banque 1 avec UserF et qui travailleront ainsi avec la banque 0. Les routines seront par exemple créées avec un Assembleur et chargées depuis le programme en TURBO PASCAL en haut de la TPA. (Voir par exemple le programme d'utilisation des lutins graphiques à la fin de cet ouvrage.)

Exemple :

```
{
 A partir de $F000 est implantée une routine travaillant sur la mémoire
 vidéo. Elle a été préalablement chargée en mémoire par le programme
 (voir 'Manipuler les fichiers binaires'). La routine remplit l'écran
 avec le motif binaire stocké dans l'accumulateur.

  F000 : 21 00 40  Debut : Ld   HL,$4000 ;début RAM vidéo
  F003 : 11 01 40          Ld   DE,$4001
  F006 : 01 FF 3F          Ld   BC,$3FFF ;nombre de tours pour 16 KO.
  F009 : 77                Ld   (HL),A   ;1er remplissage
```

```
  F00A : ED B0            Ldir          ;remplit
  F00C : C9               Ret           ;fin !
}

PROCEDURE RemplitEcran (Motif : Byte);
Begin
 Inline (
  $3A/>Motif/    { Ld   A,(Motif);motif dans l'accumulateur}
  $CD/>$FC5A/    { Call UserF    ;prépare pour banque 0}
  $F000        {DW Debut         }
 )
End;
```

Le paramètre peut aussi être transmis en déclarant la routine EXTERNAL, elle doit alors retirer le paramètre elle-même.

```
  F000 : E1        Debut : Pop  HL       ;retire adresse de retour
  F001 : C1              Pop  BC       ;retire paramètre Motif (dans C)
  F002 : 79              Ld   A,C      ;paramètre dans A
  F003 : E5              Push HL       ;replace adresse de retour
  F004 : 21 00 40        Ld   HL,$4000 ;début RAM vidéo
  F007 : 11 01 40        Ld   DE,$4001
  F00A : 01 FF 3F        Ld   BC,$3FFF ;nombre de tours
  F00D : 77              Ld   (HL),A   ;1er remplissage
  F00E : ED B0           Ldir          ;remplit
  F010 : C9              Ret           ;fin !

PROCEDURE RemplitEcran (Motif : Byte); EXTERNAL $F000;
```

Il y a aussi plusieurs fonctions du BIOS du CP/M Plus qui permettent de commuter directement les banques et d'effectuer des transferts d'octets entre les deux banques.

La fonction BIOS n° 25 (MOVE) en $FC4B permet d'effectuer une copie de bloc mémoire de la même façon que LDIR mais en inversant le rôle de DE et HL. DE contient l'adresse de départ, HL l'adresse de destination et BC le nombre d'octets à transférer.

La fonction BIOS n° 29 (XMOVE) en $FC57 permet de sélectionner les banques de départ et de destination pour la fonction MOVE décrite ci-dessus. B contient le numéro de la banque destination et C le numéro de la banque de départ.

Ces deux fonctions ne peuvent être appelées directement, il faut utiliser la fonction BDOS n° 50 (Appel direct du BIOS).

On peut avec ces deux fonctions créer une procédure générale de transfert entre les banques. Différentes versions de cette procédure seront très souvent utilisées dans les programmes de cet ouvrage.

```
PROCEDURE Transfert (BanqueD,BanqueF : Byte; AdrD,AdrF,Nombre : Integer);
{
 Transfère 'Nombre' octets à partir de l'adresse 'AdrD' en 'BanqueD'
 vers l'adresse 'AdrF' en 'BanqueF'.
}
Type Parametres = Record                { Bloc de paramètres }
                   Fon,                 { Num{ro de fonction BIOS }
                   Accu      : Byte;    { Contenu des registres à passer}
                   BC,DE,HL  : Integer
                  End;
Var BiosPb : Parametres;

Begin
 With BiosPb Do
  Begin
   Fon :=29;                        { BIOS XMOVE }
   BC :=BanqueF*256+BanqueD;        { B=destination, C=départ }
   Bdos (50,Addr (BiosPb));         { Exécute XMOVE }

   Fon :=25;                        { BIOS MOVE }
   BC :=Nombre;                     { Charge registres }
   DE :=AdrD;
   HL :=AdrF;
   Bdos (50,Addr(BiosPb))           { Exécute MOVE }

  End {with}
End;
```

Nous pouvons à l'aide de cette fonction de transfert reproduire les exemples de manipulation de l'écran du CP/M 2.2 :

Exemples :

Lecture d'un octet de la RAM vidéo ou de la banque 0.

```
FUNCTION LectureVideo (Vadr : Integer) : Byte;
Var Octet : Byte; {contiendra le r{sultat}
Begin
 Transfert (0,1,Vadr,Addr (Octet),1);
 LectureVideo :=Octet
End;
```

Dans l'exemple suivant, la procédure SauveLigne1 stocke la première ligne de texte dans la variable Buffer et la procédure RemetLigne1 replace le contenu du Buffer sur l'écran. Les deux routines ne fonctionnent correctement que si l'écran n'a jamais été scrollé auparavant.

```
{
 Une ligne de texte est large de 80 octets et haute de 8.
 BUFFER est une variable Array [1..8,1..80] Of Byte qui contiendra les
 octets correspondants de la mémoire vidéo.
}

PROCEDURE SauveLigne1;
Var adresse : Integer;
    i       : Byte;
Begin
 adresse :=$4000;          {Adresse de l'octet le plus à gauche de la ...}
                           {... lère ligne vidéo.}
 For i :=1 to 8 do         {Pour 8 lignes vidéo}
  Begin
   Transfert (0,1,adresse,Addr(Buffer [i,1]),80 ); {transfère 80 octets}
   Adresse :=Adresse+$800                    {descend d'une ligne vidéo}
  End
End;

PROCEDURE RemetLigne1;
Var adresse : Integer;
    i       : Byte;
Begin
 adresse :=$4000;          {Adresse de l'octet le plus à gauche de la ...}
```

```
                        {... 1ère ligne vidéo.}
 For i :=1 to 8 do      {Pour 8 lignes vidéo}
  Begin
   Transfert (1,0,Addr(Buffer[i,1]),adresse,80); {transfère 80 octets}
   Adresse :=Adresse+$800                        {descend d'une ligne vidéo}
  End
End;
```

CHAPITRE 3

LE COMPILATEUR TURBO PASCAL

Le compilateur est un excellent outil de programmation en Pascal, on appréciera particulièrement l'éditeur très complet et la possibilité de redéfinir toutes les commandes clavier ainsi que les codes de la console. Dans ce chapitre sont présentés des 'trucs et astuces' facilitant encore plus la vie du programmeur.

Lors de l'écriture d'un gros programme et surtout avec CP/M2.2, des gros problèmes de place mémoire se posent et il est fortement conseillé après la mise au point de passer le texte au crible et d'éliminer toutes les sources de gaspillage. Vous pourrez vous rendre compte à chaque phase du 'dégraissage' de la place gagnée en comparant le nombre d'octets FREE affichés par le TURBO PASCAL à la fin de chaque compilation.

GAGNER DE LA PLACE DANS L'INSTRUCTION WRITE

La plus grande source de gaspillage vient sûrement de l'instruction WRITE(LN). Dans le code généré par le compilateur, chaque virgule dans l'argument de WRITE indique que l'expression suivante est une référence extérieure et consomme au minimum 6 octets, les éléments déclarés littéralement, par exemple une chaîne de caractères sont directement codés en ASCII (1 octet par caractère).

Exemples :

Pour visualiser le code de votre programme, compilez-le avec l'option CHN, le fichier programme ne comportera pas la bibliothèque du TURBO PASCAL et ne contient donc que le programme, puis vous pouvez l'examiner avec les utilitaires du CP/M, par exemple DUMP, DDT ou SID.

La routine $17BA de la bibliothèque du TURBO PASCAL affiche la chaîne de caractères située en mémoire après le Call, la chaîne commence par un octet indiquant le nombre de caractères.

```
{
 Fon est une FUNCTION quelconque de type CHAR, dans cet exemple FON
est située à l'adresse réelle $20FF.
}
```

Write ('aaa',Fon,'bbb') est codé :

```
                ;Affiche la chaîne 'aaa'
CD BA 17            Call $17BA      ;routine d'affichage dans bibliothèque
03                  Defb $03        ;nombre de caractères 'a'
61 61 61            Defw 'aaa'      ;1ère expression chaîne

CD FF 20            Call Fon        ;Appelle la fonction, résultat dans HL
CD 22 17            Call $1722      ;Sort le caractère dans HL

                ;Affiche la chaîne 'bbb'
CD BA 17            Call $17BA
03                  Defb $03
62 62 62            Defw 'bbb'
```

TOTAL : 20 octets.

L'argument FON utilise 6 octets (CD FF 20 CD 22 17) plus 4 autres pour afficher le reste de l'instruction (CD BA 17 03).

Write ('aaa',#7,'bbb') est codé :

```
                ;Affiche la chaîne 'aaa'
CD BA 17            Call $17BA      ;routine d'affichage dans bibliothèque
03                  Defb $03        ;nombre de caractères 'a'
61 61 61            Defw 'aaa'      ;1ère expression chaîne
21 07 00            Ld   HL,$0007   ;Numéro caractère (7)
CD 22 17            Call $1722      ;Sort le caractère dans HL

                ;Affiche la chaîne 'bbb'
CD BA 17            Call $17BA              (a)
03                  Defb $03                (b)
62 62 62            Defw 'bbb'
```

TOTAL : 20 octets.

Nous voyons dans ces deux exemples que l'utilisation d'un caractère non référencé directement ou placé entre virgules nécessite

6 octets pour l'affichage sans compter les octets nécessaires pour afficher le reste de la chaîne (lignes (a) et (b)).

En reécrivant l'exemple n° 2 :

Write ('aaa'#7'bbb') est codé :

```
CD BA 17        Call $17BA          ;routine d'affichage dans bibliothèque
07              Defb $07            ;nombre de caractères
61 61 61        Defw 'aaa'#7'bbb'   ;expression chaîne complète
07
62 62 62
```

TOTAL : 11 octets soit un gain de 9 octets et une plus grande rapidité d'exécution.

Il ne faut donc pas écrire de virgules entre des caractères et des codes de contrôle, seules les variables, constantes et fonctions doivent être délimitées par des virgules. De plus il faut éviter d'utiliser WRITELN lorsque WRITE suffit car WRITELN consomme 3 octets supplémentaires.

Exemples :

```
WRITE (#7,'Bip')
```

doit être écrit :

```
WRITE (#7'Bip !').
```

```
WRITE ('Ecriture ',#27,'p','inverse',#27,'q')
```

doit être écrit :

```
WRITE ('Ecriture '#27'pinverse'#27'q').
```

Sous CP/M Plus, il est affiché :

```
'Ecriture inverse' ('inverse' en mode inverse)
```

```
WRITE ('De m',#$E6,'me avec les accentu',#$E1,'es')
```

doit être écrit :

```
WRITE ('De m'#$E6'me avec les accentu'#$E1'es').
```

Sous CP/M Plus, il est affiché :

```
'De même avec les accentuées'
```

Il peut aussi être avantageux de placer des codes de positionnement du curseur pour éviter d'écrire plusieurs instructions WRITE.

Exemple :

```
WRITELN ('Société UNTEL');
WRITE   (' Négociant   ');
```

peut s'écrire :

```
WRITE ('Société UNTEL'^J^M' Négociant ');
```

^J est le code de nouvelle ligne, ^M est le code du retour chariot.

UTILISATION DES CONSTANTES

Toute variable qui n'est affectée qu'une seule fois au cours du programme peut être déclarée comme une constante.

En effet, supposons qu'un programme utilise un tableau de chaînes qui est initialisé au début du programme et qui n'est ensuite jamais modifié. Il faut déclarer le tableau avec :

```
VAR Tableau : Array [1..x] Of String [y];
```

puis ensuite affecter les éléments du tableau par :

```
Tableau [1] :='.....'
Tableau [2] :='xxxxx'
etc
```

ce qui consomme doublement la mémoire, lors de la déclaration VAR, x * y octets sont nécessaires et chaque affectation consomme de la mémoire, à la fin, chaque élément du tableau se trouvera doublement en mémoire : dans le tableau et dans les affectations.

Une première solution serait de placer le contenu du tableau dans un fichier et de recharger celui-ci en mémoire, ce qui permet de modifier le tableau sans reprendre la liste du programme mais en modifiant directement le fichier. Mais cela nécessite un travail de programmation supplémentaire.

Exemple :

```
TYPE Tableau : Array [1..7] Of String [9];

VAR  Menus    : Tableau;
     MenusFic : File Of Tableau;

PROCEDURE ChargeMenus;
{
 Charge le contenu de Tableau.
}
Begin
 Assign (MenusFic,'NOMFIC.DAT');
 Reset (MenusFic);
 Read (MenusFic,Menus);    { Le tableau est entièrement chargé }
 Close (MenusFic)
End;
```

Le TURBO PASCAL apporte une solution élégante au problème avec les CONSTANTES et surtout les CONSTANTES AVEC TYPE.

Les constantes sans type ne sont qu'une facilité pour l'écriture des programmes, elles sont déclarées avec CONST et leur valeur est incluse à chaque fois dans le programme.

Exemple :

```
CONST Mot = 'Bonjour';

Write (Mot,'Comment allez-vous ?',Mot,' chez-vous !');
```

Il sera codé deux fois le mot 'Bonjour' dans l'expression WRITE. Les fonctions ADDR et SIZEOF appliquées aux constantes sans type donnent des résultats aberrants.

Les constantes avec type se comportent par contre comme des variables du même type, on peut même les réaffecter dans le programme bien que ce ne soit pas leur fonction première. Elles n'occupent alors la mémoire qu'une seule fois, ce sont elles qu'il faut utiliser pour gagner de la place mémoire. En effet, dans l'exemple suivant, le tableau JOURS est déclaré et affecté en une seule fois, la place mémoire utilisée ne contient que les éléments du tableau. Les fonctions ADDR et SIZEOF fonctionnent parfaitement avec les constantes avec type et on peut aussi les utiliser en langage machine avec Inline.

Exemple :

```
CONST Jours : Array [1..7] Of String [8]=
('Lundi   ','Mardi   ','Mercredi','Jeudi   ','Vendredi','Samedi  ',
'Dimanche');
```

LES ADRESSES DE DEBUT ET DE FIN DE COMPILATION

Lorsque la compilation est effectuée sur le disque, deux lignes sont affichées dans le Menu des Options, elles indiquent les adresses des bornes inférieures et supérieures de la zone mémoire qui sera utilisée par le programme. Pour chaque ligne la première

adresse indique la valeur actuelle et la deuxième indique la borne à ne pas dépasser.

Le programme compilé est constitué de la bibliothèque du TURBO PASCAL qui prend environ 8 Ko des adresses $0100 à $20E1, elle est identique pour chaque programme et contient des routines de traitement, ensuite vient le code du programme de l'utilisateur, celui-ci fait appel aux routines de la bibliothèque. A partir de la dernière adresse disponible sont placées les variables du programme dans le sens des adresses décroissantes.

Start-Address

La première ligne affichée concerne l'adresse de départ du code (Start-Address), l'adresse actuelle est normalement $20E2 et la valeur minimale est aussi $20E2, ce qui indique que le code commence en $20E2 et que la bibliothèque du TURBO PASCAL se termine en $20E1, donc $20E2 est la première adresse libre. On ne peut mettre une valeur plus petite que $20E2 car la bibliothèque serait en partie 'écrasée', mais on peut augmenter la valeur, cela permet de réserver de la place mémoire avant le code du programme pour stocker des variables communes entre deux programmes chaînés par exemple.

Exemple :

Soient PROG-A.COM et PROG-B.COM deux programmes TURBO PASCAL compilés avec une Start-Address égale à $2100. La variable Commune est déclarée en dehors de la zone réservée au programme.

```
PROGRAM PROG-A;
VAR Commune : Byte ABSOLUTE $20F0;
    Fichier : File;
Begin
 ClrScr;
{ Affecter COMMUNE }
 Repeat
```

```
  Write ('Entrer un chiffre de 1 à 10 ?');
  Readln (Commune);
 Until Commune in [1..10];
{ Appel de PROG-B }
 Assign (Fic,'PROG-2.COM');
 Chain (Fic)
End.

PROGRAM PROG-B;
VAR Commune : Byte ABSOLUTE $20F0;
Begin
 ClrScr;
 Write ('Programme B --- Vous avez tapé ',Commune);
End.
```

Si l'on exécute PROG-A, celui-ci affecte la variable Commune placée grâce à l'instruction ABSOLUTE à l'adresse $20F0 qui est protégée car la Start-Address est de $2100. Puis le programme PROG-B est enchaîné, dans celui-ci la même variable Commune (ou tout autre nom) est affectée par ABSOLUTE en $2010 et si l'on demande sa valeur on retrouve la même valeur que dans PROG-A.

End-Address

La deuxième ligne affichée concerne la dernière adresse disponible pour le programme, cela correspond à la commande MEMORY en BASIC. sous CP/M Plus, la valeur normale est $F342 et la valeur maxi est $F606. Il y a donc apparemment de la place à gagner (708 octets) mais cette zone est normalement utilisée par le TURBO PASCAL pour y placer un petit programme lanceur (*loader*) qui permet de revenir au compilateur après l'exécution d'un programme COM à partir de celui-ci (commande EXECUTE).

Néanmoins, si l'on n'a pas besoin de cette facilité, nous pouvons récupérer cet emplacement pour le programme, il faut alors placer la End-Address à $F606, si l'on va au-delà, une partie du BDOS sera écrasé.

Lorsqu'il est nécessaire de charger des routines externes (voir 'Manipulation des fichiers binaires') en haut de la TPA, il faudra

modifier 'End-Address' pour protéger les routines, dans ce cas on peut aussi récupérer la place du Loader.

Certaines routines du Firmware ont besoin de paramètres en mémoire vive mais le Firmware utilise la banque 0 alors que CP/M Plus, donc le programme d'application, se trouve dans la banque 1, il faut alors placer les paramètres dans la zone jamais commutée de la mémoire (adresses $C000 à $FFFF) pour les rendre accessibles. En Turbo Pascal, les variables étant codées à partir du haut de la mémoire (environ $F300), les premières variables déclarées dans le texte source se trouvent donc en zone non commutée et pourront donc être utilisées en paramètres, dans les cas extrêmes il faudra peut-être diminuer End-Address pour réserver de la place pour des variables Absolutes placées à la suite du programme en zone non commutée. Si le nombre d'octets nécessaires est très petit on peut aussi se servir de certaines adresses non utilisées dans la partie réservée au CP/M de la zone non commutée, par exemple les emplacements des fonctions BDOS 31 et 32 qui sont non implantées (adresses $FC5D à $FC62).

VERIFIER LE TEXTE SOURCE...

Quel programmeur n'est pas furieux quand lors d'une longue et pénible compilation sur disque, arrivé vers la fin du texte source, le système affiche laconiquement 'Error Found in ...', il faut corriger et recompiler entièrement et au bout d'une heure le programme n'est toujours pas au point.

Il est cependant possible de vérifier la justesse du programme en utilisant la fonction FIND RUN-TIME ERROR (Trouver la position de l'erreur à l'exécution). Celle-ci permet de retrouver l'emplacement d'une erreur survenue pendant l'exécution d'un programme, il faut indiquer la valeur du PC (compteur de programme) affichée en même temps que le message d'erreur, le texte source est alors lu et une compilation 'dans le vide' est effectuée, c'est-à-dire que le texte est compilé mais le code objet n'est stocké nulle part. En détournant l'usage de la fonction, il est ainsi possible de vérifier très rapidement un texte car la compilation ne se fait pas sur le

disque, pas même en mémoire, il suffit d'indiquer une adresse très haute, au-delà de la limite possible (End-Address), par exemple $FFFF. Le gain de temps est très appréciable pour les gros programmes qui doivent être obligatoirement compilés sur disque.

Si le texte comporte une erreur, elle est détectée et le message adéquat est affiché sinon la recherche continue jusqu'à la fin du texte et le compilateur affiche alors 'Run-time error position not found' (Position de l'erreur à l'exécution non trouvée), la véritable compilation peut alors être effectuée.

CHAPITRE 4

MANIPULATION DES FICHIERS BINAIRES

CHARGEMENT DES FICHIERS BINAIRES

Dans de nombreux cas, il peut être nécessaire de placer des routines en code machine Z 80 à une adresse déterminée, par exemple si l'on veut accéder à la mémoire vidéo sous CP/M Plus (voir chapitre à ce sujet). Ces routines sont appelées EXTERNES et on peut les utiliser avec la commande EXTERNAL décrite précédemment ou directement avec un CALL dans une instruction INLINE. Comme il est impossible de forcer l'emplacement d'une procédure en TURBO PASCAL, il faut créer les routines avec un Assembleur et placer le code obtenu dans un fichier qui sera chargé par le programme en TURBO PASCAL à l'adresse souhaitée.

La même technique sera utilisée pour charger des images graphiques ou tout autre fichier binaire, c'est l'équivalent en BASIC :

```
LOAD fichier-binaire, adresse
```

Si le fichier est créé sous CP/M (par exemple avec un assembleur compatible CP/M), il n'y a pas d'en-tête de fichier, le premier octet du fichier correspond au premier octet de code. Dans les autres cas, avec AMSDOS (image d'écran issue d'un logiciel graphique ou programme binaire par exemple), le fichier comporte une en-tête préliminaire de 128 octets (soit la taille d'un enregistrement sous CP/M) contenant des informations inutiles dans le cadre du TURBO PASCAL. Pour charger correctement de tels fichiers, il faudra 'sauter' cette en-tête. Les fichiers ASCII AMSDOS n'ont cependant pas d'en-tête et sont donc compatibles directement avec CP/M, ainsi les textes des programmes du TURBO PASCAL peuvent être utilisés sans problèmes avec un traitement de texte.

Si le fichier doit s'implanter dans la TPA, il faut réserver de la place en mémoire pour que le TURBO PASCAL n'écrase pas les données, il suffit de diminuer la 'End-Address' de compilation, on peut aussi récupérer la place du Loader du TURBO PASCAL (voir Chapitre 3 : 'Le compilateur TURBO PASCAL').

Le chargement est très simple, on affecte une variable Byte (par exemple) absolue à l'adresse de départ, le fichier est associé à une

variable Fichier sans type (File) et la lecture s'effectue avec BlockRead. Si le fichier comporte une en-tête AMSDOS, celle-ci est 'sautée' avec l'instruction SEEK (Fichier,1) qui positionne au deuxième enregistrement (le premier a le numéro 0).

Exemples :

Voici une procédure d'usage général pour charger des fichiers binaires :

```
{
 XXX.BIN est un fichier disque qui doit être chargé à l'adresse $9000.
 Le fichier fait 9 enregistrements et n'a pas d'entête.
 Le programme doit être compilé sur disque et End-Address doit être
 inférieure à $9000.
}

PROCEDURE ChargeBin;
{
 Charge XXX.BIN
}
Var Fic   : File;
    Debut : Byte ABSOLUTE $9000;

Begin
 Assign (Fic,'XXX.BIN');
 Reset (Fic);
 BlockRead (Fic,Debut,9);
 Close (Fic)
End;
```

Le programme de démonstration des lutins graphiques dans cet ouvrage utilise la même méthode de chargement.

Sous CP/M 2.2, on peut charger des images écran très simplement :

```
{
 Une image fait 16 KO soit 128 enregistrements de 128 octets, le fichier
 comporte une entête.
```

```
}

Type Nomfichier = String [14];

PROCEDURE ChargeImage (Nom : Nomfichier);
Var Fic  : File;
    Debut : Byte ABSOLUTE $C000;  {d{but de la RAM Vidéo}

Begin
 Assign (Fic,Nom);
 Reset (Fic);
 Seek (Fic,1);                { Sur véritable début du fichier, }
                              { seulement pour les fichiers AMSDOS. }
 BlockRead (Fic,Debut,128); { Charge 16 KO dans la RAM Vidéo. }
 Close (Fic)
End;
```

CONVERSION DE FICHIERS AMSDOS/CP/M

Dans certains cas, il peut être nécessaire de convertir un fichier AMSDOS avec en-tête en fichier CP/M sans en-tête et vice versa. Cela peut être utile en dehors de l'utilisaton du TURBO PASCAL, par exemple pour créer des programmes exécutables COM avec un assembleur AMSDOS classique ou pour désassembler des fichiers COM sous AMSDOS. C'est d'autant plus utile que les outils (assembleurs, désassembleurs, moniteurs) fournis avec CP/M par Amstrad fonctionnent avec le code du microprocesseur 8080, au lieu du Z 80 qui est installé dans la machine !

CONVERSION AMSDOS --> CPM

C'est la plus simple, si l'on charge en mémoire un fichier avec en-tête sous CP/M, l'en-tête sera aussi chargée, il suffit alors de reécrire sur disque le contenu de la mémoire sans l'en-tête.

On pourrait aussi écrire un programme qui effectuerait automatiquement l'opération, mais on peut utiliser les utilitaires standard du CP/M comme nous allons le voir. Il y a deux méthodes pour chaque version du CP/M, on appelle le fichier AMSDOS 'fichier1' et le nouveau fichier CP/M 'fichier2'. Toutes deux utilisent des utilitaires CP/M similaires et le principe est identique, le fichier AMSDOS est chargé entièrement à partir de l'adresse $0100 et l'on reécrit sur disque le contenu de la mémoire à partir de $0180.

Note : Les indications entre crochets sont facultatives.

Sous CP/M 2.2

La façon de procéder est un peu complexe, on utilise DDT qui se trouve sur le disque système. DDT nous permet de charger le fichier AMSDOS en mémoire mais il ne peut faire la reécriture du fichier sur le disque, il faudra sortir de DDT et utiliser la commande interne SAVE qui permet de sauver dans un fichier le contenu de la mémoire par blocs de 256 octets et à partir de l'adresse $0100. Il faudra donc au préalable décaler avec DDT les octets du fichier de 128 octets, de l'adresse $0180 vers l'adresse $0100.

- Copier DDT sur le disque contenant le fichier de travail. Le disque doit être au format système.
- Charger DDT et le fichier avec 'DDT (nom du fichier1)'.
- DDT affiche alors l'adresse du prochain octet disponible (NEXT). C'est l'adresse du prochain octet libre après le fichier. Notez cette valeur.
- Vous pouvez examiner la mémoire en hexadécimal et en ASCII avec la commande 'D<adresse de début>'.
- Vous voyez alors le contenu de l'en-tête des octets $0100 à $017F et le premier octet de code à partir de l'adresse$0180.
- Il faut alors décaler le contenu de la mémoire avec la commande 'M180,(NEXT-1),100'.
- Sortir de DDT avec CONTROL C. Le fichier est toujours en mémoire.

- Calculer le nombre de blocs, le nombre d'octets est : NEXT-$0180 (en hexadécimal). Le nombre de blocs correspond à l'octet fort du résultat, plus 1 si l'octet faible est différent de 0.

Exemples :

```
NEXT=$4080. Nombre d'octets=$4080-$0180=$3F00. Octet faible=$00
            Nombre de blocs=$3F=63.

NEXT=$4085. Nombre d'octets=$4085-$0180=$3F05. Octet faible=$05
            Nombre de blocs=$3F+1=64.
```

Le nombre de blocs peut aussi être obtenu par :

Nombre d'octets (en décimal) divisé par 256, plus 1 si le reste est non nul.

- Entrer alors la commande :

SAVE (Nombre de blocs en décimal) (nom du fichier2) pour créer le nouveau fichier.

- Le fichier contient alors (Nombre de blocs)*2 enregistrements et cette valeur sera utile pour le chargement en Turbo Pascal.

Sous CP/M Plus

On peut se servir de SID qui est le 'debugger' fourni avec la machine.

- Charger SID et le fichier en tapant 'SID (nom du fichier1)'.
- Le fichier se charge alors, en-tête comprise, à partir de l'adresse $0100.

- SID affiche l'adresse du prochain octet disponible (NEXT), cette valeur peut aussi être obtenue avec la commande V. C'est l'adresse du premier octet libre après le fichier.
- Vous pouvez examiner la mémoire en hexadécimal et en ASCII avec la commande 'D<adresse de début><,adresse de fin>'.
- Vous voyez alors le contenu de l'en-tête des octets $0100 à $017F et le premier octet de code à partir de l'adresse $0180.
- Il suffit alors de sauver sur disque la zone de mémoire entre $0180 et NEXT-1 avec la commande 'W(nom de fichier2),$0180,(NEXT-1)'.
- Après l'écriture sur le disque, SID affiche la taille du fichier en enregistrements, cette valeur sera nécessaire pour le chargement en TURBO PASCAL pour déterminer la place à réserver ainsi que le nombre d'enregistrements à lire.
- Sortir de SID avec CONTROL-C.

CONVERSION CP/M --> AMSDOS

Il suffit de charger en mémoire le fichier CP/M et de le sauver par une commande SAVE sous BASIC, le fichier ainsi créé sera au format AMSDOS. Il n'est pas possible sous AMSDOS de charger un fichier sans en-tête, sauf s'il s'agit d'un fichier ASCII. Or, les fichiers ASCII utilisent le code CONTROL Z (#26) comme marque de fin de texte et un fichier de codes machine peut contenir ce code à plusieurs endroits différents. Il faudra écrire une petite routine en langage machine pour lire le fichier CP/M, la fin de la lecture se fera lorsque la fin physique du fichier sera rencontrée, c'est-à-dire lorsque le dernier enregistrement sera lu.

La routine suivante effectue le chargement de tous fichiers CP/M, elle est appelée à partir du BASIC par :

```
CALL &4000,@(nom-du-fichier),@(adresse-debut),@(dernier-octet)
```

Nom-du-fichier est une variable chaîne. Adresse-début et dernier-octet sont des entiers.

Il faut fournir le nom du fichier et l'adresse de début pour le chargement, la routine charge tous les blocs du disque affectés au fichier, le fichier peut en fait être plus court car un bloc est une unité indivisible de 1 Ko. La routine renvoie l'adresse du prochain octet libre après le dernier bloc chargé. La routine est relogeable en mémoire, elle a besoin d'un buffer de 2 Ko, celui-ci est placé par défaut dans la mémoire d'écran à l'adresse $C000 mais ceci peut être modifié.

```
                ;**************************************
                ;*      Conversion CP/M --> AMSDOS      *
                ;**************************************
                ;
                Buffer : EQU #C000          ;Buffer 2 KO modifiable.
                ;
                ;--------------------------
                ; Nom du fichier ?
                ;--------------------------
FE 03                   Cp   3              ;3 paramètres ?
C0                      Ret  NZ             ;Non, ne fait rien !
DD 6E 04                Ld   L,(IX+04)      ;adresse descripteur nom-fich.
DD 66 05                Ld   H,(IX+05)      ;dans HL
46                      Ld   B,(HL)         ;Longueur de la chaîne dans B
23                      Inc  HL             ;sur adresse de la chaîne
7E                      Ld   A,(HL)
23                      Inc  HL
66                      Ld   H,(HL)
6F                      Ld   L,A            ;HL=adresse de la chaîne
DD E5                   Push IX             ;sauve pointeur paramètres
                ;--------------------------
                ; Ouvre le fichier
                ;--------------------------
11 00 C0                Ld   DE,Buffer      ;adresse modifiable
CD 77 BC                Call #BC77          ;DISC IN OPEN.
                                            ;Ne charge pas d'entête AMSDOS.
DD E1                   Pop  IX             ;récupère pointeur paramètres
DD 6E 02                Ld   L,(IX+02)
DD 66 03                Ld   H,(IX+03)      ;HL=adresse variable adr-début
```

```
7E                  Ld   A,(HL)
23                  Inc  HL
66                  Ld   H,(HL)
6F                  Ld   L,A            ;HL=adresse chargement
DD E5               Push IX             ;sauve pointeur paramètres
                ;--------------------------
                ; Lecture octets
                ;--------------------------
CD 80 BC        Lire :   Call #BC80     ;DISC IN CHAR. Lit un caractère
38 04               Jr   C,Lire1        ;OK, continue.
                ;Erreur...
FE 0F               Cp   #0F            ;Fin physique du fichier ?
28 04               Jr   Z,Fin          ;Oui, fin.
                ;
77              Lire1 :  Ld   (HL),A     ;place en mémoire.
23                  Inc  HL             ;prochaine adresse
18 F3               Jr   Lire           ;boucle
                ;--------------------------
                ; Fin du traitement
                ;--------------------------
DD E1           Fin :    Pop  IX        ;récupère pointeur paramètres
DD 5E 00            Ld   E,(IX+00)
DD 56 01            Ld   D,(IX+01)     ;DE=adresse variable (dern.oct)
EB                  Ex   DE,HL         ;place HL dans la variable
73                  Ld   (HL),E
23                  Inc  HL
72                  Ld   (HL),D
C3 7A BC            Jp   #BC7A         ;DISC IN CLOSE et fin.
                ;
                    End
```

La routine AMSDOS DISC-IN-CHAR effectue la lecture octet par octet du fichier désigné, un buffer de 2 Ko est nécessaire pour son fonctionnement, l'adresse de celui-ci est fournie dans DE lors de l'ouverture du fichier par DISC-IN-OPEN. DISC-IN-CHAR place dans A le caractère lu, Carry est vrai s'il n'y a pas d'erreur, sinon Carry est faux et A contient un code d'erreur. Si A = #1A, la fin du texte a été rencontrée (pour les fichiers ASCII), si A = #0F, c'est la fin physique du fichier (fin du dernier bloc). Comme il faut lire aussi des fichiers binaires qui peuvent contenir l'octet #1A, il ne

faut pas tester ce caractère pour reconnaître la fin du fichier, mais il faut tester la fin physique du fichier.

Voici un exemple de programme BASIC permettant l'appel de la routine :

La routine est préalablement implantée en &4000 dans cet exemple.

```
10 DEFINT a-z
20 INPUT "Nom du fichier ?",nom$
30 INPUT "Adresse début  ?",adr
40 adrfin=0
50 CALL &4000,@nom$,@adr,@adrfin
60 END
```

La sauvegarde au format AMSDOS s'effectue par :

```
SAVE "NomFic",B,adr,adrfin-adr
```

CHAPITRE 5

PROBLEMES DE COMPATIBILITE

APPEL DU FIRMWARE

Comme nous l'avons vu, il y a pour chaque version du CP/M un vecteur différent pour l'utilisation du Firmware, Entrer-Firmware en $BE9B pour CP/M 2.2 et UserF en $FC5A pour CP/M Plus.

Exemple :

Sous CP/M 2.2, si l'on veut par exemple utiliser la routine Firmware en $BC0E, il faut écrire :

```
Inline (
 {...}
 $CD/>$BE9B/     { Call Entrer-Firmware }
 $BC0E/         {DW Adresse routine      }
 {...}
)
```

Sous CP/M Plus, on écrira :

```
Inline (
 {...}
 $CD/>$FC5A/     { Call UserF      }
 $BC0E/         {DW Adresse Routine }
 {...}
)
```

Pour chaque version du programme, il faudra changer l'adresse du vecteur d'appel, ce qui est très fastidieux car les instructions INLINE ne sont pas très lisibles. Heureusement, comme le mécanisme d'appel est similaire dans les deux versions, l'adresse est en paramètre Inline, on peut faciliter la conversion en créant une procédure commune FIRM qu'il suffit d'appeler à la place des vecteurs Entrer-Firmware et UserF, la procédure contiendra uniquement un JUMP vers le véritable vecteur, il suffira de changer ce Jump pour chaque version du programme.

Exemple :

```
PROCEDURE FIRM;
Begin
 Inline (
  $C3/          { Jump }
                { adresse à choisir selon CP/M utilisé : }
  $BE9B         { pour CP/M 2.2 }
  $FC5A         { pour CP/M Plus }
 )
End;

PROCEDURE Essai;
{
 Exemple d'utilisation.
}
Begin
 Inline (
  {...}
  $CD/>FIRM/    { Call Firm       ;Rien à modifier ici }

  $BC0E/      {DW Adresse routine }
  {...}
 )
End;
```

NOTE IMPORTANTE

Dans le reste de cet ouvrage, lorsqu'il le sera nécessaire, la routine FIRM sera utilisée dans les exemples de programmes bien qu'elle ne soit pas décrite à chaque fois.

LE JEU DE CARACTERES ET LES ATTRIBUTS VIDEO

Un programme écrit en TURBO PASCAL sans utiliser l'Assembleur peut normalement fonctionner avec les deux systèmes mais il y a néammoins incompatibilité des jeux de caractères affichables et de l'utilisation des attributs vidéo. En effet, sous CP/M 2.2 les caractères sont ceux normalement utilisés sur l'Amstrad, en BASIC par exemple, et les attributs s'utilisent avec les caractères de contrôle (codes 0 à 31) standard eux aussi. Sous CP/M Plus, les concepteurs ont installé un émulateur de terminal Zénith pour rendre les programmes facilement transposables sur d'autres systèmes, mais les caractères sont en partie différents, ils comportent les lettres accentuées par exemple, et les attributs vidéo s'utilisent avec des séquences de codes ESCAPE (code ESCAPE n° 27 suivi de paramètres).

LE JEU DE CARACTERES

Le jeu de caractères du CP/M Plus se révèle nettement plus intéressant que le jeu standard car il permet de réaliser facilement des programmes françisés ainsi que des encadrements divers (pour un tableau par exemple). Hélas, CP/M 2.2 n'a pas le même jeu, ce qui rend la conversion difficile.

Voici un utilitaire permettant d'utiliser sous CP/M 2.2 les mêmes caractères que CP/M Plus. Pour cela, le programme CreeMatrices crée d'abord un fichier contenant les matrices des caractères 0 à 255 du CP/M Plus, la police de caractères française étant choisie, les accents et autres signes français sont disponibles. Ce programme est inutile dans vos applications.

```
PROGRAM CreeMatrices;
{
 Crée le fichier MATRICES.DAT contenant les matrices des caractères du
CP/M Plus. Ce programme doit fonctionner sous CP/M Plus.
}
VAR Matrices : Array [0..255,1..8] Of Byte; { Recevra la table }
    Fic      : File;                        { Fichier disque }
    i,c      : Byte;                        { Divers compteurs }
```

```
    Adr      : Integer;                    { Pour calculer l'adresse
                                             dans la mémoire vidéo. }

PROCEDURE Transfert (AdrD,AdrF :Integer);
{
 Cette procédure transfère un octet de la mémoire vidéo (adresse AdrD en
 Banque 0) vers la variable Matrices (adresse AdrF en Banque 1), voir
 les 'Accès à la mémoire vidéo et à la banque 0 du CP/M Plus' pour plus
 de détails.
}
VAR BiosPb : Array [0..7] of Byte; { Pour appeler le BIOS par le BDOS }
Begin
 Inline (
  $3E/29/         { Ld   A,&29       ;Fonction Bios XMOVE}
  $32/>BiosPb/    { Ld   (Pb+00),A   ;Numéro}
  $21/>$0100/     { Ld   HL,$0100    ;H=banque destin. L=banque origine}
  $22/>BiosPb+2/  { Ld   (Pb+02),HL  ;Paramètre BC}
  $0E/50/         { Ld   C,&50       ;BDOS BiosCall}
  $11/>BiosPb/    { Ld   DE,BiosPb   ;DE=adresse table}
  $CD/>$05/       { Call BDOS        ;Exécute XMOVE}
                  {                  }
  $3E/25/         { Ld   A,&25       ;Fonction Bios MOVE}
  $32/>BiosPb/    { Ld   (Pb+00),A   ;Numéro}
  $21/>1/         { Ld   HL,$0001    ;Nombre d'octets}
  $22/>BiosPb+2/  { Ld   (Pb+02),HL  ;Paramètre BC}
  $2A/>AdrD/      { Ld   HL,(AdrD)   ;Adresse de depart}
  $22/>BiosPb+4/  { Ld   (Pb+04),HL  ;Paramètre DE}
  $2A/>AdrF/      { Ld   HL,(AdrF)   ;Adresse d'arrivee}
  $22/>BiosPb+6/  { Ld   (Pb+06),HL  ;Paramètre HL}
  $0E/50/         { Ld   C,&50       ;BDOS BiosCall}
  $11/>BiosPb/    { Ld   DE,BiosPb   ;DE=adresse table}
  $CD/>$05 )      { Call BDOS        ;Ex{cute MOVE}
End;

Begin
 Write (^['3B'^['2'#1);          { 80 colonnes et caractères français. }
 Assign (Fic,'MATRICES.DAT');    { Nomme le fichier. }
 ReWrite (Fic);                  { et l'ouvre en écriture. }
 For c :=0 To 255 Do              { Pour tout les caractères...}
  Begin
   Gotoxy (1,1);                 { Affiche le caractère en haut à gauche }
```

```
   If c<32 Then Write (^[);     { Pour les codes ASCII inf. à 32. }
   Write (Chr(c));

   Adr :=$4000;                  { Lecture directe de la mémoire d'écran
}
   For i :=1 to 8 Do             { Du haut vers le bas du caractère. }
    Begin                       { Transfert dans Matrices. }
     Ldir (Adr,Addr(Matrices[c,i]));
     Adr :=Adr+$800              { Une ligne au dessous. }
    End

  End;
 BlockWrite (Fic,Matrices,16); { Ecrit la table et ferme le fichier. }
 Close (Fic)
End.
```

Voici maintenant la procédure qui est à inclure dans vos programmes sous CP/M 2.2 et qui permet le chargement de ces nouvelles matrices. Elle fait appel à la routine Firmware 'TXT-Set-M-Table' qui reçoit dans DE le numéro du premier caractère à redéfinir, ici 0, et dans HL l'adresse où s'implante la nouvelle table. Cette table doit faire N*8 octets de long, N étant le nombre de caractères à redéfinir, ici 256. La table utilise 2 Ko de mémoire vive et il faudra en tenir compte. TXT-Set-M-Table remplit d'abord la nouvelle table avec les anciennes matrices, il faut donc l'appeler avant le chargement de la nouvelle table depuis le disque.

```
PROCEDURE ChargeMatrices;
{
 CP/M 2.2 seulement, à inclure au début de vos applications.
}
Var Matrices : Array [0..255,1..8] Of Byte;
    Fic      : File;
Begin
 Assign (Fic,'MATRICES.DAT');
 Reset (Fic);
 Inline (
  $11/>0/        { Ld   DE,$0000    ;A partir de Chr 0. }
  $21/>Matrices/{ Ld   HL,Matrices ;Adresse de la table. }
  $CD/$BE9B/     { Call Entrer-Firmware }
  $BBAB        {DW Txt Set-M-Table  ;Initialise la table. }
```

```
 );
 BlockRead (Fic,Matrices,16);      { Lecture directe de la table. }
 Close (Fic)
End.
```

A la fin du programme, si l'on retourne sous CP/M 2.2, il est nécessaire de réinitialiser les matrices car celles-ci pourront être recouvertes par un quelconque programme donnant alors des caractères illisibles à l'écran. Le plus simple est d'appeler la routine Firmware 'TXT-Initialise' à la fin de votre application. Cette routine réinitialise entièrement toutes les fonctions de gestion du texte du Firmware.

Exemple :

```
Inline (
 $CD/>$BE9B/   { Call Entrer-Firmware }
 $BB4E       {DW TXT-Initialise}
)
```

LES CODES DE CONTROLE ET LES SEQUENCES ESCAPE

Sous CP/M 2.2, les codes de contrôle sont ceux installés en standard sur l'Amstrad (caractères 0 à 31), il suffit de les inclure dans les instructions WRITE pour obtenir l'effet désiré, de la même façon qu'en BASIC avec l'instruction PRINT.

Exemples :

Write (#7) ou Write (^G) émet un bip. (En BASIC : PRINT CHR$(7);)
Write (#4#2) met l'écran en mode 2. (En BASIC : PRINT CHR$(4);CHR$(2);)

Par contre, avec CP/M Plus, la majorité des codes de contrôle sont inopérants. Voici les seuls codes interprétés :

Ctrl G (#7) : signal sonore.
Ctrl H (#8) : curseur en arrière d'un caractère.
Ctrl J (#10) : curseur en bas d'une ligne.
Ctrl M (#13) : retour chariot (curseur à gauche de la ligne).
Ctrl [(#27) : code ESCAPE.

Ces codes sont tous compatibles avec CP/M 2.2.

En CP/M Plus, le code ESCAPE suivi de un ou plusieurs paramètres permet d'activer les fonctions de contrôle. Ce code peut être généré directement par le clavier si la commande SETKEYS KEYS.CCP a été préalablement introduite. (Voir le manuel du 6128 pour la liste des codes.)

Exemple :

Write (#27'3B') ou Write (^['3B') met l'écran en mode 2.

La commande de mode est ESC '3' et le paramètre est 'B' pour le mode 2.

Pour rendre compatible un programme qui fait usage de fonctions de contrôle, le plus simple est de remplacer les codes de contrôle dans le programme par une CONSTANTE SANS TYPE, on pourra bien sûr conserver uniquement les codes de contrôle communs aux deux CP/M (^G,^H,^J,^M,^[). Pour obtenir l'une ou l'autre version du programme il suffira de changer la valeur des constantes dans l'en-tête du programme.

Exemple :

```
PROGRAM Essai; {version CP/M Plus}
{
 Modifier uniquement les valeurs suivantes
}
CONST ModeEcran = ^['3'; { Mode écran  : mettre #4  pour CP/M 2.2 }
      ClrEcran  = ^['E'; { Efface écran : mettre #12 pour CP/M 2.2 }
```

```
VAR   Choix      : Byte;

PROCEDURE Mode (m :Byte);
Begin
 Write (ModeEcran,Chr(m))
End;

PROCEDURE EffaceEcran;
{
 Même action que ClrScr.
}
Begin
 Write (ClrEcran)
End;

Begin
 EffaceEcran;
 Repeat
  Write (^G'Mode {cran (0,1 ou 2) ?');
  ReadLn (Choix);
 Until Choix in [0,1,2];
 ModeEcran (Choix+64)  { Donne '@' pour 0, 'A' pour 1, 'B' pour 2. }
End.                   { sous CP/M 2.2, la valeur est masquée avec ...}
                       { ... 3 par le système ce qui redonne 0,1,2.}
```

Cette méthode présente néanmoins deux défauts. Tout d'abord, les constantes utilisent plus de mémoire car elles doivent être enfermées dans des virgules dans les expressions WRITE (voir Chapitre 3 : 'Le compilateur TURBO PASCAL'), le surplus peut être de plusieurs kilos octets pour les gros programmes, dans les cas extrêmes cette méthode peut être inutilisable, il faudra laisser les codes de contrôle tels quels dans le programme.

Le deuxième problème vient du fait que certaines séquences ESCAPE du CP/M Plus n'ont pas d'équivalence sous CP/M 2.2. Ce sont en particulier des fonctions bien utiles telles que :

ESC j : sauve la position du curseur.
ESC k : restitue la position du curseur.
ESC p : se met en vidéo inverse.
ESC q : se met en vidéo normale.

Pour les fonctions de vidéo normale/inverse, il existe un code sous CP/M 2.2 (#24) qui inverse les couleurs du papier et du stylo mais celui-ci agit en bascule à chaque utilisation et au bout d'un certain moment nous ne savons plus dans quel mode nous nous trouvons.

Il y a néanmoins un moyen pour utiliser sous CP/M 2.2 ces fonctions particulières, sans changer la syntaxe du code, en détournant l'action du code ESCAPE sous CP/M 2.2 (normalement sans action). La procédure présentée émule sous CP/M 2.2 l'action de certaines séquences ESCAPE du CP/M Plus, elles sont alors utilisables sous CP/M 2.2 avec leur syntaxe normale.

Exemple :

```
Write ('Ce texte est en '#27'pvidéo inverse'#27'q à l'écran.');
```

Sous CP/M Plus, il est affiché :

```
Ce texte est en VIDEO INVERSE à l'écran.
```

(Ce qui est en majuscules est en vidéo inverse.)

Sous CP/M 2.2, avec l'émulateur, le résultat est identique.

Seules les séquences citées ci-dessus sont simulées mais rien ne vous empêche d'ajouter de nouvelles fonctions et d'émuler complètement CP/M Plus, réglant ainsi tous les problèmes. Ceci peut être facilité par le fait que certaines séquences ESCAPE ont un caractère de contrôle du CP/M 2.2 à l'action identique, néanmoins le programme cité en exemple ne peut simuler que les séquences ESCAPE sans paramètres, si vous voulez en utiliser, il faudra le modifier.

Les codes de contrôle standard de l'Amstrad sont interprétés à travers une table gérée par le Firmware qui définit pour chaque code le nombre de paramètres à attendre ainsi que l'adresse mémoire de la routine de traitement. Cette routine doit se situer dans les 32 Ko centraux de RAM ($4000 à $BFFF), c'est-à-dire non recouvrables par les ROM, comme le code d'un programme TURBO PASCAL commence à l'adresse $2000 environ, il n'est pas certain

que la procédure d'émulation se trouve bien dans les 32 Ko centraux, il faudra sans doute la placer le plus près possible de la fin du texte de votre programme pour qu'elle soit à une adresse supérieure à $4000. Dans les cas vraiment impossibles, le mieux est de la loger en dehors de la zone du programme en la plaçant dans un fichier binaire chargé à part ou en écrivant une procédure chargeuse qui transférera les codes à l'adresse adéquate, dans les deux cas il faudra protéger la routine en diminuant la 'End-Address' de compilation (voir Chapitre 4 : 'Manipulation des fichiers binaires').

Le code de contrôle ESCAPE (ASCII n° 27) n'est normalement pas interprété par le Firmware de l'Amstrad, la procédure d'émulation active le code et le dirige vers une routine qui reconnaît la séquence Escape en testant le caractère envoyé immédiatement après ESC ; si le caractère est reconnu, le sous-programme traitant la séquence est appelé, sinon le caractère est affiché à l'écran, ce qui permet d'afficher les caractères de code ASCII inférieur à 32 comme le fait CP/M Plus.

```
PROCEDURE SequESC;
{
 Ne fonctionne qu'en CP/M 2.2.

 DOIT se trouver dans les 32 KO centraux de RAM ($4000 à $BFFF).

 Il est inutile d'utiliser Entrer-Firmware pour appeler des routines
 Firmware à l'intérieur de SequESC car cette procédure est incluse dans
 le traitement de l'affichage des caractères qui est sécurisé quant à
 l'emploi des registres auxiliaires et de IY.
 Emulation des séquences du CP/M Plus :

 (ESC j), (ESC k), (ESC p), (ESC q) et (ESC caractère).

                              -------------

 Variables initialisées, seront modifiées par l'émulateur :

}
CONST XYs : Integer=$0101;  { Pour sauvegarder position du curseur. }
                            { x,y sont initialisés à 1,1.           }
   ModeInv : Boolean=False;  { Drapeau mode inverse/normal.          }
                            {              True   /False            }
```

```
PROCEDURE ESCj;
{
 Simule ESC j, sauver la position du curseur.
}
Begin
 Inline (
  $CD/>$BB78/    { Call Txt-Get-Cursor ;Position du curseur ?}
  $22/>XYs       { Ld   (XYs),HL       ;Sauve}
 )
End;

PROCEDURE ESCk;
{
 Simule ESC k, récuperer la position du curseur.
}
Begin
 Inline (
  $2A/>XYs/      { Ld   HL,(XYs)       ;Récupère XY}
  $C3/>$BB75     { Jp   Txt-Set-Cursor ;Positionne le curseur}
 )
End;

PROCEDURE ESCp;
{
 Simule ESC p, établir la vidéo inverse.
}
Begin
 Inline (
  $21/>ModeInv/ { Ld   HL,ModInv      ;Adresse drapeau état actuel}
  $AF/          { Xor  A              }
  $BE/          { Cp   (HL)           ;Etat actuel ?}
  $C0/          { Ret  NZ             ;TRUE : déjà en vidéo inverse :
FIN}
  $34/          { Inc  (HL)           ;FALSE : positionne en TRUE et
...}
  $C3/>$BB9C    { Jp   Txt-Inverse    ;... met en vidéo inverse}
 )
End;

PROCEDURE ESCq;
```

```
{
 Simule ESC q, établir vidéo normale.
}
Begin
 Inline (
  $21/>ModeInv/ { Ld   HL,ModInv      ;Adresse drapeau état actuel}
  $AF/          { Xor  A              }
  $BE/          { Cp   (HL)           ;Etat actuel ?}
  $C8/          { Ret  Z              ;FALSE : déjà en vidéo nor-
male :FIN}
  $77/          { Ld   (HL),A         ;TRUE : positionne en FALSE et
...}
  $C3/>$BB9C    { Jp   Txt-Inverse    ;... repasse en vidéo normale}
 )
End;

PROCEDURE Table;
{
 Table des séquences à reconnaitre.
 Il y a 4 séquences émulées.
 La table comporte un octet pour le code ASCII de la séquence et deux
 octets pour l'adresse de la routine de traitement de la séquence.
}
Begin
 Inline (
  $6A/>ESCj/    { 'j',DW ESCj }
  $6B/>ESCk/    { 'k',DW ESCk }
  $70/>ESCp/    { 'p',DW ESCp }
  $71/>ESCq     { 'q',DW ESCq }
 )
End;

Begin
{
 Coeur de l'émulateur.

 Le Firmware gère un buffer pour stocker les fonctions de contrôle
 complètes (le code suivi des paramètres).

 A l'entrée, le Firmware transmet dans les registres :
  A =dernier caractère du buffer de contrôle (càd le nom de la séquence)
```

```
  B =longueur du buffer (ici 2 : ESC et le nom de la séquence)
  C =idem A
  HL=adresse du buffer (sur le caractère ESC)
}
 Inline (
                { Recherche si le caractère dans A est dans la table }
  $21/>Table/   { Ld   HL,Table ;Début de la table}
  $06/4/        { Ld   B,$04    ;4 éléments dans la table}
  $BE/       {L00 :Cp   (HL)     }
  $23/          { Inc  HL       }
  $28/7/        { Jr   Z,L01    ;OK,dans table}
  $23/          { Inc  HL       }
  $23/          { Inc  HL       }
  $10/<-8/      { Djnz L00      }
                { N'est pas dans la table, affiche le caractère }
                { Permet d'afficher les caractères de code < 32 }
  $C3/>$BB5D/   { Jp   Txt-Wr-Char ;et FIN}
                {               }
                { OK, appelle le sous-programme}
  $7E/       {L01 :Ld   A,(HL)   ;Charge HL avec l'adresse dans la table}
  $23/          { Inc  HL       }
  $66/          { Ld   H,(HL)   }
  $6F/          { Ld   L,A      }
  $E9           { Jp   (HL)     ;Exécute et FIN}
 )
End;
```

Pour activer la routine, il faut d'abord détourner le code ESCAPE au tout début de l'exécution de votre programme, vous devrez appeler la procédure suivante qui fait appel à la routine Firmware Txt-Get-Controls laquelle retourne dans HL l'adresse de la table des codes de contrôle. La table possède 32 entrées, la première pour le code #00 et la dernière pour le code #31, chaque entrée fait 3 octets, le premier indique le nombre de paramètres nécessaires pour le code de contrôle (ici 1 : j,k,p ou q) et les deux autres sont l'adresse de la routine de traitement du code.

```
PROCEDURE ConfigESC;
{
 Redirige code de contrôle ESC vers SequESC.
}
```

```
Begin
 Inline (
  $CD/>$BE9B/    { Call Entrer-Firmware }
  >$BBB1/      {DW Txt-Get-Controls }
  $11/>81/       { Ld   DE,3*27  ;Offset pour le code ESC (No 27)}
  $19/           { Add  HL,DE    }
                 { Modifie la table }
  $36/1/         { Ld   (HL),$01 ;1 paramètre nécessaire}
  $11/>SequESC/  { Ld   DE,SequESC}
  $23/           { Inc  HL       }
  $73/           { Ld   (HL),E   ;Nouvelle adresse}
  $23/           { Inc  HL       }
  $72            { Ld   (HL),D   }
 )
End;
```

Note : Il faut bien sûr que la routine SequESC soit déjà définie dans le texte du programme.

A la fin de votre programme, il faut désactiver l'émulateur car celui-ci risque d'être effacé lors du retour au CP/M 2.2. Le plus simple est d'appeler la routine Firmware ' TXT-INITIALISE' déjà décrite pour le problème des jeux de caractères. Elle réinitialise entièrement toutes les fonctions de gestion du texte du Firmware.

CHAPITRE 6

GESTION DES PERIPHERIQUES

Dans ce chapitre nous allons examiner quelques routines utiles pour gérer efficacement l'écran, le clavier, l'imprimante et le lecteur de disquettes.

GESTION DU CLAVIER

En BASIC, le clavier est entièrement paramétrable, ce n'est pas prévu en TURBO PASCAL pour ne pas compromettre la portabilité des programmes. CP/M peut par contre redéfinir l'assignation des touches grâce aux utilitaires SETUP (CP/M 2.2) ou SETKEYS (CP/M Plus). Nous allons voir comment obtenir le même résultat ainsi que d'autres facilités directement dans un programme TURBO PASCAL.

Nous allons utiliser les nombreuses routines Firmware de gestion du clavier, certaines de celles-ci ont besoin de données placées en mémoire vive, ce qui pose des problèmes sous CP/M Plus car les données sont attendues en banque 0 alors que le programme se situe en banque 1, il faut alors utiliser la partie commune de la mémoire vive.

REDEFINITION DES TOUCHES

La procédure suivante effectue la redéfinition des touches du clavier dans le mode Normal, Shift ou Control. Elle gère l'ensemble des paramètres avec une table placée en mémoire vive. La table est divisée en blocs, le premier concerne le mode Normal, le suivant le mode Shift et ensuite le mode Control, il y a donc 3 blocs au maximum mais il peut y en avoir moins pourvu que cet ordre soit respecté. Le premier octet d'un bloc indique le nombre d'entrées dans le bloc, un octet nul indique la fin de la table, chaque entrée est constituée de 2 octets, le premier indique le numéro de la touche à redéfinir (voir le manuel Amstrad), le second indique le nouveau code ASCII qui sera assigné. Vous pouvez bien sûr modifier ces caractéristiques selon votre application.

Pour faciliter le travail, vous pouvez placer la table dans un fichier disque, ce qui permet de créer plusieurs fichiers de configurations différentes. Le programme suivant crée un fichier de configuration qui assigne au pavé numérique certaines lettres accentuées et des signes graphiques du CP/M Plus, si le jeu de caractères du CP/M 2.2 est redéfini comme CP/M Plus (voir Chapitre 5 : 'Problèmes de compatibilité'), le même fichier peut servir avec les deux CP/M.

```
PROGRAM CreeConfigClavier;
{
 Génère un fichier de configuration clavier.
 Les données sont dans la constante CODES.
}
VAR Fic     : File;
    NomFic  : String [14];

CONST Codes : Array [1..80] Of Byte=
(
{Mode NORMAL}
11,          { Nombre d'entrées }
{ Pavé numérique }
{
 Numéro de touche / Code ASCII
}
10,$A8, 11,$A9, 03,$AA, 20,$AB, 12,$BC, 04,$AC, 13,$DD, 14,$B5, 05,$DC,
15,$DE, 07,$DF,

{Mode SHIFT}
11,          { Nombre d'entrées }
{ Pavé numérique }
10,$96, 11,$9E, 03,$9C, 20,$97, 12,$9F, 04,$9D, 13,$93, 14,$9B, 05,$99,
15,$95, 07,$9A,

{Mode CONTROL}
16,          { Nombre d'entrées }
{ Pavé numérique }
10,$86, 11,$8E, 03,$8C, 20,$87, 12,$8F, 04,$8D, 13,$83, 14,$8B, 05,$89,
15,$85, 07,$8A,
{ Voyelles }
69,$E5, 58,$E6, 35,$E7, 34,$E8, 42,$E9,
```

```
{ Fin de la table }
0
);

Begin
 ClrScr;
 Write ('Nom du fichier ?');
 Readln (NomFic);
 Assign (Fic,NomFic);
 ReWrite (Fic);
 BlockWrite (Fic,Codes,1);    { Ecrit le fichier d'un seul trait }
 Close (Fic)
End.
```

Voici maintenant la procédure à inclure dans vos programmes permettant l'exploitation du fichier de configuration, elle est utilisable avec les deux systèmes CP/M en ajustant le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
PROCEDURE FixeTranslation;
VAR Fic    : File;
    Table : Array [1..128] Of Byte; { contiendra 1 enregistrement }
Begin
{
 Lecture des codes.
 Celle-ci peut être effectuée par divers moyens, les codes devront se
 trouver dans la variable TABLE.
 En reprenant l'exemple ci-dessus, le fichier s'appelant CLAVIER.DAT.
}
 Assign (Fic,'CLAVIER.DAT');
 Reset (Fic);
 BlockRead (Fic,Table,1);
 Close (Fic);
{
 Fin de la lecture des codes.
}
 Inline (        { Routine principale }
  $21/>Table/    { Ld   HL,Table }
  $0E/1/         { Ld   C,$01    ;Type entrée (1 :Norm.,2 :Shift,3 :Ctrl}
                 { Boucle principale }
```

```
  $7E/        {L00 :Ld   A,(HL)    ;Nombre entrées pour le type}
  $B7/           { Or   A          ;0?}
  $C8/           { Ret  Z          ;Oui-->Fin}
                 { Boucle pour un bloc }
  $47/           { Ld   B,A        ;Charge compteur avec nombre entrées}
  $C5/        {L01 :Push BC         ;Sauve compteur et type}
  $23/           { Inc  HL         }
  $5E/           { Ld   E,(HL)     ;Num{ro de touche dans E}
  $23/           { Inc  HL         }
  $46/           { Ld   B,(HL)     ;Code ASCII dans B}
  $E5/           { Push HL         ;Sauve pointeur}
                 { Test du type-->Appel de la routine adéquate }
  $79/           { Ld   A,C        ;Type ?}
  $FE/2/         { Cp   $02        }
  $7B/           { Ld   A,E        ;A=numéro touche}
  $30/7/         { Jr   NC,L02     }
  $CD/>Firm/     { Call Firmware ;Type=1-->Mode normal}
  $BB27/       {DW KM-SET-TRANSLATE }
  $18/14/        { Jr   L04        }
  $20/7/      {L02 :Jr   NZ,L03    }
  $CD/>Firm/     { Call Firmware ;Type=2-->Mode shifté}
  $BB2D/       {DW KM-SET-SHIFT   }
  $18/5/         { Jr   L04        }
  $CD/>Firm/ {L03 :Call Firmware ;Type=3-->Mode control}
  $BB33/       {DW KM-SET-CONTROL}
  $E1/        {L04 :Pop  HL         ;Récupère pointeur}
  $C1/           { Pop  BC         ;Récupère compteur et type}
  $10/<-37/      { Djnz L01        ;Boucle pour le type}

  $0C/           { Inc  C          ;Type+1}
  $23/           { Inc  HL         ;Sur prochain bloc}
  $18/<-45       { Jr   L00        ;Boucle principale}
 )
End;
```

REPETITION DES TOUCHES

Pour décider si une touche doit ou non répéter lors de son appui, il suffit d'utiliser la routine Firmware KM-SET-REPEAT, A doit

contenir le numéro de la touche, B contient $FF si la touche doit répéter et $00 si non.

La procédure exemple fixe le pavé numérique en répétition, elle est compatible avec les deux systèmes CP/M, il suffit d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
PROCEDURE FixeRepetition;
PROCEDURE Table;
{
 Table des num{ros des touches devant être répétées.
 La fin de la table est marquée par le code $FF.
}
Begin
 Inline (
  3/4/5/7/10/11/12/13/14/15/20/ { Touches f0 à f9 et . }
  $FF
 )
End;

Begin
 Inline (
  $21/>Table/    { Ld   HL,Table ;Table des numéros}
  $7E/        {L00 :Ld    A,(HL)    }
  $FE/$FF/       { Cp   $FF       ;Fin de la table ?}
  $C8/           { Ret  Z         ;Oui}
  $E5/           { Push HL        ;Sauve pointeur}
  $06/$FF/       { Ld   B,$FF     ;Code répétion}
  $CD/>Firm/     { Call Firmware}
  $BB39/       {DW KM-SET-REPEAT}
  $E1/           { Pop  HL        ;Pointeur}
  $23/           { Inc  HL        ;Sur suivant}
  $18/<-16       { Jr   L00       ;Boucle}
 )
End;
```

FIXER LES CARACTERES D'EXTENSION

Sur l'Amstrad les codes entre $80 et $9F sont dits étendus car ils sont assignés à des chaînes de caractères programmables, par exemple en BASIC la touche Enter en mode Control renvoie la chaîne RUN"+CHR$ (13). Si l'on assigne à une touche une valeur ASCII comprise dans ces limites, on obtient la chaîne de caractères associée au lieu du caractère correspondant, pour l'obtenir il suffit que la chaîne soit tout simplement le caractère voulu.

La procédure suivante réassigne à toutes les chaînes programmables le caractère ASCII de même code. Elle utilise la routine Firmware KM-SET-EXPAND qui reçoit dans HL l'adresse de la chaîne, dans C sa longueur et dans B le code du caractère d'extension. sous CP/M Plus, pour que la chaîne soit directement accessible par le Firmware en banque 0, comme il n'y a qu'un seul caractère (1 seul octet), nous pouvons utiliser un emplacement de la zone non commutée non utilisé par le système, $FC5E dans cet exemple (voir Chapitre 3 : 'Les adresses de début et de fin de compilation').

La procédure est compatible avec les deux CP/M, il suffit de fixer la routine FIRM selon le CP/M utilisé (voir Chapitre 5 : 'Problèmes de compatibilité'), sous CP/M Plus il faut aussi placer la variable Chaîne en ABSOLUTE $FC5E. En mémoire la variable qui est du type Byte est codée sur un octet contenant la valeur, celui-ci est considéré en langage machine comme le caractère de la chaîne.

```
PROCEDURE FixeExtension;
Var Chaine : Byte ABSOLUTE $FC5E;
{
 ABSOLUTE sous CP/M Plus seulement.

 Cette procédure réaffecte aux chanes d'extension le caract}re ASCII
 correspondant.
}
Begin
 For Chaine :=$80 to $9F do
  Inline (
   $21/>Chaine/ { Ld   HL,Chaine;Adresse de la chaîne}
   $46/         { Ld   B,(HL)    ;Code du caract}re}
   $0E/$01/     { Ld   C,$01     ;Longueur de la chaîne}
```

```
   $CD/>Firm/    { Call Firmware }
   $BB0F       {DW Km-Set-Expand }
  )
End;
```

TESTER LES TOUCHES

Le Firmware offre la possibilité de tester chaque touche en fournissant son numéro (voir le manuel du BASIC). Cela permet de tester une touche précise du clavier même si d'autres sont appuyées en même temps ou si le code ASCII de celle-ci diffère suivant l'état de Shift ou de Control.

La FUNCTION suivante est TRUE si la touche de numéro T est appuyée et FALSE sinon. Elle est compatible avec les deux CP/M en ajustant le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
FUNCTION TestTouche (T : Byte) : Boolean;
Var OK  : Boolean; { Résultat du test }
    Car : Char;    { Caract}re entré éventuel }
Begin
 Inline (
  $3A/>T/        { Ld   A,(T)     ;Numéro de touche Amstrad}
  $CD/>Firm/     { Call Firmware }
  $BB1E/       {DW KM-Test-Key   ;Z=0 si appuyée}
  $3E/$00/       { Ld   A,$00     ;A=False}
  $28/$01/       { Jr   Z,L00     }
  $2F/           { CPL            ;A=True}
  $32/>OK     {L00 :Ld   (OK),A   }
 );
 If OK then While KeyPressed do Read (Kbd,Car); { Vide buffer cla-
vier }
 TestTouche :=OK
End;
```

GESTION DE L'ECRAN

MODE TEXTE

En BASIC et en Assembleur le système Amstrad gère plusieurs couleurs et mélange le texte et le graphisme. Il utilise aussi un système de 'fenêtres' qui fonctionnent comme des écrans logiques avec leurs propres attributs et systèmes de coordonnées. CP/M et le TURBO PASCAL sont hélas dépourvus de ces possibilités et nous allons voir comment y remédier. Sous CP/M 2.2, la plupart des fonctions texte sont accessibles avec les codes de contrôle (caractères 0 à 31) de la même façon qu'en BASIC. Sous CP/M Plus, il faudra écrire des procédures en langage machine pour appeler directement les routines Firmware de gestion de l'écran. Pour rendre un programme portable sur les deux CP/M, il vaut mieux ne pas utiliser les codes de contrôle sous CP/M 2.2 et écrire des procédures comme sous CP/M Plus dont il suffira de modifier le contenu.

Voici un 'pack' de routines qui vous permettront d'utiliser les fenêtres et les couleurs en texte, il faut déclarer dans votre programme la variable AncF de type Byte en global, elle est réactualisée dans la procédure SeLF (sélection d'une fenêtre) et contient le numéro de l'ancienne fenêtre en cours. Cette valeur est reprise dans certaines procédures pour conserver la fenêtre en cours au cas où une autre fenêtre est sélectionnée pour le traitement. Les routines sont compatibles avec les deux CP/M, il suffit d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
PROCEDURE SeLF (F : Byte);
{
 Sélectionne la fenêtre texte F
}
Begin
 Inline (
  $3A/>F/        { Ld   A,(F)     ;Numéro dans A}
  $CD/>Firm/     { Call Firmware }
  $BBB4/       {DW Txt-Str-Select}
```

```
  $32/>AncF      { Ld    (AncF),A ;Ancienne fenêtre}
 )
End;

PROCEDURE Papier (F,P : Byte);
{
 Fixe la couleur du papier (P) pour la fenêtre F.
 La fenêtre en cours ne change pas.
}
Begin
 SeLF (F);       { Sélectionne la fenêtre demandée }
 Inline (
  $3A/>P/        { Ld    A,(P)     ;Numéro dans A}
  $CD/>Firm/     { Call Firmware }
  $BB96        {DW Txt-Set-Paper }
 );
 SeLF (AncF)     { Remet la fenêtre en cours }
End;

PROCEDURE Stylo (F,S : Byte);
{
 Fixe la couleur du stylo (S) pour la fenêtre F.
 La fenêtre en cours ne change pas.
}
Begin
 SeLF (F);       { Sélectionne fenêtre demandée }
 Inline (
  $3A/>S/        { Ld    A,(S)     ;Numéro dans A}
  $CD/>Firm/     { Call Firmware }
  $BB90        {DW Txt-Set-Pen    }
 );
 SeLF (AncF)     { Remet la fenêtre en cours }
End;

PROCEDURE Bordure (B1,B2 : Byte);
{
 Fixe les 2 couleurs de la bordure
}
Begin
 Inline (
  $3A/>B1/       { Ld    A,(B1)    ;1}re couleur...}
```

```
  $47/          { Ld   B,A       ;...dans B}
  $3A/>B2/      { Ld   A,(B2)    ;2}me couleur...}
  $4F/          { Ld   C,A       ;...dans C}
  $CD/>Firm/    { Call Firmware }
  $BC38        {DW Scr-Set-Border}
 )
End;

PROCEDURE Encre (n,c1,c2 : Byte);
{
 Fixe les 2 couleurs (c1,c2) d'une encre (n).
}
Begin
 Inline (
  $3A/>c1/      { Ld   A,(c1)    ;1}re couleur...}
  $47/          { Ld   B,A       ;...dans B}
  $3A/>c2/      { Ld   A,(c2)    ;2}me couleur...}
  $4F/          { Ld   C,A       ;...dans C}
  $3A/>n/       { Ld   A,(n)     ;Numéro encre dans A}
  $CD/>Firm/    { Call Firmware }
  $BC32        {DW Scr-Set-Ink   }
 )
End;

PROCEDURE Fenetre (F,x1,y1,x2,y2 :Byte);
{
 Définit la fenêtre F
 La fenêtre en cours ne change pas.
}
Begin
 SeLF (F);      { Sélectionne la fenêtre demandée }
 Inline (
  $3A/>x1/      { Ld   A,(x1)    ;Limite x gauche...}
  $67/          { Ld   H,A       ;...dans H}
  $25/          { Dec  H         ;-1}
  $3A/>y1/      { Ld   A,(y1)    ;Limite y haute...}
  $6F/          { Ld   L,A       ;...dans L}
  $2D/          { Dec  L         ;-1}
  $3A/>x2/      { Ld   A,(x2)    ;Limite x droite...}
  $57/          { Ld   D,A       ;...dans D}
  $15/          { Dec  D         ;-1}
```

```
  $3A/>y2/       { Ld   A,(y2)    ;Limite y basse...}
  $5F/           { Ld   E,A       ;...dans E}
  $1D/           { Dec  E         ;-1}
  $CD/>Firm/     { Call Firmware ;
  $BB66        {DW Txt-Win-Enable}
 );
 SeLF (AncF)     { Remet la fenêtre en cours }
End;

PROCEDURE ClrFen (f : Byte);
{
 Vide la fenêtre F et place le curseur en haut à gauche.
 La fenêtre en cours ne change pas.
}
Begin
 SeLF (F);       { Sélectionne la fenêtre demandée }
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BB6C        {DW Txt-Clear-Window }
 );
 Gotoxy (1,1);  {
                 Curseur en haut à gauche. sous CP/M Plus, ce n'est pas
                 fait automatiquement selon la configuration de l'écran.
                }
 SeLF (AncF)     { Remet la fenêtre en cours }
End;
```

Il y a néanmoins quelques problèmes qui se posent avec le masquage du curseur et le défilement des fenêtres sous CP/M Plus.

En CP/M Plus, les séquences Escape ESC 'e' et ESC 'f' respectivement activent et désactivent l'affichage du curseur, elles correspondent aux routines Firmware Txt-Cur-On ($BB81) et Txt-Cur-Off ($BB84). Si le curseur est désactivé, l'affichage est plus rapide et plus agréable mais pour les entrées au clavier il faut l'activer sinon on ne sait pas du tout où l'on est ! Si le curseur est actif, il y a néanmoins possibilité d'autoriser ou non l'affichage du curseur à un moment donné en appelant les routines Firmware Txt-Cur-En-able ($BB7B) et Txt-Cur-Disable ($BB7E), par exemple si l'on veut saisir une touche du clavier sans afficher de curseur, chaque fenêtre étant paramétrable séparément.

Exemples :

Voici deux procédures pour gérer l'autorisation d'affichage du curseur.

```
PROCEDURE CursEna;
{
 Autorise le curseur.
}
Begin
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BB7B        {DW Txt-Cur-Enable}
 )
End;

PROCEDURE CursDis;
{
 N'autorise pas l'affichage.
}
Begin
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BB7E        {DW Txt-Cur-Disable}
 )
End;
```

En CP/M 2.2 comme en BASIC, le curseur est automatiquement désactivé pour l'affichage et réactivé pour les entrées au clavier. L'affichage du curseur peut être commandé avec les caractères de contrôle #2 (supprime curseur) et #3 (active le curseur).

Avec CP/M Plus, il y a un problème quand une fenêtre doit défiler (le contenu monte vers le haut). Le système de défilement de l'écran (*scrolling*) est fait de telle façon que si une écriture texte doit s'effectuer avec une ordonnée supérieure à 24 ou 25 (selon si la ligne d'état est activée en bas de l'écran), CP/M fait monter l'écran en entier d'une ligne. Si une écriture dans une fenêtre entraîne une ordonnée supérieure à ces valeurs, le même défilement se produit même si la fenêtre fait moins de 24 ou 25 lignes de haut !

Pour éviter cela vous pouvez par exemple fixer vous-même avec GOTOXY les ordonnées si elles risquent de dépasser ces valeurs ou aussi faire tous les affichages pour une fenêtre avec un GOTOXY sur une position sous la fenêtre, la fenêtre défilera bien mais elle commencera à se remplir par le bas.

Exemple :

Soit une fenêtre de limites verticales X=10 et Y=20. Si l'on effectue la boucle :

```
{ La fenêtre est sélectionnée }

Gotoxy (1,1);
For i :=1 to 40 Do WriteLn ('A');
```

Lorsque i sera supérieur à 24 ou 25, tout l'écran montera d'un coup à chaque fois (la fenêtre défilera ensuite).

Par contre, si l'on écrit :

```
For i :=1 to 40 Do
Begin
 Gotoxy (1,23); { Positionne sous la fenêtre }
 Writeln ('A')
End;
```

La fenêtre défilera correctement mais elle se remplira par le bas.

MODE GRAPHIQUE

L'utilisation des fonctions graphiques se fait tout aussi simplement que pour le texte, il faut appeler directement les routines Firmware de gestion du graphisme. Les exemples suivants sont compatibles avec les deux CP/M en ajustant le contenu de la procédure FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

Exemples :

```
PROCEDURE UserC (x,y : Integer);
{
 Saisit ds DE et HL les coordonnées x,y.
 Procédure utilisée par les autres.
}
Begin
 Inline (
  $ED/$5B/>x/    { Ld   DE,(x) }
  $2A/>y         { Ld   HL,(y) }
 )
End;

PROCEDURE DefGrFen (x,x1,y,y1 : Integer);
{
 Sélectionne la fenêtre graphique. Fixe l'origine en bas à gauche.
}
Begin
 UserC (x,x1);
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BBCF        {DW Gra-Win-Width }
 );
 UserC (y,y1);
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BBD2        {DW Gra-Win-Height}
 );
{ Fixe origine en bas a gauche }
 UserC (x,y);
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BBC9        {DW Gra-Set-Origin}
 )
End;

PROCEDURE GrWrite (Ch : Char);
{
 Affiche un caract}re à la position du curseur graphique.
```

```
}
Begin
 Inline (
  $3A/>Ch/       { Ld   A,(Ch)    ;Code ASCII dans A}
  $CD/>Firm/     { Call Firmware }
  $BBFC        {DW Gra-Wr-Char    }
 )
End;

PROCEDURE ModeGra (mg : Byte);
{
 Fixe le mode d'encre graphique.
 0 :Force  1 :XOR  2 :AND  3 :OR
}
Begin
 Inline (
  $3A/>mg/       { Ld   A,(mg)    }
  $CD/>Firm/     { Call Firmware }
  $BC59        {DW Scr-Access     }
 )
End;

PROCEDURE MoveA (x,y : Integer);
{
 Place le curseur graphique en coordonnées absolues.
}
Begin
 UserC (x,y);
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BBC0        {DW Gra-Move-Absolute }
 )
End;

PROCEDURE DrawA (x,y : Integer; m : Byte);
{
 Trace une droite jusqu'en x/y selon le mode d'encre m.
}
Begin
 GraMode (m);
 UserC (x,y);
```

```
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BBF6        {DW Gra-Line-Absolute}
 )
End;
```

GESTION DE L'IMPRIMANTE

REACTIVATION DE L'IMPRIMANTE

Si pendant une impression l'imprimante est OFF-LINE à la suite d'un manque de papier ou d'une erreur, sous CP/M Plus le message suivant apparaît :

LPT not ready - Retry, Ignore or Cancel?

Si l'on répond C (Cancel) pour continuer malgré tout le programme, l'imprimante n'est plus utilisable car CP/M l'affecte alors à l'organe physique NULL. Il faut alors sortir du programme et utiliser DEVICE.COM pour réassigner l'imprimante avec LST :=LPT.

Sous CP/M 2.2, il n'y a pas de message d'erreur et il faut utiliser STAT.COM pour modifier l'I/O Byte.

Pour réassigner l'imprimante directement dans un programme en TURBO PASCAL, il faut modifier en mémoire le ou les octets codant l'assignation de LST, il y a une méthode pour chaque CP/M utilisé car les codages sont différents.

Sous CP/M 2.2

Il suffit avant chaque impression de modifier l'I/O Byte pour obtenir LST :=LPT par :

BDOS (8,(BDOS (7) AND $3F) OR $80)

L'I/O Byte est lu par BDOS (7) et masqué pour fixer le bit 7 à 1 et le bit 6 à 0 (fixe LST :=LPT) tout en gardant les autres bits dans leurs états.

Sous CP/M PLUS

La table de combinaison des périphériques logiques se situe dans une zone de la mémoire vive système appelée SCB (Bloc de contrôle du système), l'organe logique LST est codé sur un mot de 16 bits aux offsets $2A et $2B. Le SCB est accessible en lecture et écriture grâce à la fonction BDOS n° 49, celle-ci a besoin d'une table de 6 octets en paramètre.

Le premier travail consiste à lire la précédente configuration de LST au démarrage de l'application, l'imprimante doit alors être normalement en ligne.

```
{ Il faut déclarer en global la variable ScbPb (table de paramètres). }

VAR ScpPb : Array [1..6] Of Char;

PROCEDURE InitScbPb;
{
 Initialise le ScbPb avec la configuration courante.
}
VAR HL : Integer; { Contiendra mot 16 bits de l'ancienne configuration }
Begin
 ScbPb :=#$2A#00#$2A#$FE#00#00;
{
 Explication :
 $2A,00         :Pour lecture à l'offset $2A (initialisation)
 $2A,$FE,Xb,Xh :Pour écriture du mot XhXb à l'offset $2A (réassignation)

 Effectue la lecture du mot à l'offset $2A.
}
 HL :=BdosHL (49,Addr (ScbPb)); { 16384 est la valeur normalement lue }
{ Prépare pour l'écriture }
 ScbPb [5] :=Chr(Lo (HL));
 ScbPb [6] :=Chr(Hi (HL))
End;
```

Avant chaque impression, pour que l'imprimante soit bien affectée à LPT, il suffit de faire :

```
BDOS (49,Addr (ScbPb)+2)
```

(écrit dans le SCB à l'offset $2A le mot de la configuration initiale.)

Il faut noter que le mot lu dans InitScbPb est normalement $4000 (16384), on peut alors simplifier en ne faisant pas la lecture de l'état initial et en plaçant une table de paramètres réduite pour l'écriture en CONSTante.

```
CONST ScbPb : Array [1..4] Of Char = #$2A#$FE#00#$40;
```

La réassignation se fait alors par :

```
BDOS (49,Addr(ScpPb))
```

GESTION DES COMMANDES DE L'IMPRIMANTE

Pour rendre vos impressions plus attrayantes, il faut utiliser les multiples commandes des imprimantes mais les codes sont loin d'être standardisés. Le mieux est d'inclure dans votre application un petit programme de redéfinition des séquences de codes de l'imprimante, de plus les codes pourront être sauvés sur disquette, ce qui permet de constituer une bibliothèque de fichiers de configuration.

Le programme suivant permet la modification d'un tableau de caractères CTRL, il y a 5 chaînes de 8 caractères (CONSTantes NCode=5 et LCode=8), chacune d'entre elles correspond à une séquence de commande.

CTRL [1] : Initialisation
CTRL [2] : Italiques
CTRL [3] : Fin de page
CTRL [4] : Double largeur
CTRL [5] : Simple largeur

Pour les utiliser il suffit de les inclure dans les instructions WRITE :

```
WRITE (Lst,'Ceci est en ',CTRL [4],'double',CTRL [5],' largeur');
```

Sur l'imprimante le mot 'double' est en double largeur.

Vous pouvez bien sûr modifier les caractéristiques du programme pour augmenter ou modifier les codes et leurs fonctions. Le tableau CTRL est considéré comme déjà chargé dans l'exemple, il peut par exemple avoir été précédemment lu au début de l'application par :

```
{ Fic étant un fichier du même type que CTRL }

Assign (Fic,'CLAVIER.DAT');
Reset (Fic);
Read (Fic,Ctrl);
Close (Fic);

PROGRAM Imprimante;
{
 Edition des codes de l'imprimante.
}
CONST HexStr : Array [0..15] of Char='0123456789ABCDEF';
      NCode = 5;        { Nombre de séquences }
      LCode = 8;        { Nombre de codes par séquence }

TYPE  Str2  = String [2];
   CodeCtrl = Array [1..NCode,1..LCode] Of Char;

VAR   Ctrl  : CodeCtrl;          { Tableau regroupant les séquences }
      Fic   : File Of CodeCtrl;  { Fichier associé }
      Car   : Char;              { Entrée du clavier }
      x,y   : Byte;              { Positions logiques sur écran }

FUNCTION Hexa (n : Char) : Str2;
{
 Convertit l'octet N en héxadécimal.
}
Begin
 Hexa :=HexStr [Ord(n) DIV 16]+HexStr [Ord(n) MOD 16]
```

```
End;

PROCEDURE Presentation;
{
 Page de présentation.
}
Begin
 ClrScr;
 Gotoxy (8,1);
 Writeln ('Réglage de votre imprimante');
 Writeln;
 Writeln ('Initialisation. :');
 Writeln ('Italiques..... :');
 Writeln ('Fin de page.... :');
 Writeln ('Double largeur. :');
 Writeln ('Simple largeur. :');

 For x :=3 To NCode+2 Do
  Begin
   Gotoxy (18,x);
   For y :=1 To LCode Do Write (Hexa(Ctrl [x][y]),' ')
  End;

 Gotoxy (16,10);
 Write ('ESC=Fin')
End;

PROCEDURE Edition;
Var Num : Byte;         { Indique sur quel chiffre se fait l'entrée
                          Num=0 : Chiffre gauche.
                          Num=1 : Chiffre droite.}

PROCEDURE Curseur;
{
 Affiche le curseur selon x/y.
}
Begin
 Gotoxy (18+Num+3*Pred(x),y+2)
End;

PROCEDURE Entrees;
```

```
{
 Affiche le caractère entrée.
 Calcule la valeur du code.
}
VAR Valeur,Err : Integer;
Begin
 Write (Car);
 If Num=0 Then
        Val ('$'+Copy(Hexa(Ctrl[y][x]),1,1)+Car,Valeur,Err)   { Gauche }
   Else Val ('$'+Car+Copy(Hexa(Ctrl[y][x]),2,1),Valeur,Err); { Droite }
 Ctrl[y][x] :=Chr(Valeur);
 Num :=(Not Num) AND 1   { Bascule entre 0 et 1 }
End;

Begin
 x :=1;y :=1;
 Num :=0;  { Défaut : chiffre à gauche }

 REPEAT
  Curseur;
  Read (Kbd,Car);
  Car :=UpCase (Car);

  CASE Car Of
{ Mouvements du curseur (fl}ches du clavier) }
    #240 : If y>1 Then y :=Pred(y);       {haut}
    #241 : If y<NCode Then y :=Succ(y);   {bas}
    #242 : If x>1 Then x :=Pred(x);       {gauche}
    #243 : If x<SCode Then x :=Succ(x);   {droite}

{ Entree d'une valeur }
   Else If Car in ['0'..'9','A'..'F'] Then Entrees

  End; {case}

 UNTIL Car=#252         { Touche ESC }
End;

PROCEDURE Sauvegarde;
Begin
 Gotoxy (5,10);
```

```
 Write (#7'Sauvegarde de ces paramètres ?');

 Repeat
  Read (Kbd,Car);
  Car :=UpCase (Car);
 Until Car in ['O','N'];

 If Car='O' Then
  Begin
   Assign (Fic,'CLAVIER.DAT');
   Rewrite (Fic);
   Write (Fic,Ctrl);
   Close (Fic)
  End
End;

Begin
 Presentation;
 Edition;
 Sauvegarde
End.
```

GESTION DES DISQUETTES

Toute bonne gestion d'un lecteur de disque commence par un programme de catalogue, l'exemple suivant effectue le DIRectory du disque, il n'affiche pas la taille de chaque fichier mais en version CP/M Plus, il affiche la taille de la place libre sur le disque. De plus un astérisque est affiché si le fichier est en Read-Only.

```
PROGRAM Catalogue;
Var Drive,   { A=1,B=2 }
    Statut,  { FF=Erreur. 0,1,2,3=Emplacement entrée DIR dans Buffer }
    i        : Byte;  { Compteur divers }
    Car      : Char;  { Nom du lecteur choisi "A" ou "B" }
    Fcb      : Array [0..35] Of Byte ABSOLUTE $5C; { FCB par défaut }
    Buffer   : Array [1..128] Of Byte; { Contiendra 1 enregistrement }
```

```
PROCEDURE EntreeDir;
{
 Affiche un nom de fichier.
}
VAR Debut : Byte;     { Emplacement des octets pour le fichier dans }
                      { le buffer }
Begin
 Debut := Statut*32;  { Une entrée du catalogue fait 32 octets }
{ Affiche le nom du lecteur }
 Write (Car,' :');
{ Affiche le nom }
 For i := Debut+2 To Debut+9 Do Write(Chr(Buffer[i]));
 Write('.');
{ Affiche l'extension }
 For i :=Debut+10 To Debut+12 Do Write(Chr(Buffer[i] and $7F));
{ Affiche "*" pour Read Only }
 If (Buffer[Debut+10] and $80)<>0 then Write (' *')
                                  else Write ('  ')
End;

Begin
 ClrScr;
 Write ('Numero du lecteur (A,B ou RETURN) ? ');

 Repeat
  Read (Kbd,Car);
  Car :=UpCase (Car);
 Until Car in ['A','B',#13];

 If Car=#13 then Car :=Chr(Bdos (25)+65); { Lecteur courant ? }
 Drive :=Ord(Car)-64;                     { 1 pour A, 2 pour B }

 Writeln (Car);
 Bdos (13);       { Disk Reset }
{ Crée le FCB pour la recherche }
 Fcb [0] :=Drive;
 For i :=1 to 11 do Fcb [i] :=$3F; { ??????????? }
{ Fixe l'adresse de DMA }
 BDos(26,Addr(Buffer));

{ 1ère recherche dans le répertoire }
```

```
 Statut := BDos(17,$5C);

 While Statut<>255 Do    { Tant que recherche OK... }
  Begin
   EntreeDir;            { Affiche l'entrée et }
   Statut :=BDos(18)      { Continue recherche }
  End;  {while}

 Writeln;

{
 CP/M Plus seulement.
 Affiche la place libre en KO.
}
 Bdos (46,Pred(Drive)); { Lecture espace libre }
 Writeln;
 Writeln
('  ',((Buffer[3]*$FFFF+Buffer[2]*256+Buffer[1]) DIV 8) :3,' KO
libres');

 Bdos (13)              { Disk Reset }
End.
```

CHAPITRE 7
APPLICATIONS

DETERMINER SI UN FICHIER EST DEJA OUVERT EN LECTURE

Les types fichiers occupent en mémoire un bloc de 176 octets appelé FIB (Bloc d'interface des fichiers). Il faut noter que les fichiers sans type (File) n'occupent que 48 octets. Les bits du premier octet indiquent l'état du fichier (ouvert/fermé et en lecture/écriture). Le bit 7 (Input Flag) est mis à 1 si le fichier est déjà ouvert par l'instruction RESET par exemple.

Dans certains cas, lorsque l'on utilise souvent un fichier, on ne peut prévoir si le fichier est ouvert ou fermé lors d'une demande de lecture. Le plus simple est de toujours ouvrir le fichier par Reset avant la lecture, mais si le fichier est déjà ouvert, cela fait perdre du temps inutilement. On peut éviter cela en testant d'abord l'état du bit 7, le fichier ne sera ouvert que si le bit 7 est à 0.

Exemple :

```
{
 Enrg est une variable de type quelconque.
 Fichier est un fichier de Enrg (File Of Enrg)..
}

PROCEDURE LireFichier (Index :Integer);
{
 Lecture de l'enregistrement INDEX.
}
Begin
 If Mem [Addr(Fichier)] < 128 Then Reset (Fichier);
 Seek (Fichier,Index);  { Positionne... }
 Read (Fichier,Enrg)    { ...et lit }
End;
```

CALCULER LA TAILLE D'UN FICHIER

Le FIB décrit ci-dessus contient aussi le FCB (Bloc de contrôle du fichier) utilisé par CP/M. Le FCB occupe les octets d'offsets 12 à 47 dans le FIB, tous les autres octets sont exclusivement utilisés par le TURBO PASCAL et n'ont aucune utilité sous CP/M. Le FCB est utilisé par CP/M pour toutes les opérations sur les fichiers.

Pour calculer la taille d'un fichier, il faut utiliser la fonction BDOS n° 35 qui fournit le nombre d'enregistrements de 128 octets utilisés, DE doit contenir l'adresse du FCB du fichier. Le résultat est retourné dans les octets d'offsets 33 et 34 du FCB. La fonction exemple nécessite comme argument un nom valide de fichier et retourne dans un entier sa taille.

Note : Le FCB d'un fichier est complètement défini avec l'instruction ASSIGN.

```
TYPE Str14 = String [14]; { Pour le nom du fichier }

FUNCTION TailleFichier (NomFichier : Str14) : Integer;
VAR Fic : File;
{
 On utilise un fichier sans type car il est compatible avec tous les
 types fichier et occupe moins de mémoire.
}

Begin
 Assign (Fic,NomFichier);    { Le FCB est cr{{ }
 Bdos (35,Addr (Fic+12));    { Calcule la taille du fichier }
 TailleFichier :=128*( Mem [Addr(Fic)+12+33] +256* Mem
[Addr(Fic)+12+34])
End;
```

TESTER LE TEMPS ECOULE

Dans de nombreuses applications, il peut être nécessaire de déclencher ou arrêter une opération après un certain temps, d'une façon similaire à l'instruction AFTER du BASIC.

Dans l'exemple suivant, le programme boucle sur lui-même jusqu'à l'appui de la touche RETURN, si elle n'est pas appuyée au bout d'un temps donné, on sort de la boucle.

Il faut définir une variable globale DELAI de type entier qui contiendra la valeur du délai souhaité en secondes, une variable globale TEMPS de type Boolean qui sera positionnée par le programme à TRUE dès que le délai sera dépassé ainsi qu'une variable globale entière HEURE0 à l'usage du programme qui contiendra l'heure en secondes du début du comptage.

La procédure RAZDELAI doit être appelée pour initialiser le comptage du temps, elle positionne la variable TEMPS sur FALSE. Le nombre de secondes à compter doit être placé dans la variable DELAI et la procédure TESTDELAI positionne la variable TEMPS sur TRUE si le délai est écoulé. Il suffit de l'appeler à l'intérieur de la boucle et de tester TEMPS pour savoir si le délai est écoulé.

Il y a deux versions du programme, la première ne fonctionne qu'avec CP/M Plus car elle utilise une fonction BDOS spécifique à ce système, elle est en partie en langage machine pour accélérer le traitement. La deuxième version est spécifique à l'Amstrad car elle utilise une routine du Firmware, elle fonctionne indifféremment avec les deux CP/M.

VERSION CP/M PLUS

La fonction BDOS 105 renvoie la date et l'heure sur 4 octets à l'adresse fournie dans DE et place dans l'accumulateur le nombre de secondes. Ces données sont en BCD (décimal codé binaire), elles sont reçues dans une variable ARRAY [1..4] OF BYTE (ou CHAR).

```
PROGRAM Temps;
VAR Touche : Char;      { Touche appuyée }

    Delai,              { Variables nécessaires au programme }
    Heure0 : Integer;
     Temps : Boolean;

FUNCTION Time : Integer;
{
 Retourne le temps système en secondes en format binaire.
}
Var NSec   : Integer;
    Buffer : Array [1..4] Of Byte;

PROCEDURE Conv;
{
 Convertit le nombre BCD dans A en binaire, BC est modifié.
}
Begin
 Inline (
  $47/          { Ld   B,A      ;Sauve}
  $E6/$0F/      { And  $0F      ;A=unités}
  $4F/          { Ld   C,A      ;C=unités}
                { Calcule les dizaines }
  $78/          { Ld   A,B      ;Récupère valeur BCD initiale}
  $E6/$F0/      { And  $F0    ;A=dizaines (*16) car dans octet fort}
  $CB/$3F/      { SrL  A        ;A=dizaines*8}
  $47/          { Ld   B,A      ;B=dizaines*8}
  $CB/$38/      { SrL  B        ;B=dizaines*4}
  $CB/$38/      { SrL  B        ;B=dizaines*2}
  $80/          { Add  A,B      ;A=diz.*2+diz.*8=dizaines*10}
                { Additionne les dizaines et les unités }
  $81           { Add  A,C      ;A=nombre en binaire}
 )
End;

Begin
 Bdos (105,Addr(Buffer));  { Time Please }
 Inline (       { Nombre de secondes dans A }
  $CD/>Conv/    { Call Conv     ;A=nombre de secondes en binaire}
  $26/$00/      { Ld   H,$00    }
```

```
  $6F/           { Ld   L,A       }
  $E5/           { Push HL        ;Dans HL et sauve}
  $3A/>Buffer+3/{ Ld   A,(Buffer[4]) ;Nombre minutes}
  $CD/>Conv/     { Call Conv       ;A=nombre de minutes en binaire}
                 { Effectue HL :=60*A pour avoir le nombre de}
                 { de secondes pour A minutes}
  $6C/           { Ld   L,H        ;HL=$0000}
  $11/>60/       { Ld   DE,60      }
  $B7/           { Or   A          }
  $28/04/        { Jr   Z,L01      ;Ne fait rien si 0 minutes}
  $19/         {L00 :Add  HL,DE     ;Additionne 60}
  $3D/           { Dec  A          }
  $20/<-4/       { Jr   NZ,L00     ;A fois}
                 {                 }
  $D1/         {L01 :Pop  DE        ;Récupère le nombre de secondes}
  $19/           { Add  HL,DE      ;HL=nombre de secondes total}
  $22/>NSec      { Ld   (NSec),HL;Résultat}
 );
 Time :=NSec
End;

PROCEDURE RazDelai;
{
 Initialise le comptage.
 Place l'heure (minutes + secondes) initiale dans HEURE0
}
Begin
 Heure0 :=Time;
 Temps :=False
End;

PROCEDURE TestDelai;
{
 Teste si le délai est écoulé.
 Si le délai choisi est nul ou si le délai est déjà écoulé, alors ne
 fait rien.
}
Var Heure : Integer; { Contient l'heure actuelle }
Begin
 If Temps Or (Delai=0) Then EXIT;
 Heure :=Time;
```

```
{ Corrige si une nouvelle tranche horaire est passée. }
 If Heure < Heure0 Then Heure :=Heure+3600;
{ Teste }
 Temps :=(Heure-Heure0 >= Delai)
End;

Begin
 ClrScr;
 Write ('Temps d'attente en secondes ?');
 ReadLn (Delai);
 RazDelai;              { Initialise le comptage }

 Repeat
  Ik KeyPressed Then Read (Kbd,Touche);
  TestDelai;            { Teste le temps écoulé }
 Until (Touche=^M) or Temps;

 If Temps Then Writeln ('Temps écoulé !')
End.
```

VERSION SPECIFIQUE AMSTRAD

Dans cette version, la variable globale HEURE0 n'est pas nécessaire. La procédure RAZDELAI remet à zéro l'horloge interne du système, ce qui facilite le test du délai écoulé, alors que dans la précédente version il fallait garder dans HEURE0 l'heure lors de l'initialisation. Les procédures sont compatibles avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
PROCEDURE RazDelai;
{
 Remet à z{ro l'horloge interne.
}
Begin
 Inline (
  $11/>0/       { Ld   DE,$0000 }
  $21/>0/       { Ld   HL,$0000 }
  $CD/>Firm/    { Call FirmWare }
```

```
  $BD10        {DW KL-Time-Set    ;RAZ Time}
 );
 Temps :=False
End;

PROCEDURE TestDelai;
Var TimeH,TimeL :Integer;
Begin
 If Temps or (PagInf.Delai=0) then EXIT;
 Inline (
  $CD/>Firm/      { Call FirmWare    }
  $BD0D/        {DW KL-Time-Please   }
  $22/>TimeL/     { Ld    (TimeL),HL }
  $ED/$53/>TimeH { Ld    (TimeH),DE }
 );
{ Effectue la conversion en secondes }
 Temps :=(Trunc(TimeH*218.45+TimeL/300) > Delai)
End;
```

PROCEDURES EVOLUEES DE SAISIE

Pour effectuer une saisie du clavier, le TURBO PASCAL ne dispose que de l'instruction READLN (Var). Il n'y a aucune vérification du type des caractères entrés et si l'on entre des caractères alphanumériques alors qu'une entrée numérique est attendue, une erreur se produit. De plus, on ne peut limiter la taille de l'entrée et les possibilités d'édition sont des plus réduites.

Pour pallier les carences de l'instruction READLN, il faut disposer de fonctions de saisie évoluées. Dans l'exemple suivant, on peut définir la taille de l'entrée en nombre de caractères, le type de caractères acceptés et pour les entrées numériques l'intervalle des valeurs possibles. Les procédures InputString, InputInteger et InputReal permettent respectivement la saisie d'une chaîne de caractères, d'une valeur entière et réelle. Le champ de la saisie est symbolisé par des caractères '_' et est situé à la position courante du curseur. Quelques touches du clavier ont une fonction particulière :

COPY : Redonne la valeur par défaut.
CLR : Efface le caractère sous le curseur.
DEL : Efface le caractère à gauche du curseur et recule le curseur.
Flèches droites et gauches : Déplacent le curseur.
TAB : Bascule du mode Insertion. Par défaut, les caractères se recouvrent.

InputStrinG

Effectue la saisie d'une chaîne de caractères. Sa taille maximum est de 80 caractères, la syntaxe est :

```
InputString (Lg,Maj,Def)
```

LG est une valeur de type Byte indiquant la taille de la saisie (1 à 80).

MAJ est une valeur de type Boolean. Si TRUE, les caractères entrés sont automatiquement convertis en majuscules.

DEF est une chaîne de caractères, elle contient lors de l'appel la valeur par défaut de la chaîne. C'est un paramètre par variable (VAR), elle contiendra le résultat.

InputInteger

Effectue la saisie d'une valeur entière. La syntaxe est :

```
InputInteger (LG,Mini,Maxi,Def)
```

LG est une valeur de type Byte indiquant la taille de la saisie (1 à 7).

MINI et MAXI sont deux valeurs entières indiquants les bornes acceptées par le programme.

DEF est la valeur de type Integer par défaut. C'est un paramètre par variable (VAR), elle contiendra le résultat.

InputReal

Effectue la saisie d'une valeur réelle. La syntaxe est :

```
InputReal (LG,Dec,Def)
```

LG est une valeur de type Byte indiquant la taille de la saisie (1 à 18).

DEC est du type Byte et indique le nombre de décimales voulues.

DEF est la valeur de type Real par défaut. C'est un paramètre par variable (VAR), elle contiendra le résultat.

Entrée

Les trois procédures ci-dessus appellent la procédure générale de saisie ENTREE. Celle-ci peut être appelée directement par l'utilisateur. Sa syntaxe est :

```
ENTREE (Maj,LG,Type,Def)
```

MAJ est une valeur de type Boolean. Si TRUE, les caractères entrés sont automatiquement convertis en majuscules.

LG est une valeur de type Byte indiquant la taille de la saisie (1 à 80).

TYPE est un ensemble de caractères (Set Of Char) indiquant quels sont les caractères acceptés pour la saisie.

DEF est une valeur chaîne (String) par défaut. C'est un paramètre par variable (VAR), elle contiendra le résultat.

Pour rendre le programme compatible avec les deux CP/M, il faut ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

```
PROCEDURE Entree (Maj : Boolean; Lg : Byte; VAR TypeCar,DefStr);
{
 Forcer majuscules. Largeur de l'entrée. Type de caractères.
```

```
 Chaîne par défaut.
}
VAR   ch      : Char;          { Caractère entré au clavier }
      Buffer  : String [80];   { Buffer de travail (80 caractères maxi) }
      x,y,                     { Position initiale du curseur }
      pos,i,p : Byte;          {.Position relative dans le buffer.
                                .Divers compteurs.
                                .Divers pour procédures DebutBuff et
                                 FinBuff.}
      Ins     : Boolean;       { Flag pour mode insertion }
      Typ     : Set Of Char;   { Type de caractères }

PROCEDURE Bip;
{ émet un bip }
Begin
 Write (^G)
End;

FUNCTION DebutBuff :Byte;
{
 Donne la position relative du 1er caractère de la chaîne.
 Renvoie 1 si la chaîne est vide.
}
Begin
 Inline (
  $3A/>lg/       { Ld   A,(lg)    ;Longueur du champ dans A}
  $47/           { Ld   B,A       ;Charge compteur avec longueur champ}
  $0E/1/         { Ld   C,$01     ;C=numéro position}
  $21/>Buffer+1/{ Ld   HL,Buffer[1] ;sur 1er caractère}
  $3E/$5F/       { Ld   A,'_'     ;Caractère de 'remplissage' du champ}
  $BE/        {L00 :Cp   (HL)      ;Teste le buffer}
  $20/6/         { Jr   NZ,L01    ;Trouvé un caractère, position dans C}
  $23/           { Inc  HL        ;Pas trouvé : prochain caractère...}
  $0C/           { Inc  C         ;...position +1}
  $10/<-7/       { Djnz L00       ;Boucle sur tout le champ}
  $0E/1/         { Ld   C,$01     ;Pas trouvé du tout : résultat=1}
  $79/        {L01 :Ld   A,C       }
  $32/>p         { Ld   (p),A     ;Résultat dans p}
 );
 DebutBuff :=p
End;
```

```
FUNCTION FinBuff :Byte;
{
 Donne la position relative du dernier caractère de la chaîne.
 Renvoie 1 si la chaîne est vide.
}
Begin
 Inline (
  $3A/>lg/       { Ld   A,(lg)    ;Longueur du champ dans A}
                 { Positionne HL sur la fin du champ }
  $06/0/         { Ld   B,$00     }
  $4F/           { Ld   C,A       }
  $21/>Buffer/   { Ld   HL,Buffer[0] }
  $09/           { Add  HL,BC     ;HL sur Buffer [lg]}
                 {                }
  $47/           { Ld   B,A       ;B=compteur et position relative}
  $3E/$5F/       { Ld   A,'_'     ;Caract}re de remplissage du champ}
  $BE/       {L00 :Cp   (HL)      ;Teste le buffer}
  $20/4/         { Jr   NZ,L01    ;Trouvé un caractère, position dans B}
  $2B/           { Dec  HL        ;Pas trouvé : caractère précédent}
  $10/<-6/       { Djnz L00       ;Boucle sur tout le buffer}
  4/             { Inc  B         ;Pas trouvé du tout : résultat=1}
  $78/       {L01 :Ld   A,B       }
  $32/>p         { Ld   (p),A     ;Résultat dans p}
 );
 FinBuff :=p
End;

PROCEDURE GotoPos;
{
 Place le curseur selon la position relative POS.
}
Begin
{ Passe à la ligne suivante si nécessaire }
 If x+Pred(Pos) > 80 Then Gotoxy (x+Pred(Pos)-80,Succ(y))
                     Else Gotoxy (x+Pred(Pos),y)
End;

PROCEDURE AffBuff;
{
 Réaffiche le buffer en entier et place le curseur selon POS.
```

```
}
Begin
 Gotoxy (x,y);
 Write (Buffer);
 GotoPos
End;

PROCEDURE DeLCar;
{
 Supprime le caractère à la position POS. Le buffer est  décalé  sur  la
 gauche puis réaffiché en entier.
}
Begin
 Delete (Buffer,Pos,1);
 Buffer :=Buffer+'_';
 AffBuff
End;

PROCEDURE EntreeCar;
{
 Le caractère dans CH est bien un caractère.
 Traite le placement de celui-ci dans le buffer.
}
Begin
{ Transforme en majuscules si demandé }
 If Maj then ch :=UpCase(ch);
{ Si le caractère est du type demandé...(sinon ne fait rien) }
 If ch in Typ Then
     Begin
      If Ins Then    { Si mode insertion activé }
         Begin
          Delete (Buffer,lg,1);{Supprime le dernier caractère du buffer}
          Insert (ch,Buffer,pos); { Insère le caractère entré }
          AffBuff                 { Réaffiche le buffer en entier }
         End
        Else
         Begin       { Si pas d'insertion }
          Buffer [pos] :=Ch;       { Réactualise le buffer }
          Write (ch)              { Affiche le caractère }
      End;  {If Ins}
     { Curseur à droite sauf si à l'extrémité }
```

```
      If Pos<lg then Pos :=Succ(Pos);
      GotoPos
  End
End;

PROCEDURE Init;
{
 Place la chaîne par défaut (DEFSTR) dans le buffer de travail.
 Celui-ci est complété par des '_' et affiché.
 Le curseur se place sur le premier caractère de l'entrée.
}
Begin
{ '_' par défaut }
 FillChar (Buffer,Succ(Lg),'_');
{ Copie DefStr }
 Move (DefStr,Buffer,Succ(Mem [Addr(DefStr)]));
{ Réajuste taille }
 Buffer [0] :=Chr(lg);
{ Positionne sur 1er caractère et affiche }
 Pos :=DebutBuff;
 AffBuff
End;

PROCEDURE TraiteEntree;
{
 Traite le caractère entré :
  Si c'est une touche de fonction, éxécute l'action correspondante sinon
  appelle EntreeCar pour traiter le caractère.
}
Begin
 Case Ch of
{Copy}
  #224 : Init;  { Remet la valeur initiale }
{Clr}
  #16  : If Pos>FinBuff Then Bip    { Si curseur après le dernier car. }
           Else DeLCar;             { Ne fait rien }
{Del}
  #127 : If Pos=1 Then Bip          { Ne fait rien si curseur à ...}
         Else                       { ... l'extrémité gauche }
          Begin
           Pos :=Pred (Pos);
```

```
            DeLCar
           End;
{Tab}
  #9    : Ins :=Not Ins;              { Bascule le flag insertion }
{CurG}
  #242 : If Pos=1 Then Bip           { Ne fait rien si curseur à ...}
         Else                        { ...l'extrémité gauche }
          Begin
           Pos :=Pred (Pos);
           Gotopos
          End;
{CurD}
  #243 : Begin                       { Ne fait rien si curseur à ...}
         If Pos<lg Then              { ...l'extrémité droite }
            Begin
             Pos :=Succ (Pos);
             Gotopos
            End
           Else Bip
        End
{
 Ce n'est pas une touche de fonction : appelle EntreeCar.
}
  Else EntreeCar
 End {case}
End;

Begin  {Principal}
{ Retranscrit type de caractères dans la variable interne TYP }
 Move (TypeCar,Typ,SizeOf (Typ));
{ Donne la position du curseur x,y initiale }
 Inline (
  $CD/>Firm/     { Call Firmware }
  $BB78/       {DW Txt-Get-Cursor;Résultat dans HL}
  $7C/           { Ld   A,H      }
  $32/>x/        { Ld   (x),A    }
  $7D/           { Ld   A,L      }
  $32/>y         { Ld   (y),A    }
 );
{ Retranscrit chaîne dans Buffer et l'affiche }
 Init;
```

```
{ Mode insertion désactivé par défaut }
 Ins :=False;

{ Boucle principale }
 REPEAT
  Read (Kbd,Ch); { Le caractère entré n'est pas affiché }
  TraiteEntree;
 UNTIL ch=#13;    { Jusqu'à touche RETURN }

{
 Fin de l'entrée. Reformate le résultat.
}
{ Elimine les 'blancs' aux extrémités }
 i :=DebutBuff;
 If Buffer[i]='_' then Buffer :=''    { Cas d'une entree vide }
   { Conserve seulement les caractères }
    Else Buffer :=Copy (Buffer,i,FinBuff-Pred(i));
{ Retranscrit résultat dans DEFSTR }
 Move (Buffer,DefStr,Succ(Mem [Addr(Buffer)]));
{
 Ramène curseur a la position initiale.
 Permet de recommencer au même emplacement si le r{sultat est rejeté par
 la procédure appelante.
}
 Gotoxy (x,y)
End;

PROCEDURE InputString (Lg : Byte; Maj : Boolean; VAR Def);
{
 Saisie d'une chaîne de caractères.
}
Var Typ : Set of Char;
Begin
 Typ :=[' '..#$FF];          { Tous caractères acceptés }
 Entree (Maj,lg,Typ,Def)
End;

PROCEDURE InputInteger (Lg : Byte; Mini,Maxi : Integer; VAR Def : Inte-
ger);
{
 Saisie d'une valeur entière.
```

```
}
Var Err,
    Value : Integer;
    Num   : String [7];
    Typ   : Set of Char;
Begin
 Typ :=['0'..'9','-'];       { Accepte chiffres et signe moins }
 Repeat                     { Boucle en cas de rejet }
{
 Transforme valeur en chaîne, le 0 indique que l'on ne veut pas
 d'espaces avant.
}
  Str (Def :0,Num);
  Entree (False,Lg,Typ,Num);
  Val (Num,Value,Err);      { Retransforme en valeur numérique }
{
 Teste si reconversion valide et si résultat dans la fourchette
 choisie.
}
 Until (Err=0) and (Value in [Mini..Maxi]);
 Def :=Value
End;

PROCEDURE InputReal (Lg,dec : Byte; VAR Def : Real);
{
 Saisie d'une valeur réelle.
}
VAR Err : Integer;
    Typ : Set of Char;
    Num : String [18];
    Value : Real;
Begin
{ Accepte chiffres, signe moins et point décimal }
 Typ :=['0'..'9','-','.'];
 Repeat                     { Boucle en cas de rejet }
{
 Transforme valeur en chaîne avec le nombre de décimales souhaité, le 0
 indique que l'on ne veut pas d'espaces avant.
}
  Str (Def :0 :dec,Num);
  Entree (False,Lg,Typ,Num);
```

```
{ Reconvertit en valeur réelle }
  Val (Num,Value,Err);
{ Teste si reconversion valide }
 Until Err=0;
 Def :=Value
End;
```

GESTION DE MENUS DEROULANTS

DESCRIPTION

Les fameux menus déroulants sont à la mode actuellement, ils permettent d'accéder facilement et rapidement à toutes les fonctionnalités d'un programme. Les menus d'une application ne sont plus enchaînés mais peuvent être tous atteints directement en même temps. Ils sont constitués d'une ligne en haut de l'écran où sont inscrits les titres des menus du programme, par exemple :

Fichier Edition Caractères Aide Utilitaires

Pour les utiliser, on promène à l'aide du clavier un curseur sur ces titres, le titre sélectionné apparaît en vidéo inverse par rapport aux autres. Pour choisir un menu, on appuie sur une touche et juste sous le titre apparaît (se 'déroule') le menu complet dans lequel on peut choisir une option à l'aide du clavier tout aussi simplement. La première ligne du menu est normalement réservée pour la fermeture du menu au cas où aucune option n'est choisie. Le fin du fin est de rendre ce système accessible à tout moment dans le programme et que l'affichage du menu ne perturbe pas l'écran.

Exemple :

Le menu 'Fichier' est ouvert.

```
Fichier__Edition   Caractères   Aide   Utilitaires
 *          <--- Ligne de fermeture du menu.
 Ouvrir
 Fermer
>Catalogue< <--- Option en vidéo inversée.
 Effacer
 Renommer
 Copier
```

De multiples améliorations peuvent être apportées, on peut rajouter des mots (messages, lignes de séparation) sur lesquels le curseur ne peut aller, rendre des options valides qu'à certaines conditions, indiquer quelle est l'option actuellement en cours et appeler des menus secondaires.

Exemple :

Les options en cours sont marquées avec un petit signe les précédant (généralement le signe 'racine carrée'), l'option SOULIGNE est impossible (en italiques sur l'imprimante et marquée selon votre choix à l'écran). Une ligne de tirets sépare deux zones du menu.

```
Caractères_
  *
^ Gras        <--- Les caractères gras et italiques sont actuellement
^ Italiques   <--- sélectionnés.
  Relief
 ISoulignési <--- L'option 'Souligné' est impossible à sélectionner.
 ----------  <--- Message inaccessible au curseur.
  Normal
  Moyen
  Grand
```

CARACTERISTIQUES DU PROGRAMME

Le programme fourni reprend une partie de ces caractéristiques ; selon le cas, vous pourrez le modifier pour l'adapter à vos besoins. Il est capable de gérer des options conditionnelles et de marquer des sélections.

Chaque option des menus est associée à une expression booléenne. Avant chaque appel du gestionnaire des menus, les expressions sont recalculées. Par exemple, si votre application utilise un fichier et utilise une variable booléenne OUVERT indiquant si le fichier est ouvert, en reprenant l'exemple au-dessus, on peut écrire que l'option 'Ouvert' n'est valide que si OUVERT=FALSE et l'option 'Fermer' si OUVERT=TRUE. Si l'option est invalide, il faut le faire apparaître sur l'écran. Dans le programme le nom de l'option apparaît avec des caractères altérés, cet effet est obtenu en modifiant les masques de couleur utilisés par le Firmware pour l'affichage des lettres. Pour adapter le programme aux autres ordinateurs, on peut utiliser une autre méthode.

Certaines variables définissant les menus sont initialisées une fois pour toutes et ne sont pas réactualisées au cours de l'exécution de l'application, il s'agit de la position x,y sur l'écran, de la taille du menu, des noms des options, etc. Pour gagner de la place mémoire, comme nous l'avons vu dans le chapitre consacré au TURBO PASCAL, ces variables seront définies avec des constantes typées.

Après l'appel, le gestionnaire renvoie simplement deux valeurs de type Byte, MenuH (horizontal) et MenuV (vertical) qui doivent être déclarées globales dans votre application. Elles indiquent respectivement le numéro du menu et de l'option choisis. Tous les aiguillages de votre application se feront très simplement avec des structures CASE OF..END sur ces deux variables. Vous pourrez regrouper toutes les procédures se rapportant à un menu dans une procédure de même nom que le menu, pour les gros programmes, vous pouvez placer ces procédures séparément sur le disque dans des fichiers INClude de même nom.

En reprenant l'exemple ci-dessus, on peut écrire le squelette de votre programme de cette façon :

On suppose que l'option QUITTER (fin du programme) est la 3e du menu UTILITAIRES (MenuH=5 et MenuV=3).

```
PROGRAM Exemple;
{
 Définitions des variables.
}

{$I MENU.INC}           { Gestionnaire des menus }
{$I FICHIER.INC}        { Gestion des fichiers }
{$I EDITION.INC}        { Gestion de l'édition }
{$I CARACT.INC}         { Gestion des jeux caractères }
{$I AIDE.INC}           { Gestion de l'aide }
{$I UTIL.INC}           { Gestion des utilitaires }

{
 Divers
}

Begin                   { PROGRAMME PRINCIPAL }
{
 Procédures d'initialisation si besoin.
}

 Repeat
  MenusDeroulants;      { Appel du gestionnaire des menus }
  Case MenuH of         { Aiguillage selon le menu }
   1 : Fichier;
   2 : Edition;
   3 : Graphismes;
   4 : Aide;
   5 : Utilitaires
  End;
 Until (MenuH=5) and (MenuV=3) { Ou tout autre condition à votre
choix }
End.
```

La procédure FICHIER peut s'écrire ainsi :

```
PROCEDURE Fichier;
{
```

```
 Placée dans le fichier INClude FICHIER.INC.
}

PROCEDURE Ouvrir;
Begin
{ Ouvre le fichier }
 Ouvert :=True          { Instanciation de la condition OUVERT }
End;

PROCEDURE Fermer;
Begin
{ Ferme le fichier }
 Ouvert :=False         { Instanciation de la condition OUVERT }
End;

PROCEDURE Catalogue;
Begin
{...}
End;

PROCEDURE Effacer;
Begin
{...}
End;

PROCEDURE Renommer;
Begin
{...}
End;

PROCEDURE Copier;
Begin
{...}
End;

Begin                   { Corps de la procédure }
 Case MenuV Of          { Aiguillage selon l'option dans le menu }
  1 : Ouvrir;
  2 : Fermer;
  3 : Catalogue;
  4 : Effacer;
```

```
      5 : Renommer;
      6 : Copier
     End
    End;
```

Le gestionnaire de menus détruit le contenu de l'écran ; si vous voulez éviter cela, il y a plusieurs points clés indiqués où l'on peut effectuer des opérations de sauvegarde et de rappel de l'écran ; la façon de procéder est laissée à votre choix, vous pouvez vous inspirer des procédures de manipulation de la mémoire vidéo du chapitre consacré aux 'Accès à la mémoire vidéo sous CP/M'.

Les flèches du clavier et la barre d'espacement sont utilisées pour faire avancer le curseur, un menu est ouvert et une option est validée avec la touche Return, on peut refermer rapidement un menu avec la touche ESCape.

Le programme est compatible avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité'). Néanmoins des séquences Escape propres au CP/M Plus sont utilisées (CurseurON, CurseurOFF et les inversion vidéo ESC"p" et ESC"q") ainsi que les jeux de caractères du CP/M Plus, pour que le programme fonctionne parfaitement sous CP/M 2.2 il est préférable d'installer l'émulateur des séquences Escape décrit dans le Chapitre 5 consacré aux 'Problèmes de compatiblité'.

```
{
 DESCRIPTION DES MENUS :

  Fichiers      Edition     Caracteres    Aide     Utilitaires
 *              *           *             *        *
 Ouvrir         Texte       Gras          Infos    Couleurs
 Fermer         Graphisme   Italiques              Imprimante
 Catalogue                  Relief                 Quitter
 Effacer                    Souligne
 Renommer                   Normal
 Copier                     Moyen
                            Grand

                          ^ indiquer la sélection pour ce menu.
}
```

```
CONST

 CurseurON  = ^['e';  { #3 sous CP/M 2.2 }
 CurseurOFF = ^['f';  { #2 sous CP/M 2.2 }
 NMenus     = 5;      { Nombre de menus, modifiable }
 OptMax     = 7;      { Nombre d'options maxi pour un menu, modifiable }

{ Description de la barre de menu }

 MenuPr     : Array [1..NMenus] of
              Record
               x   : Byte;       { Position X pour l'affichage }
               Nom : String [11] { Titre du menu }
              End              { 11=taille maxi des titres, modifiable }
            = (
              (x :3 ; Nom :'Fichiers'   ),
              (x :14; Nom :'Edition'    ),
              (x :24; Nom :'Caracteres' ),
              (x :37; Nom :'Aide'       ),
              (x :44; Nom :'Utilitaires')
              );

{
 Liste de tous les noms des options, les paramètres sont modifiables.
 Si un même nom doit être répété, il est inutile de l'écrire  plusieurs
 fois.
}
 Options    : Array [1..19,1..10] Of Char=    {19 mots,largeur maxi 10}
              (
       'Ouvrir    ','Fermer    ','Catalogue ','Effacer   ','Renommer  ',
       'Copier    ','Texte     ','Graphisme ','Gras      ','Italiques ',
       'Relief    ','Souligne  ','Normal    ','Moyen     ','Grand     ',

       'Infos     ','Couleurs  ','Imprimante','Quitter   '
             );

{ Description des menus }

    SsMenus : Array [1..NMenus] Of
               Record
```

```
                x,              { Position X du bord gauche }
                n,              { Nombre d'options }
                lg  : Byte;     { Largeur maxi en caractères des noms }
                Opt : Array [1..OptMax] Of Byte
                                {
                                  Index dans la table OPTIONS.
                                  0 n'est pas pris en compte.
                                }
              End
            = (
              (x :1 ; n :6; lg :9 ; Opt :(1 ,2 ,3 ,4 ,5 ,6 ,0 )),
              (x :12; n :2; lg :9 ; Opt :(7 ,8 ,0 ,0 ,0 ,0 ,0 )),
              (x :22; n :7; lg :9 ; Opt :(9 ,10,11,12,13,14,15)),
              (x :35; n :1; lg :5 ; Opt :(16,0 ,0 ,0 ,0 ,0 ,0 )),
              (x :42; n :3; lg :10; Opt :(17,18,19,0 ,0 ,0 ,0 ))
              );

VAR
{ Conditions de validation des options }
 OptCond   : Array [1..NMenus,1..OptMax] Of Boolean;
{ Conditions pour indiquer les options en cours }
 OptSelect : Array [1..NMenus,1..OptMax] Of Boolean;

{ Résultats de la sélection des menus }
 MenuH,MenuV : Byte;

{
 Exemples de variables gérées par l'application de l'utilisateur.
 ( Pour démonstration seulement )
}
{ Gestion des caractères }
 Gras,Italiques,Relief,Souligne,Normal,Moyen,Grand,
{ Gestion des fichiers }
 FicOuvert : Boolean;

PROCEDURE ModifieCar;
{
 Altère le masque des couleurs pour l'affichage des options impossibles.
}
Begin
{--- VERSION CP/M PLUS ---}
```

```
 Inline (
  $CD/>Firm/    { Call Firmware  ;Permet l'accès à la banque 0 de RAM}
  $FC5D       {DW Adr.routine    ;Appelle la routine qui modifie l'encre}
 )
{---VERSION CP/M 2.2 ---}
 Inline (
  $3E/$55/      { Ld   A,$55     ;Pour 80 colonnes, mettre  $50  pour  40
                                  colonnes}
  $32/>0        { Ld   (Adr),A   ;L'adresse est initialisée dans
 )                                 InitMenus}
End;

PROCEDURE InitMenus;
{
 Initialisations pour utiliser les menus déroulants.
}
Begin
{--- VERSION CP/M PLUS ---}
{
 Installe la routine   FC5D : Ld   A,$55
                       FC5F : Ld   (adresse),A
                       FC62 : RET
 en zone non commutée de la m{moire.
 ADRESSE est l'adresse de la variable Firmware des masques de couleurs.
}
{ LD   A,$55 }
 Mem [$FC5D] :=$3E;
 Mem [$FC5E] :=$55; { Pour 80 colonnes, mettre $50 pour 40 colonnes }
{ LD   (xxxx),A }
 Mem [$FC5F] :=$32;
{ RET }
 Mem [$FC62] :=$C9;
{ Recherche la valeur de ADRESSE }
 Inline (
  $CD/>Firm/    { Call Firmware   }
  $BB93/      {DW Txt-Get-Pen     }
  $CD/>Firm/    { Call Firmware   }
  $BB90/      {DW Txt-Set-Pen     ;Donne HL=adresse variable système
                                   du masque de l'encre}
  $22/>$FC60    { Ld   ($FC60),HL ;Place dans routine}
 );
```

```
 MenuH :=1;      { Premier titre de menu par d{faut }

{--- VERSION CP/M 2.2 ---}

{ Recherche l'adresse de la variable Firmware du masque de l'encre }
 Inline (
  $CD/>Firm/    { Call Firmware    }
  $BB93/      {DW Txt-Get-Pen      }
  $CD/>Firm/    { Call Firmware    }
  $BB90/      {DW Txt-Set-Pen      ;Donne HL=adresse variable système
                                    du masque de l'encre}
  $22/>ModifieCar+3
               { Ld (xxxx),HL      ;Place dans routine ModifieCar}
 );

 MenuH :=1       { Premier titre de menu par défaut }
End;

PROCEDURE RemetCar;
{
 R{tablit les masques normaux des couleurs (voir ci-dessus).
 La routine fait un PAPER 0, PEN 1 .Vous pouvez modifier les paramètres
 selon les valeurs utilisées dans votre application.
}
Begin
 Inline (
  $3E/0/        { Ld   A,$00      ;PAPER 0}
  $CD/>Firm/    { Call Firmware   }
  $BB96/      {DW Txt-Set-Paper   }
  $3E/1/        { Ld   A,$01      ;PEN 1}
  $CD/>Firm/    { Call Firmware   }
  $BB90       {DW Txt-Set-Pen     }
 )
End;

{--- Gestion de la barre de sélection principale ---}

PROCEDURE SeLMenuPr (n :Byte; Select :Boolean);
{
 Affiche la sélection N, si Select est TRUE (en vidéo normale).
```

```
 Si la sélection doit être cachée (Select=FALSE), affiche en vidéo
 inverse.
}
Begin
 If Not Select Then Write (^['p');
 With MenuPr [n] Do
  Begin
   Gotoxy (Pred (x),1);
   Write (' ',Nom,' '^['q')
  End
End;

PROCEDURE OuvreMenuPr;
{
 Sauve ici 1ère ligne d'ecran et affiche la barre de sélection.
}
Var i :Byte;
Begin
{
 Sauver 1}re ligne d'{cran ici.
}
{ Met la première ligne d'écran en vidéo inverse }
 Gotoxy (1,1);
 Write (^['p','' :80,^['q');
{ Place tous les titres des menus }
 Gotoxy (1,1);
 For i :=1 to NMenus do SeLMenuPr (i,False)
End;

PROCEDURE FermeMenuPr;
{
 Rétablit ici 1}re ligne d'écran ou toute autre action au choix.
}
Begin
 {...}
End;

{--- Instanciation des conditions ---}

PROCEDURE Conditions;
Begin
```

```
{
 Associer ici à chaque option des menus une expression booléenne.
}
{ Exemples : }
{ tout FALSE par défaut }
 FillChar (OptCond,SizeOf(OptCond),False);
{ Menu FICHIER, option OUVRIR }
 OptCond [1,1] :=Not FicOuvert;
{ Menu FICHIER, option FERMER }
 OptCond [1,2] :=FicOuvert;
{ Option Quitter seulement si fichier fermé }
 OptCond [5,3] :=Not FicOuvert

{
 Pour les options marquées, il faut que OptSelect soit TRUE.
}
{ Exemples : Seul le menu CARACTERES doit avoir les options marquées }
{ Tout à False par défaut }
 FillChar (OptSelect,SizeOf (OptSelect),False);
{
 Menu Caractères, les options sont marquées selon les expressions
 booléennes associées.
}
 OptSelect [3,1] :=Gras;
 OptSelect [3,2] :=Italiques;
 OptSelect [3,3] :=Relief;
 OptSelect [3,4] :=Souligne;
 OptSelect [3,5] :=Normal;
 OptSelect [3,6] :=Moyen;
 OptSelect [3,7] :=Grand
End;

{--- Corps du programme ---}

PROCEDURE MenusDeroulants;
VAR i    : Byte; { Divers }
    Car : Char; { Pour lecture du clavier }

PROCEDURE SousMenu;
VAR Choix,              { Choix dans le menu }
    xz,                 { Position x du bord gauche du menu }
```

```
    lz,                { Largeur en caract}res des noms des options x }
    nz : Byte;         { Nombre d'options }

PROCEDURE SeLSsMenu (n : Byte; Select : Boolean);
{
 Affiche l'option N du menu ouvert si Select est TRUE (en vidéo normale)
 Si l'option doit être cachée (Select=FALSE), elle est en vidéo inverse.
 Si N=0, affiche la ligne de fermeture du menu.
}
Begin
 Gotoxy (Succ(xz),4+n);
 If Not Select then Write (^['p');     { Vidéo inverse si option cachée }
 If n=0 Then Write (' '#$FF,'' :lz,^['q') { Ligne de fermeture du menu }
    Else
     Begin                                  { Sinon, option }
      { Marquage n{cessaire ? }
       If OptSelect [MenuH,n] Then Write ('>')   { Marque }
                              Else Write (' ');
      { Caractères altérés si option impossible }
       If Not OptCond [MenuH,n] Then ModifieCar;
      { Affiche l'option }
       Write (Copy (Options [SsMenus[MenuH].Opt[n]],1,lz),' '^['q');
      { Rétablit encre altérée (Note : Même si pas altérée) }
       RemetCar
     End {else}
End;

PROCEDURE OuvreSsMenu;
{
 Sauve écran ici, affecte variables internes et affiche sous-menu.
}
Begin
{
 Affecte les variables NZ,LZ,XZ avec les valeurs tirées de
 l'enregistrement SsMenus : permet une manipulation plus rapide des ces
 données.
}
 With SsMenus [MenuH] Do
  Begin
   nz :=n;
   lz :=lg;
```

```
   xz :=x
  End;

{ Sauver ici la portion d'écran qui sera effacée }

{ Afficher le cadre du menu }
{ Ligne du haut }
 Gotoxy (xz,2);write (#$96);
 For i :=1 to lz+2 do Write (#$9A);
 Write (#$9C);

{ Première ligne (vide) }
 Gotoxy (xz,3);Write (#$95^['p','' :lz+2,^['q'#$85);

{ Bords du cadre }
 For i :=4 To nz+4 Do
  Begin
   Gotoxy (xz,i);Write (#$95);
   Gotoxy (xz+lz+3,i);Write (#$85)
  End;

{ Ligne du bas }
 Gotoxy (xz,5+nz);Write (#$93);
 For i :=1 to lz+2 do Write (#$8A);
 Write (#$89);

{
 Affiche toutes les options du menu.
 La ligne de fermeture n'est pas affichée à ce moment là car elle sera
 automatiquement affichée ensuite lors de la boucle de sélection dans le
 menu.
}
 For i :=1 To nz Do SeLSsMenu (i,False)
End;

PROCEDURE FermeSsMenu;
{
 Efface le menu.
 Ou remettre ici le contenu de l'écran.
}
VAR i : Byte;
```

```
Begin
{ Met des 'espaces' à la place du menu }
 For i :=2 To nz+5 Do
  Begin
   Gotoxy (xz,i);
   Write ('' :lz+4)
  End
End;

PROCEDURE ChoixSsMenu;
{
 Sélection dans le menu ouvert.
}
Begin
{
 Boucle de sélection. Sortie quand CAR=#254 (Inaccessible au clavier).
}
 Repeat
  SeLSsMenu (Choix,True);                { Montre la sélection }
  Read (Kbd,Car);
  Case Car Of
{ Descendre }
  #241,' ' : Begin
             SeLSsMenu (Choix,False);    { Cache la sélection }
             If Choix=nz Then Choix :=0
                         Else Choix :=Succ(Choix)
            End;
{ Monter }
     #240 : Begin
             SeLSsMenu (Choix,False);    { Cache la sélection }
             If Choix=0 Then Choix :=nz
                        Else Choix :=Pred(Choix)
            End;
{ ESCape }
     #252 : Begin
              Choix :=0;  { Simule la ligne de fermeture }
              Car :=#254  { Code 'Fin de sélection' }
             End;
{ RETURN }
      #13 : {
             Valide la sélection seulement si l'option est 'possible'
```

```
            ou si l'on veut fermer le menu (Choix=0).
           }
           If (Choix=0) Or OptCond [MenuH,Choix] Then Car :=#254
  End;
 Until Car=#254;

 FermeSsMenu          { Ferme (efface) le menu }
End;

{--- Procédure principale de gestion d'un menu ouvert ---}

Begin   {SOUSMENU}

 Choix :=0;           { Curseur sur ligne de fermeture par défaut }

 Repeat               {Boucle principale}
  OuvreSsMenu;        { Ouvre le menu et... }
  ChoixSsMenu;        { ...fait la sélection... }
 Until Car=#254;      { ...jusqu'à une sélection valide. }

 MenuV :=Choix        { Affecte ici la valeur  de  sélection
'verticale'
                       retournée par MenuSDeroulants }
End;

{--- Corps de la procédure MenusDeroulants ---}

Begin
 Write (CurseurOFF);       { Désactive le curseur }
 Conditions;               { Actualise les conditions }
 OuvreMenuPr;              { Affiche la barre de sélection principale }
 MenuV :=0;

 REPEAT                    { Sélection dans la barre }
  SeLMenuPr (MenuH,True); { Affiche sélection en cours }
  Read (Kbd,Car);
  Case Car of
{ A droite }
 ' ',#243 : Begin
             SeLMenuPr (MenuH,False);      { Cache sélection }
             If MenuH=NMenus then MenuH :=1
```

```
                              else MenuH :=Succ(MenuH)
            End;
{ A gauche }
     #242 : Begin
             SeLMenuPr (MenuH,False);      { Cache sélection }
             If MenuH=1 then MenuH :=NMenus
                        else MenuH :=Pred(MenuH)
            End;
{ RETURN }
      #13 : SousMenu    é Ouvre un menu et le gère }
  End;   {Case}
 UNTIL MenuV<>0;        { Arrt quand une option valide d'un menu est
                          choisie.}

{ Ferme la barre de sélection principale }
 FermeMenuPr;
{ Active le curseur (ou non selon le cas de votre application) }
 Write (CurseurON)
End;

{--- EXEMPLE D'UTILISATION DE LA PROCEDURE MENU-DEROULANTS ---}

Begin
 InitMenus;    { A appeler avant toute utilisation de Menus-Déroulants }

 REPEAT
  MenusDeroulants;
{ Affiche les données de la sélection }
  Gotoxy (1,20);
  Write (MenuH :2,'   ',MenuV :2)
 UNTIL (MenuH=5) and (MenuV=3)   { option QUITTER }
End.
```

PROCEDURES GRAPHIQUES

Nous allons étudier maintenant quelques procédures pour les manipulations des graphismes. Elles pourront par exemple servir de base pour l'écriture d'un éditeur graphique.

SAISIE DE L'ORGANE D'ENTREE

Il faut d'abord s'attarder sur le problème de la saisie du joystick ou du clavier pour la manipulation du curseur graphique. La FUNCTION suivante scrute le joystick et le clavier et renvoie un résultat de type Byte dont les bits sont utilisés de la façon suivante :

Bit 7 : ESCAPE
Bit 6 : ESPACE ou bouton du joystick supplémentaire
Bit 5 : Fire 1 ou Control
Bit 4 : Fire 2 ou Control
Bit 3 : Droite
Bit 2 : Gauche
Bit 1 : Bas
Bit 0 : Haut

Un bit à 1 signifie que la touche est appuyée.

Le joystick fonctionne dans les 8 positions possibles ; on peut aussi utiliser le pavé numérique de façon analogue au joystick, la touche CONTROL étant utilisée à la place du bouton Fire. Les touches ESCAPE et ESPACE sont aussi testées et pourront par exemple signaler un abandon ou une validation dans un programme.

Configuration du pavé numérique :

f7 f8 f9
f4 f6
f1 f2 f3

La fonction est compatible avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité'). Elle utilise la routine Firmware KM-TEST-KEY qui regarde si la touche dont le numéro est fourni dans A est appuyée, si oui Z=0 sinon Z=1, le registre C contient aussi l'état de SHIFT (bits 5 à 1) et CONTROL (bits 7 à 1).

```
FUNCTION LocateurGr : Byte;
VAR CurG : Byte;              { contiendra le résultat }

PROCEDURE PaveNum;
{
 Table pour la gestion du pavé numérique.
}
Begin
 Inline (
 {f7 f8 f9 f6 f3 f1 f2 f4}
  10/11/3 /4 /5 /13/14/20/  {Codes des touches du pavé numérique}
  5 /1 /9 /8 /10/6 /2 /4 /  {Configuration de bits correspondante }
  0                         {Configuration si aucune touche appuyée }
 )
End;

Begin
 Inline (
  $16/0/          { Ld   D,$00     ;Contiendra le résultat }
                  { TEST DE ESCAPE }
  $3E/66/         { Ld   A,&66     ;Numéro de la touche ESCAPE }
  $CD/>Firm/      { Call Firmware  ;Teste ESCAPE }
  $BB1E/        {DW Km-Test-Key    }
  $28/2/          { Jr   Z,L00     ;Pas appuyée }
  $16/$80/        { Ld   D,$80     ;Appuyée, met le Bit 7 du résultat à 1 }
                  { TEST DE ESPACE }
  $3E/47/     {L00 :Ld   A,&47     ;Numéro de la touche ESPACE }
  $CD/>Firm/      { Call Firmware  ;Teste ESPACE }
  $BB1E/        {DW Km-Test-Key    }
  $28/2/          { Jr   Z,L01     ;Pas appuyée }
  $CB/$F2/        { Set  6,D       ;Appuyée, met le Bit 6 du résultat à 1 }
                  { SAISIE DU JOYSTICK }
  $CD/>Firm/ {L01 :Call Firmware }
  $BB24/        {DW Km-Get-Joystick;Place un résultat dans les 6  premiers
                                    bits de A selon la table décrite plus
                                    haut }
  $B2/            { Or   D         ;Fusionne avec le résultat  du  test  de
                                    ESCAPE et ESPACE }
  $57/            { Ld   D,A       ;Résultat dans D }
  $E6/$3F/        { And  $3F       ;Le joystick était-il manipulé ? }
  $7A/            { Ld   A,D       ;Reprend résultat }
```

```
  $20/30/        { Jr   NZ,L05   ;Le joystick était manipulé, ne teste
                                  pas le pavé numérique }
                 { TEST DU PAVE NUMERIQUE }
  $06/8/         { Ld   B,$08    ;Nombre de touches à tester}
  $21/>PaveNum/  { Ld   HL,PaveNum;HL sur début table }
  $7E/         {L02 :Ld   A,(HL)   ;Numéro de la touche dans A }
  $E5/           { Push HL        ;Sauve pointeur }
  $CD/>Firm/     { Call Firmware }
  $BB1E/       {DW Km Test Key    ;Teste la touche }
  $E1/           { Pop  HL        ;Récupère pointeur }
  $20/3/         { Jr   NZ,L03    ;La touche est appuyée }
  $23/           { Inc  HL        ;Elle n'est pas appuyée, pointe num{ro
                                  suivant }
  $10/<-13/      { Djnz L02       ;boucle }
                 {                }
                 { Une touche du pavé numérique est appuyée }
                 { ou aucune touche n'est appuyée }
                 { TEST DE CONTROL }
  $7A/         {L03 :Ld   A,D      ;Résultat dans A }
  $CB/$11/       { RL   C         ;ControL ? (B7 de C à 1) }
  $30/2/         { Jr   NC,L04    ;Non }
  $F6/$30/       { Or   $30       ;Oui, met à 1 les bits 4 et 5 (Fire) }
                 {
                   Place la configuration de bits correspondante à la
                   touche appuyée du pavé numérique.
                 }
  1/>8/        {L04 :Ld   BC,$08   ;Offset dans table }
  9/             { Add  HL,BC     ;HL pointe sur les config. de bits }
  $B6/           { Or   (HL)      ;Fusionne avec résultat }
  $32/>CurG    {L05 :Ld   (CurG),A ;Place résultat dans variable résultat
}
 );
 LocateurGr :=CurG
End;
```

GRAPHISMES

Nous allons maintenant étudier quelques procédures pour la création de rectangles, d'ellipses ainsi qu'une procédure de remplissage de zone (FILL).

Rectangles

La procédure suivante trace un rectangle dont un des côtés est de coordonnées x,y et le côté opposé x1,y1. Les instructions équivalentes en BASIC sont :

```
MOVE x,y
DRAW x,y1
DRAW x1,y1
DRAW x1,y
DRAW x,y
```

La procédure est compatible avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité'). Elle utilise les routines Firmware GRA MOVE ABSOLUTE et GRA LINE ABSOLUTE qui nécessitent les coordonnées x et y dans DE et HL.

```
PROCEDURE Rectangle (x,y,x1,y1 : Integer);
Begin
 Inline (
                { MOVE X,Y       }
  $ED/$5B/>x/   { Ld   DE,(x)    }
  $2A/>y/       { Ld   HL,(y)    }
  $D5/          { Push DE        ;Sauve x }
  $CD/>Firm/    { Call Firmware  }
  $BBC0/       {DW Gra-Move-Absolute }
                { DRAW X,Y1      }
  $D1/          { Pop  DE        ;DE=x }
  $2A/>y1/      { Ld   HL,(y1)   }
  $E5/          { Push HL        ;Sauve y1 }
  $CD/>Firm/    { Call Firmware  }
  $BBF6/       {DW Gra-Line-Absolute }
```

```
                { DRAW X1,Y1     }
  $E1/          { Pop  HL        ;HL=y1 }
  $ED/$5B/>x1/  { Ld   DE,(x1)   }
  $D5/          { Push DE        ;Sauve x1 }
  $CD/>Firm/    { Call Firmware  }
  $BBF6/       {DW Gra-Line-Absolute }
                { DRAW X1,Y      }
  $D1/          { Pop  DE        ;DE=x1 }
  $2A/>y/       { Ld   HL,(y)    }
  $E5/          { Push HL        ;Sauve y }
  $CD/>Firm/    { Call Firmware  }
  $BBF6/       {DW Gra-Line-Absolute }
                { DRAW X,Y       }
  $E1/          { Pop  HL        ;HL=y }
  $ED/$5B/>x/   { Ld   DE,(x)    }
  $CD/>Firm/    { Call Firmware  }
  $BBF6        {DW Gra-Line-Absolute }
 )
End;
```

Ellipses

La procédure suivante trace une ellipse, un cercle ou un polygone. Elle permet aussi de ne tracer qu'une partie de ces figures. Voici le rôle de chacun de ses arguments :

```
   Xc,Yc    : Coordonnées du centre.
   Rh,Rv    : Rayon horizontal et vertical.
 AngD,AngF  : Angles de début et de fin de tracé.
```

Ces deux arguments permettent le tracé d'arcs d'ellipses. Les angles sont en degrés et sont comptés à partir de l'angle plat horizontal situé à 'droite' du centre et dans le sens contraire des aiguilles d'une montre. Les valeurs sont comprises entre 0 et 720 et AngG ne peut être supérieur à AngF. Le tracé complet d'une ellipse est obtenu par exemple avec AngD=0 et AngF=360.

Step : Incrément des angles.

Pour tracer la figure, la procédure incrémente l'angle de début avec la valeur de Step jusqu'à ce que l'angle de fin soit atteint et relie chacun des points correspondants par un segment de droite. Avec une valeur Step élevée, la procédure trace des polygones, avec une valeur petite, les côtés du polygone sont si petits que l'on a l'impression d'une courbe continue. Par exemple, si Step=90, un losange ou un carré est tracé.

Ligne : Type de tracé.

Dans certaines applications telles que les éditeurs graphiques, le dessin de l'ellipse peut être déformé à volonté avant le tracé définitif. Le tracé est donc exécuté plusieurs fois et pour gagner en rapidité, on peut demander que seuls les points au sommet du polygone soient tracés, il faut que LIGNE soit FALSE. Cette possibilité combinée avec un argument Step assez élevé permet de faire apparaître une ébauche du dessin définitif.

La formule utilisée pour le tracé de l'ellipse est :

ANGLE varie de AngD à AngF avec un incrément STEP.

```
x :=Xc+Rh*Cosinus (Angle)
y :=Yc+Rv*Sinus (Angle)
```

A titre d'information, si l'on désire que l'ellipse soit inclinée d'un angle ROT à l'écran, il faut modifier le programme et utiliser les formules suivantes :

```
x :=Xc+(Rh*Cosinus(Angle)*Cosinus(Rot)-Rv*Sinus(Angle)*Sinus(Rot))
y :=Yc+(Rh*Cosinus(Angle)*Sinus(Rot)+Rv*Sinus(Angle)*Cosinus(Rot))
```

Pour des raisons de rapidité, les calculs de SINUS, COSINUS ainsi que les multiplications et divisions sont effectués en langage machine sans utiliser les routines arithmétiques du Firmware. Indépendamment du programme, il peut être intéressant que vous examiniez la technique employée pour l'utiliser à d'autres fins.

```
PROCEDURE Ellipse (Ligne : Booléan; Xc,Yc,Rh,Rv,Step,AngD,AngF : Inte-
ger);
VAR xPt,yPt,          { Coordonnées de travail }
    xDp,yDp : Integer; { Sauvegarde des  coordonnées  du  premier  point
```

```
                        placé }

PROCEDURE NCotes;
{
 Calcule le nombre de points à tracer (c.-à-d. le nombre de côtés du
 polygone) et l'incrément d'angle pour le dernier point.

 En entrée :
  HL=différence d'arc (1 à 359).
  DE=Pas (Step) en degrés.
 La procédure effectue l'opération : HL :=HL DIV DE, DE :=reste.
 En sortie :
  HL=nombre de points à tracer.
  DE=incrément d'angle pour le dernier point.

 Registres modifiés :
  A,BC

 L'opérande HL est placé dans AC. Les bits du résultat de la division
 seront entrés 1 à 1 par la droite de AC. A la fin le résultat est remis
 dans HL, HL étant utilisé pour le calcul du reste.
}
Begin    { NCotes }
 Inline (
  $7C/          { Ld   A,H       ;AC=opérande }
  $4D/          { Ld   C,L       }
  $21/>$00/     { Ld   HL,$0000 ;HL=reste, initialisé à 0 }
  $06/16/       { Ld   B,&16     ;16 bits de l'opérande à tester }

  $CB/$11/   {L00 :RL   C         ;Teste bit à gauche de l'opérande }
  $17/          { RLA             }
  $ED/$6A/      { Adc  HL,HL     ;Décale reste à gauche.+1 si bit à 1 }
                { Reste >= Opérateur ? }
  $ED/$52/      { Sbc  HL,DE     ;Reste :=Reste-Opérateur }
  $30/$01/      { Jr   NC,L01    ;Si oui }
  $19/          { Add  HL,DE     ;Non, rétablit reste }
  $3F/       {L01 :Ccf            ;CY=1 si reste >= opérateur, alors
                                  placera dans résultat AC }
  $10/<-13/     { Djnz L00       }

  $CB/$11/      { RL   C         ;Corrige résultat (multiplie par 2) }
```

```
  $17/          { RLA           }
  $EB/          { Ex   DE,HL    ;Place reste dans DE }
  $67/          { Ld   H,A      ;Place résultat dans HL }
  $69           { Ld   L,C      }
 )
End;     { NCotes }

PROCEDURE Mult;
{
 Effectue l'opération :
  HL=HL*DE DIV 256.

 Permet le calcul de Rh*Cosinus (Angle) et Rv*Sinus (Angle).
 La division par 256 est nécessaire car les valeurs de Sinus et Cosinus
 sont multipliées par 255. (Note : Petite erreur ici car ce n'est pas
par
 256).

 HL est transféré dans C et A, le résultat est placé dans 3 octets :
  (IX+00),H,L
 La multiplication est posée comme si elle était faite à la main.
 Exemple :

      1001  (9)        Les 16 bits de l'opérateur sont testés 1 à 1 du
    * 1010  (10)       plus fort au  plus  faible. A chaque test, le
   -------             résultat est décalé sur la gauche. Si le bit est
      0000             à 1, l'opérande est ajouté au résultat.
  +  1001
  + 0000
  +1001
   -------
   1011010  (90)
}
VAR Temp : Byte; { Pour stocker 1er octet du résultat }
Begin    { Mult }
 Inline (
  $DD/$21/>Temp/{ Ld   IX,Temp  ;IX pointe sur 1er octet du résultat }
  $7D/          { Ld   A,L      }
  $4C/          { Ld   C,H      ;Opérateur=CA}
                { Initialise résultat à 0 }
  $21/>$00/     { Ld   HL,$0000 }
```

```
  $DD/$75/$00/  { Ld   (IX+00),L}
                { Boucle de calcul }
  $06/16/       { Ld   B,16     ;16 bits de CA à tester}
                { Décale résultat sur la gauche }
  $29/        {L20 :Add  HL,HL    }
  $DD/$CB/0/$16/{ RL   (IX+00)  ;(Passe retenue) }
                { Teste bits de l'opérateur en le décalant à gauche }
  $17/          { RLA           }
  $CB/$11/      { RL   C        }
  $30/$06/      { Jr   NC,L21   ;Si bit à 0, ne fait rien }
  $19/          { Add  HL,DE    ;Si bit à 1, Résultat :=Rés.+Opérateur }
  $30/$03/      { Jr   NC,L22   ;Passe l'éventuelle retenue dans le ...}
  $DD/$34/$00/  { Inc  (IX+00)  ;3}me octet du résultat }
  $10/<-18/   {L21 :Djnz L20     }
                {               }
                { Prend les deux octets de poids fort du résultat.
                  Ce qui donne une division par 256 }
  $6C/          { Ld   L,H      }
  $DD/$66/$00   { Ld   H,(IX+0) ;HL=Résultat }
 )
End;     { Mult }

PROCEDURE SinCos;
{
 Calcule les expressions :
 En entr{e :
  HL=Angle entre 0 et 90.
 En sortie :
  DE=Rh*Cosinus (Angle)
  BC=Rv*Sinus (Angle)
}
PROCEDURE Table;
{
 Table des Sinus et Cosinus des angles entre 0 et 90.

 Les valeurs réelles sont multipiées par 255 pour être codées sur un oc-
tet.
 Les valeurs sont dans l'ordre pour les Cosinus. Pour les Sinus, il faut
 lire la table à l'envers.
}
Begin    { Table }
```

```
 Inline (
$FF/$FF/$FF/$FF/$FF/$FF/$FE/$FE/$FD/$FC/$FC/$FB/$FA/$F9/$F8/$F7/$F6/$F4/
$F3/$F2/$F0/$EE/$ED/$EB/$E9/$E8/$E6/$E4/$E2/$DF/$DD/$DB/$D9/$D6/$D4/$D1/
$CF/$CC/$C9/$C6/$C4/$C1/$BE/$BB/$B8/$B5/$B1/$AE/$AB/$A7/$A4/$A1/$9D/$9A/
$96/$92/$8F/$8B/$87/$83/$7F/$7C/$78/$74/$70/$6C/$68/$64/$5F/$5B/$57/$53/
$4F/$4A/$46/$42/$3D/$39/$35/$30/$2C/$28/$23/$1F/$1A/$16/$11/$0D/$08/$04/
$01
 )
End;     { Table }

Begin    { SinCos }
 Inline (
  $E5/           { Push HL       ;Sauve angle }
                 { Effectue DE=Cosinus (HL) }
  $01/>Table/    { Ld   BC,Table ;Sur début table (0 degrés) }
  $09/           { Add  HL,BC    ;HL pointe sur le Cosinus }
  $5E/           { Ld   E,(HL)   ;Saisit Cosinus dans DE (255 à 1) }
  $16/$00/       { Ld   D,$00    }
                 { Effectue HL=Rh*Cosinus }
  $2A/>Rh/       { Ld   HL,(Rh)  ;HL=Rh }
  $CD/>Mult/     { Call Mult     }
  $EB/           { Ex   DE,HL    ;DE=Rh*Cosinus }

  $C1/           { Pop  BC       ;Récupère Angle }
  $21/>Table+90/{ Ld   HL,Table+90 ;Sur dernier élément de la table }
  $B7/           { Or   A        }
  $ED/$42/       { Sbc  HL,BC    ;HL pointe sur le Sinus }
  $D5/           { Push DE       ;Sauve valeur calculée précédement }
  $5E/           { Ld   E,(HL)   ;Saisit Sinus dans DE (1 à 255) }
  $16/$00/       { Ld   D,$00    }
                 { Effectue HL=Rv*Sinus }
  $2A/>Rv/       { Ld   HL,(Rv)  ;HL=Rv }
  $CD/>Mult/     { Call Mult     }
  $44/           { Ld   B,H      }
  $4D/           { Ld   C,L      ;BC=Rh*Sinus }
  $D1            { Pop  DE       ;Récupère premi}re valeur }
 )
End;     { SinCos }

PROCEDURE Wxy;
{
```

```
Calcule les coordonnées x,y d'un point (dans DE,HL) selon l'Angle (dans
HL) et Xc,Yc.
Selon la valeur de l'angle, utilise les formules suivantes :

 ---------------------------------------------------------------
         |                          |
  ANGLE  |          X               |          Y
 --------|--------------------------|--------------------------
         |                          |
 0  -89  | Xc+Rh*Cos (Angle)        | Yc+Rv*Sin (Angle)
 90 -179 | Xc-Rh*Cos (180-Angle)    | Yc+Rv*Sin (180-Angle)
 180-269 | Xc-Rh*Cos (Angle-180)    | Yc-Rv*Sin (Angle-180)
 270-359 | Xc+Rh*Cos (360-Angle)    | Yc-Rv*Sin (360-Angle)
 ---------------------------------------------------------------

Si l'angle est supérieur à 359, la procédure lui soustrait 360 de façon
à obtenir une valeur entre 0 et 359.
}
Begin
 Inline (
  $11/>90/       { Ld   DE,&90    ;90 degrés }
  $E5/           { Push HL        ;Sauve Angle }
  $B7/           { Or   A         }
  $ED/$52/       { Sbc  HL,DE     ;Angle-90 }
  $30/<14/       { Jr   NC,L10    ;si Angle >= 90 }
  $E1/           { Pop  HL        ;* HL=Angle entre  0-89 }
  $CD/>SinCos/   { Call SinCos    }
  $2A/>Xc/       { Ld   HL,(Xc)   }
  $19/           { Add  HL,DE     }
  $EB/           { Ex   DE,HL     ;X=Xc+DE }
  $2A/>Yc/       { Ld   HL,(Yc)   }
  $09/           { Add  HL,BC     ;Y=Yc+BC }
  $C9/           { Ret            }
                 {                }
  $B7/        {L10 :Or   A         }
  $ED/$52/       { Sbc  HL,DE     ;Angle-180 }
  $30/<22/       { Jr   NC,L11    ;Si Angle >= 180 }
  $D1/           { Pop  DE        ;* DE=Angle entre 90-179 }
  $21/>180/      { Ld   HL,&180   }
  $B7/           { Or   A         }
  $ED/$52/       { Sbc  HL,DE     ;HL=Angle entre 90-1 }
  $CD/>SinCos/   { Call SinCos    }
```

```
$2A/>Xc/        { Ld   HL,(Xc)   }
$B7/            { Or   A         }
$ED/$52/        { Sbc  HL,DE     }
$EB/            { Ex   DE,HL     ;X=Xc-DE }
$2A/>Yc/        { Ld   HL,(Yc)   }
$09/            { Add  HL,BC     ;Y=Yc+BC }
$C9/            { Ret            }
                {                }
$B7/        {L11 :Or   A          }
$ED/$52/        { Sbc  HL,DE     ;Angle-270 }
$30/<24/        { Jr   NC,L12    ;Si Angle >= 270 }
$E1/            { Pop  HL        ;* HL=Angle entre 180-269 }
$11/>180/       { Ld   DE,&180   }
$B7/            { Or   A         }
$ED/$52/        { Sbc  HL,DE     ;HL=Angle entre 0-89 }
$CD/SinCos/     { Call SinCos    }
$2A/>Xc/        { Ld   HL,(Xc)   }
$B7/            { Or   A         }
$ED/$52/        { Sbc  HL,DE     }
$EB/            { Ex   DE,HL     ;X=Xc-DE }
$2A/>Yc/        { Ld   HL,(Yc)   }
$B7/            { Or   A         }
$ED/$42/        { Sbc  HL,BC     ;Y=Yc-BC }
$C9/            { Ret            }
                {                }
$B7/        {L12 :Or   A          }
$ED/$52/        { Sbc  HL,DE     ;Angle-360 }
$30/<22/        { Jr   NC,L13    ;Si angle >= 360 }
$D1/            { Pop  DE        ;* DE=Angle entre  270-359 }
$21/>360/       { Ld   HL,&360   }
$B7/            { Or   A         }
$ED/$52/        { Sbc  HL,DE     ;HL=Angle entre 90-1 }
$CD/>SinCos/    { Call SinCos    }
$2A/>Xc/        { Ld   HL,(Xc)   }
$19/            { Add  HL,DE     }
$EB/            { Ex   DE,HL     ;X=Xc+DE }
$2A/>Yc/        { Ld   HL,(Yc)   }
$B7/            { Or   A         }
$ED/$42/        { Sbc  HL,BC     ;Y=Yc-BC }
$C9/            { Ret            }
                {                }
```

```
  $E1/        {L13 :Pop  HL        ;HL=Angle, il est >= 360 }
  $11/>360/       { Ld   DE,&360   }
  $B7/            { Or   A         }
  $ED/$52/        { Sbc  HL,DE     ;Retire 360 degr{s }
  $C3/>Wxy        { Jp   Wxy       ;Recommence le travail }
 )
End;      { Wxy }

{--- Corps de la procédure ELLIPSE ---}

Begin
 Inline (
                 { Teste si Angle début <= Angle fin }
  $2A/>AngF/     { Ld   HL,(AngF)}
  $ED/$5B/>AngD/{ Ld   DE,(AngD)}
  $B7/           { Or   A         }
  $ED/$52/       { Sbc  HL,DE     }
  $D8/           { Ret  C         ;Retour immédiat si AngD > AngF }
  $F5/           { Push AF        ;Sauve flags (Z=1 si AngD = AngF) }

                 { Calcule le nombre de points et le dernier angle }
  $20/$03/       { Jr   NZ,L00    ;Si AngD<>AngF}
  $21/>359/      { Ld   HL,359    ;Si AngD=AngF. Cas d'un tour complet, la
                                   différence d'arc est de 359 }
$ED/$5B/>Step/{L00 :Ld  DE,(Step)}
  $CD/>NCotes/   { Call NCotes    ;HL=nombre de points, DE=dernier angle }
  $E5/           { Push HL        ;Sauve nombre de points }
                 { Recherche coordonnées du premier point à tracer }
  $2A/>AngD/     { Ld   HL,(AngD);Premier angle demandé }
  $E5/           { Push HL        ;Sauve }
  $CD/>Wxy/      { Call Wxy       ;Calcule x,y (DE,HL) selon la formule.
                                   (HL=Angle) }
  $ED/$53/>xDp/ { Ld    (xDp),DE ;Sauve coordonnées du point de départ }
  $22/>yDp/      { Ld   (yDp),HL }
  $ED/$53/>xPt/ { Ld    (xPt),DE ;Prépare coordonnées de travail (idem
                                   point de départ) }
  $22/>yPt/      { Ld   (yPt),HL }

  $E1/           { Pop  HL        ;Récupère premier angle (AngD) }
  $C1/           { Pop  BC        ;Récupère le nombre de points dans BC }
                 { Boucle principale }
```

```
$C5/        {L01 :Push BC        ;BC=compteur du nombre de points }
$ED/$5B/>Step/{ Ld   DE,(Step)}
$19/           { Add  HL,DE      ;Angle + Step }
$E5/           { Push HL         ;Sauve Angle }
$CD/>Wxy/      { Call Wxy        ;Calcule coordonnées }
$D5/           { Push DE         ;Sauve coordonnées }
$E5/           { Push HL         }
$ED/$5B/>xPt/ { Ld   DE,(xPt) ;Récupère coordonnées dernier point }
$2A/>yPt/      { Ld   HL,(yPt) }
$CD/>Firm/     { Call Firmware ;Place un point }
$BBEA/       {DW Gra-Plot-Absolute }
$E1/           { Pop  HL         ;Récupère coordonnées du point }
$D1/           { Pop  DE         }
$ED/$53/>xPt/ { Ld   (xPt),DE ;Sauve comme dernier point }
$22/>yPt/      { Ld   (yPt),HL }
$3A/>Ligne/    { Ld   A,(Ligne);Doit faire des points ou des lignes ? }
$B7/           { Or   A          }
$20/07/        { Jr   NZ,L02     ;Si fait des lignes }
$CD/>Firm/     { Call Firmware ;Fait des points (ne trace rien) }
$BBC0/       {DW Gra-Move-Absolute }
$18/05/        { Jr   L03        }
$CD/>Firm/ {L02 :Call Firmware ;Fait des lignes }
$BBF6/       {DW Gra-Line-Absolute }
$E1/        {L03 :Pop  HL        ;Récupère Angle }
$C1/           { Pop  BC         ;Récupère compteur du nombre de points }
$0B/           { Dec  BC         ;Compteur -1 }
$78/           { Ld   A,B        }
$B1/           { Or   C          }
$20/<-58/      { Jr   NZ,L01     ;et boucle }
               {                 }
$F1/           { Pop  AF         ;Récupère flags }
$C0/           { Ret  NZ         ;Retour immédiat si AngD<>AngF }
               { AngD=AngF (359 degrés de différence d'arc) }
$ED/$5B/>xPt/ { Ld   DE,(xPt) ;Place le dernier point }
$2A/>yPt/      { Ld   HL,(yPt) }
$CD/>Firm/     { Call Firmware }
$BBEA/       {DW Gra-Plot-Absolute }
$ED/$5B/>xDp/ { Ld   DE,(xDp) ;Relie au point de départ }
$2A/>yDp/      { Ld   HL,(yDp) }
$3A/>Ligne/    { Ld   A,(Ligne);Fait des lignes ou des points ? }
$B7/           {{ Or   A         }
```

```
  $20/06/       { Jr   NZ,L04   ;Va faire des lignes }
  $CD/>Firm/    { Call Firmware ;Fait un point }
  $BBC0/       {DW Gra-Move-Absolute }
  $C9/          { Ret           ;FIN }
  $CD/>Firm/ {L04 :Call Firmware }
  $BBF6        {DW Gra-Line-Absolute ;Fait une ligne et FIN }
 )
End;     { Ellipse }
```

Exemples :

Pour obtenir un cercle de centre 320/200 et de rayon 100 :

```
Ellipse (True,320,200,100,100,1,0,360)
```

Pour obtenir une ellipse de centre 300/100, de rayon horizontal 200 et vertical 50 :

```
Ellipse (True,300,100,200,50,1,0,360)
```

La même ellipse avec seulement la moitié supérieure :

```
Ellipse (True,300,100,200,50,1,0,180)
```

Avec des courbes brisées (Step=10) :

```
Ellipse (True,300,100,200,50,10,0,180)
```

Et enfin en n'affichant que les sommets des angles :

```
Ellipse (False,300,100,200,50,10,0,180)
```

Remplissage de zone

Tout bon programme graphique doit posséder une fonction de remplissage de zone, sur les 664 et 6128 il est possible d'appeler directement la routine Firmware GRA-FILL de remplissage, celui-ci

s'effectue alors dans les mêmes conditions que l'instruction FILL du BASIC (voir le manuel de l'utilisateur). La routine 'GRA-FILL' fait partie d'un pack de vecteurs d'extension propres aux 664 et 6128, avant toute utilisation il faut initialiser l'extension en appelant la routine Firmware GRA-EXTENDED-INITIALISE en $BD52.

Exemple :

```
Inline (
 $CD/>Firm/      { Call Firmware}
 $BD52       {DW GRA-EXT-INIT ;Initialise les routines d'extensions }
)
```

La routine GRA-FILL nécessite un buffer en mémoire vive, ce qui pose un petit problème sous CP/M Plus car il doit se situer en banque 0, il faudra donc le placer dans la zone non commutée de la RAM (voir Chapitre 2 : ' CP/M'). L'emplacement du buffer doit être indiqué dans le registre HL, sa taille maximum dans DE et la couleur de remplissage dans A. Le remplissage s'arrête sur des points de la même couleur que la couleur actuelle du stylo graphique ou que la couleur de remplissage.

Les procédures suivantes sont compatibles avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 : 'Problèmes de compatibilité').

Exemple :

```
PROCEDURE Remplir (couleur : Byte);
{
 Equivaut à 'FILL coul' en BASIC (sur CPC 664 et 6128).
}
VAR Buffer : Array [1..1024] Of Byte; { Buffer de travail pour GRA FILL }
{
 sous CP/M Plus, placer BUFFER en ABSOLUTE avec une adresse supérieure à
 $F000.
}
```

```
Begin
 Inline (
  $21/>Buffer/   { Ld   HL,Buffer;Adresse du buffer }
  $11/>1024/     { Ld   DE,&1024 ;Taille du buffer }
  $3A/>coul/     { Ld   A,(coul) ;Couleur de remplissage }
  $CD/>Firm/     { Call Firmware }
  $BD52        {DW Gra-Fill       }
 )
End;
```

La procédure suivante fonctionne sur tous les systèmes. Le remplissage commence à la position du curseur graphique et avec la couleur de l'encre graphique actuelle, les bordures sont délimitées par des points d'une couleur différente de celle du point de départ. Le remplissage est aussi limité dans la fenêtre graphique, l'origine des coordonnées étant supposée dans le coin en bas à gauche de la fenêtre. L'appui de la touche ESC arrête le programme. La taille du buffer de travail est réglable, elle doit être un multiple de 4 et ne peut dépasser 256*4=1 024 octets car le compteur du nombre de points empilés (COUNT) est de type Byte et varie donc de 0 à 255, soit 256 valeurs. Il faut aussi modifier la valeur de MAX dans le listing (voir (*)) qui doit être : (Taille Buffer) DIV 4.

```
PROCEDURE Remplir;
{
 Compatible avec tout les systèmes.
}
Var coul,              { Couleur actuelle du stylo }
    flag,              { Indique si un point est déjà empilé en haut ou
                         en bas .
                         Bit 0 à 1 si un point au-dessous est empilé.
                         Bit 1 à 1 si un point au-dessus est empilé.
                       }
    Count : Byte;      { Compteur du nombre de points empilés }
    ptr,               { Pointeur dans le buffer }
    x,y,               { Coordonnées du point le plus à gauche d'une
                         ligne }
    xd,yh : Integer;   { Limites X à droite et Y en haut de la fenêtre }
    Buffer : Array [1..128] Of Byte; {
                                     Buffer de travail.
```

```
                                        La taille est réglable,  doit  être
                                        un multiple de 4.
                                       }

PROCEDURE Test;
{
 Teste si la couleur du point de coordonnées DE,HL (X et Y) est égale à
 la couleur du point de départ.

 Z=1 si oui, il faudra colorier ce point.
 Z=0 si non, c'est une bordure.
}
Begin
 Inline (
  $C5/          { Push BC       ;Sauve registres }
  $D5/          { Push DE       }
  $E5/          { Push HL       }
  $CD/>Firm/    { Call Firmware }
  $BBF0/      {DW Gra-Test-Absolute ;Couleur du point dans A }
  $4F/          { Ld   C,A      ;Sauve }
  $3A/>coul/    { Ld   A,(coul) ;Couleur de remplissage }
  $B9/          { Cp   C        ;Egales ? }
  $E1/          { Pop  HL       ;Récupère registres }
  $D1/          { Pop  DE       }
  $C1           { Pop  BC       }
 )
End;

PROCEDURE Sauve;
{
 Empile les coordonnées actuelles (DE,HL) dans le buffer, réactualise le
 compteur du nombre de points empilés (la valeur actuelle est dans A) et
 le pointeur dans le buffer.
 DE et HL ne sont pas modifiés.
}
Begin
 Inline (
  $3C/          { Inc  A        ;Compteur+1 }
  $32/>count/   { Ld   (count),A}
  $D5/          { Push DE       ;Sauve X }
  $E5/          { Push HL       ;Sauve Y }
```

```
  $2A/>ptr/      { Ld   HL,(ptr) ;Ptr est sur la 1ère adresse libre }
  $73/           { Ld   (HL),E   ;Sauve X }
  $23/           { Inc  HL       }
  $72/           { Ld   (HL),D   }
  $23/           { Inc  HL       }
  $D1/           { Pop  DE       ;Récupère Y }
  $73/           { Ld   (HL),E   ;Sauve Y }
  $23/           { Inc  HL       }
  $72/           { Ld   (HL),D   }
  $23/           { Inc  HL       }
  $22/>ptr/      { Ld   (ptr),HL ;Réactualise ptr }
  $EB/           { Ex   DE,HL    ;Récupère Y dans HL }
  $D1            { Pop  DE       ;Récupère X }
 )
End;

{--- Corps de la procédure ---}

Begin
 Inline (
                 { Position actuelle du curseur graphique }
  $CD/>Firm/     { Call Firmware }
  $BBC6/       {DW Gra-Ask-Cursor;X et Y dans DE et HL }
  $ED/$53/>x/    { Ld   (x),DE   ;Sauve position du curseur }
  $22/>y/        { Ld   (y),HL   }
                 { Couleur du point de départ (référence) }
  $CD/>Firm/     { Call Firmware }
  $BBF0/       {DW Gra-Test-Absolute;couleur dans A }
  $32/>coul/     { Ld   (coul),A ;Sauve }
  $4F/           { Ld   C,A      }
                 { Couleur de remplissage (stylo graphique actuel) }
  $CD/>Firm/     { Call Firmware }
  $BBE1/       {DW Gra-Get-Pen   }
  $B9/           { Cp   C        ;Couleur du point actuel=couleur du
                                  stylo ? }
  $C8/           { Ret  Z        ;Retour immédiat si oui }
                 { Coordonnées de la fenêtre ? }
  $CD/>Firm/     { Call Firmware }
  $BBD5/       {DW Gra-Get-W-Width ;Bord X droite dans HL }
  $E5/           { Push HL       ;Sauve }
  $CD/>Firm/     { Call Firmware }
```

```
$BBD8/       {DW Gra-Get-W-Height ;Bord Y haut dans DE }
$D5/           { Push DE        ;Sauve }
$CD/>Firm/     { Call Firmware }
$BBCC/       {DW Gra-Get-Origin; Origine X,Y dans DE,HL }
$E3/           { Ex   (SP),HL  ;HL=Bord Y haut }
$C1/           { Pop  BC       ;BC=coordonnée Y origine }
$B7/           { Or   A        }
$ED/$42/       { Sbc  HL,BC    ;HL=Bord Y haut par rapport à l'origine}
$22/>yh/       { Ld   (yh),HL  ;Stocke }
$E1/           { Pop  HL       ;HL=bord X droite }
$B7/           { Or   A        }
$ED/$52/       { Sbc  HL,DE    ;HL=Bord X droite par rapport à l'orig.}
$22/>xd/       { Ld   (xd),HL  ;Stocke }
               { Initialisations }
$21/>Buffer/   { Ld   HL,Buffer;Initialise pointeur dans buffer }
$22/>Ptr/      { Ld   (Ptr),HL }
$ED/$5B/>x/    { Ld   DE,(x)   ;DE,HL=coordonnées du point de départ }
$2A/>y/        { Ld   HL,(y)   }
$AF/           { Xor  A        ;Compteur de points empilés à 0 }
$32/>count/    { Ld   (count),A}
               { Calcule le pas en X selon le mode }
               { Indique ce qu'il faut ajouter ou retrancher à X pour
                 passer au point voisin }
$CD/>Firm/     { Call Firmware }
$BC11/       {DW Scr-Get-Mode  }
$3E/4/         { Ld   A,$04    }
$38/4/         { Jr   C,L00    ;Si mode 0 : +4 }
$1F/           { Rra           }
$28/1/         { Jr   Z,L00    ;Si mode 1 : +2 }
$1F/           { Rra           ;Si mode 2 : +1 }
$4F/         {L00 :Ld   C,A     ;BC=pas en X }
$06/0/         { Ld   B,$00    }
$DD/$21/>flag/{ Ld   IX,flag  ;IX pointe sur FLAG }
               {               }
               { Recherche la bordure à gauche }
$EB/         {L01 :Ex   DE,HL   ;X-pas }
$B7/           { Or   A        }
$ED/$42/       { Sbc  HL,BC    }
$EB/           { Ex   DE,HL    }
$CB/$7A/       { Bit  7,D      ;Coordonnée X inférieur à 0 ? }
$20/05/        { Jr   NZ,L02   ;Oui, sort de la fenêtre : bordure }
```

```
 $CD/>test/     { Call Test      ;Teste si bordure }
 $28/<-14/      { Jr   Z,L01     ;Non, boucle }
                {                }
                { Le bord gauche est atteint }
$DD/$36/0/0/{L02 :Ld   (IX+00),0;Initialise FLAG }
 $EB/           { Ex   DE,HL     ;X+pas, réajuste X à l'intérieur de ...}
 $09/           { Add  HL,BC     ;...la zone à remplir }
 $EB/           { Ex   DE,HL     }
 $ED/$53/>x/    { Ld   (x),DE    ;Sauve coordonnées du point le plus ...}
 $22/>y/        { Ld   (y),HL    ;... à gauche }
                {                }
                {
                  Recherche le bord droit de la zone.
                  Teste tous les points au-dessus et  au-dessous de la
                  ligne horizontale ainsi délimitée pour savoir si ils
                  sont à remplir. Le FLAG permet d'éviter de stocker
                  plusieurs coordonnées de points pour chaque zone à
                  remplir qui sera trouvée. Seul le point le plus à
                  gauche de chaque zone sera conservé en mémoire (dans
                  le Buffer).
                }
                { Test si appui de ESC }
 $3E/66/     {L03 :Ld   A,66      ;Numéro Amstrad de la touche ESC }
 $C5/           { Push BC        ;Sauve registres }
 $E5/           { Push HL        }
 $CD/>Firm/     { Call Firmware  }
 $BB1E/       {DW Km-Test-Key    ;Teste la touche. Z=0 si appuyée }
 $E1/           { Pop  HL        ;Récupère registres }
 $C1/           { Pop  BC        }
 $C0/           { Ret  NZ        ;FIN si ESC appuyée }
                {                }
                { Teste le point SOUS le point actuel }
 $2B/           { Dec  HL        ;Y-2, va une ligne au-dessous }
 $2B/           { Dec  HL        }
 $CB/$7C/       { Bit  7,H       ;Coordonnée Y inférieure à 0 ? }
 $CC/>test/     { Call Z,Test    ;Non, teste le point }
 $20/21/        { Jr   NZ,n_sous;1/ Sort de la fenêtre (voir BIT 7,H)
                                 ;2/ Le point n'est pas à remplir (TEST)
                                 Alors ne fait rien }
                { Le point au-dessous est à remplir }
 $DD/$CB/0/$46/{ Bit  0,(IX+0) ;déjà un point au-dessous empilé ? }
```

```
$20/19/       { Jr   NZ,dessus;Oui, ne fait rien }
$DD/$CB/0/$C6/{ Set  0,(IX+0) ;Non, active FLAG }
$3A/>count/   { Ld   A,(count);Le buffer est-il plein ? }
$FE/32/       { Cp   Max      ;(*) }
$C8/          { Ret  Z        ;Oui, FIN }
$CD/>Sauve/   { Call Sauve    ;Non, empile coordonnées }
$18/04/       { Jr   dessus   ;Va tester le point au-dessus }
$DD/$CB/0/$86/{Le point au-dessous n'est pas à remplir }
        {n_sous :Res 0,(IX+0)  ;Désactive FLAG }
              {                }
              { Teste le point au-dessus }
$23/    {dessus :Inc  HL       ;(Y-2)+4 donne Y+2 : ligne au-dessus }
$23/          { Inc  HL       }
$23/          { Inc  HL       }
$23/          { Inc  HL       }
$D5/          { Push DE       ;Sauve coordonnées }
$E5/          { Push HL       }
$ED/$5B/>yh/  { Ld   DE,(yh)  ;Y > Bord Y haut de la fenêtre ? }
$13/          { Inc  DE       }
$B7/          { Or   A        }
$ED/$52/      { Sbc  HL,DE    }
$E1/          { Pop  HL       ;Récupère coordonnées }
$D1/          { Pop  DE       }
$30/26/       { Jr   NC,n_sur ;Si Y sort fenêtre, ne fait rien }
$CD/>test/    { Call test     ;Y dans fenêtre, teste le point }
$20/21/       { Jr   NZ,n_sur ;Si n'est pas à remplir }
$DD/$CB/0/$4E/{ Bit  1,(IX+0) ;Déjà un point au-dessus à remplir ? }
$20/19/       { Jr   NZ,adrte ;Oui, ne fait rien }
$DD/$CB/0/$CE/{ Set  1,(IX+0) ;Non, active FLAG }
$3A/>count/   { Ld   A,(count);Buffer plein ? }
$FE/32/       { Cp   Max      ;(*) }
$C8/          { Ret  Z        ;Oui, FIN }
$CD/>Sauve/   { Call Sauve    ;Empile les coordonnées }
$18/04/       { Jr   adrte    ;Va un point à droite }
$DD/$CB/0/$8E/{ Le point au-dessus n'est pas à remplir }
        {n_sur :Res 1,(IX;0)  ;Désactive FLAG }
              {               }
              { Va un point à droite }
$2B/    {adrte :Dec  HL       ;Récupère Y initial (Y-2) }
$2B/          { Dec  HL       }
$EB/          { Ex   DE,HL    }
```

```
  $09/          { Add  HL,BC     ;X+pas, va un point à droite }
  $D5/          { Push DE        ;Sauve coordonn{es }
  $E5/          { Push HL        }
  $ED/$5B/>xd/  { Ld   DE,(xd)   ;Dépasse le bord droit de la fenêtre ? }
  $13/          { Inc  DE        }
  $B7/          { Or   A         }
  $ED/$52/      { Sbc  HL,DE     }
  $E1/          { Pop  HL        ;Récupère coordonnées }
  $D1/          { Pop  DE        }
  $EB/          { Ex   DE,HL     }
  $30/06/       { Jr   NC.depile;Si dépasse bord droit, ne fait rein }
  $CD/>test/    { Call test      ;Point dans fenêtre, teste celui-ci }
  $CA/>*-$75/   { Jp   Z,L03     ;Boucle si point à remplir }
                {                }
                { Le bord droit est atteint }
  $EB/    {depile :Ex   DE,HL    ;Ajuste X de façon à être DANS la zone
}
  $B7/          { Or   A         }
  $ED/$42/      { Sbc  HL,BC     ;X-pas }
  $EB/          { Ex   DE,HL     }
                { Trace une ligne horizontale jusqu'au bord gauche }
  $C5/          { Push BC        ;Sauve pas en X }
  $CD/>Firm/    { Call Firmware ;Place le curseur sur le bord droit }
  $BBC0/      {DW Gra-Move-Absolute}
  $ED/$5B/>x/   { Ld   DE,(x)    ;Récupère coordonnées du bord gauche }
  $2A/>y/       { Ld   HL,(y)    }
  $CD/>Firm/    { Call Firmware ;Fait une ligne jusqu'au bord gauche }
  $BBF6/      {DW Gra-Line-Absolute}
  $C1/          { Pop  BC        ;Récupère pas en X }
                {                }
                { Récupère une coordonnée du buffer }
  $3A/>count/{d_1 :Ld   A,(count);Buffer vide ? }
  $B7/          { Or   A         }
  $C8/          { Ret  Z         ;Oui, FIN }
  $3D/          { Dec  A         ;Non, compteur -1 }
  $32/>count/   { Ld   (count),A}
  $2A/>ptr/     { Ld   HL,(ptr) ;Récupère coordonnées }
  $2B/          { Dec  HL        }
  $56/          { Ld   D,(HL)    ;Coordonnée Y dans DE }
  $2B/          { Dec  HL        }
  $5E/          { Ld   E,(HL)    }
```

```
  $2B/          { Dec  HL        }
  $D5/          { Push DE        ;Sauve Y }
  $56/          { Ld   D,(HL)    ;Coordonnée X dans DE }
  $2B/          { Dec  HL        }
  $5E/          { Ld   E,(HL)    }
  $22/>ptr/     { Ld   (Ptr),HL ;Réactualise pointeur }
  $E1/          { Pop  HL        ;Récupère Y dans HL }
                {                }
                { Le nouveau point de départ est il déjà rempli ? }
                { (En effet, il peut avoir été rempli lors d'opérations
                  précédentes ) }
  $CD/>test/    { Call test      ;Teste }
  $20/<-30/     { Jr   NZ,d_1    ;Déjà rempli, dépile un autre point }
  $C3/>*-$CA    { Jp   L00       ;Non, Boucle principale }
 )
End;
```

SAUVEGARDE ET RAPPEL D'IMAGES D'ECRAN

Nous allons voir maintenant comment charger et sauver sur disque des images complètes ou partielles. Les images complètes peuvent être issues de logiciels standard de l'Amstrad, les fichiers seront alors au format AMSDOS (voir Chapitre 4 : 'Manipulation de fichiers binaires'). En raison de l'organisation mémoire très différente des deux CP/M, les deux versions des programmes pourront être très différentes.

CHARGEMENT D'IMAGES COMPLETES

Version CP/M 2.2

Le programme est déjà décrit dans le Chapitre 4 : 'Manipulation des fichiers binaires'.

```
Type NomFichier = String [14];

PROCEDURE ChargeImage (Nom : NomFichier);
Var Fic   : File;
    Debut : Byte ABSOLUTE $C000;  {début de la RAM Vidéo}

Begin
 Assign (Fic,Nom);
 Reset (Fic);

 Seek (Fic,1);          { Fichiers AMSDOS seulement : Saute l'entête}

 BlockRead (Fic,Debut,128); { Charge 16 KO dans la RAM Vidéo}
 Close (Fic)
End;
```

Version CP/M Plus

Il faut transférer les enregistrements lus dans la RAM vidéo qui se trouve en banque 0. Pour cela, on peut comme dans cet exemple placer l'instruction LDIR du microprocesseur en zone non commutée de la mémoire et l'appeler avec la fonction UserF en s'assurant que la variable Buffer soit aussi en zone non commutée (voir 'Accès à la mémoire vidéo sous CP/M Plus' Chapitre 2 : 'CP/M'). Dans le cadre de vos applications, il sera peut-être nécessaire d'utiliser une routine de transfert comme celle décrite dans le même chapitre ainsi que dans le paragraphe suivant.

```
Type NomFichier = String [14];

PROCEDURE ChargeImage (Nom : NomFichier);
Var Adr   : Integer;       { Adresse en mémoire vidéo }
    Fic   : File;          { Fichier pour lecture }
    Buffer : Array [1..128] Of Byte;  { Buffer pour un enregistrement }
Begin
 { Installe LDIR en zone non commutée }
 Mem [$FC5D] :=$ED;      { LDIR }
 Mem [$FC5E] :=$B0;
 Mem [$FC5F] :=$C9;      { RET  }
```

```
Assign (Fic,Nom);
Reset (Fic);

Seek (Fic,1);          { Fichiers AMSDOS seulement : Saute l'entête }

Adr :=$4000;           { Début RAM vidéo }

While Adr > 0 Do       { Fin de RAM vidéo en $8000 qui est un nombre
                         négatif en représentation ' complément à 2' }
 Begin
  BlockRead (Fic,Buffer,1);      { Lit un enregistrement }
  Inline (       { Transfère Buffer en RAM vidéo }
   $21/>Buffer/ { Ld HL,Buffer   ;Adresse départ }
   $ED/$5B/>adr/{ Ld   DE,(adr)  ;Adresse destination }
   $01/>128/     { Ld   BC,128    ;Nombre d'octets }
   $CD/$FC5A/    { Call UserF     ;Connectera banque 0 }
   $FC5D/      {DW Adresse routine}
   $ED/$53/>Adr { Ld   (Adr),DE  ;Nouvelle adresse destination (+128) }
  )
 End;   { While }

Close (Fic)
End;
```

SAUVEGARDE D'IMAGES COMPLETES

Comme le travail s'effectue sous CP/M, si l'on sauvegardait des images complètes, elles ne serait pas réutilisables en BASIC en raison de l'absence de l'en-tête AMSDOS dans le fichier. Pour travailler avec des images complètes, il est préférable d'utiliser les utilitaires décrits dans le paragraphe suivant en spécifiant des données pour un écran complet.

CHARGEMENT ET SAUVEGARDE D'IMAGES PARTIELLES

Il peut aussi être très intéressant de ne sauver qu'une portion rectangulaire de l'écran. Nous allons étudier des routines permettant cela, elles ne peuvent bien fonctionner que si l'écran n'a jamais été 'scrollé' auparavant. Elles permettent le chargement et la sauvegarde d'images sur le disque ; comme le travail s'effectue sous CP/M, les fichiers ne comportent pas d'en-tête AMSDOS. Un programme en BASIC et en Assembleur vous permettra de recharger et même fusionner ces portions d'images en BASIC et ainsi de créer des fichiers AMSDOS standard.

Pour des raisons de commodité, la portion d'image est restreinte à des octets entiers, de la même façon que les fenêtres graphiques du Firmware. Ainsi, la coordonnée X gauche sera arrondie à un multiple de 8 par défaut et celle de droite à un multiple de 8 par excès moins 1. Les fichiers créés ont le format suivant :

1er enregistrement :

Octets 1 et 2 : Coordonnée X du point en bas à gauche.
Octets 3 et 4 : Coordonnée Y du point en bas à gauche.
Octet 5 : Largeur en octets (de 1 à 80).
Octet 6 : Hauteur en lignes réelles (de 1 à 200).
Les autres octets ne sont pas utilisés.

Enregistrements suivants :

Tous les octets de la mémoire vidéo concernée. On commence par la ligne la plus basse de gauche à droite puis on passe à la ligne au-dessus. Toutes les lignes sont contiguës dans le fichier.

La procédure SAUVEGRAPH sauve dans le fichier NOM la fenêtre d'écran dont le point en bas à gauche est de coordonnées X,Y et le point en haut à droite est de coordonnées X1,Y1. La routine arrondit ces valeurs de façon à ne sauver que des octets entiers.

La procédure LIREGRAPH affiche le dessin sauvé avec SauveGraph, elle renvoie dans les variables XX,YY,XX1,YY1 les coordonnées de la fenêtre.

Les procédures sont compatibles avec les deux CP/M à condition d'ajuster le contenu de la routine FIRM (voir Chapitre 5 :

'Problèmes de compatibilité'), néanmoins la routine TRANSFERT qui permet d'accéder à la mémoire vidéo n'est pas identique pour les deux CP/M.

```
{
 Variables globales nécessaires au programme :
}
TYPE NomFic = String [14];       { Pour noms des fichiers }
VAR  Fic    : File;              { Fichier disque }
     Buffer : Array [1..128] Of Byte; { Pour contenir un enregistrement}

{
 PROCEDURE TRANSFERT.

 Version CP/M Plus.

 La procédure est déjà décrite dans le chapitre 'Le CP/M', 'Accès à la
 mémoire vidéo sous CP/M Plus').
}

PROCEDURE Transfert (BanqueD,BanqueF : Byte; AdrD,AdrF,Nombre :Integer);
Type Parametres = Record                { Bloc de paramètres }
                   Fon,                 { Numéro de fonction BIOS }
                   Accu     : Byte;     { Contenu des registres a passer}
                   BC,DE,HL : Integer
                  End;
Var BiosPb : Parametres;

Begin
 With BiosPb Do
  Begin
   Fon :=29;                      { BIOS XMOVE }
   BC :=BanqueF*256+BanqueD;      { B=destination, C=départ }
   Bdos (50,Addr (BiosPb));       { Exécute XMOVE }

   Fon :=25;                      { BIOS MOVE }
   BC :=Nombre;                   { Charge registres }
   DE :=AdrD;
   HL :=AdrF;
   Bdos (50,Addr(BiosPb))         { Exécute MOVE }
```

```
  End {with}
End;

{
 Version CP/M 2.2.

 Les paramètres de numéros de banque sont inutiles.
 La routine est déjà décrite dans le chapitre 'Le CP/M','Accès à la
 mémoire vidéo sous CP/M 2.2.
}

PROCEDURE Transfert (Depart,Dest,Nombre : Integer);
Begin
 Inline (
  $2A/>Depart/      { Ld   HL,(depart) ;Adresse départ}
  $ED/$5B/>Dest/    { Ld   DE,(dest)   ;Adresse destination}
  $ED/$4B/>Nombre/ { Ld   BC,(Nombre) ;Nombre d'octets}
  $CD/$BE9B/        { Call Entrer-Firmware }
  $B91B           {DW KL-LDIR          }
 )
End;

FUNCTION AdresseVideo (x,y : Integer) : Integer;
{
 Calcule l'adresse vidéo du point X,Y.
}
VAR Adr : Integer;      { Adresse vidéo }
    Mode : Byte;         { Mode de l'écran }
Begin
{
  X et Y sont transformés en coordonnées de base (1 unité = 1 pixel ).
}
Begin
{ Mode de l'écran ? }
 Inline (
  $CD/>Firm/    { Call Firmware }
  $BC11/      {DW Scr-Get-Mode  ;Mode dans A }
  $32/>Mode     { Ld   (Mode),A }
 );
{ Convertit X selon le mode (Note : En mode 2, il n'y a rien à faire }
 Case Mode Of
```

```
  1 : x :=x DIV 2;
  0 : x :=x DIV 4
 End;
{ Convertit Y (400 points pour 200 pixels) }
 y :=y DIV 2;
 Inline (
  $ED/$5B/>x/    { Ld   DE,(x)    }
  $2A/>y/        { Ld   HL,(y)    }
  $CD/>Firm/     { Call Firmware }
  $BC1D/       {DW Scr-Dot-Position ;Adresse dans HL }
  $22/>Adr       { Ld   (Adr),HL }
 );
 AdresseVideo :=Adr
End;

PROCEDURE SauveGraph (Nom : NomFic; x,y,x1,y1 : Integer);
{
 Sauve la fentre d'écran X,Y,X1,Y1 dans le fichier NOM.
}
VAR BaseBuffer,          { Pour simplifier calculs }
    Adr       : Integer;{ Adresse vidéo }
    lg,ht,               { Largeur en octets, hauteur en lignes }
    Nombre,              { Nombre d'octets pour le transfert }
    Lignes,              { Nombre de lignes traitées }
    BuffLibre,           { Place libre dans le buffer }
    VideoOff : Byte;     { Position dans une ligne d'écran, à partir de 0}
Begin
{ Création du fichier }
 Assign (Fic,Nom);
 ReWrite (Fic);
{ Calcule de LG et HT }
 lg :=Succ((x1 DIV 8)-(x DIV 8));
 ht :=Succ((y1 DIV 2)-(y DIV 2));
{{ Prépare 1er enregistrement }
 Buffer[1] :=Lo (x);
 Buffer[2] :=Hi (x);
 Buffer[3] :=Lo (y);
 Buffer[4] :=Hi (y);
 Buffer[5] :=lg;
 Buffer[6] :=ht;
{ Ecrit 1er enregistrement }
```

```
 BlockWrite (Fic,Buffer,1);

{ Initialise données }
 Adr :=AdresseVideo (x,y); { Adresse vidéo du point en bas à droite ? }
 VideoOff :=0;                    { Offset sur la ligne vidéo }
 Lignes :=0;                      { Nombre de lignes traitées }
 BaseBuffer :=Addr(Buffer)+128;   { Adresse de la fin du Buffer }

{ Boucle principale }
 Repeat                  { Boucle pour toutes les lignes à traiter }
 | BuffLibre :=128;       { Place libre dans Buffer }

 |{ Tant que buffer pas plein... }
 | While BuffLibre<>0 Do
 |  Begin
 |   {... si il reste plus d'octets sur une ligne vidéo que de place dans
 |    le buffer ...}
 |   If lg-VideoOff >= BuffLibre
 |   | { remplira la fin du buffer seulement }
 |   |  Then Nombre :=BuffLibre
 |   | { sinom transfèrera ce qu'il reste sur la ligne }
 |   |  Else Nombre :=lg-VideoOff;

 |  { Transf}re le nombre adéquat d'octets dans le buffer }
{| CP/M Plus : }
 |   Transfert (0,1,Adr+VideoOff,BaseBuffer-BuffLibre,Nombre);
{| CP/M 2.2 : }
 |   Transfert (Adr+VideoOff,BaseBuffer-BuffLibre,Nombre);

 |  { Nouvelle offset sur la ligne }
 |   VideoOff :=VideoOff+Nombre;
 |  { Arrivé à l'extrémité droite de la ligne ? }
 |   If VideoOff=lg Then
 |    Begin
 |     Lignes :=Succ(Lignes);      { Oui, une ligne de plus traitée }
 |     VideoOff :=0;               { Réinitialise Offset }
 |     Inline (
 |      $2A/>Adr/    { Ld   HL,(Adr) ;Calcule nouvelle adresse de base }
 |      $CD/>Firm/   { Call Firmware }
 |      $BC29/     {DW Scr-Prev-Line ;pour la ligne au dessus }
 |      $22/>Adr     { Ld   (Adr),HL }
```

```
      )
     End;
    BuffLibre :=BuffLibre-Nombre { Réactualise place dans Buffer }
   End;

 { Le Buffer est plein, l'écrit sur le disque }
  BlockWrite (Fic,Buffer,1);
 Until Lignes >= ht;        { Fin quand toutes les lignes sont sorties }

 Close (Fic)
End;

PROCEDURE LireGraph (Nom : NomFic; VAR x,y,x1,y1 : Integer);
VAR BaseBuffer,          { Pour simplifier calculs }
    Adr       : Integer;{ Adresse vidéo }
    lg,ht,               { Largeur en octets, hauteur en lignes }
    Nombre,              { Nombre d'octets pour le transfert }
    Lignes,              { Nombre de lignes traitées }
    BuffLibre,           { Place libre dans le buffer }
    VideoOff : Byte;     { Position dans une ligne d'écran, à partir de 0}
Begin
{ Ouvre le fichier }
 Assign (Fic,Nom);
 Reset (Fic);
{ Lecture des informations du 1er enregistrement }
 BlockRead (Fic,Buffer,1);
 x :=Buffer[1]+256*Buffer[2];  { Position X,Y du point en bas à gauche}
 y :=Buffer[3]+256*Buffer[4];
 lg :=Buffer[5];               { Largeur en octets }
 ht :=Buffer[6];               { Hauteur en lignes }
 x1 :=Pred(x+8*lg);            { Calcule les coordonnées du point... }
 y1 :=y+2*ht;                  { ...en haut à droite }

{ Initialise données }
 Adr :=AdresseVideo (x,y); { Recherche de l'adresse vidéo correspon-
dante}
 VideoOff :=0;
 Lignes :=0;
 BuffLibre :=0;                { Indique Buffer vide }
 BaseBuffer :=Addr(Buffer)+128;
```

```
{ Boucle principale }
 Repeat
  If BuffLibre=0 Then           { Si le buffer est vide ... }
   Begin
    BlockRead (Fic,Buffer,1);   {...alors lit un nouvel enregistrement }
    BuffLibre :=128
   End;
{ Voir routine SauveGraph pour les explications suivantes }
  If lg-VideoOff >= BuffLibre Then Nombre :=BuffLibre
                              Else Nombre :=lg-VideoOff;
  Transfert (1,0,BaseBuffer-BuffLibre,Adr+VideoOff,Nombre);
  VideoOff :=VideoOff+Nombre;
  If VideoOff=lg Then
   Begin
    Lignes :=Succ(Lignes);
    VideoOff :=0;
    Inline (
     $2A/>Adr/    { Ld   HL,(Adr) }
     $CD/>Firm/   { Call Firmware }
     $BC29/     {DW Scr-Prev-Line }
     $22/>Adr     { Ld   (Adr),HL }
    )
   End;
  BuffLibre :=BuffLibre-Nombre;
 Until Lignes >= ht;
 Close (Fic)
End;
```

Pour les passionnés de l'Assembleur, voici des routines équivalentes à celles présentées ci-dessus. Elles présentent l'avantage d'être plus compactes et plus rapides, l'algorithme utilisé étant toujours le même.

Note concernant l'utilisation avec CP/M Plus :

Pour faciliter le problème du transfert des octets entre les deux banques de RAM sous CP/M Plus, on supposera que la variable BUFFER est située dans la zone non commutée de la mémoire (adresse supérieure à $C000). Pour effectuer le transfert, il suffit alors de placer la routine : LDIR/RET en zone non commutée aussi et de l'appeler à travers la fonction USERF. Voir le chapitre 2 : 'Le

CP/M' pour tous les détails. On supposera donc que les instructions suivantes sont déjà exécutées :

```
MEM [$FC5D] :=$ED; { LDIR }
MEM [$FC5E] :=$B0;
MEM [$FC5F] :=$C9; { RET }


{
 Variables globales nécessaires au programme :
}
TYPE NomFic = String [14];       { Pour noms des fichiers }
VAR  Fic    : File;              { Fichier disque }
     Buffer : Array [1..128] Of Byte; { Pour contenir un enregistrement}

PROCEDURE RWGra (Vadresse : Integer; Sauver : Boolean);
{
 Utilisée par Lire/Sauve GRAPH.
 Vadresse= Adresse en mémoire vidéo. Au départ c'est l'adresse de
 l'octet en bas à gauche.
 Sauver est TRUE si c'est une sauvegarde, sinon effectue une lecture.
 Buffer [5] contient déjà la largeur en octets.
 Buffer [6] contient la hauteur en lignes.
}
VAR lg,ht,
    Nombre,
    Lignes,
    VideoOff,
    BuffLibre : Byte;
    Adr       : Integer;

PROCEDURE InitDatas;
{
 Initialise variables.
}
Begin
 Inline (
  $3A/>Buffer+4/{ Ld   A,(Buffer[5]) ;LG :=Buffer[5] }
  $32/>lg/      { Ld   (lg),A        }
  $3A/>Buffer+5/{ Ld   A,(Buffer[6]) ;HT :=Buffer[6] }
  $32/>ht/      { Ld   (ht),A        }
  $22/>adr/     { Ld   (adr),HL      ;Adresse vidéo }
```

```
  $AF/          { Xor  A              }
  $32/>VideoOff/{ Ld   (VideoOff),A   ;VideoOff :=0 }
  $32/>Lignes   { Ld   (Lignes),A     ;Lignes :=0 }
 )
End;

PROCEDURE QNombre;
{
 Calcule le nombre d'octets à transférer entre la mémoire vidéo et le
 buffer ou vice-versa.

 A=BuffLibre
}
Begin
 Inline (
  $4F/          { Ld   C,A            }
  $3A/>VideoOff/{ Ld   A,(VideoOff)   }
  $47/          { Ld   B,A            }
  $3A/>lg/      { Ld   A,(lg)         }
  $90/          { Sub  B              }
  $B9/          { Cp   C              ;Lg-VideoOff >= BuffLibre ? }
  $38/1/        { Jr   C,L00          }
  $79/          { Ld   A,C            ;Non : Nombre :=BuffLibre }
  $32/>Nombre{L00 :Ld   (Nombre),A    ;Oui : Nombre :=Lg-VideoOff }
 )
End;

PROCEDURE LdirDatas;
{
 Calcule les données pour le transfert.
 Note :
  A=Nombre
  B=VideoOff
  C=BuffLibre
}
Begin
 Inline (
                { Calcule position dans RAM vidéo }
  $2A/>Adr/     { Ld   HL,(Adr)       }
  $58/          { Ld   E,B            }
  $16/0/        { Ld   D,#00          }
```

```
  $19/          { Add  HL,DE          ;HL=Adr+VideoOff }
  $E5/          { Push HL             ;Sauve }
                { Calcule position dans Buffer }
$21/>Buffer+128/{ Ld   HL,Buffer+128 }
  $59/          { Ld   E,C            }
  $B7/          { Or   A              }
  $ED/$52/      { Sbc  HL,DE          ;HL=Addr(Buffer [129-BuffLibre]) }
  $D1/          { Pop  DE             ;Récupère adresse vidéo dans DE }
  $4F/          { Ld   C,A            ;C=Nombre d'octets à transférer }
  $80/          { Add  A,B            ;A=VideoOff+Nombre  (nouvel  Offset
                                        vidéo) }
  $06/0         { Ld   B,#00          ;BC=Nombre d'octets à transférer
                                        Il y en a moins  que  80  car  une
                                        ligne d'écran fait 80 octets de
                                        large. }
 )
End;

PROCEDURE Transfert;
{
 Effectue le transfert et calcule les nouvelles valeurs des variables.
 En entrée :
  A=nouveau VideoOff - BC,DE,HL=valeurs pour LDIR
}
Begin
 Inline (
                { Transfert           }
  $CD/>Firm/    { Call Firmware       }

  $B91B/      {DW KL-LDIR             ;Version CP/M 2.2 }
  $FC5D/      {DW Adresse routine     ;Version CP/M Plus }

  $47/          { Ld   B,A            }
  $3A/>lg/      { Ld   A,(lg)         ;lg=VideoOff (fin de ligne) ? }
  $B8/          { Cp   B              }
  $78/          { Ld   A,B            }
  $20/16/       { Jr   NZ,L00         ;Non, A=Nouveau VideoOff }
                { Oui, une ligne vidéo entière a été traitée. Passe à la
                  ligne suivante      }
  $2A/>Adr/     { Ld   HL,(Adr)       ;Adresse  de  l'octet  le  plus  à
                                       gauche de la ligne actuelle }
```

```
  $CD/>Firm/     { Call Firmware       ;Calcule  adresse  de   l'octet...}
  $BC29/       {DW Scr-Prev-Line       ;... au-dessus. }
  $22/>Adr/      { Ld   (Adr),HL       }
  $21/>Lignes/   { Ld   HL,Lignes      ;Nombre de lignes traitées + 1 }
  $34/           { Inc  (HL)           }
  $AF/           { Xor  A              ;VideoOff remis à zéro }
                 {                     }
$32/>VideoOff/{L00 :Ld   (VideoOff),A  ;Réactualise VideoOff }
                 { Effectue BuffLibre :=BuffLibre-Nombre }
  $3A/>Nombre/   { Ld   A,(Nombre)     }
  $4F/           { Ld   C,A            }
 $3A/>BuffLibre/{ Ld   A,(BuffLibre)  }
  $91/           { Sub  C              }
  $32/>BuffLibre{ Ld   (BuffLibre),A  }
 )
End;

PROCEDURE TstFin;
{
 Teste la fin du traitement.
 Fin si le nombre de lignes traitées est égal au nombre de lignes total.
}
Begin
 Inline (
  $3A/>ht/       { Ld   A,(ht)         }
  $4F/           { Ld   C,A            }
  $3A/>Lignes/   { Ld   A,(Lignes)     }
  $B9            { Cp   C              ;Lignes=Ht ? (modifie CARRY) }
 )
End;

PROCEDURE LireSec;
{ Lecture d'un enregistrement }
Begin
 BlockRead (Fic,Buffer,1)
End;

PROCEDURE WriteSec;
{ Ecriture d'un enregistrement }
Begin
 BlockWrite (Fic,Buffer,1)
```

```
End;

{--- Corps de la procédure RWGRA ---}

Begin
 Inline (
  $2A/>Vadresse/      { Ld   HL,(Vadresse) ;HL= Vadresse }
  $3A/>Sauver/        { Ld   A,(Sauver)    ;Sauvegarde ? }
  $B7/                { Or   A             }
  $20/35/             { Jr   NZ,save       ;Si OUI }
                      {                    }
                      { CHARGEMENT         }
  $CD/>InitDatas/     { Call InitDatas     ;Initialise données }
  $32/>BuffLibre/     { Ld   (BuffLibre),A ;BuffLibre=0 }
  $3A/>BuffLibre/{lod0 :Ld   A,(BuffLibre) ;BuffLibre=0 ? }
  $B7/                { Or   A             }
  $20/08/             { Jr   NZ,lod1       ;Si <>0 }
  $CD/>LireSec/       { Call LireSec       ;0, recharge le buffer }
  $3E/<128/           { Ld   A,&128        }
  $32/>BuffLibre/     { Ld   (BuffLibre),A ;Nouvelle valeur }
  $CD/>QNombre/  {lod1 :Call QNombre       ;Calcule le nombre
d'octets..}
  $CD/>LdirDatas/     { Call LdirDatas     ;... pour le transfert }
  $CD/>Transfert/     { Call Transfert     ;Transfert et réact. var. }
  $CD/>TstFin/        { Call TstFin        ;Fin ? }
  $38/<-28/           { Jr   C,lod0        ;Si non }
  $C9/                { Ret                ;FIN }
                      {                    }
                      { SAUVEGARDE         }
  $CD/>InitDatas/{save :Call InitDatas     ;Initialise données }
  $3E/<128/      {sav0 :Ld   A,&128        ;BuffLibre :=128 }
  $32/>BuffLibre/     { Ld   (BuffLibre),A }
  $3A/>BuffLibre/{sav1 :Ld   A,(BuffLibre) ;BuffLibre=0 ? }
  $B7/                { Or   A             }
  $28/12/             { Jr   Z,sav2        ;Si oui (Buffer plein) }
  $CD/>QNombre/       { Call QNombre       ;Non, calcule nombre d'octets}
  $CD/>LdirDatas/     { Call LdirDatas     ;et valeurs pour le transfert}
  $EB/                { Ex   DE,HL         ;Réajuste adresses de départ
                                            et de destination }
  $CD/>Transfert/     { Call Transfert     ;Transfert et réact. var. }
  $18/<-18/           { Jr   sav1          ;Boucle }
```

```
  $CD/>WriteSec/ {sav2 :Call WriteSec      ;Buffer plein, le décharge }
  $CD/>TstFin/       { Call TstFin         ;Fin ? }
  $38/<-31           { Jr   C,sav0         ;Si non }
 )
End;

PROCEDURE SauveGraph (Nom : NomFic; x,y,x1,y1 : Integer);
{
 Sauve la fentre d'écran X,Y,X1,Y1 dans le fichier NOM.
}
Begin
{ Création du fichier }
 Assign (Fic,Nom);
 ReWrite (Fic);
{ Calcule de LG et HT }
{ Prépare ler enregistrement }
 Buffer[1] :=Lo (x);
 Buffer[2] :=Hi (x);
 Buffer[3] :=Lo (y);
 Buffer[4] :=Hi (y);
 Buffer[5] :=Succ((x1 DIV 8)-(x DIV 8));    { Largeur en octets }
 Buffer[6] :=Succ((y1 DIV 2)-(y DIV 2));    { Hauteur en lignes }

{ Ecrit ler enregistrement }
 BlockWrite (Fic,Buffer,1);

{ Ecrit le reste }
 RWGra (AdresseVideo (x,y),True);

 Close (Fic)
 End;

PROCEDURE LireGraph (Nom : NomFic; VAR x,y,x1,y1 : Integer);
Begin
{ Ouvre le fichier }
 Assign (Fic,Nom);
 Reset (Fic);
{ Lecture des informations du ler enregistrement }
 BlockRead (Fic,Buffer,1);
 x :=Buffer[1]+256*Buffer[2];  { Position X,Y du point en bas à gauche}
 y :=Buffer[3]+256*Buffer[4];
```

```
 x1 :=Pred(x+8*Buffer[5]);      { Calcule les coordonnées du point... }
 y1 :=y+2*Buffer[6];            { ...en haut à droite }

{ Lecture du reste }
 RWGra (AdresseVideo (x,y),False);

 Close (Fic)
End;
```

RECHARGEMENT DES IMAGES PARTIELLES EN BASIC

Voici maintenant le listing d'un petit programme BASIC permettant de recharger les images sauvées avec les procédures PASCAL précédentes. De plus, on peut au préalable effectuer le chargement d'une image de fond et recadrer l'emplacement de l'image.

Il faut répondre à 5 questions :

1. Mode Ecran ?

   Sélectionner le mode de travail.

2. Nom du graphisme ?

   Nom de l'image crée sous PASCAL.

3. Nom du dessin de fond ?

   Nom d'une image au format standard AMSDOS à charger au préalable. Cela permet d'avoir une image de fond. Cette option est facultative, il suffit de répondre par RETURN pour la désactiver.

4. Nom du dessin à écrire ?

   C'est le nom sous lequel sera sauvée l'image complète. Cette option est facultative, si l'on répond par RETURN, aucune sauvegarde ne sera effectuée.

5. Clavier ou Joystick ?

Lors du chargement de la portion d'image, seul le cadre de l'image apparaît, on peut alors le déplacer sur l'écran afin de choisir une nouvelle position d'affichage. Cette option permet de choisir si l'on désire effectuer le déplacement à l'aide du joystick ou du clavier. Le déplacement s'effectue avec les flèches du curseur ou le joystick ; pour valider la position il faut utiliser la barre d'espace ou un bouton Feu du joystick.

Il faut compiler le programme Assembleur dont le listing source est donné à la suite dans un fichier code nommé CHARGEGR.BIN. Ce programme reprend l'algorithme décrit précédemment pour le chargement de l'image.

Note : Le programme utilise l'instruction MASK qui n'existe pas sur le CPC 464, il suffit de retirer la ligne 600 pour ce modèle. Cette instruction ne sert qu'à agrémenter la présentation du cadre du dessin.

```
10 MODE 2
20 DEFINT a-z
30 Base=&4000
40 PRINT CHR$(24);SPC(13);" Utilitaire pour recharger les graphismes
crées sous CP/M";CHR$(18)
50 PRINT
60 MEMORY Base-1
70 IF PEEK(Base)<>&C3 THEN LOAD"CHARGEGR.BIN",Base
80 PRINT" Mode ecran (0,1,2)................."
90 PRINT" Nom du graphisme ..................."
100 PRINT" Nom du dessin de fond (facultatif)."
110 PRINT" Nom du dessin a ecrire (facultatif)"
120 PRINT" Clavier ou Joystick (C/J).........";CHR$(24)
130 LOCATE 37,4 :INPUT Larg
140 Larg=Larg AND 3 :xpas=2^(2-Larg)
150 LOCATE 37,5 :INPUT NomGra$
160 NomGra$=NomGra$
170 LOCATE 37,6 :INPUT InDessin$
180 LOCATE 37,7 :INPUT OutDessin$
190 LOCATE 37,8 :INPUT organe$
200 organe$=UPPER$(LEFT$(organe$,1))
210 IF(organe$<>"C")AND(organe$<>"J")THEN 190
220 IF organe$="C"THEN RESTORE 660 ELSE RESTORE 670
230 READ KeyH,KeyB,KeyG,KeyD,KeyStop
240 IF OutDessin$<>""THEN PRINT :PRINT"Apres le signal,insérez le disque
```

```
qui contiendra :";OutDessin$
250 PRINT"...Appuyer sur une touche pour commencer..."
260 CALL &BB18
270 MODE Larg
280 IF InDessin$<>""THEN LOAD InDessin$,&C000
290 x=0 :y=0 :ll=0 :hh=0
300 CALL Base,@NomGra$,@x,@y,@ll,@hh
310 GOSUB 390
320 CALL Base+3,x\xpas,y\2
330 PRINT CHR$(7); :BORDER 1,26
340 WHILE INKEY$<>"" :WEND
350 CALL &BB18
360 BORDER 1,1
370 IF OutDessin$<>""THEN SAVE OutDessin$,b,&C000,&4000
380 END
390 'Selection de la position ecran
400 'convertit en coord.cadre
410 x=x-xpas :larg=8*ll+xpas
420 y=y-2 :haut=2*(hh+1)
430 masque=7
440 PRINT CHR$(23);CHR$(1) : 'Mode graphique XOR
450 WHILE INKEY(KeyStop)<>0
460 GOSUB 590 :   'Affiche
470 GOSUB 590 :   'Efface
480 IF(INKEY(KeyH)=0)AND(y+haut<400)THEN y=y+2
490 IF(INKEY(KeyB)=0)AND(y>=0)THEN y=y-2
500 IF(INKEY(KeyG)=0)AND(x>=8-xpas)THEN x=x-8
510 IF(INKEY(KeyD)=0)AND(x+larg<640)THEN x=x+8
520 masque=masque*2 :IF masque>255 THEN masque=7
530 WEND
540 PRINT CHR$(23);CHR$(4) : 'Mode graphique normal
550 'convertit en coordonnées dessin
560 x=x+xpas :y=y+2
570 RETURN
580 'Dessin du cadre
590 MOVE x,y
600 MASK Masque :  'Pas pour CPC 464
610 DRAWR 0,haut
620 DRAWR larg,0
630 DRAWR 0,-haut
640 DRAWR-larg,0
```

```
650 RETURN
660 DATA 0,2,8,1,47      :'Fleches et barre espace
670 DATA 72,73,74,75,76 :'Joystick et Fire
```

Voici maintenant le listing source du programme Assembleur, il faut que le code soit dans le fichier CHARGEGR.BIN.

```
                ;Chargeur de graphismes
                ;
                PrevLi :  Equ  #BC29            ;Scr Prev Line
                InOpen :  Equ  #BC77            ;Disc In Open
                InChar :  Equ  #BC80            ;Disc In Char
                InClos :  Equ  #BC7A            ;Disc In Close
                DotPos :  Equ  #BC1D            ;Scr Dot Position
                GetMod :  Equ  #BC11            ;Scr Get Mode
                ;
                        Org  #4000
                ;
                ;Bloc d'interface avec le BASIC :
C3 06 40                Jp   Do_1               ;Charge le 1er enregistrement.
C3 26 40                Jp   Do_2               ;Charge le graphisme

                ;-------------------------------------------------
                ; Ouvre le fichier et charge le 1er enregistrement.
                ; Celui-ci contient les renseignements sur le dessin.
                ;-------------------------------------------------
                ;saisit parametres du BASIC
FE 05           Do_1 :    Cp   5
C0                      Ret  NZ
DD 6E 08                Ld   L,(IX+08)          ;Paramètre adresse descripteur
DD 66 09                Ld   H,(IX+09)
46                      Ld   B,(HL)             ;Longueur de la chaîne
23                      Inc  HL                 ;Sur adresse de la chaîne
7E                      Ld   A,(HL)
23                      Inc  HL
66                      Ld   H,(HL)
6F                      Ld   L,A                ;HL=adresse de la chaîne
                ;Ouvre le fichier
11 78 41                Ld   DE,discbuffer
DD E5                   Push IX
CD 77 BC                Call InOpen             ;Ne charge pas le Header AMSDOS
```

```
                ;Extrait renseignements de l'en-tête
CD E3 40        Call LireSecteur        ;Charge 1er enregistrement.
DD E1           Pop  IX
CD AE 40        Call initdatas
C9              Ret

                ;-------------------------------
                ;Effectue le chargement complet.
                ;-------------------------------
FE 02    Do_2 :   Cp   2
C0              Ret  NZ
DD 6E 00        Ld   L,(IX+00)          ;Paramètres X,Y
DD 66 01        Ld   H,(IX+01)
DD 5E 02        Ld   E,(IX+02)
DD 56 03        Ld   D,(IX+03)
CD 1D BC        Call DotPos             ;Adresse écran ?
22 F6 40        Ld   (Adr),HL
AF              Xor  A
32 F3 40        Ld   (VideoOff),A
32 F4 40        Ld   (Lignes),A
32 F5 40        Ld   (BuffLibre),A
CD 4B 40        Call Load
C3 7A BC        Jp   InClose

                ;-------------------------------
                ; Sous programmes
                ;-------------------------------

                load :
3A F5 40 lod0 :    Ld   A,(BuffLibre)
B7              Or   A
20 08           Jr   NZ,lod1
CD E3 40        Call LireSec
3E 80           Ld   A,128
32 F5 40        Ld   (BuffLibre),A
4F       lod1 :    Ld   C,A
3A F3 40        Ld   A,(VideoOff)
47              Ld   B,A
3A F0 40        Ld   A,(lg)
90              Sub  B
B9              Cp   C
```

```
38 01             Jr   C,QS_1
79                Ld   A,C
32 F2 40 QS_1 :    Ld   (Nombre),A
2A F6 40          Ld   HL,(Adr)
58                Ld   E,B
16 00             Ld   D,#00
19                Add  HL,DE
E5                Push HL
21 78 41          Ld   HL,Buffer+128
59                Ld   E,C
B7                Or   A
ED 52             Sbc  HL,DE
D1                Pop  DE
4F                Ld   C,A
80                Add  A,B
06 00             Ld   B,#00
ED B0             Ldir
47                Ld   B,A
3A F0 40          Ld   A,(lg)
B8                Cp   B
78                Ld   A,B
20 0E             Jr   NZ,Tr_1
2A F6 40          Ld   HL,(Adr)
CD 29 BC          Call PrevLine
22 F6 40          Ld   (Adr),HL
21 F4 40          Ld   HL,Lignes
34                Inc  (HL)
AF                Xor  A
32 F3 40 Tr_1 :    Ld   (VideoOff),A
3A F2 40          Ld   A,(Nombre)
4F                Ld   C,A
3A F5 40          Ld   A,(BuffLibre)
91                Sub  C
32 F5 40          Ld   (BuffLibre),A
3A F1 40          Ld   A,(ht)
4F                Ld   C,A
3A F4 40          Ld   A,(Lignes)
B9                Cp   C
38 9E             Jr   C,lod0
C9                Ret
        ;
```

```
          initdatas :
DD 6E 00            Ld   L,(IX+00)          ;ht
DD 66 01            Ld   H,(IX+01)
3A FD 40            Ld   A,(Buffer+5)
77                  Ld   (HL),A
32 F1 40            Ld   (ht),A
DD 6E 02            Ld   L,(IX+02)          ;lg
DD 66 03            Ld   H,(IX+03)
3A FC 40            Ld   A,(Buffer+4)
77                  Ld   (HL),A
32 F0 40            Ld   (lg),A
DD 5E 04            Ld   E,(IX+04)          ;y
DD 56 05            Ld   D,(IX+05)
21 FA 40            Ld   HL,Buffer+02
ED A0               Ldi
ED A0               Ldi
DD 5E 06            Ld   E,(IX+06)          ;x
DD 56 07            Ld   D,(IX+07)
21 F8 40            Ld   HL,Buffer+00
ED A0               Ldi
ED A0               Ldi
C9                  Ret

          ;----------------------------------------
          ; Lecture d'un enregistrement.
          ; Voir 'Manipulation des fichiers binaires'.
          ;----------------------------------------
06 80     LireSec : Ld   B,128
21 F8 40            Ld   HL,Buffer
CD 80 BC lire_1 :  Call InChar
77                  Ld   (HL),A
23                  Inc  HL
10 F9               Djnz lire_1
C9                  Ret
          ;
          lg :      Defs 1
          ht :      Defs 1
          Nombre :  Defs 1
          VideoOff :Defs 1
          Lignes :  Defs 1
          BuffLibr :Defs 1
```

```
Adr :      Defs 2
Buffer :   Defs 128
Discbuf :  Defs 2048              ;Buffer pour DISC IN CHAR
           End
```

UTILISATION DE LUTINS GRAPHIQUES

Pour clore cet ouvrage, nous allons voir un exemple d'utilisation de la mémoire vidéo à l'aide d'une routine assembleur appelée à partir du TURBO PASCAL. Les lutins ou *sprites* en anglais sont des formes de plusieurs couleurs se déplaçant sur l'écran sans détruire le contenu de celui-ci, le lutin donne l'impression de se mouvoir par-dessus le fond de l'écran. Le programme suivant donne un exemple d'utilisation de lutins, il est tiré d'un logiciel d'édition graphique : ainsi les formes proposées sont un stylo, une gomme et une flèche. Vous pouvez très simplement rajouter d'autres formes, de plus le programme est capable de travailler en 80 et en 40 colonnes (modes 2 et 1) et utilise des masques. Ceux-ci permettent de bien détacher la forme du fond sur lequel elle se déplace, en effet si la forme est entièrement noire et que le fond est lui aussi noir, on ne verra plus rien ! Le remède est de laisser un liseré blanc tout autour de la forme.

Le principe de fonctionnement est très simple, les formes sont définies dans des zones de 2 octets sur 16 lignes, soit en largeur 8 pixels en 40 colonnes et 16 pixels en 80 colonnes. Lors de l'écriture d'une forme, le fond de l'écran à l'endroit choisi est préalablement sauvegardé puis la programme applique un ET (AND) entre le fond et le masque de la forme, les bits à 1 du masque seront conservés sur l'écran, les bits à 0 mettront les bits correspondants sur l'écran à 0, ensuite un OU (OR) est appliqué avec la forme, les bits à 1 de la forme seront à 1 sur l'écran. Pour que l'effet de masque soit apparent, il faut que celui-ci soit légèrement plus grand que la forme à dessiner. Si la forme doit être déplacée, le fond sauvegardé est alors replacé sur l'écran et la même traitement recommence.

Rappelons pour les néophytes qu'en mode 2, un bit à 0 donnera un point de couleur 0 (INK 0) et un bit à 1 de couleur 1, un octet

est utilisé pour 8 pixels donc. En mode 1, cela se complique un peu, chaque octet ne correspond plus à 8 pixels mais à 4. 2 bits sont nécessaires pour coder la couleur de chaque pixel selon la table suivante :

Bit	Pixel
7	1
6	2
5	3
4	4
3	1
2	2
1	3
0	4

Les couleurs sont codées de la façon suivante :

Couleur	Bits
0	0 0
1	1 0
2	0 1
3	1 1

Ainsi si l'on veut 4 points de couleurs 1,2,0 et 3, l'octet sera codé :

1001 0101

Soit :

1 0 pour le 1er pixel.
0 1 pour le 2e pixel.
0 0 pour le 3e pixel.
1 1 pour le 4e pixel.

Les formes et les masques sont codés directement en mémoire suivant ce principe, la table se compose d'abord des formes pour le mode 80 colonnes puis celles pour le mode 40 colonnes. Pour chaque forme, le masque est stocké avant le dessin de la forme, soit deux fois 32 octets. Pour rajouter une forme, il faut stocker

dans les tables pour les deux modes les octets de masque et de forme de la même façon.

Comme le lutin peut être placé n'importe où sur l'écran, il y a peu de chance pour que la forme utilise exactement 2 octets en largeur dans la mémoire vidéo mais en fait 3 octets, le premier et le dernier n'étant pas utilisés en entier. Il faut donc au préalable décaler les octets de masque et de forme avant d'appliquer le traitement sur l'écran. Lors du décalage, il faut noter qu'il faut prévoir un buffer de 3 octets de large pour stocker les formes décalées. Une autre méthode plus gourmande en mémoire est de stocker toutes les formes décalées en mémoire, ce qui multiplierait la place occupée par les tables mais le programme ne perdrait plus de temps à effectuer les décalages.

Il y a en outre un dernier petit problème, le rayon du tube cathodique de l'écran balaie celui-ci de haut en bas 50 fois par seconde ; si certains traitements s'effectuent dans le haut de l'écran, il est fort probable que ceux-ci ne seront jamais correctement visualisés si le rayon vidéo est en train d'afficher le bas de l'écran ! Il faudra donc synchroniser certaines opérations avec le rayon : cela est possible car l'instant où le rayon doit remonter du bas vers le haut de l'écran (le *frame flyback* en anglais) peut être détecté par le système. Comme ce retour prend un certain temps et que rien est affiché pendant ce temps, on en profitera pour effectuer les opérations d'écriture sur l'écran, celles-ci devront être assez rapides car elles devront être terminées avant que le rayon ne commence l'affichage de la première ligne.

Pour ces impératifs de rapidité, la gestion des lutins est écrite en Assembleur et certaines routines utiles du Firmware ne seront pas appelées mais sont directement recopiées dans le programme (MC WAIT FLYBACK et SCR PREV LINE), évitant ainsi de perdre du temps avec les commutations des ROM.

En CP/M Plus, pour que le programme Assembleur accède à la mémoire vidéo, il devra être placé en zone non commutée de la mémoire (adresse supérieure à $C000) et appelé à l'aide de la fonction BIOS UserF (voir 'Accès à la mémoire vidéo sous CP/M Plus'). sous CP/M 2.2, l'accès à la mémoire vidéo ne peut se faire que si la ROM BIOS est déconnectée (voir Chapitre 2 : 'Accès à la mémoire vidéo, sous CP/M 2.2). Le listing source suivant a été obtenu à l'aide d'un assembleur classique sous AMSDOS et le

fichier code est donc au format AMSDOS. Il fait en tout 9 enregistrements de 128 octets de long, en-tête AMSDOS comprise, le programme TURBO PASCAL chargera celui-ci le plus haut possible dans la mémoire de travail (TPA), soit $97C0 sous CP/M 2.2 et $EEC0 sous CP/M Plus, le lanceur du compilateur étant préservé (voir Chapitre 4 : 'Manipulation des fichiers binaires'). Si vous avez modifié le nombre de formes, il faudra alors recalculer les adresses d'implantation.

Listing du programme Assembleur

Le programme possède trois entrées : (Soit @ l'adresse de départ du programme.)

@+0 : Sauve le fond, place la forme.
Doit être utilisée pour placer le lutin la première fois.
@+3 : Remet l'ancien fond, sauve le nouveau fond et place la forme.
Pour placer le lutin qui doit déjà être sur l'écran.
@+6 : Remet l'ancien fond.
Pour effacer le lutin qui doit déjà être sur l'écran.

Note importante

Les adresses du listing sont celles obtenues pour la version CP/M Plus (implantation en $EEC0).

```
;*****************************************************
;* ROUTINE ASSEMBLEUR.                               *
;* Assembler dans un fichier code SPRITE.BIN         *
;*                                                   *
;* Paramètres en entrée :                             *
;* DE=x standard (0 à 639) du point en bas à gauche  *
;* HL=y standard (0 à 399) du point en bas à gauche  *
;* A =numéro forme (1 à 3)                           *
;*****************************************************
```

```
                ;--- Routines FIRMWARE utilisées ----------------------

BC1D            DotPos : Equ  #BC1D        ;SCR DOT POSITION
                                           ;Adresse vidéo dans HL.
                                           ;Masque bits dans C.
BC11            GetMod : Equ  #BC11        ;SCR GET MODE
                                           ;Mode écran dans A.

                ;*** Pour CP/M 2.2 seulement ***
                RomDis : Equ  #B903        ;KL U ROM DISABLE
                                           ;déconnecte ROM BIOS
                RomRes : Equ  #B90C        ;KL ROM RESTORE
                                           ;restaure état ROM BIOS
                ;***

546D                    Org   #97C0        ;Pour CP/M 2.2
EEC0                    Org   #EEC0        ;Pour CP/M Plus

                ;--- Interface du programme ---------------------------

EEC0  C3C9EE            Jp    INIT         ;Entrée 1
EEC3  C3D5EE            Jp    SET          ;Entrée 2
EEC6  C342EF            Jp    R_FOND       ;Entrée 3

                ;--- ENTREE 1 -----------------------------------------

EEC9  CD23EF    INIT :   Call PARAM        ;Saisit paramètres.
EECC  E5                Push HL           ;Sauve adresse vidéo.
EECD  CD5FEF            Call DECALE       ;Décale formes et masques.
EED0  CD3AEF            Call WAITFLY      ;Attend retour rayon.
EED3  180A              Jr   RUN_1        ;Traitement.

                ;--- ENTREE 2 -----------------------------------------

EED5  CD23EF    SET :    Call PARAM        ;Saisit paramètres.
EED8  E5                Push HL           ;Sauve adresse vidéo.
EED9  CD5FEF            Call DECALE       ;Décale formes et masques.
EEDC  CD42EF            Call R_FOND       ;Retour rayon puis remet
                                          ;ancien fond.
```

```
                ;--- Traitement principal ------------------------------

EEDF  E1        RUN_1 :  Pop  HL

                ;*** Pour CP/M 2.2 seulement ***
EEE0                     Call RomDis       ;Déconnecte ROM BIOS.
EEE0                     Push AF           ;Sauve ancienne sélection ROM.
                ;***

EEE0  2291F0             Ld   (SAVPOS),HL  ;Sauve adresse vidéo.
EEE3  1193F0             Ld   DE,SAVE      ;Sur buffer de sauvegarde.
EEE6  DD21C3F0           Ld   IX,FORM_B    ;Sur buffer des décalages.
EEEA  010310             Ld   BC,#1003     ;B=16 (Nombre lignes),
                                           ;C=3 (largeur en octets).

                ;Traite les 16 lignes.
EEED  C5        L00 :     Push BC          ;Sauve compteurs.
EEEE  E5                 Push HL           ;Sauve adresse à gauche.
EEEF  41                 Ld   B,C          ;Reprend nombre d'octets (3).

                ;Traite une ligne de 3 octets.
EEF0  7E        L01 :     Ld   A,(HL)      ;Lit un octet vidéo...
EEF1  12                 Ld   (DE),A       ;...sauve dans buffer SAVE...
EEF2  DDA630             And  (IX+48)      ;...AND avec masque décalé...
EEF5  DDB600             Or   (IX+00)      ;...OR avec dessin décalé...
EEF8  77                 Ld   (HL),A       ;Replace dans RAM vidéo.
EEF9  DD23               Inc  IX           ;Pointeurs +1.
EEFB  23                 Inc  HL
EEFC  13                 Inc  DE
EEFD  10F1               Djnz L01          ;Boucle.

EEFF  E1                 Pop  HL           ;Récupère adresse à gauche.
EF00  CD07EF             Call PREVLINE     ;Adresse ligne au dessus ?
EF03  C1                 Pop  BC           ;Récupère compteurs.
EF04  10E7               Djnz L00          ;Boucle.

                ;*** Pour CP/M 2.2 seulement ***
EF06                     Pop  AF           ;Rappelle ancienne sélect. ROM
EF06                     Call RomRes       ;Rétablit ROM BIOS.
                ;***
EF06  C9                 Ret               ;FIN DU PROGRAMME.
```

```
                ;--- Calcul de l'adresse vidéo de l'octet 'au dessus' --
                ; La routine est identique à celle en ROM Firmware (SCR-
                ;PREV-LINE).

EF07  7C        PREVLI : Ld   A,H
EF08  D608              Sub  #08
EF0A  67                Ld   H,A
EF0B  E638              And  #38
EF0D  FE38              Cp   #38
EF0F  C0                Ret  NZ
EF10  7C                Ld   A,H
EF11  C640              Add  A,#40
EF13  67                Ld   H,A
EF14  7D                Ld   A,L
EF15  D650              Sub  #50
EF17  6F                Ld   L,A
EF18  D0                Ret  NC
EF19  7C                Ld   A,H
EF1A  25                Dec  H
EF1B  E607              And  #07
EF1D  C0                Ret  NZ
EF1E  7C                Ld   A,H
EF1F  C608              Add  A,#08
EF21  67                Ld   H,A
EF22  C9                Ret

                ;--- Saisie et traitement des paramètres en entrée -----

EF23  328FF0    PARAM :  Ld   (ICONE),A    ;Sauve numéro forme.
EF26  CD11BC             Call GetMode      ;Numéro mode écran dans A.
EF29  3290F0             Ld   (MODE),A     ;Sauve.
                ;Convertit x,y en coordonnées de base (1 point=1 pixel).
EF2C  3D                 Dec  A
EF2D  2004               Jr   NZ,PAR_0     ;Si Mode 2
EF2F  CB3A               Srl  D            ;Mode 1--> X :=X DIV 2
EF31  CB1B               Rr   E
EF33  CB3C      PAR_0 :  Srl  H             ;Y :=Y DIV 2
EF35  CB1D               Rr   L
EF37  C31DBC             Jp   DotPos       ;C=masque. HL=adresse vidéo.
```

```
                ;--- Attend le retour du rayon vidéo ------------------
                ; La routine est  identique  à  celle  en  ROM  Firmware
                ;(MC-WAIT-FLYBACK).
                ;
EF3A  06F5      WAITFL : Ld   B,#F5
EF3C  ED78      flybac : In   A,(C)
EF3E  1F                 Rra
EF3F  30FB               Jr   NC,flyback
EF41  C9                 Ret

                ;--- ENTREE 3 ----------------------------------------

                ;Remet le contenu du fond sauvegardé.

EF42  ED5B91F0  R_FOND : Ld   DE,(SAVPOS)  ;Adresse vidéo dans DE.
EF46  2193F0             Ld   HL,SAVE      ;Sur buffer de sauvegarde.
EF49  CD3AEF             Call WAITFLY      ;Attend retour rayon.
EF4C  010310             Ld   BC,#1003     ;B=16 (Nombre lignes),
                                           ;C=3 (largeur en octets).
EF4F  C5        r_f1 :    Push BC           ;Sauve compteurs.
EF50  D5                 Push DE           ;Sauve adresse vidéo à gauche.
EF51  0600               Ld   B,#00        ;BC=3.
EF53  EDB0               Ldir               ;Remet une ligne de 3 octets.
EF55  D1                 Pop  DE           ;Récupère adresse vidéo.
EF56  EB                 Ex   DE,HL        ;Ligne au dessus ?
EF57  CD07EF             Call PREVLINE     ;(travaille avec HL)
EF5A  EB                 Ex   DE,HL
EF5B  C1                 Pop  BC           ;Récupère compteurs.
EF5C  10F1               Djnz r_f1         ;Boucle
EF5E  C9                 Ret

                ;--- Décalages des masques et des formes dans le  buffer
                ;--- FORM_B.

                ;Le  décalage  se  fait  selon  la  valeur  du  masque
                ;(registre C) obtenu  par  la  routine  Firmware  DotPos
                ;(voir 'PARAM'). Le décalage est optimisé,  par  exemple
                ;il vaut mieux faire 1 décalage à gauche que 7 décalages
                ;à droite.
                ;Note : Ne modifie pas C.
```

```
                ;Recherche l'adresse de la forme choisie.
EF5F  3A8FF0    DECALE : Ld   A,(ICONE)
EF62  47                 Ld   B,A           ;Numéro de la forme dans B.
EF63  21E3F0             Ld   HL,FORM_2-64 ;Table pour mode 2.
EF66  3A90F0             Ld   A,(MODE)
EF69  3D                 Dec  A             ;Mode 1-->Z=1
EF6A  F5                 Push AF            ;Sauve Z.
EF6B  2003               Jr   NZ,DEC_00     ;Si mode 2.
EF6D  21A3F1             Ld   HL,FORM_1-64 ;Table pour mode 1.

EF70  114000    DEC_00 : Ld   DE,64         ;Calcule adresse dans HL.
EF73  19        DEC_01 : Add  HL,DE
EF74  10FD               Djnz DEC_01
EF76  E5                 Push HL
EF77  DDE1               Pop  IX            ;Adresse table dans IX.

EF79  2122F1             Ld   HL,FORM_B+95 ;Dernier octet du buffer
EF7C  0610               Ld   B,16          ;Compteur (16 lignes).
EF7E  F1                 Pop  AF            ;Récupère flag Z (1 si mode 1)
EF7F  79                 Ld   A,C           ;A=masque bit du départ.
EF80  2862               Jr   Z,DEC_m1      ;Si mode 1.

                ;*** Décalages pour le mode 2 ***

EF82  E60F               And  #0F           ;Décale à droite ou à gauche ?
EF84  282D               Jr   Z,DEC_d1      ;Si à droite.

                ;DECALAGES A GAUCHE (1 à 4 fois).

                ;Décale le masque à gauche.
EF86  3EFF      DEC_g1 : Ld   A,#FF         ;Bits à 1 par défaut.
EF88  CD06F0             Call DEC_s1        ;Saisit 2 octets du masque.
EF8B  C5                 Push BC
                ;Décale DE dans ADE.

EF8C  37        DEC_g2 : Scf                ;Rentre 1 par la droite...
EF8D  CB13               Rl   E             ;...de l'octet.
EF8F  CB12               Rl   D
EF91  17                 Rla
EF92  CB19               Rr   C             ;Test fin des décalages.
EF94  30F6               Jr   NC,DEC_g2
```

```
EF96  C1              Pop  BC
EF97  CD11F0          Call DEC_s2       ;Place dans buffer de décalage
EF9A  10EA            Djnz DEC_g1       ;16 tours.
                ;IX et HL bien positionnés dans buffer de décalage et
                table des formes.

                ;Décale le dessin à gauche.
EF9C  0610            Ld   B,16         ;16 tours.
EF9E  AF        DEC_g3 : Xor  A          ;Bits à 0 par défaut.
EF9F  CD06F0          Call DEC_s1       ;Saisit 2 octets du dessin.
EFA2  C5              Push BC
                ;Décale DE dans ADE.
EFA3  CB13      DEC_g4 : RL   E          ;Rentre 0 par la droite...
EFA5  CB12            Rl   D            ;... car Carry=0 avec XOR A.
EFA7  17              Rla
EFA8  CB19            Rr   C            ;Test fin des décalages.
EFAA  30F7            Jr   NC,DEC_g4
EFAC  C1              Pop  BC
EFAD  CD11F0          Call DEC_s2       ;Place dans buffer de décalage
EFB0  10EC            Djnz DEC_g3
EFB2  C9              Ret               ;FIN

                ;DECALAGES A DROITE (0 à 3 fois).

                ;Décale le masque à droite.
EFB3  3EFF      DEC_d1 : Ld   A,#FF      ;Bits à 1 par défaut.
EFB5  CD06F0          Call DEC_s1       ;Saisit 2 octets du masque.
EFB8  C5              Push BC
EFB9  1806            Jr   DEC_d3
                ;Décale DE dans DEA.
EFBB  37        DEC_d2 : Scf             ;Rentre 1 par la gauche.
EFBC  CB1A            Rr   D
EFBE  CB1B            Rr   E
EFC0  1F              Rra
EFC1  CB11      DEC_d3 : Rl   C          ;Test fin des décalages.
EFC3  30F6            Jr   NC,DEC_d2
EFC5  C1              Pop  BC
EFC6  CD18F0          Call DEC_s3       ;Place dans buffer de décalage
EFC9  10E8            Djnz DEC_d1       ;16 tours.

                ;Décale le dessin à droite.
```

```
EFCB  0610            Ld   B,16          ;16 tours.
EFCD  AF        DEC_d4 : Xor  A           ;Bits à 0 par défaut.
EFCE  CD06F0           Call DEC_s1        ;Saisit 2 octets du dessin.
EFD1  C5               Push BC
EFD2  1805             Jr   DEC_d6
                ;Décale DE dans DEA.
EFD4  CB1A      DEC_d5 : Rr   D           ;CY=0 : rentre des 0 par la...
EFD6  CB1B             Rr   E             ;...gauche.
EFD8  1F               Rra
EFD9  CB11      DEC_d6 : Rl   C           ;Test fin des décalages.
EFDB  30F7             Jr   NC,DEC_d5
EFDD  C1               Pop  BC
EFDE  CD18F0           Call DEC_s3        ;Place dans buffer de décalage
EFE1  10EA             Djnz DEC_d4        ;16 tours.
EFE3  C9               Ret                ;FIN

                ;*** Décalages pour le mode 1 ***

EFE4  E6C0      DEC_m1 : And  #C0          ;Décale à droite ou à gauche
?
EFE6  200F             Jr   NZ,d_m10      ;Si à droite.

                ;DECALAGES A GAUCHE (1 ou 2 fois).

EFE8  1111FF           Ld   DE,#FF11
EFEB  CD2EF0           Call DEC_s5        ;Décale masque.

EFEE  0610             Ld   B,16
EFF0  110000           Ld   DE,#0000
EFF3  CD2EF0           Call DEC_s5        ;Décale dessin.
EFF6  C9               Ret

                ;DECALAGES A DROITE (0 ou 1 fois).

EFF7  1188FF    d_m10 :  Ld   DE,#FF88
EFFA  CD65F0           Call DEC_s7        ;Décale masque.

EFFD  0610             Ld   B,16
EFFF  110000           Ld   DE,#0000
F002  CD65F0           Call DEC_s7        ;Décale dessin.
F005  C9               Ret
```

```
                    ;*** Sous-programmes pour les décalages ***

                    ;Saisit 2 octets de la table des formes dans DE.

F006  DD5600    DEC_s1 : Ld   D,(IX+00)
F009  DD5E01             Ld   E,(IX+01)
F00C  DD23               Inc  IX
F00E  DD23               Inc  IX
F010  C9                 Ret

                    ;Place ADE dans buffer des décalages.
F011  73        DEC_s2 : Ld   (HL),E
F012  2B                 Dec  HL
F013  72                 Ld   (HL),D
F014  2B                 Dec  HL
F015  77                 Ld   (HL),A
F016  2B                 Dec  HL
F017  C9                 Ret

                    ;Place DEA dans buffer des décalages.
F018  77        DEC_s3 : Ld   (HL),A
F019  2B                 Dec  HL
F01A  73                 Ld   (HL),E
F01B  2B                 Dec  HL
F01C  72                 Ld   (HL),D
F01D  2B                 Dec  HL
F01E  C9                 Ret

                    ;Décalages à gauche en mode 1.

                    ;Décale A à gauche selon l'organisation en mode 1.
                    ;Résultat dans B. Retenue dans L.
F01F  47        DEC_s4 : Ld   B,A          ;Sauve A.
F020  17                 Rla               ;Décale A.
F021  E6EE               And  %11101110    ;Applique...
F023  B5                 Or   L            ;...la retenue (registre L).
F024  6F                 Ld   L,A          ;Sauve resultat.
F025  78                 Ld   A,B          ;Récupère A initial.
F026  45                 Ld   B,L          ;Récupère retenue dans B.
F027  E688               And  %10001000    ;Calcule nouvelle retenue
```

```
F029  1F              Rra
F02A  1F              Rra
F02B  1F              Rra
F02C  6F              Ld   L,A          ;Nouvelle retenue dans L.
F02D  C9              Ret

                ;Décalages à gauche en mode 1, en paramètre D=3e octet
                 ;pour décalages et E=retenue.
F02E  D5        DEC_s5 : Push DE          ;Sauve 3e octet et retenue.
F02F  E5              Push HL           ;Ptr dans buffer des décalages
F030  C5              Push BC           ;Compteur lignes et masque bit
F031  43              Ld   B,E          ;B=retenue.
F032  62              Ld   H,D          ;H=3e octet.
F033  CD06F0          Call DEC_s1       ;Saisit 2 octets de la table
                                        ;dans DE.
                 ;Décale HDE, L est la retenue.
F036  68        d_s50 :  Ld   L,B         ;L= retenue par défaut.
F037  C5              Push BC           ;Sauve retenue (B).
F038  7B              Ld   A,E          ;Décale E.
F039  CD1FF0          Call DEC_s4
F03C  58              Ld   E,B
F03D  7A              Ld   A,D          ;Décale D.
F03E  CD1FF0          Call DEC_s4
F041  50              Ld   D,B
F042  7C              Ld   A,H          ;Décale H.
F043  CD1FF0          Call DEC_s4
F046  60              Ld   H,B
F047  C1              Pop  BC           ;Récupère retenue (B).
F048  CB19            Rr   C            ;Test fin des décalages.
F04A  30EA            Jr   NC,d_s50

F04C  C1              Pop  BC           ;Compteur ligne et masque bit.
F04D  7C              Ld   A,H          ;Sauve 3e octet dans A.
F04E  E1              Pop  HL           ;Récupère ptr dans buffer déc.
F04F  CD11F0          Call DEC_s2       ;Place ADE dans le buffer.
F052  D1              Pop  DE           ;Récup. 3e octet et retenue.
F053  10D9            Djnz DEC_s5       ;16 tours.
F055  C9              Ret

                ;Décalages à droite en mode 1.
```

```
                ;Décale A à droite selon l'organisation en mode 1.
                ;Résultat dans B. Retenue dans H.
F056  47        DEC_s6 : Ld   B,A          ;Sauve A.
F057  1F                 Rra               ;Décale A.
F058  E677               And  %01110111    ;Applique...
F05A  B4                 Or   H            ;...la retenue (registre H).
F05B  67                 Ld   H,A          ;Sauve résultat.
F05C  78                 Ld   A,B          ;Récupère A initial.
F05D  44                 Ld   B,H          ;Récupère retenue dans B.
F05E  E611               And  %00010001    ;Calcule nouvelle retenue.
F060  17                 Rla
F061  17                 Rla
F062  17                 Rla
F063  67                 Ld   H,A          ;Nouvelle retenue dans H.
F064  C9                 Ret

               ;Décalages à droite en mode 1, en paramètre D=3e octet
                ;pour décalages et E=retenue.
F065  D5        DEC_s7 : Push DE           ;Sauve 3e octet et retenue.
F066  E5                 Push HL           ;Ptr dans buffer des décalages
F067  C5                 Push BC           ;Compteur lignes et masque bit
F068  43                 Ld   B,E          ;B=retenue.
F069  6A                 Ld   L,D          ;L=3e octet.
F06A  CD06F0             Call DEC_s1       ;Saisit 2 octets de la table
                                           ;dans DE.
                ;Décale DEL, H est la retenue.
F06D  1812               Jr   d_s71
F06F  60        d_s70 :  Ld   H,B          ;H= retenue par défaut.
F070  C5                 Push BC           ;Sauve retenue (B).
F071  7A                 Ld   A,D          ;Décale D.
F072  CD56F0             Call DEC_s6
F075  50                 Ld   D,B
F076  7B                 Ld   A,E          ;Décale E.
F077  CD56F0             Call DEC_s6
F07A  58                 Ld   E,B
F07B  7D                 Ld   A,L          ;Décale L.
F07C  CD56F0             Call DEC_s6
F07F  68                 Ld   L,B
F080  C1                 Pop  BC           ;Récupère retenue (B).
F081  CB11      d_s71 :  Rl   C            ;Test fin des décalages.
F083  30EA               Jr   NC,d_s70
```

```
F085  C1              Pop  BC          ;Compteur lignes et masque bit
F086  7D              Ld   A,L         ;Sauve 3e octet dans A.
F087  E1              Pop  HL          ;Récupère ptr dans buffer déc.
F088  CD18F0          Call DEC_s3      ;Place DEA dans buffer déc.
F08B  D1              Pop  DE          ;Récup. 3e octet et retenue.
F08C  10D7            Djnz DEC_s7      ;16 tours.
F08E  C9              Ret

                ;--- Variables ----------------------------------------

F08F            ICONE :  Defs 1         ;Numéro de la forme choisie.
F090            MODE :   Defs 1         ;Mode de l'écran (1 ou 2).

                ;*** Paramètres de la sauvegarde du fond de l'écran.
F091            SAVPOS : Defs 2         ;Adresse du point en bas à
...
                                        ;...gauche.
F093            SAVE :   Defs 48        ;Buffer de sauvegarde (16
                                        ;lignes de 3 octets).

                ;*** Buffers pour les décalages (16 lignes de 3 octets).
F0C3            FORM_B : Defs 48        ;Pour dessin.
F0F3                     Defs 48        ;Pour masque.

                ;*** Formes en mode 2 (80 colonnes). ***
F123            FORM_2 :

                ;FLECHE (masque).
F123  FFEF              Defb %11111111,%11101111
F125  FFC7              Defb %11111111,%11000111
F127  FFC3              Defb %11111111,%11000011
F129  FF80              Defb %11111111,%10000000
F12B  FF01              Defb %11111111,%00000001
F12D  BE07              Defb %10111110,%00000111
F12F  1C0F              Defb %00011100,%00001111
F131  081F              Defb %00001000,%00011111
F133  003F              Defb %00000000,%00111111
F135  001F              Defb %00000000,%00011111
F137  000F              Defb %00000000,%00001111
F139  001F              Defb %00000000,%00011111
```

```
F13B  003F            Defb %00000000,%00111111
F13D  00FF            Defb %00000000,%11111111
F13F  03FF            Defb %00000011,%11111111
F141  9FFF            Defb %10011111,%11111111

                ;FLECHE (dessin).
F143  0000            Defb %00000000,%00000000
F145  0010            Defb %00000000,%00010000
F147  0018            Defb %00000000,%00011000
F149  003E            Defb %00000000,%00111110
F14B  0078            Defb %00000000,%01111000
F14D  00F0            Defb %00000000,%11110000
F14F  41E0            Defb %01000001,%11100000
F151  63C0            Defb %01100011,%11000000
F153  6780            Defb %01100111,%10000000
F155  7F00            Defb %01111111,%00000000
F157  7FE0            Defb %01111111,%11100000
F159  7FC0            Defb %01111111,%11000000
F15B  7F00            Defb %01111111,%00000000
F15D  7C00            Defb %01111100,%00000000
F15F  6000            Defb %01100000,%00000000
F161  0000            Defb %00000000,%00000000

                ;GOMME (masque).
F163  FFFF            Defb %11111111,%11111111
F165  FFFF            Defb %11111111,%11111111
F167  FFFF            Defb %11111111,%11111111
F169  FFFF            Defb %11111111,%11111111
F16B  FFFF            Defb %11111111,%11111111
F16D  FFFF            Defb %11111111,%11111111
F16F  FFFF            Defb %11111111,%11111111
F171  FFFF            Defb %11111111,%11111111
F173  00FF            Defb %00000000,%11111111
F175  00FF            Defb %00000000,%11111111
F177  00FF            Defb %00000000,%11111111
F179  00FF            Defb %00000000,%11111111
F17B  00FF            Defb %00000000,%11111111
F17D  00FF            Defb %00000000,%11111111
F17F  00FF            Defb %00000000,%11111111
F181  00FF            Defb %00000000,%11111111
```

```
                ;GOMME (dessin).
F183  0000              Defb %00000000,%00000000
F185  0000              Defb %00000000,%00000000
F187  0000              Defb %00000000,%00000000
F189  0000              Defb %00000000,%00000000
F18B  0000              Defb %00000000,%00000000
F18D  0000              Defb %00000000,%00000000
F18F  0000              Defb %00000000,%00000000
F191  0000              Defb %00000000,%00000000
F193  8100              Defb %10000001,%00000000
F195  7E00              Defb %01111110,%00000000
F197  4200              Defb %01000010,%00000000
F199  4200              Defb %01000010,%00000000
F19B  4200              Defb %01000010,%00000000
F19D  4200              Defb %01000010,%00000000
F19F  7E00              Defb %01111110,%00000000
F1A1  8100              Defb %10000001,%00000000

                ;STYLO (masque).
F1A3  FFE7              Defb %11111111,%11100111
F1A5  FFC3              Defb %11111111,%11000011
F1A7  FF81              Defb %11111111,%10000001
F1A9  FF00              Defb %11111111,%00000000
F1AB  FE00              Defb %11111110,%00000000
F1AD  FC00              Defb %11111100,%00000000
F1AF  F801              Defb %11111000,%00000001
F1B1  F003              Defb %11110000,%00000011
F1B3  E007              Defb %11100000,%00000111
F1B5  C02F              Defb %11000000,%00101111
F1B7  807F              Defb %10000000,%01111111
F1B9  00FF              Defb %00000000,%11111111
F1BB  01FF              Defb %00000001,%11111111
F1BD  03FF              Defb %00000011,%11111111
F1BF  03FF              Defb %00000011,%11111111
F1C1  87FF              Defb %10000111,%11111111

                ;STYLO (dessin).
F1C3  0000              Defb %00000000,%00000000
F1C5  0018              Defb %00000000,%00011000
F1C7  0024              Defb %00000000,%00100100
F1C9  0042              Defb %00000000,%01000010
```

```
F1CB  0086            Defb %00000000,%10000110
F1CD  014A            Defb %00000001,%01001010
F1CF  0234            Defb %00000010,%00110100
F1D1  0528            Defb %00000101,%00101000
F1D3  0A50            Defb %00001010,%01010000
F1D5  1480            Defb %00010100,%10000000
F1D7  1900            Defb %00011001,%00000000
F1D9  7200            Defb %01110010,%00000000
F1DB  4C00            Defb %01001100,%00000000
F1DD  6800            Defb %01101000,%00000000
F1DF  7800            Defb %01111000,%00000000
F1E1  0000            Defb %00000000,%00000000

                      ;*** Formes en mode 1 (40 colonnes). ***
F1E3                  FORM_1 :

                      ;FLECHE (masque).
F1E3  FFFF            Defb %11111111,%11111111
F1E5  FFFF            Defb %11111111,%11111111
F1E7  FF99            Defb %11111111,%10011001
F1E9  FF00            Defb %11111111,%00000000
F1EB  FF00            Defb %11111111,%00000000
F1ED  EE00            Defb %11101110,%00000000
F1EF  EE11            Defb %11101110,%00010001
F1F1  EE11            Defb %11101110,%00010001
F1F3  0033            Defb %00000000,%00110011
F1F5  0033            Defb %00000000,%00110011
F1F7  0033            Defb %00000000,%00110011
F1F9  0011            Defb %00000000,%00010001
F1FB  0033            Defb %00000000,%00110011
F1FD  0077            Defb %00000000,%01110111
F1FF  00FF            Defb %00000000,%11111111
F201  99FF            Defb %10011001,%11111111

                      ;FLECHE (dessin).
F203  0000            Defb %00000000,%00000000
F205  0000            Defb %00000000,%00000000
F207  0000            Defb %00000000,%00000000
F209  0060            Defb %00000000,%01100000
F20B  0060            Defb %00000000,%01100000
F20D  00E0            Defb %00000000,%11100000
```

```
F20F  00C0            Defb %00000000,%11000000
F211  00C0            Defb %00000000,%11000000
F213  1080            Defb %00010000,%10000000
F215  5080            Defb %01010000,%10000000
F217  7000            Defb %01110000,%00000000
F219  70C0            Defb %01110000,%11000000
F21B  7080            Defb %01110000,%10000000
F21D  7000            Defb %01110000,%00000000
F21F  6000            Defb %01100000,%00000000
F221  0000            Defb %00000000,%00000000

                ;GOMME (masque).
F223  FFFF            Defb %11111111,%11111111
F225  FFFF            Defb %11111111,%11111111
F227  FFFF            Defb %11111111,%11111111
F229  FFFF            Defb %11111111,%11111111
F22B  FFFF            Defb %11111111,%11111111
F22D  FFFF            Defb %11111111,%11111111
F22F  FFFF            Defb %11111111,%11111111
F231  FFFF            Defb %11111111,%11111111
F233  0000            Defb %00000000,%00000000
F235  0000            Defb %00000000,%00000000
F237  0000            Defb %00000000,%00000000
F239  0000            Defb %00000000,%00000000
F23B  0000            Defb %00000000,%00000000
F23D  0000            Defb %00000000,%00000000
F23F  0000            Defb %00000000,%00000000
F241  0000            Defb %00000000,%00000000

                ;GOMME (dessin).
F243  0000            Defb %00000000,%00000000
F245  0000            Defb %00000000,%00000000
F247  0000            Defb %00000000,%00000000
F249  0000            Defb %00000000,%00000000
F24B  0000            Defb %00000000,%00000000
F24D  0000            Defb %00000000,%00000000
F24F  0000            Defb %00000000,%00000000
F251  0000            Defb %00000000,%00000000
F253  8010            Defb %10000000,%00010000
F255  70E0            Defb %01110000,%11100000
F257  4020            Defb %01000000,%00100000
```

```
F259  4020            Defb %01000000,%00100000
F25B  4020            Defb %01000000,%00100000
F25D  4020            Defb %01000000,%00100000
F25F  70E0            Defb %01110000,%11100000
F261  8010            Defb %10000000,%00010000

                ;STYLO (masque).
F263  EE77            Defb %11101110,%01110111
F265  CC33            Defb %11001100,%00110011
F267  8811            Defb %10001000,%00010001
F269  8811            Defb %10001000,%00010001
F26B  8811            Defb %10001000,%00010001
F26D  8811            Defb %10001000,%00010001
F26F  8811            Defb %10001000,%00010001
F271  8811            Defb %10001000,%00010001
F273  8833            Defb %10001000,%00110011
F275  8833            Defb %10001000,%00110011
F277  8833            Defb %10001000,%00110011
F279  8833            Defb %10001000,%00110011
F27B  8833            Defb %10001000,%00110011
F27D  0077            Defb %00000000,%01110111
F27F  0077            Defb %00000000,%01110111
F281  88FF            Defb %10001000,%11111111

                ;STYLO (dessin).
F283  0000            Defb %00000000,%00000000
F285  1080            Defb %00010000,%10000000
F287  20C0            Defb %00100000,%11000000
F289  2040            Defb %00100000,%01000000
F28B  2040            Defb %00100000,%01000000
F28D  2040            Defb %00100000,%01000000
F28F  30C0            Defb %00110000,%11000000
F291  20C0            Defb %00100000,%11000000
F293  2080            Defb %00100000,%10000000
F295  2080            Defb %00100000,%10000000
F297  2080            Defb %00100000,%10000000
F299  2080            Defb %00100000,%10000000
F29B  3080            Defb %00110000,%10000000
F29D  7000            Defb %01110000,%00000000
F29F  7000            Defb %01110000,%00000000
F2A1  0000            Defb %00000000,%00000000
```

Voici un exemple de programme TURBO PASCAL utilisant la routine précédente, le programme est prévu pour CP/M Plus, d'où une adresse d'implantation de la routine en code machine en $EEC0. Pour la version CP/M 2.2, il faudra mettre $97C0 par exemple.

```
PROGRAM TestSprite;

CONST AdrBin=$EEC0;     { Implantation de la routine }
VAR x,y : Integer;       { Coordonnées du curseur graphique }

PROCEDURE Firm;
{
 Routine pour appeler le Firmware. (voir 'Problèmes de compatibilité).
}
Begin
 Inline ($C3/$FC5A)
End;

PROCEDURE ChargeSprite;
{
 Chargement de la routine en langage machine.
 Voir 'Manipulation des fichiers binaires'.
}
VAR Adr : Byte Absolute AdrBin;
    n   : Integer;
    Fic : File;
Begin
 Assign (Fic,'B :SPRITE.BIN');
 Reset (Fic);
 Seek (Fic,1);                    { Saute entte AMSDOS }
 BlockRead (Fic,Adr,99,n);        { Lecture du fichier complet }
 Close (Fic)
End;

PROCEDURE SpriteParam;
{
 Place dans DE/HL les coordonnées standard  (0..639,0..399)  du  curseur
 graphique.
 X :=(X actuel)+(X origine).
```

```
 Y :=(Y actuel)+(Y origine).
}
Begin
 Inline (
  $CD/>Firm/   { Call Firmware }
  $BBCC/     {DW Gra Get Origin ;Coordonnées origine dans DE/HL }
  $E5/         { Push HL        ;Sauve YO }
  $2A/>x/      { Ld   HL,(x)    }
  $19/         { Add  HL,DE     ;HL :=X+XO }
  $EB/         { Ex   DE,HL     ;DE :=X+XO }
  $C1/         { Pop  BC        }
  $2A/>y/      { Ld   HL,(y)    }
  $09          { Add  HL,BC     ;HL :=Y+YO }
 )
End;

PROCEDURE SpriteInit (n : Byte);
{
 Entrée 1.
 Place la forme N la première fois.
}
Begin
 SpriteParam;
 Inline (
  $3A/>n/      { Ld   A,(n)     }
  $CD/>Firm/   { Call Firmware }
  $EEC0      {DW Sprite_1       }
 )
End;

PROCEDURE SpriteSet (n : Byte);
{
 Entrée 2.
 Place la forme N à un nouvel endroit.
}
Begin
 SpriteParam;
 Inline (
  $3A/>n/      { Ld   A,(n)     }
  $CD/>Firm/   { Call Firmware }
  $EEC3      {DW Sprite_2       }
```

```
 )
End;

PROCEDURE SpriteOff;
{
 Entrée 3.
 Efface la forme à l'écran.
}
Begin
 Inline (
  $CD/>Firm/  { Call Firmware }
  $EEC6      {DW Sprite_3       }
 )
End;

Begin
 ChargeSprite;                  { Charge les codes machine }
 Write (^['3B');                { Place en mode 2 }
 Gotoxy (1,1);                  { Remplit l'écran avec des 'a' }
 For x :=1 to 1999 Do Write ('a');

 x :=100;y :=100;                { Place la forme 3 la première fois }
 SpriteInit (3);

 For x :=100 to 600 Do          { Déplace celle-ci }
  Begin
   SpriteSet (3);
   y :=100+(Succ (y) MOD 50)
  End;

 SpriteOff                      { et l'efface à la fin }
End.
```

LIVRES PROFESSIONNELS

Bases de données

Introduction à dBASE II
Alan Simpson
Réf. 0064, 280 pages.

dBASE II Applications
Christophe Stehly
Réf. 416, 248 pages.

Introduction à dBASE III
Alan Simpson
Réf. 0131, 272 pages.

Introduction à dBASE III Plus
Alan Simpson
Réf. 0212, 400 pages.

dBASE III Plus Applications
Alan Simpson
Réf. 0151, 500 pages.

Bibliothèque de procédures dBASE III Plus
Alan Simpson
Réf. 0228, 320 pages.

Dictionnaire dBASE III
Gerhard Renner
Réf. 0238, 352 pages.

Tableurs

Introduction à Lotus 1.2.3
Chris Gilbert et Laurie Williams
Réf. 0106, 272 pages.

Lotus 1.2.3
Programmation des macro-commandes
Goulven Habasque
Réf. 0150, 144 pages.

Lotus 1.2.3 pour l'entreprise
Dominique Hellé et Guy Boussand
Réf. 0147, 256 pages.

Multiplan pour l'entreprise
Dominique Hellé et Guy Boussand
Réf. 0079, 304 pages.

Multiplan sur Macintosh
Goulven Habasque
Réf. 0099, 236 pages.

Traitements de texte

Introduction à MS-WORD
Matthew Holtz
Réf. 0187, 400 pages.

Introduction à WordStar
Arthur Naiman
Réf. 0062, 200 pages.

WordStar Applications
Julie Anne Arca
Réf. 0005, 320 pages.

Introduction à WordStar 2000
Thomas Blackadar et David Kolodney
Réf. 0160, 256 pages.

Introduction à WordPerfect
Susan Baake Kelly
Réf. 0245, 432 pages.

Guide de l'AppleWorks
Pierre-Marc Beaufils
Réf. 0188, 208 pages.

Graphiques sur IBM PC
Stanley R. Trost
Réf. 357, 250 pages.

La Solution RS 232
Joe Campbell
Réf. 0052, 208 pages.

■ Matériels

Macintosh pour la Presse, l'Édition et la Publicité
Bernard Le Dû
Réf. 0173, 160 pages.

IBM PC, Guide de l'utilisateur
Joan Lasselle et Carol Ramsay
Réf. 301, 130 pages.

IBM PC 66 programmes BASIC
Stanley R. Trost
Réf. 359, 174 pages.

■ Amstrad PC

Amstrad PC 1512, Guide de l'utilisateur
Nellie Saumont et Michel Laurent
Réf. 0237, 272 pages.

Amstrad PC 1512, Guide de BASIC II
Jean-Louis Gréco et Michel Laurent
Réf. 0235, 384 pages.

Amstrad PC 1512, DOS Plus
Jean-Louis Gréco et Michel Laurent
Réf. 0236, 256 pages.

LIVRES TECHNIQUES

■ Microprocesseurs

Programmation du 6502
Rodnay Zaks
Réf. 0031, 376 pages.

Applications du 6502
Rodnay Zaks
Réf. 0032, 288 pages.

Programmation du Z80
Rodnay Zaks
Réf. 0058, 618 pages.

Applications du Z80
James W. Coffron
Réf. 0181, 304 pages.

Programmation du 6809
Rodnay Zaks et William Labiak
Réf. 0139, 392 pages.

Programmation du 8086/8088
James W. Coffron
Réf. 0016, 304 pages.

80286 Mise en œuvre et programmation
C. Vieillefond
Réf. 0138, 544 pages.

Mise en œuvre du 68000
C. Vieillefond
Réf. 0133, 352 pages.

Techniques d'interface aux microprocesseurs
Austin Lesea et Rodnay Zaks
Réf. 0039, 456 pages.

Du composant au système
Rodnay Zaks
Réf. 0040, 640 pages.

■ Programmation/Langage

Introduction au BASIC
Pierre Le Beux
Réf. 335, 336 pages.

Le BASIC par la pratique
Jean-Pierre Lamoitier
Réf. 0095, 252 pages.

Fichiers en BASIC
Alan Simpson
Réf. 0102, 256 pages.

Programmes en BASIC
Alan R. Miller
Réf. 259, 318 pages.

Le BASIC pour l'entreprise
Xuan Tung Bui
Réf. 253, 180 pages.

Techniques de programmation en BASIC
Crosmarie, Perron et Philippine
Réf. 0124, 152 pages.

Introduction à Mac BASIC
Pierre Le Beux
Réf. 0140, 420 pages.

Introduction au Pascal
Pierre Le Beux
Réf. 0030, 496 pages.

Le Pascal par la Pratique
Pierre Le Beux et Henri Tavernier
Réf. 0061, 562 pages.

Le Guide du Pascal
Jacques Tiberghien
Réf. 423, 504 pages.

Programmes en Pascal
Alan R. Miller
Réf. 240, 392 pages.

Introduction au Turbo Pascal
Douglas Stivison
Réf. 0180, 260 pages.

Introduction à Mac Pascal
Pierre Le Beux
Réf. 0145, 406 pages.

Introduction à C
Bruce Hunter
Réf. 0092, 300 pages.

Langage C, Structure Application
René Benoit
Réf. 0211, 174 pages.

C par la pratique
Craig Bolon
Réf. 0232, 496 pages.

Gestion des données en C
Robert A. Radcliffe
Réf. 0225, 500 pages.

Programmes en assembleur
sous MS-DOS/PC-DOS
Jean-Paul Pruniaux
Réf. 0241, 352 pages.

Programmation en assembleur du 8086/8088
François Retoreau
Réf. 0093, 616 pages.

MS-DOS/PC-DOS
Techniques de programmation en assembleur
Alan R. Miller
Réf. 0224, 432 pages.

La Rom de l'Apple II
Marcel Cottini
Réf. 0165, 320 pages.

La Rom de l'Apple IIc
Marcel Cottini
Réf. 0182, 272 pages.

Algorithmes
Pierre Beaufils et Wolfram Luther
Réf. 0149, 296 pages.

Le filtrage numérique
Pierre-Marc Beaufils et Michel Rami
Réf. 0192, 316 pages.

Introduction à Ada
Pierre Le Beux
Réf. 360, 376 pages.

MS-DOS, guide du programmeur
Richard Allen King
Réf. 0234, 444 pages.

Apple II, DOS 3.3, guide du programmeur
Marcel Cottini
Réf. 0210, 240 pages.

Guide de ProDOS
Pierre-Marc Beaufils et Wolfram Luther
Réf. 0146, 248 pages.

ProDOS, guide du programmeur
Marcel Cottini
Réf. 0214, 336 pages.

Introduction à Unix
John D. Halamka
Réf. 0098, 240 pages.

■ Système d'exploitation

MS-DOS Approfondi
Jonathan Kamin
Réf. 0227, 400 pages.

POUR UN CATALOGUE COMPLET
DE NOS PUBLICATIONS

FRANCE

6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.

2021 Challenger Drive, 100
Alameda, CA 94501
Tél. : (415) 523-8233
(800) 227-2346 (Outside CA)
Télex : 287639

ALLEMAGNE

Vogelsanger. Weg 111
4000 Düsseldorf 30
Tel. : (211) 61.80.2-0
Telex : 8588163

Paris • San Francisco • Düsseldorf • Londres

Achevé d'imprimer le 28 septembre 1987 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 3e trimestre 1987 - Imprimeur n° 2947

TURBO PASCAL
TECHNIQUES DE PROGRAMMATION

Paris • San Francisco • Düsseldorf • Londres

Cet ouvrage présente, en première partie, les outils de base nécessaires à une programmation avancée en Turbo Pascal, accompagnés de nombreux exemples. Il explique ensuite les différences et les problèmes de compatibilité entre les deux systèmes CPM de l'Amstrad.

En seconde partie, le livre propose plusieurs programmes dont certains sont uniquement en Assembleur, et d'autres en Turbo Pascal standard. Ces programmes comprennent, entre autres, la saisie contrôlée par clavier, l'utilisation de l'éditeur graphique, l'utilisation de sprites, la sauvegarde et le rappel d'écran.

L'AUTEUR

Après une formation de physicien-chimiste, BERTRAND NELSON se tourne vers l'informatique en 1984. Il devient Professeur Assistant en micro-informatique. Il a aussi écrit un logiciel de comptabilité générale en Turbo Pascal pour Amstrad.

0300 0987 148 F

9 782736 103002

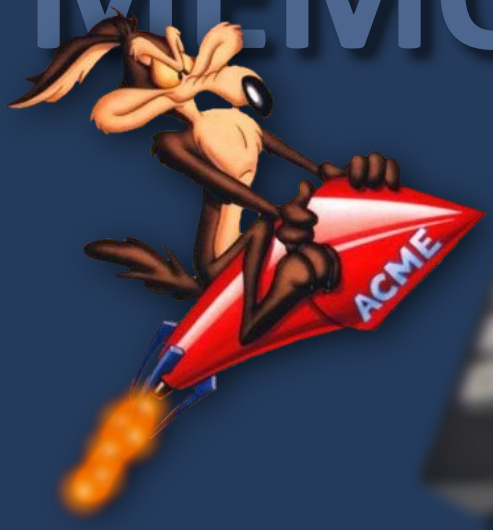
SYBEX **6-8, impasse du Curé, 75018 Paris**

TURBO PASCAL
TECHNIQUES DE PROGRAMMATION

Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/