AMSTRAD

MANUEL D'UTILISATION DES EXTENSIONS de 64 Ko et 256 Ko

EDITION N° 1

ATTENTION

VOTRE ORDINATEUR DOIT ÊTRE ÉTEINT lorsque vous branchez l'interface sur la prise d'extension. Vous risqueriez, en ne respectant pas cette consigne, d'endommager le module de RAM ou l'ordinateur de façon irréversible.

TABLE DES MATIÈRES

1. MISE EN PLACE DU MODULE DE MÉMOIRE VIVE (RAM)

Assurez-vous que votre ordinateur Amstrad n'est pas sous tension. Enfichez le module de RAM dans la prise située à l'arrière de l'ordinateur, qui est dénommé « Floppy Disc » sur le CPC 464 et « Expansion » sur le CPC 664 et le CPC 6128. D'autres extensions ou périphériques, tels que l'interface pour unité de disquette Amstrad, destinée au CPC 464, le crayon optique et le synthétiseur de parole de DK TRONICS, ou les extensions de mémoire morte (ROM) peuvent être branchés sur le connecteur d'extension situé à l'arrière du module de RAM. A présent, mettez l'ordinateur sous tension.

La mise sous tension de l'ordinateur devrait s'effectuer de façon normale. Si tel n'était pas le cas, vérifiez que les prises sont correctement branchées. Remarquez que tous les produits DK TRONICS ont un connecteur avec rainure pour éviter les problèmes d'alignement lors de la connexion. (D'autres interfaces peuvent ne pas être munies d'une rainure ; c'est le cas, par exemple, de l'interface pour unité de disquette Amstrad.) Les problèmes de connexion proviendront donc, en général, du branchement des extensions à l'arrière du module de RAM. Dans ce cas, rebranchez les interfaces AVANT d'insérer le module de RAM dans l'ordinateur : il vous sera plus facile de voir comment s'opère l'insertion des broches.

Si l'ordinateur ne se met pas sous tension ou bien ne fonctionne pas normalement (motifs divers apparaissant sur l'écran), le moniteur coupera l'alimentation de l'ordinateur. Eteindre le moniteur couleur et recommencez l'opération de connexion ainsi qu'il est dit ci-dessus. Si votre moniteur est monochrome, attendre plusieurs secondes avant de remettre l'ordinateur sous tension.

Il est extrêmement rare que l'ordinateur ne se mette pas sous tension de façon normale, lorsque la seule extension utilisée est le module de RAM. Mais si tel était le cas, cela signifierait sans doute que le module de RAM est défectueux. Il vous faudrait alors rapporter le module à votre distributeur.

2. UTILISATION DE LA RAM D'EXTENSION

Il existe deux façons de se servir de la mémoire vive d'extension. Une cassette comportant des commandes supplémentaires de BASIC est fournie avec ce module de RAM. Il suffit d'utiliser ces commandes supplémentaires dans un programme BASIC pour lire ou écrire dans la RAM d'extension. On peut également accéder à la

RAM d'extension avec un programme en BASIC ou en code machine, grâce à la commande OUT. Les programmeurs expérimentés seront capables d'utiliser la RAM de l'une ou l'autre façon et d'écrire des programmes en conséquence. Les logiciels que l'on trouve dans le commerce procéderont sans aucun doute de même.

La deuxième méthode d'utilisation de la RAM d'extension est expliquée en détail au chapitre 10. Nous allons examiner dans les chapitres qui suivent la première de ces deux méthodes.

L'installation du module de RAM ayant été effectuée comme il est indiqué au chapitre 1), chargez en mémoire le logiciel RSX qui se trouve sur la cassette fournie avec le module de RAM :

a) Si votre ordinateur est muni d'une unité de disquettes, tapez « \| TAPE » et appuyez sur la touche ENTER.
(Rappelez-vous que le signe « \| » se trouve sur la touche où figure le caracère *â*).

b) Tapez « RUN » et appuyez sur la touche ENTER.

c) La séquence de chargement est décrite en détail dans le manuel d'utilisation de votre ordinateur.

d) Lorsque le chargement du programme est terminé, l'ordinateur vous demande de lui indiquer l'adresse où le ranger. Appuyez sur la touche ENTER : le programme sera alors rangé à l'adresse disponible la plus élevée.
(Se reporter au chapitre 11.)

e) L'ordinateur testera le bon fonctionnement de la RAM et affichera l'espace mémoire (en nombre d'octets) dont vous disposez. La mémoire de l'ordinateur est alors prête à recevoir vos programmes.

La cassette contient les mêmes programmes sur ses deux faces, de sorte que si le chargement des programmes ne s'opérait pas à partir de l'une des faces, il suffirait d'utiliser l'autre face de la cassette.

La cassette contient, outre le logiciel RSX, les programmes indiqués dans la suite de ce manuel. Vous pouvez charger ces derniers en mémoire si vous ne voulez pas les taper sur le clavier.

REMARQUE. — *â* : ce signe signifie touche A commercial du clavier chaque fois qu'il se présente dans le texte.

NOTE. — Les programmes figurant dans le présent manuel sont identiques à ceux contenus dans la cassette, mais ils ont été francisés. Les programmes de la cassette sont entièrement en anglais.

3. TEST DE LA RAM D'EXTENSION

Une fois qu'il est chargé, le logiciel RSX procède à un test complet de la RAM d'extension. Si la RAM était défectueuse, un premier message vous en informerait, puis un deuxième message vous indiquerait la nature du problème.

Au cas, fort improbable, où un fonctionnement défectueux serait détecté, prenez note du message de diagnostic et rapportez le module de mémoire vive à votre distributeur.

4. COMMANDES DE BASIC ÉTENDU

Les RSX contenus sur la cassette vous permettent de disposer de douze nouvelles commandes. Certaines devront être accompagnées de paramètres, d'autres non. Le format et le nombre des paramètres variera d'une commande à l'autre. Nous allons commencer par vous exposer l'utilisation la plus simple de chacune de ces commandes. Par la suite, nous indiquerons comment leur adjoindre d'autres paramètres afin de les rendre plus flexibles et d'économiser l'espace mémoire. Certains utilisateurs inexpérimentés souhaiteront ne connaître que l'essentiel, dans un premier temps. Aussi, avons-nous repéré par un astérisque (*) les chapitres ou paragraphes qui traitent de l'utilisation approfondie des commandes et qui peuvent être ignorés lors d'une première lecture de ce manuel.

Toutes les nouvelles commandes sont précédées d'une barre verticale « | ». Ce caractère se trouve sur la touche « *â* » située à droite de la touche « p ».

Vous avez probablement remarqué, au cours du test de la RAM, que l'ordinateur affichait le numéro du bloc de mémoire qu'il était en train de tester. Chaque bloc de mémoire est constitué de 16 Koctets. Le module de RAM de 64 Ko comporte donc 4 blocs, et celui de 256 Ko, 16 blocs. Pour accéder à une zone donnée de la mémoire d'extension, il faut donc indiquer un numéro de bloc et, éventuellement, une adresse au sein du bloc.

Par exemple, tapez sur le clavier :

| SAVES, 1 et appuyez sur la touche ENTER

L'ordinateur affichera le message READY. En fait, vous venez de stocker ce qui était sur l'écran dans le bloc 1 de la RAM d'extension.

Maintenant, effacez la page-écran (le contenu de l'écran) en tapant CLS sur le clavier. Pour rappelez la page-écran, tapez :

| LOADS,1 et appuyez sur la touche ENTER

Le nombre de pages-écran qui peut être sauvegardé dépend de l'espace mémoire dont vous disposez. Vous pouvez sauvegarder 4 pages-écran avec une RAM de 64 Ko, et 16 pages-écran avec une RAM de 256 Ko.

Les pages-écran peuvent être créées par un autre programme ou dessinées à l'aide d'un crayon lumineux. Stockez-les sur cassette ou sur disquette, puis chargez-les dans la RAM d'extension pour les utiliser lors de l'exécution d'un programme. Les pages-écran dont la création au sein d'un programme requiert beaucoup de temps, telles que les labyrinthes, peuvent être créées et stockées dans la RAM d'extension, puis appelées pour utilisation immédiate, chaque fois que nécessaire.

Voici la forme-type des commandes :

| SAVES, [n° de bloc]
stocker la page-écran dans le bloc de mémoire n° x

| LOADS, [n° de bloc]
lire la page-écran située dans le bloc de mémoire n° x

5. FENÊTRES ET MENUS APPELABLES SUR L'ÉCRAN

L'une des raisons pour lesquelles la flexibilité d'utilisation des fenêtres produites par les ordinateurs Amstrad est moins grande que celle des fenêtres créées sur des ordinateurs professionnels plus gros est que le contenu d'une fenêtre s'efface lorsqu'il est recouvert par une autre fenêtre.

Vous disposez de deux commandes qui vous permettent de sauvegarder des fenêtres dans la RAM d'extension ou bien de charger des fenêtres à partir de la RAM d'extension. Vous pourrez ainsi réaliser de vrais menus appelables sur l'écran, qui couvriront le texte mais ne l'effaceront pas.

Exemple 1 :

```
10 MODE 1
20 FOR i=0.05 TO 1 STEP 0.05 : REM Dessiner grille sur l'écran
30 MOVE 640*i,0 : DRAW 640*i,400
40 MOVE 0,400*i : DRAW 640,400*i
50 NEXT i
60 WHILE INKEY$="" : WEND : REM Attendre qu'une touche soit
   enfoncée
70 WINDOW #1, INT (RND(1)*19+1), INT (RND(0)*19+INT
   (RND(1)*5+17)), INT (RND(1)*14+1),
   INT (RND(0)*14+INT (RND(1)*10+5))
```

```
80 PEN #1,2 : PAPER # 1,3
90 |SAVEW,1,1 : REM Sauvegarder le contenu de la fenêtre dans
   la RAM
100 CLS #1 : REM Effacer fenêtre
110 WHILE INKEY$="" : REM Attendre qu'une deuxième touche
   soit enfoncée
120 PRINT #1, « Ceci est une fenêtre »
130 WEND
140 |LOADW,1, : REM Rappeler le contenu de la fenêtre
150 GOTO 60
```

Le programme ci-dessus utilise deux nouvelles commandes : |LOADW et |SAVEW. Vous savez probablement qu'il vous est possible de définir jusqu'à 8 fenêtres (0 à 7). Le premier paramètre correspondant au numéro de la fenêtre et le second, au numéro du bloc de mémoire.

|SAVEW, [n° de fenêtre], [n° de bloc]
stocker la fenêtre n° x dans le bloc de mémoire n° x

|LOADW, [n° de fenêtre], [n° de bloc]
charger la fenêtre n° x située dans le bloc de mémoire n° x

Pour plus de détails, se reporter aux chapitres du manuel de l'utilisateur traitant des fenêtres.

5a. **Quelques précisions à propos des fenêtres** (*)

Une fenêtre, quelles que soient ses dimensions, et même si elle occupe tout l'écran, pourra être stockée dans un seul des blocs de la RAM d'extension. Si votre fenêtre occupe tout l'écran, ou bien si ses dimensions varient comme dans l'exemple ci-dessus, elle occupera un bloc de mémoire. Mais si elle est de 10 × 10 en MODE 1, l'espace mémoire requis pour la stocker sera inférieur à 16 Ko ; il sera, en fait, égal à 1 600 octets (voir le paragraphe suivant pour calculer l'encombrement mémoire d'une fenêtre que vous avez définie). Ainsi, si vous utilisez un bloc entier, vous perdrez environ 14 Ko de mémoire.

Voici comment résoudre le problème. Les commandes RSX relatives aux fenêtres acceptent un paramètre supplémentaire qui vous permet de spécifier l'adresse mémoire où vous voulez stocker la fenêtre :

|SAVEW, [n° de fenêtre], [n° de bloc], [adresse . bloc]
|LOADW, [n° de fenêtre], [n° de bloc], [adresse . bloc]

Les adresses au sein d'un bloc vont de 0 à 16383. Pour obtenir la gamme des adresses auxquelles peuvent être stockées une fenêtre, il faut ôter de l'adresse maximale, l'encombrement mémoire (en nombre d'octets) de la fenêtre. Ainsi, dans notre exemple, la fenêtre

de 1 600 octets, peut être stockée à partir de l'une quelconque des adresses comprises dans la gamme 0 à 14783. Si vous stockez la fenêtre au bas de la RAM, à l'adresse 0, le bloc de mémoire pourra accueillir d'autres fenêtres ou tableaux de données de l'adresse 1600 à l'adresse 16383.

COMMENT CALCULER L'ENCOMBREMENT MÉMOIRE D'UNE FENÊTRE

Si vous voulez stocker plus d'une fenêtre par bloc de mémoire, il vous faut calculer la taille de la fenêtre, c'est-à-dire son encombrement mémoire. Si les dimensions de la fenêtre varient entre deux valeurs, utiliser celle qui est la plus élevée. Voici comment calculer la taille d'une fenêtre pour chacun des modes d'affichage :

Pour tous les modes :

X1 représente la coordonnée x la plus à gauche
X2 représente la coordonnée x la plus à droite
Y1 représente la coordonnée y la plus haute
Y2 représente la coordonnée y la plus basse

MODE 0 TAILLE = (X2 — X1 + 1)★4★(Y2 — Y1 + 1)★8

MODE 1 TAILLE = (X2 — X1 + 1)★2★(Y2 — Y1 + 1)★8

MODE 2 TAILLE = (X2 — X1 + 1)★(Y2 — Y1 + 1)★8

L'ordinateur affichera un message d'erreur si la fenêtre est trop grande pour l'espace mémoire que vous lui avez alloué. Si vous avez mal calculé la taille des fenêtres, celles-ci pourront se chevaucher dans le bloc de mémoire, et des phénomènes curieux se produiront.

Exemple 2 :

```
 10 PEN 1 : PAPER 0 : MODE 1
 20 taille = 14★2★10★8
 30 LOCATE 1,13 : PRINT " 'n' pour nouvelle fenêtre 'd' pour effacer
    fenêtre
 40 WINDOW 1,14,1,10 : PAPER 3 : CLS
 50 adresse = 0 : canal = 0
 60 PRINT ♯ canal, « Fenêtre » : canal
 70 touchact$=LOWERS$ (INKEY$)
 80 IF touchact$="n" THEN GOSUB 110
 90 IF touchact$="d" THEN GOSUB 190
100 GOTO 60
110 IF canal= 7 THEN RETURN
120 canal = canal + 1
```

```
130 WINDOW #canal,1+canal*3,14+canal*3,1+canal*2,10
    +canal*2
140 |SAVEW,canal,1,adresse
150 adresse = adresse + taille
160 PEN #canal,0 : PAPER #canal, (canal AND 1)+1
170 CLS #canal
180 RETURN
190 IF canal = 0 THEN RETURN
200 adresse = adresse — taille
210 |LOADW,canal,1,adresse
220 canal = canal — 1
230 RETURN
```

Le programme ci-dessus n'utilise qu'un bloc de la RAM, mais définit 8 fenêtres. La variable « canal » ("level" en anglais) indique le numéro de canal (ou si l'on préfère, le numéro de fenêtre) ; la variable « adresse » (bankadress en anglais) indique la prochaine adresse libre au sein du bloc de mémoire.

6. TABLEAUX, VARIABLES ET CHAINES (*)

Deux commandes d'usage général permettent de transférer des données d'un programme vers la RAM d'extension et de la RAM d'extension dans un programme.

Il s'agit des commandes :

|SAVED, [n° de bloc], [point départ], [longueur], [adresse . bloc]
|LOADD, [n° de bloc], [point départ], [longueur], [adresse . bloc]

Le premier paramètre indique le bloc de mémoire que vous voulez utiliser. Le point départ est une adresse dans la mémoire d'origine où se trouvent des données. La quantité de donnée est identifiée par le paramètre longueur. Eventuellement, vous pouvez indiquer une adresse au sein du bloc de mémoire si vous voulez stocker plusieurs types de données dans la RAM.

Il est possible de sauvegarder ou d'appeler toutes sortes de données à l'aide de ces deux commandes, mais nous examinerons tout d'abord comment sauvegarder des tableaux numériques simples, car c'est l'opération la plus aisée à comprendre.

Imaginons, par exemple, que vous vouliez écrire un programme de gestion des stocks, votre stock étant composé de 60 articles différents. Vous pouvez avoir un tableau alpha indiquant le nom ou la référence de chaque article et un tableau numérique indiquant le nombre de pièces dont vous disposez pour chaque nom (ou référence) d'article.

Les noms des articles occuperaient environ 1 Ko et les chiffres 300 octets. Mais que se passera-t-il si vous voulez enregistrer chaque semaine l'état du stock et conserver les données hebdomadaires de l'année précédente, voire les données hebdomadaires des cinq dernières années ? Vous auriez besoin pour stocker ces chiffres de 15 Ko dans le premier cas, et de 75 Ko dans le deuxième cas.

On peut aisément stocker les données relatives à une année sur disquette, ou sur cassette, et les lire chaque fois que l'on en a besoin poru un calcul. Mais, vous conviendrez avec moi que l'opération de lecture qui devrait avoir lieu chaque fois que l'on veut étudier la distribution d'un article au cours d'une période donnée prendrait beaucoup de temps.

De toute évidence, il serait plus simple de stocker toutes les données dans la RAM d'extension, car l'accès aux données serait alors immédiat.

Au lieu de définir un tableau de dimensions « stock(60,52) » qui occuperait un espace mémoire de 15 Ko, lequel pourrait être utilisé pour stocker un programme, définissez un tableau « stock(60 ». Lisez les données stockées sur la disquette, semaine par semaine, et stockez les données de chaque semaine dans le bloc mémoire de la RAM. Pour être à même de faire cela, il vous faut savoir deux choses : 1° l'emplacement du tableau en mémoire et 2° l'encombrement mémoire du tableau (cf. le nombre d'octets nécessaire pour le stocker).

EMPLACEMENT DU TABLEAU EN MÉMOIRE

Pour connaître l'adresse mémoire d'une variable quelconque, il suffit de placer le signe « *â* » devant la variable. Par exemple, dimensionnez le tableau ci-dessous :

```
DIM stock(60)
```

Maintenant, tapez :

```
PRINT â stock(0)
```

L'ordinateur répondra en vous donnant l'adresse du premier élément du tableau stocké en mémoire. Maintenant, tapez sur le clavier :

```
PRINT â stock(1)
```

L'adresse mémoire indiquée par l'ordinateur sera plus élevée de 5 points. Il s'agit de l'adresse du deuxième élément du tableau.

Vous pouvez placer le préfixe « *â* » devant n'importe quelle variable. Pour connaître l'adresse du premier élément d'un tableau unidimensionnel, vous taperez : « *â* stock(0) », et pour connaître l'adresse du

premier élément d'un tableau bidimensionnel, ou tridimensionnel, vous taperez : « *â* stock(0,0) », ou « *â* stock(0,0,0) », etc.

COMMENT CALCULER L'ENCOMBREMENT MÉMOIRE D'UN TABLEAU

L'encombrement mémoire d'un tableau = encombrement mémoire de chaque élément du tableau × le nombre total d'éléments du tableau

L'encombrement mémoire de chaque élément du tableau (eu nombre d'octets) dépend du type d'éléments que contient le tableau. Pour les tableaux de nombres réels, il faut compter 5 octets par élément, et pour les tableaux de nombres entiers, 2 octets par éléments. La longueur des éléments des tableaux alpha peut être variable : nous traiterons de cette question au paragraphe 6b).

Il faut ensuite calculer le nombre total d'éléments du tableau, en tenant compte du nombre de ses dimensions. Rappelez-vous que le premier élément d'un tableau est 0. Cela signifie qu'un tableau qui indique « stock(60) » comporte, en fait, 61 éléments. Vous pouvez choisir d'utiliser ou de ne pas utiliser l'élément 0, mais si vous oubliez son existence, des « bogues », apparemment inexplicables, se produiront dans votre programme.

Si le tableau a plusieurs dimensions, il faut multiplier entre eux le nombre d'éléments de chacune des dimensions pour connaître le nombre total d'éléments du tableau.

Ainsi :

« stock(60) » comporte 61 éléments
« stock(60,52) » comporte 61★53 éléments = 3 233 éléments
« stock%(10,5,12) » comporte 11★6★13 éléments = 858 éléments.

A présent, multiplions l'encombrement mémoire de chaque élément du tableau par le nombre total d'éléments du tableau. Nous obtenons :

« stock(60) » occupe 5★61 = 305 octets
« stock(60,52) » occupe 5★3233 = 16165 octets
« stock%(10,5,12) » occupe 2★858 = 1716 octets

Le tableau que nous utilisons occupe 305 octets et commence à l'adresse *â* stock(0).

Dans un bloc de la RAM, nous pouvons stocker 53 fois 305 octets. La première adresse est 0, puis l'incrémentation se fait par pas de 305 octets :

0 305 610 915 1 220 1 525, etc.

Nous stockerons la semaine 1 à l'adresse 305, la semaine 2, à l'adresse 610 et ainsi de suite pour les 52 semaines.

Le programme ci-dessous permet de stocker sur disquette ou cassette des données. Constituez un fichier de données de test et conservez-le : vous l'utiliserez pour tester votre programme :

```
10 OPENOUT « donstock »
20 FOR semaine = 1 TO 52
30 FOR element = 1 TO 60
40 PRINT ♯9, INT (RND(1)★3000★100)
50 NEXT element
60 NEXT semaine
70 CLOSEOUT
80 END
```

Maintenant, tapez « NEW » et introduisez le programme suivant dans l'ordinateur :

```
10 DIM stock(60)
20 INPUT « lire fichier (o/n) » ; rep$
30 IF LOWER$ (rep$) = « o » or LOWER$ (rep$) = « oui »
   GOSUB 1000
40 REM reste du programme...
1000 REM sous-programme lisant les données sur disquette
1010 OPENIN « donstock »
1020 FOR semaine = 1 TO 52
1030 FOR element = 1 TO 60
1040 INPUT ♯9, stock (element)
1050 NEXT element
1060 | SAVED, 4, â stock(0), 61★5, semaine★305
1070 NEXT semaine
1080 CLOSE IN
1090 RETURN
```

Le programme ci-dessus pourrait être utilisé pour lire le fichier sur disquette ou cassette. Une fois que le fichier est dans la RAM, il y restera et vous pourrez l'utiliser tant que vous n'éteignez pas votre ordinateur ou que vous n'écrisez pas les données de stock avec d'autres données. Cela signifie qu'il suffit de lire les données

stockées sur la disquette une seule fois et que le programme peut être exécuté autant de fois que vous voulez sans que les données soient perdues. Cela vous permet également d'exécuter des programmes différents qui utilisent le même ensemble de données. Lorsque les données sont en mémoire, vous pouvez accéder aux données relatives à chaque semaine en utilisant l'instruction |LOADD. Ajoutez les lignes de programme ci-dessous pour dessiner un diagramme à barres (diagramme de Gantt) :

```
100 MODE 2
110 LOCATE 1,1
120 INPUT « quel élément analyser » ; elemntno
130 IF elemntno < 1 OR elemntno > 60 THEN 120
140 CLS : LOCATE 30,1
150 PRINT « Diagramme à barres pour élément » ; elemntno
160 LOCATE 10,25
170 PRINT « Jan  Fev  Mar  Avr  Mai  Jun  Jul  Aot
    Sep  Oct  Nov  Dec » : REM 3 espaces entre chaque
180 FOR loop = 0 TO 4
190 LOCATE 1,24—loop★5
200 PRINT STR$ (loop) ; "000"
210 NEXT loop
220 MASK 255 : MOVE 60,368 : DRAW 60,0 : DRAW 61,0 : DRAW
    61,368 : MOVE 640,24 : DRAW 48,24
230 FOR loop 1 TO 4
240 MOVE 48, loop★80 + 24 : DRAW 60, loop★ 80 + 24
250 NEXT loop
260 FOR semaine = 1 TO 52
    IF semaine/2 = semaine\2 THEN MASK 170 :
    ELSE MASK 255
280 |LOADD,4,â stock(0), 61★5, semaine★305
290 ycoord = (stock(elemntno)/4000★320) AND 4092
300 FOR xcoord = 1 TO 11
310 MOVE 49 + xcoord + semaine★11,ycoord + 26 :
    DRAW 49 + xcoord + semaine★11,26
320 NEXT xcoord
330 NEXT semaine
340 GOTO 110
```

6a. **Quelques précisions sur les tableaux, variables et chaines**

Si vous avez un programme qui utilise toute la mémoire de l'ordinateur parce qu'il comporte un grand tableau, vous pouvez utiliser un bloc mémoire de la RAM d'extension pour stocker les données sans qu'il vous soit nécessaire de dimensionner le tableau.

Par exemple, si vous avez un tableau à deux dimensions « ventes% (365,30) », indiquant pour certains types d'articles la quantité vendue chaque jour des 365 jours de l'année, l'encombrement mémoire du tableau sera supérieur à 22 Ko, bien que vous utilisiez des nombres entiers.

Au lieu de stocker le tableau dans la mémoire BASIC, vous pouvez le stocker dans la RAM d'extension et utiliser deux sous-programmes : l'un vous permettant d'y lire une valeur, l'autre vous permettant d'y stocker une valeur.

```
10000 REM lire ds bloc RAM « stock% » en fonction de l' « année »
      et du « type »
10010 p = (année★31 + type)★2
10020 bloc = 1 : IF p >=16000 THEN p = p-16000 : bloc = 2
10030 !LOADD, bloc, â stock%, 2, p
10040 RETURN

11000 REM copier « stock% » ds bloc RAM en fonction de
      l' « année » et du « type »
11010 p = (année★31 + type)★2
11020 bloc = 1: IF p >=16000 THEN p = p-16000 : bloc = 2
11030 !SAVED, bloc, â stock% ; 2, p
11040 RETURN
```

Deux blocs de la RAM d'extension, le bloc 1 et le bloc 2, sont utilisés et les variables « année » et « type » permettent de référencer l'élément requis. Les instructions 10030 et 11030 ne lisent ou ne stockent dans la RAM d'extension que 2 octets, car nous utilisons des nombres entiers. Le « ★2 » des lignes de programmes 10010 et 11010 reflète le fait qu'un nombre entier occupe 2 octets. Si le programme utilisait des nombres réels, il faudrait 5 octets. Les instructions 10020 et 11020 permettent de calculer si l'élément doit être lu, ou stocké, dans le bloc 1 ou le bloc 2.

Si le tableau doit contenir des données provenant d'une cassette ou d'une disquette, il n'est pas nécessaire de procéder à une remise à zéro. Il suffit de sauvegarder, au début du programme, un écran vierge dans chaque bloc pour que tous les éléments soient initialisés :

```
10 MODE 1 : PAPER 0 : CLS
20 !SAVES,1
30 !SAVES,2
```

6b. **Le stockage des chaînes**

La principale difficulté que l'on rencontre lorsque l'on veut stocker des chaînes est que leur longueur peut varier. Par ailleurs, il est

possible de les stocker n'importe où dans la mémoire, y compris dans un programme BASIC. Nous vous indiquons ci-dessous une méthode de stockage des tableaux alpha. Il vous est, cependant, possible d'en trouver une plus simple si vous savez exactement ce que vous voulez faire.

Imaginons que vous vouliez stocker 500 noms de 20 caractères chacun. Le bloc de mémoire sera divisé en zones de 21 octets chacune, pour permettre un accès direct à chacun des noms. Chaque zone de 21 octets comportera une chaîne, et un octet indiquant le nombre de caractères de la chaîne. Les 500 noms occuperont donc un espace mémoire légèrement supérieur à 10 Ko. Si nous utilisons la variable « nom » pour indiquer la chaîne que nous voulons, nous pouvons alors écire deux sous-programmes : le premier pour transférer une chaîne du bloc 1 dans « nom$ » et le deuxième pour stocker le contenu de « nom$ » dans le bloc 1 de la RAM d'extension.

```
20000 REM affecte la chaîne « nom » numéro X à « name$ »
20010 b$ = ″                              ″ : REM 21 espaces
20020 | LOADD, 1, PEEK(â b$=1) + PEEK(â b$+2)★256, 21,
      nom★21
20030 nom$ = MID$ (b$, 2, ASC(b$) ) : RETURN

21000 REM stocke le contenu de « noms » ds le bloc 1 en tant que
      « nom » numéro X
21010 b$ = ″                              ″ : REM 21 espaces
21020 MID$ (b$,1,21) = CHR$ ( LEN(nom$) ) = nom$
21030 | SAVED 1, PEEK ( â b$+1 ) + PEEK ( â b$+2 )★256, 21,
      nom★21
21040 RETURN
```

Une chaîne fictive b$ est utilisée pour former l'élément avant qu'il ne soit stocké dans la RAM. Le premier caractère indique la longueur de « nom$ ». Les 20 autres caractères se trouvent là où se trouve le contenu de « nom$ ». Puis les 21 caractères sont copiés dans le bloc de la RAM d'extension. Quand la chaîne est rappelée, les caractères sont recopiés et « nom$ » est mis à la bonne longueur grâce au premier caractère.

Le stockage des chaînes serait très simple si tous les mots avaient la même longueur : il n'y aurait alors aucune perte de l'espace mémoire. Prenons l'exemple d'un jeu du pendu dans lequel on utiliserait des mots de cinq, six ou sept lettres. Un bloc de mémoire pourrait être utilisé pour chaque longueur de mots. Un programme de chargement positionnerait les données dans la RAM, et un autre

programme, qui serait chaîné au premier, pourrait utiliser 36 Ko de RAM pour le programme.

Un tableau numérique pourrait également être stocké dans le bloc RAM pour indicer les premières lettres et ainsi accélérer la vitesse d'accès à un mot donné.

7. ANIMATION ET IMAGES (*)

Nous avons examiné aux chapitres 4 et 5 comment lire ou stocker les pages-écran et les fenêtres dans la RAM d'extension. Un effet d'animation est obtenu lorsque les images se succèdent à l'écran avec une rapidité suffisante pour donner une impression de mouvement. Grâce aux RAM d'extension de 64 Ko et de 256 Ko, des pages-écran complètes peuvent être rangées en mémoire et appelées sur l'écran de façon à produire une animation.

Vous avez pu remarquer, en pratiquant les commandes du chapitre 4, que lorsqu'une page-écran est appelée sur l'écran, vous voyez les lignes s'afficher les unes après les autres. Pour vous en convaincre, entrez le programme suivant dans votre ordinateur :

```
 10 MODE 1
 20 BORDER 0
 30 FOR col = 0 TO 3
 40 INK col,0
 50 NEXT col
 60 FOR col = 0 TO 3
 70 PAPER col : CLS
 80 |SAVES,col+1
 90 NEXT col
100 INK 0,1 : INK1,6 : INK 2,21 : INK 3,13
110 PEN 1 : PAPER 0
120 WHILE INKEY$=""
130 FOR écran = 1 TO 4
140 |LOADS, écran
150 NEXT écran
160 WEND
170 END
```

Le programme stocke 4 pages-écran colorées dans la RAM, puis les charge les unes à la suite des autres. Malheureusement, aucun effet d'animation n'est créé.

Pour obtenir un effet d'animation, il faut que l'ordinateur créée la page-écran, puis l'affiche instantanément.

Vous disposez de trois nouvelles commandes qui vous permettront de réaliser cela. Il s'agit des commandes :

| LOW | HIGH | & | SWAP

Pour pouvoir comprendre l'utilisation de ces commandes, il est nécessaire de savoir utiliser la mémoire écran. La page-écran normale est stockée en mémoire à partir de l'adresse 49152. L'Amstrad est, cependant, capable de repérer une page-écran quel que soit le bloc mémoire de 16 K où elle se trouve. Il est difficile d'utiliser le premier bloc commençant à l'adresse 0) et le troisième bloc (commençant à l'adresse 32768), car l'ordinateur les utilise comme parties de l'interpréteur BASIC. Le bloc de mémoire commençant à l'adresse 16384 peut être utilisé si HIMEM (=l'adresse de l'octet le plus haut utilisé par la mémoire BASIC) est abaissé en dessous de 16384. En supposant que cette opération soit effectuée, nous appellerons la page-écran stockée dans la mémoire écran d'origine « page-écran haute » (high screen), et la page-écran stockée à partir de l'adresse 16384 « page-écran basse » (low screen).

Utilisez les commandes suivantes :

| LOW pour obtenir l'affichage de la page-écran « basse » ;
| HIGH pour réobtenir l'affichage de la page-écran « haute » ;
| SWAP pour changer de page-écran affichée (page-écran haute à page-écran basse ou inversement)

Chaque fois que la commande | SWAP est émise, l'ordinateur fait apparaître tout texte ou tout graphique ultérieurs sur la page-écran (haute ou basse) sélectionnée.

Pour se servir de cette possibilité de changer instantanément d'affichage, on peut ajouter un paramètre aux commandes relatives aux pages-écran et fenêtres, qui indique à l'ordinateur de charger les données dans la mémoire de la page-écran qui n'est pas affichée ou bien de sauvegarder, dans la RAM d'extension, les données de la page-écran qui n'est pas affichée.

Vous pouvez donc écrire :

| SAVES, [n° de bloc], [sélection page-écran]
| LOADS, [n° de bloc], [sélection page-écran]
| SAVEW, [n° de fenêtre], [n° de bloc], [adresse . bloc], [sélection page-écran]
| LOADW, [n° de fenêtre], [n° de bloc], [adresse . bloc], [sélection page-écran]

Si la valeur de sélection page-écran est zéro, par défaut, la commande agira sur la page-écran qui est affichée. Si la valeur est un,

l'ordinateur chargera les données dans la mémoire de la page-écran qui n'est pas affichée ou bien sauvegardera les données de ladite page-écran dans la RAM d'extension. Lorsque cette opération est effectuée, l'ordinateur peut afficher alternativement les pages-écran et l'on obtient ainsi un changement instantané d'affichage.

Ajouter au programme de la page précédente les lignes suivantes :

```
  5 MEMORY 16383 : |HIGH
135 IF écran\2 = écran/2 THEN t = TIME : WHILE TIME<t+20 :
    WEND
140 |LOADS, écran, 1 : |SWAP
```

Maintenant que l'ordinateur peut créer la page-écran pendant qu'une autre page est affichée, la page-écran colorée donne l'impression de changer instantanément.

En raison du fait que le bloc mémoire se déplace dans l'espace-adresse commençant à 16 Ko, le transfert d'une page-écran vers la mémoire d'écran « basse » prend plus de temps qu'un transfert de page-écran vers la mémoire d'écran « haute ». Aussi, la ligne de programme 135 est-elle là pour retarder le chargement dans la mémoire d'écran « haute ». On obtient alors une durée d'affichage des pages-écran égale. Essayez d'ôter la ligne de programme 135 pour voir la différence.

Si un délai plus long était introduit entre les lignes 140 et 150, vous obtiendrez une succession d'images. Vous pourriez aussi sélectionner l'affichage des pages-écran par pression d'une touche.

A plus petite échelle, on peut définir une fenêtre et afficher des graphiques rapidement sans qu'il soit nécessaire de recourir à la permutation (swap) des pages-écran affichées.

Remarquez qu'il est est possible de sauvegarder les pages-écran et les fenêtres qui ne sont pas affichées, en choisissant la valeur « un » pour le paramètre de sélection page-écran. Par exemple, si vous voulez charger une suite de pages-écran stockées sur cassette ou disquette, chargez-les dans la mémoire d'écran « basse » (16 Ko).

Il n'est pas nécessaire d'effacer les messages générés par le système de cassettes, car la page-écran basse n'en sera pas affectée.

```
10 LOAD « ecran1 » 16384 : |SAVES, 3, 1
```

La ligne de programme ci-dessus chargera une page-écran dans la mémoire d'écran basse, puis la sauvegardera dans le bloc n° 3 de la RAM d'extension. La page-écran visualisée par l'utilisateur pendant ce temps pourra être différente.

8. PROGRAMMATION AVANCÉE (*)

Ce chapitre traite de l'utilisation d'une nouvelle commande et apporte quelques précisions qui vous seront utiles pour élaborer vos programmes.

Cette nouvelle commande est :

| ASKRAM, [type de demande], [variable]*

Cette commande permet au programme que vous écrivez de trouver certaines constantes. Elle permet, par exemple, de connaître le nombre de blocs mémoire dont dispose le programme, ce nombre variant en fonction du fait que vous utilisez une extension mémoire de 64 K ou de 256 K. Le paramètre « type de demande » est un chiffre (1, 2 ou 3) qui définit ce que vous voulez savoir. La réponse est donnée sous la forme d'une variable, de type entier, définie par le deuxième paramètre.

1000 a%=0 : | ASKRAM, 1, â a% affectera a% à la quantité de RAM

1100 a%=0 : | ASKRAM, 2, â a% affectera a% au nombre de blocs mémoire

1200 a%=0 : | ASKRAM, 2, â a% mettra a% à 0 ou à 1 selon que la RAM est ou non défectueuse

La dernière commande peut être utilisée pour s'assurer que la RAM d'extension est bien là et prête à être utilisée. Si vous voulez éviter que l'exécution de vos programmes débute par le chargement du chargeur RSX, il est possible de charger le code-machine RSX de façon indépendante :

```
 20 MODE 1 : PRINT « Chargement du programme »
 30 1 = HIMEM
 40 MEMORY 9999
 50 LOAD « rsx », 10000
 60 1 = 1—(PEEK(10004) + PEEK(10005)★256+1)
 70 POKE 10002, 1—INT(1/256)★256
 80 POKE 10003, INT(1/256)
 90 PRINT CHR$(30) ; CHR$(21) ;
100 CALL 10000
110 PRINT CHR$(30) ; CHR$(6) ;
120 a%=0 : | ASKRAM, 3, â a%
130 IF a% THEN PRINT « RAM défectueuse » : END
```

* Se reporter à l'annexe II pour la définition de [variable].

```
140 CLEAR : MEMORY PEEK(10002) + PEEK(10003)*256--1
160 CHAIN « 2e partie »
```

Le programme ci-dessus chargera le code machine RSX et le mettra en mémoire. Rien ne s'affichera sur l'écran à moins que la RAM ne se révèle défectueuse ou ne soit pas connectée. Le programme « 2e partie » constitue le programme que vous voulez exécuter. Le fait de charger le programme en deux parties supprime la nécessité de recharger le code RSX chaque fois que le programme est exécuté.

Le code machine doit être chargé en mémoire à l'adresse 10000 avant qu'il soit possible de le ranger à une autre adresse pour l'utiliser. La valeur de 16 bits contenue dans les adresses 10002 et 10003 indique l'adresse à laquelle vous voulez ranger le code-machine RSX. La valeur de 16 bits contenue dans les adresses 10004 et 10005 indique la longueur du code machine RSX qui est déplacé vers le haut de la mémoire. Les lignes de programme grâce auxquelles s'effectuent le changement d'adresse du code-machine RSX et le test de la RAM ne sont utilisées qu'une seule fois et ne sont donc pas déplacées vers le haut de la mémoire. (Elles occupent environ 1 Ko.)

Si vous voulez utiliser des caractères définis par l'utilisateur, ajoutez les lignes de programme suivantes :

```
 10 SYMBOL AFTER 256
150 SYMBOL AFTER 0
```

La valeur figurant à la ligne 150 dépendra du nombre de caractères définis par l'utilisateur que vous voulez.

Vous pouvez désirer disposer dans un programme de plusieurs jeux de caractères. Après la commande SYMBOL AFTER, HIMEM est fixé à une valeur juste au-dessous de la mémoire des caractères définis par l'utilisateur. Il est alors possible de se servir des commandes |LOADD et |SAVED pour transférer des caractères dans ou hors de la mémoire des caractères graphiques.

Si vous avez un programme qui définit un jeu de caractères, les définitions peuvent être sauvegardées ou chargées dans le bloc RAM si bien qu'un programme peut disposer de multiples jeux de caractères.

```
 10 SYMBOL AFTER 0
 20 cars = HIMEM + 1
 30 REM définition des caractères
1000 SAVE « jeul. grp », B, cars, 2048
```

Ce programme sauvegardera votre jeu de caractères sur disquette ou cassette.

Sur votre programme définitif, vous pouvez souhaiter charger un certain nombre de jeux de caractères :

```
10 SYMBOL AFTER 0
20 cars = HIMEM + 1
30 LOAD « jeu1 . grp », cars : | SAVED, 1, cars, 2048, 0
40 LOAD « jeu2 . grp », cars : | SAVED, 1, cars, 2048, 2048
50 LOAD « jeu4 . grp », cars : | SAVED, 1, cars, 2048, 4096
```

La raison pour laquelle la variable « cars » (caractères) est positionnée est que la valeur de HIMEM se modifie lors de l'accès à la disquette ou à la cassette.

Un sous-programme peut être utilisé pour sélectionner un jeu de caractères :

```
1000 REM charger les caractères en fonction de la variable jeu
1010 | LOADD, 1, cars, 2048, (jeu-1)★2048
1020 RETURN
```

Notez l'utilisation de la variable « jeu » (de caractères). Dans la séquence de chargement ci-dessus, les jeux de caractères allant de 1 à 3 seront valides. Vous pourriez, à votre convenance, en ajouter ou en supprimer.

Pour ne procéder à l'implantation des caractères qu'une seule fois, il suffit que les instructions de positionnement fassent partie du programme de chargement. Il ne sera alors pas nécessaire de charger les caractères à chaque exécution de programme.

Le positionnement des caractères peut être trouvé grâce à l'instruction :

```
200 CLEAR : SYMBOL AFTER 0 : cars = HIMEM + 1
```

Les zones de mémoire-tampon seront supprimées et la variable caractère désignera l'emplacement des caractères.

9. LECTURE ET MODIFICATION DU CONTENU D'UNE CASE-MÉMOIRE (*)

Vous disposez de deux commandes vous permettant de visualiser et de modifier, octet par octet, le contenu d'une case mémoire référencée par une adresse :

```
| PEEK, [n° de bloc], [adresse . bloc], [variable]*
| POKE, [n° de bloc], [adresse . bloc], [valeur]
```

La commande | POKE est similaire à la commande POKE que vous utilisez habituellement. Mais, il vous faudra indiquer, en plus de

l'adresse au sein du bloc et de la valeur que vous voulez inscrire dans la case mémoire, le numéro du bloc de mémoire. L'adresse au sein du bloc peut prendre une valeur allant de 0 à 16383.

| PEEK est ici une commande, plutôt qu'une fonction. Vous devez indiquer, de la même façon que pour | POKE, le numéro de bloc mémoire et l'adresse au sein du bloc de mémoire. Pour trouver la valeur, il vous faut, de même que pour la commande | ASKRAM, indiquer une variable de type entier.

Par exemple :

```
10 valeur% = 0
20 | PEEK, 3, 12345, â valeur%
30 PRINT valeur%
```

Les lignes de programme ci-dessus liront l'octet de l'adresse 12345 du bloc mémoire nº 3. Le caractère *â* indique à l'extension RSX où se trouve la variable en mémoire de sorte que son contenu puisse être modifié : l'octet requis y sera inscrit.

| PEEK et | POKE ne sont pas des commandes destinées aux programmeurs débutants. Elles ont été prévues pour permettre aux programmeurs expérimentés d'utiliser les blocs de RAM à leur convenance.

Il existe une autre commande destinée aux programmeurs expérimentés. Il s'agit de la commande | BANK :

```
| BANK, [numéro de bloc]
```

La commande est suivie d'un paramètre. Si celui-ci n'est pas indiqué, l'ordinateur prend le paramètre 0 par défaut. Le bloc de mémoire référencé est implanté dans l'espace adresse allant de 16 Ko à 32 Ko. Un numéro de bloc égal à zéro rétablira la topographie originale de la RAM ; un numéro de bloc allant de 1 au numéro maximum de bloc implantera ce bloc dans l'espace adresse ci-dessus indiqué. Si un bloc de mémoire est implanté, l'ordinateur utilisera le bloc mémoire au lieu de la RAM normale. Cependant, si la commande | LOW est utilisée, la page-écran sera lue dans la RAM d'origine. Grâce à la commande | BANK, il est possible d'utiliser l'ensemble de la mémoire pour la programmation au lieu d'avoir à positionner HIMEM (octet le plus haut de la mémoire BASIC) à l'adresse 16383. Mais il faut savoir que si le programme est interrompu tandis que la page-écran du bloc 16384-32767 est affichée, l'ordinateur écrira les données de la page-écran dans le programme BASIC, ce qui provoquera le chaos.

Pratiquez l'utilisation des commandes | BANK, | POKE et | PEEK avant de vous lancer dans l'écriture d'un grand programme faisant

appel à ces commandes. Sauvegardez fréquemment les lignes de votre programme afin de ne pas perdre l'ensemble de votre travail si vous faites une erreur.

10. PROGRAMMATION SANS LES RSX

Le programmeur peut accéder à la RAM d'extension sans passer par le logiciel RSX. Mais il lui faut comprendre la topographie de la mémoire de l'Amstrad.

Le bloc de la mémoire d'origine allant de l'adresse 16384 à l'adresse 32767 NE PEUT ETRE UTILISE pour y stocker un programme, que celui-ci soit en BASIC ou en code-machine. En BASIC, le sommet de la mémoire doit être fixé à l'adresse 16383. Le code-machine ne peut pas, lui non plus, utiliser le bloc n° 2.

La RAM d'extension est implantée en 16 blocs de l'adresse 16384 à l'adresse 32767. Lorsque le bloc est implanté, vous pouvez vous servir de la RAM d'extension comme vous le feriez de la RAM d'origine. Nous ne vous conseillons pas d'utiliser un bloc de la RAM pour un programme en code-machine, parce que si, par la suite, vous changiez le bloc, le programme disparaîtrait. Il est, cependant, possible d'écrire des programmes qui seront exécutés dans les blocs et même dans le bloc n° 2, mais il est nécessaire d'opérer le changement de bloc en dehors de cette gamme d'adresses. Avec le BASIC, il serait très difficile, mais non pas impossible, d'utiliser le bloc de RAM implanté pour y stocker des programmes. Il vous appartiendra d'utiliser ou non cette possibilité.

Voici comment procéder pour sélectionner les blocs de mémoire :

En BASIC, où « bloc » représente le numéro de bloc à implanter.

OUT &7F00, 196 + (bloc AND 3) + (bloc AND 28)★2

Note : dans ce cas, le premier numéro de bloc est 0.

Pour les extensions de 64 K, les numéros de bloc vont de 0 à 3. Pour les extensions de 256 K, les numéros de bloc vont de 0 à 15. Pour revenir à l'implantation d'origine, écrire :

OUT &7F00, 192

En CODE MACHINE, où le numéro de bloc se trouve dans l'accumulateur (A).

Sélection :

```
PUSH BC      ; A sélectionne le bloc A (sauvegarde tous les
LD C,A       ;   registres sauf A et les indicateurs)
AND 3        ; (bloc AND 3) +
LD B,A
LD A,C
```

```
AND 28          ; (bloc AND 28)★2
ADD A,A
OR B
OR 196          ; + 196
LD BC,07F00H ; BC =&7F00
OUT (C),A
POP BC
RET
```

Le premier numéro de bloc qui se trouve dans l'accumulateur est 0.

Pour revenir à l'implantation d'origine, écrire :

REMISE A L'ETAT INITIAL :

```
PUSH BC       - revenir à l'implantation initiale
LD BC,07F00H - BC=&7F00
LDA,192
OUT (C),A
POP BC
RET
```

11. QUELQUES PRÉCISIONS TECHNIQUES

ADRESSE DE CHARGEMENT

Il est possible de changer l'implantation en mémoire du logiciel RSX, après l'avoir chargé. Cependant, le programme ne peut être rangé qu'entre l'adresse 32768 et le haut de la mémoire, car le bloc de mémoire implanté apparaît, ainsi que nous l'avons vu au chapitre précédent, dans le bloc 16384-32767. Au-dessous de cette limite de 16 Ko, la table des commandes RSX ne fonctionnera plus. C'est la raison pour laquelle le code est chargé à l'adresse 10000 et est déplacé plus haut en mémoire. Si l'on appuie sur la touche ENTER au moment du chargement, le logiciel RSX se placera aussi haut que possible dans la mémoire. Inversement, vous pouvez désirer ranger le logiciel à une adresse inférieure pour réserver de la place pour vos programmes.

SAUVEGARDE SUR DISQUETTE

Le logiciel qui se trouve sur la cassette *n'est pas* protégé. Ainsi, si vous voulez le transférer sur une disquette ou une autre cassette avec une vitesse de transmission de 2000 bauds (SPEED WRITE 1), il suffit de charger les données en mémoire, puis de les sauvegarder sur le support voulu.

1. Tapez | TAPE et appuyez sur la touche ENTER (pour les systèmes à disquette)

2. LOAD « n° de bloc »
3. MEMORY 9999
4. LOAD « rsx », 10000
5. Tapez | DISC ou fixez la vitesse de transmission (SPEED WRITE) à la valeur voulue
6. SAVE « n° de bloc »
7. SAVE « rsx », B, 10000, 4000

COMPATIBILITÉ AVEC LES LOGICIELS DU COMMERCE

La RAM d'extension est compatible avec la RAM divisée en blocs qui est fournie avec le CPC 6128. Cela signifie qu'un certain nombre de programmes écrits pour le CPC 6128 pourront tourner sur le CPC 464 et le CPC 664.

En fait, le logiciel RSX fonctionnera sur le CPC 6128 qui, avec une extension de 64 Ko, aura une RAM de 128 K divisée en blocs et avec une extension de 256 Ko, une RAM de 320 K divisée en blocs.

Le logiciel RSX, tel qu'il est fourni, ne peut accéder qu'à 256 Ko de mémoire divisée en blocs (soit 16 blocs). Si vous ajoutez davantage de mémoire ou utilisez le CPC 6128 avec un module de mémoire vive de 256 Ko, il est possible de faire accéder le logiciel RSX aux 512 Ko de mémoire divisée en blocs (soit 32 blocs) en écrivant un « 1 » à l'adresse 10006. (Vous trouverez au chapitre 8 comment procéder pour charger le logiciel RSX de façon indépendante.)

Pour ce faire, ajoutez la ligne de programme suivante :

```
55 POKE 10006,1
```

Si un programme ne fonctionne pas sur votre CPC 464 ou CPC 664, procédez comme suit :

1. Il se peut que le logiciel utilise le nouveau microprogramme situé en ROM à l'adresse &BD5B. Si tel est le cas, essayez d'exécuter le programme RSX avant d'exécuter votre programme d'application.

Voici quelques-uns des programmes qui fonctionnent correctement après le chargement du logiciel RSX : le traitement de texte Tasword, et les logiciels Tas-spell et Tasprint, destinés au CPC 6128, de la société TASMAN. Le logiciel Masterfile 128 de la société Campell Systems fournira un espace de rangement de 64 Ko et les interfaces avec les logiciels de la société TASMAN.

2. Certains logiciels, qu'ils soient chargés à partir d'une disquette ou d'une cassette, ou bien lancés à partir d'une ROM, vérifieront

si la ROM de votre ordinateur est identique à celle du CPC 6128 en appelant le microprogramme situé à l'adresse &B915.

Le logiciel RSX comporte une commande supplémentaire qui permet à un CPC 464 ou 664 d'apparaître comme ayant une ROM identique au CPC 6128.

Tapez : |EMULATE et appuyez sur la touche ENTER.

Tout logiciel qui fera appel au sous-programme de vérification chargé de tester l'identité des ROM obtiendra l'information que l'ordinateur utilisé est un CPC 6128 et qu'il peut donc fonctionner normalement.

3. Le logiciel fait peut-être appel à certains éléments de la ROM du CPC 6128 qui n'existent pas dans les ROM du CPC 464 et CPC 664.

UTILISATION DU CP/M 2.2

Le système d'exploitation CP/M 2.2, présent sur les ordinateurs Amstrad, fonctionnera comme d'habitude lorsque vous utiliserez la RAM d'extension. Dans des conditions normales d'utilisation, les logiciels fonctionnant sous CP/M n'ont pas accès à cette RAM d'extension.

Mais les programmes écrits par vous-mêmes sous CP/M peuvent fort bien utiliser cet espace mémoire supplémentaire. Reportez-vous au chapitre 10 pour plus amples informations sur la façon d'utiliser la mémoire d'extension avec un programme en code-machine.

ANNEXE I. — MESSAGES D'ERREUR

Si vous commettez une erreur dans l'utilisation des commandes RSX, c'est-à-dire si vous donnez à votre ordinateur une instruction qu'il ne peut comprendre ou exécuter, il affichera un message d'erreur. Voici la liste des messages susceptibles d'apparaître sur l'écran :

1. Bad bank command
   Commande relative au bloc erronée.
   Ce message apparaît si le nombre des paramètres indiqués n'est pas exact où s'il n'y a pas de variable là où il devrait y en avoir une.
2. Bank unavailable
   Numéro de bloc non disponible.
   Vous avez essayé d'accéder à un bloc de mémoire qui n'existe pas sur votre système.
3. Bad bank parameter
   Paramètre de bloc impossible.
   Vous avez indiqué un numéro de bloc qui ne peut pas exister.
4. Bad bank address
   Adresse au sein du bloc, erronée.
   Vous avez indiqué une adresse au sein d'un bloc supérieure à 16383.
5. Value invalid
   Valeur incorrecte.
   L'adresse au sein du bloc est trop élevée pour la quantité de données définie. Le paramètre utilisé pour la commande |ASKRAM n'est pas 1, 2 ou 3. La taille d'un bloc de données à sauvegarder est supérieure à 16 Ko.
6. Bad window definition
   Numéro de fenêtre impossible.
   Le numéro de fenêtre indiqué dans les commandes |SAVEW est supérieur à 7.

ANNEXE II. — GLOSSAIRE DES COMMANDES RSX

Vous trouverez ci-dessous la liste des commandes supplémentaires dont nous avons traité dans ce manuel.

Commandes relatives aux *pages-écran*

| SAVES, [n° de bloc], [sélection page-écran]
| LOADS, [n° de bloc], [sélection page-écran]

Commandes relatives aux *fenêtres*

| SAVEW, [n° de fenêtre], [n° de bloc], [adresse . bloc], [sélect. page-écran]
| LOADW, [n° de fenêtre], [n° de bloc], [adresse . bloc], [sélect. page-écran]

Commandes relatives aux *blocs de données*

| SAVED, [n° de bloc], [point . départ], [longueur], [adresse . bloc]
| LOADD, [n° de bloc], [point . départ], [longueur], [adresse . bloc]

Commandes relatives aux *animations*

| LOW (page-écran basse)
| HIGH (page-écran haute)
| SWAP (passer de la page-écran haute à la page-écran basse et inversement)

Autres

| POKE, [n° de bloc], [adresse . bloc], [valeur]
| PEEK, [n° de bloc], [adresse . bloc], [variable]
| BANK, [n° de bloc]
| ASKRAM, [type de demande], [variable]

([type de demande], 1 = espace RAM disponible ?, 3 = nombre de blocs mémoire disponibles ?, 3 = la RAM est-elle connectée et en état de marche ?)

Définitions :

[n° de bloc]

peut aller de 1 à 4 pour les extensions de 64 Ko, et de 1 à 16 pour les extensions de 256 Ko.

[adresse-bloc]

adresse au sein du bloc. Peut prendre une valeur allant de 0 à 16383.

[sélection page-écran]

un 0 ou l'absence de valeur pour ce paramètre indique que la commande porte sur la page-écran affichée. Un 1 signifie que la commande porte sur la page-écran qui n'est pas affichée.

[point-départ] et [longueur]

définissent l'adresse de départ et la longueur d'un bloc de données situé dans la mémoire d'origine.

[variable]

donne l'adresse d'une variable de type entier qui doit être affectée, par exemple â b%.

Il est préférable de brancher l'extension quelques minutes avant l'utilisation.

Imp. DUJON - Paris