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

EXTENSIONS MEMOIRE 64 & 256 k

Ces accesoires sont disponibles pour les AMSTRAD CPC 464, 664 et 6128
En utilisant l'extension 64 k, un CPC 464 aura la même capacité et la même configuration de mémoire RAM qu'un CPC 6128. L'extension 256k donne 192 k en plus... Cette extension permettra d'utiliser le CP/M + (®) tel qu'il est livré avec le 6128, et qui ouvre à l'utilisateur une large gamme d'applications . Cela sert également à étendre la mémoire TPA à 61k sous CP/M 2.2.

La mémoire est accessible en commutant entre les banques en utilisant un seul port d'E/S. Le Z80 ne peut adresser que 64 k via un seul port d'Entrée/Sortie. Dans les CPC, il adresse la mémoire par blocs de 16k comme il le fait pour la ROM.
Avec l'extension mémoire, il adresse toujours des blocs de 16k, mais le port d'E/S détermine la combinaison de blocs en les prenant dans la mémoire centrale ou dans l'extension de RAM. Le contrôle de l'adressage du port d'E/S peut être fait en langage-machine comme en BASIC. Pour utiliser les 64 ou 256k de RAM supplémentaires, la carte d'extension est fournie avec un logiciel de commutation des blocs, bien que cette commutation puisse se faire sans ce logiciel.

Ce logiciel ajoute quelques nouvelles commandes BASIC, appelées RSX, qui permet d'utiliser la banque supplémentaire de 64 k (ou l'une des 4 banques dans le cas de l'extension 256k) pour le stockage d'écrans, de fenêtres, de graphismes et de tableaux en BASIC. On peut ainsi écrire des programmes BASIC beaucoup plus importants, puisque la mémoire normale des CPC 464 est d'habitude occupée par les tableaux, les variables et les graphiques en BASIC.

Les commandes additionnelles sont :

IBANK, n	Configure une banque de 16k directement en mémoire
ISWAP	Permute entre l'écran haut et l'écran bas
ILOW	Charge l'écran bas
IHIGH	Charge l'écran haut (écran par défaut)
ISAVES,n	Sauvegarde un écran dans une banque mémoire de 16 k.
ILOADS,n	Rappelle un écran depuis une banque de 16 k.
ISAVEW,w,n	Stocke le contenu d'une fenêtre dans la RAM d'extension
ILOAD,w,n	Charge une fenêtre avec les données venant de la RAM d'extension
ISAVED n,s,1	Transfère le contenu de la RAM normale dans la RAM d'extension
ILOADD,n,s,1	Tranfére le contenu d'une banque de la RAM d'extension dans la RAM normale
IPEEK,n,s,v	Lit la valeur d'un octet dans la RAM d'extension
IPOKE	Change un octet dans la RAM d'extension

Ces fonctions facilitent la programmation en BASIC des menus déroulants, les des animations d'écrans, des tableurs ou bases de données de grande taille, et de toutes programmations sophistiquées qui n'étaient pas possibles jusqu'ici avec les AMSTRAD CPC 464/664 sans extension mémoire.

Attention !
VOTRE ORDINATEUR DOIT ETRE ETEINT 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 MATIERES

1. MISE EN PLACE DU MODULE DE MEMOIRE 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 dénommée "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 de la connexion. (D'autres interfaces peuvent ne pas être munis de rainures ; 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 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 caractère 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.

Le nombre de pages-écran qui peuvent 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 dessinnées 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 :

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

5. FENETRES ET MENUS APPELABLES SUR L'ECRAN

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 losqu'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 la 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 x10 en MODE 1, l'espace mémoire requis pour la stocker sera inférieur à 16 Ko ; il sera, en effet, égal à 1600 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 supplèmentaire 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 1600 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 MEMOIRE D'UNE FENETRE

Si vous voulez stocker plus d'une fenêtre par bloc de mémoire, il vous faut calculer la taille de le 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$ = LOWER$ (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 ADN 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 de 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 de 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 pour un calcul. Mais, vous en 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 de15 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 MEMOIRE

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 bidimensionnel, ou tridimensionnel, vous taperez : "â stock(0,0)", ou "â stock(0,0,0)", etc.

COMMENT CALCULER L'ENCOMBREMENT MEMOIRE D'UN TABLEAU

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

L'encombrement mémoire de chaque élément du tableau (en 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 = 3233 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 @ 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 1525, 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 (RDN(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 I SAVED, 4, @ 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'écrivez 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 I LOADD. Ajoutez les lignes 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
270 IF semaine / 2 = semaine \ 2 THEN MASK 170 : ELSE MASK 255
280 | LOADD, 4, @ 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 chaînes

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, @ 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, @ 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 programme 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 zone 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 écrire 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(@ b$=1) + PEEK(@ b$+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 ( @ b$ +1 ) + PEEK ( @ 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 : INK 1,6 : 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 crée 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 appelerons 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], [sélestion 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 obtiendriez 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 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" (16Ko).

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 AVANCEE (*)

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 commande], [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,âa% affectera a% au nombre de blocs mémoire
1200 a% = 0 : | ASKRAM, 2, âa% 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
140 CLEAR / MEMORY PEEK (10002) + PEEK (10003)*256 - 1
160 CHAIN "2e partie"
```

* Se reporter à l'annexe II pour la définition de (variable)

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 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 I LOADD et I 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 "jeu1. 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 jeu 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 fonctoin de la variable jeu
1010 | LOADD, 1, cars, (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'implatation 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-MEMOIRE (*)

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 :

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

La commande I 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.

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

Par exemple :

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

Les lignes de programme ci-dessus liront l'octet de l'adresse 12345 du bloc mémoire n°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.

I PEEK et I 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 I BANK :

I 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 I LOW est utilisée, la page-écran sera lue dans la RAM d'origine. Grâce à la commande I 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 I BANK, I POKE et I 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 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 registres sauf A
LD C,A       ;    et les indicateurs)
AND 3        ; (bloc AND 3) +
LD B,A
LD A,C
AND 28           ; (bloc ADN 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 PRECISIONS 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 tranférer 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 I `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 I `DISC` ou fixez la vitesse de transmission (SPEED WRITE) à la valeur voulue
6. SAVE "n° de bloc"
7. SAVE "rsx", B, 10000, 4000

COMPATIBILITE 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 programme 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'appilcation.

   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 situéà 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 : I `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'exploitatoin CP/M2.2, présent sur les ordinateurs Amstad, 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 ou 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, erronnée.
   Vous avez indiqué une adresse au sein d'un bloc supérieure à 16383.

5. **Value invalid**
   Valeur incorrecte.
   L'adrese au sein du bloc est trop élevée pour la quantité de données définies. Le paramètre utilisé pour la commande I ASKRAM n'est pas 1, 2 ou ". 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 I 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], [valeur]
| 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 ' 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 s'origine.

[variable]

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

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

CONTENTS

DKtronics MEMORY EXPANSION

64K and 256K MEMORY EXPANSIONS.

These units are available for the CPC 464, 664 and 6128 computers.

By using the 64K upgrade the 464 and 664 computers will have the same amount and configuration of RAM as the CPC 6128. The 256K gives an extra 192K on top of this! The expansion will allow the use of CP/M+ as supplied with the CPC 6128 with its massive 61K TPA opening up an even larger software base for Amstrad users. There is also an utility for increasing the TPA on CP/M 2.2 to 61K.

The RAM can be accessed by means of bank switching using a single I/O port. Memory is actually switched in and out of the 64K Z80 address space in 16K sub-blocks, as are the ROMs. The port determines which particular combination of the original four 16K sub-blocks and any new sub-blocks from the expansion RAM will occupy the 64K address space at any time. Control of the I/O port can be from either BASIC or machine code.

To use the additional 64K/256K of RAM, the expansion is supplied with bank switching software (although it can be switched without this software).

The software adds some extra BASIC commands, RSXs, which make it possible to use the second 64K (or 3rd, 4th and 5th in the case of the 256K expansion) for storage for screens, windows, graphics and BASIC arrays. This ability means that you can write much larger BASIC programs, as most of the memory on the unexpanded CPC464/664 is normally used for arrays, variables and graphics.

The additional BASIC commands are :

|BANK,n	Map a bank of 16K directly into memory space.
|SWAP	Alternate between the low an high screens.
|LOW	Change to the low screen.
|HIGH	Change to the high screen. (Default screen).
|SAVES,n	Store a screen to a 16K bank.
|LOADS,n	Retrieve a screen from a 16K bank.
|SAVEW,w,n	Store a window's contents into expansion RAM.
|LOADW,w,n	Load a window with data from the expansion RAM.
|SAVED,n,s,l	Transfer original RAM to expansion RAM.
|LOADD,n,s,l	Load original RAM from expansion RAM.
|PEEK,n,s,v	Read the value of a byte in expansion RAM.
|POKE,n,s,v	Change a byte in the expansion RAM.

These commands make such features as pull down menus, full screen animation and large spreadsheet type programs or databases very easily programmed from BASIC as never before possible on the unexpanded CPC464 and 664 computers.

WARNING.

Ensure that the power to your Amstrad computer is switched OFF before you fit the interface to the expansion socket. Failure to comply with these instructions may cause permanent damage to the RAM pack or the computer.

1.1 Installation.

Power down your Amstrad computer. Plug the RAM pack into the socket on the back of the computer. On the CPC 464 this socket is labelled 'Floppy Disc', on the CPC 664 and CPC 6128 the socket is labelled 'Expansion'. Other expansions such as the Amstrad Disc interface for the CPC 464, DK'tronics Lightpen and Speech Synthesizer, or ROM expansions can be fitted into the expansion socket on the back of the RAM pack. Now switch on the computer.

The computer should power up as normal. If it fails to do so, check that all the connections are correctly made. Note that all DK'tronics products have a key location on the connector to ensure that there can be no alignment problems. OTHER interfaces may not have this keyway (the Amstrad disc interface is the most familiar example). Hence any connection problems will usually lie between the RAM pack and these expansions. If this is the case, try reconnecting the interfaces BEFORE inserting the RAM pack into the computer. This will give you a better view when lining up the pins.

If the computer fails to power up, or crashes on power up (Miscellaneous patterns all over the screen!), the monitor may cut out the power to the computer. On the colour monitor, just switch the MONITOR off and then attempt to reconnect as above. The monochrome monitor may have to remain switched off for several seconds before power will be reinstated to the computer.

It is very unlikely that the computer will fail to power up with the RAM pack alone. If this is the case, then the fault will probably lie with the RAM pack. * Return the RAM pack to RAM ELECTRONICS if this is the case.

* IT IS ESSENTIAL THAT YOU COMPLETE YOUR WARRANTY REGISTRATION CARD AND RETURN IT TO US IMMEDIATELY UPON PURCHASING THIS PRODUCT FROM YOUR DEALER (UK ONLY).

1.2 USING YOUR EXTRA RAM.

There are two ways to use the extra RAM. There is a cassette supplied with the RAM pack containing extensions to BASIC. Here the extra RAM can be used simply from BASIC programs. Alternatively, the RAM is accessible both from BASIC and machine code using the OUT command. The experienced programmer will be able to use the RAM for whatever he pleases and write custom software for that purpose. Commercial programs will no doubt use this approach.

The second method is described in detail in section 1.10. The first way is explained in the following chapters:-

With the computer set up as above, load the RSX software from the cassette tape supplied:-

On disc systems type '|TAPE' and press <ENTER>

b) Type 'RUN”' and press <ENTER>.
c) The loading sequence is described in detail in your Amstrad user manual.
d) When the program has finished loading, you will be asked to enter a loading address. Just press <ENTER> for now. (See section 11.)
e) The computer will test the RAM and then print out how much RAM you have got, then the computer memory will be clear ready for your own programs.

1.3 RAM TEST.

When the RSX code is first loaded, it does an extensive RAM test. Should the RAM not function correctly the program will inform you that an error has been found. Along with this, it will print out diagnostic information to help in the repair of the RAM pack.

In the unlikely event that an error is found, please note the information that is given and return the RAM pack for replacement or repair.
(See warranty registration note.)

1.4 EXTENDED BASIC COMMANDS.

There are a total of twelve extra commands provided by the RSXs on tape. Some may have parameters, some will not. Sometimes the command may have different formats and numbers of parameters. We have tried to discuss each command in its simplest form and later sections will describe added parameters which make the command more flexible and economic on memory.

You may have noticed that during the RAM test, the computer printed out the number of the 'bank' it was testing. Each bank is 16K of memory. For the 64K expansion there are 4 banks while the 256K RAM pack has 16 banks. To access a particular part of the expansion's memory there has to be a bank number and possibly a bank address.

For example, type:- '|SAVES,1' and press <ENTER>

The computer will respond with READY. What you have done is to store what was on the screen into bank 1.

Now clear the screen using CLS. To get the screen's contents back, type:- '|LOADS,1' and press <ENTER>

You can save as many screens as you have memory for. That means four screens on the 64K RAM and sixteen screens for the 256K RAM. Screen displays could be created from another program or drawn using a lightpen. Store these on tape or disc then load them back into RAM for use throughout the program. Screen displays which take a long time to create within a program, for example mazes, can be created once, then stored for instant use whenever necessary.

The command can be summarized:-

```
|SAVES,[n]  save data to bank (n= the bank number)
|LOADS,[n]  load data from bank
```

1.5 WINDOWS AND PULLDOWN MENUS.

One of the features that makes the Amstrad's windows less flexible than those on larger business machines, is the fact that the contents of a window which overlaps another are lost when the other window is used.

There are two new commands which allow the contents of windows to be saved and reloaded from RAM. This will allow the use of true pulldown menus, that can cover text, but not remove it.

EXAMPLE:-

```
NEW
 10 MODE 1
 20 FOR i=0.05 TO 1 STEP 0.05 : REM Draw grid on screen
 30     MOVE 640*i,0 : DRAW 640*i,400
 40     MOVE 0,400*i : DRAW 640,400*i
 50 NEXT i
 60 WHILE INKEY$="" : WEND : REM Wait for a key press
 70 WINDOW#1, INT(RND(1)*19+1),INT(RND(0)*9+INT(RND(1)*5+17)),
    INT(RND(1)*14+1),INT(RND(0)*14+INT(RND(1)*10+5))
 80 PEN#1, 2 : PAPER#1, 3
 90 |SAVEW,1,1 : REM Save contents of window into RAM
100 CLS#1 : REM Clear window
110 WHILE INKEY$="" : REM Wait for 2nd key press
120 PRINT#1, "This is a window"
130 WEND
140 |LOADW,1,1 : REM restore window's contents
150 GOTO 60
```

The above program uses two new commands: |LOADW and |SAVEW. As you are probably aware, there are eight windows (0-7) which can be defined. The first parameter is the reference to a window. The second is the bank number.

```
|SAVEW, [window number], (bank] save window to bank

|LOADW, (window number], (bank] load window from bank
```

See the chapters in the user manual about windows for more details.

1.5.1 MORE WINDOWING.

A window of any size, even the whole screen, will fit into a single bank of expansion RAM. This is fine if your window is nearly a full screen or will vary in size like the above example. On the other hand if your window was defined as 10 x 10 in Mode 1, then the amount of memory needed to store this window would be less than 16K. In fact only 1600 bytes are needed (see below). Thus to use a whole bank would mean wasting over 14K of memory.

To deal with this problem, the RSX window command can take an extra parameter to define where you want the window's contents to reside in the RAM bank. In the 10 x 10 window you could place the data anywhere between 0 and 14783.

The command can be written:-

```
|SAVEW, [window number], [bank], [bank address]

|LOADW, [window number], [bank], [bank address]
```

The bank address is an address between 0 and 16383. The amount of data in bytes used to store a window needs to be taken away from the top value and this leaves the range between which the data can be stored. If you put the data at the bottom of the RAM bank, at address 0, then the memory from 1600 to 16383 is free for other windows or data arrays.

HOW TO CALCULATE A WINDOW'S SIZE.

In order to have more than one window per bank, you need to know how much memory the window will take up. If the window will vary in size between two limits, use the higher of the two. Depending on which mode you are using, the figures are calculated as below.

```
In each case:     X1 is the left most x coordinate
                  X2 is the right most x coordinate
                  Y1 is the top y coordinate
                  Y2 is the bottom y coordinate

MODE 0 SIZE=(X2-X1+1) * 4 * (Y2-Y1+1) * 8

MODE 1 SIZE=(X2-X1+1) * 2 * (Y2-Y1+1) * 8

MODE 2 SIZE=(X2-X1+1) * (Y2-Y1+1) * 8
```

The computer will give an error if the window is too large to fit in the space you have allotted for it. Also if the size is miscalculated the windows may overlap in the bank and cause strange effects.

EXAMPLE 2:-

```
10 PEN 1 : PAPER 0 : MODE 1
20 size = 14 * 2 * 10 * 8
30 LOCATE 1, 13 : PRINT " 'n' for new window 'd' to remove window"
40 WINDOW 1, 14, 1, 10 : PAPER 3 : CLS
50 bankaddress=0 : level=0
60 PRINT#level, "Window";level
70 keypress$=LOWER$(INKEY$)
```

```
 80 IF keypress$="n" THEN GOSUB 110
 90 IF keypress$="d" THEN GOSUB 190
100 GOTO 60
110 If level=7 THEN RETURN
120 level=level+1
130 WINDOW#level, 1+level*3, 14+level*3, 1+level*2, 10+level*2
140 |SAVEW,level,1,bankaddress
150 bankaddress=bankaddress+size
160 PEN#level,0 : PAPER#level, (level AND 1) + 1
170 CLS#level
180 RETURN
190 IF level=0 THEN RETURN
200 bankaddress=bankaddress - size
210 |LOADW,level,1,bankaddress
220 level=level-1
230 RETURN
```

The above program only uses one bank of RAM but all 8 windows are defined. The variable 'level' is used to stand for the level of windows and the variable 'bankaddress' points to the next free place in the bank RAM.

1.6 ARRAYS, VARIABLES AND STRINGS.

There are two general purpose data moving commands to allow data from the program to be moved to and from the RAM pack.

These two commands are :

|SAVED, [bank], [Start location], [length], [bank address]

|LOADD, [bank], [Start location], [length], [bank address]

The first parameter references which bank you want to use. The start location is a memory address where there is some data. The amount of data is given as the length. Optionally a bank address can be given to allow more than one type of data to be stored in the RAM.

It is possible to save all kinds of data using these commands, but we will firstly discuss how to save simple numerical arrays these being the easiest to understand.

Say for example that your program deals with stock control of up to 60 items. You may have a string array containing the names and a numerical array containing the number of each item you have in stock.

This would use about 1K for the names and 300 bytes for the stock figures. However what if you update the stock value every week and you want to keep the last year of stock on record! Or even the last five years. Now the figures would take up about 15K or even 75K.

These could be comfortably stored on disc, or even tape for a year's stock, and the data read every time a calculation was needed, but you will probably agree that a long time would be spent waiting for reading the data each time a distribution is calculated for each item.

Obviously, it would be easier to load all the records into RAM, then access the data immediately:-

Instead of defining an array of dimensions 'stock(60,52)' taking over 15K of valuable RAM which could be used for programs, define and array 'stock(60)' Read all the data from disc a week at a time, and store each week of data into bank RAM. To do this you need to know two things. One, where does the array lie in memory? and two, how many bytes is it necessary to save?

1) Where is an array stored?

The address of any variable can be quickly found using the '@' before a variable. For example, dimension the above array:-

```
DIM stock (60)
```

Now type: PRINT @stock(0)

The computer will reply by giving the memory address where the first element of the array is stored. Try:-

```
PRINT @stock(1)
```

The number returned will be five higher in value. This is the address of the second variable.

The '@' prefix will work in front of any variable. The first item of an array is obviously '@stock(0)'. If you are using multi-dimensional arrays, the first item is '@stock(0,0)' or '@stock(0,0,0,0)' depending on the number of dimensions.

2) How long is an array?

First of all, different types of array take different numbers of bytes per element. For real numbers, there are 5 bytes per element. Integer arrays take 2 bytes per element. String arrays are of variable length. And will be dealt with later.

Next, the number of dimensions and elements needs to be taken into account. Remember that elements start from 0. This means that an array of 'stock(60)' has 61 elements. Whether or not you prefer to use the 0 element is up to you, but if you forget it, there could be some unexplainable bugs appearing in your program. Once you know the real number of elements in every dimension, simply multiply together all the dimensions to find out the total number of elements in all dimensions.

For example: 'stock(60)' has a total of 61 elements.
'stock(60,52)' has 61*53 elements = 3233 elements.
'stock%(10,5,12)' has 11*6*13 elements = 858 in all.

To find the total memory, multiply the total number of elements by the amount of memory needed by each element.

For example: 'stock(60)' takes 61*5 = 305 bytes.
'stock(60,52)' takes 3233*5 = 16165 bytes.
'stock%(10,5,12)' takes 858*2 = 1716 bytes in all.

The array we are using is 304 bytes long, and starts at @stock(0). In a single bank of RAM we can store 305 bytes about 53 times. The bank address starts at 0 and goes up in steps of 305 bytes:-

```
0 305 610 915 1220 1525 etc.
```

We shall store week 1 at bank address 305, week 2 at address 610 and so on for all 52 weeks.

Data for test purposes could be written onto disc or tape by the program below. Once the test file is written, keep it for use while you are developing your program.

```
10 OPENOUT "stock.dat"
20 FOR week=1 TO 52
30     FOR item=1 TO 60
40        Print#90, INT(RND(1)*3000+100)
40     NEXT item
60 NEXT week
70 CLOSEOUT
80 END
```

Now type 'NEW' and enter the following program:-

```
  10 DIM stock(60)
  20 INPUT "read file (y/n)";ans$
  30 IF LOWER$(ans$)="y" or LOWER$(ans$)="yes" THEN GOSUB 1000
  40 REM rest of program ....
1000 REM subroutine to read data from disc.
1010 OPENIN "stock.dat"
1020 FOR week=1 TO 52
1030       FOR item=1 TO 60
1040           INPUT#9, stock(item)
1050       NEXT item
1060      |SAVED,4,@stock(0),61*5,week*305
1070 NEXT week
1080 CLOSEIN
1090 RETURN
```

The above program could be used to read the file from disc or tape. Once the file is in bank RAM, the contents will stay there for use until the computer is switched off, or some other data is put in that bank. This means that data need only be read once from disc, then the program can be rerun without losing the data. This could also be useful too if you wish to write a number of programs to use the same data.

Once the data is in memory, you can access each week's data simply by reloading the stock array. Add the section below to draw a bar graph for a given section.

```
100 MODE 2
110 LOCATE 1,1
120 INPUT "Which item to analyse",itemno
130 IF itemno < 1 OR itemno > 60 THEN 120
```

```
140 CLS : LOCATE 30,1
150 PRINT "Bar Chart For Item"; itemno
160 LOCATE 10,25
170 PRINT"Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec": REM 3 spaces
between each.
180 FOR loop=0 TO 4
190     LOCATE 1,24-loop*5
200     PRINT STR$(loop);"000"
210 NEXT loop
220 MOVE 60,328 : 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 week=1 TO 52
270     If week/2=week/2 THEN n=1 ELSE n=2
280 |LOADD,4,@stock(0),61*5,week*305
290 ycoord=(stock(itemno)/4000*320) AND 4092
300 FOR xcoord=1 TO 11 STEP n
310     MOVE 49 + xcoord + week*11,ycoord+26 :
        DRAW 49 + xcoord + week*11, 26
320 NEXT xcoord
330 NEXT week : GOTO 110
```

1.6.1 MORE ARRAYS, VARIABLES AND STRINGS.

If you have a program that uses all the memory of the computer due to needing a large array, you can use the bank RAM for storing data without even dimensioning an array.

For example if you have a two dimensional array 'sales%(365,30)' to store the amounts of certain types of stock you sell for each day in one year. Even though you are using integers, the array uses over 22K of memory.

Instead of having the whole array in BASIC memory, each element can be accessed by using a subroutine to read out a value, and one to store a value.

```
10000 REM load 'store%' from bank memory using 'year' & 'type'.
10010 p= (year*31 + type)*2
10020 bank=1 : IF p >=16000 THEN p=p-16000 : bank=2
10030 |LOADD, bank, @store%, 2, p
10040 RETURN
11000 rem copy 'store%' to bank using 'year' & 'type'
11010 p= (year*31 + type)*2
11020 bank=1 : IF p >=16000 THEN p=p-16000 : bank=2
11030 |SAVED, bank, @store%,2, p
11040 RETURN
```

Two banks are used, 1 and 2, and the variables 'year' and 'type' are used to reference which element is required. On line 10030 and 11030, there are just 2 bytes moved to and from the bank RAM because we are using integers. The '*2' in lines 10010 and 11010 reflect the fact that an integer is stored in two bytes. If real variables were used, 5 bytes would need to be used instead. Lines 10020 and 11020 decide whether the element is in the first bank or the second.

If the array is to be filled with data from tape or disc, there is no need to initially clear the values to nil. If you want all elements preset to zero then the easiest way is to save a blank screen into each bank at the start of the program:-

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

1.6.2 STRING STORAGE.

The major obstacle in storing strings is that they can vary in length and can be stored anywhere in memory, including in a BASIC program. One method of storing string arrays is outlined below. However you may find an easier way to store strings than the one described below when you consider exactly what you want to do.

Suppose that you wanted to store 500 names, up to 20 characters long each. A bank is separated into units of memory 21 bytes each so that strings can be randomly accessed. In each 21 byte segment there is one string, and one byte to say how many letters there are in that string. That means that we will use a total of just over 10K. If we use the variable 'name' to specify the string we want then we can enter two subroutines; one to put a string from bank 1 into 'name$' and one to store the contents of 'name$' into RAM bank number 1:-

```
20000 REM assign 'name$' to string number 'name'
20010 b$="                          " : REM 21 spaces
20020 |LOADD, 1, PEEK(@b$+1) + PEEK(@b$+2)*256, 21, name*21
20030 name$=MID$(b$, 2, ASC(b$)) : RETURN
21000 REM Store 'name$' in bank as element 'name'
21010 b$="                           " : REM 21 spaces
21020 MID$(b$,1,21) = CHR$(LEN(name$)) + name$
21030 |SAVED, 1, PEEK(@b$+1) + PEEK(@b$+2)*256, 21, name*21
21040 RETURN
```

A dummy string b$ is used to form the element before it is saved into RAM. The first character is set to the length of 'name$'. The latter 20 characters are where the contents of 'name$' are stored. Then 21 characters are copied into bank RAM. When the string is retrieved the characters are copied out and 'name$' is set to the right length by looking at the first character.

String storage would come into its own if all the words were of the same length because there would be no wastage. For example a word quiz program using five, six and seven letter words. A bank of RAM could be used for each length of word. A loader program would set up the data into the RAM, then another could be CHAINed and use up to 36K of RAM for program.

A number array could also be stored in bank RAM to index the first letters and so aid the speed of access to a particular word.

1.7 ANIMATION AND PICTURE SHOWS.

We have seen how screens and windows can be stored and retrieved. Animation is the act of putting pictures on the screen quickly enough so that the eyes see something move. With 64K or 256K of memory whole screens can be stored away, then put on the screen to produce animation.

You may have noticed in section 4 that when a screen loads onto the screen, you can see each line appear. To illustrate, type in the following program:-

```
 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 : INK 1,6 : INK 2,21 : INK 3,13
110 PEN 1 : PAPER 0
120 WHILE INKEY$=""
130     FOR screen=1 TO 4
140        |LOADS, screen
150     NEXT screen
160 WEND
170 END
```

The program saves four coloured screens into bank RAM, then loads then up in sequence. Unfortunately, the effect is a striped pattern.

In order to create animation which is easy on the eye the computer needs to create the screen display, then instantly display it.

Three new instructions that allow this to be done are:-

|LOW, |HIGH and |SWAP

Before the commands can be understood it is necessary to know how the Amstrad's screen can be used. The normal screen is located at 49152. However the Amstrad is capable of viewing a screen anywhere in memory in 16K blocks. The first block at 0 and the third block at 32768 are difficult to use for screen as the computer uses these as part of the BASIC interpreter. The block of memory at 16384 is free for use as long as BASICs HIMEM is lowered to below 16384. Using this, we have called the original screen the high screen and the new screen at 16384 is called the low screen. To go from one to the other just use:-

|LOW to set the low screen in action
|HIGH to reset the high screen
|SWAP to swap from low to high and vice-versa

Whenever the swap is made, the computer is told, and all further text and graphics appear on the selected screen.

To use this facility of swapping from one screen to another instantly, the screen and window commands can have an added parameter which tells the computer to load or save the data to and from the alternate screen.

The new forms can be written:-

```
|SAVES, [ bank ], [ swap ]
|LOADS, [ bank ], [ swap ]
|SAVEW, [ window number ], [ bank ], [ bank address ], [ swap ]
|LOADW, [ window number ], [ bank ], [ bank address ], [ swap ]
```

If the swap value is zero, by default, then the command will act on the screen that is presently being displayed. Alternatively, if the value is one the computer will load and save data from the screen which is not being displayed. When the work is done, the computer can swap screens and the effect is that the screen appears to change instantly.

In the above program type these lines:-

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

Now that the computer can build up the screen while another is being displayed there is no pattern. The coloured screen appears to change instantly.

Due to the fact that the bank memory moves into the address space at 16K it takes longer for the transfer of screens to be made to the low screen than to the high screen. Hence line 135 delays the computer as it is to load the high screen. This means the time each screen is on the screen remains the same. Try removing line 135 to see the difference.

If a longer delay were to be put between line 140 and 150 you would get a picture show effect. Alternatively, you could select screens when a key is pressed.

On a small scale, a window could be defined and graphics could be rapidly displayed without resorting to swapping screens.

Note, that of less use is the fact that the contents of screens and windows can be saved from a screen which is not on display simply by adding a one for the swap parameter. For example if you want to load a series of screens from tape or disc, load them into the low (16K) screen. Messages generated by the tape system need not be switched off as the screen's contents will not be changed in the low memory screen.

```
10 LOAD "screen1",16384 : |SAVES, 3, 1
```

The above will load a screen then save it to bank 3. The screen the user sees can have something else on it.

1.8 ADVANCED PROGRAMMING.

This section introduces one new command and some other programming aspects which you may find useful.

The new command is:-

```
|ASKRAM, [ enquiry ], ( variable ]
```

The command allows certain constants to be found by the program you are writing. For example it can return the number of banks available to the program as this will change depending on whether you are using the 64K RAM pack or the 256K RAM pack. The 'enquiry' value is a number 1 to 3 which selects what you want to know. The answer is placed in an INTEGER variable defined by the second parameter.

```
1000 a%=0 : |ASKRAM, 1, @a% .. will assign a% to the amount of RAM
1100 a%=0 : |ASKRAM, 2, @a% .. will assign a% to the number of banks
1200 a%=0 : |ASKRAM, 3, @a% .. will set a% to 0 or 1 depending on
whether there is a problem with the RAM
```

The last command can be used to make sure the RAM is there and ready to use if in your programs you do not want to have to load the RSX loader first. It is possible to load just the RSX machine code on its own:-

```
 20 MODE 1 : PRINT "Program Loading!"
 30 I=HIMEM
 40 MEMORY 9999
 50 LOAD "rsx", 10000
 60 I=I-( PEEK(10004) + PEEK(10005)*256+1)
 70 POKE 10002, I-INT(I/256)*256
 80 POKE 10003, INT(I/256)
 90 PRINT CHR$(30);CHR$(21);
100 CALL 10000
110 PRINT CHR$(30);CHR$(6);
120 a%=0 : |ASKRAM, 3, @a%
130 IF a% THEN PRINT "RAM is faulty" : END
140 CLEAR : MEMORY PEEK(10002) + PEEK(10003)*256-1
160 CHAIN "part2"
```

The program above will load the RSX machine code and put it into memory. Nothing will be printed on the screen unless the RAM proves to be faulty or not even there! The program 'part2' would be the bulk of the program. Loading the program in two parts saves reloading the RSX code every time the program is run.

The code has to be loaded in at 10000 in memory before it is relocated for use. The 16 bit value in locations 10002 and 10003 is the place you want the code to be located at. Another 16 bit value in locations 10004 and 10005 contains the length of the code which is moved higher in memory. Nearly 1K of the program is only needed once - the relocation and the RAM test programs, and hence this part is not moved higher in RAM.

If you want to use user defined graphics then add the following lines:-

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

The value in line 150 will be different depending on how many user defined graphics you want.

In your program you may want to have a number of different styles of character set. After you issue a SYMBOL AFTER command, HIMEM is set just below the user defined graphics. Hence it is possible to use the |LOADD and |SAVED commands to move graphics to and from the graphics characters.

If you have a program that defines the character set, the definitions can be saved and loaded into bank RAM so that a program may have multiple character sets.

```
  10 SYMBOL AFTER 0
  20 chars=HIMEM+1
  30 REM define symbols here
1000 SAVE "set1.grp", B, chars, 2048
```

This program will save your character set onto disc or tape.

On your final program you may wish to load a number of sets:-

```
10 SYMBOL AFTER 0
20 chars=HIMEM+1
30 LOAD "set1,grp", chars : |SAVED, 1, chars, 2048, 0
40 LOAD "set2.grp", chars : |SAVED, 1, chars, 2048, 2048
50 LOAD "set4.grp", chars : |SAVED, 1, chars, 2048, 4096
```

The reason the variable 'chars' is set up is because the value of HIMEM alters when the disc or tape is accessed.

During the program, a subroutine could be used to select a character set:-

```
1000 REM given the variable 'set', load the characters
1010 |LOADD, 1, chars, 2048,(set-1)*2048
1020 RETURN
```

Note that the variable 'set' is used. In the above loading sequence sets 1 to 3 will be valid. More or less could be added as it suits you.

All of this setting up can be done on the loader program, just once. When the program is subsequently run, there is no need to reload the bank RAM. The setting of chars can be found whenever needed by:-

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

This will remove any disc buffers that have been set up and 'chars' will indeed point to the characters.

1.9 PEEKING AND POKING.

There are two commands which allow the memory in the banks to be viewed and changed byte by byte.

```
|POKE, [ bank ], [ bank address ], [ value ]
|PEEK, [ bank ], [ bank address ], [ variable ]
```

|POKE works in a similar way to the original POKE. You need to supply a bank number in addition to the normal address and value. The bank address is in the range 0 to 16383 or 0 to 16K.

|PEEK is a command rather than the normal function. The bank and bank address are the same as for |POKE. To find out the value, you need to supply an integer variable in a similar way to the |ASKRAM command.

For example:-

```
10 value%=0
20 |PEEK, 3, 12345, @value%
30 PRINT value%
```

The above will read the byte from location 12345 in bank number 3. The @ character tells the RSX extension where the variable is in memory so that its contents can be changed to the byte required.

|PEEK and |POKE are not really commands for the beginner, in fact they have only been included for the more advanced programmer who wishes to use the bank RAM in his own way.

Another advanced command which has been included for the experienced programmer is |BANK.

```
|BANK, [ bank number ]
```

The command is followed by one parameter. If this parameter is not present, a zero is assumed. The bank referenced is mapped into the address space at 16K to 32K. A bank number of zero will map the original RAM back in, numbers 1 to the maximum bank number will map that bank into the address space. If a bank is mapped in, the computer will use the bank memory instead of the normal RAM. However, the screen will still be taken from the original RAM if |LOW was issued. The advantage of this is that the whole memory can be used for programming instead of having to set the top of memory to 16383. The disadvantage is that if the program is halted while the low screen is being displayed, the computer will write screen data into the BASIC program - causing chaos.

Make sure you are accustomed to using |BANK, |POKE and |PEEK before you risk creating a large program using them. Save the program frequently in case you make a mistake and lose your work.

1.10 PROGRAMMING WITHOUT RSX's.

With no RSX software the programmer can still access the memory from the RAM banks. To use the RAM yourself, some degree of understanding of the memory map of the Amstrad is necessary.

From both BASIC and machine code, the original block of memory from 16384 to 32767 CANNOT be used for program. Hence in BASIC, you need to set the top of memory to 16383. Machine code is free to use any memory that it can normally except the block mentioned.

The extra RAM is mapped into the addresses 16384 to 32767 in 16 banks. Once the bank is mapped in, you can do anything with the RAM you normally would. It is inadvisable to use the bank RAM for machine code because if you subsequently change the bank, the program disappears! Nevertheless, programs can be written to run in banks and indeed in the original 16K block that is banked out, but it is necessary to do the bank changing outside of this memory range. In BASIC it would be extremely difficult to use the banked RAM for extra programs, but not impossible, but we shall leave that possibility up to you!

The way that banks are selected is defined below:-

```
       IN BASIC: Where 'bank' is the number of the bank to map in
       OUT &7F00, 196+ (bank AND 3) + (bank AND 28)*2
       NOTE: the bank numbers in this case START AT 0
```

For 64K expansions the banks are 0 to 3. On the 256K, bank numbers are 0 to 15.(bank 0-3 = 196-199:bank 4-7 = 204-207 etc.)

To reset the original bank:-

```
       OUT &7F00, 192
IN MACHINE CODE: Where the bank number is in the accumulator. (A)

SELECT: PUSH BC          ; select bank A (save all registers except A
        LD C,A           ; and flags)
        AND 3            ; (bank AND 3) +
        LD B,A
        LD A,C
        AND 28           ;(bank AND 28)*2
        ADD A,A
        OR B
        OR 196           ;+ 196
        LD BC,07F00H     ;BC=&7F00
        OUT (C),A
        POP BC
        RET
```

Again the bank number in the accumulator starts at 0. To reset the original bank:-

```
RESET: PUSH BC          ;reset original memory
       LD BC,07F00H     ;BC=&7F00
       LD A,192
       OUT (C),A
       POP BC
       RET
```

1.11 TECHNICAL DETAILS.

1.11.1 The Load Address.

The software which loads from tape is relocatable. However the areas of memory in which the program can go is limited to between 32768 and the top of memory. This is because the banked RAM appears in the block 16384 to 32767. (See previous chapter for explanation of why!) Below the 16K boundary, the RSX command table will no longer function. Hence, during relocation, the code is loaded at 10000 in memory and moves to a place higher in memory. Pressing <ENTER> while loading, will automatically select the highest location available. Alternatively you may wish to load the code to a lower address and reserve some space for your own programs.

1.11.2 Saving to Disc.

The software on the cassette is NOT protected. Hence to save it onto disc or even onto another tape at speed write 1 is a matter of loading the data into memory, then saving it.

1) Type '|TAPE' and press <ENTER> (for disc systems)
2) LOAD "bank"
3) MEMORY 9999
4) LOAD "rsx", 10000
5) Type '|DISC' or set SPEED WRITE as desired
6) SAVE "bank"
7) SAVE "rsx", B, 10000, 4000

1.11.3 INCREASING CP/M 2.2 TPA.

Boot up CP/M 2.2 that has CLOAD.COM on it. Copy the two programs NEWCPM.COM and OLDCPM.COM from cassette to disc by typing:-

```
CLOAD "NEWCPM.COM <enter>.(Repeat for OLDCPM.COM)
```

Then create a new CP/M system file by typing:-

```
A> MOVCPM 255 * <enter>
A> SAVE 34 NEWCPM.SYS <enter>
```

The new working disc now contains your increased TPA CP/M, invoked at any time by typing:-

```
A> NEWCPM
```

You can return to the original CP/M by typing:-

```
A> OLDCPM
```

This must be done before using some of the DFS utilities such as format etc. as these will only work with OLDCPM.

1.11.4 Commercial Program Compatibility.

The RAM expansion is compatible with the banked RAM supplied with the 6128. This means that a number of programs written for the 6128 will now work on the CPC464. The RSX software provided will work on the 6128 where the 256K pack will give 320K of banked RAM.

The bank switching software in its supplied state will only access 256K or 16 banks of memory. If you add more memory or have the CPC6128 with a 256K memory pack, the RSX software can be told to access a full 512K of banked memory (32 banks) by poking location 10006 with 1. See section 8 for explanation of how to load the RSX software on its own.

```
55 POKE 10006, 1  This line will do the trick!
```

If a commercial program fails to work on your CPC464 or CPC664 then try the suggestions below.

1) The software may be using the new firmware vector at &BD58. If this is the case, try running the RSX program before running your application program.
Some programs which will function correctly after the RSX software tape has been loaded in are Tasman's Tasword(R) word processor, Tasspell and Tasprint for the CPC6128. In conjunction with these, Campbell Systems' Masterfile 128 will provide a 64K filespace and interfaces with Tasman's software.

2) Some software, whether loaded from disc/tape or booted from a background ROM will check the ROM identity by using the firmware call &B915. There is one more command included in the RSX software on tape which will cause a CPC464 or 664 to emulate the ROM identity of the CPC6128:

Type |EMULATE and press <ENTER>

Any programs that call the ROM identity routine will now be informed that the computer is a CPC6128 and may now work correctly.

3) The software may use some features of the CPC6128 ROM which are unavailable on the CPC464 and CPC664 machines. In this instance, you may be able to get information on how the program can be altered to work on the CPC464 or 664 from the manufacturers of the program in question.

1.11.5 Using CP/M 2.2.

CP/M 2.2 as supplied with all Amstrad computers will function as normal with the extra memory fitted. However if you create and use the NEWCPM program the TPA on CP/M 2.2 will be increased to 61K.

Programs of your own devising written under this operating system are free to use the extra memory. See section 1.10 for details of how to use the extra memory from machine code.

1.12 ERROR MESSAGES.

While you are using the RSX software, there will be some occasions when the computer does not understand, or cannot carry out what you have instructed. The software may issue some error messages in addition to the normal messages that the computer will give. The errors and why they are likely to occur are outlined below:-

1) Bad bank command	Given if you have given the wrong number of parameters or if a variable is not present where there should be one.
2) Bank unavailable	You have tried to access a bank which is not present on your system.

3) Bad bank parameter	You have referenced a bank which can never be fitted to the computer.
4) Bad bank address	The address you have given is out of range: bank addresses range from 0 to 16383.
5) Value invalid	The bank address may be too large for the block of data defined. The parameter for |ASKRAM is other than 1, 2 or 3. The size of a block to be saved is larger that 16K.
6) Bad window definition	The window referenced in |SAVEW or |LOADW is above 7.

1.13 REFERENCE OF RSX COMMANDS.

All the additional commands are listed below as a reminder to their functions and syntax.

SCREENS.

```
|SAVES, [ bank ], [ swap ]
|LOADS, [ bank ], [ swap ]
```

WINDOWS.

```
|SAVEW, [ window number], [ bank ], [ bank address ], [ swap ]
|LOADW, [ window number], [ bank ], [ bank address ], [ swap ]
```

DATA BLOCKS.

```
|SAVED, [ bank ], [ start location ], [ length ], [ bank address ]
|LOADD, [ bank ], [ start location ], [ length ], [ bank address ]
```

ANIMATION.

```
|LOW  (Low screen)
|HIGH (High screen)
|SWAP (Alternate between High and Low screens)
```

OTHER.

```
|POKE, [ bank ], [ bank address ], [ value ]
|PEEK, [ bank ], [ bank address ], [ variable ]
|BANK, [ bank ]
|ASKRAM, [ enquiry ], [ variable ]
                ([ enquiry ]: 1 = RAM, 2 = banks, 3 = error occurred?)
```

DEFINITIONS.

[bank]	Bank number 1-4 or 1-16 for 64K and 256K expansions.
[bank address]	Address within bank, 0 to 16383.

[swap]	0 or omitted means act on present screen, 1 means act on alternate screen.
[start location] and [length]	Define a block of original memory.
[variable]	Give the location of an integer variable to be assigned, for example @b%.

1.14 Technical details (hardware organization).

These interfaces add either a single block (64K) or four blocks (4 x 64K) of RAM to an existing CPC464, 664 or 6128. Thus, if 64K (one block) is added to a 464, the total memory is two blocks, 128K.

For a given setup, calculate the total number of 64K blocks, this will determine which of the block select codes mentioned later are relevant to your system. The blocks are referred to by number: block one is the original 64K, block two is equivalent to the second block present in the 6128, and so on.
Memory is actually switched in and out of the 64K Z80 address space in 16K sub blocks (as are the ROMS). Which particular combination of the original four 16K sub blocks used, and any 'new' sub blocks from RAM beyond the original 64K, is called the memory map. The map is determined by an 8-bit code byte sent to the gate array control port, &7F00, with the two top bits set to 1. The following description of the codes refers only to the remaining six bits, D5-D0.

Control Codes.

Bits D2-D0 control the way 16K sub blocks are arranged in the Z80 memory space, bits D5-D3 control selection of whichever 'new' 64K block is to be used.

Bits D2-D0 - 16K Map Codes.

These bits select one of the eight maps into the 64K as follows:-

CODE	0	1	2	3	4	5	6	7
SUB BLOCK								
C000-FFFF	3	3*	3*	3*	3	3	3	3
8000-BFFF	2	2	2*	2	2	2	2	2
4000-7FFF	1	1	1*	3	0*	1*	2*	3*
0000-3FFF	0	0	0*	0	0	0	0	0

The numbers 0, 1, 2, 3 refer to the four 16K sub blocks in a 64K block in the obvious way. The star (*) indicates that the memory is from a 'new' block, i.e. block 2 or higher, otherwise the 'original', block 1, is implied.
Thus, code 0 selects the original, unmapped 64K, code 2 selects a completely new block of 64K, the other codes are a mixture.

Notes.

1. On power-up, code 0 is selected.

2. The VDU circuitry always reads from the original 64K (block 1), independently of the code.

3. If code 3 is used, reads from &4000 to &7FFF, on CPC 464 and 664 machines, will only return the correct data if the upper ROM is disabled. This is at variance with CPC 6128 operation, but is unlikely to be a significant difference.

4. If code 3 is used, addresses &4000 to &7FFF must not be used to run programs, they are intended for VDU or data access only.

Bits D5-D3, 64K Block Select Codes.

D5	D4	D3	BLOCK.
0	0	0	2 (ie, 'new' memory sub blocks came from block 2, as in CPC 6128.
0	0	1	3
0	1	0	4
0	1	1	5

Which of the above codes are relevant to your machine depends on total memory (see previous remarks).

Notes.

1. On power-up, code 0, 0, 0 is selected.

2. Bits D5, D4, D3 above 'count up' as blocks are selected. This may assist the programmer.

3. If 2x256K memory expanders are fitted to the machine, and option links are set appropriately, all patterns on D5-D3 can be used giving a maximum of 512K extra memory. (The memory that is used for a 256K silicon disc is correctly mapped to provide the extra 256K to give 512K total!)

1.15 CUSTOMIZING YOUR CP/M+ SYSTEM DISC.

Converting A 464 Keyboard Scan To That Of A 6128.

Some CP/M+ programs will not run correctly on the 464 computer because of the way the 6128 scans the keyboard.

The following program and instructions convert (fool) CP/M+ into thinking that it is running on a 6128.

1. Make a working copy of both sides of your system disc (sides 1 and 2).

   a) Use your standard CP/M 2.2 System disc and type '|CPM'

   b) For Single drive systems use DISCCOPY. For dual drives use COPYDISC. (Remember to do both sides).

   c) Put your original system disc away in a safe place to keep as a backup in case your working disc is damaged.

2. Put 'BANK' and 'RSX' onto side 1 of the working disc.

   a) Follow the instructions in the manual to get a copy onto disc. Alternatively, use 'FILECOPY BANK.BAS' and 'FILECOPY RSX.BIN' if you have transferred the software to disc already.

3. Reset the machine, then enter CP/M+. Only one drive is necessary but if you have two you may wish to disconnect the second drive because all the changes are to be made on the working disc and with a single drive the computer can use both sides.

   a) RUN "BANK" <enter> in response to 'LOAD ADDRESS?'

   b) Type in |EMULATE:|CPM <enter>

4. Type in the following, pressing <enter> after each line except where shown. Turn the disc over when the computer asks.

```
ED PATCH.ASM
i
ORG 100H
XRA A
STA 0FDEFH
JMP 0000
END
<control z>       DO NOT PRESS ENTER!
e
ED PROFILE.SUB
i
PATCH
<control z>       DO NOT PRESS ENTER!
e
(now insert the disc containing MAC.COM)
```

(continued over page...)

```
B:MAC PATCH
B:HEXCOM PATCH
ERA PATCH.HEX
ERA PATCH.SYM
ERA PATCH.PRN
ERA PATCH.ASM
```

5. It is possible to alter the CP/M+ disc to boot up without loading the BANK program first. Type the following if you want your system disc altered in this way:-

```
B:SAVE
B:SID C10CPM3.EMS
S1E0
C9
<control c>   DO NOT PRESS ENTER!
C10CPM3.EMS
Y
100
6500
```

THE DISC WILL NOW BOOT UP WITHOUT BANK BEING RUN.

dKtronics Limited,
Longs Industrial Estate, Englands Lane, Gorleston, Great Yarmouth,
Norfolk NR31 6BE, England. Tel: (0493) 602926 (5 Lines).

Printed by Lowestoft Printing Co. Ltd., Walton Road Tel: (0502) 2502.

AMSTRAD

Manual de instrucciones para la conexion de las expansiones de memoria RAM de 64K Y 256K

EDITION 1

PRECAUCION: **Asegurese de mantener desconectado su AMSTRAD antes de conectar el interface al bus de expansión. De lo contrario, se puede causar un dano permanente al paquete RAM o al ordenador.**

CONTENIDO

1 Conectando al paquete RAM

Desconecte su ordenador AMSTRAD. Conecte el paquete RAM al enchufe de la parte trasera del ordenador. En el CPC 464 esta entrada se llama "Floppy Disc", en el CPV 664 y CPC 6128 esta se llama "Expansión". En el bus de expansión de la parte trasera del paquete RAM se pueden conectar expansiones tales como el interface de disco para el CPV 464, el lapiz de luz y sintetizador de voz de Dk'tronics, o expansiones ROM.

Ahora conecte el ordenador. Este debe encenderse normalmente. Si esto no ocurre, compruebe que todas las conexiones esten hechas correctamente. Todos los productos Dk'tronics tienen una situación de tecla en el conector para impedir problemas de alineamiento. Otros interfaces pueden no tener este sistema de tecla (el caso mas conocido es el interface de disc de Amstrad). De ahí que la causa de este tipo de problemas esta entre el paquete RAM y estas expansiones. Si éste es el caso, intente conectar de nuevo los interfaces antes de insertar el paquete RAM al ordenador. Esto le dara más visión al alinear los pins.

Si el ordenador no enciende o no functiona correctamente (distintos patronas en la pantalla), es posible que el monitor puede haber cortado la corriente del ordenador.

En caso de que el monitor sea de color, desconectelo e intente la conexión de nuevo tal y como se indica arriba. El monitor monocromo deberá desconectarse unos segundos hasta que la corriente llegue al ordenador.

ES muy dificil qqu el ordenador falle al encenderse si solo esta conectado el paquete RAM. Si este es su caso, la falla probablemente sea del paquete RAM.

Devuelva el paquete RAM a Dk'tronics si es este su caso.

2 Usando el RAM adicional

Hay dos formas de usar el RAM adicional. En el cassette adjunto al paquete RAM se continen algunas extensiones al BASIC. En este caso, el RAM adicional puede ser usado simplemente para programas BASIC. Alternativamente, el RAM es accesible ya sea por BASIC y codigo máquina utilizando el comando OUT. El programador experimentado sabrá usar el RAM para aquello que necesite y escribir software adaptado a su propósito. Los programas comerciales usaran sin duda este acceso o via.

El segundo método se describe detallademente en el capítulo 10. El primer método del uso del paquete RAM se explica a continuación:

Una vez que el ordenador este listo para funcionar, cargue el cassette de software RSX adjunto al paquete RAM.

a) En sistemas de disco teclee "|TAPE" y pulse ENTER. (recuerde que el signo "|" esta en la tecla "@").
b) Teclee "RUN" y pulse ENTER.
c) La secuencia de carga esta descrita con detalle en el manual de usuario de su Amstrad.
d) Cuando el programa haya terminado de cargar, se le indicará que introduzca una dirección de memoria. Por ahora, simplemente pulse ENTER. (Ver capítulo 11).
e) El ordenador probará el RAM y le indicará el RAM disponible, y la memoria del ordenador estará libre para sus propios programas.

El cassette contiene el mismo programa por ambas caras, por lo que si una: cara falla al cargar, dispondrá de la otra.

El resto de los programas del cassette son extractos del manual que pueden ser cargados desde la cinta si no quiere teclearlos.

3 Prueba de RAM

Cuando se carga el código RSX, este hace una prueba de RAM. Si este no funciona correctamente, el programa detectará el error y le informará. Además, imprimirá la información de diagnóstico para ayudarle a reparar el paquete RAM.

En el caso poco probable de que se detecte un error, anote la información recibida por el ordenador y devuelva el paquete RAM para su reparación o recambio.

4 Comandos BASIC extendidos

Existen un total de 12 comandos en el cassette RSX. Unos tendran parámetros, otros no. Algunas veces el comando podra tener diferentes formatos y números de parámetros. Hemos intentado explicar cada comando en la forma mas simple y en los siguientes capítulos describiremos mas parámetros que hacen los comandos más flexibles y económicos para la memoria. De todas formas, los usuarios poco experimentados preferiran pasar de largo algunas secciones quizás innecesarias al leer el manual por primera vez. Estas secciones innecesarias estarán marcadas con un asterisco (★).

Los nuevos comandos van antepuestos de una barra vertical "|". Este caracter esta en la tecla del caracter "@", que se encuentra directamente a la derecha de la tecla "P".

Probablemente haya notado que durante la prueba RAM, el ordenador imprimió el número del "banco" que estaba probando. Cada banco tiene 16K de memoria. En el expansor de 64K hay 4 bancos. El paquete de 256K RAM tiene 16 bancos. Para tener acceso a alguna zona en particular de la memoria del expansor debe haber un número de banco y posiblemente una dirección de banco.

Por ejemplo, teclee:
"|SAVES, 1" y pulse ENTER

El ordenador responderá imprimiendo READY. Al hacer esto, se ha almacenado lo que estaba en la pantalla en el banco 1.

Ahora, borre el contenido de la memoria usando CLS. Para obtener el contenido de la pantalla de nuevo, teclee:

"|LOADS, 1" y pulse ENTER

Puede salver tantas pantallas como le permita la memoria. Esto es, cuatro pantallas en el RAM de 64K y dieciseis en el RAM de 256K.

Estas pantallas pueden ser creadas a partir de otro programa o ser dibujadas utilizando un "light pen". Almacene estas en cassette o disco y cárguelas de nuevo en el RAM para ser usadas en el programa. Aquellas pantallas que tardan en generarse en un programa, por ejemplo, laberintos, pueden ser creadas una vez y almacenadas para su uso inmediato cuando sea necesario.

Estos comandos pueden ser resumidos:

|SAVES, [banco]	Salvar datos del banco
|LOADS, [banco]	Cargar datos del banco

5 Ventanas y menus pulldown

Una de las caracteríisticas que hasen que las ventanas del Amstrad sean menos flexiblas que las de ordenadores de gestión más grandes, es el hecho de que el contenido de una ventana que se solapa a la otra se pierde al usarse la otra ventana.

Hay dos neuvos comandos que permiten que el contenido de una ventana sea almacenado y vuelto a cargar de la RAM. Esto permitirá el uso de verdaderos menus pulldown, que pueden cubrir el texto, pero ne quitarlo

EJEMPLO 1:

```
10 MODE 1
20 FOR I=0.05 TO 1 STEP 0.05 : REM DIBUJO POR PANTALLA
30      MOVE 640*I,0 : DRAW 640*I,400
40      MOVE 0,400*I : DRAW 640,400*I
50 NEXT I
60 WHILE INKEY$=" " : WEND : REM ESPERA QUE SE PULSE UNA TECLA
70 WINDOW #1,INT( RND(1)*19+1),INT(RND(0)*19+INT
   (RND(1)*5+17)), INT (RND(1)*14+1), INT (RND(0)*14+INT
   (RND(1)*10+5))
80 PEN #1,2:PAPER #1,3
90 |SAVEW,1,1:REM SALVAR CONTENIDO DE VENTANA AL RAM
100 CLS #1:REM LIMPIAR VENTANA
110 WHILE INKEY$=" ":REM ESPERA QUE SE PULSE UNA TECLA
120 PRINT #1, "ESTOS ES UNA VENTANA"
130 WEND
140 |LOADW,1,1:REM RESTURAR CONTENIDO DE VENTANA
150 GOTO 60
```

El programa anterior utiliza dos nueves comandos: |LOADW y |SAVEW. Como probablemente ya sepa, hay oche ventanas (0 a 7) que pueden ser definidas. El primer parámetro as la referencia de una ventana y el segundo es el número de banco.

|SAVEW, [número de ventana], [banco] almacena la ventana al banco

|LOADW, [número de ventana], [banco] carga la ventana desde el banco

Vea los capítulos sobre ventanas más detalladamente en el manual del usuario.

5a Máe sobre ventanas (★)

Une ventana de cualquier tamano, incluso de toda la pantalla, puede caber en un sólo banco del RAM de expansión. Este esta bien si la ventana ocupa casi toda la totalidad de la pantella o varia de tamano como en el ejemplo anterior. Por otro lado, si la ventana fue definida como de 10 × 10 en MODE 1, la memoria necesaria para almacenar esta ventana sera menor de 16K. De hecho, sólo se necesitan 1.600 bytes, por lo tanto utilizar un banco completo supondría perder aproximadamente 14K de memoria.

Para resolver este problema, el comando de ventana RSX puede definir un parámetro adicional para definir el lugar del paquete RAM donde desee que resida el contenido de la ventana. En la ventana de 10 × 10 podrá colocar los datos en cualquier sitio entre 0 y 14783. El comando se escribe de la siguiente forma:

|SAVEW, [número de ventana], [banco], [dirección de banco]

|LOADW, [número de ventana], [banco], [dirección de banco]

La dirección de banco corresponde a una dirección entre 0 y 16383. La cantidad de memoria en bytes, utilizada para almacenar una ventana necesita ser tomado del valor superior y esto deja el límite entre la cual los datos pueden almacenarse. Si pone los datos en el fonso del paquete RAM, en la dirección 0, entonces la memoria desde la dirección 1600 hasta 16383 esta libre para otras ventanas o para otro conjunto de datos.

Como calcular el tamano de una ventana

Para tener más de una ventana por banco, necesitará saber cuanta memoria ocupará la ventana. Si la ventana varia en tamano entre dos límites, utilize el más alto de los dos. Dependiendo del modo que este usando, las figuras se calculan como se explica abajo.

En cado caso: X1 Es la coordenada X izquierda de mayor valor
X2 Es la coordenada X derecha de mayor valor
Y1 Es la coordenada Y superior
Y2 Es la coordenada Y inferior

MODE 0 TAMANO = (X2 – X1 + 1) ★ 4 ★ (Y2 – Y1 + 1) ★ 8

MODE 1 TAMANO = (X2 – X1 + 1) ★ 2 ★ (Y2 – Y1 + 1) ★ 8

MODE 2 TAMANO = (X2 – X1 + 1) ★ (Y2 – Y1 + 1) ★ 8

El ordenador dará un error si la ventana es demasiado grande para ocupar el espacio que usted le ha asignado. De la misma forma, si el tamano es mal calculado, las ventanas pueden solaparse en el banco causando efectos raros.

EJEMPLO 2:

```
 10 PEN 1:PAPER 0:MODE 1
 20 SIZE = 14 ★ 2 ★ 10 ★ 8
 30 LOCATE 1,13 : PRINT " 'n' PARA NUEVA VENTANA 'd' PARA QUITAR
VENTANA
 40 WINDOW 1,14,1,10:PAPER 3:CLS
 50 BANKADRESS = 0:level = 0
 60 PRINT #LEVEL, "WINDOW";LEVEL
 70 KEYPRESS$ = LOWER$(INKEY$)
 80 IF KEYPRESS$ = "n" THEN GOSUB 110
 90 IF KEYPRESS$ = "d" THEN GOSUB 190
100 GOTO 60
110 IF LEVEL = 7 THEN RETURN
120 LEVEL = LEVEL + 1
130 WINDOW #LEVEL,1 + LEVEL ★ 3,14 + LEVEL ★ 3,1 + LEVEL ★ 2,10 +
LEVEL ★ 2
140 |SAVEW,LEVEL,1,BANKADRESS
150 BANKADRESS = BANKADRESS – SIZE
160 PEN #LEVEL,0:PAPER #LEVEL,(LEVEL AND 1) + 1
170 CLS #LEVEL
180 RETURN
190 IF LEVEL = 0 THEN RETURN
200 BANKADRESS = BANKADRESS – SIZE
210 |LOADW,LEVEL,1,BANKADRESS
220 LEVEL = LEVEL – 1
230 RETURN
```

El programa anterior utiliza solo un banco de RAM pero todas las ventanas (8) estan definidas. La variable "nivel" se usa para determinar el nivel de las ventanas y la variable "dirección de banco" indica el próxima lugar libre en le banco de RAM.

Hay dos comandos generales de movimiento de datos para permitir que los datos del programa sean movidos del y al paquete RAM.
Estos dos comandos son:

|SAVED, [banco], [ubicación de comienzo], [longitud], [dirección de banco]

|LOADD, [banco], [ubicación de comienzo], [longitud], [dirección de banco]

El primer parámetro indica el banco que desea usar. La ubicación de comienzo es una dirección de memoria donde hay algunos datos. La cantidad de datos viene dada como longitud. Opcionalmente una dirección de banco puede venir dada de tal forma que permita el almacenaje de más de un tipo de datos en la RAM.
Es posible salvar todo tipo de datos utilizando estos comandos, pero primeramente explicaremos como salvar arrays numéricos simples, siendo estos los más faciles de entender.
Pongamos el caso de que su programa trata de control de stock de hasta 60 items. Puede tener un array de cadena que contenga el número de cada item que tiene stock.
Así, se usará aproximadamente 1K para los nombres y 300 bytes para las figuras de stock. ?Que pasaría si quiere actualizar el valor del stock cada semana y quiere guardar el último ano de stock en un registro? o incluso los últimos cinco anos. En este caso las figuras ocuparían aproximadamente 15K o incluso 75K.
Esto podría almacenarse comodamente en disco, o incluso en cassette para un ano de stock, y los datos estarían disponibles cada vez que hiciera falta hacer un cálculo, pero probablemente estará de acuerdo en que se gastaría mucho tiempo en leer los datos cada vez que se calculase una distribución para cada item.
Obviamente sería más facil cargar todos los registros en el RAM, y tener acceso a los datos inmediatamente:
En vez de definir un array de dimensiones "stock (60,52)" ocupando aproximadamente 15K de RAM que podría ser usado para programas, definia un array "stock(60)". Lea todos los datos desde disco semanelmente y almacene los datos desde disco semanalmente y almacene los datos de cada semana en el banco RAM. Para hacer esto, necesitará saber dos cosas. Una ? en que parte de la memoria se almacena un array?, y dos, ?cuantos bytes es necesario almacenar?.

1) ?Donde se almacena un array?
La dirección de cualquier variable puede encontrarse rapidamente usando "@" antes de una variable. Por ejemplo, dimensione el array anterior:

```
DIM stock (60)
```

Ahora teclee: `PRINT @stock(0)`

El ordenador contestará dando la dirección de la memoria donde se ha almacenado el primer elemento del array. Intente:

```
PRINT @stock(1)
```

El número que sale, será cinco veces más alto en valor. Esta es la dirección de la segunda variable.
El prefijo "@" funcionará situandose en frente de cualquier variable. El primer item de cualquier array es obviamente "@stock(0)". Si esta utilizando arrays multi-direccionales, el primer item será "@stock(0,0) o "@stock(0,0,0,0) dependien do del número de dimensiones.

2) ¿Que longitud tiene un array?

Primero, los distintos tipos de arrays ocupan distintos números de bytes por elemento. Para números reales, hay cinco bytes por elemento. Los arrays de números enteros ocupan dos bytes por elemento.

Los arrays de cadena son de longitud variable. Todo esto se explicará más tarde.

Por ejemplo, "stock(60) tiene un total de 61 elementos
"stock(60,52) tiene 61 ★ 53 elementos = 3233 elementos
"stock%(10,5,12)" tiene 11 ★ 6 ★ 13 elementos = 858 elementos

Para hallar la memoria total, multiplique el número total de elementos por la cantidad de memoria necesaria para cada elmento.

Por ejemplo, "stock(60)" ocupa 61 ★ 5 = 305 bytes
"stock(60,52)" ocupa 3233 ★ 5 = 16165 bytes
"stock%(10,5,12)" ocupa 858 ★ 2 = 1716 bytes

El array que estamos usando tiene 305 bytes de longitud, y empieza en @ stock(0).

En un solo banco del RAM podemos almacenar 305 bytes aproximadamente 53 veces. La dirección de banco comienza en 0 y aumenta en pasos de 305 bytes:
0 305 610 915 1220 1525 etc.

Deberemos almacenar la semana no. 1 en la dirección de banco 305, la semana no. 2 en la dirección 610 y así sucesivamente hasta 52 semanas.

Los datos usados con el propósito de hacer tests pueden escribirse en disco o cassette mediante el situiente programa. Una vez escrito el test de archivo, guardelo para utilizarlo mientras desarrolla su programa.

```
10 OPENOUT "stock.dat"
20 FOR WEEK = 1 TO 52
30     FOR ITEM = 1 TO 60
40         PRINT #9, INT(RND(1) ★ 3000 + 100)
50     NEXT WEEK
70 CLOSEOUT
80 END
```

Ahora teclee "NEW" e intruduzca el siguiente programa:

```
10 DIM stock(60)
20 INPUT "leer archivo (s/n)";ANS$
30 IF LOWER$(ANS$) = "s" OR LOWER$(ANS$) = "si" GOSUB 1000
40 RESTO DEL PROGRAMA.
1000 REM subrutina para leer datos de disco.
1010 OPENIN"stock.dat"
1020 FOR WEEK = 1 TO 52
1030     FOR ITEM = 1 TO 60
1040         INPUT #9, STOCK(ITEM)
1050     NEXT ITEM
1060     |SAVED,4,@STOCK(0),61 ★ 5,WEEK ★ 305
1070 NEXT WEEK
1080 CLOSE IN
1090 RETURN
```


El almacenaje de cadenas podría venir por si solo si todas las palabras fuesen de la misma longitud porque así no habría desperdicio. Por ejemplo, un examen de palabras que usase palabras de cinco, seis y siete letras. Un banco de RAM puede usarse para cada longitud de palabra. Un programa cargador establecería los datos en el RAM, y otro podría encadenarse y usar hasta 36K de RAM para programar.

Un array numérico podría tambien almacenarse en el banco RAM para indexar las primeras letras y así contribuir a la velocidad de acceso a una palabra en particular.

El programa enterior puede usarse para leer el archivo de disco o cassette. Una vez que el archivo este en el banco RAM, el contenido se quedará ahí para usarse hasta que el ordenador se apague, o hasta que otros datos sean puestos en ese banco. Esto significa que los datos solo necesitan ser leidos una vez del disco, y entonces el programa puede ejecutarse de nuevo sin perder los datos. Esto puede ser tambien util si desea escribir un número de programas para utilizar los mismos datos.

Una vez que los datos estan en memoria, puede tener acceso a los datos de cada semana simplemente cargando de nuevo el array de stock. Anada la sección que damos a continuación para dibujar una gráfica de barras para una sección dada.

```
100 MODE 2
110 LOCATE 1,1
120 INPUT "cual item para analizar";ITEMNO
130 IF ITEMNO<1 OR ITEMNO>60 THEN 120
140 CLS:LOCATE 30,1
150 PRINT "grafica de barras para item";ITEMNO
160 LOCATE 10,25
170 PRINT "ene feb mar abr may jun jul ago sep oct nov dic" :
    REM 3 espacios entre cada una
180 FOR LOOP = 0 TO 4
190     LOCATE 1,24-LOOP ★ 5
200     PRINT STR$(LOOP);"000"
210 NEXT LOOP
220 MASK 225:MOVE 60,368:DRAW 60,0:DRAW 61,0: DRAW 61,368 :
    MOVE640,24: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 WEEK = 1 TO 52
270     IF WEEK/2 = WEEK/2 THEN MASK 170 : ELSE MASK 255
280 |LOADD,4,@STOCK(0),61 ★ 5,WEEK ★ 305
290 YCOORD = (STOCK(ITEMNO)/4000 ★ 320 ) AND 4092
300 FOR XCOORD = 1 TO 11
310     MOVE 49 + XCOORD + WEEK ★ 11,YCOORD + 26 :
        DRAW 49 + XCOORD + WEEK ★ 11,26
320 NEXT XCOORD
330 NEXT WEEK
340 GOTO 110
```


Si usted tiene un programa que utiliza toda la memoria del ordenador debido a la necesidad de usar un array grande, puede usar el banco RAM para almacenar datos sin necesidad de dimensionar un array.

Por ejemplo, si tiene un array bi-dimensional "sales%(366,30)" para almacenar las cantidades de ciertos tipos de stock que se venden cada día en un ano. Aunque este usando números enteros, el array usa aproximadamente 22K de memoria.

En vez de tener el array completo en la meoria BASIC, sa peude tener acceso a cada elemento una subrutina para leer un valor, y una para almacenar un valor.

```
100000 REM cargue 'store%'de la memoria del banco usando 'year' y 'type'.
10010 P=(YEAR*31+TYPE)*2
10020 BANK=1:IF P>=16000 THEN P=P-16000:BANK=2
10030 |LOADD,BANK,@STORE%,2,P
10040 RETURN
11000 REM copie 'store$' al banco usando 'year' y 'type'
11010 P=(YEAR*31+TYPE)*2
11020 BANK=1:IF P>=16000 THEN P=P-16000:BANK=2
11030 |SAVED,BANK,@store%,2,P
11040 RETURN
```

Se usan dos bancos, 1 y 2, y las variables "ano" "tipo" se usan para referenciar que elemento se requiere. En las lineas 10030 y 11030, solamente han sido movidos 2 bytes al banco RAM y desde al banco RAM, ya que usamos números enteros. El "*2" que hay en las lineas 10010 y 11010 reflejan el hecho de que un número entero se almacena en 2 bytes. Si se usasen variables reales, se necesitarían 5 bytes. Las lineas 10020 y 11020 deciden si el elemento está en el primero o segundo banco.

Si es necesario que el array se rellene con datos de disco o cinta, no hay necesidad de limpiar los valores a cero. Si quiere que todos los elementos esten a 0, la forma más facil es almacenar una pantalla en blanco en cada banco al comienzo del programa:

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

6b

Almacenaje de cadenas

El mayor obstacule para almacenar cadenas as que pueden variar de longitud y pueden almacenarse en cualquier lugar de la memoria, incluso en un programa BASIC. Un método para almacenar arrays de cadena se explica abajo. De todas formas puede encontrar un método más facil para almacenar cadenas que el que vamos a explicar, cuando considere exactamente lo que quiere hacer.

Suponga que ha querido almacenar 500 nombres, de 20 cracteres cada uno. Se separa un banco en unidades de memoria de 21 bytes cade uno de forma que se pueda acceder aleatoriamente a las cadenes.

En cada segmento de 21 bytes hay una cadena. Esto significa que usaremos un total aproximadamente de 10K. Si usamos la variable "nombre para especificar la cadena que queremos entonces podamos introducir dos subrutinas, una para poner una cadena desde el banco 1 al "name $" y otro para almacenar el contenido de "name $" al banco RAM número 1:

```
20000 REM asignar 'name$' al numero de cadena 'name'
20010 B$="                    ":REM 21 spaces
  20020 |LOADD, 1, PEEK(@B$+1)+PEEK(@B+2)*256,21,NAME*21
20030 NAME$=MID$(B$,2,ASC(B$) ):RETURN
```



```
21000 REM almacene 'name$' en el banco como elemento 'name'
21010 B$="                    ":REM 21 espacios
21020 MID$(B$,1,21 )=CHR$( LEN(NAME$) )+NAME$
21030 |SAVED,1,PEEK(@B$+1)+PEEK(@B$+2)*256,21,NAME*21
21040 RETURN
```

Una cadena imaginaria "b$" se usa para formar el elemento antes de almacenarse en el RAM. El primer carácter se establece a la longitud de "name$". Los siguientes 20 caracteres se encuentram en el lugar donde se ha almacenado el contenido de "name$". Entances se copian 21 caracteres en el banco RAM. Quando la cadena es recobrada, los caracteres son copiados y "name$" se ajusta a la longitud correcta mirando al primer carácter.

El almacenaje de cadenas podría venir por si solo si todas las palabras fuesen de la misma longitud porque así no habría desperdicio. Por ejemplo, un examen de palabras que usase palabras de cinco, seis y siete letras. Un banco de RAM puede usarse para cada longitud de palabra. Un programa cargador establecería los datos en el RAM, y otro podría encadenarse y usar hasta 36K de RAM para programar.

Un array numérico podría tambien almacenarse en el banco RAM para indexar las primeras letras y así contribuir a la velocidad de acceso a una palabra en particular.

7

Animación e imágenes (*)

Hemos visto como se pueden almacenar y volver a sacar pantallas y ventanas. La animación es el acto de poner imágenes en la pantalla tan rapidamente que los ojos vean algo en movimiento. Con 64K ó 256K de memoria se pueden almacenar pantallas completas, y luego puestas en la pantalla para crear animación.

Habra notado en la sección 4 que cuando una pantalla se carga a la pantalla, puede ver como aparece cada linea. Para ilustrarse, teclee el siguiente programa:

```
 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:INK 1,6:INK 2,21:INK 3,13
110 PEN 1:PAPER 0
120 WHILE INKEY$=" "
130     FOR SCREEN=1 TO 4
140         |LOADS, SCREEN
150     NEXT SCREEN
160 WEND
170 END
```

El programa almacena cuatro pantallas con color al banco RAM, y luego las carga secuencialmente. Sin embargo, el efecto que se obtiene es un patrón con rayas.

Para crear una animación que se adapte al ojo humano, el ordenador necesita crear el despliegue de pantalla, y desplegarse al instante.

Las tres nuevas instrucciones que permiten esto son:

|LOW |HIGH y |SWAP

Antes de entender estos comandos, es necesario saber como usar la pantalla. La pantalla normal esta localizada en 49152. De todas formas el Amstrad es capaz de "var" una pantalla de cualquier sitio de la memoria en 16384 esta libre mientras que el margen de BASIC se ponga por debajo de 16384. Usando esto, hemos llamado a la pantalla original pantalla superior y la nueva pantalla, en 16384, se llama inferior. Para ir de una a otra, simplemente use:

|LOW para poner en acción la pantalla inferior
|HIGH para reajustar la pantalla superior
|SWAP para permutar de superior a inferiror y viceversa

Cada vez que se hace una permutación, se le indica al ordenador, y todos los gráficos y textos subsiguientes aparecen en la pantalla seleccionade.

Para usar esta facilidad de permutar de una pantalla a otra instantaneamente, los comandos de pantalla y ventana pueden tener un parámetro adicional que le indique al ordenador que cargue o salve los datos de y a la pantalla alternativa.

Esta nueva forma se puede escribir:

|SAVES, [banco] , [permutación]

|LOADS, [banco] , [permutación]

|SAVEW, [número de ventana] , [banco] , [dirección de banco] , [permutación]

|LOADW, [número de ventana] , [banco] , [dirección de banco] , [permutación]

Si es valor de permutación es cero, por omisión, entonces el comando actuará en la pantalla que se este mostrando en ese momento. Alternativamente, si el valor es uno, el ordenador cargará y grabare datos de la pantalle que no se esten mostrando. Cuando el trabajo ha sido hecho, el ordenador puede permutar pantallas, y el efecto es que la pantalla parace cambiar instantaneamente.

En al programa anterior teclee estas lineas:

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

Ahora que el ordenador puede construir la pantalla mientras se muestra otra, no hay patrón. La pantalla de color parece cambiar instantaneamente.

Dado que el banco de memoria se mueve en la dirección de espacio en 16K se invierte más tiempo para la transferencia de pantallas a la pantalla inferior que a la superior. Por lo tanto, la linea 135 demors al ordenador ya que debe cargar la pantalla superior. Esto significa que el tiempo que esta cada pantalla en pantalla se mantiene igual. Intente quitar la linea 135 para ver la diferencia.

Si se pone una demora mayor entre las lineas 140 y 150 se conseguiré el efecto de muestra de pantallas. Alternativemente se pueden seleccionar pantallas cuando se pulsa una tecla.

En pequena escala, se podría definir una ventana y los gráficos se podrían mastrar rapidamente sin recurrir a pantallas permutantes.

Hay que hacer notar, que es de poca utilidad el hecho de que el contenido de pantallas y ventenas pueden almacenarse de una pantalla que no se este mostrando

simplemente anadiendo un uno para el parámetro de permutación. Por ejemplo, si quiere cargar una serie de pantallas de cassette o disco, cárguelos en la pantalla inferiro (16K). Los mensajes generados por el sistema de cassette no necesitan ser desenchufados, ya que el contenido de la pantalla no se cambierá en la pantalla de baja meoria.

```
10 LOAD "pantalla 1", 16384 : |SAVES, 3, 1
```

Esta linea hara que se cargue una pantalla y se salve al banco 3. La pantalla que ve el usuario puede tener algo mas en ella.

8 Programación avanzada (★)

Esta sección introduce un nuevo comando y algunos otros aspectos de programación que pueden serle de utilidad.

El nuevo comando es:

|ASKRAM, [pregunta] , [variable]

El comando permite que ciertas constantes se encuentren por el programa que este escribiendo. Por ejemplo puede indicar el número de bancos libres pare el programma, ya que cambiará dependiendo de si el paquete es de 64K o 256K. El valor de la "pregunta" es un número de 1 a 3 que selecciona lo que usted quiere saber. La respuesta se coloca en una variable de números enteros definida por el segundo parámetro.

```
1000 a% = 0 : |ASKRAM, 1, @a% . . asignará a% a la cantidad de RAM.

1100 a% = 0 : |ASKRAM, 2, @a% . . asignará a% al número de bancos

1200 a% = 0 : |ASKRAM, 3, @a% . . asignará a% a 0 ó 1 dependiendo si hay un
problema con al RAM.
```

El último comando puede usarse para asegurarse que el RAM esta ahí y listo para usarse si en sus programas no quiere cargar el cargador RSX primero. Es posible cargar el código maquina por separado:

```
 20 MODE 1:PRINT "programa cargando"
 30 L = HIMEM
 40 MEMORY 9999
 50 LOAD "rsx", 10000
 60 L = L - (PEEK(10004) + PEEK(10005) ★ 256 + 1)
 70 POKE 10002,L-INT(L/256) ★ 256
 80 POKE 10003, INT(L/256)
 90 PRINT CHR$(30);CHR$(21);
100 CALL 10000
110 PRINT CHR$(30);CHR$(6);
120 A% = 0 : |ASKRAM,3,@A%
130 IF A% THEN PRINT "el RAM falla":END
140 CLEAR:MEMORY PEEK(10002) + PEEK(10003) ★ 256-1
160 CHAIN "parte 2"
```

El programa anterior cargaré al codigo maquina RSX y lo pondré en memoria. No se imprimirá nada en la pantalla a menos que se compruebe que hay un fallo en al RAM o que no haya nada ahí. El programa "part 2" sería la mayoria del programa. Si se carga el programa en dos partes se puede ahorrar el tener que cargar el codigo RSX cada vez que el programa es ejecutado.

El codigo tiene que cargarse en 10000 en meoria antes de ser relocalizado para su uso. El valor de 16 bit en las localizaciones 10002 y 10003 corresponde al sitio donde quiere que el codigo se localize. Otro valor de 16 bits localizados en 10004 y 10005 contiene le longitud del codigo que se mueve en lo más alto de la memoria. Casi 1 K del programa se necesita solo una vez – la relocalización y los programas del test de RAM – y por tanto esta parte no se mueve a una zona mas alta del RAM.

Si dessea usar gráficos definidos por el usuario (UDG), anada las siguientes lineas:

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

El valor en la linea 140 será diferente dependiendo de cuantos gráficos definidos por el usuario (UDG) desee.

Quizá quiera que en su programa haya un número diferente de estilos de grupos de caracteres. Despues de dar un comando SYMBOL AFTER, el HIMEM se situa justamente debajo de los gráficos definidos por el usuario. Por lo tanto es posible utilizar los comandos |LOADD y |SAVED para mover gráficos desde y a los caracteres gráficos.

Si tiene un programa que defina el grupo de caracteres, las definiciones pueden ser salvadas y cargadas al banco RAM para así tener múltiples grupos de caracteres.

```
 10 SYMBOL AFTER 0
 20 CHARS = HIMEM + 1
 30 REM definir simbolos aqui
```

Este programa salvará su grupo de caracteres a disco o cinta. En su programa final puede querer cargar un número determinado de grupos:

```
10 SYMBOL AFTER 0
20 CHARS = HIMEM + 1
30 LOAD "set1.grp",CHARS:|SAVED,1,CHARS,2048,0
40 LOAD "set2.grp",CHARS:|SAVED,1,CHARS,2048,2048
50 LOAD "Set4.grp",CHARS:|SAVED,1,CHARS,2048,4096
```

La razón por la cual la variable "chars" se establece es que el valor de HIMEM se altera cuando se accede mediante disco o cinta.

Durante el programa, se puede usar una subrutine para seleccionar un grupo de caracteres:

```
1000 REM dada la variable 'set', cargue los caracteres
1010 |LOADD,1,CHARS,2048,(SET-1)*2048
1020 RETURN
```

Observe que se usa la variable "grupa". En la secuencia de carga anterior grupos de 1 a 3 serán válidas. Se pueden anadir mas o menos de acuerdo a sus necesidades.

Todo este establecimiente puede hacerse en el programa cargador, solo una vez. Cuando 1 programa se ejecuta, no hay que volver a cargar el banco RAM.

El establecimiento de chars puede encontrarse en todo momento mediante:


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

Esto quitará cualquier buffer de disco que se haya establecido y "chars" sin duda apuntará a los caracteres.

9 Peekeando y Pokeando (★)

Hay dos comandos que permiten que la memoria de los bancos sea visualizada y cambiada byte a byte.

|POKE, [banco] , [dirección de banco] , [valor]

|PEEK, [banco] , [dirección de banco] , [variable]

|POKE trabaja de manera similar al POKE original. Necesita suministrar un número de banco además de la dirección normal y valor. La dirección de banco esta entre los margenes do 0 a 16383 o 0 a 16K.

|PEEK es un comando, a diferencia de la función normal. El banco y dirección de banco son las mismas que para |POKE. Para hallar el valor, necesita suministrar una variable de números enteros de la misma forma que en el comando|ASKRAM.

Por ejemplo:

```
10 VALOR% = 0
20 |PEEK, 3, 12345, @valor%
30 PRINT valor%
```

El programna anterior leerá el byte de situación 12345 en el banco número 3. El caracter " " indica a la extensión RSX en que parte de la memoria esta la variable para asi poder cambiar su contenido al byte requerido.

|PEEK y |POKE no son realmente comandos para un principiante. Es más, han sido incluidos para el programador más avanzado que desee usar el banco RAM de su manera.

Otro comando avanzado que se ha incluido para el programador avanzado es |BANK.

|BANK, número de banco

El comando viene seguido de un parámetro. Si este parámetro no esta presente, se asume que es 0. El banco al que se hace referencia está desde 16K a 32K. Un número de banco 0 situará el RAM original a su sitio. Desde el número 1 hasta el número mayor de banco situaran ese banco en el espacio de direcciones. Si un banco es situado en el mapa, el ordenador usará la memoria del banco en vez del RAM normal. De todas formas, la pantalla se seguirá tomando del RAM original si se ha usado |LOW. La ventaja de esto, es que la memoria completa puede usarse para programación en vez de tener que situar la parte superior de la memoria a 16383. La desventaja es que si el programa es detenido mientras se muestra la pantalla inferior, el ordenador escribirá datos de pantalla en el programa BASIC, causando un caos.

Asegurese conocer |BANK, |POKE y |PEEK antes de crear grandes programas utilizándolos. Salve el programa que este realizando cada cierto tiempo en caso de cometer un error y perder su trabajo.

Sin software RSX el programador podrá acceder a la memoria desde los bancos RAM. Para usar el RAM por usted mismo, se necesita cierto grado de conocimiento del mapa de memoria de su Amstrad.

Para ambos BASIC y codigo maquina, el bloque de memoria original desde 16384 hasta 32767 no puede usarse para programar. Por lo tanto, en BASIC necesitará establecer la parte superior de la memoria a 16383. En Código Máquina se puede utilizar toda la emoria excepto el bloque mencionado.

El RAM adicional esta situado en las direcciones 16384 hasta 32767 en 16 bancos. Una vez que el banco este situado, puede hacer con el RAM cualquier coas. No es indicado usar el banco RAM para código máquina porque si subsecuentemente se cambia el banco, !el programa desaparece!. En cualquier caso, los programas, pueden escribirse para ejecutarse en bancos y por lo tanto en el bloque original de 16K que se ha sacado, pero es necesario hacer el cambio de banco fuera de este margen de memoria. En BASIC seria extremadamente dificil usar el RAM del banco para un programa adicional, pero no imposible. Esa posibilidad se la dejamos a usted.

La manera en que son seleccionados los bancos se define abajo:

EN BASIC: Donde "banco" es el número de banco a situar.

OUT &7F00, 196 + (banco and 3) + (banco and 28) ★ 2

NOTA: Los números de banco son de 0 a 15.

Para re-establecer el banco original:

OUT &7F00, 192

EN CODIGO MAQUINA: Donde el número de banco esta en el acumulador. (A)

```
SELECCIONE: PUSH BC        ; seleccione banco A (salva todos los registros
                           excepto A 7 banderas)
LD C,A                     ;
AND 3                      ; (banco and 3) +
                           LD B,A
                           LD A,C
AND 28                     ; (banco and 28) ★ 2
                           ADD A,A
                           OR B
OR 196                     ; + 196
LD BC,07F00H               ; BC = &7F00
                           OUT (C),A
                           POP BC
                           RET
```

De nuevo el número de banco en el acumulador comienza en 0. Para reestablecer el banco original:

```
RESTABLECER: PUSH BC       ; restablece la memoria original
             LD BC,07F00   ; BC = &7F00
             LD, A, 192
             OUT (C),A
             POP BC
             RET
```


La dirección de carga

El software que se carga desde cinta es relocalizable. De todas formas las areas de memoria en las cuales el programa puede ir estan limitadas entre 32768 y la parte superior de la memoria. Esto es asi porque el RAM banqueado aparece en el bloque 16384 hasta 32767. (Ver el capítulo anterior para averiguar porque). Por debajo del limite de 16K, la tabla de comando RSX no funcionará. Por lo tanto, durante la relocalización, el código se carga en memoria en 10000 y se mueve un espacio por encima en la memoria. Si se pulsa ENTER durante la carga, automaticamente se seleccionará la localización más alta disponible. Alternativamente, puede querer cargar el código en una dirección inferior y reservar algun espacio para sus propios programas.

Salvando el disco

El software del cassette no tiene protección. De ahí que para salvarlo a disco o incluso a otro cassette a la velocidad write 1, es cuestión de grabar los datos en la memoria, y despues salvarlo.

1) Teclee "|TAPE" y pulse ENTER (para sistema de disco).

2) LOAD "Bank"

3) MEMORY 9999

4) LOAD "rsx", 10000

5) Teclee "|DISC" o seleccionne SPEED WRITE como desee.

6) SAVE "bank"

7) SAVE "rsx", B, 10000, 4000

Compatibilidad con programas comerciales

El expansor RAM es compatible con el RAM banqueado que se suministra como estandar con el CPC 6128. Esto significa que una cantidad de programas escritos para el CPC 6128 funcionarán en el CPC 464 y el CPC 664.

De hecho, el software RSX suministrado funcionará en el CPC 6128 donde el expansor de 64K dará un total de 128K de RAM banqueado y el expansor de 256K dará un total de 320K de RAM banqueado.

Si algún programa falla al cargarlo en el CPC 464 o CPC 664, le sugerimos lo siguiente:

1) El software puede estar usando el vector de Firmware en &BD5B. Si este es el caso intente ejecutar el programa RSX antes de ejecutar el programa de aplicación.

2) El software puede chequear la ROM para determinar si se esta ejecutando en un CPC 6128. En este caso, cantactando con el fabricante del producto o con nosotros, puede alternarse el programa para que se pueda ejecutar en su ordenador.

3) El software puede usar algunas características del ROM CPC 6128, que no se encuentran en el CPC 464 y CPC 664. En este caso, contactando con el fabricante, podrá obtener información de como ejectuar el paquete.

Using CP/M

La operación del CP/M 2.2 es normal. No hay RAM adicional disponible para los programas de CP/M.

La utilización del CP/M 3 o CP/M + se explica en otro manual adjunto al expansor RAM.

Apendice **mensajes de error**

Al usar el software RSX, habré ocasiones en las cuales el ordenador no entiende, o no puede ejecutar aquello que se le ha ordenado. El software puede dar mensajes de error además de los mensajes normales. Los errores, y el porque se dan son explicados abajo:

1) Bad bank command (Mal comando de banco)	Se da si ha dado un número equivocado de pará metros o si no hay una variable donde debiese haber una.
2) Bank unavailable (Banco no disponible)	Ha intentado acceder a un banco que no está presente en su sistema.
3) Bad bank parameter (Mal parámetro de banco)	Ha referenciado un banco que nunca podrá caber en en ordenador.
4) Bad bank address (Mala dirección de banco)	La dirección que ha dado, está fuera de margen: los límites de direcciones de banco van de 0 a 16383.
5) Value invalid (Valor no valido)	La dirección de banco puede ser muy grande para el bloque de datos definido. El parámetro para |ASKRAM es distincto de 1, 2, o 3. El tamano de un bloque a salvar es mayor de 16K.
6) Bad window definition (Mala definición de ventana)	La ventana referenciada en |SAVEW o |LOADW está por encima de 7.

Apendice **Referencia de comandos RSX**

Todos los comandos adicionales se listan a continuación, como recordatorio de sus funciones y síntaxis.

PANTALLAS

|SAVES, [banco] , [permutación]
|LOADS, [banco] , [permutación]

VENTANAS

|SAVEW, [no. de ventana] , [banco] , [dirección de banco] , [permutación]
|LOADW, [no. de ventana] , [banco] , [dirección de banco] , [permutación]

BLOQUES DE DATOS

|SAVED, [banco] , [localización de comienzo] , [longitud] , [dirección de banco]
|LOADD, [banco] , [localización de comienzo] , [longituS] , [dirección de banco]

ANIMACION

|LOW	(Pantalla inferior).
|HIGH	(Pantalla superior).
|SWAP	(Alternación entre pantallas superior e inferior).

OTROS

|POKE, [banco] , [dirección de banco] , [valor]
|PEEK, [banco] , [dirección de banco] , [variable]
|BANK, [banco]
|ASKRAM, [pregunta] , [variable]
([pregunta], 1 = RAM, 2 = bancos, 3 = ?Ha occurrido error?)

DEFINICIONES

[banco]	no. de banco 1-4 o 1-6 para expansores de 64K y 256K.
[dirección de banco]	dirección dentro del banco 0 a 16383.
[permutación]	0 u omisión significa que actua en la pantalla presente, 1 significa que actua en una pantalla alternativa.
[dirección de comienzo] y [longitud]	Define un bloque de memoria original.
[variable]	Da la localización de una variable de número entero para ser asignado, por ejemplo @ b%.

PRECAUCION: **Asegurese de mantener desconectado su AMSTRAD antes de conectar el interface al bus de expansión. De lo contrario, se puede causar un dano permanente al paquete RAM o al ordenador.**