MEMOIRES DK TRONICS
64K Ram et 256K Ram

Ce manuel a été imprimé individuellement sur papier bleu "Wedgewood" A4 80g/m2 au moyen d'une imprimante EPSON EX1000. Il a été composé sur ordinateur EPSON PCAX avec le progiciel PSION XCHANGE.

Ces extensions sont disponibles pour ordinateurs CPC464 / 664 et 6128.
Une extension 64K permet un total de 128K RAM sur les 464 et 664 sans toutefois les transformer intégralement en 6128. Pour effectuer la transformation complète, une carte **FD.DOS** est indispensable.
Les extensions 256K ajoutent évidemment 192K de plus.
Elles permettent d'utiliser le CP/M+ du 6128 (avec 61K TPA); un utilitaire est fourni pour obtenir 63K TPA en CP/M 2.2

La RAM est accédée en passant de bank à bank au moyen d'un seul port I/O. La mémoire est en fait gérée par le Z80 à partir des 64K en sous-blocs de 16K (tout comme les ROMS). Le port déterminera les adresses où évolueront les combinaisons des 4 sous-blocs 16K originaux avec les sous-blocs nouveaux de toute mémoire additionnelle. Le port I/O se contrôle à partir du Basic ou en langage machine.

Pour utiliser/valider les 64K/256K additionnels, un logiciel gérant est fourni. La validation/gestion peut toutefois être effectuée sans ce logiciel.

La cassette (disquette en cas de version Française) contient des commandes extensions au Basic (RSX) qui permettent (entre autres) d'utiliser la deuxième portion 64K (ou les 3è, 4è et 5è avec les 256K RAM) pour y stocker écrans, fenêtres, graphiques et arrays (on dit aussi tableaux) Basic. En pratique, cela veut dire que vous avez dorénavant la possibilité d'écrire de très longs programmes Basic car la mémoire des CPC de base est limitée quant à l'utilisation du graphisme, variables et arrays.

Les commandes BASIC supplémentaires sont les suivantes:

|BANK,n	Positionne une bank "n" de 16K directement en mémoire
|SWAP	Passe de l'écran haut à l'écran bas
|LOW	Passe à l'écran bas
|HIGH	Passe à l'écran haut (écran par défaut)
|SAVES,n	Emmagasine un écran dans une bank "n" de 16K
|LOADS,n	Reprend un écran à partir d'une bank de 16K
|SAVEW,w,n	Emmagasine le contenu d'une fenêtre dans l'extension RAM
|LOADW,w,n	Charge une fenêtre + data à partir d'une extension RAM
|SAVED,n,s,l	Transfère la RAM originale dans l'extension RAM
|LOADD,n,s,l	Charge la RAM originale à partir de l'extension RAM
|PEEK,n,s,v	Lit la valeur d'un octet dans l'extension RAM
|POKE,n,s,v	Change la valeur d'un octet dans l'extension RAM

Des opérations qui jusqu'alors figuraient dans le domaine de l'impossible (ou du presque impossible) telles que menus déroulants, animation d'écrans entiers, vastes bases de données / tableurs seront désormais très facilement réalisables en BASIC et donc à la portée de tout possesseur de 464 / 664 / 6128!

|saves,(1à4) ou call &8656,(1à4) => &C000 à &FFFF dans la bank (1à4)
|loads,(1à4) ou call &86EA,(1à4) => la bank (1à4) dans &C000 à &FFFF

1 — INSTALLATION / BRANCHEMENT

Attention — attention !!!

Pour toute opération de branchement ou débranchement de l'extension DK TRONICS, l'ordinateur **ne DOIT PAS être sous tension.** Si vous ignorez cette précaution, vous risquez de causer des dégats permanents à l'extension et à votre ordinateur. Dans ce cas, la garantie de 6 mois est automatiquement invalidée.
Vous êtes prévenu... Ces mésaventures n'arrivent pas qu'aux autres...

1/ Coupez la tension
2/ Branchez votre extension sur le port d'extension de l'ordinateur. Celui-ci se trouve situé à l'arrière de l'ordinateur et comprend le même nombre de connecteurs (50) que le module d'extension mémoire.
Notez que le connecteur de l'extension mémoire comprend un détrompeur. Il est donc impossible (jusqu'à preuve du contraire par un surdoué de démolition artistique...) de brancher de travers!
3/ Si vous possédez lecteur DDI-1, branchez-le. Note: l'interface DDI-1 ne comporte pas de détrompeur, ce qui fait économiser quelques centimes au fabricant de l'interface; il est donc aisément possible de connecter de travers! Lisez la suite et tout ira bien:
l'encoche du connecteur arriére de l'extension DKTRONICS doit être obligatoirement insérée exactement entre les 11è et 12è broches de l'interface DDI-1
4/ Ordinateur sous tension. L'écran normal se présente. Sinon retournez à l'instruction "1/" ci-dessus, déconnectez le lecteur ainsi que l'extension puis reconnectez tout et remettez sous tension.
Note: lors de branchements mal effectués, le moniteur peut couper lui-même le courant. Avec un moniteur couleur, coupez la tension au moniteur avant toute autre chose.

2 — CHARGEMENT

Il existe deux manières d'utiliser l'extension mémoire.

1/ A l'aide de la cassette/disquette fournie avec l'extension; la RAM peut alors être directement manipulée à partir du Basic.
Alternativement, l'on peut accéder la RAM à partir du Basic et du langage machine grâce à la commande OUT. Le programmeur chevronné pourra alors s'en donner à coeur joie et écrira des tonnes de programmes et routines pour son usage personnel (et celui de ses amis s'il est généreux).

2/ La deuxième méthode est décrite en détail au chapitre 10. Léchez votre index droit et tournez les pages.

La première méthode est considérée en détail dans les chapitres suivants.

Chargeons le logiciel gérant.
1/ A partir de 6128: RUN"BANK <ENTER>
464 RUN" <ENTER>
664 |TAPE <ENTER> RUN" <ENTER>

2/ Le chargement s'effectue. Les impatients devront attendre patiemment.

3/ Une fois le programme chargé une question vous est posée dans la langue maternelle de

Bugs Bunny: **LOAD ADDRESS**. Ne vous alarmez pas et appuyez sur <ENTER> pour l'instant. (Les acharnés trouveront plus de détails au chapitre 11).

4/ L'ordinateur va tester la RAM et afficher la quantité disponible. La mémoire sera ensuite vidée, prête et à votre disposition.

3 Le test de la RAM

Un test approfondi est effectué lors du chargement des RSX. Si votre extension est défectueuse, un message vous en informera.
Si un jour (ou une nuit) un message d'erreur survient, débranchez et recommencez; si vous tapez une adresse ridicule en dehors des limites données, vous méritez un message d'erreur.
Si, après avoir vérifié instructions et connections le message survient toujours, ne commencez pas par nous retourner l'appareil avec une note disant "ça marche plus"!!! Prenez note du message affiché sur l'écran et si vous vous êtes procuré l'extension chez nous, contactez-nous par téléphone ou par courrier, il se peut que la panne soit bénigne.
Le bon fonctionnement de chaque extension a été scrupuleusement testé avant envoi, chaque appareil est donc en parfaite condition lorsqu'il nous quitte. A moins qu'une extension ne transite par les pieds ou mains d'un facteur fanatique de rugby -et dans ce cas vous aurez eu le bon sens de refuser le paquet!-, il n'y a aucune raison pour qu'elle ne fonctionne pas à l'arrivée! Les principaux cas de retours sont généralement des appareils en parfait état et que l'utilisateur s'acharnait à brancher incorrectement, des appareils "grillés" par un branchement sous tension ou des extensions défectueuses achetées "ailleurs"! (Les appareils en provenance de chez DUCHET COMPUTERS portent une marque distinctive indélébile et invisible à l'oeil nu).
Nous remplaçons bien sûr gratuitement et immédiatement une extension qui tombe en panne durant la période de garantie, mais nous facturons un minimum de 100FF les retours inutiles afin de couvrir nos frais de tests, administration, emballage, etc...

4 Les commandes RSX

Elles sont listées en page 1. Les mathématiciens endurcis en auront compté une douzaine: ils ont raison et pas tort du tout.
Certaines de ces commandes ont des paramètres, d'autres en sont dépourvues. Quelquefois, une commande peut se présenter sous des formats différents et avoir des nombres de paramètres différents.
Nous allons tout d'abord présenter ces commandes sous leur forme la plus simple; un peu plus loin nous nous aventurerons dans des manipulations plus complexes.
Si vous ne dormiez pas durant le test de RAM, vous aurez remarqué que l'ordinateur affichait le nombre de banks testées une par une. Chaque bank a 16K; donc une 64K a 4 banks alors qu'une 256K RAM en aura 16.
Pour accéder à une partie particulière d'une extension, il suffit d'indiquer la bank de 16K source/destination; en option, une adresse précise peut être indiquée.
Par exemple, si vous tapez:

```
!SAVES,1          <ENTER>
```

L'ordinateur affichera "Ready". Entretemps, ce qui se trouvait sur l'écran aura été envoyé dans la bank numéro 1.
Maintenant, videz l'écran en tapant CLS. Pour faire ré-apparaitre l'écran initial,

tapez:

```
        !LOADS,1            <ENTER>
```

Miracle!
Vous pouvez sauvegarder autant d'écrans que vous aurez de banks. Donc avec une 64K vous en aurez 4, avec un maximum de 16 écrans si vous utilisez une extension mémoire de 256K. Des écrans peuvent être créés avec un autre programme ou dessinés avec un crayon lumineux. Vous les sauvegardez sur K7 ou disquette, puis vous les chargez en RAM pour utilisation au cours du programme! Des écrans difficiles -des labyrinthes par exemple- peuvent être créés une fois et stockés en RAM pour être appelés au moment voulu.
Les commandes sont donc:
!SAVES,n Emmagasine un écran dans une bank "n" de 16K
!LOADS,n Reprend un écran à partir d'une bank de 16K
"n" est le numéro de la bank de RAM. Ne tapez pas "n" aveuglément!!!! Remplacez "n" par le chiffre/nombre qualifiant la bank!

5 FENETRES ET MENUS DEROULANTS

Tonton Alan Sugar dira peut-être le contraire et ses avocats peuvent venir tirer notre sonnette (nous avons une bassine d'eau toute prête), mais il faut bien avouer qu'avec un CPC ce n'est pas marrant de perdre le contenu d'un écran lorsque l'on superpose une fenêtre.
Voici deux commandes qui éviteront cet inconvénient. Le contenu d'une fenêtre est sauvegardé en RAM et rechargé à partir de RAM. Cette opération permettra l'utilisation de menus déroulants qui couvriront du texte sans l'effacer!
Par exemple:

```
        NEW
10      MODE 2
20      FOR i=0.02 TO 1 STEP 0.02 : REM On dessine la grille
30              MOVE 640*i,0 : DRAW 640*i,400
40              MOVE 0,400*i : DRAW 640,400*i
50      NEXT i
60      WHILE INKEY$="" : WEND
70      WINDOW#1,INT(RND(1)*19+1),INT(RND(0)*9+INT(RND(1)*5+17)),
        INT(RND(1)*14+1),INT(RND(0)*14+INT(RND(1)*10+5))
80      PEN#1,2 : PAPER#1,3
90      !SAVEW,1,1 : REM On envoie le contenu de la fenetre en RAM
100     CLS#1 : REM On vide la fenetre
110     WHILE INKEY$=""
120     PRINT#1, "Une fenetre"
130     WEND
140     !LOADW,1,1 : REM On recharge le contenu de la fenetre
150     GOTO 50
```

Lancez ce programme en faisant RUN. Lorsque la grille s'affiche, amusez-vous à appuyer sur des touches au hasard. Miracle!

Vous aurez remarqué que le programme ci-dessus utilise les deux commandes nouvelles !LOADW et !SAVEW. Comme vous le savez probablement déjà (sinon voici une bonne occasion de l'apprendre) il est possible de définir huit (0 à 7) fenêtres. Le premier paramètre est le numéro de la fenêtre, le second est celui de la bank de 16K.
Récapitulons:

```
        !SAVEW, [numéro de fenêtre], [bank]  >>>>>> sauvegarde fenêtre en bank
```

!LOADW, [numéro de fenêtre], [bank] >>>>>> charge fenêtre à partir de bank.
Si vous n'y comprenez rien, consultez le manuel de votre ordinateur au chapitre des fenêtres.

5.1 Détails des fenêtres

Une fenêtre de toute dimension, même occupant tout l'écran, "tiendra" dans une seule bank de l'extension mémoire.
C'est fantastique lorsque votre fenêtre prend tout un écran ou varie comme dans l'exemple précédent, MAIS c'est navrant si la fenêtre est définie (par exemple) comme 10x10 en mode 1; en effet cette fenêtre n'occupera que 1600 octets... il serait donc navrant de gâcher environ 14K en utilisant toute une bank!
C'est simple: il suffit de spécifier l'adresse (dans l'extension mémoire) à laquelle vous aller placer la fenêtre! Dans le cas d'une fenêtre 10x10, vous pouvez placer le data entre 0 et 14783 de la façon suivante:

!SAVEW, [numero de fenêtre], [bank], [adresse dans la bank]

et le rappeler ainsi:

!LOADW, [numero de fenêtre], [bank], [adresse dans la bank]

14783 est uniquement pour l'exemple ci-dessus! Comment calculer la bonne adresse? Lisez la suite...
L'adresse dans une bank se situera entre 0 et 16383. Déduisez la quantité de data à placer (par exemple: 16383 moins 1600) et vous obtiendrez l'adresse maximale disponible (14783 de notre exemple).
Si vous placez votre fenêtre à l'adresse 0 (zéro), tout le reste de la mémoire de 1600 jusqu'à 16383 sera disponible pour placer d'autres fenêtres, arrays, etc...

Comment calculer les dimensions, etc... d'une fenêtre

Afin d'emmagasiner plusieurs fenêtres par bank, il vous faudra savoir quelle sera la quantité de mémoire utilisée par chacune d'elles. Si les dimensions d'une fenêtre varient entre deux limites, choisissez le nombre le plus grand.
Selon le mode utilisé, les calculations seront effectuées de la manière suivante:

X1 sera le point le plus à gauche sur l'axe des x
X2 - - - - - droite - - - x
Y1 - - - - haut sur l'axe des y
Y2 - - - - bas - - - y

Mode 0 Quantité= (X2-X1+1) * 4 * (Y2-Y1+1) * 8

Mode 1 Quantité= (X2-X1+1) * 2 * (Y2-Y1+1) * 8

Mode 2 Quantité= (X2-X1+1) * (Y2-Y1+1) * 8

Si la fenêtre est trop grande pour tenir dans l'espace alloué, l'ordinateur vous enverra un message d'erreur. Si vous faites une bêtise dans vos calculations, les fenêtres vont se marcher sur les pieds et l'écran sera marrant...

EXEMPLE 2

```
10      PEN 1 : PAPER 0 : MODE 1
20      size=14 * 2 * 10 * 8
30      LOCATE 1,13 : PRINT " 'n' pour nouvelle fenetre 'e' pour effacer la nouvelle
        fenetre"
40      WINDOW 1,14,1,10 : PAPER 3 : CLS
50      bankaddress=0 : level=0
60      PRINT#level, "BONJOUR!";level
70      keypress$=LOWER$(INKEY$)
80      IF keypress$="n" THEN GOSUB 110
90      IF keypress$="e" 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
```

Le programme ci-dessus utilise une seule bank de RAM mais les 8 fenêtres sont toutes définies. La variable "level" est utilisée pour le niveau des fenêtres. La variable "bankaddress" pointe vers l'emplacement libre suivant dans la bank de RAM.

6 Variables, arrays et chaînes

Vous avez maintenant 2 commandes vous permettant d'envoyer/charger le data d'un programme vers/ à partir de l'extension mémoire.
Ces 2 commandes nouvelles sont:

|SAVED, [bank], [adresse départ], [longueur], [adresse de bank]
|LOADD, [bank], [adresse départ], [longueur], [adresse de bank]

Le premier paramètre (bank) désigne la bank que vous désirez utiliser.
L'adresse de départ est une adresse en mémoire où se trouve du data.
La quantité de data est la "longueur"
Une adresse optionelle de bank peut être donnée lorsque plus d'un type de data doit être emmagasiné en RAM.

Il est possible de sauvegarder toutes sortes de data avec ces commandes; commençons par examiner les sauvegardes d'arrays numériques:
Supposons que vous ayez un programme de gestion de stock avec jusqu'à 60 articles. Vous pourriez avoir une chaine array contenant les noms et une array numérique contenant le nombre de chaque article en stock. Cela "prendrait" environ 1K pour les noms et 300

octets pour le stock. Que va-t-il se passer si vous mettez le stock à jour toutes les semaines et désirez garder les chiffres de l'année passée en même temps? Vous allez vous amuser avec des fichiers de 15K à 75K. Pas de problème pour archiver sur disquette (ou K7...), mais les manipulations de recherches risquent ensuite de prendre un certain temps...
Avec votre extension RAM, il suffit de tout charger en RAM puis accéder le data immédiatement!
Comment faire??? Ajustez vos lunettes et lisez la suite....

Au lieu de définir une array de dimensions 'stock(60,52)' -60 articles sur 52 semaines- qui prendrait 15K de précieuse mémoire, définissez une array 'stock(60)'.
Lisez le data à partir de la disquette (ou K7 - nous n'allons plus vous casser les pieds avec des "ou K7" entre parenthèses à chaque fois; vous êtes assez grand pour comprendre tout(e) seul(e)!) une semaine à la fois puis stockez chaque semaine de data en bank de RAM.
Il vous faudra évidemment savoir 2 choses
1/ Où est-ce que l'array réside en mémoire?
2/ Combien d'octets faut-il sauvegarder?
Ce ne sont pas des questions super-banco à 5000FF... Gardez vos lunettes.

1/ Où est l'array? (pas de ricanements S.V.P.)
L'adresse de toute variable est rapidement trouvée en utilisant "@" avant cette variable. Par exemple, la dimension de l'array ci-dessus serait

```
DIM stock (60)
```

Maintenant en tapant

```
PRINT @stock (0)
```

L'ordinateur va afficher l'adresse de mémoire où réside le premier élément de l'array.
En tapant

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

l'ordinateur aura ajouté 5 au nombre précédent. C'est l'adresse de la deuxième variable. Facile, non? Si vous n'y comprenez rien, rangez tout et apprenez à tricoter.
Le préfixe "@" fonctionnera devant n'importe quel type de variable.
Le premier article de l'array est évidemment '@stock(0)'. Si vous utilisez des arrays multi-dimensionelles le premier article sera '@stock(0,0)' ou '@stock(0,0,0,0)' etc... suivant le nombre de dimensions (on dit aussi "indices").

2/ Quelle est la longueur de l'array?
Il faut tout d'abord rappeler que différents types d'arrays prennent un nombre différent d'octets par élément. En cas de simple précision (real numbers) il y a 5 octets par élément. Les types entiers (integer arrays) prennent 2 octets par élément. Les chaines seront de longueurs variables, nous y reviendrons un peu plus loin (si vous le voulez bien).
Ensuite, il faut tenir compte du nombre de dimensions et d'éléments. Souvenez-vous que l'on commence à compter à partir de 0 (ZERO), ce qui veut dire qu'une array 'stock(60)' aura 61 (soixante et un) éléments. Si vous oubliez l'élément 0 (zéro) des bugs inexplicables risquent de venir se promener dans votre programme... Lorsque vous connaissez le nombre d'éléments de chaque dimension, il suffit de multiplier les dimensions entre elles pour obtenir le nombre total d'éléments de toutes les dimensions.
Prenons quelques petits exemples pour simplifier le jargon:

'stock(60)'	a 61 éléments
'stock(60,52)'	a 61*53= 3233 éléments
'stock%(10,5,12)'	a 11*6*13= 858 éléments

Pour trouver la quantité totale de mémoire nécessaire, il suffit de multiplier le nombre

total d'éléments par la quantité de mémoire requise par chacun d'eux. Par exemple:

```
'stock(60)'          prendra 61*5= 305 octets
'stock(60,52)'       prendra 3233*5= 16165 octets
'stock%(10,5,12)'    prendra 858*2= 1716 octets
```

Simple, non?

Nous nous servons donc d'une array de 305 octets et qui commence à @stock(0). Dans une seule bank de RAM nous pouvons stocker les 305 octets environ 53 fois. L'adresse de bank commencera donc à zéro et continuera en étapes de 305 octets:

0 305 610 915 1220 1525 etc...

Dans notre exemple de gestion de stock, nous placerons la première semaine à l'adresse de bank 305, la deuxième à 610 et ainsi de suite pour les 52 semaines.

Le programme ci-dessous vous permet de créer un fichier test (sauvegardable sur disc et/ou K7). Une fois le fichier écrit, gardez-le pour développer votre programme

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

Maintenant tapez 'NEW' puis tapez le programme suivant:

```
  10    DIM stock(60)
  15    Mode 1
  20    INPUT "On lit le fichier (o/n)";reponse$
  30    IF LOWER$(reponse$)="o" or LOWER$(reponse$)="oui" THEN GOSUB 1000
  40    REM suite du programme.....
1000    REM sous routine pour lire data a partir de la disquette/K7
1011    OPENIN "stock.dat"
1020    FOR semaine=1 TO 52
1030       FOR article=1 TO 60
1040          INPUT#9, stock(article)
1050       NEXT article
1060       |SAVED,4,@stock(0),61*5,semaine*305
1070    NEXT semaine
1080    CLOSEIN
1090    END
```

Le programme ci-dessus permet de lire le data à partir de disquette/K7. Quand le fichier est en bank RAM, son contenu y restera tant que vous n'éteignez pas l'ordinateur, ou faites une remise à zéro ou envoyez d'autre data par-dessus celui qui est déjà présent dans la même bank.

Cela signifie que le data ne doit être chargé qu'une seule fois par "session", le programme pouvant être RUN sans risque de perdre le data! Une note de précaution toutefois: expérimentez un peu avant de vous lancer dans des opérations "sérieuses" risquant d'entrainer la faillite d'une douzaine d'entreprises (même nationalisées) si vous faites tout planter!!!

Le listing ci-dessus peut aussi vous rendre service si vous écrivez un certain nombre de programmes utilisant le même data.

Une fois le data en mémoire, vous pouvez accéder aux données de chaque semaine tout simplement en rechargeant l'array de stock!
Ajoutez le listing ci-dessous et vous aurez les graphiques se rapportant à une section donnée!

```
100     MODE 2
110     LOCATE 1,1
120     INPUT "Quel article va-t-on analyser",articleno
130     IF articleno<1 OR articleno>60 THEN 120
140     CLS:LOCATE 30,1
150     PRINT "Representation graphique ARTICLE" ; articleno
160     LOCATE 10,25
170     PRINT"Jan Fev Mar Avr Mai Jun Jui Aou Sep Oct Nov Dec":
        REM 3 espaces entre chacun
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 60,24 :
        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 n=1 ELSE n=2
280     |LOADD,4,@stock(0),61*5,semaine*305
290     ycoord=(stock(articleno)/4000*320) AND 4092
300     FOR xcoord=1 TO 11 STEP n
310        MOVE 49+xcoord+semaine*11,ycoord+26 : DRAW 49+xcoor +semaine*11,26
320     NEXT xcoord
330     NEXT semaine:GOTO 110
```

6.1 D'AUTRES ARRAYS, CHAINES & VARIABLES

Si un de vos programmes utilise toute la mémoire disponible parce qu'il demande une très large array, vous pouvez vous servir d'une bank RAM pour stocker du data sans même avoir à dimensionner une array!
Par exemple, si vous avez une array bi-dimensionnelle 'ventes%(365,30)' pour emmagasiner les quantités de certains types de stocks vendus chaque jour de l'année: bien que vous utilisiez des entiers, l'array dévorera plus de 22K de mémoire!
Au lieu de laisser l'array se promener en mémoire basic, chaque élément peut être accéder au moyen d'une sous-routine pour lire une valeur et d'une autre pour emmagasiner une valeur.

```
10000  REM charger 'emmag%' a partir de bank memoire en utilisant 'annee' & 'type'
10010  p=(annee*31+type)*2
10020  bank=1 : IF p >=16000 THEN p=p-16000 : bank=2
10030  |LOADD, bank, @emmag%, 2, p
10040  RETURN
11000  REM copy 'emmag%' en bank en utilisant 'annee' & 'type'
11010  p=(annee*31+type)*2
```

```
11020  bank=1 : IF p >=16000 THEN p=p-16000 : bank=2
11030  !SAVED, bank, @emmag%,2, p
11040  RETURN
```

On utilise les banks 1 et 2; les variables 'annee' et 'type' réfèrent aux éléments requis. Comme nous utilisons des entiers, seulement 2 octets vont et viennent en bank RAM (lignes 10030 et 11030)
Si nous avions pris des types de simple précision (real numbers) 5 octets auraient été requis.
Les lignes 10020 et 11020 décideront si l'élément est dans la première ou seconde bank.
Si l'array doit être remplie de data à partir de disc ou K7, ce n'est pas la peine de mettre les valeurs à zéro. Si vous voulez avoir tous les éléments réglés à zéro il suffit tout bonnement de sauvegarder un écran vide dans chaque bank au début du programme:

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

6.2. Stockage de variables

Il n'est pas toujours très simple de stocker les chaines car elles varient en longueur et peuvent être placées un peu partout en mémoire et dans les programmes BASIC.
Nous vous indiquons ci-dessous une méthode pour stocker les chaines; les malins superhypersurdoués en connaitront sûrement d'autres... tant mieux.
Restons toutefois avec notre exemple destiné au commun des mortels!

Supposons (par exemple) que vous aimeriez établir une liste de 500 noms jusqu'à 20 caractères chacun. La bank est découpée en petits morceaux de mémoire d'une longueur de 21 octets chacun pour que l'on puisse accéder aux chaines de façon aléatoire.
Dans chaque segment de 21 octets, on aura la chaine + 1 octet pour dire le nombre de caractères de cette chaine.
Nous aurons donc un total de un peu plus de 10K.
Si nous prenons une varaible 'nom' pour spécifier la chaine voulue, nous pouvons écrire 2 sous-routines; l'une pour mettre la chaine voulue de bank1 dans 'nom$', l'autre pour emmagasiner le contenu de 'nom$' dans la bank RAM numéro1. Allons-y:

```
20000  REM mettons 'nom$' a la chaine 'nom'
20010  b$="                    " : 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 emmagasine 'nom$' en bank comme element 'nom'
21010  b$="                    " : REM 21 espaces
21020  MID$( b$,1,21 )=CHR$( LEN(nom$) )+nom$
21030  !SAVED, 1, PEEK( @b$+1 )+PEEK( @bs+2 )*256, 21, nom*21
21040  RETURN
```

Une chaine "dummy" b$ est utilisée pour former l'élément avant qu'il ne soit sauvegardé en RAM. Le premier caractère est indexé à la longueur de 'nom$'. Les 20 derniers caractères sont où le contenu de 'nom$' est emmagasiné. Les 21 caractères sont ensuite copiés en bank RAM.
Quand la chaine est récupérée, les caractères sont recopiés en sens inverse et le premier caractère est inspecté pour donner sa longueur requise à 'nom$'.

Ce type de stockage de chaine serait idéalement utilisé pour des aplications de mots de même longueur. Fantastique pour les fanatiques de mots fléchés, Scrabble, etc...! Une bank de RAM pourrait être utilisée pour chaque groupe de même longueur; un programme loader enverrait tout le data en RAM et un autre pourrait être CHAIN-é, utilisant jusqu'à 36K de RAM pour le programme!
Une array numérique pourrait aussi être stockée en RAM pour indexer les premières lettres, ce qui permettrait d'accélérer la vitesse d'accès à un mot particulier.

7 Dessins animés!

Nous avons déjà vu comment écrans et fenêtres peuvent être stockés et rechargés.
L'animation est la manière de placer rapidement des images sur un écran afin que les yeux perçoivent un mouvement. Avec des extensions de 64K -ou mieux de 256K- des écrans entiers peuvent être stockés en RAM puis renvoyés sur l'écran en créant des effets optiques.
Vous aurez remarqué -chapitre 1.4- que lorsqu'un écran est chargé vous pouvez voir apparaitre chaque ligne. Maintenant examinons l'exemple suivant

```
 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 ecran=1 TO 4
140          |LOADS, ecran
150          NEXT ecran
160    WEND
170    END
```

Le programme sauvegarde 4 écrans colorés en bank RAM, puis les recharge en séquence. Malheureusement l'effet est à rayures!
Pour créer des animations agréables à l'oeil, l'ordinateur doit créer son écran puis l'afficher immédiatement.
Trois commandes nouvelles vont nous aider:
|LOW |HIGH |SWAP

Michael Jackson, qui parle Anglais lorsqu'il ne chante pas, vous dira -si vous lui demandez gentiment- que LOW veut dire BAS, HIGH=HAUT et que SWAP veut dire échanger.
Expliquons... L'écran normal est situé à l'adresse 49152 (&C000); toutefois un AMSTRAD CPC peut "regarder" un écran un peu n'importe où dans la mémoire en blocs de 16K. Le premier bloc (commence à l'adresse 0) et le troisième bloc (qui commence à 32768) ne sont pas trop pratiques car le système BASIC de l'ordinateur va aussi se promener par là. Le deuxième bloc (qui commence à 16384) = y a bon! BASIC HIMEM doit toutefois être situé en-dessous de 16384.
Récapitulons donc: nous allons appeler l'écran original "Haut" (=HIGH) et le nouvel

écran à 16384 sera l'écran "Bas" (=LOW).
Pour passer de l'un à l'autre, il suffit d'utiliser:

```
|LOW      pour activer l'écran BAS
|HIGH     pour ré-activer l'écran HAUT
|SWAP     pour passer de haut en bas et vice versa
```

Chaque fois que "SWAP" est validé, l'ordinateur comprendra et tout texte ou graphisme subséquent sera automatiquement dirigé vers l'écran activé. Faites donc attention de ne pas tout mélanger entre 2 écrans!!!

Un peu plus difficile:
Afin de mieux utiliser cette facilité permettant de passer d'un écran à l'autre instantanément, il suffit d'ajouter un paramètre aux commandes d'écran et de fenêtre! Ce paramètre ordonnera à votre CPC de charger ou sauvegarder votre data vers / à partir de l'écran alternatif.
Cela s'écrira sous la forme suivante

```
|SAVES, [bank], [swap]
|LOADS, [bank], [swap]
|SAVEW, [fenetre n], [bank], [adresse bank], [swap]
|LOADW, [fenetre n], [bank], [adresse bank], [swap]
```

Si la valeur de swap est 0 (zéro) -sa valeur par défaut- la commande agira sur l'écran affiché. Si la valeur est 1, la commande agira sur l'écran qui n'est pas affiché! Une fois le travail fini, l'ordinateur pourra échanger les écrans... L'effet produit étant que l'écran paraisse changer instantanément.

Ajoutez maintenant ces lignes au programme ci-dessus:

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

L'ordinateur peut maintenant préparer un écran pendant qu'un autre est affiché. L'écran affiché ne va donc plus représenter des rayures bizarres! L'écran couleur semble changer instantanément.

Une précision technique: il faut plus longtemps à un écran "bas" pour se placer en mémoire qu'un écran "haut"; la ligne 135 retarde l'ordinateur alors qu'il se prépare à charger l'écran "haut". Le temps d'affichage de chaque écran est alors le même. Si vous en doutez, enlevez la ligne 135 et vous verrez bien!
Si un délai plus long était placé entre les lignes 140 et 150, l'on pourrait obtenir un effet "lanterne magique". Il serait aussi possible de sélectionner un écran en pressant une touche.

A une échelle réduite, une fenêtre pourrait être définie avec des graphiques apparaissant rapidement sans même avoir à changer d'écrans!

Une précision ne servant pas à grand chose mais pouvant éventuellement être utile: le contenu d'un écran ou fenêtre peut être sauvegardé à partir d'un écran qui n'est pas affiché en ajoutant 1 au paramètre "swap". Par exemple, si voulez charger une série d'écrans à partir de disc ou K7, chargez-les dans l'écran BAS (16k). Les messages de K7 n'ont pas besoin d'être invalidés car ils ne seront pas sur le même écran!

```
10 LOAD "ecran1",16384 : |SAVES, 3, 1
```

Le tout petit programme ci-dessus chargera un écran et le sauvegardera en bank 3. L'écran vu par l'opérateur peut afficher autre chose!

8 Programmation avancée

Cette section introduit une commande nouvelle ainsi que certains aspects pouvant -souhaitons-le!- vous être utiles un jour (ou une nuit).
La nouvelle commande est:

```
|ASKRAM, [question], [variable]
```

Cette commande permet au programme en cours de trouver certaines constantes. Elle peut, par exemple, vous indiquer le nombre de banks disponibles au programme, celui-ci pouvant changer suivant l'extension(s) utilisée(s). La valeur "question" est un nombre entre 1 et 3 qui sélectionne ce que vous désirez savoir. La réponse est placée dans une variable entière définie par le second paramètre.

```
1000 a%=0 :|ASKRAM, 1, @a%          Assignera a% à la quantité de RAM
1100 a%=0 :|ASKRAM, 2, @a%              -       - au nombre de banks
1200 a%=0 :|ASKRAM, 3, @a%          Mettra 0 ou 1 en a%, suivant qu'il y ait ou non un
problème avec la RAM
```

La dernière commande permet de vérifier que la RAM soit présente et prête à être utilisée. Si, par exemple vous ne voulez pas charger le loader des RSX, il est possible de charger directement la partie langage machine des RSX:

```
20      MODE 1 : PRINT "Le programme se charge"
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 "La RAM n'est pas contente" : END
140     CLEAR : MEMORY PEEK (10002 )+PEEK( 10003 )*256-1
160     CHAIN "part2"
```

Le programme ci-dessus va charger le langage machine des RSXs et le placer en mémoire. Rien n'apparaitra sur l'écran à moins que la RAM n'ait un problème. Le programme "part2" serait votre programme personnel (il ne s'intitulera certainement pas "part2", nous avons seulement mis ce titre arbitraire afin de donner un exemple!!!). Le fait de charger le programme en deux parties vous évite d'avoir à recharger le code RSX à chaque lancement du programme.

Le code doit être chargé à l'adresse 10000 avant d'être relogé pour usage. La valeur 16 bits aux adresses 10002 et 10003 est l'endoit où vous désirez que le code soit placé. Une autre location 16 bits à 10004 et 10005 contient la longueur du code qui est placé plus haut en mémoire. Environ 1K du programme n'est utile qu'une seule fois -la partie relocation et celle du test de RAM- et n'est pas envoyé plus haut en mémoire.

Si vous désirez utiliser des graphiques redéfinis, ajoutez les lignes suivantes:

```
10  SYMBOL AFTER 256
```

```
150 SYMBOL AFTER 0
```

La valeur en ligne 150 sera différente suivant le nombre de graphiques redéfinis utilisés.

Il se peut que vous vouliez intégrer dans votre programme un certain nombre de styles de caractères. Après avoir placé la commande SYMBOL AFTER, HIMEM doit être logé juste en-dessous des caractères graphiques.
Il est donc maintenant possible de déplacer des caractères graphiques rapidement avec les commandes |LOADD et |SAVED.

Si vous avez un programme qui définit un jeu de caractères, il est possible grâce à l'extension RAM de sauvegarder et recharger les définitions de telle façon qu'un programme possédera plusieurs jeux de caractères.

```
10  SYMBOL AFTER 0
20  caract=HIMEM+1
30  REM on definit les symboles ici
1000  SAVE "jeu1.grp", B, caract, 2048
```

Ce programme sauvegardera un jeu de caractères (que vous aurez redéfinis) sur disc (ou K7). Note: au lieu de "jeu1" vous pouvez taper un titre de votre choix...!

Dans votre programme final, il se peut que vous désiriez intégrer plusieurs jeux de caractères. Procédez alors selon l'exemple suivant:

```
10  SYMBOL AFTER 0
20  caract=HIMEM+1
30  LOAD "jeu1.grp", caract : |SAVED, 1, caract, 2048, 0
40  LOAD "jeu2.grp", caract : |SAVED, 1, caract, 2048, 2048
50  LOAD "jeu3.grp", caract : |SAVED, 1, caract, 2048, 4096
```

La variable "caract" doit être présente car la valeur de HIMEM change lors de l'accès à la disquette ou K7.

Durant le programme, une sous-routine peut être utilisée pour sélectionner un jeu de caractères:

```
1000  REM avec la variable "jeu", chargeons les caractères
1010  |LOADD, 1, caract, 2048, (jeu-1)*2048
1020  RETURN
```

Notez bien que la variable "jeu" est utilisée. Dans la séquence ci-dessus, les jeux 1 à 3 seront validés. Vous pouvez ajouter ou retrancher selon votre goût.

Il est bien évident que toutes ces validations peuvent être effectuées directement -et une seule fois- à partir du loader! Quand le programme est relancé, il n'y a pas à recharger la bank de RAM.
Les caractères peuvent être retrouvés en faisant:

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

Ce qui aura pour effet d'enlever tout tampon de disc qui aurait pû être validé et 'caract' vous dirigera vers les caractères.

9 PEEK et POKE

Ce sont deux commandes qui permettent d'examiner et modifier les banks de RAM octet par octet.

```
|POKE, [bank], [adresse de bank], [valeur]
|PEEK, [bank], [adresse de bank], [variable]
```

!POKE fonctionne de façon similaire au POKE connu des Anciens. Vous devez toutefois spécifier un numéro de bank en plus de l'adresse et valeur.
Rappelons que l'adresse de bank est de 0 à 16383.

!PEEK est une commande plutôt qu'une fonction. La bank et l'adresse de bank sont les mêmes qu'avec !POKE. Pour trouver une valeur, vous devez fournir une variable entière (comme avec !ASKRAM).

Par exemple:

```
10 valeur%=0
20 !PEEK, 3, 12345, @valeur%
30 PRINT valeur%
```

Dans cet exemple vous examinerez l'octet 12345 en bank 3. Le caractère @ dit à l'extension RSX où la variable est en mémoire afin que son contenu puisse être modifié à l'octet requis.

Note: les commandes !PEEK et !POKE ne sont pas vraiment des commandes pour le débutant ni pour le curieux touche-à-tout! Elles sont destinées au programmeur chevronné qui en aura compris leur utilité.

Une autre commande pas pour les débutants est **!BANK**

```
!BANK, [numéro de bank]
```

La commande est suivie d'un paramètre. Si celui est omis, il devient automatiquement 0 (zéro).
La bank spécifiée est répertoriée dans les 16K compris entre entre 16K et 32K.
Une bank de 0 indexera un retour de la RAM originale. Les numéros de 1 jusqu'au maximum vont répertorier cette bank dans l'emplacement 16K/32K. Lorsqu'une bank est ainsi répertoriée, l'ordinateur utilisera la mémoire de cette bank au lieu de la mémoire RAM normale.
Toutefois, l'écran proviendra toujours de de la RAM original si la commande !LOW a été tapée.
Avantage: au lieu d'avoir à fixer le haut de la mémoire à 16383, toute la mémoire peut être utilisée pour programmer.
Désavantage: si le programme est stoppé alors que l'écran bas est affiché, l'ordinateur écrira le contenu de l'écran dans le programme BASIC, ce qui créera un désordre digne d'une campagne électorale!

Un conseil: n'utilisez sérieusement les commandes !POKE, !PEEK et !BANK que lorsque vous les aurez bien en main! Sauvegardez fréquemment le programme en cours... Pourquoi? Parce que si vous faites une erreur, vous risquez de tout faire planter... Marrant lorsque l'on a passé tout un dimanche à créer un chef-d'oeuvre homérique!!! Le beau langage qui en résulte fait généralement bien rire les voisins...

10 Programmons sans les RSXs

Il est possible, sans les RSXs, d'adresser directement l'extension RAM. Il est toutefois nécessaire de comprendre le fonctionnement de la carte de la mémoire d'un AMSTRAD CPC.
A partir du langage et du BASIC, le bloc de mémoire originale de 16384 à 32767 ne peut **PAS** être utilisé pour programmer. C'est la raison pour laquelle en BASIC, vous devez

fixer le haut de la mémoire à 16383. Le langage machine peut se promener un peu partout sauf dans ce bloc.

L'extension mémoire est répertoriée en 16 banks dans les adresses 16384 à 32767. Une fois la bank répertoriée, vous pouvez entreprendre toutes les opérations possibles et imaginables avec l'extension RAM. Il n'est pas recommandé (mais faites ce que voulez, nous sommes en démocratie) d'utiliser les banks de RAM avec du langage machine. Pourquoi? Parce que si vous changez de bank, le programme disparait (de toute façon, ça n'arrive qu'aux autres). Néanmoins, il est possible d'écrire des programmes qui "tourneront" dans les banks et même dans les 16K du bloc original 16K. Le changement de bank doit cependant être effectué en dehors de cette mémoire.
En BASIC, il serait très difficile d'utiliser les banks pour d'autres programmes. Ce n'est pas impossible, mais nous laisserons aux surdoués le soin de développer leurs méthodes personnelles.

Les banks sont définies de la manière suivante:

En BASIC: Où 'bank' est le numéro de la bank à répertorier

```
      OUT &7F00, 196+(bank AND 3)+(bank AND 28)*2
```

Note: dans ce cas, les banks commencent à 0 (zéro)
Pour les extensions 64K, les banks sont 0 à 3
" " " 256K, " " " 0 à 15
bank 0-3 = 196-199 ... et ainsi de suite

Pour retourner à la bank originale:

```
      OUT &7F000, 192
```

En LANGAGE MACHINE: Le numéro de bank est dans l'accumulateur

SELECTION:

```
C5        PUSH BC          ;sélectionne bank A (save registres
4F        LD C,A           ;sauf a et drapeaux)
E6,03     AND 3            ;bank AND 3)+
47        LD B,A
79        LD A,C
E6,1C     AND 28           ;(bank AND 28)*2
87        ADD A,A
B0        OR B
F6,C4     OR 196           ; +196
01,00,7F  LD BC,07F00H     ;BC=&7F00
ED,79     OUT (C),A
C1        POP BC
C9        RET
```

Le numéro de bank dans l'accumulateur commence à zéro. Pour réinitialiser à la bank originale:
RESET:

```
C5        PUSH BC          ;réinitialise mémoire originale
01,00,7F  LC BC,07F00H     ;BC=&7F00
3E,C0     LD A,192
ED,79     OUT (C),A
C1        POP BC
C9        RET
```

11 Détails techniques

11.1 Adresse de chargement (Message "LOAD ADDRESS")

Le logiciel gérant est relogeable entre 32768 et le haut de la mémoire. Pourquoi? Parce que pas touche à 16384-32767 (voir chapitre précédent). En-dessous de 16384, les RSX ne fonctionneraient pas; c'est la raison pour laquelle le code est d'abord placé à 10000 pour être ensuite placé plus haut. En appuyant sur <ENTER> après chargement, le code sera automatiquement rechargé à l'adresse disponible la plus haute. Alternativement, si vous désirez réserver de la place pour un autre programme, il suffit de taper une autre adresse plus basse (valable!) de votre choix.

11.2 Transfert sur disquette

Si votre logiciel gérant est livré sur cassette, vous pouvez sans problème le transférer sur disquette car il n'est pas protégé. Ce n'est pas la peine de vous armer de Discology car vous vous emmêlerez les doigts avec le programme des RSXs (essayez, vous verrez bien!).
Suivez plutôt les instructions suivantes:

a/ Disquette formatée dans lecteur
b/ Cassette DK TRONICS dans datacorder
c/ Tapez |TAPE puis <ENTER>
d/ LOAD"BANK"
e/ MEMORY 9999
f/ LOAD"RSX",10000
g/ Tapez |DISC
h/ SAVE"BANK"
i/ SAVE"RSX",B,10000,4000

11.2a Sauvegarde sur cassette

Même marche à suivre que ci-dessus MAIS à g/ au lieu de taper |DISC vous tapez:
SPEED WRITE 1 - ce qui permettra de recharger 2 fois plus vite à l'avenir. Si vous êtes patient, omettez g/

11.2b Sauvegarde disc/disc

Si votre logiciel gérant est fourni sur disquette, celle-ci n'est pas protégée. Cela vous donnera l'occasion de consulter votre manuel AMSTRAD pour découvrir comment faire les copies de disquettes non-protégées.

11.3 Augmenter CP/M 2.2 TPA

Allons-y pour les 63K CP/M 2.2!
Marche à suivre:
Faites une copie de votre disquette CP/M 2.2
Retirez les fichiers demo du jeu d'arcade antédiluvien pour faire de la place.

Si logiciel gérant sur K7:
CLOAD "NEWCPM.COM <ENTER>
CLOAD "OLDCPM.COM <ENTER>

Si logiciel gérant sur disc:
Copiez NEWCPM.COM et OLDCPM.COM sur l'autre disquette

Maintenant tapez:

A>MOVCPM 255 * <ENTER>   un message s'affichera
A>SAVE 34 NEWCPM.SYS <ENTER>

Maintenant, 63K TPA!

Pour l'appeler, vous tapez
A>NEWCPM   et les programmes CPM de 63K fonctionneront.
Pour retourner au système original, vous tapez
A>OLDCPM   et le tour est joué

Note: vous devrez appeler OLDCPM pour utiliser certains utilitaires (comme FORMAT par exemple) qui ne tournent qu'avec le système original.

11.4 Compatibilité avec les programmes commerciaux.

464 > l'extension mémoire est compatible avec les banks ram du 6128. En conséquence, de nombreux programmes conçus pour le 6128 fonctionneront sur le 464. Toutefois, à moins que vous ne possédiez aussi une carte **FD.DOS**, vous ne l'aurez pas transformé en 6128.
Les extensions RSX fonctionneront sur un 6128 où l'extension 256K donnera accès à 320 K de bank RAM.
Le logiciel gérant ne pourra accéder, dans son état actuel "que" 256K ou 16 banks de mémoire.
Si vous rajoutez de la mémoire (256K silicon disc DK TRONICS), il est facile de "dire" au logiciel d'accéder 512K (32 banks!) en faisant un petit POKE 1 à l'adresse 10006. (voir chapitre 8 en ce qui concerne le chargement direct des RSX)
Vous insérez alors la ligne suivante:

```
55 POKE 10006,1
```

et vous aurez 32 banks pour 512K

Si un programme commercial refuse de fonctionner sur votre 464 - 664 muni d'une extension RAM DK TRONICS, essayez les remèdes suivants:

1/ Le programme peut utiliser le nouveau vector à &BD58. Si c'est le cas, essayez de lancer les RSX avant de charger votre programme.
Certains de ces programmes, qui fonctionnent correctement après avoir lancé les RSX sont TASWORD 6128 et TASPELL de chez TASMAN SOFTWARE.

2/ Certains programmes, chargés à partir de Disc, cassette, ou lancés à partir d'une carte d'extension ROM vont renifler l'identité de la ROM en utilisant le call du firmware &B915. S'ils reniflent une ROM 464 ou 664: le plantage! Horreur...
Une autre commande RSX contenue dans le logiciel gérant permet à un 464 ou 664 d'émuler l'identité de la ROM du 6128. Attention! Nous n'avons pas dit émuler le 6128! **Seule la carte FD.DOS (encore une pub' gratuite...) le permet à 100% avec un 464.** Pour appeler cette commande, vous tapez:

```
|EMULATE      <ENTER>
```

Le programme croira que votre ROM est celle d'un 6128 et si votre horoscope est favorable il tournera (peut-être).

3/ Le programme commercial peut faire appel à certains aspects de la ROM 6128 et qui sont absents sur les ROMS 464 - 664. Dans ce cas-là, il vous faudra modifier le programme... ce qui n'est guère réjouissant. Vous écrirez à l'éditeur du programme et lui demanderez ce que vous devez modifier pour que son joli programme protégé à mort tourne sur un 464; si vous avez de la chance, il vous répondra peut-être (poliment).
Sinon, vous l'avez deviné(!!!!), il ne vous reste plus qu'à équiper votre CPC464 d'une carte **FD.DOS**...

12 Messages d'erreur

Lorsque vous utiliserez le logiciel gérant, il arrivera des moments où l'ordinateur ne comprendra pas, ou bien ne pourra pas accomplir ce que vous lui demandez. Le logiciel pourra vous envoyer des messages d'erreur en plus des messages habituels du CPC.
Les erreurs et leurs causes sont les suivantes:

1/ Bad bank command
Vous avez fait une erreur de paramètres ou bien une variable n'est pas là alors qu'elle devrait l'être.

2/ Bank unavailable
Vous appelez une bank qui n'est pas là

3/ Bad bank parameter
Vous avez tapé une grosse bêtise

4/ Bad bank address
Les adresses vont de 0 à 16383. Qu'avez-vous tapé?

5/ Value invalid
L'adresse de bank peut être trop grande pour le bloc de data défini.
ou: le paramètre pour !ASKRAM est autre que 1, 2 ou 3 (m'enfin...)
ou: vous essayez de sauvegarder un bloc de plus de 16K

6/ Bad window définition
Vous avez tapé un n° de fenêtre supérieur à 7 avec !SAVEW ou !LOADW
Relisez les instructions, sacrébleu!

13 Détails du Hardware

Les extensions mémoire DK TRONICS ajoutent un seul bloc (64K) ou quatre blocs (64K x 4 = 256K) de RAM aux 464 / 664 / 6128. Ainsi, lorsqu'une extension 64K est branchée sur un CPC 464 la mémoire totale est de deux blocs soit 128K.
Combien de blocs sur un 6128 avec 256K? sur un 6128 avec 512K? Envoyez vos réponses sur cartes postales érotiques S.V.P.!

Pour une application déterminée avec un système déterminé, calculez le nombre total de blocs 64K. Vous trouverez ainsi quels sont les codes ci-dessous se rapportant à votre système.

Les blocs sont identifiés chacun par un numéro. Le 1 est le bloc original 64K, le 2 est équivalent au second bloc du 6128, et ainsi de suite.
La mémoire est indexée en sous-blocs de 16K.
La combinaison des 4 sous-blocs originaux + sous-blocs additionnels forme la carte de la mémoire. La carte est déterminée par un code 8bits envoyé à &7F00, avec les deux bits hauts à 1.
La description suivante des codes concerne les 6 bits restants D5-D0

Codes de contrôle.

Les bits D2-D0 contrôlent la place des sous-blocs 16K dans la mémoire Z80
Les bits D5-D3 contrôlent la sélection de tout nouveau bloc 64K devant être utilisé.

Bits D2-D0 - Codes de carte de mémoire (16K)

CODE	0	1	2	3	4	5	6	7
Sous-bloc								
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

Les numéros 0, 1, 2, 3 concernent les 4 sous-blocs d'une portion de 64K.
L'astérisque (*) indique que la mémoire provient d'un bloc nouveau, c'est-à-dire bloc 2 ou plus haut, sinon le bloc 1 original est sous-entendu
Résumons: code 0 sélectionne les 64K originaux, code 2 sélectionne un nouveau bloc 64K, les autres codes étant un mélange.

Notes:
1. Code 0 est sélectionné à la mise sous tension.
2. Le circuit VDU lit toujours à partir du bloc original de 64K (bloc 1) quel que soit le code.
3. Si code 3 est utilisé, lecture de &4000 à &7FFF, sur 464-664 le data correct sera envoyé uniquement si la ROM haute est invalidée. C'est une variante de l'opération du 6128 mais n'apporte pas vraiment une différence trop importante.
4. Si code 3 est utilisé, les adresses &4000 à &7FFF ne doivent pas servir à faire tourner des programmes. Elles sont destinées au VDU et accès data.

Bits D5-D3 - Codes de sélection blocs 64K

D5	D4	D3	BLOCK
0	0	0	2 (nouveau sous-bloc venant de bloc 2, comme avec un CPC 6128)
0	0	1	3
0	1	0	4
0	1	1	5

La quantité totale de mémoire déterminera quels codes ci-dessus concernent votre ordinateur.

Notes:
1. Code 0, 0, 0 est sélectionné à la mise sous tension
2. Les bits D5, D4, D3 comptent alors que les blocs sont sélectionnés
3. Si 2 modules d'extension 256K sont reliés à une machine et les options validées correctement, toutes les combinaisons D5-D3 peuvent être utilisées pour donner un maximum de 512K mémoire RAM. Les mémoires de 256K des silicon discs DK TRONICS sont correctement indexées pour permettre aux 256K supplémentaires de donner un total de 512K lorsque ceux-ci sont reliés aux 256K RAM DK TRONICS.

14 Modification du disc CP/M+

Pour qu'un clavier de 464 soit "lu" comme celui d'un 6128.

Certains programmes CP/M+ ne voudront pas tourner correctement sur un 464 à cause de la manière par laquelle un 6128 inspecte son clavier.

Les manipulations et programmes suivants feront croire aux programmes CP/M+ récalcitrants qu'ils tournent en fait sur un 6128!

Marche à suivre à respecter scrupuleusement:

1/ Faites une copie des 2 faces de votre disquette CP/M+

a/ Consultez votre manuel AMSTRAD si vous ne savez pas comment faire une copie intégrale de disquette. Alternativement, utilisez un quelconque utilitaire de piratage de disquette.

b/ N'oubliez pas de copier les 2 faces. Retournez les 2 disquettes, sinon vous copiez la même chose sur les 2 faces (ça n'arrive qu'aux autres)

c/ Rangez la disquette originale avant que votre chien ne l'enterre dans le jardin.

2/ Copiez "BANK" et "RSX" sur la face A de la disquette fraichement copiée.

a/ Pour cela regardez les instructions du chapitre 11 ci-dessus. Alternativement, consultez le manuel AMSTRAD pour apprendre comment copier des fichiers sur une disquette.

3/ Si vous avez 2 lecteurs, coupez la tension et DEBRANCHEZ le 2ème lecteur. Cela évitera de se mélanger les doigts et les discs dans quelques minutes (et évitera les affreux jurons retentissants qui ne sont proférés que par les autres!).
Remettez sous tension.

a/ Disquette de 2a ci-dessus dans lecteur, face A regardant le plafond.

b/ RUN"BANK <ENTER>

c/ Quand on vous demande "LOAD ADDRESS" (ce qui veut dire adresse de chargement dans la langue de Frank Sinatra), appuyez sur <ENTER>

d/ |EMULATE:|CPM <ENTER>

4/ (et pour 5/) Maintenant, vous allez taper bêtement et surtout sans chercher à comprendre les choses ci-dessous. Lorsque l'ordinateur vous demandera de mettre dans lecteur B, il faut comprendre "face B"; lorsqu'il vous dira de mettre dans lecteur A, il faudra comprendre "face A"; vous retournerez alors la disquette. De temps à autre, le lecteur tournera, des caractères et/ou chiffres dérouleront ou se mettront à votre gauche. Ne cherchez pas à comprendre, tout va bien. Si des messages d'erreur surviennent c'est soit parce que vous avez tapé des bêtises, soit que vous n'avez pas tenu compte des instructions ci-dessus!
Vous tapez <ENTER> après chaque ligne, aussi courte soit-elle.
Lorsque l'on vous dit <CONTROL> Z ou <CONTROL> C, vous appuyez simultanément sur la touche CONTROL et sur la touche Z ou C, et ne touchez pas à <ENTER>

```
ED PATCH.ASM
i
ORG 100H
XRA A
STA 0FDEFH
JMP 0000
END
<CONTROL> Z
e

ED PROFILE.SUB
i
PATCH
<CONTROL> Z
e

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

C'est fait. (Ne tapez pas "c'est fait"!)

5/ Il est possible de modifier la disquette CP/M+ pour qu'elle se lance sans avoir d'abord à charger le programme BANK. Il suffit de taper:

```
B:SAVE
B:SID C10CPM3.EMS
S1E0
C9
<CONTROL> C
C10CPM3.EMS
Y
100
6500
```

La disquette se lancera sans avoir à charger BANK.

****

Les marques de nombreux produits ci-dessus sont déposées. Si cela vous incommode, déposez-les ailleurs mais ne vous les faites pas tomber sur les pieds!

****

Bon courage!
****

P.S. Possesseurs de 464s, avez-vous votre carte FD.DOS?