LA MÉMOIRE DU CPC : RAM -ROM

Dans les années 80, dans l'esprit des utilisateurs, la puissance et les possibilités d'un ordinateur se mesuraient en fonction de la capacité de sa mémoire. Plus il y en avait, plus les programmes pouvaient être complexes et évolués. La quantité de mémoire qu'il y avait dans un ordinateur est vite devenue un argument commercial majeur ! Mais derrière ces chiffres aguicheurs se cachaient parfois des réalités moins glorieuses car la quantité de mémoire n'est pas tout. Il faut tenir compte de la manière dont l'ordinateur l'exploite.

Ainsi, quelle ne fut pas la déception des premiers utilisateurs d'Amstrad CPC 6128 en s'apercevant que les monstrueux 64 Ko supplémentaires de mémoire vive n'étaient pas utilisés par le Basic Locomotive. Les possesseurs de MSX 64 Ko et MSX2 n'avaient que 24 kilo-octets de mémoire vive pour leurs programmes Basic, pas plus qu'un MSX 32 Ko, ô rage !
Encore plus fort, l'Exelvision EXL 100, ordinateur français, était vendu avec 32 Ko de mémoire partagée entre le Basic et la mémoire vidéo. On ne pouvait pas avoir en même temps un programme Basic complexe et un affichage en mode haute résolution...

Bref, même pour un profane, savoir comment la mémoire est gérée par sa machine est utile, voire indispensable si on veut l'utiliser à bon escient. Comme cette série d'articles est dédiée à l'Amstrad CPC, nous allons faire le point sur les différents types de mémoire qu'on rencontre dessus. Par chance, la gestion de la mémoire sur notre ordinateur fétiche est bien conçue et simple d'accès.

La mémoire du CPC est composée d'une part des ROMs et d'autre part des blocs de RAMs.

Mémoire ROM : pour Read Only Memory (Mémoire morte en français). Mémoire pouvant être lue, mais ne permettant pas l'écriture. Les données présentes dans les ROMs ne s'effacent pas lorsqu'on éteint le CPC.

Mémoire RAM : pour Random Access Memory (Mémoire vive en français). Mémoire en lecture et écriture. Dès qu'on éteint le CPC, les données présentes dans ce type de mémoire disparaissent, ce qui est plutôt gênant, étant donné que nos programmes saisis par exemple en Basic sont justement dans cette mémoire. C'est pour ça que nous avons besoin de la mémoire de masse.

Mémoire Vidéo : une zone de mémoire utilisée par la puce électronique chargée de l'affichage de ce que vous voyez sur l'écran. Elle peut être partagée avec la RAM standard (ce qui est le cas pour l'Amstrad CPC), ou accessible uniquement par la puce vidéo.

Mémoire de Masse : tout support permettant de sauvegarder des données pour pouvoir ensuite les recharger en mémoire. Sur Amstrad CPC, principalement les cassettes audio et les disquettes 3 pouces.

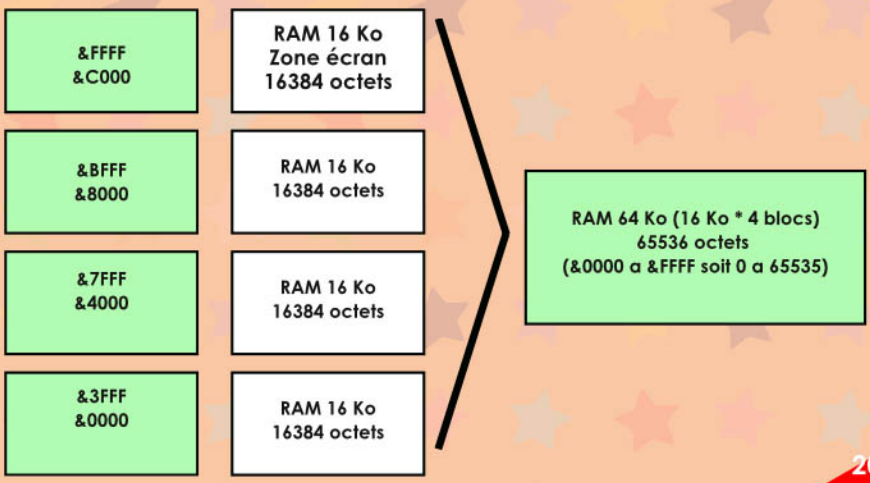
Lorsqu'on débute sur Amstrad CPC 464, on a bien du mal à visualiser l'architecture des mémoires. Le manuel de l'utilisateur est assez obscur de prime abord et trop simpliste concernant ce sujet.

Un programmeur en BASIC peut réaliser des prouesses sans pour autant connaître la mémoire du CPC. Le Z80, épaulé par le *Gate Array*, gérent automatiquement la mémoire allouée pour nos variables, notre programme, la zone écran ou encore les échanges entre RAM et ROM.

Le souci commencera pour notre programmeur en herbe dès qu'il va vouloir utiliser un mélange entre BASIC, données BINAIRES et/ou ASSEMBLEUR.

Nous allons essayer de démystifier la bête de façon simple en partant d'un Amstrad CPC 464 et en lui ajoutant un DDI-1 (lecteur externe de disquettes 3 pouces) puis une extension mémoire 64 Ko DK'Tronic, histoire de comprendre l'évolution de l'architecture pour la mémoire de notre CPC ou en passant d'une machine à une autre.

La mémoire vive de 64 Ko est composée de 4 blocs de 16Ko. Vous allez imaginer que chaque bloc de 16 Ko correspond à une ruche qui possède 16384 alvéoles (16 Ko * 1024 octets), chaque alvéole pouvant recevoir une valeur comprise entre &00 à &FF (0 à 255) et vous empilez les 4 blocs les uns sur les autres, ce qui nous donnera :



Toute la mémoire vive (RAM) n'est pas disponible pour le BASIC :

&C000 à &FFFF	Zone écran (16 Ko)
&BF00 à &BFFF	Zone réservée par la pile du système. Stockage décroissant a partir de &BFFF
&BE80 à &BEFF	**Zone libre. Non vidée après un reset.**
&BE40 à &BE7F	Zone réservée par l'AMSDOS (si nous avons un lecteur de disquette de branché)
&BDF7 à &BE3F	**Zone libre.**
&BDCD à &BDF6	Les vecteurs d'indirection
&BD3D à &BDCC	Les vecteurs d'appel des routines mathématiques
&BD10 à &BD3C	Interface avec le matériel
&BCA7 à &BD0F	Le gestionnaire sonore
&BC65 à &BCA6	Le gestionnaire cassette
&BBFF à &BC64	Le gestionnaire d'écran
&BBBA à &BBFE	Le gestionnaire graphique
&BB4E à &BBB9	Le gestionnaire de mode texte
&BB00 à &BB4D	Le gestionnaire clavier
&B900 à &BAFF	Vecteurs en haut de mémoire. Gestion de la sélection et de l'état des ROMs...
&AC00 à &B8FF	Emplacement des variables systèmes
&A700 à &ABFF	Variables pour l'AMDOS (si nous avons un lecteur de disquette de branché)
&A67C à &A6FF	Comprend les caractères 240 à 255, pour récupérer cette zone : SYMBOL AFTER 256
&9FFC à &A67B	Comprend les caractères 32 à 239 (207 caractères * 8 octets) SYMBOL AFTER 32

Le tableau de gauche indique les variables pour notre programme BASIC, dépendantes de la hauteur de notre HIMEM, et qui s'empileront vers le bas jusqu'à rencontrer le programme BASIC (à éviter sous peine d'erreur ou de plantage).

Notre programme BASIC, lorsqu'il grossira, prendra les octets au-dessus de &0170, jusqu'à rencontrer nos variables.

&0170	Adresse de début pour la programmation en BASIC.
&0040 à &016F	Réservé au BASIC pour la conversion des saisies clavier.
&0000 a &003F	Réservée au système pour les Restart du Z80 (Vecteurs en bas de la mémoire).

Comme on peut le constater avec le tableau ci-dessus, la zone mémoire réellement disponible pour les programmes BASIC est variable suivant votre machine, l'activation de la redéfinition d'une partie ou de l'intégralité du jeu de caractères, mais peux aussi dépendre des ROMs supplémentaires présentes. Il n'y a donc pas une mais plusieurs possibilités et charge à vous de vous y adapter !

Connaître la taille disponible pour un programme BASIC :

La commande **PRINT FRE(0)** permet justement de connaître la taille disponible pour notre programme BASIC. Logiquement, avec un lecteur de disquettes de branché par défaut, nous devons avoir 42249 octets de libre soit &A509 en hexadécimal. Si nous devions enregistrer un programme de cette taille, nous obtiendrions un fichier de 42 Ko.

Le système calcule l'adresse mémoire maximum utilisée par le BASIC, nous pouvons l'obtenir en tapant **PRINT HIMEM**, nous devons logiquement obtenir 42619 octets de libres soit &A67B.

Étant donné que le **BASI**C commence en &0170, nous pouvons vérifier en faisant &A67B - &0170 que nous obtenons &A50B. Tiens, nous avons une différence de 2 petits octets, comment ça se fait ?

En fait, il faut tenir compte de la taille d'un hypothétique programme **BASIC**. Si nous n'avons aucune ligne, les 2 octets en &0170 et &0171 correspondent à la taille de la première ligne BASIC et nous perdons quoi qu'il arrive les 2 octets.

Que nous ayons un CPC avec 64 Ko ou 128 Ko, seuls 42 Ko seront disponibles pour le **BASIC**. Il ne faut pas oublier que notre processeur Z80 ne sait adresser que 64 Ko simultanément.

Mais dans la réalité, le plus gros fichier BASIC réellement chargeable en mémoire et pouvant fonctionner ne pourra pas dépasser les 38 Ko (Exemple : "Le Bagne De Nepharia © 1985 France Logiciel" ou encore "Série Noire © 1985 Micro Application").

Notre HIMEM :

C'est la contraction de ***HI****gh* et ***MEM****ory*, c'est l'adresse mémoire maximum utilisée par le BASIC. Nos variables s'empileront en dessous de cette adresse.

Le MEMORY :

Permet de changer l'adresse du HIMEM.
Par exemple : MEMORY &3FFF
Notre HIMEM sera donc égal à &3FFF, et maintenant nos variables viendront s'empiler à partir et en dessous de cette adresse, permettant de préserver le cas échéant des routines en assembleur, des données en binaire ou des graphismes stockés à partir de &4000.

Attention, le fait de changer de MEMORY plusieurs fois dans un programme peut engendrer des soucis d'incompatibilité entre les CPC (le CPC 464 est assez sensible et retourne assez souvent un MEMORY FULL). Autre petite subtilité, si nous avons beaucoup de variables utilisées par notre programme, le fait de changer de MEMORY peut engendrer des lenteurs car notre *Operating System* (O.S.) va être obligé de migrer les informations en dessous de la nouvelle adresse.
Il existe un cas de figure un peu particulier qui écrasera tout de même vos données malgré le fait d'avoir mis notre MEMORY en &3FFF. Si vous utilisez par exemple la commande SYMBOL AFTER 32, l'**O.S**. va réaliser une copie du jeu de caractères de la ROM en RAM à partir de &9FFC. Donc mieux vaut le savoir et prévoir de ne pas utiliser de données personnelles au-delà de &9FFC sous peine d'être écrasées, à moins, bien entendu, de ne pas utiliser de caractères redéfinis en BASIC.

PEEK ou POKE
Ces deux instructions BASIC permettent respectivement de lire et d'écrire une valeur dans une des 65536 adresses mémoires (pour reprendre l'exemple de nos ruches, ici, on travaille au niveau d'une alvéole).

Affiche la valeur présente à l'adresse mémoire &BDEE. Retournera la valeur 195 :
PRINT PEEK (&BDEE)

Affiche la valeur présente en &BDEE convertie en HEXA, c'est-à-dire C3 :
PRINT HEX$(PEEK(&BDEE))

Maintenant, nous allons écrire un C9 à l'adresse &BDEE :
POKE &BDEE,&C9

L'adresse mémoire n'a pas été choisie au hasard, ni la valeur à « poker ».

Quelques explications :
En assembleur, le &C9 correspond à un RET (Retour), ce qui permet aussi d'annuler l'exécution d'une routine.
En ce qui concerne le &C3,xx,yy ça correspond a un saut en mémoire (JUMP yyxx).

Autre petite chose intéressante, l'adresse &BDEE teste la touche ESC (BREAK) et c'est un des éléments de ce qu'on appelle les vecteurs d'indirection. Késako ? Les vecteurs d'indirection permettent d'altérer ou d'intercepter des actions prédéfinies du logiciel système (ROM) sans avoir besoin de réécrire les routines.

Notre action sur l'adresse &BDEE à tout simplement court-circuité le CTRL+SHIFT+ESC qui, avant, engendrez un RESET de la machine.

En **ASSEMBLEUR**, voici l'équivalent de nos **PEEK** et **POKE**

```
;PEEK : Charger dans le registre A le contenu présent a l'adresse &BDEE
3A EE BD    LD A,(&BDEE)

;POKE
3E C9       LD A,&C9        ;Charger dans le registre A la valeur &C9
32 EE BD    LD (&BDEE),A    ;Ecrire le contenu du registre A à l'adresse &BDEE
```

Pour les plus anciens d'entre vous, les chaînes HEXA vous rappelerons surement des souvenirs. Vous pouviez les retrouver dans des revues comme Joystick Hebdo, *Amstrad Cent Pour Cent, Amstrad & CPC*... Les *pokes* et chaînes hexa étaient la base pour pouvoir bidouiller et tricher dans les jeux.

Les Ordinateurs Amstrad CPC

Voici la configuration de base au niveau de la RAM et des ROMs pour les ordinateurs de gamme CPC ancienne génération (nous ne parlerons pas des machines CPC 464 Plus, CPC 6128 Plus et console GX-4000).

Vous pourriez trouver bizarre le fait que le KC COMPACT ait été ajouté au même niveau que le CPC 6128. Même si le KC Compact est un clone du CPC 6128, qui n'est pas entièrement fidèle, au niveau de la RAM et de la ROM, les informations dans le tableau ci-dessous seront tout de même correctes.

	CPC 464 CPC 472 (1)	CPC 664	CPC 6128 KC COMPACT
RAM principale	64 Ko		
RAM étendue	non		64 Ko
Total RAM	64 Ko		128 Ko

ROM inférieure O.S.	16 Ko	
ROM slot 0 - BASIC	BASIC 1.0 16 Ko	BASIC 1.1 16 Ko
ROM slot 7 - AMSDOS	Non (Lecteur de cassette)	16 Ko (Lecteur de disquettes 3 pouces)
Total ROM	32 Ko	48 Ko

HIMEM	**43903 (&AB7F)**	**42619 (&A67B)**

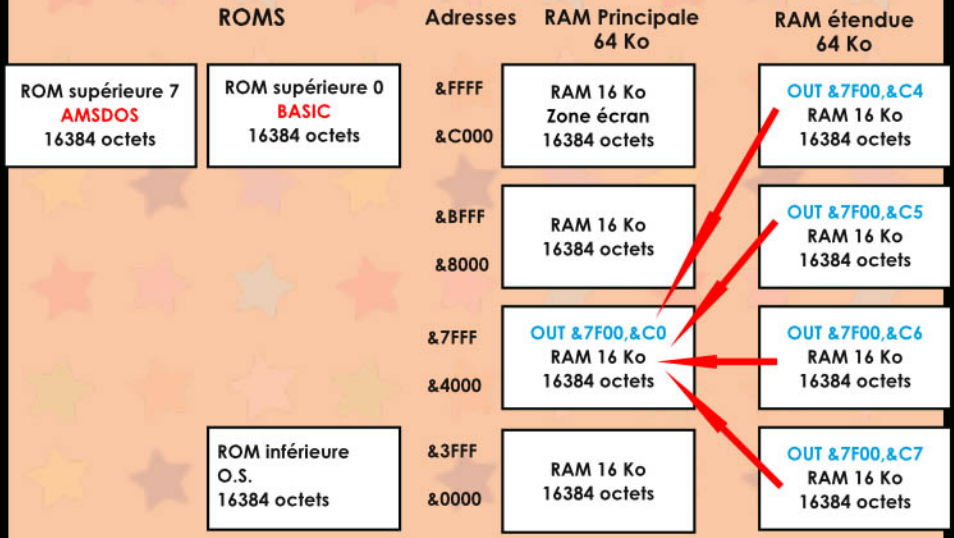
(1) Le CPC 472, comme son nom l'indique, possédait 72 Ko de RAM (plus 8 Ko par rapport au CPC 464) pour la raison suivante : à l'époque, le gouvernement espagnol avait fait passer une loi instaurant une taxe sur les machines ayant jusqu'à 64 Ko de RAM et pas de clavier localisé en espagnol. Rajouter 8 Ko sur un 464 QWERTY anglais permettait d'éviter de payer la taxe. Cette machine reste identique à un CPC 464 dans son fonctionnement lorsqu'il a un clavier espagnol. En revanche, si vous avez un CPC 472 avec un clavier anglais, le BASIC sera du 1.1 car c'est la ROM du CPC 664 qui sera à l'intérieur. Eh oui, une machine, deux cas de figure. À noter que les 8 Ko ne sont pas utilisables.

ARCHITECTURE MEMOIRE ROM/RAM :

Maintenant, voici l'architecture des différents blocs de 16 Ko concernant les RAMs et les ROMs pour un ordinateur CPC 6128 (pour un CPC 664, c'est la même chose, à part que la colonne "RAM étendue" disparaît dans la configuration de base, sauf si vous avez une extension mémoire de 64 Ko de branchée).

On s'aperçoit que pour une même zone mémoire, il existe plusieurs blocs et comme le Z80 n'est capable de gérer simultanément que 64 Ko de données, comment est-ce qu'il fait pour arriver à gérer autant de mémoire ?

La puissance de l'Amstrad CPC se situe au niveau du *Bank switching* ou *commutation de banque mémoire* permettant de libérer le maximum de mémoire pour l'utilisateur, il est capable d'échanger très rapidement un bloc mémoire par un autre. En fait, lorsqu'on utilise un programme BASIC, nous ne le voyons pas mais le Z80 échange le bloc ROM inférieur de l'*O.S.* avec notre RAM **(ça vient en superposition et les données en RAM ne sont plus accessibles le temps qu'il puisse réaliser ce dont il a besoin, puis il restaure la RAM)**, pour l'accès à la ROM BASIC ou à la ROM AMSDOS, c'est la même chose avec la zone écran. Vous aurez beau fixer votre écran, vous ne vous rendrez compte de rien.



En ce qui concerne, la RAM Etendue (nos 64 Ko supplémentaires sur CPC 6128 divisés en 4 *banks* de 16 Ko), grâce au *Bank switching* et toujours grâce à notre brave *Gate Array* nous pouvons très facilement remplacer la zone mémoire comprise entre &4000 et &7FFF par l'une des 4 banks de la RAM étendue. Un petit OUT &7F00,&C4 nous permettra d'avoir accès aux données de la première *bank* de la RAM étendue et un OUT &7F00,&C0 permettra de revenir à la zone mémoire principale.

Une particularité bien pratique de la RAM étendue est la persistance de son contenu même après un *RESET* de la machine alors que la RAM principale est réinitialisée par l'OS systématiquement. Les *crackers* et les pirates en herbe ont très vite compris l'utilité de créer des routines qui copient le contenu du programme dans les bancs mémoire pour pouvoir récupérer les données très facilement après le *reset* du programme.

Exemple d'affichage de 5 images et de l'utilisation des *banks* :

Cette technique est très facile a utiliser. Le principe consistant à charger les 4 images dans nos *banks* et la dernière dans la mémoire principale. Après, il suffit de se connecter à la bonne *bank* et de réaliser un transfert à l'écran. À noter que si l'on utilise des images compressées, nous pourrions en afficher beaucoup plus, mais ceci est une autre histoire.

La routine ci-dessous reste simpliste et mériterait d'être optimisée (*Listing PICTBANK.BAS*) :

```
10 MODE 1:CALL &BC02:PEN 1
20 'Baisser la HIMEM en &3FFF
30 MEMORY &3FFF
40 'Test 64K ou 128K
50 OUT &7F00,&C0:POKE &4000,0
60 OUT &7F00,&C4:POKE &4000,1
70 OUT &7F00,&C0:a=PEEK(&4000)
80 IF A=0 THEN 90 ELSE PRINT "128 Ko only":END
90 'chargement 5 * 16 Ko (80 Ko)
100 RESTORE 340
110 FOR i=1 TO 5:READ bank,file$
120 LOCATE 1,1:PRINT "BANK &";HEX$(bank)
130 OUT &7F00,bank
140 LOAD file$,&4000
150 NEXT
160 'Routine ASM transfert vers écran
170 RESTORE 350
180 FOR i=&BE80 TO &BE8B:READ a$:POKE i,VAL("&"+a$):NEXT
190 'Se connecter à la bonne bank et traitement pour affichage
200 OUT &7F00,&C4:GOSUB 260
210 OUT &7F00,&C5:GOSUB 260
220 OUT &7F00,&C6:GOSUB 260
230 OUT &7F00,&C7:GOSUB 260
240 OUT &7F00,&C0:GOSUB 260
250 GOTO 200
260 'Masquer les encres
270 FOR i=0 TO 15:INK i,0:NEXT
280 'Transfert vers écran
290 CALL &BE80
300 'mode, border et couleurs
310 CALL &FFD0
320 'Temporisation
330 FOR A=1 TO 60:CALL &BD19:NEXT:RETURN
340 DATA &C4,"1.ecr",&C5,"2.ecr",&C6,"3.ecr",&C7,"4.ecr",&C0,"5.ecr"
350 DATA 21,FF,7F,11,FF,FF,01,00,40,ED,B8,C9
```

Ce petit bout de code tout simple (ligne 40 à 80), permet de tester si nous avons une machine avec 64 Ko ou plus. Le principe consiste à se positionner en mémoire principale et à écrire en &4000 la valeur 0.

Ensuite, en nous connectant dans la RAM étendue n° 1 (&C4), nous pokons l'adresse &4000 en écrivant la valeur 1.

Nous revenons en mémoire principale (si vous n'avez que 64 Ko, vous y êtes déjà et ça ne changera donc rien du tout), et on charge dans la variable "a" la valeur écrite en &4000.
Si vous avez une machine avec 64 Ko vous aurez comme valeur 1 et si vous avez 128 Ko, vous aurez comme valeur 0.

Si vous avez du mal à visualiser ce qui se passe réellement, le tableau ci-dessous doit vous permettre de mieux comprendre ce qui est réellement modifié suivant la quantité de RAM que vous avez.

128 Ko de RAM ou plus	**RAM PRINCIPALE**	**RAM ÉTENDUE**
OUT &7F00,C0:POKE &4000,0	&4000 = 0	Existante
OUT &7F00,&C4:POKE &4000,1	&4000 = 0	&4000 = 1

64 Ko de RAM	**RAM PRINCIPALE**	**RAM ÉTENDUE**
OUT &7F00,&C0:POKE &4000,0	&4000 = 0	Inexistante
OUT &7F00,&C4:POKE &4000,1	&4000 = 1	Inexistante

Histoire de changer, pour transférer les données, j'ai utilisé un LDDR :

La différence par rapport au LDIR classique, c'est que nous partons de la zone mémoire la plus haute et que nous descendons. La routine, chargez l'adresse pointée par le registre DE avec le contenu de l'adresse pointée par le registre HL, puis décrémente les registres DE, HL et BC. L'opération est répétée jusqu'à ce que le registre BC soit égal à zéro.

```
                    ORG &BE80
#BE80               ; Copie de &7FFF vers &FFFF 16 Ko en descendant
#BE80  21 FF 7F     LD HL,&7FFF     ;adresse source
#BE83  11 FF FF     LD DE,&FFFF     ;adresse de destination
#BE86  01 00 40     LD BC,&4000     ;longueur à copier(16ko)
#BE89  ED B8        LDDR            ;transfert des données
#BE8B  C9           RET             ;Retour
```

Les plus attentifs d'entre vous auront sûrement remarqué que pour changer le mode écran, la couleur de la bordure ainsi que les couleurs de chacune des images, j'utilise un petit CALL &FFD0.

Qu'est-ce qui se cache derrière ? Le petit utilitaire très simple en Basic, ci-dessous, permet d'insérer une routine en assembleur dans une image créée sous OCP (.SCR) qui change le mode graphique sans effacement de l'écran, change également la couleur de la bordure ainsi que les 16 couleurs. Tout ceci est écrit dans une zone de 48 octets non visible (&FFD0 à &FFFF).

Les informations sont décodées à partir du fichier (.PAL) qui est généré par ConvImgCpc ou le programme de dessin OCP Advanced Art Studio.

Ensuite, l'affichage se fait très facilement par un LOAD "IMAGE.ECR",&C000:CALL &FFD0

En résumé, le programme va prendre votre FICHIER.SCR, votre FICHIER.PAL et il va générer le FICHIER.ECR qui aura les données incrustées à l'intérieur, magique !!!

L'incrustateur de couleurs (*Listing PICTCONV.BAS*) :

```
1 'Kukulcan 2016
2 'Incrustation Couleurs CPC - écrire
10 OPENOUT "D":MEMORY &8800:CLOSEOUT
20 GOSUB 280
30 MODE 1:BORDER 0:INK 0,0:INK 1,26:PEN 1
40 INPUT "Nom du fichier (.SCR)",file$
50 IF file$="" THEN a$="*.SCR":|DIR,@a$:GOTO 40
60 CLS
70 LOAD file$+".PAL",&8809
80 LOAD file$+".SCR",&C000
90 REM *** ÉCRIRE ASM DANS IMAGE ***
100 GOSUB 370
110 REM *** MODE ***
120 a=PEEK(&8809)
130 POKE &FFD1,a
140 REM *** BORDER ***
150 a=PEEK(&880C)
160 POKE &FFD6,couleur(a)
170 POKE &FFD7,couleur(a)
180 REM *** LES 16 COULEURS ***
190 FOR i=0 TO 15
200 a=PEEK(&880C+(i*12))
210 POKE &FFF0+i,couleur(a)
220 NEXT
230 CALL &FFD0:'Changer Mode, Border, Couleurs
240 SAVE file$+".ECR",B,&C000,&4000
250 LOCATE 1,1:PRINT "SAUVEGARDE FINIE"
260 PRINT CHR$(7):CALL &BB06:GOTO 30
270 END
280 '*** ConvImgCpc - DÉCODAGE DES ENCRES DU PAL POUR CPC ***
290 DIM couleur(128)
300 RESTORE 360
310 FOR i=0 TO 26
320 READ a
330 couleur(a)=i
340 NEXT
350 RETURN
360DATA 84,68,85,92,88,93,76,69,77,86,70,87,94,64,95,78,71,79,82,66,83,90,89,91,74,67,75
370 '*** Routine ASM ***
380 RESTORE 400
390 FOR i=&FFD0 TO &FFEE:READ a$:POKE i,VAL("&"+a$):NEXT:RETURN
400 DATA 3E,00,CD,1C,BD
410 DATA 01,00,00,CD,38,BC
420 DATA 21,F0,FF,AF,F5,E5,46,48
430 DATA CD,32,BC,E1,23,F1,3C,FE,10,20,F1,C9
```

Les explications concernant la routine assembleur :

```
                ORG &FFD0
#FFD0  3E 00    LD A,0          ;choix du mode graphique (0 à 3)
#FFD2  CD 1C BD CALL &BD1C      ;changer le mode graphique sans effacement
#FFD5  01 00 00 LD BC,&0000     ;b = couleur 1 et C = couleur 2
#FFD8  CD 38 BC CALL &BC38      ;équivalent à BORDER B,C
#FFDB  21 F0 FF LD HL,&FFF0     ;adresse mémoire des 16 couleurs
#FFDE  AF       XOR A           ;registre A = 0
boucle
#FFEF  F5       PUSH AF         ;sauver les registres A et F
#FFE0  E5       PUSH HL         ;sauver les registres H et L
#FFE1  46       LD B,(HL)       ;équivalent a B=PEEK(HL)
#FFE2  48       LD C,B          ;C=B (pour obtenir 1ère et 2ème couleurs
                                 identiques)
#FFE3  CD 32 BC CALL &BC32      ;équivalent à INK A,B,C
#FFE6  E1       POP HL          ;restaurer les registres H et L
#FFE7  23       INC HL          ;HL = HL + 1 (adresse suivante)
#FFE8  F1       POP AF          ;restaurer les registres A et F
#FFE9  3C       INC A           ;A = A + 1
#FFEA  FE 10    CP &10          ;Est-ce que A = 16 ?
#FFEC  20 F1    NZ,boucle       ;NON alors on retourne a boucle
#FFEE  C9       RET             ;SINON fin de la sous-routine = Retour
```

Le code utilise 31 octets, et les données pour les couleurs, 16 octets, nous arrivons donc à 47 octets d'utilisés dans cette zone de 48 octets. Si nous avions besoin de plus de place, on aurait utilisé une autre zone non visible dans un écran dit « standard ». Du reste, en parlant de ça, voici les fameuses 8 zones de 48 octets qui n'influencent pas l'affichage de l'image (ce qui veut dire que vous avez le droit à 384 octets utilisables pour cacher du code, des données, un message...).

```
&C7D0 à &C7FF ; &CFD0 à &CFFF ; &D7D0 à &D7FF ; &DFD0 à &DFFF
&E7D0 à &E7FF ; &EFD0 à &EFFF ; &F7D0 à &F7FF ; &FFD0 à &FFFF
```

Commercialement parlant, la société Titus a utilisé la technique d'écrire les couleurs dans l'image et ensuite de les charger via un loader en Basic. Du reste, si ma mémoire est bonne, c'est justement grâce à eux que j'ai pu découvrir la première zone de 48 octets.

Fichier basic déprotégé concernant le *loader* du jeu « Fire & Forget © 1988 Titus » : *FIRE.BAS*

```
5 MODE 1:BORDER 0:INK 0,0:INK 1,26:PAPER 0:PEN 1:CLS
10 OPENOUT"q":MEMORY &F00:CLOSEOUT
30 LOCATE 2,12 :PRINT"Now press any key to skip presentation"
40 cp=600
50 a$=INKEY$:IF a$<>"" THEN 100
60 cp=cp-1:IF cp<>0 THEN 50
70 MODE 0:FOR r=0 TO 15:INK r,0:NEXT:LOAD"screen",&C000:LOAD"sound",&1000:POKE
&10B5,&C:POKE &10C2,&A:POKE &105F,&FF
80 FOR r=0 TO 15:INK r,PEEK (65500+r):NEXT:FOR r=0 TO 50:NEXT:CALL &1000
100 LOAD"prog.bin",&F80:CALL &1000
```

Les 3 morceaux qui nous intéressent sont surlignés.
La partie en jaune correspond au passage de toutes les encres dans la couleur noire.
Ensuite, la partie surlignée en bleu correspond au chargement de l'image dans la zone écran, mais l'utilisateur ne voit rien du tout.

Et lorsque nous arrivons à la partie surlignée en vert, la boucle permet de charger les 16 couleurs en lisant les données inscrites dans l'image de &FFDC à &FFEB (65500 à 65515).

Pour revenir à mon petit programme, voici ce qui permettra d'afficher les images (*Listing PICTAFF.BAS*) :

```
1 'Kukulcan 2016
2 'Incrustation Couleurs CPC - lecture
10 MODE 1:BORDER 0:INK 0,0:INK 1,26:PEN 1
20 INPUT "Nom du fichier (.ECR)",file$
30 IF file$="" THEN a$="*.ECR": |DIR,@a$:GOTO 20
40 LOAD file$+".ECR",&C000
50 CALL &FFD0
60 PRINT CHR$(7):CALL &BB06:GOTO 10
```

En cadeau, une conversion d'image d'un jeu d'arcade au format CPC :

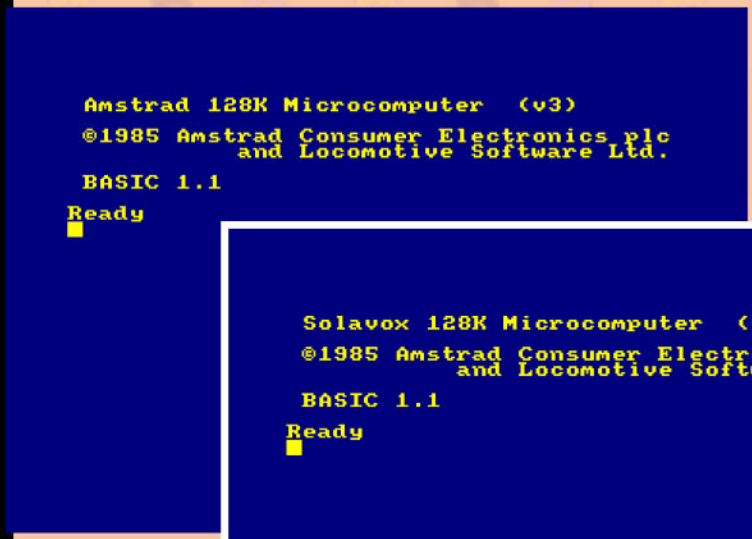
En ce qui concerne la RSX |DIR, vous aurez le droit à quelques explications dans le paragraphe nommé « ROM supérieure slot 7 – L'AMSDOS ».

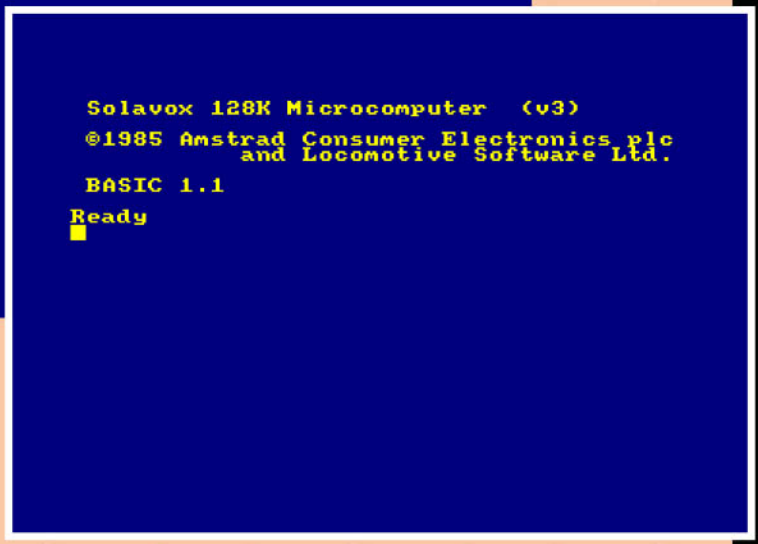
ROM inférieure – L'O.S. (Operating System)

Cette ROM de 16 Ko contient les différents gestionnaires (clavier, mode texte, graphique, écran, cassette, son...), les routines mathématiques ainsi que le générateur de caractères.

Du reste, la ROM possède dans ses 64 (&40) premiers octets les mêmes informations qu'en RAM, étant donné que ce sont des informations réservées au système pour les Restart du Z80, si elles n'y était pas, notre CPC planterait tout simplement.
Au démarrage, sur notre fond bleu, nous avons comme nom de machine : AMSTRAD

Mais les cartes mères ont été prévues pour être paramétrées pour différentes marques très facilement, grâce à 3 *jumpers* (ce qui nous donne 8 possibilités, &000 à &111). Nous pouvons alors obtenir : Amstrad, Orion, Schneider, Awa, Solavox, Saisho, Triumph et Isp.





Autre petite chose intéressante, juste avant les 8 noms dans la ROM, nous trouvons le nom de code de lancement du projet Amstrad CPC : ARNOLD.

Le jeu de caractère ASCII du CPC est stocké dans la ROM et il est composé de 256 caractères : 0 à 255.

En temps normal, nous ne pouvons pas voir les caractères de contrôle (0 a 31), mais grâce au petit programme que nous allons vous fournir ci-dessous, il vous sera possible de les voir. Et ô miracle, pour ceux qui ont déjà obtenu des caractères bizarres à l'écran comme l'espèce de petit sablier (CTRL +X

qui correspond au code 24) ou encore le rond avec un point dedans (CTRL + O qui correspond au code 15), vous pourrez maintenant tous les visualiser. Les explications concernant les caractères ASCII seront abordées dans un futur numéro.

Dans l'écran à gauche, nous sommes en présence d'un CPC 464 avec clavier QWERTY, et à droite, d'un CPC 6128 avec clavier AZERTY. On constate la différence dans le jeu de caractères et dans l'adresse de début concernant les noms.

En fait, pour arriver à les visualiser, il suffit de savoir que les caractères, quel que soit le mode graphique, sont toujours stockés dans une matrice de 8x8 pixels. Chaque ligne est stockée dans un octet (vous vous rappelez, un octet = 8 bits. En conséquence, les 64 points correspondant à notre caractère ne nécessitent que 8 octets (8 octets * 8 bits, ça nous donne bien 64 bits).

	bits							
ligne	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	1	0	1	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	1	0	0	0	1	0
7	0	0	0	1	1	1	0	0
8	0	0	0	0	0	0	0	0

La syntaxe pour redéfinir un caractère :
SYMBOL numéro_du_caractère_à_redéfinir, ligne1 , ligne2, ligne3 , ligne4, ligne5, ligne6, ligne7, ligne8

Imaginons que nous voulions redéfinir nos espaces (caractère 32) avec le dessin présent dans le tableau au-dessus, nous devrions écrire :

```
SYMBOL 32,&X00011100,&X0,&X00010100,&X0,&X00001000,&X00100010,&X00011100,&X0
```

Le seul souci, c'est que tous nos espaces sont maintenant remplacé par ce symbole.

Si l'on souhaite annuler tous les symboles redéfini, il suffit de faire **CALL &BB4E**

L'affichage des caractères sera très lent, mais le listing est surtout là pour montrer comment ça fonctionne (*Listing LOWER.BAS*) :

```
1 'Kukulcan 2016
10 MODE 1:BORDER 0:INK 0,0:INK 1,26:INK 2,6:PEN 1
20 MEMORY &3FFF
30 DATA F3,CD,06,B9,21,00,00,11,00,40,01,00,40,ED,B0,CD,09,B9,FB,C9
40 RESTORE 30:FOR i=&A400 TO &A413:READ A$:POKE i,VAL("&"+a$):NEXT
50 REM *** TRANFERT ROM INFÉRIEURE
60 REM *** EN RAM À PARTIR DE &4000
70 CALL &A400
80 REM *** AFFICHAGE DES NOMS DE MACHINE
90 FOR i=&4700 TO &4800
100 IF PEEK(i)=ASC("A") AND PEEK(i+1)=ASC("r") AND PEEK(i+2)=ASC("n") AND PEEK(i+3)=ASC("o")
THEN adr=i:i=&4800
110 NEXT i
120 PRINT "DEBUT DES NOMS EN ROM : &";HEX$(ADR-&4000)
130 PRINT:PRINT "  NOM HISTORIQUE : ";:GOSUB 260
140 PRINT:PRINT "NOMS COMMERCIAUX :"
150 FOR nom=0 TO 7
160 PRINT SPC(11);"&X";BIN$(nom,3);" = ";
170 GOSUB 260
180 NEXT nom
190 REM *** LA TABLE DES CARACTÈRES ***
200 ADR=&7800
210 FOR base=460 TO 632 STEP 16
220 GOSUB 350
230 NEXT base
240 CALL &BB06
250 END
260 'lecture NOM en RAM
270 a=-1
280 WHILE a<>0
290 a=PEEK(adr)
300 IF a<>0 AND a<>10 AND a<>32 THEN PRINT CHR$(a);:'exclure 0, 10 (retour), 32 (espace)
310 adr=adr+1
320 WEND
330 PRINT
340 RETURN
350 REM DESSINE LES CARACTÈRES
360 c=0:couleur=1
370 FOR ligne=398 TO 16 STEP -2
380 FOR colonne=1 TO 8
```

```
390 a=PEEK(adr):'lire la valeur en mémoire
400 huitpoint$=BIN$(a,8):'convertir la valeur en binaire 8 bits dans une chaîne
410 PLOT base+(colonne*2),ligne,VAL(MID$(huitpoint$,colonne,1))*couleur:'dessine le point
420 NEXT colonne
430 adr=adr+1
440 c=c+1
450 IF c=8 THEN c=0:GOSUB 480
460 NEXT ligne
470 RETURN
480 REM changer la couleur
490 IF couleur=1 THEN couleur=2 ELSE couleur=1
500 RETURN
```

ROM supérieure slot 0 - LE BASIC -

BASIC est l'acronyme de *Beginner's All-purpose Symbolic Instruction Code*, qui pourrait être traduit par Code d'instruction symbolique multi-usages pour débutant. Ce langage, créé en 1964 à l'université de Dartmouth College aux états-unis par John George KEMENY et Thomas Eugène KURTZ, n'a cessé d'évoluer pour permettre aux personnes qui ne sont pas des scientifiques ou n'ayant aucune formation technique de pouvoir utiliser un ordinateur le plus facilement possible sans avoir besoin de connaître le matériel.

Comme vous le savez sûrement, nous avons du Locomotive BASIC 1.0 pour les ordinateurs CPC 464 et CPC 472 et du Locomotive BASIC 1.1 pour les ordinateurs CPC 664, CPC 6128, KC Compact, CPC 464 Plus et CPC 6128 Plus.
Mais il est possible de connaître le numéro de *release* pour chaque version du BASIC en se connectant à la ROM supérieure et en la transférant dans la mémoire principale (en copiant le contenu de la ROM vers RAM).

Le numéro de *release* est stocké dans la ROM aux adresses suivantes &C001, &C002 et &C003.

Nous pouvons aussi en profiter pour afficher si c'est du BASIC 1.0 (information stocké à partir de &C040) ou du BASIC 1.1 (information stockée à partir de &C034).

Bien entendu, comme nous allons transférer les données de &C000 en &4000, les adresses sont à adapter pour afficher les bonnes informations : &C001 à &C003 devient &4001 à &4003, et &C034 deviendra &4034...

Nous allons écrire un tout petit programme en assembleur pour arriver à nos fins :

```
                        ORG &A400
#A400  F3               DI ;Désactive les interruptions
#A401                   ;sélectionne la ROM supérieure (BASIC)
#A401  CD 00 B9         CALL &B900
#A404                   ;Copie de &C000 vers &4000 16 Ko en montant
#A404  21 00 C0         LD HL,&C000 ;adresse source
#A407  11 00 40         LD DE,&4000 ;adresse de destination
#A40A  01 00 40         LD BC,&4000 ;longueur à copié(16ko)
#A40D  ED B0            LDIR        ;transfert des donnees
#A40F                   ;Coupe la ROM supérieure pour resélectionner la RAM
#A40F  CD 03 B9         CALL &B903
#A412  FB               EI          ;Réactive les interruptions
#A413  C9               RET         ;Retour
```

Nous allons utiliser la routine en langage machine (code HEXA dans des DATA) dans un petit programme en BASIC *(Listing RELEASE.BAS)* :

```
10 MODE 1:BORDER 0:INK 0,0:INK 1,26:PEN 1
20 DATA F3,CD,00,B9,21,00,C0,11,00,40,01,00,40,ED,B0,CD,03,B9,FB,C9
30 RESTORE 20:FOR i=&A400 TO &A413:READ A$:POKE i,VAL("&"+a$):NEXT
40 CALL &A400
50 decal=0:IF PEEK(&4002)=0 THEN decal=12
60 FOR I=&4034+decal TO &403C+decal:PRINT CHR$(PEEK(I));:NEXT
70 ver$ = CHR$(48+PEEK(&4001))+"."+CHR$(48+PEEK(&4002))+CHR$(48+PEEK(&4003))
80 PRINT " release ";ver$
90 PRINT:PRINT "Sauver la ROM (O/N) ?"
100 a$=INKEY$:IF a$="" THEN 100 ELSE a$=UPPER$(a$)
110 IF a$="O" THEN SAVE "BAS"+ver$+".rom",B,&4000,&4000
```

Le programme proposera de sauver Oui ou Non la ROM.

Voici les résultats de nos tests :
BASIC 1.0 *release* 1.00 = CPC 464 Anglais
BASIC 1.0 *release* 1.02 = CPC 464 Danois
BASIC 1.0 *release* 1.03 = CPC 464 Français

BASIC 1.1 *release* 1.10 = CPC 664 Anglais

BASIC 1.1 *release* 1.20 = CPC 6128 Anglais ou KC Compact
BASIC 1.1 *release* 1.21 = CPC 6128 Espagnol
BASIC 1.1 *release* 1.23 = CPC 6128 Français

Bizarrement nous n'avons pas trouvé de *release* 1.30, existerait-il un prototype caché ?

BASIC 1.1 *release* 1.40 = CPC 464+ Anglais ou CPC 6128+ Anglais

ROM supérieure slot 7 - L'AMSDOS

La société anglaise AMSTRAD a été créée par Alan Michael SUGAR en 1968. Le nom de la société est la contraction de **A**lan **M**ichael **S**ugar et **TRAD**ing (Commerce), fallait y penser.

Pour le terme AMSDOS, c'est le même principe, c'est l'acronyme d'**A**lan **M**ichael **S**ugar (surnommé affectueusement Tonton Sususcre en France) et ***D**isk **O**perating **S**ystem*, en conclusion, on pourrait le traduire par Système d'exploitation de disque de la société AMSTRAD, simple mais efficace.

Le lecteur de disquette externe DDI-1 / FD1 (contenant l'AMSDOS) est sorti pour la première fois en fin d'année 1984 pour équiper les ordinateurs Amstrad CPC 464. En 1985, les ordinateurs CPC 664 et CPC 6128 étaient équipés d'un lecteur de disquettes interne. Uniquement deux lecteurs pouvaient être supportés par l'AMSDOS, le lecteur A et le lecteur B.

Le système d'exploitation d'Amstrad pour les lecteurs de disquettes 3 pouces (bien plus rapide d'écrire AMSDOS) donnant la possibilité aux utilisateurs d'accéder à de nouvelles instructions par le biais des RSX :

Nous pouvons les classer en 2 grandes familles :

- Les 11 RSX concernant le lecteur de disquettes

|CPM ***C**ontrol **P**rogram for **M**icrocomputers*.
L'exécution de cette RSX engendre la lecture des données présentes en Piste 0 du secteur &41 pour les charger en mémoire à partir de &100 (jusqu'à &2FF). Il faut savoir que ce secteur est de taille 2 (un secteur de taille normale qui fait 512 octets, &200 en hexa). Une fois le chargement effectué, le CPC va faire un saut en &100 (C3,00,01 ; JP #100). À noter que ça viendra écraser une partie de votre listing BASIC si vous en avez un (le BASIC commençant en &170).

|DISC Sélectionner le lecteur de disquettes et force la lecture et l'écriture sur celui-ci
|DISC.IN Force la lecture sur disquette
|DISC.OUT Force l'écriture sur disquette
|A Sélectionner le lecteur A ou POKE &A700,0
|B Sélectionner le lecteur B ou POKE &A700,1
|DRIVE Sélectionner le lecteur de disquette et se positionner sur le lecteur A ou B.

```
Exemple : lecteur$="A":|DRIVE,lecteur$
```

|USER Permet de changer le *user* qui est par défaut sur le zéro. Cette RSX ne permet d'accéder qu'aux user 0 à 15, mais il est possible grâce à un *poke* d'accéder aux 256 possibilités. À noter que les fichiers effacés sont enregistré dans le *user* 229. POKE &A701,229:CAT

|DIR Permet d'afficher le *Directory de l'user* courant avec la possibilité d'ajouter un filtre.

```
Exemple : fichier$="*.rom":|DIR,@fichier$
```

|ERA Permet d'effacer un ou plusieurs fichiers.

```
Exemple : fichier$="*.bak":|ERA,@fichier$
```

|REN Permet de renommer un fichier

```
Exemple : avant$="ANCIEN.BAS":apres$="NOUVEAU.BAS":|REN,@apres$,@avant$
```

- Les 3 RSX concernant le lecteur de cassettes

|TAPE Force la lecture et l'écriture sur le lecteur de cassettes
|TAPE.IN Force la lecture sur cassette
|TAPE.OUT Force l'écriture sur cassette

Le petit programme en assembleur ci-dessous est une évolution du listing précédent. Le gros avantage de ce dernier, consiste dans le fait qu'il permet de sélectionner une ROM supérieure (0 à 255) puis de copier le contenu en RAM de &4000 a &7FFF :

```
                        ORG &A400
#A400                   ;numéro du SLOT dans le registre C
#A400  DD 4E 00         LD C,(IX+&00)
#A403                   ;Selectionner la ROM supérieure présente dans le SLOT C
#A403  CD 0F B9         CALL &B90F
#A406                   ;écrire dans la PILE le registre BC
#A406  C5               PUSH BC
#A407  F3               DI ;Désactive les interruptions
#A408                   ;Copie de &C000 vers &4000 avec une longueur de 16 Ko
#A408  21 00 C0         LD HL,&C000 ;adresse source
#A40B  11 00 40         LD DE,&4000 ;adresse de destination
#A40E  01 00 40         LD BC,&4000 ;longueur à copier(16 Ko)
#A411  ED B0            LDIR        ;transfert des donnees
#A413                   ;lire dans la PILE et charge dans le BC
#A413  C1               POP BC
#A414                   ;Relectionne la ROM supérieure précedemment sélectionnée
#A414  CD 18 B9         CALL &B918
#A417  FB               EI          ;Réactive les interruptions
#A418  C9               RET         ;Retour
```

Nous allons utiliser la routine en langage machine (code HEXA dans des DATA) dans un petit programme en BASIC (*Listing ROMUPPER.BAS*) :

```
10 MODE 1:BORDER 0:INK 0,0:INK 1,26:PEN 1
20 DATA DD,4E,00,CD,0F,B9,C5,F3,21,00,C0,11,00,40,01,00,40
30 DATA ED,B0,C1,CD,18,B9,FB,C9
40 RESTORE 20:FOR i=&A400 TO &A418:READ A$:POKE i,VAL("&"+a$):NEXT
50 LOCATE 1,1:INPUT "ROM SUPERIEURE, NUMERO DU SLOT";slot
60 IF slot<0 OR slot>255 THEN GOTO 50
70 CALL &A400,slot:'transfert de la ROM dans la RAM
80 PRINT:PRINT "Sauver la ROM (O/N) ? ";
100 a$=INKEY$:IF a$="" THEN 100 ELSE a$=UPPER$(a$)
110 slot$=STR$(slot):'conversion de slot (numérique) en slot$ (chaîne)
120 lg=LEN(slot$):'longueur de la chaîne enregistrée dans lg
130 slot$=MID$(slot$,2,lg-1):'retirer le premier caractère de la chaîne qui correspond au signe devant le nombre, un espace pour le + et un – pour du négatif, étant donné que les espaces sont interdits dans le nom d'un fichier
140 file$="ROM-"+slot$+".rom"
150 IF a$="O" THEN GOSUB 340:SAVE file$,B,&4000,&4000 ELSE GOSUB 350
160 PRINT:PRINT "Lister les RSX (O/N) ? ";:a$=""
170 a$=INKEY$:IF a$="" THEN 170 ELSE a$=UPPER$(a$)
180 IF a$="N" THEN GOSUB 350:END ELSE GOSUB 340
190 a=peek(&4005)-&80:'poid fort -&80 à cause du transfert en &4000
200 b=peek(&4004):'poids faible
210 adresse=(a*256)+b:'adresse des RSX
220 a=-1
230 WHILE a<>0
240 a=peek(adresse)
250 adresse=adresse+1
260 IF a<>0 THEN GOSUB 290
270 WEND
280 END
```

```
290 finmot=0
300 IF a>&7F THEN a=a-&80:finmot=1
310 IF a<32 THEN a=0:RETURN:'fin de la liste des RSX
320 IF finmot=0 THEN PRINT chr$(a); ELSE PRINT chr$(a)
330 RETURN
340 PRINT CHR$(24);"OUI";CHR$(24):RETURN
350 PRINT CHR$(24);"NON";CHR$(24):RETURN
```

Le programme proposera de sauver Oui ou Non la ROM et, petite nouveauté, proposera aussi de lister les RSX (Attention, toutes les ROMs n'ont pas de RSX).

En ce qui concerne la liste des RSX, l'adresse réelle est stockée aux adresses &C004 et &C005. Étant donné qu'on a transféré les données en &4000, nous retirons tout simplement &80 à la valeur lue en &C005. Sur CPC, les adresses 16 bits sont stockées en octets poids faible en premier, octets poids fort en dernier.
Comme dans le dump mémoire ci-dessous, nous avons les 2 valeurs en hexadécimales suivantes 72 C0, il faudra en fait les lire &C072, et comme on a retiré &80 à &C0, l'adresse devient &4072.

Le fait d'avoir récupéré le poids fort dans la variable A et le poid faible dans la variable B nous oblige à multiplier la variable A par 256 (&FF), ce qui correspondrait à un décalage de bit vers la gauche de 2 rangs en assembleur, pour obtenir notre valeur 16 bits. Et non, l'aspirine n'est pas fournie avec l'article, désolé.

Qu'avons-nous besoin de savoir pour lister les noms des RSX ?

Chaque mot est terminé avec l'ajout sur le dernier caractère de la valeur &80
Si on regarde plus attentivement, en ce qui concerne par exemple le mot CPM ROM, nous ne voyons pas le M final. "M"+&80

La fin de la liste est toujours signalée par un &00.

Petite subtilité, dans certaines ROMs et notamment dans celle de l'AMSDOS, nous avons une chaîne de valeurs 81 82 83 84 85 86 87 88 89. Une fois qu'on aura retiré &80, nous aurons des soucis d'affichage étant donné que les caractères ASCII de &00 à &1F (00 à 31) sont des caractères de contrôle ayant des effets des plus intéressants, mais ceci fera l'objet d''explications dans un prochain article. Pour simplifier et tenir compte de cette spécificité, dans le listing BASIC après avoir retiré &80 à la valeur du caractère, on vérifiera si la valeur qui reste est inférieure à 32 (&20 qui correspond à l'espace) et dans ce cas-là, on arrêtera tout simplement la lecture.

Dump mémoire de l'AMSDOS transféré en &4000 :

```
4000:  01 00 05 00 72 C0 C3 BC C1 C3 B2 C1 C3 D1 CC C3      ....r...........
4010:  D5 CC C3 E4 CC C3 FD CC C3 01 CD C3 18 CD C3 DA      ................
4020:  CD C3 DD CD C3 E4 CD C3 FE CD C3 2E D4 C3 8A D4      ................
4030:  C3 C4 D4 C3 72 CA C3 0D C6 C3 81 C5 C3 66 C6 C3      ....r........f..
4040:  4E C6 C3 52 C6 C3 63 C7 C3 30 C6 C3 03 C6 C3 68      N..R..c..0.....h
4050:  C1 C3 DB C0 C3 89 C3 C3 01 C3 C3 DB C3 C3 F7 C3      ................
4060:  C3 35 C4 C3 45 C4 C3 E3 C3 C3 FF C3 C3 3A C4 C3      .5..E........:..
4070:  4B C4 43 50 4D 20 52 4F CD 43 50 CD 44 49 53 C3      K.CPM RO.CP.DIS.
4080:  44 49 53 43 2E 49 CE 44 49 53 43 2E 4F 55 D4 54      DISC.I.DISC.OU.T
4090:  41 50 C5 54 41 50 45 2E 49 CE 54 41 50 45 2E 4F      AP.TAPE.I.TAPE.O
40A0:  55 D4 C1 C2 44 52 49 56 C5 55 53 45 D2 44 49 D2      U...DRIV.USE.DI.
40B0:  45 52 C1 52 45 CE 81 82 83 84 85 86 87 88 89 00      ER.RE..........
```

```
ROM SUPERIEURE, NUMERO DU SLOT? 7
Sauver la ROM (O/N) ? NON
Lister les RSX (O/N) ? OUI
CPM ROM
CPM
DISC
DISC.IN
DISC.OUT
TAPE
TAPE.IN
TAPE.OUT
A
B
DRIVE
USER
DIR
ERA
REN
Ready
```

Le premier nom listé est en fait le nom de la ROM et non une RSX.

RSX

Les RSX permettent d'étendre les fonctionnalités du BASIC. Pour les claviers anglais (QWERTY), les commandes sont précédés du caractère | (appelé *pipe*) et pour les claviers français (AZERTY) du caractère ù, ce qui permet de les différencier des commandes internes.

Pour créer une RSX, malheureusement, nous n'avons pas le choix, nous sommes obligés d'utiliser la programmation en assembleur.
Nous avons vu plus haut qu'il était possible de changer de mode graphique sans effacement de l'écran grâce au CALL &BD1C en précisant le mode souhaité dans le registre A. Malheureusement, si on fait CALL &BD1C,1 ou CALL &BD1C,0, rien ne se passera. Nous allons donc créer une RSX très simple permettant de réaliser le changement de mode avec |**MODE,x** (x=0 à 2)

Nous allons travailler sur l'une des 5 images qui a servi dans les banks. Cette image est en MODE 0. La routine en assembleur, avec à gauche l'adresse mémoire, en surligné jaune le code hexadécimal généré suite à l'assemblage du code écrit dans la partie à droite.

```
                        ORG &A400
                        ;*****************************
                        ;*** RSX - INITIALISATION ***
                        ;*****************************
                        RSX_Init
#A400  21 00 A4         LD HL,RSX_Init
                        ;Écrire un RET pour empêcher une autre initialisation
#A403  36 C9            LD (HL),&C9
                        ;BC pointe sur la table des Commandes RSX
#A405  01 12 A4         LD BC,RSX_Commandes
                        ;HL pointe sur 4 octets libres
#A408  21 0E A4         LD HL,RSX_Tampon
                        ;Introduit une RSX dans le logiciel interne
#A40B  C3 D1 BC         JP &BCD1
                        RSX_Tampon
                        ;Tampon de quatre octets.
#A40E  00 00 00 00      DEFS 4
                        RSX_Commandes
                        ;Adresse des mots-clés
#A412  17 A4            DEFW RSX_Mots_Clefs
                        ;saut vers les fonctions
                        ;saut vers RSX 1
```

```
#A414  C3 1C A4     JP MODE_SANS_CLS
                    ;saut vers RSX 2
                    ;saut vers RSX ...
                    RSX_Mots_Clefs
                    ;nom des RSX
                    ;+&80 sur la dernière lettre du nom de la RSX
                    ;permettra de signaler la fin du mot
                    ;nom RSX 1
#A417  4D 4F 44 C5  DEFB "MOD","E"+&80
                    ;nom RSX 2
                    ;nom RSX ...
                    ;La fin de la liste des noms pour les RSX
                    ;est signalée par un zéro.
#A41B  00           DEFB 0
                    ;******************
                    ;**** FONCTIONS ***
                    ;******************
                    ;syntax |MODE,valeur
                    MODE_SANS_CLS
                    ;lecture valeur mode
#A41C  DD 7E 00     LD A,(IX+0)
                    ;CONDITION D'ENTRÉE
                    ;le registre A contient le mode (0, 1 ou 2)
                    ;positionne le mode écran
#A41F  CD 1C BD     CALL &BD1C
                    ;CONDITION DE SORTIE
                    ;les registres A et F sont modifiés
                    ;Retour
#A422  C9           RET
```

Nous allons utiliser la routine en langage machine (code HEXA mais cette fois-ci dans une variable) dans un petit programme en BASIC (Listing RSX.BAS) :

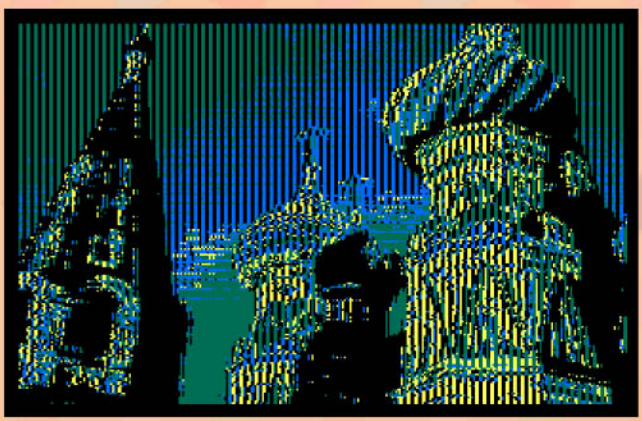
Vous n'aurez pas à saisir les lignes de couleur verte, ce sont des commentaires.

```
Baisser le HIMEM en &9FFF
10 MEMORY &9FFF
Charger l'image "3.ECR" et afficher les bonnes couleurs
20 MODE 0:LOAD"3.ECR":CALL &FFD0
Le code HEXA est stocké dans la chaine A$
30  A$="2100A436C90112A4210EA4C3D1BC0000000017A4C31CA44D4F44C500DD7E00CD1CBDC9"
Boucle i=1 jusqu'à longueur de A$ avec une augmentation de i de +2
40 FOR i=1 TO LEN(A$) STEP 2
b$ = 2 caractères de la chaîne A$ à partir de la position i (retourne 21 00 ...)
50 b$=MID$(a$,i,2)
Écrire en mémoire le code HEXA à partir de &A400
lorsque i=1 la formule (i\2) va retourner 0 (0,5 arrondi au chiffre inférieur)
lorsque i=3 la formule (i\2) va retourner 1 (1,5 arrondi au chiffre inférieur)
VAL permet de transformer une chaîne en VALEUR, en l'occurence &21 &00...
60 POKE &A400+(i\2),VAL("&"+b$)
C'est ICI grâce au NEXT que nous aurons notre i=i+2 (prise en compte du STEP 2)
70 NEXT
initialisation des RSX
90 CALL &A400
test dans les 3 modes sans CLS
```

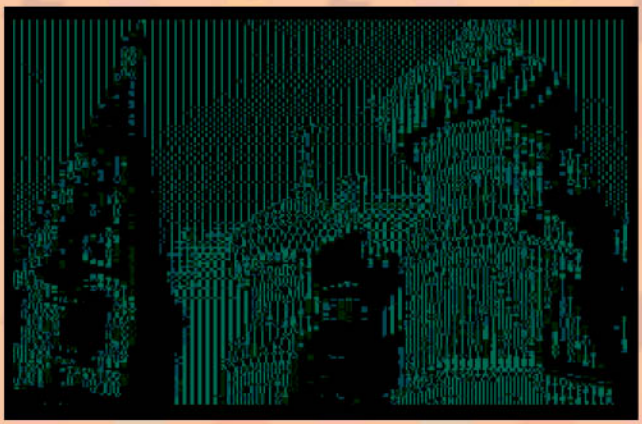
```
110 FOR i=0 TO 2
Notre nouvelle RSX qui va prendre le paramètre 0, 1 puis 2
120 |MODE,i
engendrera un BEEP
130 PRINT CHR$(7)
attend qu'on appuie sur une touche pour passer a l'instruction suivante
140 CALL &BB06
i=i+1 etant donné qu'il n'y a pas de STEP, c'est sous-entendu STEP 1
150 NEXT
on retourne en ligne 110
160 GOTO 110
```

Le petit programme ci-dessus reste assez simple et permet de comprendre l'initialisation et l'utilisation de la nouvelle RSX (l'intérêt est uniquement à ce niveau).

En MODE 1, eh oui ça devient moche, mais notre RSX fonctionne et nous n'avons plus que 4 couleurs au lieu des 16 du départ.



En MODE 2, maintenant c'est vraiment très laid, en même temps il ne nous reste plus que 2 couleurs.



Nos essais sont concluants, la technique pour créer une RSX fonctionne. Maintenant, vous pouvez vous amuser à créer vos propres RSX.

un CAT, CATastrophique

Lorsque nous tapons la commande CAT qui nous permet d'obtenir la liste des fichiers présent sur notre disquette (fonctionne aussi sur cassette, mais ici, nous nous penchons uniquement sur le possible souci avec une disquette), il faut savoir que le résultat est inscrit en RAM.
Si par malheur une de nos routines en assembleur ou des données en binaire sont présentes pile poil dans la mauvaise zone mémoire, ça engendrera invariablement une corruption.

Le premier exemple commercial ayant un *bug* à cause de ce soucis qui me vient en tête, c'est **Hammer Boy** © Dinamic Software (1991). Corruption du sprite pour le chiffre "5" dans la phase 2 du jeu, uniquement sur les originaux disquettes. La version cassette n'a aucun problème et le jeu sur disquette est strictement identique. À noter que nous retrouvons le *bug* dans les compilations disquettes proposant Hammer Boy. Comme quoi, même les professionnels peuvent se faire avoir. La preuve en image.

Qu'avons-nous besoin de savoir ?

Si on fait un ***PRINT HEX$(HIMEM-&7FF)***,
nous obtiendrons l'adresse de début des informations du catalogue en mémoire vive.

Si nous n'avons pas joué avec le MEMORY, nous devons avoir comme réponse : &9E7C.
À partir de cette adresse chaque fichier sera stocké sur 14 octets et quoi qu'il arrive &800 (2048 octets soit 2 Ko) de données seront modifiées en RAM. Ce qui veut dire que, dans le cas présent, toutes les données de &9E7C à &A67B seront altérées.

Voici le détail des 14 octets, classé du rang 0 à 13

0	1 2 3 4 5 6 7 8	9 10 11	12	13
&FF				&00
Début	Nom du fichier	Extension	Ko	Fin

Le rang 0 commence toujours par un &FF.

Les rangs 1 à 8 correspondent au nom du fichier (longueur de 8 caractères)

Les rangs 9 à 11 concernent l'extension du fichier (longueur de 3 caractères) avec une petite subtilité :
- si le rang 9 a une valeur supérieur à &80 = fichier en lecture seule (READ ONLY)

Malheureusement, les fichiers cachés sont filtrés et n'apparaissent pas dans la liste, alors que logiquement, on devrait avoir l'information sur le rang 10 (si valeur supérieure a &80 = fichier caché (HIDDEN))

Le rang 12 concerne le poids du fichier, et vu qu'il est codé sur un seul octet, nous ne pouvons pas avoir de fichier avec une taille supérieure à 255 Ko (en même temps, on serait bien embêté pour le charger).

Le rang 13 est toujours à &00 pour signaler la fin de l'enregistrement.

Maintenant que nous connaissons comment fonctionne le CAT, nous pourrions créer notre propre afficheur.

Et hop, voici un premier jet, que vous pourrez améliorer.

```
1 ' Kukulcan 2016
10 MODE 1:BORDER 0:INK 0,0:INK 1,26:INK 2,15:INK 3,6
20 REM *** ADRESSE DE DEBUT Du CATALOGUE ***
30 adr=HIMEM-&7FF
40 REM *** CHOIX DU USER ***
50 PEN 2:INPUT "USER (0/255): ";US:POKE &A701,US:PEN 1
60 REM *** MASQUER L'AFFICHAGE DES CARACTÈRES ***
70 a=PEEK(&BB5A)
80 POKE &BB5A,&C9
90 CAT
100 POKE &BB5A,a:'restaurer l'affichage des caractères
110 REM *** MON CATALOGUE SPÉCIAL ***
120 deb=adr:GOTO 320
130 'user
140 PRINT US;
150 'NOM 8 caractères
160 FOR i=1 TO 8
170 PRINT CHR$(PEEK(deb+i));
180 NEXT i
190 PRINT ".";
200 'EXTENSION
210 a=PEEK(deb+9):IF a>&80 THEN a=a-&80:ro$=" READ ONLY " ELSE ro$=SPACE$(11)
220 PRINT CHR$(a);
230 PRINT CHR$(PEEK(deb+10));
240 PRINT CHR$(PEEK(deb+11));
250 'KO
260 PRINT USING "###";PEEK(deb+12);
270 PRINT " KO";
280 'READ ONLY
290 PEN 3:PRINT ro$:PEN 1
300 'enregistrement suivant
310 deb=deb+14
320 IF PEEK(deb)<>&FF THEN GOTO 40 ELSE GOTO 130
```

L'équivalent de la poubelle est présent dans le user 229 :

RÉALISATION D'UN PETIT PROGRAMME DE *DUMP* MÉMOIRE EN BASIC

Nous avons vu plus haut comment sauvegarder la ROM de l'AMSDOS sur notre disquette. Maintenant, grâce au programme BASIC ci-dessous, nous allons pouvoir visualiser les valeurs HEXADÉCIMALES, ainsi que les caractères ASCII contenus dans la ROM très facilement. Certes, ce programme n'a pas la puissance, ni la rapidité d'un visualiseur en assembleur, mais il a l'avantage d'être assez court (surtout après avoir retiré les lignes de commentaires) pour bien comprendre le principe.

Comme vous pourrez le constater, on fait appel à des notions abordées plus haut : le MEMORY, le PEEK, la RSX |DIR ou dans les précédents numéros de Côté Gamers. (*Listing MINIDUMP.BAS*)

```
10 ON ERROR GOTO 60:REM en cas d'erreur le code reprend en ligne 60
20 MODE 2:BORDER 9:INK 0,9:INK 1,26:PEN 1
30 MEMORY &3FFF:'abaisser la mémoire disponible pour le BASIC en &3FFF
40 GOSUB 550:GOSUB 360:'appel sous routine cadre/titre puis infos
50 WINDOW #0,3,78,4,19:'définition de la fenêtre de travail
60 adr=&4000:'adresse de démarrage pour la visualisation
70 n=&100:'nombre d'octets à lire
80 GOSUB 180:'dump mémoire
90 REM touches
100 IF INKEY(0)=0 THEN adr=adr+n:GOSUB 180:'curseur haut
110 IF INKEY(2)=0 THEN adr=adr-n:GOSUB 180:'curseur bas
120 IF INKEY(8)=0 THEN adr=adr-&1000:GOSUB 180:'curseur gauche
130 IF INKEY(1)=0 THEN adr=adr+&1000:GOSUB 180:'curseur droit
140 IF INKEY(62)=0 THEN GOTO 450:'touche C pour CHARGER une ROM
150 IF INKEY(53)=0 THEN CALL &BB03:END:'touche F pour FIN
160 GOTO 100
170 END
180 REM *** dump mémoire ***
190 CLS:'effacer l'écran (en fait uniquement la fenêtre courante)
200 FOR i=adr TO adr+n-16 STEP 16
210 REM affichage adresse mémoire
220 PRINT "&";HEX$((i),4);": ";
230 b$=""
240 REM lecture et traitement des 16 valeurs pour une ligne
250 FOR k=0 TO 15
260 c=PEEK(i+k):'lecture de l'octet en mémoire
270 REM affichage hexadécimales
280 PRINT HEX$((c),2);" ";:'affichage de l'octet en HEXADÉCIMAL
290 REM construction de la chaîne ASCII
300 IF c>31 AND c<128 THEN b$=b$+CHR$(c) ELSE b$=b$+CHR$(46)
310 NEXT k
320 REM affichage ASCII une fois les 16 valeurs analysées
330 PRINT SPC(3);:PRINT b$
340 NEXT i
350 RETURN
360 REM *** infos ***
370 LOCATE 2,22
380 FOR i=240 TO 243:'bouche pour afficher les 4 flèches
390 READ a$:'lecture du texte à afficher présent dans la ligne de DATA
400 PRINT CHR$(i);" adresse ";a$;SPC(4);
```

```
410 NEXT i
420 LOCATE 2,24:PRINT "(C)harger une ROM   (F)in"
430 RETURN
440 DATA "+ &100","- &100","- &1000","+ &1000"
450 REM *** charger une ROM ***
460 CLS:'effacer le contenu du window
470 a$="*.rom"
480 |DIR,@a$:'afficher uniquement les fichiers ayant comme extension .rom
490 CALL &BB03:'permet de remplacer le clear input pour mon CPC 464
500 INPUT "fichier de ROM à charger (sans extension)";file$
510 IF file$="" THEN 470
520 LOAD file$+".rom",&4000:'charger la rom en &4000
530 adr=&4000:'on reaffiche à partir de &4000
540 GOTO 80
550 REM Cadre
560 PLOT 8,84,1:DRAWR 0,276,1:DRAWR 624,0,1:DRAWR 0,-276,1:DRAWR -624,0,1
570 REM Titre
580 a$="COTE GAMERS 4 - MINI DUMP MEMOIRE en BASIC"
590 lg=LEN(A$):LOCATE (80-lg)\2,2:PRINT a$:RETURN:'centrer le titre
```

```
COTE GAMERS 4 - MINI DUMP MEMOIRE en BASIC

&4000:  01 00 05 00 72 C0 C3 BC C1 C3 B2 C1 C3 D1 CC C3    ....r...........
&4010:  D5 CC C3 E4 CC C3 FD CC C3 01 CD C3 18 CD C3 DA    ................
&4020:  CD C3 DD CD C3 E4 CD C3 FE CD C3 2E D4 C3 8A D4    ................
&4030:  C3 C4 D4 C3 72 CA C3 0D C6 C3 81 C5 C3 66 C6 C3    ....r........f..
&4040:  4E C6 C3 52 C6 C3 63 C7 C3 30 C6 C3 03 C6 C3 68    N..R..c..0.....h
&4050:  C1 C3 DB C0 C3 89 C3 C3 01 C3 C3 DB C3 C3 F7 C3    ................
&4060:  C3 35 C4 C3 45 C4 C3 E3 C3 C3 FF C3 C3 3A C4 C3    .5..E..........:..
&4070:  4B C4 43 50 4D 20 52 4F CD 43 50 CD 44 49 53 C3    K.CPM RO.CP.DIS.
&4080:  44 49 53 43 2E 49 CE 44 49 53 43 2E 4F 55 D4 54    DISC.I.DISC.OU.T
&4090:  41 50 C5 54 41 50 45 2E 49 CE 54 41 50 45 2E 4F    AP.TAPE.I.TAPE.O
&40A0:  55 D4 C1 C2 44 52 49 56 C5 55 53 45 D2 44 49 D2    U...DRIV.USE.DI.
&40B0:  45 52 C1 52 45 CE 81 82 83 84 85 86 87 88 89 00    ER.RE...........
&40C0:  2A 39 00 22 3E AD 3E C3 32 33 AD AF 32 40 AD F3    *9."}.}.23..2@..
&40D0:  D9 ED 43 3C AD D9 21 FA C0 18 1A 21 40 AD BE C8    ..C<..!....!@...
&40E0:  C5 46 77 B7 78 C1 28 E7 F3 08 D9 ED 4B 3C AD B7    .Fw.x.(.....K<..
&40F0:  08 D9 21 32 C1 22 34 AD FB C9 F3 08 D9 22 38 AD    ..!2."4......."8.

↑ adresse + &100    ↓ adresse - &100    ← adresse - &1000    → adresse + &1000
(C)harger une ROM   (F)in
```

Nous continuerons à vous parler de la mémoire dans le prochain numéro en abordant la mémoire écran et les *sprites*.

Je vous rappelle que pour vous faciliter la tâche et tester plus facilement les listings et exemples présents dans cet article, une compilation est mise à disposition gratuitement sur le site de **cpc-power**.