9/8.5.3

Edition de la mémoire centrale

Dans ce chapitre vous trouverez un programme bien utile si vous avez l'âme d'un bidouilleur, que vous travailliez en Basic ou sous CP/M. Le programme EDIMEM permet en effet d'éditer la mémoire centrale. Editer signifie visualiser mais également modifier le contenu d'une ou de plusieurs mémoires. Il sera par exemple possible de franciser un programme ou de modifier les données qu'il affiche sur l'écran. Mais attention, je dois vous mettre en garde ! Si vous modifiez un ou plusieurs octets utilisés par le système ou le CP/M, cela plantera irrémédiablement votre machine, et la seule solution consistera à la mettre hors tension puis sous tension. Les modifications effectuées en mémoire seront bien entendu perdues...

Le programme d'édition est proposé dans deux versions :

- une version Basic qui permet d'éditer la mémoire lorsque le Basic est actif, en d'autres termes lorsque vous n'êtes pas sous CP/M ;
- une version Turbo Pascal qui permet d'éditer la mémoire lorsque le CP/M est actif.

COMMENT UTILISER LE PROGRAMME

Quelle que soit la version utilisée, l'écran affiché par le logiciel se divise en quatre zones :

- dans la partie gauche de l'écran apparaissent les adresses des premiers octets de chaque ligne ;
- dans la partie centrale de l'écran sont affichés 16 octets consécutifs en hexadécimal ;
- dans la partie droite de l'écran apparaissent les codes ASCII des octets. Cela est bien pratique pour localiser des zones de texte ... ;
- enfin, en bas de l'écran apparaît le menu principal à cinq options :
  F1 affiche les 256 octets suivants,
  F2 affiche les 256 octets précédents,
  F3 permet d'entrer l'adresse de la première mémoire à afficher,

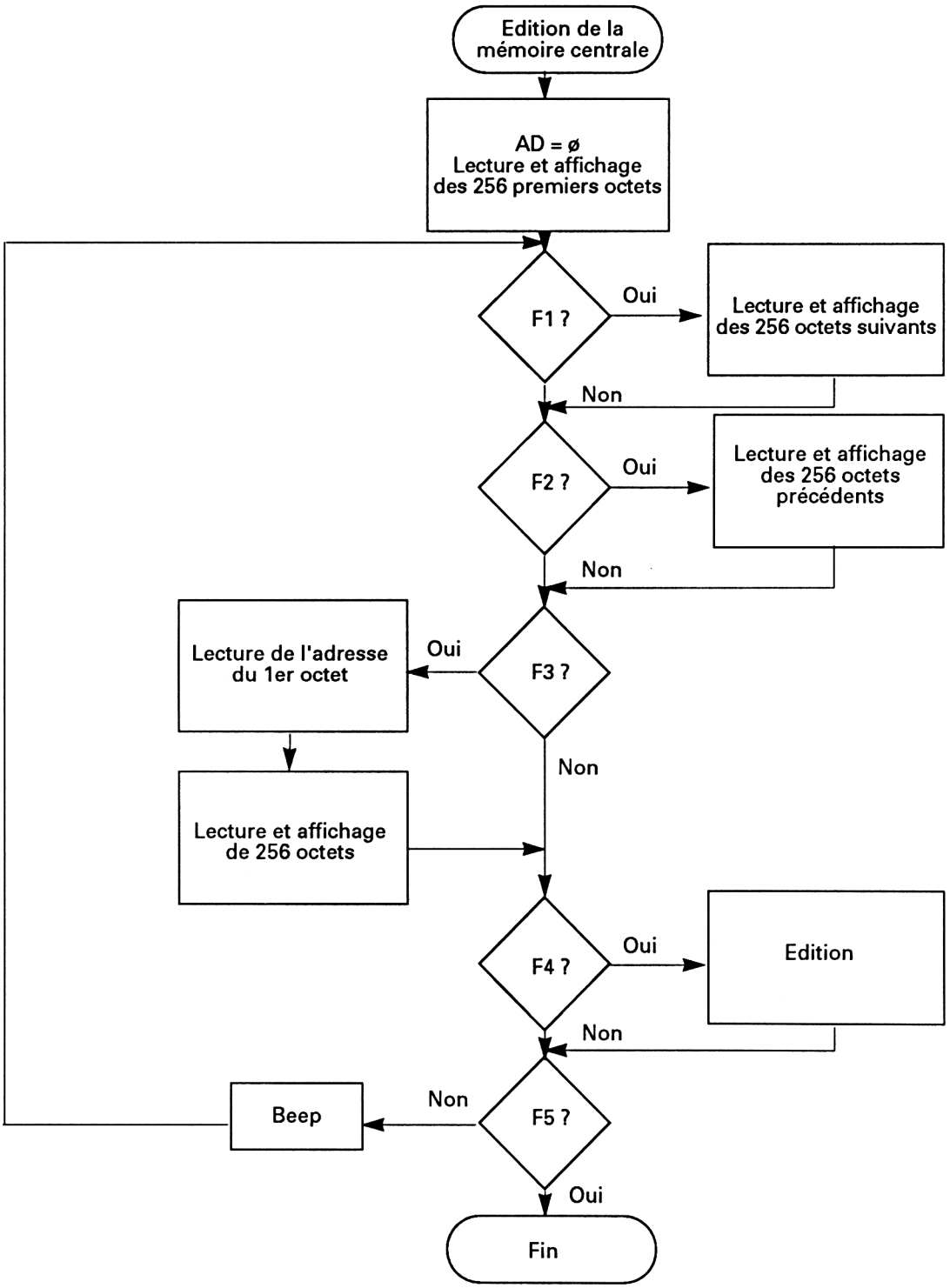
F4 permet de modifier un ou plusieurs des octets affichés sur l'écran courant,
F5 met fin au programme.

Lorsque le mode **EDITION** est activé, vous devez respecter les points suivants :

- Pour modifier un octet, utilisez les touches flèches du clavier pour que le curseur se trouve sur son poids fort. Tapez alors la nouvelle valeur hexadécimale à stocker. Si vous tentez d'entrer une valeur hexadécimale alors que le curseur est mal positionné, l'affichage est brouillé.
- Lorsque les modifications ont été effectuées, appuyez sur la touche « **Q** » (version Basic) ou sur la touche **ENTER** (version Turbo Pascal) pour revenir au menu général. Les données entrées sont alors stockées en mémoire. Si vous avez modifié des mémoires illicites, l'ordinateur peut alors être « planté ». Dans ce cas, mettez-le hors puis sous tension et relancez le programme.
- La touche **F3** du menu général doit être utilisée lorsque les 256 octets à éditer se trouvent à une adresse très distante des données en cours d'édition. Dans les autres cas, utilisez les touches **F1** et **F2**.

EDITION DE LA MÉMOIRE EN BASIC

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme est essentiellement écrit en Basic. Il utilise une RSX pour afficher les codes ASCII des octets visualisés. En voici le listing :

```
 1                              ORG   9000H
 2                              LOAD  9000H
 3                    ;--------------------------------
 4                    ; RSX SPRINT
 5                    ; Format : |SPRINT,CH
 6                    ; Entree : CH=Code ASCII du
 7                    ;          caractere a imprimer
 8                    ; Sortie : Affichage caractere
 9                    ;--------------------------------
10                    ;
11                    ;
12                    ;---------------------------------
13                    ; Declaration des constantes
14                    ; et des variables du programme
15                    ;---------------------------------
16                    ;
17                    WRCHAR:     EQU   0BB5DH          ;TXT WR CHAR
18                    LOGEXT:     EQU   0BCD1H          ;KL LOG EXT
19                    BUF:        DS    4               ;ZONE RAM POUR LOG EXT
20 9004 0990          PTRTAB:     DW    TABLE           ;Pointeur TABLE
21 9006 C31A90                    JP    SPRINT          ;Affichage du caractere
22 9009 53505249      TABLE:      DB    "SPRIN"
22 900D 4E
23 900E D4                        DB    "T"+80H
24 900F 00                        DB    0               ;Fin de table
25                    ;
26                    ;---------------------------------
```

```
27                ; Definition de la RSX
28                ;---------------------------------
29                ;
30                DEFRSX:     EQU  $                ;Point d'entree
31 9010 010490              LD   BC,PTRTAB        ;Ptr table definition
32 9013 210090              LD   HL,BUF           ;Buffer pour LOG EXT
33 9016 CDD1BC              CALL LOGEXT           ;Definition de la RSX
34 9019 C9                  RET
35                ;
36                ;----------------------
37                ; Traitement de SPRINT
38                ;----------------------
39                ;
40                SPRINT:     EQU  $                ;Point d'entree
41 901A DD7E00              LD   A,(IX+0)         ;Caract a afficher
42 901D CD5DBB              CALL WRCHAR           ;Affichage
43 9020 C9                  RET
44                          END
```

Cette RSX utilise la macro **TXT WR CHAR** du FIRMWARE qui permet d'afficher le caractère dont le code ASCII lui est passé (ce qui n'est pas le cas de l'instruction **PRINT** classique pour des codes ASCII inférieurs à 32).

Le code ASCII du caractère à afficher est passé à la RSX sous la forme d'un nombre entier compris entre 0 et 255. Il est récupéré dans la pile à l'aide du registre IX et passé à la macro **TXT WR CHAR** à travers le registre A :

LD A, (IX+0) ; **Caractère à afficher**
CALL WRCHAR ; **Affichage**

Passons au programme Basic. Ses premières lignes contiennent le programme principal. Ce dernier est fort simple. Il active les deux

sous-programmes principaux (lignes 1040 et 1050) et affiche le message « **Au revoir** » lorsque la touche **F5** a été pressée (ligne 1060).

Le sous-programme d'initialisation occupe les lignes 1090 à 1200. Les tâches qui lui incombent sont multiples :

- Tout d'abord, il active le mode d'affichage 80 colonnes à l'aide d'une instruction **MODE** (ligne 1130).
- La ligne 1140 dimensionne le tableau **TMEM** dans lequel seront stockés les 256 octets en cours d'édition.
- La ligne 1150 dévalide l'affichage du curseur, cela dans un but purement esthétique, pour ne pas brouiller l'affichage des 256 octets. L'affichage du curseur sera à nouveau validé dans le mode **EDITION**.
- La RSX **¦SPRINT** dont nous avons parlé plus haut est ensuite mémorisée à l'aide d'un sous-programme situé en 2540 (ligne 1160). Les données hexadécimales de la RSX sont incorporées au programme Basic sous la forme de **DATA**. Ces données sont stockées en mémoire à partir de l'adresse **&9000**. Lorsque toutes les données sont mémorisées, la RSX est installée à l'aide d'une instruction **CALL** (ligne 2630).
- L'adresse de la première mémoire affichée est initialisée à zéro dans la variable **ad** (ligne 1170). Les 256 octets situés à partir de cette adresse sont lus (ligne 1180) et affichés sur l'écran (ligne 1190).

Le programme se poursuit par le sous-programme principal dont la tâche consiste à scruter le clavier dans l'attente d'une commande (touches F1 à F5 et à activer le sous-programme de traitement correspondant).

La ligne 1260 scrute le clavier à l'aide de la fonction **INKEY$**. Cette ligne est exécutée jusqu'à ce qu'une touche soit pressée. Le code de la touche pressée est alors converti en ASCII et comparé aux divers codes autorisés (49 pour la touche F1, 50 pour la touche F2, ..., 53 pour la touche F5).

Lorsque la touche **F1** est pressée, l'adresse **ad** est incrémentée de 256 octets lignes 1320 à 1340. Les opérations effectuées dans ces lignes peuvent paraître complexes. Il est en effet légitime de penser que pour incrémenter la variable **ad** de 256, il suffit de faire :

```
ad = ad + 256
```

En fait, il n'en est rien. Pourquoi ? Et bien tout simplement parce que les variables entières manipulées par l'Amstrad sont des variables 16 bits signées comprises entre - 32768 et 32767. Essayez par exemple l'addition suivante :

```
PRINT &8000 + 256
```

Le résultat n'est pas celui attendu (32768+256 = 33024). En effet, **&8000** est codé - 32768 en mémoire, et - 32768+256 a pour résultat - 32515...

Lorsque la première adresse des octets à éditer est calculée, les 256 octets sont lus et affichés sur l'écran (lignes 1350 et 1360). La tâche correspondant à l'appui sur la touche **F1** étant terminée, le contrôle est redonné au sous-programme de scrutation du clavier (ligne 1370).

Lorsque la touche **F2** est pressée, l'adresse du premier octet affiché est décrémentée de 256 (lignes 1420 et 1430). Là aussi, la modification de l'adresse n'est pas aussi simple que l'on pouvait s'y attendre. Les 256 nouveaux octets sont lus et affichés (lignes 1440 et 1450), et le contrôle est redonné au sous-programme de scrutation du clavier (ligne 1460).

Lorsque la touche **F3** est pressée, le programme demande d'entrer l'adresse du premier octet à éditer (lignes 1520 et 1530). Les 256 nouveaux octets sont lus et affichés (lignes 1540 et 1550), et le contrôle est redonné au sous-programme de scrutation du clavier (ligne 1560).

Lorsque la touche **F4** est pressée, le sous-programme situé en 1710 est activé (ligne 1610). Etudions le fonctionnement de ce sous-programme. L'affichage du curseur est validé à l'aide d'une instruction **CURSOR**. Ceci, pour visualiser la position de la prochaine donnée éditée.

Les coordonnées du curseur se trouvent dans les variables **PX** et **PY**. Au début du sous-programme, ces données sont initialisées aux valeurs 12 et 5, ce qui correspond au poids fort du premier octet affiché sur l'écran (lignes 1770 et 1780).

Le programme se met en attente de l'appui sur une touche du clavier ligne 1800. Lorsque la touche pressée est une touche flèche, le curseur est déplacé dans le sens demandé (lignes 1820 à 1870).

Lorsque la touche pressée est un digit hexadécimal (0 à 9, A à F majuscule ou minuscule), le programme affiche ce code sur l'écran (ligne 1880). Si le code entré est une lettre minuscule, elle est transformée en la lettre majuscule correspondante ligne 1890. Si la donnée entrée est un poids fort, elle est simplement affichée sur l'écran (ligne 1900). Si la donnée entrée est un poids faible, le code ASCII correspondant est calculé (ligne 1930), affiché à l'aide de la RSX **SPRINT** (ligne 1950) et stocké dans le tableau **TMEM** (lignes 1960 et 1970).

Lorsque la touche pressée est la lettre « q » (ou « Q »), le sous-programme d'édition de la mémoire prend fin. L'affichage du curseur est dévalidé (ligne 2050) et les données du tableau **TMEM** sont stockées en mémoire (ligne 2060).

Le programme se poursuit par des sous-programmes de bas niveau, fréquemment appelés.

Le sous-programme de lecture de 256 octets se trouve entre les lignes 2090 et 2160. Il utilise une boucle **FOR NEXT** et l'instruction **PEEK** pour stocker les 256 octets commençant en **ad** dans le tableau **TMEM**.

Le sous-programme d'affichage des 256 octets en mémoire se trouve entre les lignes 2180 et 2440. Il débute par l'affichage de l'en-tête lignes 2230 à 2270.

Une première boucle **FOR NEXT** permet d'afficher les premières adresses de chaque ligne (lignes 2280 à 2400).

Une seconde boucle **FOR NEXT** affiche les codes hexadécimaux des 16 octets de chaque ligne (lignes 2310 à 2330).

Enfin, une troisième boucle **FOR NEXT** affiche les codes ASCII des 16 octets de chaque ligne à l'aide de la RSX **¦SPRINT** (lignes 2350 à 2370).

Le sous-programme se termine par l'affichage du menu général lignes 2410 à 2430.

Le sous-programme d'écriture des 256 octets en mémoire utilise la même technique que le sous-programme de lecture. Il occupe les lignes 2460 à 2530.

Le programme se termine par le sous-programme de stockage de la RSX **¦SPRINT** dont nous avons déjà parlé plus haut (lignes 2540 à 2680).

Le listing du programme est le suivant :

```
1000 REM ------------------------------
1010 REM Edition de la memoire centrale
1020 REM ------------------------------
1030 REM
1040 GOSUB 1090 'Initialisation
1050 GOSUB 1220 'Attente d'une commande
1060 PRINT "Au revoir."
1070 END
1080 REM
1090 REM - - - - - - - - - - - - - - - - - - - - - -
1100 REM Initialisation des variables du programme
1110 REM - - - - - - - - - - - - - - - - - - - - - -
1120 REM
1130 MODE 2
1140 DIM tmem(256)
1150 CURSOR 0,0 'Curseur off
1160 GOSUB 2540 'Memorisation du S/P Assembleur
1170 ad=0 'Adresse 1ere memoire affichee
1180 GOSUB 2090 'Lecture de 256 octets
1190 GOSUB 2180 'Affichage
1200 RETURN
1210 REM
1220 REM - - - - - - - - - - - - - - - - - - - - - - - -
1230 REM Attente d'une commande tapee par l'utilisateur
1240 REM - - - - - - - - - - - - - - - - - - - - - - - -
1250 REM
1260 ch$=INKEY$:IF ch$="" THEN 1260
1270 a=ASC(ch$)
1280 IF a<>49 THEN 1410
1290 '- = - = - = - = - = -
1300 ' 256 octets suivants
1310 '- = - = - = - = - = -
1320 IF ad<&FF00 THEN ad=ad+256 ELSE ad=255-&FFFF+ad
1330 IF ad<0 THEN ad=ad+65536
1340 IF ad>=65536 THEN ad=ad-65536
1350 GOSUB 2090 'Lecture de 256 octets
1360 GOSUB 2180 'Affichage
1370 GOTO 1220
1380 '- = - = - = - = - = - =
1390 ' 256 octets precedents
1400 '- = - = - = - = - = - =
1410 IF a<>50 THEN 1510
1420 IF ad>=&100 THEN ad=ad-256 ELSE ad=&FFFF-255+ad
1430 IF ad<0 THEN ad=ad+65536
1440 GOSUB 2090 'Lecture de 256 octets
1450 GOSUB 2180 'Affichage
1460 GOTO 1220
1470 '- = - = - = - = - = - = - = - =
```

```
1480 ' Edition de l'adresse du 1er octet
1490 '- = - = - = - = - = - = - = - = - = -
1500 REM
1510 IF a<>51 THEN 1600
1520 LOCATE 20,24
1530 INPUT"Adresse : ";ad
1540 GOSUB 2090 'Lecture de 256 octets
1550 GOSUB 2180 'Affichage
1560 GOTO 1220
1570 '- = - = - = - = - = - =
1580 ' Edition de 256 octets
1590 '- = - = - = - = - = - =
1600 IF a<>52 THEN 1660
1610 GOSUB 1710 'Edition
1620 GOTO 1220
1630 '- = - = - = - = - = - = - =
1640 ' Fin du programme ou erreur
1650 '- = - = - = - = - = - = - =
1660 IF a<>53 THEN SOUND 1,100,10:GOTO 1220
1670 CLS
1680 CURSOR 1,1
1690 RETURN
1700 REM
1710 REM - - - - - - - - - - - - - - - - - - - - - -
1720 REM Edition de 256 octets en mode plein ecran
1730 REM - - - - - - - - - - - - - - - - - - - - - -
1740 REM
1750 CURSOR 1,1
1760 n=0
1770 px=12:py=5 'Position du curseur au depart
1780 LOCATE px,py
1790 '
1800 ch$=INKEY$:IF ch$="" THEN 1800
1810 ch=ASC(ch$)
1820 IF ch=240 AND py>5 THEN py=py-1:LOCATE px,py
1830 IF ch=241 AND py<20 THEN py=py+1:LOCATE px,py
1840 IF ch=242 AND px>12 AND ((px-13) MOD 3)=0 THEN px=px-1:
LOCATE px,py:GOTO 1860
1850 IF ch=242 AND px>12 AND ((px-13) MOD 3)<>0 THEN px=px-2
:LOCATE px,py
1860 IF ch=243 AND px<58 AND ((px-12) MOD 3)=0 THEN px=px+1:
LOCATE px,py:GOTO 1880
1870 IF ch=243 AND px<58 AND ((px-12) MOD 3)<>0 THEN px=px+2
:LOCATE px,py
1880 IF (ch>=48 AND ch<=57) OR (ch>=65 AND ch<=70) OR (ch>=9
7 AND ch<=102) THEN 1890 ELSE 2040
1890 IF ch>=97 AND ch<=102 THEN ch=ch-&20
1900 PRINT CHR$(ch);
```

```
1910 IF n=1 THEN 1920 ELSE 2020
1920 px=px+3
1930 IF (ch>=48 AND ch<=57) THEN va=va+ch-48 ELSE va=va+ch-5
5
1940 LOCATE 61+(px-12)\3,py
1950 !SPRINT,va
1960 a=(px-12)\3+(py-5)*16-1+ad
1970 tmem(a-ad+1)=va
1980 IF px>57 THEN px=57
1990 LOCATE px,py
2000 n=0
2010 GOTO 2040

2020 n=1
2030 IF (ch>=48 AND ch<=55) THEN va=(ch-48)*16 ELSE va=(ch-5
5)*16
2040 IF UPPER$(ch$)<>"Q" THEN 1800
2050 CURSOR 0,0
2060 GOSUB 2460 'Ecriture des 256 octets en memoire
2070 RETURN
2080 REM
2090 REM - - - - - - - - - - - -
2100 REM Lecture de 256 octets
2110 REM - - - - - - - - - - - -
2120 REM
2130 FOR i=0 TO 255
2140   IF ad+i<65536 THEN tmem(i+1)=PEEK(ad+i) ELSE tmem(i+1
)=PEEK(ad+i-65536)
2150 NEXT i
2160 RETURN
2170 REM
2180 REM - - - - - - - - - - - - - - - - - - - -
2190 REM Affichage de 256 octets consecutifs
2200 REM - - - - - - - - - - - - - - - - - - - -
2210 REM
2220 vi=1 'Pointeur dans le tampon
2230 CLS
2240 LOCATE 1,3
2250 PRINT"  Deplacements ----------------- Codes Hexa";
2260 PRINT"-----------------  Valeurs ASCII"
2270 PRINT
2280 FOR i=0 TO 15
2290   PRINT "     ";HEX$(ad+vi-1,4);
2300   PRINT"  ";
2310   FOR j=0 TO 15
2320     PRINT HEX$(tmem(vi+j),2);" ";
2330   NEXT j
2340   PRINT"  ";
2350   FOR j=0 TO 15
```

```
2360      |SPRINT,tmem(vi+j)
2370    NEXT j
2380    PRINT
2390    vi=vi+16
2400 NEXT i
2410 LOCATE 1,22
2420 PRINT"       F1 = 256 octets suivants,  F2 = 256 octets
precedents"
2430 PRINT"       F3 = Edition adresse,      F4 = Edition,
F5 = Fin"
2440 RETURN
2450 REM
2460 REM - - - - - - - - - - - - -
2470 REM Ecriture de 256 octets
2480 REM - - - - - - - - - - - - -
2490 REM
2500 FOR i=0 TO 255
2510    POKE(ad+i),tmem(i+1)
2520 NEXT i
2530 RETURN
2540 REM - - - - - - - - - - - - - - - - - -
2550 REM Memorisation du S/P Assembleur
2560 REM - - - - - - - - - - - - - - - - - -
2570 REM
2580 FOR i=&9000 TO &9020
2590    READ a$
2600    a=VAL("&"+a$)
2610    POKE i,a
2620 NEXT i
2630 CALL &9010
2640 REM
2650 DATA FC,A6,4,90,9,90,C3,1A,90,53, 50,52,49,4E,D4,0
2660 DATA 1,4,90,21,0,90,CD,D1,BC,C9,DD,7E,0,CD,5D,BB
2670 DATA C9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2680 RETURN
```

EDITION DE LA MÉMOIRE SOUS CP/M

La logique du programme Turbo Pascal est la même que celle du programme Basic décrit précédemment. Nous n'y reviendrons pas.

Les sous-programmes Basic sont remplacés par des procédures Turbo Pascal.

Le programme principal se trouve non plus au début mais à la fin du listing.

Trois fonctions supplémentaires ont été implémentées :

– **Hex2** qui convertit en hexadécimal sur deux digits un entier compris entre 0 et 255 ;
– **Hex4** qui convertit en hexadécimal sur quatre digits un entier compris entre 0 et 65535 ;
– **AscDec** qui convertit en décimal une donnée ASCII.

Nous allons analyser le fonctionnement de ces fonctions. Pour le reste du programme, vous vous reporterez aux sous-programmes décrits précédemment.

Fonction Hex2

La conversion hexadécimale d'un entier compris entre 0 et 255 est élémentaire. Pour ce faire, il suffit d'isoler le poids fort et le poids faible et de convertir ces deux données en hexadécimal.

Pour isoler le poids fort, il suffit de diviser l'entier à convertir par 16 :

```
I:=NAC Div 16;
```

Pour isoler le poids faible, il suffit de soustraire le poids fort multiplié par 16 du nombre à convertir :

```
I:=NAC-((NAC DIV 16) * 16);
```

Les poids fort et faible étant isolés, il faut les convertir en hexadécimal. Pour cela, il suffit de les comparer à la valeur 9. Une valeur inférieure ou égale à 9 correspond à un chiffre. Une valeur supérieure à 9 correspond à une lettre :

```
IF I<=9 then Ch:=Chr(I+48)
        else  Ch:=Chr (I+55);
```

Fonction Hex4

La conversion hexadécimale d'un entier compris entre 0 et 65536 est plus complexe. Cela est principalement dû au fait que les données entières manipulées par le Turbo Pascal sont des données signées comprises entre - 32768 et 32767. Lorsque la donnée à convertir est supérieure à 32767 (c'est-à-dire négative), elle est rendue positive par une soustraction de **$8000** et la variable **Sup** est initialisée à **True**. Cela permettra par la suite d'ajouter **$8** au poids fort de la conversion.

Les quatre quartets sont isolés selon la méthode utilisée dans la fonction **Hex2**. Le premier quartet est cependant incrémenté de $8 dans le cas où la variable **Sup** a pour valeur **True** :

```
If Sup then If (I+8)<=9 then Ch1:=Chr(I+8+48)
                        else  Ch1:=Chr(I+8+55)
       else Ch1:=Chr(I+48);
```

Fonction AscDec

La fonction **AscDec** est très simple. En fonction du code ASCII de la valeur passée, elle soustrait 48 ou 55 pour obtenir :

– un chiffre compris entre 0 et 9
ou – une lettre comprise entre A et F

```
Case Ord (A) of
   48...57 : AscDec:=Ord (A)-48;
   65..70  : AscDec:=Ord(A)-55;
end;
```

Le listing du programme est le suivant :

```
Program EdiMem;
{---------------------------------}
{ Edition de la memoire centrale }
{---------------------------------}

Type
  St2 = String[2];
  St4 = String[4];

Var
  Ch1,
  Ch2,
  Ch3,
  Ch   : Char;
  TMem : Array[1..256] of Byte;
  VI,
  I,
  J,
  A,
  Ad   : Integer;
  Sup,
  Stop : Boolean;
  Val_Asc,
  N,
  PX,
  PY   : Byte;


Function Hex2(NAC:Integer):St2;
{---------------------------------------}
{ Conversion Hexadecimale sur 2 digits }
{---------------------------------------}

begin
  I:=NAC Div 16;
  If I<=9 then Ch:=Chr(I+48)
          else Ch:=Chr(I+55);
  I:=NAC-((NAC Div 16) * 16);
  If I<=9 then Hex2:=Ch+Chr(I+48)
          else Hex2:=Ch+Chr(I+55);
end;


Function Hex4(NAC:Integer):St4;
{---------------------------------------}
{ Conversion Hexadecimale sur 4 digits }
{---------------------------------------}
```

```
begin
  Sup:=False;
  If NAC<0 Then begin
                  NAC:=NAC-$8000;
                  Sup:=True;
                end;

  I:=NAC Div 4096;
  If Sup then If (I+8)<=9 then Ch1:=Chr(I+8+48)
                          else Ch1:=Chr(I+8+55)
         else Ch1:=Chr(I+48);
  NAC:=NAC-((NAC Div 4096) * 4096);
  I:=NAC Div 256;
  If I<=9 then Ch2:=Chr(I+48)
          else Ch2:=Chr(I+55);
  NAC:=NAC-((NAC Div 256) * 256);
  I:=NAC Div 16;
  If I<=9 then Ch3:=Chr(I+48)
          else Ch3:=Chr(I+55);
  NAC:=NAC-((NAC Div 16) * 16);
  If NAC<=9 then Hex4:=Ch1+Ch2+Ch3+Chr(NAC+48)
            else Hex4:=Ch1+Ch2+Ch3+Chr(NAC+55);
end;


Function AscDec(A:Char):Integer;
{--------------------------------------}
{ Conversion Decimale d'un code ASCII }
{--------------------------------------}

begin
  Case Ord(A) of
    48..57 : AscDec:=Ord(A)-48;
    65..70 : AscDec:=Ord(A)-55;
  end;
end;


Procedure Lit_256;
{----------------------------------------}
{ Lecture de 256 octets et memorisation }
{----------------------------------------}

begin
  For I:=0 to 255 do
    TMem[I+1]:=Mem[Ad+I];
end;
```

```
Procedure Ecrit_256;
{---------------------------------------}
{ Sauvegarde des 256 octets en memoire }
{---------------------------------------}

begin
  For I:=0 to 255 do
    Mem[Ad+I]:=TMem[I+1];
end;


Procedure Affiche;

{---------------------------------------}
{ Affichage des 256 octets en memoire }
{---------------------------------------}

begin
  VI:=1;
  ClrScr;
  GotoXY(1,3);
  Write('Deplacements ------------------ Codes Hexa');
  Writeln('-----------------  Valeurs ASCII');
  Writeln;
  For I:=0 to 15 do
  begin
    Write('     ',Hex4(Ad+VI-1),'   ');
    For J:=0 to 15 do
      Write(Hex2(TMem[VI+J]),' ');
    Write('  ');
    For J:=0 to 15 do
      If TMem[VI+J]>=32
      then Write(Chr(TMem[VI+J]))
      else Write(' ');
    Writeln;
    VI:=VI+16;
  end;
  GotoXY(1,22);
  Writeln('      F1 = 256 octets suivants,  F2 = 256 octets
precedents');
  Writeln('      F3 = Edition adresse,      F4=Edition,
 F5 = Fin');
end;


Procedure Initialisation;
{-----------------------------------------------}
{ Initialisation des variables du programme }
{-----------------------------------------------}
```

```
  Ad:=0;
  Stop:=False;
  Lit_256;
  Affiche;
end;


Procedure Edition;
{-------------------------------------}
{ Edition des 256 octets en memoire }
{-------------------------------------}

begin
  N:=0;
  PX:=12;
  PY:=5;

  GotoXY(PX,PY);
  Repeat
    While Not KeyPressed do;
    Read(Kbd,Ch);
    If (Ord(Ch)<=244) And (Ord(Ch)>=240) And (N=0) then
    begin
      Case Ord(Ch) of
        240 : If PY>5  then PY:=PY-1;
        241 : If PY<20 then PY:=PY+1;
        242 : If PX>12 then If (PX-13) MOD 3=0 then PX:=PX-1

                                                    else PX:=PX-2
;
        243 : If PX<58 then If (PX-12) MOD 3=0 then PX:=PX+1

                                                    else PX:=PX+2
;
      end;
      GotoXY(PX,PY);
    end
    else
    begin
      Case Ord(Ch) of
        48..57,
        65..70,
        97..102 : begin
                    If Ord(Ch) IN [97..102] then Ch:=Chr(Ord
(Ch)-$20);
                    Write(Ch);
                    If N=1 then begin
                                  PX:=PX+3;
```

```
                                        Val_Asc:=Val_Asc+AscDec(Ch
);
                                        GotoXY(61+(PX-12) DIV 3,PY
);
                                        If Val_Asc<32 then Write('
 ')
                                                        else Write(C
hr(Val_Asc));
                                        A:=(PX-12) DIV 3+(PY-5)*16
-1;
                                        TMem[A+1]:=Val_Asc;
                                        If PX>57 then PX:=57;
                                        GotoXY(PX,PY);
                                        N:=0;
                                      end
                                 else begin
                                        N:=1;
                                        Val_Asc:=AscDec(Ch)*16;
                                      end;
                    end;
      end;
    end;
  until Ord(Ch)=13;

  Ecrit_256;
end;


Procedure Attente;
{-----------------------------------------------}
{ Acquisition des commandes entrees au clavier }
{-----------------------------------------------}

begin
  Repeat
    Repeat
    Until KeyPressed;
    Read(Kbd,Ch);
    Case Ch of
      '1' : begin
              If Ad<$FF00 then Ad:=Ad+256
                          else Ad:=255-$FFFF+Ad;
              Lit_256;
              Affiche;
            end;
      '2' : begin
              If Ad>=$100 then Ad:=Ad-256
                          else Ad:=$FFFF-255+Ad;
```

```
              Lit_256;
              Affiche;
            end;
      '3' : begin
              GotoXY(20,24);
              Write('Adresse : ');
              Readln(Ad);
              Lit_256;
              Affiche;
            end;
      '4' : Edition;
      '5' : Stop:=True;
    end;
  until Stop;
  ClrScr;
  Writeln('Au revoir.');
end;

                          {----------------------}
                          { PROGRAMME PRINCIPAL }
                          {----------------------}

begin
  Initialisation; { Initialisation des variables du programm
e }
  Attente;        { Attente des commandes entrees au clavier
  }
end.
```