9/8.11

CAPS LOCK intéractif

Qui n'a pas, un jour, été confronté à l'angoissante question : « Suis-je en caractère MAJUSCULE ou minuscule ? », lors de la frappe d'un listing, ou de l'utilisation d'un logiciel Basic.

L'idéal est de posséder un voyant **CAPS LOCK** comme sur beaucoup de claviers de type professionnel, mais la modification du clavier demande une certaine habitude dans le maniement du fer à souder, et la possession d'un matériel adéquat, sans compter la perte de la garantie si vous avez acheté votre CPC récemment.

Nous nous sommes donc attachés à la possibilité de reconnaître cet état dans les abîmes des adresses de la mémoire RAM des CPC 664 et 6128. Vous pourrez faire de même avec un CPC 464 en sachant que les adresses mémoires que nous vous donnons ci-dessous ne sont pas forcément identiques, mais doivent certainement être très proches.

Présentation de CAPS LOCK

Après quelques Pokes appropriés aux environs de la zone d'adresses &B6xx, et quelques désagréments — vous pouvez effectuer des Pokes partout dans la mémoire de votre CPC sans aucun danger pour celui-ci, mais nous vous conseillons d'avoir sauvegardé vos programmes si vous en aviez entré un, et de retirer toute disquette ou cassette de votre lecteur — nous avons découvert quelques fonctions intéressantes, dont la possibilité sur CPC 664 et CPC 6128 de forcer le mode majuscule. En effet, l'adresse mémoire &B632 contient l'état **CAPS LOCK** et permet de forcer celui-ci.

Essayez l'instruction : **POKE &B632, &FF**

puis frappez sur les touches de caractères alphabétiques : les caractères sont affichés en majuscule.

Le retour à l'état initial est effectué par l'instruction :

POKE &B632,&00.

Cet état peut, de plus, être lu grâce à l'instruction :

X = PEEK (&B632)

qui place la valeur &00 (ou zéro) dans la variable **X** lorsque vous êtes en mode minuscule, et la valeur &FF (ou 255) lorsque la touche **CAPS LOCK** a été activée.

Nous pouvons ainsi, dans un logiciel basic, forcer le mode majuscule pour éviter d'avoir recours à l'instruction : **A$ = UPPER$ (A$)** qui transforme le contenu d'une variable alphanumérique en caractères majuscules.

Mais nous pouvons tirer parti, plus efficacement, de cette fonctionnalité en l'utilisant dans un programme Basic, pour afficher l'état de l'activité **CAPS LOCK**.

Utilisation basic de l'adresse mémoire &B632

Nous vous proposons, ci-après, un programme qui permettra l'affichage dans le bord supérieur droit de l'état **CAPS LOCK** de votre CPC.

EXPOSE DU PROBLEME

La méthode pour reconnaître cet état étant expliquée ci-dessus, il nous reste à en décrire la mise en œuvre.

Il nous faut tester régulièrement le contenu de l'adresse mémoire &B632. Ce test est impensable avec l'instruction : **X = INP(&B632)** et le traitement de ce test, ou l'appel à un sous-programme de traitement, toutes les deux ou trois lignes Basic, nous gâcherions de la place mémoire.

Une possibilité intéressante du Basic Locomotive est celle d'intégrer, dans un programme basic, des interruptions logicielles grâce à l'instruction :

EVERY <delai> [,<N° de chronomètre>] GOSUB < N° ligne> delai est le temps au bout duquel le sous-programme est appelé une nouvelle fois, et exprimé en multiples de 50 millisecondes.

Nous allons donc placer dès le début de notre programme Basic cette instruction, qui permettra de l'interrompre régulièrement, et d'effectuer un sous-programme permettant le traitement du test de l'activité **CAPS LOCK** (on se reportera avec intérêt à la description et l'exemple donné sur cette instruction, à la Partie 4 chapitre 1.2 page 54) ; nous placerons ce sous-programme en fin de programme, par exemple en lignes 60000 et suivantes.

Nous choisirons, par exemple, d'effectuer le traitement toutes les 2 secondes, afin de ne pas trop ralentir le programme en cours. La valeur de **delai** sera donc égale à :

$$delai = 2 / (50 * 10^{-3}) = 40$$

Ce qui nous amène à la ligne d'instruction :

EVERY 40,2 GOSUB 60000.

Ainsi, toutes les deux secondes, si CAPS LOCK est active, nous viendrons écrire grâce à l'instruction **LOCATE** en haut et à droite de l'écran.

Deux autres problèmes subsistent tout de même :

1/ Le curseur texte doit revenir à sa position initiale, quelle que soit celle-ci suite à cet affichage.

2/ Les coordonnées d'affichage ne sont pas identiques selon le mode d'affichage choisi.

Le premier problème est résolu grâce à une autre adresse mémoire, dans laquelle est enregistrée la position du curseur texte : **&B726** pour les CPC 664 et CPC 6128.

Il nous suffira donc de sauvegarder temporairement le contenu de cette adresse dès le début de la routine, et de le restituer quand la routine sera exécutée. — Vous pouvez essayer, par quelques POKEs hasardeux (POKE &B726,xx) de modifier directement la position du curseur de texte, en mode direct.

Le deuxième problème nous préoccupe surtout si nous ne connaissons pas par avance le mode graphique dans lequel notre routine va opérer. Heureusement, la RAM de votre CPC renferme encore ce précieux renseignement à l'adresse hexadécimale &B7C3.

Nous lirons donc le contenu de cette adresse par l'instruction :

Variable = PEEK (&B7C3)

— Vous pouvez de même essayer la commande :

PRINT PEEK (&B7C3) en mode direct, et selon les trois modes écran choisis.

Une autre particularité de cette adresse est qu'elle permet de modifier le mode courant au niveau de l'affichage. Si, par exemple, vous vous trouvez en mode 0 et que vous forcez l'affichage en mode 1 par la commande : **POKE &B7C3, 1** vous obtiendrez alors un style d'affichage quelque peu bizarre, que certains concepteurs de logiciels de jeux utilisent quelquefois.

Afin de ne pas trop nous écarter du sujet qui nous préoccupe, nous vous laisserons le soin d'expérimenter cette particularité.

APPLICATION AU PROGRAMME

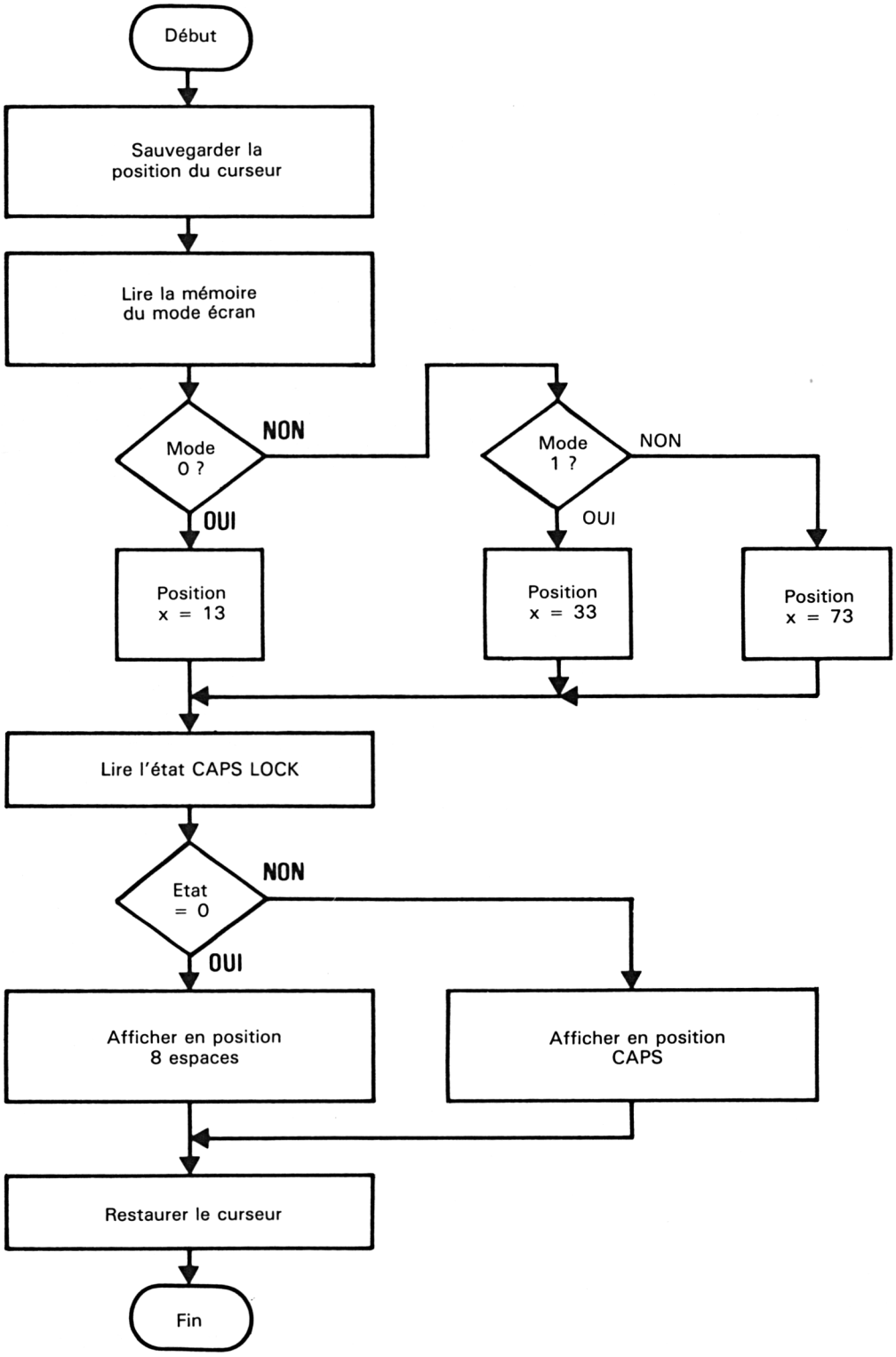
Les différentes positions que nous avons choisies sont ainsi, selon le mode :

— MODE 0 → LOCATE 13,1

— MODE 1 → LOCATE 33,1

— MODE 2 → LOCATE 73,1

Nous obtiendrons l'algorigramme de fonctionnement suivant :



La traduction en basic de cet algorigramme donne le programme suivant :

```
1 EVERY 40,2 GOSUB 60000
.
.
.
100 REM *** INSEREZ ENTRE LES LIGNES ***
110 REM ***        1  ET  60000        ***
120 REM ***  DE VOTRE PROGRAMME QUI     ***
130 REM ***    FONCTIONNERA SOUS        ***
140 REM ***  INTERRUPTION LOGICIELLE ***
.
.
.
.
60000 REM *** ROUTINE CAPS LOCK ***
60005 REM ************************
60010 TXTcurseur = PEEK(&B726):REM sauve
garde de la position curseur
60020 SCRmode = PEEK(&B7C3):REM lecture
du mode ecran
60030 KMcaps = PEEK(&B632):REM lecture d
e l'etat CAPS LOCK
60040 IF SCRmode = 0 THEN X = 13 : GOTO
60070:REM mode 1?
60050 IF SCRmode = 1 THEN X = 33 : GOTO
60070:REM mode 2?
60060 X = 73:REM alors mode 2
60070 IF KMcaps = 0 THEN 60110:REM test
CAPS LOCK
60080 LOCATE X,1:REM positionne
60090 PRINT CHR$(&8F);CHR$(&8F);"CAPS";C
HR$(&8F);CHR$(&8F)
60100 GOTO 60130
60110 LOCATE X,1:REM positionne
60120 PRINT SPACE$(8):REM efface CAPS
60130 POKE &B726,TXTcurseur:REM restitue
 l'ancienne position curseur
60140 RETURN:REM fin d'interruption
60150 REM ************************
```

Vous remarquerez les variables utilisées :

— **TXTcurseur** : curseur texte - TeXTe curseur

— **SCRmode** : mode écran — SCReen mode

— **KMcaps** : touche CAPS LOCK — Key Manager caps

— **X** : variable de position dans l'axe X de l'écran et veillerez à ne pas les utiliser dans le programme que vous insérerez entre les lignes 1 et 60000.

Le numéro de chronomètre choisi en ligne 1 est le chronomètre 2 qui vous laisse le choix d'utiliser les chronomètres 0 et 1, qui sont plus prioritaires, ainsi que le chronomètre 3, moins prioritaire. Ce numéro n'étant pas important, vous pouvez le modifier à votre gré.

Vous pouvez aussi faire varier le temps entre deux appels du sous-programme, mais en sachant que si vous le diminuez fortement, vous risquez de ralentir considérablement le programme à exécuter, et, dans le cas contraire, vous ne verriez pas apparaître le message **CAPS** assez souvent, notamment en cas de « scrolling » de l'écran, ou d'effacement.

UTILISATION DU SOUS-PROGRAMME

Nous vous conseillons de sauvegarder le sous-programme tel quel, sans les lignes 100 à 140, sur disquette de préférence, et en ASCII par l'instruction : SAVE "CAPS.ASC", A.

Vous pourrez ensuite l'intégrer à n'importe quel programme basic que vous possédez, et le charger dans la mémoire de votre CPC, grâce à la commande :

MERGE "CAPS.ASC" ou CHAIN MERGE "CAPS.ASC"

LIMITES DU SOUS-PROGRAMME EN BASIC

Ce type de sous-programme fonctionnant en interruption dite *interruption logicielle* est particulièrement adapté lorsque le programme est exécuté, mais non lorsque votre Amstrad attend une commande, comme, par exemple, lorsque vous frappez un listing. La solution est alors d'entrer au cœur même du fonctionnement de votre CPC et d'en détourner certaines de ses tâches, comme nous allons vous l'expliquer ci-après.

L'interruption matérielle de l'Amstrad

Contrairement à ce que vous êtes en droit de croire, le moniteur Basic faisant parfaitement illusion, le microprocesseur (le Z 80) est continuellement en train de travailler : grâce aux composants qui l'entourent et aux programmes en ROMs, il effectue entre autre la remise à jour du compteur **TIME** tous les 300e de seconde, la scrutation du clavier tous les 50e de seconde, et une somme inimaginable d'exécutions de routines internes au système d'exploitation centrale et à l'interpréteur Basic ; seul le contrôleur de visualisation **CRTC 6845** se permet d'interrompre le Z 80 (pour les plus connaisseurs d'entre vous : en positionnant la broche WAIT) pour remettre à jour l'écran.

Pour effectuer ces multiples opérations, le microprocesseur est interrompu par un signal, provenant du composant **GATE ARRAY** (composant développé spécialement par Amstrad pour les CPC), tous les 300e de seconde, et appliqué sur sa broche IRQ.

Ce signal force le Z 80 à se brancher à l'adresse (en mémoire RAM) hexadécimale &0038 — l'interruption est nommée RST 7 (ReSTart 7) (voir Partie 4, chapitre 2.8, page 4).

Par quelques POKEs à partir de cette adresse et les suivantes ou le désassemblage suivant :

```
0030  RST  0              C7          -
0031  EXX                 D9          -
0032  LD   HL,&002B       212B00      !+-
0035  LD   (HL),C         71          q
0036  JR   &0040          1808        --
0038  JP   &B941          C341B9      -A-
003B  RET                 C9          -
003C  NOP                 00          -
003D  NOP                 00          -
003E  NOP                 00          -
003F  NOP                 00          -
```

Adresse **Code Opérations** **Code hexadécimal des codes opérations** **Code ASCII s'il y a lieu**

Si vous ne possédez pas de désassembleur, vous pouvez obtenir les valeurs hexadécimales par l'instruction :

PRINT HEX$(PEEK(&0038)... et suivantes.

A ces adresses, nous trouvons un saut à l'adresse &B941 (JP B941, codé en hexadécimal : C3 41 B9).

A partir de cette dernière adresse est effectué le traitement de l'interruption IRQ, dont voici une partie désassemblée :

```
B941  DI                  F3          -
B942  EX   AF,AF          08          -
B943  JR   C,&B978        3833        83
B945  EXX                 D9          -
B946  LD   A,C            79          y
B947  SCF                 37          7
B948  EI                  FB          -
B949  EX   AF,AF          08          -
B94A  DI                  F3          -
B94B  PUSH AF             F5          -
B94C  RES  2,C            CB91        --
B94E  OUT  (C),C          ED49        -I
B950  CALL &00B1          CDB100      ---
B953  OR   A,A            B7          -
B954  EX   AF,AF          08          -
B955  LD   C,A            4F          O
```

```
B956  LD   B,&7F          067F
B958  LD   A,(&B831)      3A31B8     :1-
B95B  OR   A,A            B7         -
B95C  JR   Z,&B972        2814       (-
B95E  JP   M,&B972        FA72B9     -r-
```

Dans cette routine sont, entre autres, sauvegardés les registres du Z 80, et effectuée la commutation de la ROM BIOS du système d'exploitation, située entre l'adresse &0000 et &3FFF à la place de la RAM (on se reportera à l'organisation mémoire des CPCs (Partie 2 chapitre 1 page 3).

La commutation effectuée, un sous-programme à l'adresse &00B1 est appelé dans cette ROM, par l'instruction située à l'adresse &B950 : **CALL &00B1** (à peu près équivalent du GOSUB en Basic).

Comme cette instruction se trouve en RAM, il est possible de modifier l'adresse à laquelle elle saute, pour y mettre l'adresse d'une de nos routines en langage machine ; l'important étant d'effectuer à la fin de celle-ci un saut direct à l'adresse de la ROM.

Ces explications commençant à devenir ardues et impliquant des connaissances en langage machine, nous vous conseillons de vous reporter à la Partie, chapitre 2 traitant du langage d'assemblage, et aux instructions du Z 80.

Rassurez-vous tout de même, si vous ne possédez pas d'assembleur, un listing basic permettant de charger la routine en code machine vous sera proposé à la fin du chapitre.

Dérouter l'interruption

Pour dérouter la routine d'interruption, nous allons donc écrire aux adresses &B951 et &B952 *(attention :* ces adresses ne sont valables que pour les CPC 664 et CPC 6128) le numéro de l'adresse où débutera notre routine de traitement. Il serait, par ailleurs, préférable auparavant, d'inhiber toute nouvelle interruption lors de l'installation, et de les autoriser à nouveau à la sortie de la routine. Vous trouverez l'ordinogramme général de cette routine en page 12.

Une fois l'adresse installée, au maximun un 300e de seconde plus tard, un branchement à la nouvelle adresse sera effectué, vous comprendrez donc qu'il est nécessaire d'avoir chargé auparavant en mémoire notre sous-programme de détection et d'affichage de l'état **CAPS LOCK.**

La routine Assembleur de détection CAPS LOCK

PREPARATION DU RETOUR EN FIN DE ROUTINE

A la fin de la routine, il nous faut retourner à l'adresse que nous avons détournée, c'est-à-dire en &00B1, et pour cela, deux styles de programmes s'ouvrent à nous :

- Une façon très brutale de retourner à cette adresse est d'y effectuer un branchement inconditionnel appelé JumP, par l'instruction suivante :
  JP &00B1 ;
- Une façon plus astucieuse puisqu'elle nous permet de terminer la routine de la même façon qu'une routine machine classique, c'est-à-dire par l'instruction **RET** (de RETurn), ce qui nous semble préférable.

Il faut savoir que l'instruction **RET** prend la dernière adresse contenue dans la pile (pile repérée par un registre du Z 80 appelé SP, de Stack Pointer = pointeur de pile) et la place dans le compteur programme, c'est-à-dire qu'elle effectue un branchement à cette adresse. (Lorsque l'on appelle un sous-programme en Assembleur, c'est l'adresse de retour qui est mise dans la pile — on dit aussi empilée — donc, lors d'un **RET,** on retourne à cette adresse.)

Nous allons ainsi empiler l'adresse &00B1, mais les instructions du microprocesseur Z 80 ne permettent pas de placer directement dans la pile, par contre les registres de travail sont empilables.

Pour cela, il nous faut charger un registre 16 bits (par exemple HL avec cette adresse, et empiler HL, grâce aux instructions qui suivent :
LD HL, &00B1
PUSH HL.

L'ALGORITHME

Nous avons en partie retenu la forme de l'algorigramme proposée plus haut pour réfléchir au programme assembleur.

En revanche, contrairement à l'instruction d'interruption **EVERY** du Basic, l'interruption IRQ n'est pas modifiable logiciellement sans causer de graves perturbations (mais non destructives, rassurez-vous) dans le fonctionnement de votre CPC. Nous n'allons pas laisser le CPC afficher 300 fois par secondes (c'est-à-dire environ toutes les 3.3 millisecondes) le message **CAPS LOCK,** ce serait une perte de temps inutile, qui retarderait le fonctionnement de l'ordinateur, et absurde, car personne n'est capable de modifier l'état de cette touche aussi rapidement.

Nous utiliserons un compteur pour l'affichage, qui sera réalisé toutes les &FF fois (contenu maximal du compteur), c'est-à-dire toutes les $((15 \times 16) + 15) \times 3{,}3 = 841{,}5$ ms, proche de 1 seconde.

Notre programme devenant de plus en plus complexe, il est nécessaire de le structurer, donc d'en écrire d'abord l'algorithme que nous vous proposons ci-dessous :

DEBUT

Installer la routine de détournement de l'interruption matérielle $\overline{IRQ}$

FIN

DEBUT

CO

Routine détournée par le traitement précédent.
C'est elle qui sera appelée régulièrement lors d'une interruption matérielle $\overline{IRQ}$

FINCO

Charger l'adresse de retour dans la pile

Tester l'état de **CAPS LOCK**

CO

Cet état est contenu dans une adresse en RAM

FINCO

SI **CAPS LOCK** est activée

ALORS

Décrémenter le compteur d'affichage

CO

Afin de régénérer régulièrement l'affichage de **CAPS LOCK** (environ toutes les secondes)

FINCO

SI le compteur d'affichage a atteint la valeur 1

ALORS

Positionner à 1 le flag signalant l'affichage de **CAPS LOCK**

Afficher sur l'écran le message **CAPS**

CO

Afficher sera effectué par un sous-programme

FINCO

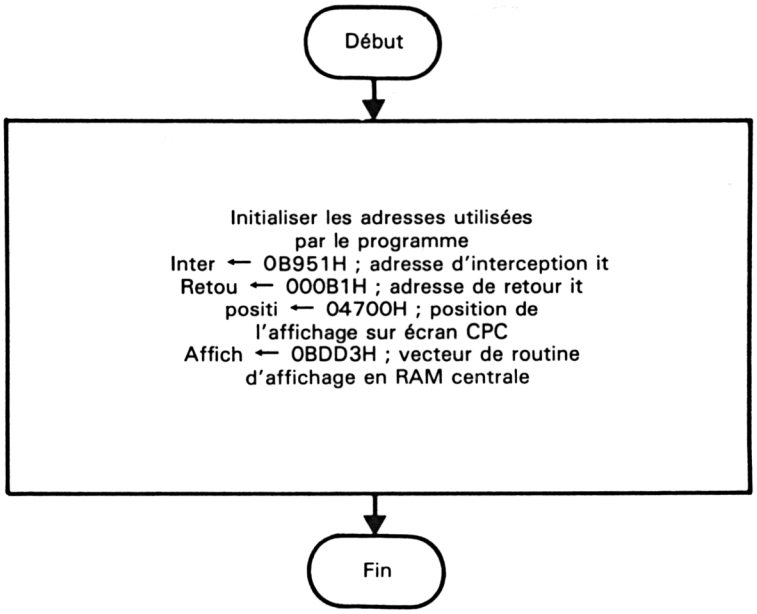
FINSI

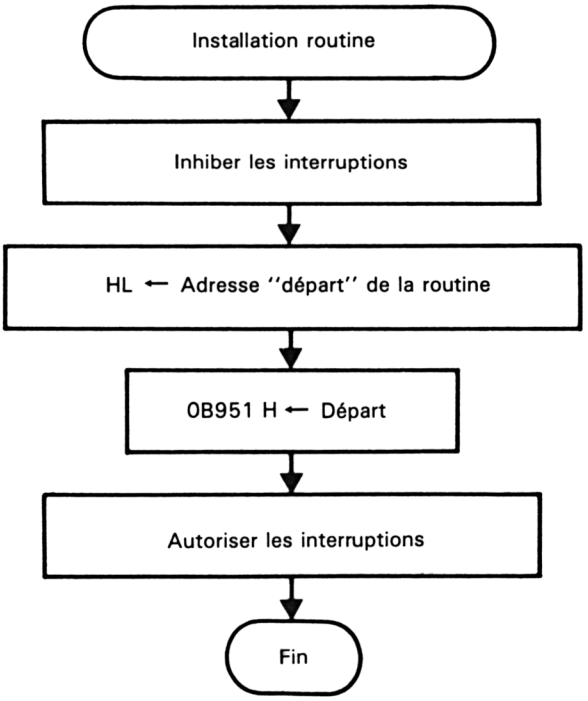
```
        SINON
                Tester l'état du flag signalant l'état de l'affichage de CAPS LOCK
                SI le flag est positionné à 1
                                CO
                                flag à 1 si CAPS LOCK est affiché
                                FINCO
                  ALORS
                  Annuler le flag d'affichage
                  Effacer le message CAPS
                FINSI
        FINSI
FIN
        CO
        Retour à la suite du traitement de la routine d'interruption
        FINCO
```

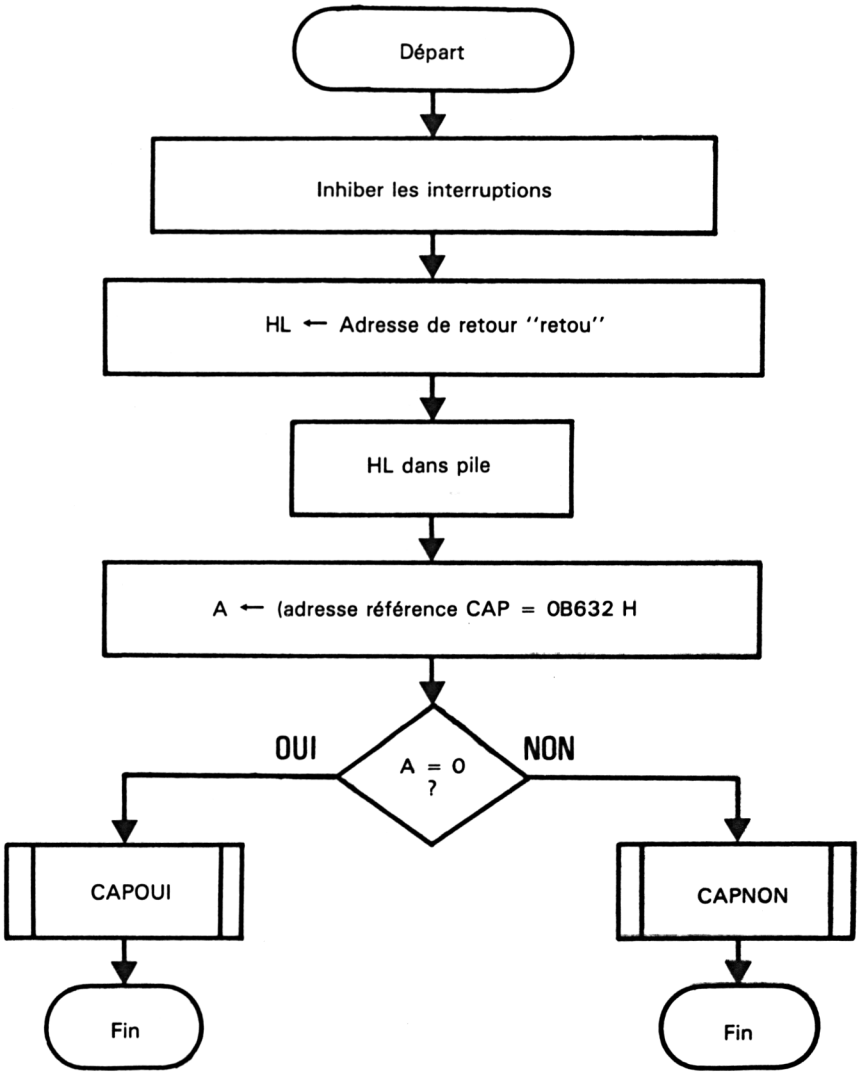
L'ORDINOGRAMME

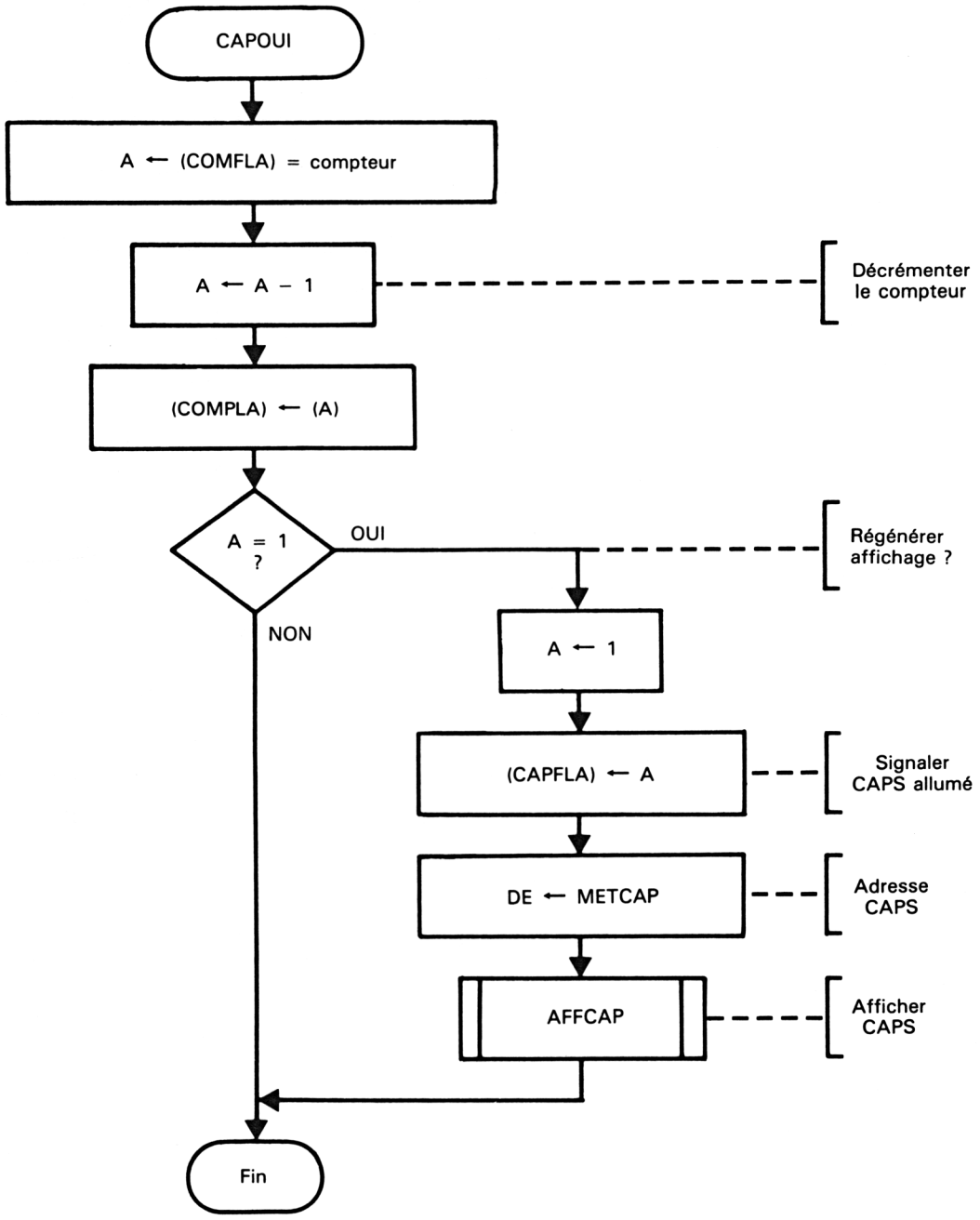
Nous vous proposons l'ordinogramme (page suivante), qui est la représentation graphique de l'algorithme précédent, faisant apparaître le langage spécifique au microprocesseur utilisé.

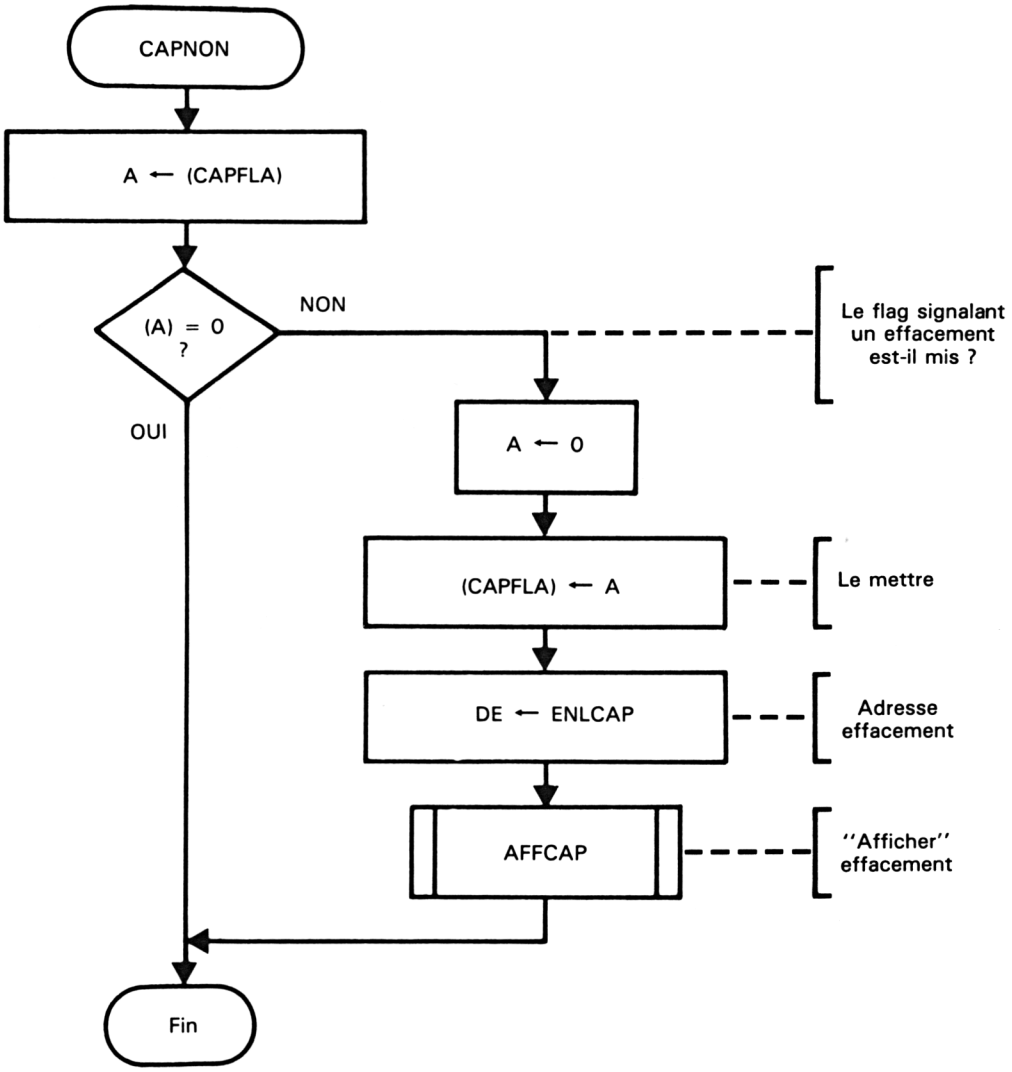
On remarquera dans cet ordinogramme l'utilisation de symboles de sous-programme (les rectangles avec deux doubles barres verticales), qui renvoient, soit à un sous-programme (dans le cas de l'affichage : **AFFCAP**) ou à un autre programme, pour éclaircir la présentation (dans le cas des traitements **CAPOUI** et **CAPNON**).

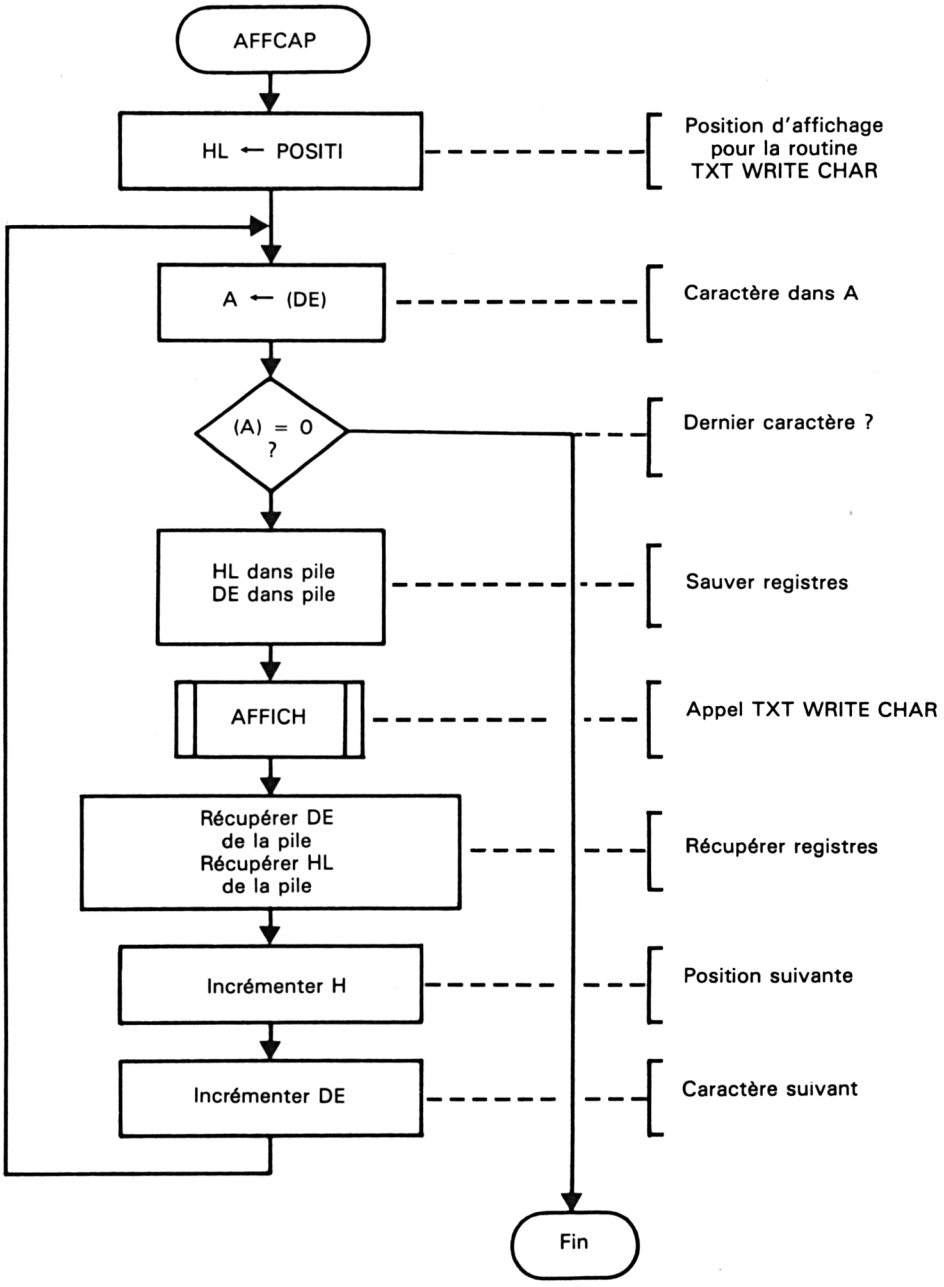

Début
Initialiser les adresses utilisées
par le programme
Inter ← 0B951H ; adresse d'interception it
Retou ← 000B1H ; adresse de retour it
positi ← 04700H ; position de
l'affichage sur écran CPC
Affich ← 0BDD3H ; vecteur de routine
d'affichage en RAM centrale
Fin


Installation routine
Inhiber les interruptions
HL ← Adresse "départ" de la routine
0B951 H ← Départ
Autoriser les interruptions
Fin


Départ
Inhiber les interruptions
HL ← Adresse de retour ''retou''
HL dans pile
A ← (adresse référence CAP = 0B632 H
OUI
A = 0
?
NON
CAPOUI
CAPNON
Fin
Fin


CAPOUI
A ← (COMFLA) = compteur
A ← A – 1
Décrémenter le compteur
(COMPLA) ← (A)
A = 1 ?
OUI
Régénérer affichage ?
NON
A ← 1
(CAPFLA) ← A
Signaler CAPS allumé
DE ← METCAP
Adresse CAPS
AFFCAP
Afficher CAPS
Fin


CAPNON
A ← (CAPFLA)
(A) = 0 ?
NON
OUI
Le flag signalant un effacement est-il mis ?
A ← 0
(CAPFLA) ← A
Le mettre
DE ← ENLCAP
Adresse effacement
AFFCAP
"Afficher" effacement
Fin



LE PROGRAMME ASSEMBLEUR

Nous vous livrons ainsi le programme Assembleur qui découle de l'ordinogramme précédent.

Les non-initiés noteront la signification des colonnes de gauche à droite :

— les numéros de lignes (de 1 à 147) qui servent uniquement de repères lors de l'écriture du programme ;

— les adresses (de A000 à A067) qui sont générées à la compilation du programme, grâce à l'origine fixée par l'ordre origine : **ORG 0A000H** ;

— les codes machine exécutables (codes hexadécimaux) qui sont générées à la compilation à partir des mnémoniques ;

— les étiquettes (suivies du signe :) ;

— les mnémoniques (LD A,xx : CP xx ; JR yyyy ; ...) qui sont les instructions symbolisées de l'assembleur ;

— les commentaires qui sont précédés du caractère ; et qui servent à donner des précisions sur les opérations effectuées (équivalent de l'instruction REM du Basic). Nous tenons, et espérons que vous ferez de même, à faire apparaître ces commentaires, car un programme en Assembleur est difficilement lisible sans commentaires, si vous désirez le réétudier ultérieurement.

```
 1                ;PROGRAMME D'IMPLANTATION
 2                ;D'UNE ROUTINE DETOURNANT
 3                ;LA ROUTINE DE TRAITEMENT
 4                ;D'INTERRUPTION (IT) IRQ
 5                ;************************
 6                ;
 7                ;
 8                ;TABLE DE REFERENCE
 9                ;******************
10                ;
11                ;ADRESSE ROUTINE DETOURNEE
12                INTER:       EQU  0B951H
13                ;ADRESSE RETOUR EN ROM INFERIEURE
14                RETOU:       EQU  000B1H
15                ;POSITION D'AFFICHAGE SUR L'ECRAN
16                POSITI:      EQU  04700H
17                ;VECTEUR D'AFFICHAGE EN RAM
18                ;(TXT WRITE CHAR)
19                AFFICH:      EQU  0BDD3H
20                ;ADRESSE DE REFERENCE ACTIVITE CAP
21                MEMCAP:      EQU  0B632H
22                ;**********************************
23                ;
24                ;
25                ;ADRESSE DE REFERENCE D'ASSEMBLAGE
26                ;**********************************
27                ;
28                             ORG  0A000H
29                             LOAD 0A000H
30                ;*************
31                ;
32                ;
33                ;ROUTINE D'INSTALLATION POUR
34                ;DETOURNEMENT DE LA ROUTINE
35                ;DE TRAITEMENT D'INTERRUPTION
36                ;****************************
37                ;
38 A000 F3                     DI                    ;INHIBE LES INTERRUPTIONS
39 A001 2109A0                 LD   HL,DEPART        ;CHARGE L'ADRESSE DE
40 A004 2251B9                 LD   (INTER),HL       ;LA ROUTINE D'IT
41 A007 FB                     EI                    ;AUTORISE LES INTERRUPTIO
42 A008 C9                     RET                   ;ROUTINE INSTALLEE -> FIN
```

```
                        ;****************************
                        ;
                        ;
                        ;PROGRAMME DE TEST DE L'ETAT
                        ;CAPS LOCK ET AFFICHAGE EN
                        ;CONSEQUENCE EN INTERRUPTION
                        ;****************************
                        ;
;ROM E                  DEPART:;PREPARATION AU RETOUR EN
                        ;ROM EN FIN DE TRAITEMENT
A009 F3                         DI                      ;INHIBITION DES INTERRUPT
A00A 21B100                     LD    HL,RETOU          ;SAUVEGARDE DANS PILE
A00D E5                         PUSH HL                 ;DE L'ADRESSE DE RETOUR E
                        ;ROM AFIN DE TERMINER SUR 'RET'
                        ;
A00E 3A32B6             DEBUT:  LD    A,(MEMCAP)        ;ETAT CAPSLOCK
A011 FE00                       CP    00                ;ACTIVE EN RAM?
A013 2818                       JR    Z,CAPNON          ;SAUT SI NON ACTIVE...
                        ;
LD62,(                  CAPOUI:;CAPS LOCK A ETE ACTIVE
A015 3A66A0                     LD    A,(COMFLA)        ;COMPTEUR AFFICHAGE
A018 3D                         DEC   A                 ;EST DECREMENTE
A019 3266A0                     LD    (COMFLA),A        ;ET A NOUVEAU SAUVE
A01C FE01                       CP    01H               ;EST-IL EGAL A 1
A01E 200B                       JR    NZ,CAPMIS         ;NON -> PAS AFFICHAGE
A020 3E01                       LD    A,01H             ;CHARGER LE FLAG CAPS POU
A022 3265A0                     LD    (CAPFLA),A        ;SIGNALER L'AFFICHAGE
                        ;DE CAPS LOCK EFFECTUE
A025 1153A0                     LD    DE,METCAP         ;CHARGER DE AVEC
                        ;L'ADRESSE DE DEBUT DU MESSAGE
                        ;CAPS LOCK ENFONCE
A028 CD41A0                     CALL  AFFCAP            ;AFFICHER MESSAGE
A02B 183A               CAPMIS: JR    RETOUR            ;SAUT A FIN
                        ;
                        ;CAPS LOCK NON ACTIVE EN RAM
A02D 3A65A0             CAPNON: LD    A,(CAPFLA)        ;CHARGE FLAG
A030 FE00                       CP    00H               ;FLAG DEJA EFFACE?
A032 280B                       JR    Z,CAPEFF          ;SI OUI -> SAUT
A034 3E00                       LD    A,00              ; SI NON -> POSITIONNER
A036 3265A0                     LD    (CAPFLA),A        ;A ZERO LE FLAG
A039 115CA0                     LD    DE,ENLCAP         ;CHARGER DE AVEC
                        ;L'ADRESSE DE DEBUT DU MESSAGE
                        ;'ESPACES' D'EFFACEMENT
A03C CD41A0                     CALL  AFFCAP            ;AFFICHER MESSAGE
A03F 1826               CAPEFF: JR    RETOUR            ;SAUT A FIN
                        ;
                        ;
                        ;AFFICHAGE D'UN MESSAGE DONT
                        ;L'ADRESSE DU PREMIER CARACTERE
                        ;EST EN (DE) ET LA POSITION
                        ;CHARGEE DANS HL
                        ;
A041 210047             AFFCAP: LD    HL,POSITI         ;POSITION ECR.
A044 1A                 AFFCA1: LD    A,(DE)            ;CHARGER CARACTER
A045 FE00                       CP    00                ;FIN DU MESSAGE?
A047 C8                         RET   Z                 ;SI OUI -> RETOUR
A048 E5                         PUSH HL                 ;SAUVER HL
A049 D5                         PUSH DE                 ;SAUVER DE
A04A CDD3BD                     CALL  AFFICH            ;EXECUTER LA ROUTINE
                        ;D'AFFICHAGE EN ROM, DONT LE
                        ;VECTEUR EST EN RAM
A04D D1                         POP   DE                ;RECUPERER DE
```

```
105 A04E E1                     POP  HL                 ;RECUPERER HL
106 A04F 24                     INC  H                  ;INCREMENTER POUR POSITIO
107                 ;SUIVANTE
108 A050 13                     INC  DE                 ;INCREMENTER DE POUR CARA
109                 ;TERE SUIVANT A AFFICHER
110 A051 18F1                   JR   AFFCA1             ;RECOMMENCER EN AFFCA1
111                 ;
112                 ;
113                 ;MESSAGE 'CAPS' + VIDEO INVERSE
114                 ;*******************************
115                 ;
116 A053 8F8F4341 METCAP:      DEFB 08FH,08FH,'CAPS'
116 A057 5053
117 A059 8F8F00                 DEFB 08FH,08FH,00
118                 ;
119                 ;
120                 ;MESSAGE ESPACES POUR PERMETTRE
121                 ;L'EFFACEMENT DU MESSAGE 'CAPS'
122                 ;*******************************
123                 ;
124 A05C 20202020 ENLCAP:      DEFB 020H,020H,020H,020
125 A060 20202020              DEFB 020H,020H,020H,020
125 A064 00
126                 ;
127                 ;
128                 ;FLAG D'ETAT DE L'AFFICHAGE 'CAP'
129                 ;*******************************
130                 ;
131 A065 00         CAPFLA:     DEFB 00
132                 ;
133                 ;
134                 ;COMPTEUR DU TEMPS DE REAFFICHAGE
135                 ;*******************************
136                 ;
137 A066 00         COMFLA:     DEFB 00H
138                 ;
139                 ;
140                 ;FIN -> RETOUR A LA SUITE DE LA
141                 ;ROUTINE D'IT SITUEE EN ROM
142                 ;*******************************
143                 ;
144 A067 C9         RETOUR:     RET
145                 ;
146                 ;
147                             END                     ;FIN DE LISTE ASSEMBLEUR
```

Après compilation et sauvegarde, vous pourrez utiliser ce programme à partir du Basic en frappant :

MEMORY &9FFF : LOAD "CAP.BIN" : CALL &A000

Note :

Certains assembleurs, tel l'Assembleur DEVPAC présenté Partie 4, chapitre 2.6, préconisent le caractère # en tant que préfixe des données hexadécimales, le nôtre préconisant le **0** (zéro) plus le caractère **H** en suffixe.

LE CHARGEUR BASIC DES CODES MACHINES

Pour ceux d'entre vous qui sont intéressés par l'utilisation de cette routine et qui ne possèdent pas d'Assembleur, nous vous donnons ci-dessous le chargeur Basic :

```
10 REM ***  INSTALLATION EN BASIC  ***
20 REM ***  DE LA ROUTINE MACHINE  ***
30 REM *** DE TRAITEMENT CAPS LOCK ***
40 REM ********************************
50 FOR ADRESSE = &A000 TO &A067:REM adre
sses de chargement
60 READ DONNEE$:REM lecture des codes ma
chines
70 DONNEE=VAL("&"+DONNEE$):REM transform
ation hexa
80 SOMME = SOMME + DONNEE:REM somme des
codes machines
90 POKE ADRESSE,DONNEE:REM chargement ef
fectif d'un code
100 NEXT:REM code suivant
110 READ CONTROLE:REM lecture de la somm
e de controle
120 IF CONTROLE = SOMME THEN 140:REM ver
ification
130 MODE 2:PRINT "ERREUR DANS LES LIGNES
 DE DATAs":LIST:REM erreur
140 MODE 2:PRINT "SAUVEGARDE PAR :":REM
chargement ok
150 PRINT "SAVE "+CHR$(34)+"CAP.BIN"+CHR
$(34)+",B,&A000,&A068-&A000":REM recomme
ndations d'utilisation
160 PRINT:PRINT
170 PRINT"UTILISATION PAR :"
180 PRINT"MEMORY &9FFF : LOAD "+CHR$(34)
+"CAP.BIN"+CHR$(34)+",&A000 : CALL &A000
"
190 REM ********************************
200 REM *** codes operations a charger *
**
210 DATA F3,21,09,A0,22,51,B9,FB
220 DATA C9,F3,21,B1,00,E5,3A,32
230 DATA B6,FE,00,28,18,3A,66,A0
240 DATA 3D,32,66,A0
250 DATA FE,01,20,0B,3E,01,32,65
260 DATA A0,11,53,A0,CD,41,A0,18
270 DATA 3A,3A,65,A0,FE,00,28,0B
280 DATA 3E,00,32,65,A0,11,5C,A0
290 DATA CD,41,A0,18,26,21,00,47
300 DATA 1A,FE,00,C8,E5,D5,CD,D3
```

```
310 DATA BD,D1,E1,24,13,18,F1,8F
320 DATA 8F,43,41,50,53,8F,8F,00
330 DATA 20,20,20,20,20,20,20,20
340 DATA 00,00,00,C9
350 DATA 10091
360 REM ********************************
```

Les contenus des lignes de DATAs correspondent aux codes hexadécimaux résultant de la compilation.

Nous vous conseillons, avant toute utilisation de ce programme de le sauvegarder sur disquette.

Lors du lancement, si une erreur de frappe des DATAs a été commise, le programme vous redonne le listing complet pour corriger.

Suite au succès de chargement des codes machines, le programme vous indique la méthode de sauvegarde de la routine ainsi créée (l'instruction **MEMORY &9FFF** sert à protéger le sous-programme d'un éventuel débordement d'un programme ou des variables Basic sur cette zone d'adresses).