4

Programmation appliquée

Les programmes-assembleur qui vont suivre sont de plus en plus évolués, partant du plus simple vers le plus compliqué. A chaque version pour TO7(70) et TO9 correspondra celle pour MO5, avec les explications particulières qui s'y rattachent. Chaque programme pourra être transposé en basic, au moyen de la structure décrite précédemment, que nous rappellerons une dernière fois lors du programme n° I. Vous pourrez ensuite travailler vous-même en basic, en utilisant les codes du programme-objet, dans la marge à gauche du programme-assembleur.

L'adresse d'implantation d'un programme est donnée, rappelons-le, par la directive ORG, pratiquement obligatoire en langage d'assembleur. L'adresse de fin, qui vous rend la main, est celle du SWI sur TO7(70) et TO9, ou STOP sur MO5, que vous remplacerez en basic par RTS = &H39. La directive END définit la fin du programme ; elle est obligatoire en assembleur.

* *Notions d'ordre général* :

Les moniteurs des TO7(70), TO9 et MO5, contiennent chacun les routines de gestion de leur appareil. Celles-ci, écrites en langage-machine, servent à la cartouche MEMO-7 ou aux basic résidents, par exemple.

La programmation-machine se pratique avec ces mêmes routines, auxquelles on accède par des points d'entrée. Chaque routine possède son point d'entrée, qui représente l'adresse de début de celle-ci. Sur TO7(70) et TO9, les points d'entrée sont directement utilisables, alors que sur MO5 il est nécessaire de passer par des codes intermédiaires.

Les paramètres de sortie sont les données éventuelles, de retour, que l'on retrouve dans certains registres (du processeur ou de la R.A.M.), lorsqu'une routine a été exécutée. S'il existe des paramètres de sortie, il existe aussi des paramètres d'entrée, qui sont les données à mettre dans certains registres (du processeur ou de la R.A.M.), avant d'accéder à la routine.

En résumé :

— Sur TO7(70) et TO9, un point d'entrée représente l'adresse d'une routine. Il porte un nom, pour être mieux mémorisé.

— Sur MO5, un point d'entrée est représenté par un code qui permet l'accès à une routine. Nous donnerons à ce code le même nom qu'au point d'entrée correspondant sur TO7(70) et TO9.

Par extension, nous appellerons les routines du nom de leur point d'entrée.

Pour accéder à une routine, on peut utiliser divers mnémoniques, dont le JSR et le JMP.

a. JSR :

Cette instruction signifie : JUMP TO SUBROUTINE. Elle permet de conserver l'état du registre PC dans la pile-système, avant l'accès à une routine. C'est le GOSUB de l'assembleur.

Lorsque la routine a fini son travail, elle se termine par RTS = retour de SUBROUTINE, qui permet la restitution du ''PC'' contenant l'adresse de retour ; c'est le RETURN de l'assembleur.

En général les routines du moniteur conservent tous les registres du processeur (sauf ''PC''), pour les restituer une fois leur tâche accomplie.

Sur MO5, la directive d'assemblage CALL suivie d'un code de routine, ou du nom de celle-ci (défini au préalable), a le même effet que JSR suivi de l'adresse de la routine ou de son nom (défini auparavant).

JSR doit être utilisé avec un adressage étendu (sans '' # '').

Sur MO5 toujours, on peut réaliser un JSR également en utilisant SWI, suivie sur la ligne du dessous de FCB + code de la routine (cartouche TO TEK International). L'interruption SWI permet le chargement du code de routine, et le ''PC'' se branche alors en $F63C*. SWI ne peut être suivie directement de '' # '' + numéro de code, à la différence du

*$F63C : adresse de la routine d'interruption SWI, contenue en $FFFA-$FFFB

logiciel ODIN. FCB n'admet pas non plus le " # ", puisqu'il s'agit d'une directive d'assemblage.

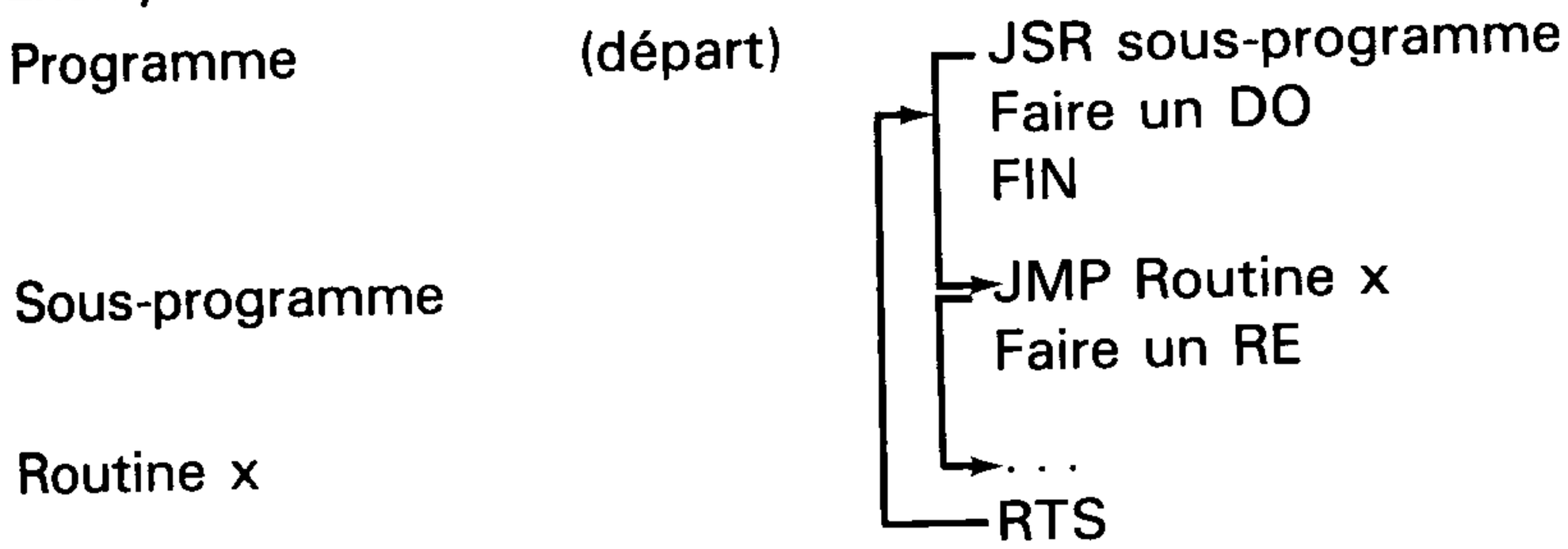
b. JMP :

Cette instruction signifie : JUMP. Elle doit être suivie de l'adresse de la routine ou de son nom (défini au préalable). Sur MO5, en forçant à 1 le bit n° 7 du code d'accès à la routine, avec CALL ou SWI plus FCB, on réalise un JMP au lieu d'un JSR.

Ici, il n'y a pas de retour possible ; c'est le GOTO de l'assembleur.

Toute routine finissant cependant par RTS, le retour peut éventuellement se faire à l'instruction suivant un JSR précédent.

Exemple :



L'ordinateur, dans cet exemple, exécutera la routine x, puis fera un DO, mais pas de RE. Il est donc préférable d'utiliser JSR pour éviter les erreurs.

Nos programmes-machine, appelés par le basic, se terminent aussi par RTS. On peut donc dire que EXEC en basic agit comme un JSR en langage-machine. Nous apprendrons par la suite qu'il est cependant nécessaire, dans ce cas, de préserver l'état des registres du processeur, au début des programmes-machine.

c. Notes pour MO5 :

— En exécutant la routine de code n° 0 :

Machine
(après assemblage (A)) :

```
7D00
7D00  3F
7D01  0
7D02  BD  B000
          0000
```

Assembleur :

```
ORG   &32000
SWI
FCB   0
STOP
END
```

Basic :

```
10  POKE 32000, &H3F
20  POKE 32001, 0
30  POKE 32002, &H39
40  EXEC 32000
```

Vous obtiendrez :

1. en assembleur :

un retour à la page d'en-tête, sans perte du listing (faites après assemblage : G & 32000 et ENTREE dans le moniteur).

2. en basic (où &H39 remplace "BD B000") :

un retour à une page vierge, sans perte du listing.

JSR et JMP sont utilisables sur MO5, pour se brancher à un sous-programme dont on connaît l'adresse.

On peut aussi se servir de ces instructions de branchement pour accéder à une routine en R.O.M., à condition de connaître son adresse et ses éventuels paramètres d'entrée.

d. Notes pour TO7(70) et TO9 :

SWI n'accède pas aux routines comme sur MO5. Il s'agit d'une interruption logicielle rendant la main à l'utilisateur, rappelons-le. Il existe même SWI1 et SWI2 qui complètent la liste des interruptions logicielles de type SWI.

Pour aller au menu, sans perdre le listing, que ce soit en assembleur ou en basic, il existe une adresse = $E82D :

Machine (après assemblage (A)) :			Assembleur :		Basic :
7D00			ORG	&32000	10 POKE 32000, &H7E
7D00	7E	E82D	JMP	$E82D	20 POKE 32001, &HE8
7D03	3F		SWI		30 POKE 32002, &H2D
		0000	END		40 POKE 32003, &H39
					50 EXEC 32000

En assembleur, faites G&32000 et ENTREE dans le moniteur (après assemblage).

La routine permettant d'accéder au menu s'appelle MENUH, et la routine qui initialise l'appareil est le RESET. Nous allons maintenant découvrir cette dernière.

e. Notion d'indirection :

L'adressage dit indirect ne sera pas utilisé dans l'ouvrage. Sachez simplement qu'il existe et permet d'aller à « l'adresse de l'adresse ». Il est représenté par des crochets : [ ].

Machine (après assemblage (A)) :				Assembleur :		Basic :	
7D00							
7D00	6E	9F	FFFE	ORG	&32000	10 POKE	32000, &H6E
7D04	3F			JMP	[$FFFE]	20 POKE	32001, &H9F
			0000	SWI		30 POKE	32002, &HFF
				END		40 POKE	32003, &HFE
						50 POKE	32004, &H39
						60 EXEC	32000

Avec la cartouche assembleur, pour faire "[", faites CNT-A ; pour faire "]", faites CNT-Z.

Sur TO7(70), les adresses $FFFE-$FFFF contiennent respectivement les valeurs $F9 et $8D, qui représentent l'adresse du RESET ou de l'initialisation de votre appareil. L'ordinateur ira donc à l'adresse $F98D, c'est-à-dire que le menu va réapparaître (le RESET passe par MENUH). A noter que cette adresse est codée sur 2 octets : $FFFE-$FFFF, mais qu'en programmation en assembleur on ne note que l'adresse de poids fort : JSR [$FFFE]. L'initialisation diffère de la routine du menu par la remise à une valeur fixée par le constructeur d'un certain nombre de registres (pages 0) en R.A.M. (voir tableau n° 16).

Sur TO9, $FFFE-$FFFF contiennent l'adresse $EC19 du RESET.

Ce qui vient d'être expliqué est valable sur MO5, mais l'adresse du RESET contenue en $FFFE-$FFFF est $F003. SWI doit être remplacée par STOP.

En basic, &H39 remplacera le "3F" de SWI ou le "BD B000" de STOP.

Pour une compatibilité entre tous les appareils auxquels ce livre est destiné, les programmes qui vont suivre seront en général implantés en 32000 décimal, soit $7D00, ou éventuellement avant cette adresse.

Ces programmes, comme nous l'avons déjà dit, ne seront pas toujours translatables à d'autres adresses, surtout en basic, sauf pour le programme n° XXXIV dont ce sera le rôle.

Nous allons maintenant découvrir les routines du moniteur-système, une à une, et nous en servir pour travailler en langage-machine.

1. Routine PUTCH

Ce chapitre prend comme exemple le mode 40 colonnes. Pour les autres modes d'affichage, sur TO9, il faudra vous reporter au chapitre P9 en fin d'ouvrage.

A. DESCRIPTION

Cette routine est sûrement la première à étudier. Elle a de multiples fonctions que voici :

a. Gestion des codes interprétables (bip, curseur allumé/éteint, etc.). Voir tableau n° 5, 1re partie.

b. Positionnement de la fenêtre de travail et du curseur.

c. Gestion des caractères alphanumériques ou codes affichables (G0 et G2, dont la séquence-accent).

d. Gestion des caractères-utilisateur (DEFGR$ en basic).

e. Gestion de divers attributs : couleur, taille des caractères, masquage, inversion-vidéo, écriture sans couleur, incrustation, scroll ou mode page (tableau n° 7). Le TO9 possède des possibilités supplémentaires.

- G0 est le générateur de caractères standards (A, B, C, D, etc.). Voir tableau N° 5, 2e partie.
- G2 est un générateur de 5 à 22 caractères spéciaux comme les accents, le tréma ou la cédille. Voir tableau n° 15.

**Point d'entrée :*

— Sur TO7(70) et TO9 : $E803
— Sur MO5 : Code N° 2

**Paramètre d'entrée :*

— *Registre B du 6809*

B. APPLICATIONS

PROGRAMME N° I

1. Description :

Nous allons gérer un code interprétable : le BIP. Nous ferons un, puis cinquante bip réalisés par une boucle en langage-machine, avec étude d'une instruction de comparaison et d'un branchement conditionnel (en basic : IF... THEN).

Pour générer un bip, il faut se servir de l'accumulateur (ou registre) B du 6809, qui est un paramètre d'entrée obligatoire. Lorsque "B" a pris la valeur ASCII du bip (7), il faut alors faire un « saut » à la routine PUTCH. Le programme doit enfin rendre la main au programmeur (SWI pour TO7(70) et TO9, STOP pour MO5, ou RTS en basic).

2. Réalisation :

a. Assembleur : TO7(70) et TO9 :

```
          E803     PUTCH  EQU    $E803

7D00                      ORG    &32000
7D00 C6   07              LDB    #7
7D02 BD   E803            JSR    PUTCH
7D05 3F                   SWI
          0000            END

00000 Total Errors
```

Puis faire G & 32000 dans la page-moniteur, après assemblage (A).

b. Assembleur : MO5 :

```
          0002     PUTCH  EQU    2

7D00                      ORG    &32000
7D00 C6   07              LDB    #7
7D02 3F                   SWI
7D03 02                   FCB    PUTCH
7D04 BD   B000            STOP
          0000            END

00000 Total Errors
```

Puis faire G & 32000 dans la page-moniteur, après assemblage (A).

c. Basic :

— TO7(70) et TO9 :

```
10 FOR I=32000 TO 32000+5'OU &H7D00 TO
&H7D05
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT I
50 DATA C6,07,BD,E8,03,39
60 'puis faire RUN et EXEC 32000
```

— MO5 :

```
10 FOR I=32000 TO 32000+4'OU &H7D00 TO
&H7D04
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT I
50 DATA C6,07,3F,02,39
60 'puis faire RUN et EXEC 32000
```

3. Explications :

La directive EQU (EQUATE) définit l'étiquette : PUTCH. Elle est facultative. On pouvait enlever cette ligne et mettre :

— Sur TO7(70) ou TO9 : JSR $E803
— Sur MO5 : SWI
FCB 02

EQU n'est qu'une directive d'assemblage et n'a pas de code en langage-machine. Ne mettez donc pas les adresses des « EQUATES » dans votre programme-basic !

Rappel :

Lors d'un « saut » à une routine, celle-ci sauvegarde en général tous les registres du processeur. Le ''PC'', qui pointe déjà sur l'instruction suivante, est quant à lui sauvegardé lors d'un JSR.

Lors du retour de la routine, les registres sont remis dans l'état où ils se trouvaient lors de l'appel, sauf pour les registres dits de retour lorsqu'ils existent. ''PC'' est appelé aussi pointeur de programme ou compteur ordinal.

Comme vous le constatez, ce programme ne permet que la réalisation d'un modeste bip, dédoublement du bip habituel du clavier (sur TO9, il n'y a pas de bip au clavier). Ceci est trop bref, nous allons donc en faire cinquante et juger de leur rapidité d'exécution.

Auparavant, étudions une « variable spécifique ».

Ce que nous appelons une variable spécifique, correspond à une position-mémoire de la R.A.M., sinon elle ne serait pas variable. Spécifique elle l'est, car la R.O.M. se la réserve pour ses propres applications, c'est-à-dire pour faire « tourner » ses routines. Ces variables sont parfois appelées vecteurs en R.A.M., ou encore registres.

Les registres de la page 0 du moniteur et ceux de la page 0 du basic représentent des variables spécifiques, pour les usages respectifs du système ou du programme d'application (voir tableaux n° 2 et 6).

L'avantage de ces variables en R.A.M. est, pour l'utilisateur, de pouvoir détourner une routine, ou quelque peu la modifier.

Une condition essentielle pour réaliser un bip sonore est d'avoir la variable en $6073 (registre BUZZ) sur TO7(70) ou TO9 égale à zéro. Cette variable est initialisée à zéro lors de la mise en route de l'appareil. La routine du bip y fait appel. En faisant POKE &H6073,1, on supprime le bip du clavier (ou celui de l'instruction BEEP en basic).

Sur MO5, cette variable se situe en $2019 (registre STATUS). Ce registre STATUS est initialisé à 4 lors de la mise en route de l'appareil. Si le bit n°3 de STATUS est à 0, la routine génère le bip sonore, sinon elle ne le fait pas. En faisant POKE &H2019,8 par exemple, on supprime le bip du clavier (ou celui de l'instruction BEEP en basic).

En assembleur, faites le petit travail suivant :

Mettez-vous dans la page du moniteur-binaire, derrière un dièse. Tapez N, ENTREE, puis :

- Sur TO7(70) : 6073/, vous voyez apparaître 0. Tapez alors 1 et ENTREE. Vérifiez en tapant de nouveau derrière un dièse : 6073/. D'abord vous n'avez plus de son, c'est normal, puis vous voyez apparaître la nouvelle valeur de l'octet : 1. Tapez de nouveau 0 et ENTREE : tout est rentré dans l'ordre.

- Sur MO5 : 2019/, vous voyez apparaître 4. Vous changerez cette valeur pour la valeur 8 et restituerez ensuite la valeur 4, comme indiqué ci-dessus pour les TO7(70).

Ces registres BUZZ et STATUS peuvent être considérés comme des paramètres d'entrée dans la routine PUTCH, pour exécuter un bip.

A noter qu'il existe sur MO5 une routine spécifique du bip (code n°8), évitant de passer par PUTCH. L'instruction SWI suivie de FCB 8 a

donc le même effet, de même CALL 8, que le programme que nous venons de réaliser.

4. Variante :

Nous allons faire cinquante bip de suite.

a. *Assembleur : TO7(70) et TO9*

```
           E803     PUTCH  EQU     $E803

7D00                       ORG     &32000
7D00 4F                    CLRA
7D01 C6    07              LDB     #7
7D03 BD    E803     DEBUT  JSR     PUTCH
7D06 4C                    INCA
7D07 81    32              CMPA    #&50
7D09 26    F8              BNE     DEBUT
7D0B 3F                    SWI
           0000            END

00000 Total Errors
```

b. *Assembleur : MO5 :*

```
           0002     PUTCH  EQU     2

7D00                       ORG     &32000
7D00 4F                    CLRA
7D01 C6    07              LDB     #7
7D03 3F             DEBUT  SWI
7D04 02                    FCB     PUTCH
7D05 4C                    INCA
7D06 81    32              CMPA    #&50
7D08 26    F9              BNE     DEBUT
7D0A BD    B000            STOP
           0000            END

00000 Total Errors
```

c. *Basic :*

— TO7(70) et TO9 :

```
10 FOR I=32000 TO 32000+11
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT
50 DATA 4F,C6,07,BD,E8,03,4C,81,32,26,F8
,39
```

— *MO5 :*

```
10 FOR I=32000 TO 32000+10
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT
50 DATA 4F,C6,07,3F,02,4C,81,32,26,F9,39
```

REMARQUES :

• CLRA signifie CLEAR A, c'est-à-dire mettre "A", qui va nous servir de compteur, à zéro.
• INCA signifie incrémente "A" d'une unité, à chaque boucle.
• CMPA signifie Compare "A" à la valeur placée dans le champ-opérande (#&50). Cette instruction est suivie d'un branchement.
• BNE DEBUT signifie branchement si non égal (si "A" n'est pas encore égal à &50, alors retour à l'étiquette : DEBUT).

Pour CMPA, c'est l'indicateur Z du registre CC, se positionnant à 1 lorsque la condition "A" = 50 est vérifiée, qui commande le branchement qui suit. En effet si "A" = 50, la différence entre la valeur du champ-opérande (50) et "A" est zéro, et "Z" égale 1. BNE ne fonctionne donc que pour "Z" =0.

Un petit mot sur les adressages :

• CLRA et INCA sont dits à adressage inhérent ou implicite, car se suffisent à eux-seuls, sans opérande.

• "#" signifie adressage immédiat, c'est-à-dire qu'il précède une VALEUR (ceci a déjà été vu).

• $E803 sur TO7(70) et TO9 (adresse représentée par PUTCH) : adressage étendu. Inutile de le forcer par ">", d'autant qu'il n'est pas possible de travailler en page directe à ce niveau (R.O.M. non programmable).

SWI, sur MO5, est une interruption, et FCB une directive d'assemblage. Il n'est donc pas question d'adressage ici.

• BNE : adressage relatif, car il se réfère au "PC". Il s'agit d'un branchement conditionnel court (sur + 127 octets maximum vers l'avant, ou – 128 vers l'arrière du programme). Lorsque "Z" = 1, BNE ne branche plus le "PC" à DEBUT, mais laisse le programme continuer vers SWI, STOP ou RTS selon l'appareil et le langage utilisés.

REMARQUES COMPLÉMENTAIRES :

La boucle FOR-NEXT n'existe plus en langage-machine. Elle est remplacée par un compteur qui s'incrémente à chaque passage, et par un test du type : IF A < 50 THEN GOTO DEBUT.

L'opérande de l'instruction CMPA ne peut excéder 255, car le registre A est un registre de 8 bits. Pour répéter une même opération 500 ou 5000 fois de suite, il faut utiliser l'accumulateur D (= A + B) et mettre l'étiquette DEBUT devant LDB, car "B" prendra deux valeurs dans le programme : 7, puis sa valeur dans le compteur "D". Quant à "D", il faut alors le stocker dans 1 case-mémoire particulière, aller le chercher et le redéposer à chaque boucle. Dans ce cas on pourrait aussi utiliser un registre d'index (X ou Y) et l'instruction CMPX ou CMPY, mais cela nécessiterait une incrémentation plus particulière de "X" ou de "Y", car "INCX" ou "INCY" n'existent pas.

Le retour à DEBUT se fait de la manière suivante : Le "PC" est déjà sur SWI, STOP ou RTS ; il faut donc reculer de 8 octets sur TO7(70) et TO9, ou de 7 octets sur MO5 (un octet de moins avec SWI suivie de FCB 2, que sur TO7(70) et TO9). En hexadécimal signé : −8 = F8, ou −7 = F9 (compléments à 2). A ce sujet, revoyez le tableau n°4.

Ce programme-ci est translatable en cas de changement d'adresse d'implantation. Si l'instruction BNE DEBUT était remplacée par BNE $7D03, cela ne serait plus vrai.

En basic, souvenez-vous : 12 Octets = 12 données en DATA (ou 11 sur MO5).

PROGRAMME N° II

1. Description :

Écrire la lettre A à la position courante du curseur. Avec la cartouche-assembleur, après exécution du programme, on verra :
"A 8 BRK @ 7D05" (ou 7D04 sur MO5) dans le moniteur. La lettre A sera en effet écrite devant 8 signifiant SWI ou STOP selon l'appareil utilisé, suivi de BRK (break) et de l'adresse du break (7D05 ou 7D04 selon l'appareil employé).

Ceux qui travaillent en basic connaissent désormais la manière de procéder.

2. Réalisation :

a. TO7(70) et TO9 :

```
           E803     PUTCH  EQU     $E803

7D00                       ORG     &32000
7D00 C6    41              LDB     #$41
7D02 BD    E803            JSR     PUTCH
7D05 3F                    SWI
           0000            END

00000 Total Errors
```

b. MO5 :

```
           0002     PUTCH  EQU     2

7D00                       ORG     &32000
7D00 C6    41              LDB     #$41
7D02 3F                    SWI
7D03 02                    FCB     PUTCH
7D04 BD    B000            STOP
           0000            END

00000 Total Errors
```

3. Explications :

Remarquez bien que l'on se sert encore de l'accumulateur B qui est un paramètre d'entrée dans la routine PUTCH. Le code ASCII étant

donné par le tableau n° 5, inutile de le rappeler. Il existe un générateur de caractères de l'alphabet standard (G0), contenant le standard ASCII affichable. Chaque caractère est composé de 8 octets (cf graphisme de la lettre A, décrit précédemment) et le premier octet du caractère correspond à la ligne inférieure de la matrice de 8*8 bits. L'adresse du début de ce générateur G0 est contenue en R.A.M. dans le registre PTGENE (60CFH-60D0H) sur TO7-70 et TO9, et GENPTR (2073H-2074H) sur MO5.

Il est possible de redéfinir l'alphabet standard, ou de créer son propre générateur de caractères, sauf sur TO7. Il faut alors charger l'adresse de ce nouveau générateur dans le registre PTGENE ou GENPTR. Nous en reparlerons plus loin.

PROGRAMME N° III

1. Description :

Nettoyage de la fenêtre, extinction du curseur. Étude de la séquence d'échappement "ESC" pour attributs : ici, inversion-vidéo et scroll doux seront les attributs considérés.

2. Réalisation :

a. TO7(70) et TO9 :

```
          E803     PUTCH  EQU    $E803

7D00                      ORG    &32000
7D00 C6   0C              LDB    #$0C     RAZ
7D02 BD   E803            JSR    PUTCH
7D05 C6   14              LDB    #$14     EFF.CURSEUR
7D07 BD   E803            JSR    PUTCH
7D0A C6   1B              LDB    #$1B     ESC
7D0C BD   E803            JSR    PUTCH
7D0F C6   5C              LDB    #$5C     INVERSION VIDEO
7D11 BD   E803            JSR    PUTCH
7D14 C6   1B              LDB    #$1B     ESC
7D16 BD   E803            JSR    PUTCH
7D19 C6   6E              LDB    #$6E     SCROLL DOUX
7D1B BD   E803            JSR    PUTCH
7D1E 3F                   SWI
          0000            END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH  EQU    2

7D00                      ORG    &32000
7D00 C6   0C              LDB    #$0C     RAZ
7D02 3F                   SWI
7D03 02                   FCB    PUTCH
7D04 C6   14              LDB    #$14     EFF. CURSEUR
7D06 3F                   SWI
7D07 02                   FCB    PUTCH
7D08 C6   1B              LDB    #$1B     ESC
7D0A 3F                   SWI
7D0B 02                   FCB    PUTCH
7D0C C6   7B              LDB    #$7B     INVERSION VIDEO
7D0E 3F                   SWI
7D0F 02                   FCB    PUTCH
```

```
7D10 C6   1B          LDB    #$1B    ESC
7D12 3F               SWI
7D13 02               FCB    PUTCH
7D14 C6   79          LDB    #$79    SCROLL DOUX
7D16 3F               SWI
7D17 02               FCB    PUTCH
7D18 BD   B000        STOP     .
          0000        END

00000 Total Errors
```

3. Explications :

Vous remarquerez le RAZ (effacement du G &32000 lors de l'exécution dans le moniteur-binaire, ou effacement du listing en basic).

L'effacement du curseur et l'inversion-vidéo ne sont visibles qu'en basic. Tout au plus vous remarquerez qu'en assembleur, dans le moniteur, le curseur est devenu jaune au lieu de bleu ; mais commandé par la cartouche, il est resté présent.

En basic, pour vérifier le scroll doux, tapez plusieurs fois LIST et ENTREE par exemple. Pour la même vérification éventuellement, en assembleur, insérez quelques lignes blanches dans le programme de la page-éditeur par la touche ENTREE, puis faites un nouvel assemblage.

La séquence dite « d'échappement » (ESC*) est une séquence obligatoire à envoyer à la routine, avant les attributs de type courant cités dans le tableau n°7, mais aussi avant le choix de la couleur, nous allons le voir.

Le programme que nous venons de réaliser demandait un certain nombre de codes que l'on aurait pu ranger dans une table, comme les données en DATA du basic. Dans le programme qui suit, nous allons étudier l'utilisation d'une table pour les attributs de couleur.

* ESC signifie : ESCAPE

PROGRAMME N° IV

1. Description :

Gestion des attributs de couleur et utilisation d'une TABLE. Étude de l'adressage indexé autoincrémenté, du branchement BEQ et d'un branchement inconditionnel.

Le codage des couleurs est réalisé sur le quartet de poids fort de l'attribut :

$4X : couleur de forme
$5X : couleur de fond
$6X : couleur du tour

et sur son quartet de poids faible, comme en basic pour les couleurs foncées (X devant être compris entre 0 et 7). Pour les couleurs pastel, voir le tableau n°7 (sauf TO7).

2. Réalisation :

a. TO7(70) et TO9 :

```
          E803     PUTCH  EQU    $E803

7D00                      ORG    &32000
7D00 8E   7D0F            LDX    #TABLE
7D03 E6   80      DEBUT   LDB    ,X+
7D05 C1   04              CMPB   #4
7D07 27   05              BEQ    FIN
7D09 BD   E803            JSR    PUTCH
7D0C 20   F5              BRA    DEBUT
7D0E 3F           FIN     SWI
7D0F 1B           TABLE   FCB    $1B      ESC
7D10 42                   FCB    $42      FORME VERTE
7D11 1B                   FCB    $1B      ESC
7D12 50                   FCB    $50      FOND NOIR
7D13 1B                   FCB    $1B      ESC
7D14 61                   FCB    $61      TOUR ROUGE
7D15 41                   FCB    $41      LETTRE A
7D16 04                   FCB    $4       E.O.T.
          0000            END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH  EQU    2

7D00                      ORG    &32000
7D00 8E   7D10            LDX    #TABLE
7D03 E6   80      DEBUT   LDB    ,X+
7D05 C1   04              CMPB   #4
7D07 27   04              BEQ    FIN
7D09 3F                   SWI
7D0A 02                   FCB    PUTCH
7D0B 20   F6              BRA    DEBUT
7D0D BD   B000    FIN     STOP
7D10 1B           TABLE   FCB    $1B      ESC
7D11 42                   FCB    $42      FORME VERTE
7D12 1B                   FCB    $1B      ESC
7D13 50                   FCB    $50      FOND NOIR
7D14 1B                   FCB    $1B      ESC
7D15 61                   FCB    $61      TOUR ROUGE
7D16 41                   FCB    $41      LETTRE A
7D17 04                   FCB    $4       E.O.T.
          0000            END

00000 Total Errors
```

c. basic (pour MO5) :

```
10 FOR I=&H7D00 TO &H7D15
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT
50 DATA 8E,7D,0E,E6,80,C1,04,27,04,3F,02
,20,F6,39,1B,42,1B,50,1B,61,41,04
```

3. Explications :

Pourquoi avoir donné la version-basic pour MO5 ? Si la transposition du langage-machine au basic ne pose pas de problème de la manière que nous connaissons, pour les TO7(70) et TO9 où SWI = $3F est remplacée par RTS = &H39, il n'en est pas de même pour le MO5. En effet il faut, sur MO5, remplacer les 3 octets contenant : $BD, $B0, 00 = STOP par un seul octet : &H39 = RTS. Cela va déplacer l'adresse de la table de $7D10 en $7D0E. Si vous ne déplacez pas cette adresse lors de son chargement dans : LDX #TABLE = ''8E, 7D, 10'', pour la mettre en ''7D, 0E'' (souligné dans la version-basic),

vous démarrerez le programme par la mise en couleur du fond, et la couleur de forme ne sera pas changée. Une autre méthode eût consisté à remplacer les deux octets manquants par deux instructions NOP = $12, c'est-à-dire : « NO OPERATION ». Cette instruction n'a aucun effet sur le programme, mais permet d'avancer l'adresse du compteur ordinal d'un octet. En basic, en ligne 10, nous avons : FOR I = &H7D00 TO &H7D15 ; avec NOP, répétée deux fois, nous aurions pu garder, comme en machine,... TO &H7D17.

Ce programme sera, cette fois encore, mieux visualisé dans sa version-basic.

Nous étudions ici beaucoup de choses. Tout d'abord, nous chargeons l'adresse de la table dans le registre X (''#'' = valeur, équivalente au numéro d'adresse défini par l'étiquette TABLE). Les registres internes du processeur n'ont pas d'adresse en mémoire externe ; ils ne peuvent que contenir une valeur (équivalente à un numéro d'adresse éventuellement, pour les registres de 16 bits).

Nous utilisons ensuite une boucle, dans laquelle l'instruction du premier branchement est BEQ, c'est-à-dire BRANCH ON EQUAL (branchement en cas d'égalité) : si ''B'' = 4, alors l'indicateur Z = 1 et le branchement BEQ se fait vers l'étiquette FIN.

BRA est un branchement sans condition, obligatoire, ne pouvant être refusé : BRANCH ALWAYS. On l'appelle branchement inconditionnel (= GOTO en basic).

Dans LDB ,X+ , l'adressage est appelé : indexé autoincrémenté d'un octet. ''B'' sera ici chargé avec la première valeur de la table, à l'adresse de ''X'', puis de la deuxième à l'adresse de ''X'' + 1, etc..

Lorsque l'accumulateur B rencontrera $4 = E.O.T., c'est-à-dire END OF TRANSMISSION, le programme sera terminé. E.O.T. est donc un test de fin de table. Attention, l'autoincrémentation d'un registre d'index se fait après le chargement, alors que l'autodécrémentation se fait avant le chargement.

Remarquez : BRA DEBUT = ''20,F5'' sur TO7(70) et TO9, ou ''20,F6'' sur MO5 (nombre négatif ou complémenté à deux, branchant vers l'arrière du programme) et BEQ FIN = ''27,05'' ou ''27,04'' (nombre positif, branchant vers l'avant du programme = 5 ou 4 octets, selon l'appareil utilisé, car le pointeur PC est déjà sur l'instruction suivante).

La directive FCB, déjà rencontrée, signifie FORM CONSTANT BYTE. FCB permet le rangement d'une constante d'un octet, dans la case-mémoire sur laquelle pointe le ''PC'' au moment où elle est rencontrée. FCB peut stocker des nombres, des étiquettes ou des expres-

sions. Les étiquettes devront cependant être définies dans le programme-assembleur. On peut placer plusieurs FCB dans un programme, ou implanter plusieurs constantes, séparées par une virgule, par une même directive FCB, ce qui revient au même.

Il existe encore FDB signifiant : FORM DOUBLE BYTE CONSTANT (implantation d'une constante de deux octets).

Il existe enfin FCC signifiant FORM CONSTANT CHARACTER STRING (implantation d'une constante-chaîne de caractères). Son utilisation correspond à celle d'une variable alphanumérique en basic.

Exemple :

En basic :
IN$ = ''INFORMATIQUE''

En assembleur :
IN FCC /INFORMATIQUE/ où IN est une étiquette que l'on peut appeler dans le programme. A noter que l'on définit IN et non IN$ qui serait, en assembleur, refusée (à cause du ''$'') comme « expression error » dans la zone-opérande, et « bad Label » (mauvaise étiquette) devant FCC. Le mot INFORMATIQUE doit être entre deux délimiteurs identiques : ici deux SLASH (/ /). Certaines étiquettes peuvent être refusées lorsqu'elles utilisent des symboles particuliers comme : ''*'' ou ''.'', qui représentent la valeur courante du compteur-programme, ou lorsqu'elles commencent par un chiffre. ENDMEM, signifiant fin de mémoire-utilisateur, sera refusée comme étiquette, ainsi que A,B,D (qui représentent les accumulateurs), etc.. Par contre A1,B1,D1 seront acceptés, etc..

A signaler que ENDMEM peut être utilisée dans le champ-opérande par exemple sous la forme ORG ENDMEM-X (fin de mémoire-X octets) pour l'implantation d'un programme. Il s'agit alors d'une expression. De même ''*'' ou ''.'' sont utilisables dans cette même zone.

En affichant la lettre A, on vérifie que la couleur de forme est bien verte.

PROGRAMME N° V

1. Description :

Gestion des attributs de type plein écran.

Comme vous avez pu le constater dans le programme précédent, le fond noir et la forme verte correspondent à un COLOR 2,0 en basic. Avec les mêmes codes, mais en plein écran, nous allons réaliser le SCREEN 2,0 du basic.

Les attributs n'ayant pas de signification en plein écran sont ignorés (par exemple la couleur du tour). Ne faites donc pas précéder une couleur de tour par une séquence de type plein écran ! Le tour peut d'ailleurs être coloré en &HE7C3 sur TO7(70), mais pas sur TO9, et en &HA7C0 sur MO5.

- *TO7(70) et TO9 :*

Les attributs de type plein écran nécessitent la séquence suivante : ESC, puis " # ", puis SP (espace), suivis du code de l'attribut. Nous aurons donc ceci : $1B, $23, $20, code de l'attribut.

- *MO5 :*

Les attributs de type plein écran nécessitent la séquence suivante : ESC, puis SP (espace), suivis du code de l'attribut. Nous aurons donc ceci : $1B, $20, code de l'attribut.

2. Réalisation :

a. TO7(70) et TO9 :

```
          E803     PUTCH   EQU     $E803

7D00                       ORG     &32000
7D00 CE   7D0F     ECRAN   LDU     #TABLE
7D03 E6   C0       DEBUT   LDB     ,U+
7D05 C1   04               CMPB    #4
7D07 27   05               BEQ     FIN
7D09 BD   E803             JSR     PUTCH
7D0C 20   F5               BRA     DEBUT
7D0E 3F                    SWI
7D0F 1B                    FCB     $1B       ESC
7D10 23                    FCB     $23       #
7D11 20                    FCB     $20       BARRE ESPACE
7D12 42                    FCB     $42       FORME VERTE
7D13 1B                    FCB     $1B       ESC
7D14 23                    FCB     $23       #
7D15 20                    FCB     $20       BARRE ESPACE
```

```
7D16 50                    FCB     $50       FOND NOIR
7D17 1B                    FCB     $1B       ESC
7D18 61                    FCB     $61       TOUR ROUGE
7D19 04                    FCB     $4        E.O.T.
          0000             END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH   EQU     2

7D00                       ORG     &32000
7D00 CE   7D10     ECRAN   LDU     #TABLE
7D03 E6   C0       DEBUT   LDB     ,U+
7D05 C1   04               CMPB    #4
7D07 27   04               BEQ     FIN
7D09 3F   02               CALL    PUTCH
7D0B 20   F6               BRA     DEBUT
7D0D BD   B000     FIN     STOP
7D10 1B            TABLE   FCB     $1B       ESC
7D11 20                    FCB     $20       BARRE ESPACE
7D12 42                    FCB     $42       FORME VERTE
7D13 1B                    FCB     $1B       ESC
7D14 20                    FCB     $20       BARRE ESPACE
7D15 50                    FCB     $50       FOND NOIR
7D16 1B                    FCB     $1B       ESC
7D17 61                    FCB     $61       TOUR ROUGE
7D18 04                    FCB     $4        E.O.T.
          0000             END

00000 Total Errors
```

c. Basic (MO5) :

Rappel :

Remplacez "BD B000" par "12, 12, 39", c'est-à-dire par NOP, NOP, RTS, où NOP signifie NO OPERATION et ne sert qu'à combler des vides pour ne pas devoir changer vos adresses. On aurait pu aussi écrire : "39, X, X", où chaque X aurait pu prendre n'importe quelle valeur puisque le programme ne dépasse pas $39 dans cet exemple.

3. Explications :

Le registre à incrémenter devient le pointeur U, qui sert ici de registre d'index. On pouvait aussi bien sûr se servir de "X", "Y" ou "S",

avec méfiance cependant pour ''S''. En effet, et ceci a déjà été évoqué, la pile S est plutôt réservée au système. En changeant l'adresse du pointeur S et en lui attribuant celle de TABLE, le programme aurait été perdu lors du « saut » vers PUTCH. D'une manière générale, il vaut mieux ne jamais changer l'adresse de la pile S ; ceci ne veut pas dire que l'on ne puisse pas se servir de celle-ci, à son adresse habituelle.

Le résultat est donc le SCREEN 2, 0, 1 du basic, à la suite de ce programme.

En basic, ne soyez pas ennuyé si RTS = &H39 se situe au milieu des données en DATA. Cette instruction est bien la dernière à exécuter, la table pouvant se situer après elle.

En assembleur, placez toujours END après la table et non avant celle-ci. Ce END signifie au programme d'application de ne pas assembler plus loin.

Nous avons vu comment coder les couleurs dans un octet de mémoire-écran. Maintenant nous savons aussi gérer les couleurs par la routine PUTCH. Il nous reste encore à connaître un registre spécifique de la page 0 du moniteur (tableau n° 6) qui code les couleurs en mode-caractère, mais pas en mode graphique.

a. Sur TO7(70) et TO9 :

Le registre COLOUR en $603B permet de faire le COLOR X, Y du basic. Faites POKE &H603B, 225 : vous obtenez un COLOR 4, 1.

b. Sur MO5 :

Le registre COLOUR est en $202B. Faites POKE &H202B, 65 pour avoir un COLOR 4, 1.

Le codage de la couleur, dans COLOUR, est le même que celui de la mise en couleur des octets de la R.A.M. d'écran.

PROGRAMME N° VI

1. Description :

Position de la fenêtre (CONSOLE en basic), et du curseur (LOCATE en basic).

Nous allons faire un CONSOLE 3,21 et un LOCATE 15,8. La colonne n° 15 du basic deviendra colonne n° 16 en langage-machine, car dans ce langage on compte les lignes de 0 à 24 et les colonnes de 1 à 40. De plus, en langage-machine toujours, pour le curseur, il faut positionner la ligne avant la colonne.

Le passage du curseur en 16,8 sera si bref qu'on ne le verra pas. Il faudra donc laisser une trace en écrivant quelque chose, par exemple la lettre A.

2. Réalisation :

a. TO7(70) et TO9 :

```
            E803     PUTCH  EQU    $E803

7D00                        ORG    &32000
7D00 8E     7D0F            LDX    #TABLE
7D03 E6     80       DEBUT  LDB    ,X+
7D05 C1     04              CMPB   #4
7D07 27     05              BEQ    FIN
7D09 BD     E803            JSR    PUTCH
7D0C 20     F5              BRA    DEBUT
7D0E 3F                 FIN SWI
7D0F 1F                 TABLE FCB  $1F      US
7D10 20                       FCB  $20      .
7D11 23                       FCB  $23      .HAUT FENETRE=03
7D12 1F                       FCB  $1F      US
7D13 12                       FCB  $12      .
7D14 11                       FCB  $11      .BAS FENETRE=21
7D15 1F                       FCB  $1F      US
7D16 48                       FCB  $48      .
7D17 50                       FCB  $50      .CURSEUR:L=8;C=16
7D18 41                       FCB  $41      LETTRE A
7D19 04                       FCB  $4       E.O.T.
             0000             END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH  EQU    2

7D00                      ORG    &32000
7D00 8E   7D10            LDX    #TABLE
7D03 E6   80      DEBUT   LDB    ,X+
7D05 C1   04              CMPB   #4
7D07 27   04              BEQ    FIN
7D09 3F                   SWI
7D0A 02                   FCB    PUTCH
7D0B 20   F6              BRA    DEBUT
7D0D BD   B000    FIN     STOP
7D10 1F           TABLE   FCB    $1F      US
7D11 20                   FCB    $20      .
7D12 23                   FCB    $23      .HAUT FENETRE=03
7D13 1F                   FCB    $1F      US
7D14 12                   FCB    $12      .
7D15 11                   FCB    $11      .BAS FENETRE =21
7D16 1F                   FCB    $1F      US
7D17 48                   FCB    $48      .
7D18 50                   FCB    $50      .CURSEUR:L=8;C=16
7D19 41                   FCB    $41      LETTRE A
7D1A 04                   FCB    $4       E.O.T.
          0000            END

00000 Total Errors
```

3. Explications :

En basic, sur MO5, n'oubliez pas de remplacer "BD B000" par "12, 12, 39".

Pour la fenêtre et le curseur, trois appels consécutifs à la routine sont nécessaires. Le 1er code à envoyer est US ($1F), ESC ($1B) étant réservé aux attributs. Les autres sont les suivants :

bas de la fenêtre :

Après la séquence US, il faut coder le numéro de ligne en deux fois. Le quartet de poids fort des 2 codes sera toujours mis à $1x. Le quartet de poids faible des 2 codes permet le codage de la ligne :

Le 1er code à envoyer est celui des dizaines : x va de 0 à 2.

Le 2e code à envoyer est celui des unités : x va de 0 à 9.

On additionne donc chaque valeur de x à $10.

Exemple :

1er code : $1 | 1 |
2e code : $1 | 2 | } ligne 12

haut de la fenêtre :

Après la séquence US, il faut coder le numéro de ligne en 2 fois, mais ici le quartet de poids fort des 2 codes doit être égal à $2x. Le codage des quartets de poids faible reste le même :

Le 1er code à envoyer est celui des dizaines : x va de 0 à 2.

Le 2e code à envoyer est celui des unités : x va de 0 à 9.

On additionne donc chaque valeur de x à $20.

curseur :

Deux possibilités sur TO7(70) et TO9, une seule sur MO5 :

a. TO7(70) et TO9 :

Pour positionner le curseur AU DÉBUT D'UNE LIGNE : le quartet de poids fort devra être égal à $3x pour les 2 codes à envoyer après la séquence US. Le codage des quartets de poids faible reste identique à celui de la fenêtre.

Le 1er code à envoyer est celui des dizaines : x va de 0 à 2.

Le 2e code à envoyer est celui des unités : x va de 0 à 9.

On additionne donc chaque valeur de x à $30.

b. TO7(70), TO9 et MO5 :

Pour positionner le curseur SUR UNE LIGNE ET SUR UNE COLONNE : après la séquence US, 2 codes sont à envoyer. Chaque code s'additionne à $40, cette fois :

Le 1er code à envoyer est celui de la ligne : de 0 à $18.

Le 2e code à envoyer est celui de la colonne : de 1 à $28.

Exemple :

Ligne n° 15 ($F) : 1er code = $40 + $F = $4F.

Colonne n° 20 ($14) : 2e code = $40 + $14 = $54 (ceci correspond à la colonne n° 19 en basic).

Les variables codant le curseur se trouvent en :

a. TO7(70) et TO9 :

601BH et 6020H

b. MO5 :

201BH et 201CH

On peut définir CONSOLE en :

a. TO7(70) et TO9 :

601DH et 601FH

b. MO5 :

201EH et 2020H

Vous retrouverez ces registres spécifiques dans le tableau n° 6.

Une petite note pour les TO7(70) et TO9 :

Une « saut » à la routine $E800 repositionne la fenêtre plein écran. Il fait disparaître le curseur (sauf en BASIC 128), sans faire CLS. Essayez : EXEC &HE800. Cette routine s'appelle INITSCH (INITIALIZATION SCREEN) et correspond à l'initialisation de l'écran en mode courant.

PROGRAMME N° VII

1. Description :

Visualisation des caractères contenus dans G0 (tableau n° 5, 2e partie) et des caractères du G2 (accents par exemple) placés à la suite du G0. Pour le G2, voir tableau n° 15.

Le générateur G0 se situe pour les TO7(70) et pour le TO9 en $E845. Pour le MO5, il est en $FC9E. Ces adresses sont données par 2 octets en ($60CF-$60D0) = registre PTGENE pour les TO7-70 et TO9 (ce registre n'existe pas sur TO7), et en ($2073-$2074) = registre GENPTR pour le MO5. Il faut multiplier l'octet de poids fort de PTGENE ou de GENPTR par 256 et y ajouter l'octet de poids faible, pour obtenir l'adresse en valeur décimale de ce générateur. Le générateur G0 est une table contenant tous les caractères standard affichables à l'écran, c'est-à-dire de code ASCII compris entre 32 et 127 en décimal ou de $20 à $7F en hexadécimal.

Ce générateur permet l'écriture des caractères sur une matrice de 8 bits sur 8 dont le codage commence d'abord par l'octet du bas. Dans le générateur, la lettre A est donc représentée, dans l'ordre, par 00H, 42H, 42H, 7EH, 42H, 24H, 18H, 00H, et ce, à partir de $E94D pour TO7(70) et TO9, et à partir de $FDA6 pour MO5. Vérifiez-le.

2. Réalisation :

Il s'agit d'un programme écrit en basic.

a. TO7(70) et TO9 :

```
10 N=&H4000:LOCATE 0,0,0
20 W=PEEK(&H60CF)*256+PEEK(&H60D0)'sur
TO7 mettre W=&HE845
30 CLS
40 POKE &HE7C3,PEEK(&HE7C3) OR 1
50 FOR I=N TO N+39'ecriture d'une ligne
60 Z=I
70 FOR A=W+7 TO W STEP -1'replacer les
caracteres a l'endroit
80 POKE Z,PEEK(A)'ecriture d'un octet
sur une ligne
90 Z=Z+40'ligne suivante
100 NEXT A
110 W=W+8'code affichable suivant
120 IF W=&HEBE5 THEN LOCATE 0,4:END' OU
W=&HEBF5 SUR TO9
130 NEXT I
140 N=N+(40*10)'40 fois(8 bits verticaux
 + 2 lignes d'espacement)
150 GOTO 50' 2e et 3e lignes a ecrire.
```

b. MO5 :

```
10 N=&H0000:LOCATE 0,0,0
20 W=PEEK(&H2073)*256+PEEK(&H2074)
30 CLS
40 POKE &HA7C0,PEEK(&HA7C0) OR 1
50 FOR I=N TO N+39'ecriture d'une ligne
60 Z=I
70 FOR A=W+7 TO W STEP -1'replacer les
caracteres a l'endroit
80 POKE Z,PEEK(A)'ecriture d'un octet
sur une ligne
90 Z=Z+40'ligne suivante
100 NEXT A
110 W=W+8'code affichable suivant
120 IF W=&HFFC6 THEN LOCATE 0,4:END
130 NEXT I
140 N=N+(40*10)'40 fois(8 bits verticaux
 + 2 lignes d'espacement)
150 GOTO 50' 2e et 3e lignes a ecrire.
```

3. Explications :

Peu d'explications à donner ici puisqu'elles ont été fournies dans le paragraphe DESCRIPTION, ceci afin de faciliter la compréhension immédiate de ce programme.

Ceux qui sont attentifs auront remarqué une légère différence entre les données en DATA du programme GRAPHISME DE A du début de l'ouvrage, et les octets du générateur. Ceci montre qu'avec un peu d'expérience on peut non seulement redéfinir son alphabet (nous allons l'étudier dans le programme qui suit), mais aussi redéfinir le graphisme de l'alphabet standard, selon ses goûts ou ses besoins.

Le générateur G2 du TO9 contenant 2 caractères supplémentaires par rapport à celui du TO7-70, il est nécessaire, sur TO9, de modifier la ligne 120 du programme lui ayant trait.

PROGRAMME N° VIII

1. Description :

Ce programme ne peut pas fonctionner sur TO7. Le générateur G2 du TO9 contient 2 caractères de plus que celui du TO7-70 ; nous en tiendrons compte dans le programme qui suit. Sur MO5 le G2 est plus restreint ; nous garderons cependant la version TO7-70 du programme pour lui.

Définir son alphabet personnel : soit donc, à titre d'exemple, à inverser tous les caractères affichables visualisés lors du programme précédent. Les 26 lettres de l'alphabet seront donc écrites à « l'envers » : ∀, B, C, D, E , ∃, Ɔ, etc..

Notre nouveau générateur de caractères comprendra les 96 caractères inversés du G0, plus les 20 caractères inversés du G2, du moins sur TO7-70, ce qui fait 116 caractères. Il faudra donc lui réserver 928 ou $3A0 octets (116*8) de mémoire sur TO7-70 et MO5, et $3B0 octets sur TO9.

Son adresse devra être mise en :

a. TO7-70 et TO9 : PTGENE ($60CF-$60D0)

b. MO5 : GENPTR ($2073-$2074)

Nous choisirons la fin de mémoire moins $3A0 octets (ou $3B0 sur TO9) comme adresse de début de ce nouveau générateur. La fin de mémoire est définie en assembleur par l'étiquette ENDMEM, qui est égale à la dernière adresse-utilisateur + 1, soit :

a. sur TO7-70 et TO9 : $DFFF + 1 = $E000 (sans tenir compte des banques-mémoire commutables automatiquement ou non).

b. sur MO5 : $9FFF + 1 = $A000

Note : sur TO7-70 et TO9, ENDMEM = $E000. Dans le moniteur, vous aurez :

ENDMEM = 0E000. En effet, le moniteur n'écrit pas le ''$'' car la commande IH (input hexadécimal) est implicite. De plus, lorsqu'un nombre hexadécimal commence par une lettre, le moniteur met un 0 devant celle-ci pour bien faire remarquer que cette « lettre » est en fait un chiffre hexadécimal.

Cette note est valable pour le $ A000 du MO5.

Notre générateur partira donc de ENDMEM-$3A0 (ou ENDMEM-$3B0 sur TO9) et ira jusqu'au dernier octet disponible. Nous implanterons cette fois-ci notre programme en &31000 pour varier l'adresse de départ.

Pour inverser les caractères des G0 et G2, il faudra inverser la matrice de 8 octets de chacun d'entr'eux et la replacer dans le nouveau générateur.

Afin de visualiser notre alphabet un peu particulier, nous écrirons le mot TABLE par la directive FCC, entre deux délimiteurs identiques : /TABLE/. Ces délimiteurs pourraient aussi bien être : %TABLE% ou d'autres caractères affichables.

Tout ceci nous ouvre des horizons sur les alphabets grec, russe, arabe, chinois, sur l'écriture gothique, etc... Pourquoi pas ?

En basic, la fin de mémoire étant à respecter, reculez le générateur de 512 octets par exemple :

- Sur TO7(70) : "DC60" deviendra "DA60", et "E000", "DE00".
- Sur TO9 : "DC50" deviendra "DA50", et "E000", "DE00".
- Sur MO5 : "9C60" deviendra "9A60", et "A000", "9E00".
- Faites RAZ, avant de lancer le programme.

2. Réalisation :

a. TO7(70) et TO9 :

```
          60CF     PTGENE EQU    $60CF
          E803     PUTCH  EQU    $E803

7918                      ORG    &31000
7918 BE   60CF            LDX    PTGENE
791B 108E DC60            LDY    #ENDMEM-$3A0
791F 10BF 60CF            STY    PTGENE
7923 5F                   CLRB
7924 31   28       D1     LEAY   8,Y
7926 A6   80       D2     LDA    ,X+
7928 A7   A2              STA    ,-Y
792A 5C                   INCB
792B C1   08              CMPB   #8
792D 26   F7              BNE    D2
792F 5F                   CLRB
7930 31   28              LEAY   8,Y
7932 108C E000            CMPY   #ENDMEM
7936 26   EC              BNE    D1
7938 CE   7947            LDU    #TABLE
793B E6   C0       D3     LDB    ,U+
793D C1   04              CMPB   #4
793F 27   05              BEQ    FIN
```

```
7941 BD   E803            JSR    PUTCH
7944 20   F5              BRA    D3
7946 3F           FIN     SWI
7947 1F 48 54     TABLE   FCB    $1F,$48,$54
794A 54 41 42 4C          FCC    /TABLE/
794E 45
794F 04                   FCB    $4
         0000             END

00000 Total Errors
```

Sur TO9, remplacer $3A0 par $3B0.

b. MO5 :

```
         2073     GENPTR EQU    $2073
         0002     PUTCH  EQU    2

7918                     ORG    &31000
7918 BE   2073           LDX    GENPTR
791B 108E 9C60           LDY    #ENDMEM-$3A0
791F 10BF 2073           STY    GENPTR
7923 5F                  CLRB
7924 31   28       D1    LEAY   8,Y
7926 A6   80       D2    LDA    ,X+
7928 A7   A2             STA    ,-Y
792A 5C                  INCB
792B C1   08             CMPB   #8
792D 26   F7             BNE    D2
792F 5F                  CLRB
7930 31   28             LEAY   8,Y
7932 108C A000           CMPY   #ENDMEM
7936 26   EC             BNE    D1
7938 CE   7948           LDU    #TABLE
793B E6   C0       D3    LDB    ,U+
793D C1   04             CMPB   #4
793F 27   04             BEQ    FIN
7941 3F                  SWI
7942 02                  FCB    PUTCH
7943 20   F6             BRA    D3
7945 BD   B000     FIN   STOP
7948 1F 48 54      TABLE FCB    $1F,$48,$54
794B 54 41 42 4C         FCC    /TABLE/
794F 45
7950 04                  FCB    $4
         0000            END

00000 Total Errors
```

Attention : En basic, n'oubliez pas de remplacer "BD B000" par "12, 12, 39".

3. Explications :

Retournez voir le programme dans la page-éditeur (X et ENTREE). En basic, tapez quelques mots au clavier et ajoutez EXEC 31000, suivi de LIST, dans le programme (après initialisation au besoin). Vous pourrez déchiffrer le texte à l'aide d'un miroir, en vous penchant sur le haut de celui-ci.

N'éteignez pas votre ordinateur, mais faites :

a. En basic (après initialisation) :

En mode-programmation, avant LIST,

— *TO7-70 et TO9 :* POKE &H60CF, &HE8
POKE &H60D0, &H45

— *MO5* : POKE&H2073, &HFC
POKE&H2074, &H9E

b. En assembleur (dans le moniteur, en vous fiant au clavier) :

— *TO7-70 et TO9 :* N2 et ENTREE, puis 60CF/ et remplacez DC60 (ou DC50 sur TO9) par E845 + ENTREE.

— *MO5 :* N2 et ENTREE, puis 2073/ et remplacez 9C60 par FC9E + ENTREE.

Notez qu'avec N2 on écrit (et on visualise) 2 octets.

Voilà, tout est rentré dans l'ordre.

Remarquez que l'on peut mettre des virgules entre les différentes constantes, et les implanter par une seule directive FCB. Il faut savoir aussi que l'on peut remplacer un FCC /TABLE/ par FCB 'T, 'A, 'B, 'L, 'E où 'T , par exemple, signifie code ASCII (apostrophe avant la lettre) de T (lettre à écrire).

Notes pour la cartouche-assembleur :

%N% et /N/ sont des délimiteurs où N est le mot à écrire. Ces délimiteurs doivent être, rappelons-le, identiques avant et après le mot. Les délimiteurs peuvent être des espaces, comme tout autre caractère affichable. Cependant l'espace est à éviter, car on risque soit de l'oublier, soit de ne pouvoir en utiliser un seul à l'intérieur d'une phrase.

On peut supprimer la tabulation automatique, gênante lors de l'écriture d'une phrase par la directive FCC, en faisant CNT-S. Alors, apparaît un carré bleu (égal à un espace seulement) en mode vidéo-inverse. Bien sûr, un espace peut aussi être créé par la flèche : →. Sachez

également que la zone des commentaires peut servir d'extension à la zone-opérande, dans son intégralité.

Une ligne permet l'écriture de 40 caractères. Signalons qu'une note se fait entendre lorsque vous écrivez le 38e, puis une note plus aiguë lors de l'écriture du 40e. C'est sur le dernier espace maximum qu'il faut mettre le délimiteur (puisque chaque ligne définie par FCC nécessite 2 délimiteurs : un à son début et un à sa fin), même au milieu d'un mot.

Essayez :

```
TABLE       FCC    %J'AIME L'INFORMATIQUE ET%
            FCC    5JE SUIS HEUREUX DE PRATIS
            FCC    2QUER LE LANGAGE-MACHINE2
            FCB    $4
            END
```

Il est quand même préférable d'éviter les ''5'' ou les ''2'' en tant que délimiteurs. Ils n'ont été employés, dans l'exemple précédent, qu'à titre de démonstration. A signaler qu'on ne peut écrire en lettres minuscules que dans la dernière zone, mais pas dans la zone de l'opérande.

L'astérisque, en tout début d'une ligne, signifie : commentaire. La tabulation automatique est alors supprimée derrière l'astérisque et la ligne n'est pas prise en compte.

Exemple :

```
*PROGRAMME D'ESSAI

            ORG &32000

            etc.
```

Explications sur le programme-même :

- On charge dans le registre X l'adresse du générateur standard, et dans ''Y'' l'adresse du début d'implantation de notre nouveau générateur. On place cette nouvelle adresse dans le registre PTGENE OU GENPTR.
- CLRB = CLEAR B, soit mettre ''B'' à zéro. ''B'' servira de compteur pour chaque caractère.

• LEAY 8, Y = LOAD EFFECTIVE ADDRESS, c'est-à-dire charge l'adresse effective (''Y'' = ''Y'' + 8), soit ''Y'' déplacé de 8 octets, dans ce même registre Y. L'adressage est dit indexé, avec déplacement constant. Le pas de déplacement de l'index (8) est aussi appelé OFFSET. Après cette opération, l'adresse de ''Y'' est donc avancée de 8 octets.

Vous connaissez l'adressage autoincrémenté (LDA ,X+) ; dans STA ,–Y, l'adressage est autodécrémenté. Attention : rappelons que lors d'une autoincrémentation, l'incrémentation a lieu après la réalisation de l'instruction, mais que dans l'autodécrémentation, la décrémentation précède la réalisation de l'instruction. Ce rappel vous aidera à bien comprendre le programme.

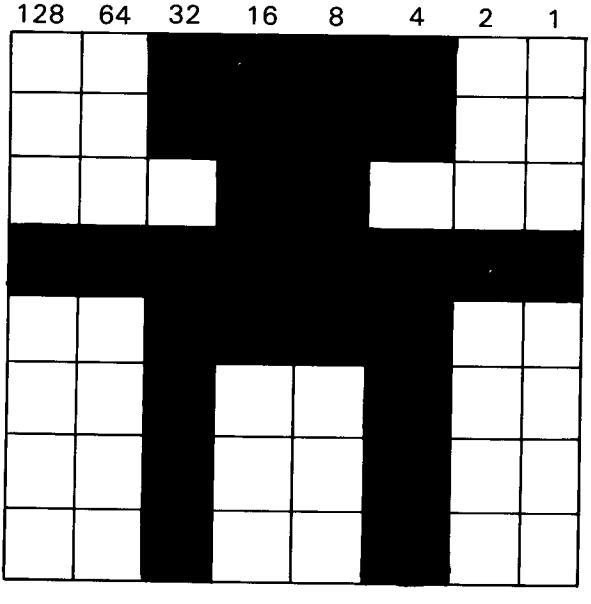
• INCB signifie bien sûr : incrémente l'accumulateur B de une unité.

PROGRAMME N° IX

1. Description :

Gestion des caractères-utilisateur.

Il s'agit de définir des GR$(x). La technique est la même que pour dessiner la lettre A ; ici nous dessinerons un pantin.



32 + 16 + 8 + 4 = 60
32 + 16 + 8 + 4 = 60
16 + 8 = 24
255
32 + 16 + 8 + 4 = 60
32 + 4 = 36
32 + 4 = 36
32 + 4 = 36

2. Réalisation :

A. En basic :

```
10 CLEAR,,1
20 CLS
30 DEFGR$(0)=60,60,24,255,60,36,36,36'
en basic: du haut vers le bas.
40 LOCATE 20,12,0:PRINTGR$(0)
50 END
```

B. En assembleur :

— 1. *On peut garder la définition en basic, pour ne faire que l'exécution en machine :*

```
10 CLEAR,,1
20 CLS
30 DEFGR$(0)=60,60,24,255,60,36,36,36
40 FOR I=32000 TO 32000+X OCTETS------->
'voir ci apres(2)
50 READ A$
60 POKE I,VAL("&H"+A$)
70 NEXT I
80 DATA------->voir ci-apres(2)
90 CLS:EXEC32000
100 'Sur TO7(70) et TO9, remplacer "3F"
par "39".
110 'Sur MO5, remplacer "BD,B0,00" Par
"12,12,39".
```

— 2. *Exécution seule, en machine :*

a. TO7(70) et TO9 :

```
          E803     PUTCH  EQU     $E803

7D00                      ORG     &32000
7D00 CE   7D0F            LDU     #TABLE
7D03 E6   C0       DEBUT  LDB     ,U+
7D05 C1   04              CMPB    #4
7D07 27   05              BEQ     FIN
7D09 BD   E803            JSR     PUTCH
7D0C 20   F5              BRA     DEBUT
7D0E 3F            FIN    SWI
7D0F 1F            TABLE  FCB     $1F      US
7D10 4C                   FCB     $4C      .L=12
7D11 54                   FCB     $54      .C=21
7D12 14                   FCB     $14      EFFACEMENT CURSEUR
7D13 80                   FCB     $80      CODE 1ER GR$
7D14 04                   FCB     $4       E.O.T.
          0000            END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH   EQU     2

7D00                       ORG     &32000
7D00 CE   7D10             LDU     #TABLE
7D03 E6   C0       DEBUT   LDB     ,U+
7D05 C1   04               CMPB    #4
7D07 27   04               BEQ     FIN
7D09 3F   02               CALL    PUTCH
7D0B 20   F6               BRA     DEBUT
7D0D BD   B000     FIN     STOP
7D10 1F            TABLE   FCB     $1F       US
7D11 4C                    FCB     $4C       .L=12
7D12 54                    FCB     $54       .C=21
7D13 14                    FCB     $14       EFFACEMENT CURSEUR
7D14 80                    FCB     $80       CODE 1ER GR$
7D15 04                    FCB     $4        E.O.T
          0000             END

00000 Total Errors
```

3. Explications :

Il faut remarquer que l'effacement du curseur est codé par $14. Il s'agit d'un code interprétable et non d'un attribut, donc ESC = $1B n'est pas nécessaire.

Le code $80 permet l'affichage du 1er GR$. Le 2e GR$ serait codé $81, etc. jusqu'à 128 caractères-utilisateur en tout ($FF = dernier GR$).

L'adresse où le programme doit prendre ses GR$ (de $80 à $FF) doit se situer dans le registre USERAF (ceci est fait automatiquement lors de la définition d'un DEFGR$(x) en basic). Voici l'adresse de ce registre :

a. TO7(70) et TO9 : $602D-$602E

b. MO5 : $2070-$2071

Pour réaliser un DEFGR$(X) en langage-machine, il faut donc définir une place pour le générateur de caractères-utilisateur, par exemple ENDMEM – (x caractères * 8 octets par caractère), et situer l'adresse du début de ce générateur dans USERAF.

Une autre routine permet la réalisation d'un GR$(x). Il s'agit de CHPLH. Sur TO7(70) et TO9 CHPLH = $E833, et sur MO5 CHPLH a le code n° $12. Nous la verrons par la suite.

PROGRAMME N° X

1. Description :

Définir et dessiner un GR$(x), en « machine ».

Soit ENDMEM-8 l'adresse du début de notre générateur. Plaçons-y notre DEFGR$(x) en langage-machine, puis « pokons » le registre USERAF avec cette adresse. Enfin, replaçons et exécutons le programme précédent.

2. Réalisation :

a. TO7(70) et TO9 :

```
            602D     USERAF EQU    $602D
            E803     PUTCH  EQU    $E803

7D00                        ORG    &32000
7D00 8E     DFF8            LDX    #ENDMEM-8
7D03 BF     602D            STX    USERAF
7D06 CE     7D22            LDU    #TABLE1
7D09 A6     C0      DEBUT1  LDA    ,U+
7D0B 81     04              CMPA   #4
7D0D 27     04              BEQ    SUITE
7D0F A7     80              STA    ,X+
7D11 20     F6              BRA    DEBUT1
7D13 CE     7D2B    SUITE   LDU    #TABLE2
7D16 E6     C0      DEBUT2  LDB    ,U+
7D18 C1     04              CMPB   #4
7D1A 27     05              BEQ    FIN
7D1C BD     E803            JSR    PUTCH
7D1F 20     F5              BRA    DEBUT2
7D21 3F                 FIN SWI
7D22 24 24 24 3C    TABLE1  FCB    36,36,36,60      .DECIMAL &
7D26 FF 18 3C 3C            FCB    255,24,60,60,$4  .AL'ENVERS
7D2A 04
7D2B 1F 4C 54       TABLE2  FCB    $1F,$4C,$54
7D2E 14 80 04               FCB    $14,$80,$4
             0000           END

00000 Total Errors
```

b. MO5 :

```
           2070     USERAF EQU    $2070
           0002     PUTCH  EQU    2

7D00                       ORG    &32000
7D00 8E    9FF8            LDX    #ENDMEM-8
7D03 BF    2070            STX    USERAF
7D06 CE    7D23            LDU    #TABLE1
7D09 A6    C0       DEBUT1 LDA    ,U+
7D0B 81    04              CMPA   #4
7D0D 27    04              BEQ    SUITE
7D0F A7    80              STA    ,X+
7D11 20    F6              BRA    DEBUT1
7D13 CE    7D2C     SUITE  LDU    #TABLE2
7D16 E6    C0       DEBUT2 LDB    ,U+
7D18 C1    04              CMPB   #4
7D1A 27    04              BEQ    FIN
7D1C 3F    02              CALL   PUTCH
7D1E 20    F6              BRA    DEBUT2
7D20 BD    B000     FIN    STOP
7D23 24 24 24 3C   TABLE1 FCB    36,36,36,60       .DECIMAL &
7D27 FF 18 3C 3C           FCB    255,24,60,60,$4   .AL'ENVERS
7D2B 04
7D2C 1F 4C 54      TABLE2 FCB    $1F,$4C,$54
7D2F 14 80 04              FCB    $14,$80,$4
           0000            END

00000 Total Errors
```

3. Explications :

Peu d'explications à donner en fait, car la non compréhension de ce programme nécessiterait un retour en arrière.

Nous animerons notre petit pantin plus tard.

Si la méthode consistant à placer l'adresse de TABLE1 dans USERAF paraît plus économique à première vue, et même plus élégante, elle oblige cependant à laisser les FCB à la même adresse, ou à tout réassembler, lorsqu'on désire étoffer le programme.

Il est bien sûr possible de dessiner 4 GR$(x) côte à côte, en les définissant en langage-machine. Nous verrons cela lors de l'étude de la routine CHPLH, mieux adaptée que PUCTH pour les GR$(x).

On peut indifféremment utiliser l'accumulateur A ou B pour charger la TABLE1. Par contre, pour la TABLE2, il est nécessaire de se servir de l'accumulateur B pour s'adresser à la routine PUTCH.

Pour éviter de faire une conversion décimal-hexadécimal, les octets

du dessin ont été placés en décimal dans la TABLE 1. Le système les a convertis lui-même. N'oubliez pas qu'en « machine » une matrice de 8*8 bits se définit à partir de l'octet du bas (comme dans le générateur standard).

Il y a enfin un $4 = E.O.T. conventionnel à la fin de chaque table.

RETENIR : Pour charger un registre, peu importe l'accumulateur, mais pour « sauter » à une routine, il faut se servir du paramètre d'entrée : ici l'accumulateur B pour PUTCH.

2. Comparaisons entre basic et langage-machine

Ces comparaisons seront faites à propos de la mémoire-écran. Ce sera l'occasion d'une étude de structuration verticale d'un programme.

Sur TO9, ce chapitre n'est valable qu'en mode 40 colonnes, compatible avec le TO7.

Nous écrirons d'abord en basic des caractères sans couleur, puis le même programme sera réalisé avec une partie-machine pour comparer les vitesses d'exécution. Enfin, un programme de même type que le quadrillage d'écran de la partie n° III (écrit en basic) sera réalisé en « machine », de manière très structurée.

PROGRAMME N° XI

1. Description :

Programme-basic d'écriture de caractères sans couleur. Pour obtenir l'absence de couleur, il faut envoyer ESC = $ 1B tout d'abord à la routine PUTCH, ce qui se traduit par : ?CHR$(27) en basic, dans lequel 27 est le code ASCII de ESC, en décimal (tableau n° 5). Il faut donner ensuite, à la même routine, le code de l'attribut « caractères sans couleur » : ?CHR$(&H68)*. Mais attention, les 2 CHR$(x) sont à mettre sur la même ligne, et à séparer par un point-virgule pour être interprétés ensemble. En effet, l'interpréteur-basic ne traduit qu'une seule ligne à la fois, et sur la même ligne qu'une seule instruction à la fois (chaque instruction étant sur une même ligne séparée d'une autre par ":"), d'où la nécessité du ";" car ESC et l'attribut doivent se suivre lors de l'appel de PUTCH.

Nous choisirons un fond blanc, une couleur de forme rouge pour le haut de l'écran, et bleue pour le bas. Enfin, pour voir nos caractères sans couleur défiler lentement, nous travaillerons en scroll lent. Ces caractères prendront la couleur rouge ou bleue, selon qu'ils seront en haut ou en bas de l'écran (les mêmes caractères !).

Mais pourquoi utiliser le basic ? Parce qu'il ne faut pas le perdre de vue, et savoir transposer le langage-machine en basic.

2. Réalisation :

a. TO7(70) et TO9 :

```
10 CLS:SCREEN2,0,0:PRINTCHR$(27);CHR$(&H
6E)'scroll lent
20 PRINTCHR$(27);CHR$(&H68)'caracteres
sans couleur
30 LOCATE0,0,0
40 POKE &HE7C3,PEEK(&HE7C3)AND 254'mise
en memoire-couleur
50 FOR I=&H4000 TO &H4000+(8000/2)-1'
moitie superieure de l'ecran
60 POKE I,&B11001111'rouge-forme sur
fond blanc, en binaire
70 NEXT I
80 FOR I=&H4000+(8000/2) TO &H5F3F'
moitie inferieure de l'ecran
90 POKE I,&B11100111'bleu-forme sur fond
blanc, en binaire
```

* CHR$(&H68) sur TO7(70) ou TO9, et CHR$(&H75) sur MO5.

```
100 NEXT I
110 FOR I=1 TO 100
120 PRINT"Bonjour monsieur"
130 NEXT I
```

b. MO5 :

```
10 CLS:SCREEN2,0,0:PRINTCHR$(27);CHR$(&H
79)'scroll lent
20 PRINTCHR$(27);CHR$(&H75)'caracteres
sans couleur
30 LOCATE0,0,0
40 POKE &HA7C0,PEEK(&HA7C0)AND 254'mise
en memoire-couleur
50 FOR I=&H0000 TO &H0000+(8000/2)-1'
  moitie superieure de l'ecran
60 POKE I,&H17'rouge-forme sur fond
blanc, en binaire
70 NEXT I
80 FOR I=&H0000+(8000/2) TO &H1F3F'
moitie inferieure de l'ecran
90 POKE I,&H47'bleu-forme sur fond
blanc, en binaire
100 NEXT I
110 FOR I=1 TO 100
120 PRINT"Bonjour monsieur"
130 NEXT I
```

3. Explications :

Rien de spécial à signaler, si ce n'est de rappeler que l'on peut mettre la couleur de forme et de fond directement dans les octets de mémoire-écran. Le binaire est bien adapté pour écrire chaque bit, l'un après l'autre, sur TO7(70) et TO9 (sur MO5, il faut faire une conversion en hexadécimal).

A noter que, pour la partie supérieure de l'écran, l'adresse du dernier octet à remplir est égale à l'adresse du premier plus 3999, une moitié d'écran faisant 4 000 octets.

PROGRAMME N° XII

1. Description :

Le précédent programme va être ici réalisé avec une partie-machine, ceci afin de comparer les vitesses d'exécution du basic et du « machine ».

2. Réalisation :

A. Partie commune en basic :

```
10 CLS:SCREEN2,0,0:PRINTCHR$(27);CHR$(&H
6E)'**
20 PRINTCHR$(27);CHR$(&H68)'**
30 FOR I=&H7D00 TO &H7D1D'**
40 READ A$
50 POKE I,VAL("&H"+A$)
60 NEXT I:EXEC32000
70 DATA-------->voir ci-apres
80 FOR I=1 TO 100
90 PRINT"Bonjour monsieur"
100 NEXT I
110 END
120 '** sur MO5,faites les modifications
necessaires.
```

B. Parties-machines :

a. DATA pour TO7(70) et TO9 :

```
7D00                        ORG     &32000
7D00 8E   4000              LDX     #$4000
7D03 B6   E7C3              LDA     $E7C3
7D06 84   FE                ANDA    #&254
7D08 B7   E7C3              STA     $E7C3
7D0B 86   CF                LDA     #$CF
7D0D A7   80        DEBUT1  STA     ,X+
7D0F 8C   4FA0              CMPX    #$4000+8000/2
7D12 26   F9                BNE     DEBUT1
7D14 86   E7                LDA     #$E7
7D16 A7   80        DEBUT2  STA     ,X+
7D18 8C   5F40              CMPX    #$5F40
7D1B 26   F9                BNE     DEBUT2
7D1D 3F                     SWI
          0000              END

00000 Total Errors
```

b. DATA pour MO5 :

```
7D00                        ORG     &32000
7D00 8E   0000              LDX     #$0000
7D03 B6   A7C0              LDA     $A7C0
7D06 84   FE                ANDA    #&254
7D08 B7   A7C0              STA     $A7C0
7D0B 86   17                LDA     #$17
7D0D A7   80        DEBUT1  STA     ,X+
7D0F 8C   0FA0              CMPX    #$0000+8000/2
7D12 26   F9                BNE     DEBUT1
7D14 86   47                LDA     #$47
7D16 A7   80        DEBUT2  STA     ,X+
7D18 8C   1F40              CMPX    #$1F40
7D1B 26   F9                BNE     DEBUT2
7D1D BD   B000              STOP
          0000              END

00000 Total Errors
```

3. Explications :

On découvre l'instruction ANDA, avec la valeur qui lui est associée dans le champ-opérande. Ceci correspond à une opération de type ET LOGIQUE en basic. Cette opération permet de forcer à zéro le bit n° 0 de l'octet contenu en $E7C3 sur TO7(70) et TO9, ou $A7C0 sur MO5*. L'accumulateur A sert au chargement, à la transformation et à la remise en place de cet octet.

On charge dans l'accumulateur A le code binaire de la couleur rouge (forme) sur fond blanc, exprimé en hexadécimal, et on le dépose dans les octets de la partie supérieure de l'écran.

On charge encore dans l'accumulateur A le code binaire de la couleur bleue (forme) sur fond blanc, exprimé en hexadécimal, et on le dépose dans les octets du bas de l'écran.

C'est toute cette partie, longue en basic, qui va être améliorée par la rapidité du langage-machine. Le reste du programme-basic, performant, ne changera pas, à moins de vouloir bien structurer l'ensemble totalement en langage-machine. Ce sera le but du programme suivant.

* Sur MO5, le passage en mémoire-caractère peut se faire plus simplement, par l'appel à la routine de code n° 6, et le passage en mémoire-couleur par l'appel à la routine de code n° 4.

Lorsque l'instruction CMPX #$4000 + 8000/2 sur TO7(70) et TO9 ou CMPX #$0000 + 8000/2 sur MO5 est vérifiée, cela signifie que la moitié supérieure de l'écran est remplie : en effet, l'autoincrémentation de "X" se faisant après l'instruction STA ,X+, la comparaison doit se faire avec l'adresse qui suit celle du dernier octet de la moitié supérieure de l'écran ($4000 + 8000/2). On remplit alors les octets de la partie inférieure de l'écran.

PROGRAMME N° XIII

1. Description :

Nous allons réaliser un programme de même type que celui du quadrillage d'écran de la partie n° III, mais en langage-machine. Les couleurs seront déterminées ainsi : rouge (forme) pour la moitié supérieure de l'écran, et bleue (forme) pour sa moitié inférieure. Nous remplirons tous les octets dans la forme (255), d'abord pour la partie supérieure, puis pour la partie inférieure. Nous éviterons ici de faire des pointillés et surtout de varier les couleurs, afin de ne pas allonger inutilement ce programme.

2. Réalisation :

A. Programme non structuré :

a. TO7(70) et TO9 :

```
7D00                        ORG     &32000
7D00 8E   4000              LDX     #$4000
7D03 B6   E7C3              LDA     $E7C3
7D06 84   FE                ANDA    #&254
7D08 B7   E7C3              STA     $E7C3
7D0B 86   C8                LDA     #$C8
7D0D A7   80      DEBUT1    STA     ,X+
7D0F 8C   4FA0              CMPX    #$4000+8000/2
7D12 26   F9                BNE     DEBUT1
7D14 8E   4000              LDX     #$4000
7D17 B6   E7C3              LDA     $E7C3
7D1A 8A   01                ORA     #1
7D1C B7   E7C3              STA     $E7C3
7D1F 86   FF                LDA     #&255
7D21 A7   80      DEBUT2    STA     ,X+
7D23 8C   4FA0              CMPX    #$4000+8000/2
7D26 26   F9                BNE     DEBUT2
7D28 B6   E7C3              LDA     $E7C3
7D2B 84   FE                ANDA    #&254
7D2D B7   E7C3              STA     $E7C3
7D30 86   E0                LDA     #$E0
7D32 A7   80      DEBUT3    STA     ,X+
7D34 8C   5F40              CMPX    #$5F40
7D37 26   F9                BNE     DEBUT3
7D39 8E   4FA0              LDX     #$4000+8000/2
7D3C B6   E7C3              LDA     $E7C3
7D3F 8A   01                ORA     #1
7D41 B7   E7C3              STA     $E7C3
7D44 86   FF                LDA     #&255
7D46 A7   80      DEBUT4    STA     ,X+
7D48 8C   5F40              CMPX    #$5F40
```

```
7D4B 26   F9            BNE     DEBUT4
7D4D 3F                 SWI
         0000           END

00000 Total Errors
```

B. Programmes structurés :

a. TO7(70) et TO9 :

```
                        * CORPS DE PROGRAMME

7D00                    ORG     &32000
7D00 BD   7D1D          JSR     A1
7D03 BD   7D23          JSR     A2
7D06 86   C8            LDA     #$C8
7D08 BD   7D2E          JSR     A3
7D0B 86   E0            LDA     #$E0
7D0D BD   7D38          JSR     A4
7D10 BD   7D1D          JSR     A1
7D13 BD   7D42          JSR     A5
7D16 BD   7D2E          JSR     A3
7D19 BD   7D38          JSR     A4
7D1C 3F           FIN   SWI

                        * SOUS-PROGRAMMES

7D1D 8E   4000    A1    LDX     #$4000
7D20 7E   7D4C          JMP     RET
7D23 B6   E7C3    A2    LDA     $E7C3
7D26 84   FE            ANDA    #&254
7D28 B7   E7C3          STA     $E7C3
7D2B 7E   7D4C          JMP     RET
7D2E A7   80      A3    STA     ,X+
7D30 8C   4FA0          CMPX    #$4000+8000/2
7D33 26   F9            BNE     A3
7D35 7E   7D4C          JMP     RET
7D38 A7   80      A4    STA     ,X+
7D3A 8C   5F40          CMPX    #$5F40
7D3D 26   F9            BNE     A4
7D3F 7E   7D4C          JMP     RET
7D42 B6   E7C3    A5    LDA     $E7C3
7D45 8A   01            ORA     #1
7D47 B7   E7C3          STA     $E7C3
7D4A 86   FF            LDA     #&255
7D4C 39           RET   RTS
         0000           END

00000 Total Errors
```

b. MO5 :

```
                        *CORPS DE PROGRAMME

7D00                            ORG     &32000
7D00 BD   7D1F                  JSR     A1
7D03 BD   7D25                  JSR     A2
7D06 86   10                    LDA     #$10
7D08 BD   7D30                  JSR     A3
7D0B 86   40                    LDA     #$40
7D0D BD   7D3A                  JSR     A4
7D10 BD   7D1F                  JSR     A1
7D13 BD   7D44                  JSR     A5
7D16 BD   7D30                  JSR     A3
7D19 BD   7D3A                  JSR     A4
7D1C BD   B000                  STOP

                        *SOUS-PROGRAMMES

7D1F 8E   0000    A1            LDX     #$0000
7D22 7E   7D4E                  JMP     RET
7D25 B6   A7C0    A2            LDA     $A7C0
7D28 84   FE                    ANDA    #&254
7D2A B7   A7C0                  STA     $A7C0
7D2D 7E   7D4E                  JMP     RET
7D30 A7   80      A3            STA     ,X+
7D32 8C   0FA0                  CMPX    #$0000+8000/2
7D35 26   F9                    BNE     A3
7D37 7E   7D4E                  JMP     RET
7D3A A7   80      A4            STA     ,X+
7D3C 8C   1F40                  CMPX    #$1F40
7D3F 26   F9                    BNE     A4
7D41 7E   7D4E                  JMP     RET
7D44 B6   A7C0    A5            LDA     $A7C0
7D47 8A   01                    ORA     #1
7D49 B7   A7C0                  STA     $A7C0
7D4C 86   FF                    LDA     #&255
7D4E 39           RET           RTS
          0000                  END

00000 Total Errors
```

3. Explications :

Le programme non structuré, linéaire, n'a été réalisé que pour TO7(70) et TO9.

Il est ensuite nécessaire de passer du linéaire au structuré, car ne pas concevoir un programme-machine par « étages » est difficilement envisageable, tant on risque de s'y perdre et de ne pouvoir le réaliser en groupe, par « modules » successifs. Par contre, un corps de pro-

gramme peut toujours être étoffé par des sous-routines, même anciennement réalisées et sauvegardées sur support magnétique.

La création d'un sous-programme n'est en général utile que lorsqu'on doit y faire appel au moins deux fois. En ce sens, les étiquettes A2 et A5 n'ont donc pas tellement d'intérêt. Quant à l'étiquette RET, elle n'est utilisée que pour faire au moins une fois des JMP (GOTO en basic), dans un des programmes de démonstration de cet ouvrage.

JMP est un « saut » inconditionnel à une adresse. Il vaut mieux, en général, employer JSR, surtout pour accéder aux routines du moniteur*. Il faut savoir aussi que JMP comme JSR devront être de préférence employés avec une étiquette, pour pouvoir translater l'adresse de celle-ci en cas de réassemblage.

Tout appel par JSR se termine bien sûr par RTS (GOSUB et RETURN en basic).

* ou le code qui lui correspond, sur MO5, lors de l'accès à une routine du moniteur.

3. Routine NOTEH

A. DESCRIPTION

Comme son nom l'indique, cette routine sert à gérer toutes les possibilités musicales de votre appareil.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E81E
— Sur MO5 : code n° $1E

* *Paramètres d'entrée :*

— *Registre B du 6809*
— Registre OCTAVE : 16 bits
  • TO7(70) et TO9 : (6036H-6037H)
  • MO5 : (203EH-203FH)
— Registre DURÉE :
  • 16 bits sur TO7(70) et TO9 : (6033H-6034H)
  • 8 bits sur MO5 : (203CH)
— Registre TEMPO :
  • 16 bits sur TO7(70) et TO9 : (6031H-6032H)
  • 8 bits sur MO5 : (203AH)
— Registre TIMBRE : 8 bits
  • TO7(70) et TO9 : (6035H)
  • MO5 : (203DH).

NOTES	*CODES (TO7(70) et TO9) :*	*CODES (MO5) :*
silence (P en basic)	30H	00
DO	31H	01
DO #	32H	02
RE	33H	03
RE #	34H	04
MI	35H	05
FA	36H	06
FA #	37H	07

SOL	38H	08
SOL #	39H	09
LA	3AH	0AH
LA #	3BH	0BH
SI	3CH	0CH
UT (DO)	3DH	0DH

OCTAVES :	*CODES :*	
1 (la plus grave)	16	
2	8	
3	4	
4	2	= référence de base (octave du LA 440)
5	1	

DURÉE :	*CODES :*
Ronde	96
Blanche pointée	72
Blanche	48
Noire pointée	36
Noire	24
Croche pointée	18
Croche	12
Double croche pointée	09
Double croche	06
Triple croche	03
Ronde triolet	64
Blanche triolet	32
Noire triolet	16
Croche triolet	08
Double croche triolet	04
Triple croche triolet	02

— *TEMPO :* de 1 à 255. Il représente le mouvement du morceau à jouer. La durée réelle d'une note = Tempo * durée.

— *TIMBRE :* de 0 à 5. Il fournit l'attaque d'une note, par modification du rapport cyclique. La valeur 0 donne une note continue.

Ces codes sont donnés à titre de référence. Vous pouvez créer toutes les variations intermédiaires.

B. APPLICATION

PROGRAMME N° XIV

1. Description :

Faire tout simplement un RE, puis, aprés avoir chargé les 4 registres d'entrée autres que "B", faire une gamme complète (avec les dièses).

2. Réalisation :

a. Un RE sur TO7(70) et TO9 :

```
            E81E      NOTEH  EQU     $E81E

7D00                         ORG     &32000
7D00 C6     33               LDB     #$33
7D02 BD     E81E             JSR     NOTEH
7D05 3F                      SWI
            0000             END

00000 Total Errors
```

b. un RE sur MO5 :

```
            001E      NOTEH  EQU     $1E

7D00                         ORG     &32000
7D00 C6     03               LDB     #3
7D02 3F     1E               CALL    NOTEH
7D04 BD     B000             STOP
            0000             END

00000 Total Errors
```

c. Une gamme sur TO7(70) et TO9 :

```
          E81E     NOTEH  EQU     $E81E

7D00                      ORG     &32000
7D00 C6   01              LDB     #1
7D02 F7   6035            STB     $6035    TIMBRE
7D05 CC   0004            LDD     #4
7D08 FD   6031            STD     $6031    TEMPO
7D0B CC   0002            LDD     #2
7D0E FD   6036            STD     $6036    OCTAVE
7D11 CC   0018            LDD     #$18
7D14 FD   6033            STD     $6033    DUREE
7D17 C6   31              LDB     #$31     .FAIRE LA GAMME EN
7D19 BD   E81E    DEBUT   JSR     NOTEH    .INCREMENTANT
7D1C 5C                   INCB               .LE REGISTRE B
7D1D C1   3D              CMPB    #$3D
7D1F 26   F8              BNE     DEBUT
7D21 3F                   SWI
          0000            END

00000 Total Errors
```

d. Une gamme sur MO5 :

```
          001E     NOTEH  EQU     $1E

7D00                      ORG     &32000
7D00 C6   01              LDB     #1
7D02 F7   203D            STB     $203D    TIMBRE
7D05 86   04              LDA     #4
7D07 B7   203A            STA     $203A    TEMPO
7D0A CC   0002            LDD     #2
7D0D FD   203E            STD     $203E    OCTAVE
7D10 86   18              LDA     #$18
7D12 B7   203C            STA     $203C    DUREE
7D15 C6   01              LDB     #1       .FAIRE LA GAMME EN
7D17 3F   1E      DEBUT   CALL    NOTEH    .INCREMENTANT
7D19 5C                   INCB               .LE REGISTRE B
7D1A C1   0D              CMPB    #$0D
7D1C 26   F9              BNE     DEBUT
7D1E BD   B000            STOP
          0000            END

00000 Total Errors
```

3. Explications :

Remarquez que certains registres peuvent contenir deux octets, le premier, de poids le plus fort ou M.S.B. (MOST SIGNIFICANT BYTE), pouvant éventuellement servir. Le deuxième octet ou L.S.B. (LEAST SIGNIFICANT BYTE) est souvent le seul utilisé. En effet, mettez 1 en 6031H et 0 en 6032H, sur TO7(70) et TO9, cela fera 256 en décimal pour le tempo. Il s'agit là d'une durée très longue (durée réelle de la note = tempo * durée) et encore, faut-il que le timbre soit à 0 pour entendre la note continuellement.

Vous pouvez, dans le moniteur de l'assembleur, changer les valeurs des registres de la manière que vous connaissez : tapez N, puis TO7(70) et TO9 : 6037/2. Tapez x (compris entre 1 et FFH), etc..

Sur MO5, seul le registre OCTAVE peut contenir 16 bits.

Vous pouvez ensuite taper dans le moniteur, sans retourner dans l'éditeur :

Sur TO7(70) et TO9 : G 7D17H, ou sur MO5 : G 7D15H.

A noter enfin que l'on peut utiliser indifféremment les accumulateurs A ou B pour charger des registres de 8 bits, mais que "B" sert ici encore de paramètre d'entrée dans la routine. Tout ceci a déjà été dit pour la routine précédente, mais pour les latinistes : BIS REPETITA PLACENT.

4. Routine CHPLH

A. DESCRIPTION

Cette routine permet l'écriture d'un caractère affichable, aux coordonnées précisées par les registres X et Y, avec une instruction de type COLOR x,y donnée par le registre COLOUR.

Elle peut traiter, sur TO9, les modes d'affichage 40 et 80 colonnes, les modes page 1, page 2, overlay, bitmap 4 couleurs, mais pas les modes bitmap 16 couleurs ou triple overlay. Ce chapitre cependant, par souci de compatibilité, se référera au mode 40 colonnes. Pour les autres, voir le chapitre P9 en fin d'ouvrage.

L'avantage de cette routine, par rapport à PUTCH, est qu'elle permet simultanément la mise en place des coordonnées programmées, et l'affichage d'un caractère dans des couleurs de fond et de forme choisies préalablement.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E833
— Sur MO5 : code n° $12

* *Paramètres d'entrée :*

— *Registres X et Y :*

Ces registres doivent contenir les coordonnées, comprises en décimal entre 1 et 40 pour la colonne (registre X) et 0 et 24 pour la ligne (registre Y), du caractère à afficher.

— *Registre CHDRAW :*

Il doit contenir le code ASCII du caractère à afficher lorsqu'il s'agit d'un caractère standard. Pour un caractère-utilisateur, ce registre doit contenir son code (de $80 à $FF). Le générateur de caractères-utilisateur doit alors être positionné en mémoire, et son adresse contenue dans le registre USERAF.

- TO7(70) et TO9 : $6041
- MO5 : $2036

— *Pointeur USERAF :*

- TO7(70) et TO9 : ($602D-$602E)
- MO5 : ($2070-$2071)

— *Registre COLOUR :*

Ce registre permet de définir les couleurs courantes de FOND et de FORME pour les caractères à afficher. Le code des couleurs est le même que celui des octets de la R.A.M. de l'écran.

- TO7(70) et TO9 : $603B
- MO5 : $202B

* *Paramètres de retour :*

— Registres PLOTX et PLOTY :

Ces registres contiennent, après utilisation de la routine CHPLH, les coordonnées du dernier caractère écrit.

- PLOTX (1,40) sut TO7(70) et TO9 : ($603D-$603E)
- PLOTY (0,24) sur TO7(70) et TO9 : ($603F-$6040)
- PLOTX (1,40) sur MO5 : ($2032-$2033)
- PLOTY (0,24) sur MO5 : ($2034-$2035)

PROGRAMME N° XV

1. Description :

Nous allons dessiner de nouveau le pantin du programme n° IX, et l'animer. Nous étudierons aussi la fonction basic ATTRB, en langage-machine (tableau n° 7). Dans le programme qui suivra, nous verrons enfin de quelle façon réaliser 4 GR$(x) côte à côte, pour améliorer un graphisme.

2. Réalisation :

a. TO7(70) et TO9 :

```
                602D    USERAF  EQU     $602D
                E803    PUTCH   EQU     $E803
                6041    CHDRAW  EQU     $6041
                E833    CHPLH   EQU     $E833

7D00                            ORG     &32000
7D00 8E         7D59            LDX     #DEFGRO
7D03 BF         602D            STX     USERAF
7D06 CE         7D54            LDU     #TABLE
7D09 E6         C0      DEBUT   LDB     ,U+
7D0B C1         04              CMPB    #4
7D0D 27         05              BEQ     SUITE
7D0F BD         E803            JSR     PUTCH
7D12 20         F5              BRA     DEBUT
7D14 4F                 SUITE   CLRA
7D15 8E         0014            LDX     #$14
7D18 108E       000C            LDY     #$0C
7D1C C6         80      SUITE2  LDB     #$80
7D1E F7         6041            STB     CHDRAW
7D21 BD         E833            JSR     CHPLH
7D24 BD         7D39            JSR     COMPT
7D27 C6         81              LDB     #$81
7D29 F7         6041            STB     CHDRAW
7D2C BD         E833            JSR     CHPLH
7D2F BD         7D39            JSR     COMPT
7D32 4C                         INCA
7D33 81         32              CMPA    #$32
7D35 26         E5              BNE     SUITE2
7D37 20         10              BRA     FIN
7D39 34         02      COMPT   PSHS    A
7D3B 4F                         CLRA
7D3C 5F                         CLRB
7D3D C3         0001    A1      ADDD    #1
7D40 1083       80FF            CMPD    #$80FF
```

```
7D44 26   F7          BNE    A1
7D46 35   02          PULS   A
7D48 39               RTS
7D49 C6   1B    FIN   LDB    #$1B
7D4B BD   E803        JSR    PUTCH
7D4E C6   4C          LDB    #$4C
7D50 BD   E803        JSR    PUTCH
7D53 3F               SWI
7D54 0C 1B 4F 14 TABLE FCB   $0C,$1B,$4F,$14,$4
7D58 04
7D59 24 24 24 3C DEFGR0 FCB  36,36,36,60,255,24,60,60
7D5D FF 18 3C 3C
7D61 81 42 24 3C DEFGR1 FCB  129,66,36,60,60,90,189,60
7D65 3C 5A BD 3C
          0000        END

00000 Total Errors
```

b. MO5 :

```
          2070    USERAF EQU   $2070
          0002    PUTCH  EQU   2
          2036    CHDRAW EQU   $2036
          0012    CHPLH  EQU   $12

7D00                     ORG   &32000
7D00 8E   7D56           LDX   #DEFGR0
7D03 BF   2070           STX   USERAF
7D06 CE   7D51           LDU   #TABLE
7D09 E6   C0      DEBUT  LDB   ,U+
7D0B C1   04             CMPB  #4
7D0D 27   04             BEQ   SUITE
7D0F 3F   02             CALL  PUTCH
7D11 20   F6             BRA   DEBUT
7D13 4F           SUITE  CLRA
7D14 8E   0014           LDX   #$14
7D17 108E 000C           LDY   #$0C
7D1B C6   80      SUITE2 LDB   #$80
7D1D F7   2036           STB   CHDRAW
7D20 3F   12             CALL  CHPLH
7D22 BD   7D36           JSR   COMPT
7D25 C6   81             LDB   #$81
7D27 F7   2036           STB   CHDRAW
7D2A 3F   12             CALL  CHPLH
7D2C BD   7D36           JSR   COMPT
7D2F 4C                  INCA
7D30 81   32             CMPA  #$32
7D32 26   E7             BNE   SUITE2
7D34 20   10             BRA   FIN
7D36 34   02      COMPT  PSHS  A
7D38 4F                  CLRA
7D39 5F                  CLRB
7D3A C3   0001    A1     ADDD  #1
```

```
7D3D 1083 80FF            CMPD    #$80FF
7D41 26   F7              BNE     A1
7D43 35   02              PULS    A
7D45 39                   RTS
7D46 C6   1B      FIN     LDB     #$1B
7D48 3F   02              CALL    PUTCH
7D4A C6   70              LDB     #$70
7D4C 3F   02              CALL    PUTCH
7D4E BD   B000            STOP
7D51 0C 1B 73 14  TABLE   FCB     $0C,$1B,$73,$14,$4
7D55 04
7D56 24 24 24 3C  DEFGR0  FCB     36,36,36,60,255,24,60,60
7D5A FF 18 3C 3C
7D5E 81 42 24 3C  DEFGR1  FCB     129,66,36,60,60,90,189,60
7D62 3C 5A BD 3C
          0000            END

00000 Total Errors
```

3. Explications :

On charge dans "X" l'adresse de DEFGR0, pour la placer dans le registre USERAF. On n'y place que l'adresse de ce 1er DEFGR$, les autres devant le suivre. On met ensuite dans "U" l'adresse de la table des attributs et des codes interprétables, gérés par la routine PUTCH, et on exécute cette dernière : RAZ (CLS), caractères de taille double (ATTRB 1, 1), effacement du curseur (LOCATE x, y, 0).

On positionne le 1er compteur (accumulateur A) à zéro = CLRA. On charge les coordonnées, ici en hexadécimal, dans les registres X et Y (ce qui fait : 20, 12 en décimal, et non 19, 12 car les nos de colonne, en « machine », vont de 1 à 40).

On charge le code ($80) du 1er GR$(0) dans "B" et on le met dans le registre CHDRAW, pour ensuite faire appel à la routine CHPLH. Le 1er dessin est alors réalisé.

Le langage-machine étant trop rapide pour l'animation du pantin, on temporise avec un compteur (COMPT). Nous allons le revoir d'ici quelques lignes. On exécute ensuite, au même endroit, le 2e dessin (GR$(1)), ce qui a pour effet d'effacer le 1er. On temporise une deuxième fois (COMPT) et on boucle... 50 fois (= 32 en hexadécimal). A la fin, on restitue une taille normale aux caractères (ATTRB 0, 0), c'est-à-dire sur TO7(70) et TO9 : $1B suivi de $4C, et sur MO5 : $1B puis $70.

Étude du compteur :

Avec les accumulateurs A ou B nous ne pouvons compter, dans

l'absolu, que jusqu'à 255 (8 bits). Ceci est trop court comme temporisation. Pour temporiser valablement, il faut que l'ordinateur compte jusque $80FF (valeur définie arbitrairement bien sûr). Force nous est donc de demander l'accumulateur D (= ''A'' + ''B'') de 16 bits. L'instruction ADDD # 1 (ajoute 1 à ''D'') nous est précieuse car ''INCD'' n'existe pas.

Un problème se pose alors : ''A'' nous sert déjà de premier compteur (50 fois l'opération à répéter) et nous ne pouvons pas y toucher, sauf s'il est préalablement sauvegardé dans une case-mémoire et qu'on lui restitue sa valeur à la fin de l'opération ; c'est justement le rôle des piles.

Avec PSHS A, c'est-à-dire PUSH A (pousse ''A'') dans le haut de la pile S, nous sauvegardons la valeur de notre premier compteur. Lorsque le 2e compteur en a fini, nous récupérons cette valeur dans ''A'' par l'instruction PULS A, c'est-à-dire : dépile ''A''. Elle nous est restituée, intacte. S'il avait fallu sauvegarder la valeur de ''D'', on aurait pu la mettre dans deux cases-mémoire ordinaires (ou se servir de l'autre pile : U) par exemple, pour éviter d'empiler ''D'' dans la pile S, et de sortir la 2e valeur de ''A'' à la place de la 1re. Mais il y a aussi une autre façon de récupérer une donnée dans une pile, nous la verrons ensuite.

Lorsqu'une pile a fini son travail, son pointeur doit toujours revenir à sa place, surtout au retour d'un sous-programme et en particulier s'il s'agit de la pile-système S. En ce sens, l'emploi de la pile U, réservée au programmeur, eût été plus judicieux.

Dans le début de l'ouvrage nous appelons du basic vers le « machine » sans jamais rien sauvegarder ni restituer. Heureusement, nous avons jusqu'à présent pris la précaution de ne pas altérer les registres essentiels que sont le pointeur de pile S et le registre DP. Désormais tous nos programmes, pour être bien compatibles avec le basic ou fonctionner comme sous-programmes, commenceront par le stockage de tous les registres du processeur dans la pile S ; l'adresse de cette pile, ne pouvant cependant pas être préservée par celle-ci, ne sera pas sauvegardée.

Ces registres seront restitués en fin de programme, dans le sens inverse de leur entrée dans la pile S (cf pile LIFO : LAST IN, FIRST OUT, c'est-à-dire dernier entré dans la pile = premier sorti).

Un petit mot sur l'empilement :

Bien que ''S'' et ''U'' puissent servir comme les registres d'index X et Y, ils ont en général un rôle de pointeur de pile.

La pile S est la seule pile initialisée à une adresse particulière, dans la mesure où elle sert au processeur. Par exemple, après avoir (avec la cartouche-assembleur) allumé votre ordinateur, allez dans le moniteur-binaire et tapez : R et ENTRÉE. Vous verrez :

— sur TO7(70) et TO9 : "S" = $6303
— sur MO5 : "S" = $2303

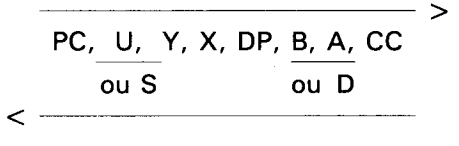
Les autres registres sont à zéro.

La pile U sera à placer avant un assemblage de programme y faisant appel.

Vous pouvez mettre les piles où vous le jugez, dans la R.A.M. . Par exemple, en faisant LDS #$7000, vous positionnez la pile S en $7000. En ce qui concerne la pile S, cela suppose que l'on prenne la précaution de lui restituer ensuite sa bonne adresse, pour que l'appareil puisse fonctionner normalement. La pile, dans le cas d'une implantation en $7000, rangera ses données à partir de $6FFF vers les adresses basses ; $6FFF sera appelée : haut de pile. La capacité d'une pile est appelée profondeur de pile (voir tableau n° 12, où la référence est le MO5).

Le pointeur d'une pile recule d'une adresse AVANT le rangement d'un octet, et il est incrémenté d'autant APRÈS la sortie de celui-ci. Lors de l'empilement d'une donnée de 16 bits, l'octet de poids faible est poussé avant l'octet de poids fort. Pour pousser les registres X et A dans la pile S, par exemple, on écrira PSHS X,A. Pour lire la valeur mise par "A" dans la pile, on pourra faire : LDA 0,S (c'est-à-dire charge "A" avec la dernière valeur entrée dans la pile). Pour lire la valeur mise par "X" dans la pile, on pourra faire : LDX 1,S (c'est-à-dire charge "X" avec l'avant-dernière valeur entrée dans la pile). Ceci est valable même lorsque l'on écrit : PSHS A,X. En effet, l'assembleur tolère n'importe quel ordre pour l'écriture des registres à entrer dans une pile, mais le processeur les prend toujours dans l'ordre que voici :

EMPILEMENT : PSHS-PSHU



DÉPILEMENT : PULS-PULU

et les fait sortir, bien sûr, dans l'ordre inverse.

Certaines instructions en assembleur, comme celles utilisant les piles, fournissent un post-octet en langage-machine, c'est-à-dire un octet après celui ou ceux de l'instruction. Ce post-octet prend une valeur à l'assemblage, variable selon les registres cités par exemple.

Exemples :

PSHS DP,PC = 34 88

PSHS X, Y, A, B = 34 36

''88'' et ''36'' (hexadécimal) sont des valeurs de post-octet pour PSHS.

Les post-octets varient donc, dans le cas des piles, selon les registres utilisés, car on peut mettre de 1 à 8 registres dans une pile, sauf le registre-pointeur de la pile dont on se sert.

Lors d'un appel de sous-programme (par JSR ou BSR*), le processeur range dans la pile-système S la valeur de retour du ''PC''. On peut donc éviter l'instruction RTS de retour de sous-programme, en incluant ''PC'' dans le dépilage de S.

Pour les registres de 16 bits, retenez que la pile charge toujours le L.S.B.* avant le M.S.B.* et dépile dans l'autre sens.

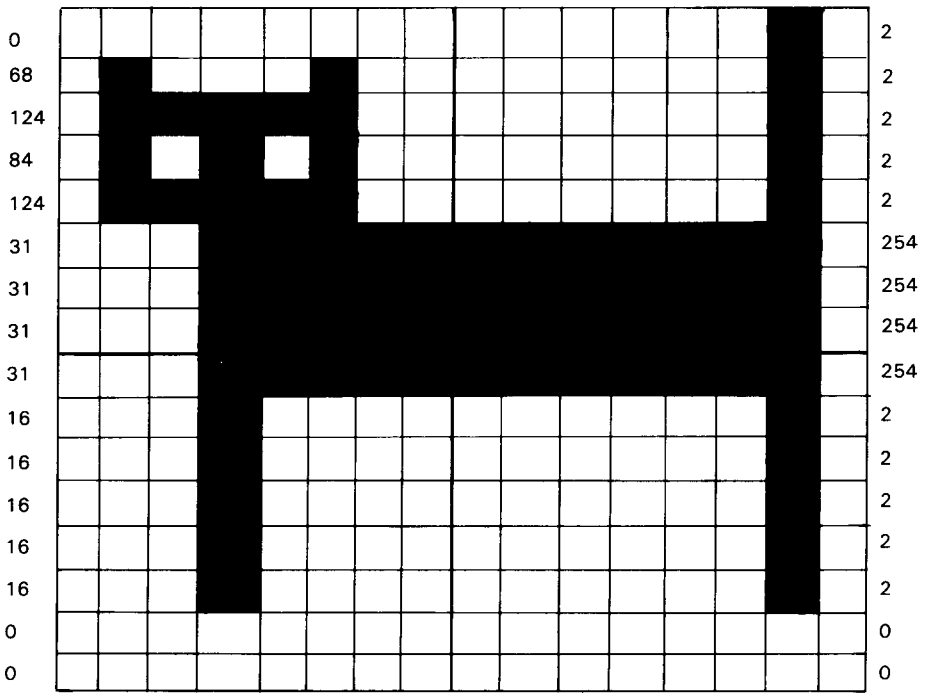
* BSR= BRANCH TO SUBROUTINE, c'est-à-dire : branche vers la routine.

* *rappel :*
— L.S.B. = LEAST SIGNIFICANT BYTE ou octet de poids le plus faible.
— M.S.B. = MOST SIGNIFICANT BYTE ou octet de poids le plus fort.

PROGRAMME N° XVI

1. Description :

Définir et réaliser 4GR$(x) côte à côte, pour aggrandir ou améliorer un graphisme. Comme exemple, nous allons dessiner un chat (le GR$ est quand même de basse résolution).



2. Réalisation :

a. TO7(70) et TO9 :

```
            602D     USERAF  EQU     $602D
            E803     PUTCH   EQU     $E803
            6041     CHDRAW  EQU     $6041
            E833     CHPLH   EQU     $E833

7D00                         ORG     &32000
7D00 34     7F               PSHS    U,Y,X,DP,D,CC
7D02 8E     7D64             LDX     #DEFGR0
7D05 BF     602D             STX     USERAF
7D08 CE     7D5F             LDU     #TABLE
7D0B E6     C0       DEBUT   LDB     ,U+
7D0D C1     04               CMPB    #4
7D0F 27     05               BEQ     SUITE
7D11 BD     E803             JSR     PUTCH
7D14 20     F5               BRA     DEBUT
7D16 8E     0014     SUITE   LDX     #20
7D19 108E   000C             LDY     #12
7D1D C6     80               LDB     #$80
7D1F F7     6041             STB     CHDRAW
7D22 BD     E833             JSR     CHPLH
7D25 8E     0016             LDX     #22
7D28 108E   000C             LDY     #12
7D2C C6     81               LDB     #$81
7D2E F7     6041             STB     CHDRAW
7D31 BD     E833             JSR     CHPLH
7D34 8E     0014             LDX     #20
7D37 108E   000E             LDY     #14
7D3B C6     82               LDB     #$82
7D3D F7     6041             STB     CHDRAW
7D40 BD     E833             JSR     CHPLH
7D43 8E     0016             LDX     #22
7D46 108E   000E             LDY     #14
7D4A C6     83               LDB     #$83
7D4C F7     6041             STB     CHDRAW
7D4F BD     E833             JSR     CHPLH
7D52 C6     1B               LDB     #$1B
7D54 BD     E803             JSR     PUTCH
7D57 C6     4C               LDB     #$4C
7D59 BD     E803             JSR     PUTCH
7D5C 35     7F               PULS    CC,A,B,DP,X,Y,U
7D5E 3F                      SWI
7D5F 0C 1B 4F 14 TABLE  FCB     $0C,$1B,$4F,$14,$4
7D63 04
7D64 1F 1F 1F 7C DEFGR0 FCB     31,31,31,124,84,124,68,0
7D68 54 7C 44 00
7D6C FE FE FE 02 DEFGR1 FCB     254,254,254,2,2,2,2,2
7D70 02 02 02 02
7D74 00 00 10 10 DEFGR2 FCB     0,0,16,16,16,16,16,31
7D78 10 10 10 1F
7D7C 00 00 02 02 DEFGR3 FCB     0,0,2,2,2,2,2,254
7D80 02 02 02 FE
              0000               END

00000 Total Errors
```

b. MO5 :

```
             2070     USERAF  EQU     $2070
             0002     PUTCH   EQU     2
             2036     CHDRAW  EQU     $2036
             0012     CHPLH   EQU     $12

7D00                          ORG     &32000
7D00 34    7F                 PSHS    U,Y,X,DP,D,CC
7D02 8E    7D5F               LDX     #DEFGR0
7D05 BF    2070               STX     USERAF
7D08 CE    7D5A               LDU     #TABLE
7D0B E6    C0       DEBUT     LDB     ,U+
7D0D C1    04                 CMPB    #4
7D0F 27    04                 BEQ     SUITE
7D11 3F    02                 CALL    PUTCH
7D13 20    F6                 BRA     DEBUT
7D15 8E    0014     SUITE     LDX     #20
7D18 108E  000C               LDY     #12
7D1C C6    80                 LDB     #$80
7D1E F7    2036               STB     CHDRAW
7D21 3F    12                 CALL    CHPLH
7D23 8E    0016               LDX     #22
7D26 108E  000C               LDY     #12
7D2A C6    81                 LDB     #$81
7D2C F7    2036               STB     CHDRAW
7D2F 3F    12                 CALL    CHPLH
7D31 8E    0014               LDX     #20
7D34 108E  000E               LDY     #14
7D38 C6    82                 LDB     #$82
7D3A F7    2036               STB     CHDRAW
7D3D 3F    12                 CALL    CHPLH
7D3F 8E    0016               LDX     #22
7D42 108E  000E               LDY     #14
7D46 C6    83                 LDB     #$83
7D48 F7    2036               STB     CHDRAW
7D4B 3F    12                 CALL    CHPLH
7D4D C6    1B                 LDB     #$1B
7D4F 3F    02                 CALL    PUTCH
7D51 C6    70                 LDB     #$70
7D53 3F    02                 CALL    PUTCH
7D55 35    03                 PULS    CC,A ,B,DP,X,Y,U
7D57 BD    B000               STOP
7D5A 0C 1B 73 14  TABLE   FCB     $0C,$1B,$73,$14,$4
7D5E 04
7D5F 1F 1F 1F 7C  DEFGR0  FCB     31,31,31,124,84,124,68,0
7D63 54 7C 44 00
7D67 FE FE FE 02  DEFGR1  FCB     254,254,254,2,2,2,2,2
7D6B 02 02 02 02
7D6F 00 00 10 10  DEFGR2  FCB     0,0,16,16,16,16,16,31
7D73 10 10 10 1F
7D77 00 00 02 02  DEFGR3  FCB     0,0,2,2,2,2,2,254
7D7B 02 02 02 FE
             0000               END

00000 Total Errors
```

3. Explications :

On sauve donc tous les registres du processeur par principe, sauf le "PC", car il pointe déjà sur l'instruction suivante*. On ne sauve pas non plus "S", car il n'est pas possible de le stocker dans la pile qu'il définit.

On charge l'adresse de début des DEFGR$, soit DEFGR0. Les étiquettes DEFGR1, DEFGR2, DEFGR3 sont inutiles car elles viennent à la suite de DEFGR0 et seront mises chacune à leur tour dans le registre USERAF. Elles ne sont là qu'en tant que points de repère.

On passe ensuite à la TABLE : effacement de la fenêtre, double taille, extinction du curseur.

On réalise enfin, un à un, les 4 GR$(x).

Ce programme pouvait aussi donner un peu de couleur au dessin par exemple, ce que vous savez faire désormais.

Nous sommes en mode-caractère (utilisateur) et on avance donc de 8 bits dans le sens des colonnes et/ou des lignes, soit d'un caractère à la fois, pour les coordonnées. La fonction ATTRB 1,1 nous oblige ici à avancer de deux fois 8 bits dans les 2 sens des coordonnées, d'où 20 et 22 en décimal pour les colonnes (ce qui fait en basic 19 et 21) et 12 et 14 pour les lignes.

Enfin, le programme se termine par la restitution des registres : PULS CC,A,B,DP,X,Y,U, puis par SWI (TO7(70) et TO9) et STOP (MO5) en assembleur.

En basic, il faut finir par PULS CC,A,B,DP,X,Y,U,PC et changer donc le code-machine "35,7F" par "35,FF". Les SWI ou STOP ne seront plus remplacés par RTS = &H39, mais par NOP = &H12 (une fois pour TO7(70) et TO9, trois fois pour MO5), pour avoir une bonne translatabilité (ou portabilité).

* L'empilement d'un registre a pour but de sauvegarder son état, afin de le lui restituer ensuite. Empiler "PC" alors qu'il contient l'adresse $7D02, pour le dépiler en fin de programme, reviendrait :

1) à vider la pile S de l'adresse $7D02, lors de PULS PC,
2) à exécuter une nouvelle fois le programme, à partir de $7D02, avec un 2e dépilement du "PC" à la fin de celui-ci.

Lors de ce 2e dépilement, "PC" prendrait alors la valeur $0000 en $6301-$6302 sur TO7(70) et TO9 ou $2301-$2302 sur MO5, la pile S étant initialisée en $6303 ou $2303. Après un retour au menu, l'appareil se « planterait ».

On ne peut en assembleur finir un corps de programme par une restitution du ''PC'' ; on pourrait cependant remplacer le RTS d'un sous-programme appelé par JSR, par PULS PC éventuellement.

Ce programme est très étalé pour bien saisir l'ensemble des opérations, mais aurait pu être mieux structuré et plus condensé.

5. Routine DRAWH

A. DESCRIPTION

Les remarques faites pour le TO9, dans le cadre de la routine CHPLH, restent valables.

Cette routine nous offre deux possibilités :

— Si le registre CHDRAW, déjà cité précédemment, contient le code ASCII d'un caractère standard ou le code d'un caractère-utilisateur défini au préalable, la routine DRAWH l'affichera le long d'un segment de droite allant du point précédent contenu dans PLOTX et PLOTY, jusqu'aux coordonnées de "X" compris entre 1 et 40 et "Y" compris entre 0 et 24. La couleur sera alors donnée par le registre COLOUR.

— Si le registre CHDRAW est mis à zéro, nous passons en mode graphique et un segment de droite sera tracé entre le point précédent (dont les coordonnées sont dans les registres PLOTX et PLOTY) et celui dont les coordonnées seront dans les registres X compris entre 0 et 319 et Y compris entre 0 et 199. La couleur sera donnée ici par le registre FORME, de −16 à +15 (TO9 et MO5), de −8 à +15 (TO7-70), ou de −8 à +7 (TO7).

Si le code de la couleur est négatif, les points du segment de droite seront écrits en « fond » et les bits correspondants à l'écran, en mémoire-caractère, automatiquement mis à zéro. Si le code est positif, les points du segment de droite seront écrits en « forme » et les bits correspondants à l'écran, en mémoire-caractère, mis automatiquement à un.

Sur TO7-70, TO9 et MO5, si le bit n°4 du registre STATUS est à 1, les bits en mémoire-caractère à l'écran seront modifiés, mais la couleur ne sera pas changée. Toute nouvelle exécution graphique restera donc invisible. Les couleurs sont alors dites « bloquées ».

A noter qu'il est possible de dessiner un contour polygonal, avec des caractères éventuellement, sans préciser à chaque fois le point de départ.

— Le code des couleurs en mode graphique est le suivant :

couleur :	*forme :*	*fond :*
noir	0	-1
rouge	1	-2
vert	2	-3
jaune	3	-4
bleu	4	-5
magenta	5	-6
cyan	6	-7
blanc	7	-8

— De plus :	*TO7-70, TO9 et MO5 :*	*TO9 et MO5 :*
couleur :	*forme :*	*fond :*
gris	8	-9
rose	9	-10
vert clair	10	-11
sable	11	-12
bleu clair	12	-13
parme	13	-14
bleu ciel	14	-15
orange	15	-16

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E80C
— Sur MO5 : code $0E

* *Paramètres d'entrée :*

— *Registres X et Y du 6809 :*
  • Pour les coordonnées.
— *Registre CHDRAW :*
  • TO7(70) et TO9 : $6041
  • MO5 : $2036
— *Registre COLOUR :*
  • TO7(70) et TO9 : $603B
  • MO5 : $202B
— *Registre FORME :*
  • TO7(70) et TO9 : $6038
  • MO5 : $2029
— *Registre STATUS :*
  • TO7(70) et TO9 : $6019
  • MO5 : $2019

— *Registre PLOTX :*
  - TO7(70) et TO9 : ($603D-$603E)
  - MO5 : ($2032-$2033)

— *Registre PLOTY :*
  - TO7(70) et TO9 : ($603F-$6040)
  - MO5 : ($2034-$2035)

* *Paramètres de sortie :*

— *Registres PLOTX et PLOTY*

B. APPLICATIONS

PROGRAMME N° XVII

1. Description :

Écrire verticalement le mot /TABLE/ à partir de la colonne 20 et de la ligne 12. L'écran sera initialisé en SCREEN 4,6,6 (par langage-machine bien sûr). Le mot /TABLE/ sera écrit en COLOR 1,0. Le code des couleurs est à mettre dans le registre COLOUR.

Pour les couleurs, RETENIR :

En mode-caractère, le code à mettre dans le registre COLOUR est le même que celui qui permet de colorer directement les octets de la R.A.M. de l'écran. Vous aurez remarqué qu'il est, par ses « mélanges », identique SUR 3 (TO7(70) et TO9) ou 4 bits (MO5) aux codes-basic positifs.

En mode graphique, le même code qu'en basic est à inscrire dans le registre FORME.

exemples sur 3 bits, pour COLOUR :

- +1 = rouge, soit :

B.V.R.
0 0 1

- +5 = magenta, soit :

B.V.R.
1 0 1

car magenta = rouge + bleu.

Remarque :

Depuis la routine CHPLH, l'accumulateur B n'est plus obligatoire pour accéder à une routine puisque les données d'entrée sont stockées dans des registres.

2. Réalisation :

a. TO7(70) et TO9 :

```
           E803     PUTCH  EQU    $E803
           603D     PLOTX  EQU    $603D
           603F     PLOTY  EQU    $603F
           603B     COLOUR EQU    $603B
           6041     CHDRAW EQU    $6041
           E80C     DRAWH  EQU    $E80C

7D00                       ORG    &32000
7D00 34    7F              PSHS   U,Y,X,DP,D,CC
7D02 CE    7D3B            LDU    #TABLE1
7D05 E6    C0      DEBUT   LDB    ,U+
7D07 C1    04              CMPB   #4
7D09 27    05              BEQ    SUITE
7D0B BD    E803            JSR    PUTCH
7D0E 20    F5              BRA    DEBUT
7D10 CE    7D46    SUITE   LDU    #TABLE2
7D13 8E    0014            LDX    #20
7D16 BF    603D            STX    PLOTX
7D19 108E  000B            LDY    #11
7D1D 10BF  603F            STY    PLOTY
7D21 31    21              LEAY   1,Y
7D23 86    C8              LDA    #$C8
7D25 B7    603B            STA    COLOUR
7D28 A6    C0      SUITE2  LDA    ,U+
7D2A 81    04              CMPA   #4
7D2C 27    0C              BEQ    FIN
7D2E B7    6041            STA    CHDRAW
7D31 BD    E80C            JSR    DRAWH
7D34 31    21              LEAY   1,Y
7D36 20    F0              BRA    SUITE2
7D38 35    7F              PULS   CC,D,DP,X,Y,U
7D3A 3F            FIN     SWI
7D3B 1B 23 20 56   TABLE1  FCB    $1B,$23,$20,$56
7D3F 1B 23 20 44           FCB    $1B,$23,$20,$44
7D43 1B 66 04              FCB    $1B,$66,$4
7D46 54 41 42 4C   TABLE2  FCC    /TABLE/
7D4A 45
7D4B 04                    FCB    $4
           0000            END

00000 Total Errors
```

b. MO5 :

```
                0002    PUTCH   EQU     2
                2032    PLOTX   EQU     $2032
                2034    PLOTY   EQU     $2034
                202B    COLOUR  EQU     $202B
                2036    CHDRAW  EQU     $2036
                000E    DRAWH   EQU     $0E

7D00                            ORG     &32000
7D00 34   7F                    PSHS    U,Y,X,DP,D,CC
7D02 CE   7D3B                  LDU     #TABLE1
7D05 E6   C0            DEBUT   LDB     ,U+
7D07 C1   04                    CMPB    #4
7D09 27   04                    BEQ     SUITE
7D0B 3F   02                    CALL    PUTCH
7D0D 20   F6                    BRA     DEBUT
7D0F CE   7D44          SUITE   LDU     #TABLE2
7D12 8E   0014                  LDX     #20
7D15 BF   2032                  STX     PLOTX
7D18 108E 000B                  LDY     #11
7D1C 10BF 2034                  STY     PLOTY
7D20 31   21                    LEAY    1,Y
7D22 86   10                    LDA     #$10
7D24 B7   202B                  STA     COLOUR
7D27 A6   C0            SUITE2  LDA     ,U+
7D29 81   04                    CMPA    #4
7D2B 27   0B                    BEQ     FIN
7D2D B7   2036                  STA     CHDRAW
7D30 3F   0E                    CALL    DRAWH
7D32 31   21                    LEAY    1,Y
7D34 20   F1                    BRA     SUITE2
7D36 35   7F                    PULS    CC,D,DP,X,Y,U
7D38 BD   B000          FIN     STOP
7D3B 1B 20 56 1B        TABLE1  FCB     $1B,$20,$56,$1B
7D3F 20 44 1B 66                FCB     $20,$44,$1B,$66
7D43 04                         FCB     $4
7D44 54 41 42 4C        TABLE2  FCC     /TABLE/
7D48 45
7D49 04                         FCB     $4
                0000            END

00000 Total Errors
```

3. Explications :

En basic, n'oubliez pas d'ajouter le ''PC'' dans le dépilage, donc de substituer ''35,FF'' à ''35,7F''. Remplacez SWI par NOP= &H12 (TO7(70) et TO9), et STOP par 3 fois NOP (MO5). Ne mettez donc plus RTS = &H39.

Vous avez dû saisir l'ensemble du programme :

On met nos coordonnées, en décimal, dans "X" (20, pour la colonne) et dans "Y" (mais ici on place la ligne – 1, soit la ligne 11 comme « ligne de départ »). On stocke ces coordonnées dans PLOTX et PLOTY qui sont les « derniers points allumés ». Ce programme avait en effet débuté avec PLOTX et PLOTY à zéro.

La routine trace une « droite » de points, ici des « points-caractère », qui s'étend de la position précédente contenue dans les registres PLOTX et PLOTY, à la position déterminée par les nouvelles coordonnées inscrites dans les registres X et Y. Il est donc nécessaire de faire LEAY 1, Y pour que la position de la lettre T soit en ligne 12, en partant de la ligne 11.

PROGRAMME N° XVIII

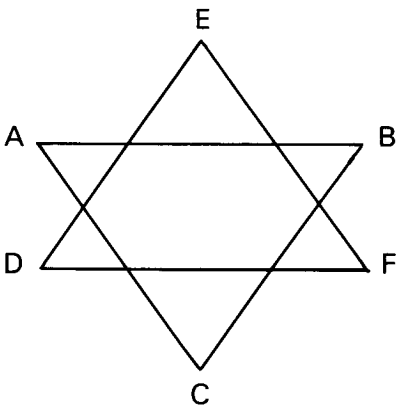
1. Description :

Graphisme avec la routine DRAWH. Il s'agira de dessiner deux triangles inversés qui se croisent, dans deux couleurs différentes, puis après une boucle de temporisation, de les effacer avec la couleur de « fond » (donc négative). L'écran sera initialisé avec une couleur de « fond » jaune.

Le triangle, pointe en bas, sera de couleur de « forme » bleue.

Le triangle, pointe en haut, sera de la couleur de « forme » rouge.

Voici leurs coordonnées en décimal :



A = 160, 96 B = 208, 96 C = 184, 144 D = 160, 136 E = 184, 88 F = 208, 136

Attention ! Sur TO7, la routine DRAWH ne sauvegarde pas le registre U. Tout appel à cette routine, en mode graphique, sera donc précédé de PSHS U = ''34,40'', et suivi de PULS U = ''35,40'', si nécessaire, c'est-à-dire si ce registre est utilisé dans le programme, comme ici.

Pour le TO7, en basic, voir le programme qui lui est réservé, ci-après.

2. Réalisation :

a. TO7(70) et TO9 :

```
                        *ATTENTION SUR TO7, FAIRE PRECEDER
                        *L'APPEL A DRAWH DE PSHS U, ET LE
                        *FAIRE SUIVRE DE PULS U.
           E803         PUTCH   EQU     $E803
           6041         CHDRAW  EQU     $6041
           6038         FORME   EQU     $6038
           603D         PLOTX   EQU     $603D
           603F         PLOTY   EQU     $603F
           E80C         DRAWH   EQU     $E80C

7D00                            ORG     &32000
7D00 34    7F                   PSHS    U,Y,X,DP,D,CC
7D02 CE    7D91                 LDU     #TABLE          ECRAN
7D05 E6    C0           D0      LDB     ,U+
7D07 C1    04                   CMPB    #4
7D09 27    05                   BEQ     SUITE
7D0B BD    E803                 JSR     PUTCH
7D0E 20    F5                   BRA     D0
7D10 4F                 SUITE   CLRA
7D11 B7    6041                 STA     CHDRAW
7D14 86    04                   LDA     #4              BLEU
7D16 B7    6038                 STA     FORME
7D19 BD    7D34                 JSR     A1              1ER TRIANGLE
7D1C 86    01                   LDA     #1              ROUGE
7D1E B7    6038                 STA     FORME
7D21 BD    7D54                 JSR     A2              2E TRIANGLE
7D24 BD    7D74                 JSR     COMPT           TEMPORISATION
7D27 86    FC                   LDA     #$FC            FOND JAUNE
7D29 B7    6038                 STA     FORME
7D2C BD    7D34                 JSR     A1              EFF.1ER TRIANG
7D2F BD    7D54                 JSR     A2              EFF.2E TRIANG
7D32 20    5A                   BRA     FIN
7D34 8E    00A0         A1      LDX     #&160
7D37 BF    603D                 STX     PLOTX
7D3A 108E  0060                 LDY     #&96
7D3E 10BF  603F                 STY     PLOTY
7D42 CE    7D96                 LDU     #TABLE1         .COORDONNEES
7D45 AE    C1           D1      LDX     ,U++            .DU 1ER
7D47 10AE  C1                   LDY     ,U++            .TRIANGLE
7D4A BD    E80C                 JSR     DRAWH           .LE DESSINER
7D4D 8C    00A0                 CMPX    #&160           FIN TABLE 1
7D50 27    3B                   BEQ     RET
7D52 20    F1                   BRA     D1
7D54 8E    00A0         A2      LDX     #&160
7D57 BF    603D                 STX     PLOTX
7D5A 108E  0088                 LDY     #&136
7D5E 10BF  603F                 STY     PLOTY
7D62 CE    7DA2                 LDU     #TABLE2         .COORDONNEES
7D65 AE    C1           D2      LDX     ,U++            .DU 2E
7D67 10AE  C1                   LDY     ,U++            .TRIANGLE
7D6A BD    E80C                 JSR     DRAWH           .LE DESSINER
```

```
7D6D 8C   00A0          CMPX   #&160        FIN TABLE 2
7D70 27   1B            BEQ    RET
7D72 20   F1            BRA    D2
7D74 5F          COMPT  CLRB                B=0
7D75 33   41            LEAU   1,U          PLAC.PILE U
7D77 36   04      D3    PSHU   B            SAUV. B
7D79 4F                 CLRA                A=0
7D7A 5F                 CLRB                B=0
7D7B C3   0001    D4    ADDD   #1           D=D+1
7D7E 1083 FFFF          CMPD   #$FFFF       .D=$FFFF?
7D82 26   F7            BNE    D4           .NON->D4
7D84 37   04            PULU   B            .OUI,SORTIR B
7D86 5C                 INCB                B=B+1
7D87 C1   0A            CMPB   #10          .B=10?
7D89 27   02            BEQ    RET          .OUI=RETOUR
7D8B 20   EA            BRA    D3           .SINON D3
7D8D 39          RET    RTS                 FIN S/S PROG.
7D8E 35   7F     FIN    PULS   CC,D,DP,X,Y,U
7D90 3F                 SWI
7D91 1B 23 20 53 TABLE  FCB    $1B,$23,$20,$53,$4
7D95 04
7D96 00D0 0060   TABLE1 FDB    208,96,184
7D9A 00B8
7D9C 0090 00A0          FDB    144,160,96
7DA0 0060
7DA2 00B8 0058   TABLE2 FDB    184,88,208
7DA6 00D0
7DA8 0088 00A0          FDB    136,160,136
7DAC 0088
          0000          END

00000 Total Errors
```

b. MO5 :

```
          0002    PUTCH  EQU    2
          2036    CHDRAW EQU    $2036
          2029    FORME  EQU    $2029
          2032    PLOTX  EQU    $2032
          2034    PLOTY  EQU    $2034
          000E    DRAWH  EQU    $0E

7D00                     ORG    &32000
7D00 34   7F            PSHS   U,Y,X,DP,D,CC
7D02 CE   7D90          LDU    #TABLE       ECRAN
7D05 E6   C0      D0    LDB    ,U+
7D07 C1   04            CMPB   #4
7D09 27   04            BEQ    SUITE
7D0B 3F   02            CALL   PUTCH
7D0D 20   F6            BRA    D0
7D0F 4F          SUITE  CLRA
7D10 B7   2036          STA    CHDRAW
```

```
7D13 86   04          LDA   #4          BLEU
7D15 B7   2029        STA   FORME
7D18 BD   7D33        JSR   A1          1ER TRIANGLE
7D1B 86   01          LDA   #1          ROUGE
7D1D B7   2029        STA   FORME
7D20 BD   7D52        JSR   A2          2E TRIANGLE
7D23 BD   7D71        JSR   COMPT       TEMPORISATION
7D26 86   FC          LDA   #$FC        FOND JAUNE
7D28 B7   2029        STA   FORME
7D2B BD   7D33        JSR   A1          EFF.1ER TRIANG
7D2E BD   7D52        JSR   A2          EFF.2E TRIANG
7D31 20   58          BRA   FIN
7D33 8E   00A0    A1  LDX   #&160
7D36 BF   2032        STX   PLOTX
7D39 108E 0060        LDY   #&96
7D3D 10BF 2034        STY   PLOTY
7D41 CE   7D94        LDU   #TABLE1     .COORDONNEES
7D44 AE   C1      D1  LDX   ,U++        .DU 1ER
7D46 10AE C1          LDY   ,U++        .TRIANGLE
7D49 3F   0E          CALL  DRAWH       .LE DESSINER
7D4B 8C   00A0        CMPX  #&160       FIN TABLE 1
7D4E 27   3A          BEQ   RET
7D50 20   F2          BRA   D1
7D52 8E   00A0    A2  LDX   #&160
7D55 BF   2032        STX   PLOTX
7D58 108E 0088        LDY   #&136
7D5C 10BF 2034        STY   PLOTY
7D60 CE   7DA0        LDU   #TABLE2      .COORDONNEES
7D63 AE   C1      D2  LDX   ,U++         .DU 2E
7D65 10AE C1          LDY   ,U++         .TRIANGLE
7D68 3F   0E          CALL  DRAWH        .LE DESSINER
7D6A 8C   00A0        CMPX  #&160        FIN TABLE 2
7D6D 27   1B          BEQ   RET
7D6F 20   F2          BRA   D2
7D71 5F   COMPT       CLRB               B=0
7D72 33   41          LEAU  1,U          PLAC.PILE U
7D74 36   04      D3  PSHU  B            SAUV.B
7D76 4F               CLRA               A=0
7D77 5F               CLRB               B=0
7D78 C3   0001    D4  ADDD  #1           D=D+1
7D7B 1083 FFFF        CMPD  #$FFFF       .D=$FFFF?
7D7F 26   F7          BNE   D4           .NON->D4
7D81 37   04          PULU  B            .OUI SORTIR B
7D83 5C               INCB               B=B+1
7D84 C1   0A          CMPB  #10          .B=10?
7D86 27   02          BEQ   RET          .OUI=RETOUR
7D88 20   EA          BRA   D3           .SINON D3
7D8A 39           RET RTS                FIN S/S PROG.
7D8B 35   7F      FIN PULS  CC,D,DP,X,Y,U
7D8D BD   B000        STOP
7D90 1B 20 53 04  TABLE FCB   $1B,$20,$53,$4
7D94 00D0 0060    TABLE1 FDB  208,96,184
7D98 00B8
7D9A 0090 00A0        FDB   144,160,96
7D9E 0060
```

```
7DA0 00B8 0058     TABLE2 FDB     184,88,208
7DA4 00D0
7DA6 0088 00A0            FDB     136,160,136
7DAA 0088
          0000            END

00000 Total Errors
```

c. Programme-basic pour TO7 :

```
10 SCREEN 2,0,0
20 FOR I= 32000 TO &H7DAD+8
30 READ A$
40 POKE I,VAL ("&H"+A$)
50 NEXT
60 LOCATE 0,0,0
70 CLS
80 EXEC 32000
90 SCREEN 0
100 '************************************
110 DATA 34,7F,CE,7D,99,E6,C0,C1,04,27,0
5,BD,E8,03,20,F5,4F,B7,60,41,86,04,B7,60
,38,BD,7D,34,86,01,B7,60,38,BD,7D,58,BD,
7D,7C,86,FC,B7,60,38,BD,7D,34,BD,7D,58,2
0,62,8E,00,A0,BF,60,3D,10,8E,00,60,10,BF
,60,3F,CE,7D,9E,AE,C1,10,AE,C1,34,40,BD,
E8,0C,35,40
120 '************************************
130 DATA 8C,00,A0,27,3F,20,ED,8E,00,A0,B
F,60,3D,10,8E,00,88,10,BF,60,3F,CE,7D,AA
,AE,C1,10,AE,C1,34,40,BD,E8,0C,35,40,8C,
00,A0,27,1B,20,ED,5F,33,41,36,04,4F,5F,C
3,00,01,10,83,FF,FF,26,F7,37,04,5C,C1,0A
,27,02,20,EA,39,35,FF,12
140 '************************************
150 DATA 1B,23,20,53,04,00,D0,00,60,00,B
8,00,90,00,A0,00,60,00,B8,00,58,00,D0,00
,88,00,A0,00,88
```

3. Explications :

• Une réflexion s'impose dès à présent, car nos programmes s'allongent. Si vous voulez pouvoir transposer sans peine votre travail sur toute « machine » équipée d'un 6809, alors définissez toujours vos étiquettes comme dans cet ouvrage, dès le début de votre programme, avec EQU. Ainsi vous pourrez rapidement translater vos programmes-assembleur, il ne restera que quelques détails à régler dans les tables (codes des attributs, des couleurs, etc.).

• La plupart des explications ont été données en marge du programme, dans la zone des commentaires. En mettant 0 en CHDRAW, on passe en mode graphique. PLOTX et PLOTY sont chargés avec les coordonnées décimales de départ.

• Le compteur : il est double et utilise la pile U en la restituant "ad integrum". Il permet de temporiser, en faisant compter le processeur (en « machine ») dix fois de 0 à $FFFF. Attention au piège de la pile U (voir ci-dessous*).

• On remarque que la couleur de « fond » jaune, pour effacer un dessin, est codée $FC, soit −4 en arithmétique signée (cf tableau n° 4).

• Les registres X et Y sont tous deux des registres de 16 bits, et l'autoincrémentation de "U" doit donc être de 2 octets à la fois (LDX, U++). Dans la table correspondante, il faut mettre à zéro les octets de poids fort, ou employer FDB. En incrémentant "U" une seule fois, le programme « se planterait ».

Bien qu'il soit loisible de faire LDX # 160, donc de ne programmer que l'octet de poids faible à charger dans "X", il convient d'être attentif lorqu'on utilise le mode indexé autoincrémenté sur 16 bits pour charger les données d'une table.

• Vous avez dû remarquer, lorsque deux lignes de couleur différente se croisent, qu'il y a un « mélange » des couleurs aux intersections (ce qui ne se produit pas sur TO9 en mode bitmap). Ceci est dû au fait qu'avec une mémoire-écran de 2*8000 octets pour 64000 points et 16 couleurs, les pixels ne peuvent être gérés que par paquets de 8. Dans un octet, ils auront la couleur « forme » OU la couleur « fond », et la modification de la couleur d'un point écrit en « forme » entraînera la modification de la couleur des autres, écrits également en « forme ». Il en va de même pour les points écrits en « fond ». Le seul moyen d'éviter cela, est de n'utiliser dans un tel cas qu'une couleur, ou de séparer un peu les dessins (difficile dans ce programme de démonstration) pour mettre les couleurs dissemblables dans des octets différents. Si vous faisiez un dessin dans la couleur de « forme », et l'autre dans la couleur de « fond », en mettant "$FE" par exemple au lieu de "$1" pour le triangle rouge, vous verriez se dessiner de grosses taches rouges. En effet, dans un octet, un seul bit serait alors en « forme » bleue, et le reste de l'octet (7bits) en « fond » rouge : on ne pourrait ajouter une troisième couleur de « fond » jaune, dans cet octet. Ce sont là des caractéristiques immuables de votre appareil.

* *Le piège de la pile U :*

Au moment d'utiliser le compteur, le registre U pointe sur l'adresse qui suit le dernier octet du programme. Ce dernier risque d'être en partie « écrasé » par la pile qui va « à reculons ». Ici, comme la pile U n'a besoin que d'un octet, on la place donc un octet plus loin : LEAU 1, U.

Dans LEAU 1,U , l'adressage est dit INDEXE A DEPLACEMENT CONSTANT sur 4 bits (1 ne varie pas, et représente l'offset d'indexation). Le déplacement constant, sur 4 bits, doit être compris entre $F0 et $0F (– 16 et + 15 en décimal). Le signe "+" ne s'écrit pas dans le cas d'un offset.

RETENEZ donc ceci : si dans un programme vous vous servez de "U" comme registre d'index, puis comme pointeur de pile, la pile U sera définie à la dernière adresse contenue dans "U". POINTEUR DE PILE U et REGISTRE D'INDEX U ne font qu'un. La pile va toujours des adresses de poids fort vers les adresses de poids faible.

6. Routine PLOTH

A. DESCRIPTION

Les divers modes d'affichage du TO9, autres que le mode 40 colonnes, sont traités en fin d'ouvrage (P9) ; ils ne sont pas concernés par ce chapitre.

Cette routine permet « l'allumage » ou « l'extinction » d'un point à l'écran. Cela correspond au PSET du basic.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E80F
— Sur MO5 : code n° $10.

* *Paramètres d'entrée :*

— *Registres X et Y :*

- Ils doivent contenir les coordonnées du point à afficher.

— *Registre CHDRAW :*

- TO7(70) et TO9 : $6041
- MO5 : $2036
- Ce registre mis à zéro indique que l'on est en mode graphique, sinon on est en mode-caractère. Mêmes remarques que pour la routine DRAWH, en particulier pour les coordonnées qui vont en mode-caractère de 1 à 40 pour "X" et de 0 à 24 pour "Y". En mode graphique, les coordonnées vont de 0 à 319 pour "X" et de 0 à 199 pour "Y".

— *Registre COLOUR :*

- TO7(70) et TO9 : $603B
- MO5 : $202B
- Ce registre sert à définir la couleur, en mode-caractère. Les codes de couleur sont ceux des octets de la R.A.M. d'écran.

— *Registre FORME :*

- TO7(70) et TO9 : $6038
- MO5 : $2029

Ce registre sert à définir la couleur, en mode graphique.

Les codes de couleur sont les mêmes que pour DRAWH.

— *Registre STATUS :*

- TO7(70) et TO9 : $6019
- MO5 : $2019

Ce registre a la même fonction que pour la routine DRAWH, à savoir que si l'on met son bit n° 4 à un, la couleur, en mode graphique, ne sera pas changée. Toute nouvelle exécution graphique sera invisible, et n'effacera pas celle qui la précède.

Il est inutile en mode-point de placer PLOTX et PLOTY au préalable.

* *Paramètres de retour :*

— *Registres PLOTX et PLOTY :*

- TO7(70) et TO9 : PLOTX = ($603D-$603E)
  et PLOTY = ($603F-$6040)
- MO5 : PLOTX = ($2032-$2033)
  et PLOTY = ($2034-$2035)

PROGRAMME N° XIX

1. Description :

Réaliser un programme classique : une nuit étoilée. Le fond de l'écran sera donc mis en noir, et des points blancs affichés en « forme ». Les étoiles étant apparemment réparties de manière désordonnée dans le ciel, il faudra disperser ces points. Diverses solutions s'offrent alors. Par exemple, réaliser une table de coordonnées des points à afficher dans le désordre, ou déterminer celles-ci, au hasard. Nous retiendrons la seconde solution.

Si le basic possède une fonction RND pseudo-aléatoire, nous n'en disposons pas en langage-machine. Il nous faudra donc imaginer une « table de RND », pour définir les coordonnées des « étoiles ».

Sans connaître les valeurs contenues dans les adresses de $0000 à $6000 sur TO7(70) et TO9, ou de $B000 à $FFFF sur MO5, nous allons nous servir de celles-ci pour déplacer les registres X et Y, représentant les coordonnées des points à afficher. C'est le registre U qui servira de registre d'index, pour les valeurs à charger. Lorsque ''X'' sera égal ou supérieur à 320, ou ''Y'' égal ou supérieur à 200, ils seront remis à zéro et affectés d'un nouveau déplacement. Il n'y aura aucun problème pour ''X'' qui peut atteindre 319, une case-mémoire ne pouvant dépasser la valeur 255. Pour ''Y'', ne pouvant dépasser 199, il sera affecté d'un autre déplacement lorsqu'une case-mémoire contiendra une valeur égale ou supérieure à 200. Le programme s'arrêtera lorsque ''U'' atteindra l'adresse $6000 + 1 ou $FFFF + 1 ; vous pourrez cependant le faire cesser plus rapidement en appuyant sur une touche quelconque.

2. Réalisation :

a. TO7(70) et TO9 :

```
                    *CORPS DE PROGRAMME
          E803      PUTCH  EQU     $E803
          6038      FORME  EQU     $6038
          6041      CHDRAW EQU     $6041
          E80F      PLOTH  EQU     $E80F

7D00                         ORG     &32000
7D00 34   7F                 PSHS    U,Y,X,DP,D,CC
```

```
7D02 8E    0000          LDX     #0
7D05 108E  0000          LDY     #0
7D09 CE    7D73          LDU     #TABLE
7D0C E6    C0      ECRAN LDB     ,U+
7D0E C1    04            CMPB    #4
7D10 27    05            BEQ     PROG
7D12 BD    E803          JSR     PUTCH
7D15 20    F5            BRA     ECRAN
7D17 86    07      PROG  LDA     #7
7D19 B7    6038          STA     FORME
7D1C 4F                  CLRA
7D1D B7    6041          STA     CHDRAW
7D20 8D    03            BSR     AFFICH
7D22 35    7F      FIN   PULS    CC,D,DP,X,Y,U
7D24 3F                  SWI

                         *SOUS-PROGRAMMES
7D25 CE    0000  AFFICH LDU     #0000
7D28 BD    E809  CLAV   JSR     $E809
7D2B 25    45           BLO     RET
7D2D E6    C0           LDB     ,U+
7D2F 1183  6001         CMPU    #$6000+1
7D33 27    3D           BEQ     RET
7D35 30    85           LEAX    B,X
7D37 8C    0140         CMPX    #320
7D3A 24    15           BHS     RAZX
7D3C E6    C0    AFFY   LDB     ,U+
7D3E 31    A5           LEAY    B,Y
7D40 108C  00C8         CMPY    #200
7D44 24    12           BHS     RAZY
7D46 BD    E80F  PSET   JSR     PLOTH
7D49 1183  6001         CMPU    #$6000+1
7D4D 27    23           BEQ     RET
7D4F 20    D7           BRA     CLAV
7D51 8E    0000  RAZX   LDX     #0
7D54 30    85           LEAX    B,X
7D56 20    E4           BRA     AFFY
7D58 108E  0000  RAZY   LDY     #0
7D5C 31    A5           LEAY    B,Y
7D5E 108C  00C8         CMPY    #200
7D62 24    02           BHS     RAZY2
7D64 20    E0           BRA     PSET
7D66 108E  0000  RAZY2  LDY     #0
7D6A 1183  6001         CMPU    #$6000+1
7D6E 27    02           BEQ     RET
7D70 20    CA           BRA     AFFY
7D72 39          RET    RTS
7D73 1B 23 20 50 TABLE  FCB     $1B,$23,$20,$50,$4
7D77 04
           0000         END

00000 Total Errors
```

b. MO5 :

```
                        * CORPS DE PROGRAMME
             0002       PUTCH   EQU     2
             2029       FORME   EQU     $2029
             2036       CHDRAW  EQU     $2036
             0010       PLOTH   EQU     $10

7D00                            ORG     &32000
7D00 34      7F                 PSHS    U,Y,X,DP,D,CC
7D02 8E      0000               LDX     #0
7D05 108E    0000               LDY     #0
7D09 CE      7D72               LDU     #TABLE
7D0C E6      C0         ECRAN   LDB     ,U+
7D0E C1      04                 CMPB    #4
7D10 27      04                 BEQ     PROG
7D12 3F      02                 CALL    PUTCH
7D14 20      F6                 BRA     ECRAN
7D16 86      07         PROG    LDA     #7
7D18 B7      2029               STA     FORME
7D1B 4F                         CLRA
7D1C B7      2036               STA     CHDRAW
7D1F 8D      05                 BSR     AFFICH
7D21 35      7F         FIN     PULS    CC,D,DP,X,Y,U
7D23 BD      B000               STOP

                        * SOUS-PROGRAMMES
7D26 CE      B000       AFFICH  LDU     #$B000
7D29 3F      0C         CLAV    CALL    $0C
7D2B 25      44                 BLO     RET
7D2D E6      C0                 LDB     ,U+
7D2F 1183    0000               CMPU    #$FFFF+1
7D33 27      3C                 BEQ     RET
7D35 30      85                 LEAX    B,X
7D37 8C      0140               CMPX    #320
7D3A 24      14                 BHS     RAZX
7D3C E6      C0         AFFY    LDB     ,U+
7D3E 31      A5                 LEAY    B,Y
7D40 108C    00C8               CMPY    #200
7D44 24      11                 BHS     RAZY
7D46 3F      10         PSET    CALL    PLOTH
7D48 1183    0000               CMPU    #$FFFF+1
7D4C 27      23                 BEQ     RET
7D4E 20      D9                 BRA     CLAV
7D50 8E      0000       RAZX    LDX     #0
7D53 30      85                 LEAX    B,X
7D55 20      E5                 BRA     AFFY
7D57 108E    0000       RAZY    LDY     #0
7D5B 31      A5                 LEAY    B,Y
7D5D 108C    00C8               CMPY    #200
7D61 24      02                 BHS     RAZY2
7D63 20      E1                 BRA     PSET
7D65 108E    0000       RAZY2   LDY     #0
7D69 1183    0000               CMPU    #$FFFF+1
```

```
7D6D 27   02              BEQ     RET
7D6F 20   CB              BRA     AFFY
7D71 39           RET     RTS
7D72 1B 20 50 04  TABLE   FCB     $1B,$20,$50,$4
          0000            END

00000 Total Errors
```

3. Explications :

Le nombre limité de possibilités pour les coordonnées ne permet que l'affichage d'un nombre restreint de points, certains d'entr'eux étant peut-être, par ailleurs, affichés plusieurs fois. Ceci est cependant largement suffisant pour ce programme.

Note sur la translatabilité :

Nous progressons dans l'étude de la translatabilité des programmes.

Souvenez-vous : tout d'abord nous avons appris l'usage des étiquettes pour que nos programmes soient « portables » d'un appareil à l'autre (6809), en ne changeant que l'adresse de celles-ci. Ces étiquettes sont aussi nécessaires pour translater ces programmes, à une autre adresse, sur un même appareil. Mais tout ceci nécessite un réassemblage !

Ici, nous voyons qu'un BSR produit à l'assemblage un code suivi d'un offset-machine, alors qu'un JSR produit un code suivi d'une adresse absolue. BSR est un branchement relatif, et le déplacement du "PC" qu'il permet est exprimé en nombre d'octets vers l'avant ou l'arrière du programme. La programmation par des branchements relatifs ne nécessite donc pas de réassemblage, lors du changement d'implantation d'un programme.

Dans le programme n° XXXIV, destiné aux TO7-70 et TO9, il nous restera à apprendre comment charger une table en mode relatif au "PC", et non plus par une adresse fixe. Ainsi nos programmes seront-ils enfin totalement translatables, même sans réassemblage !

Nous allons, après ces quelques lignes, étudier les branchements ; inutile donc pour l'instant de vous parler de BLO, etc., nous les reverrons.

Vous avez dû comprendre le programme, dont voici l'esprit :

PROG :

On met la couleur 7 (blanche) dans le registre FORME réservé au mode

graphique. Mise à zéro de CHDRAW pour passer dans ce même mode, et branchement à l'affichage des points : AFFICH.

AFFICH :

Chargement de $0000 ou $B000 (adresse de début des nombres tirés au hasard) dans "U".

CLAV :

Scrutation du clavier au cas où il faudrait faire cesser le programme avant la fin (JSR $E809 ou CALL $0C, suivis d'un branchement de retour). Chargement d'un nombre à l'adresse de "U", dans "B", qui permettra le déplacement de "X" tout d'abord (LEAX B,X), avec test de fin (CMPU #$6000 + 1 ou $FFFF + 1).

RAZX :

Si "X" $\geqslant$ 320, on le met à zéro, avant de le déplacer (et donc de l'incrémenter) par la valeur de "B".

AFFY :

Même chose que pour "X", mais le test de fin est placé après l'appel de PLOTH, car il serait inutile ici après le chargement de "B".

RAZY :

Si "Y" $\geqslant$ 200, on le met à zéro, avant de le déplacer par la valeur de "B".

RAZY2 :

Si "Y" (et donc "B") est encore supérieur ou égal à 200, on le remet à zéro. Deux éventualités peuvent alors se présenter :

1) Nous sommes en fin de programme, et nous nous branchons à l'étiquette RET.

2) Nous ne sommes pas en fin de programme, et nous allons chercher une nouvelle valeur à l'adresse suivante de "U", en AFFY.

4. Étude des branchements :

Il existe des branchements inconditionnels ne pouvant pas être refusés, et à l'opposé, des branchements pourvus d'une condition.

Parmi les branchements inconditionnels, il est des branchements relatifs et d'autres, absolus.

Un branchement est relatif lorsqu'il est effectué par un offset-machine (signé). Un branchement est absolu s'il est suivi d'une adresse fixe.

Tous les branchements conditionnels sont relatifs.

Parmi les branchements relatifs (c'est-à-dire relatifs au "PC" et commençant par la lettre B), il existe des branchements courts dont l'offset-machine est signé sur 8 bits, permettant un déplacement de - 126 à + 129 octets maximum si l'on se réfère au "PC", et des branchements longs dont l'offset-machine est signé sur 16 bits, permettant un déplacement sur 64 K-octets. Les branchements longs sont représentés par les mêmes mnémoniques que les branchements courts correspondants, mais avec la lettre L (long) comme préfixe.

Le registre d'état CC n'est pas affecté par les branchements (voir tableau n° 8).

Nous ne donnerons pas les codes-machine des branchements, ceux-ci étant répertoriés dans le tableau n° 8.

a. Branchements inconditionnels :

Nous connaissons BRA et JMP.

BRA est un branchement inconditionnel mais relatif, avec un offset-machine, alors que JMP est un branchement absolu, c'est-à-dire suivi d'une adresse fixe.

Nous connaissons encore les branchements à un sous-programme : JSR et BSR. L'un est absolu, et l'autre relatif.

Nous connaissons enfin le branchement de retour de sous-programme : RTS.

Il existe encore RTI : retour d'interruption, dont l'étude sera faite dans le chapitre sur les interruptions.

Les interruptions elles-mêmes sont des branchements, mais ce n'est pas le propos pour l'instant.

NOP permet le « saut » d'un octet et doit être classé dans les branchements inconditionnels relatifs.

BRN est un non branchement (BRANCH NEVER) inconditionnel et relatif. Il peut remplacer 2 NOP.

b. Branchements conditionnels :

Ils permettent d'effectuer des branchements selon l'état d'un ou plusieurs bits du registre "CC". Ces bits sont affectés par l'instruction qui précède le branchement (tableau n° 8). Citons parmi celles-ci une instruction de comparaison (CMP suivi du nom de registre du processeur à comparer à une valeur en mémoire), un test (TST suivi par exemple du nom de registre du processeur à tester), une instruction de chargement (LD suivi du nom de registre du processeur à charger avec une valeur en mémoire), etc..

Exemples :

LDA #$EC signifie : charge l'accumulateur A avec la valeur $EC. Cette valeur est un nombre complémenté à deux, donc négatif, égal à – 20 en décimal. L'indicateur N du registre d'état sera mis ici à 1 et un branchement ayant pour condition "N" = 1 pourra suivre ce chargement. BMI (branchement si négatif) peut ici convenir.

BLO signifie branchement si « plus bas ». Comparons la valeur contenue dans un registre avec un nombre de la zone-opérande, par exemple : CMPA #4. Dans cet exemple, si "A" est « plus bas » que 4, "C" se positionne à 1 et le branchement BLO est réalisable par l'offset-machine qui le suit. Si cet offset est F9 (– 7), par exemple, le "PC" sera décrémenté de 7 octets. BLO est spécifique au mode non signé et son homologue, en mode signé, est BLT. On différencie donc BLO (branchement si PLUS BAS) de BLT (branchement si INFÉRIEUR), par l'emploi de termes différents.

Le sens de ces instructions de branchement est parfois détourné par le programmeur, par exemple lors de l'utilisation de BLO dans tous les cas où la CARRY peut revenir à 1. C'est ce que nous avons fait dans le programme qui précède, après l'appel à la routine de décodage rapide du clavier (JSR $E809 sur TO7(70) et TO9, ou CALL $0C sur MO5). En effet, lors d'un appel à cette routine, le bit "C" de "CC" revient à 1 lorsqu'une touche est appuyée, sinon il revient à zéro. En ce sens nous avons employé BLO, mais BCS eût été plus juste comme vous le verrez d'ici quelques lignes.

- **Description des instructions de branchement conditionnel :**

1) Avec la CARRY :

- *BCC :*

Branchement si la retenue égale zéro (condition C = 0). De l'anglais : BRANCH ON CARRY CLEAR.

- *BCS :*

Branchement si la retenue égale un (condition C = 1). De l'anglais : BRANCH ON CARRY SET.

2) Avec le ZÉRO FLAG :

- *BEQ :*

Branchement si égalité (condition Z = 1). De l'anglais : BRANCH ON EQUAL.

• *BNE :*

Branchement si non égalité (condition Z = 0). De l'anglais : BRANCH IF NOT EQUAL.

3) Avec le NEGATIVE FLAG :

• *BPL :*

Branchement si positif (condition N = 0). De l'anglais : BRANCH ON PLUS.

• *BMI :*

Branchement si négatif (condition N = 1). De l'anglais : BRANCH ON MINUS.

4) Avec l'OVERFLOW FLAG :

• *BVC :*

Branchement si aucun débordement (condition V = 0). De l'anglais : BRANCH ON OVERFLOW CLEAR.

• *BVS :*

Branchement si débordement (condition V = 1). De l'anglais : BRANCH ON OVERFLOW SET.

5) Comparatives en mode non signé :

• *BHI :*

Branchement si plus haut (condition Z + C =0). De l'anglais : BRANCH IF HIGHER.

• *BLO :*

Branchement si plus bas (condition C = 1). De l'anglais : BRANCH ON LOWER.

• *BHS :*

Branchement si plus haut ou identique (condition C = 0). De l'anglais : BRANCH IF HIGHER OR SAME. Attention à l'identité possible.

• *BLS :*

Branchement si plus bas ou identique (condition C + Z = 1). De l'anglais : BRANCH ON LOWER OR SAME. Attention à l'identité possible.

6) Comparatives en mode signé :

- *BGT :*

Branchement si supérieur (condition Z + (N ⊕ V) = 0, ce qui signifie Z + (N XOR* V) = 0). Voir tableau n° 10. De l'anglais : BRANCH ON GREATER.

- *BLT :*

Branchement si inférieur (condition N ⊕ V = 1). De l'anglais : BRANCH ON LESS THAN ZERO.

- *BGE :*

Branchement si supérieur ou égal (condition N ⊕ V = 0). De l'anglais : BRANCH ON GREATER THAN OR EQUAL TO ZERO. Attention à l'égalité possible.

- *BLE :*

Branchement si inférieur ou égal (condition (N ⊕ V) + Z = 1). De l'anglais : BRANCH ON LESS THAN OR EQUAL TO ZERO. Attention à l'égalité possible.

* XOR en basic, mais l'instruction correspondante en assembleur est EOR (A ou B).

7. Routine KTSTH

A. DESCRIPTION

Cette routine, comme nous l'avons vu lors du programme précédent, permet une scrutation rapide du clavier, et donc une utilisation des programmes en mode interactif. Il suffit d'y faire un « saut », et lorsqu'une touche est appuyée au même instant, le bit C du registre CC revient avec la valeur 1, sinon il revient avec la valeur 0. Un branchement de type BCS ou BLO par exemple, permet donc de continuer le programme vers une autre étiquette.

En fait, si ceci fonctionne sur TO7(70) et TO9, comme sur MO5, la routine est plus complète sur MO5.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E809
— Sur MO5 : code n° $0C

* *Paramètres de retour :*

— *TO7(70) et TO9 :* Bit C du registre CC du 6809.
— *MO5 :* Registres CC, A et B du 6809. Si aucune touche n'a été enfoncée, le bit Z du registre CC est mis à 1, sinon il est mis à zéro. Si ''Z'' revient avec la valeur 0, alors ''B'' contient le code de la touche enfoncée, et l'état des bits 0, 1 et 2 de ''A'' indique si les touches BASIC, CNT ou JAUNE ont été enfoncées :

- bit 0 : touche BASIC
- bit 1 : touche CNT
- bit 2 : touche JAUNE.

Si le bit correspondant à la touche est à 1, la touche a été enfoncée, sinon le bit est à zéro.

Sachez que la touche ENTRÉE appuyée trop longuement au démarrage d'un programme, dans le cas où celui-ci effectue déjà une lecture rapide du clavier, peut laisser le bit C du registre CC à 1 sur TO7(70) et TO9, ou sur MO5 le bit Z du registre CC à zéro, et donc fausser la lecture faite par la routine KTSTH.

Cette routine sera encore utilisée dans le programme qui va suivre et ne nécessite pas ici de développement particulier.

8. Routine GETCH

A. DESCRIPTION

Cette routine ne scrute pas rapidement le clavier, mais décode la touche enfoncée. Elle peut fonctionner en complément de la précédente (KTSTH), c'est-à-dire que la 1re question à se poser est de savoir si une touche a été enfoncée (rôle de KTSTH) et en cas de réponse positive, de chercher en seconde intention, laquelle (rôle de GETCH).

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E806
— Sur MO5 : code n° $0A

* *Paramètres d'entrée :*

Nous allons différencier les TO7-70 et TO9, du MO5. Ces registres ne concernent pas le TO7. Sur TO9, l'usage de cette routine a été largement développé, et le périphérique « clavier » redéfini par rapport au TO7-70 : voir le chapitre P9, en fin d'ouvrage.

a. TO7-70 et TO9 :

— *Registre PTCLAV :*

• $60CD-$60CE : Registre-pointeur contenant l'adresse d'une table, où est lu le code ASCII du caractère frappé au clavier.

— *Registre BUZZ :*

• $6073 : forcé à une valeur différente de zéro, ce registre inhibe le bip sonore entendu lors de la frappe d'une touche (le clavier du TO9 n'émet pas de bip).

— *Registre LATCLV (sauf TO9) :*

• $6067 : en modifiant le contenu de ce registre, vous changerez le délai de répétition du clavier.

b. MO5 :

— *Registre CHRPTR :*

• $206D-$206E : Registre-pointeur contenant l'adresse de la table de décodage du clavier (= PTCLAV sur TO7-70 ou TO9).

— *Registre STATUS :*

• $2019 : en forçant à 1 le bit n° 3 de ce registre, vous inhiberez le bip sonore entendu lors de la frappe d'une touche.

— *Registre LATCLV :*

• $2076 : en modifiant le contenu de ce registre, vous changerez le délai de répétition du clavier.

** Paramètres de retour :*

Pour le TO9, voir aussi le chapitre P9 en fin d'ouvrage.

— *Registre CC ou Z :*

• contient en retour la « condition » d'une touche frappée (''C'' = 1 sur TO7(70) et TO9, ''Z'' = 0 sur MO5), ou non (''C'' = 0 sur TO)(70) et TO9, ''Z'' = 1 sur MO5).

— *Registre KEY (sauf TO9) :*

• contient le code de la dernière touche frappée :

• TO7(70) : $605E

• MO5 : $2037

— *Registre B :*

Différencions encore les TO7(70) et TO9, du MO5 :

• *TO7(70) et TO9 :* ''B'' retourne le code ASCII du caractère frappé (voir tableau n° 5). Il retourne 0 si aucune touche n'a été enfoncée, ou si une touche ''.'' (SHIFT) ou CNT a été enfoncée seule, ou si plusieurs touches parmi celles utilisant les ''.'' ou CNT ont été enfoncées simultanément. La touche SHIFT sélectionne le caractère se trouvant en haut d'une touche. La touche CNT force à zéro le bit 6 du code ASCII du caractère de la touche ; les minuscules sont alors forcées en majuscules.

Sur TO7-70, il est possible d'afficher certaines minuscules accentuées par deux frappes au clavier (touche ACC, suivie de la touche représentant la minuscule accentuée). Sur TO9 cette opération, de même que l'inscription d'un tréma, peut être faite en une seule frappe. Il reste néanmoins nécessaire, pour décoder ces caractères, de faire trois appels consécutifs à la routine, comme sur TO7 : le premier appel retourne dans ''B'' le code de ACC, soit $16. Le deuxième retourne dans ''B'' le code de l'accent dont il est question (ou de la cédille). Le troisième retourne enfin dans ''B'' le code de la minuscule. Voir tableau n° 15.

Sachez que la frappe simultanée de la touche "." et de la touche 0, non précédée de celle de la touche ACC, retourne la valeur zéro dans "B" (touche non reconnue) sur TO7(70), et le caractère 0 sur TO7.

Il est possible de redéfinir son clavier en changeant l'adresse contenue dans PTCLAV, sauf sur TO7, et en la remplaçant par l'adresse de sa propre table de codes ASCII, mais en séquence-accent les résultats sont signalés, par le constructeur, comme étant imprévisibles.

• *MO5 :* "B" retourne soit le code ASCII du caractère (tableau n° 5), soit le code numérique de la touche (lorsque la touche BASIC a été enfoncée : tableau n° 11). "B" retourne zéro lorsqu'aucune touche n'a été enfoncée, ou lorsque l'une des trois touches JAUNE, CNT ou BASIC a été enfoncée seule, ou lorsque plusieurs touches parmi celles qui utilisent les touches JAUNE, CNT ou BASIC ont été enfoncées simultanément.

La touche JAUNE sélectionne le caractère imprimé en jaune sur les touches. La touche CNT force à zéro le bit n° 6 du code ASCII du caractère blanc d'une touche ; les minuscules sont alors forcées en majuscules. La touche BASIC retourne le code numérique (voir tableau n° 11), et non le code ASCII de la touche enfoncée simultanément avec elle. Pour faire une minuscule accentuée, 3 appels à la routine sont nécessaires (voir tableau n° 15) :

1er appel : ACC = $16
2e appel : code de l'accent ou de la cédille
3e appel : code de la minuscule

On peut accéder à certaines minuscules accentuées, couramment utilisées dans notre langue, par 2 frappes au clavier seulement, comme sur le TO7-70 :

ACC/6 : é
ACC/7 : è
ACC/8 : ù
ACC/9 : ç
ACC/0 : à

Il reste néanmoins nécessaire de faire les 3 appels à la routine cités plus haut (séquence ACC/ code accent/ code minuscule), pour décoder ces minuscules accentuées.

La frappe simultanée des touches JAUNE et 0, non précédée de celle de la touche ACC, retourne zéro dans "B" (caractère non reconnu).

Il est possible de redéfinir son clavier en mettant, dans le registre CHRPTR, l'adresse de la table contenant ses propres codes ASCII.

La séquence-accent donne cependant, dans ce cas-là, des résultats imprévisibles, comme le signale le constructeur.

B. ORGANISATION DU CLAVIER

Les lignes qui vont suivre prennent comme exemple le TO7-70, mais le système est plus ou moins comparable sur les autres appareils.

Sous le clavier, deux séries de huit fils se coupent perpendiculairement ; chaque intersection est munie d'un interrupteur qui peut être actionné par la frappe d'une touche.

Une série de huit fils, résultant d'un démultiplexage* de trois autres fils en sortie du PORTB du 6821-système, entre dans la matrice du clavier. Les bits n^{os} 0, 1 et 2 du registre de données (PRB = $E7C9) du PORTB sont reliés aux trois fils de sortie de ce port, cités ci-dessus.

L'autre série de huit fils sort de la matrice pour entrer dans le PORTA. Ces huit fils sont ici connectés au registre de données du PORTA, appelé PRA ($E7C8).

Si aucune touche n'est appuyée, toutes les lignes du PORTA restent à 1. La valeur en $E7C8 sera donc de $FF dans ce cas.

Il existe également d'autres registres de données (PRA1, PRB1, PRA2, PRB2) pour les PIA des extensions, et des registres de contrôle pour tous. Se reporter au tableau n° 13.

C. REMARQUES CONCERNANT LES PIA

Il serait trop long de décrire dans le détail un système PIA et de le comparer à un système ACIA, comme seul en possède un le TO9 parmi les appareils étudiés dans cet ouvrage. Signalons cependant que le PIA est une interface de communication, permettant au microprocesseur de « dialoguer » avec l'extérieur par l'intermédiaire d'une série minimum de huit lignes (numérotées de 0 à 7) reliées au registre de données d'un PORT. Cette série de lignes, dans tous les cas où elle est unique, est bidirectionnelle : entrée-sortie. Le PORT com-

* Le multiplexage permet la réduction du nombre de lignes matérielles, pour une même quantité d'informations transmises. Le démultiplexage représente l'opération inverse.

prend d'autres registres qui supervisent l'ensemble des entrées-sorties.

Le PIA est un système où les données sont transmises en parallèle, c'est-à-dire que les bits d'un octet y sont envoyés tous en même temps, côte à côte. La transmission y est plus rapide que dans un système ACIA, où les 8 bits d'un octet sont transmis un à un, les uns derrière les autres. Le système PIA demande cependant plus de lignes matérielles, bien sûr, que l'ACIA.

L'ouvrage de M. BUI MINH DUC, les manuels techniques de M. OURY, le livre sur les microprocesseurs de M. GILMORE, cités en bibliographie, vous renseigneront plus amplement sur ce sujet.

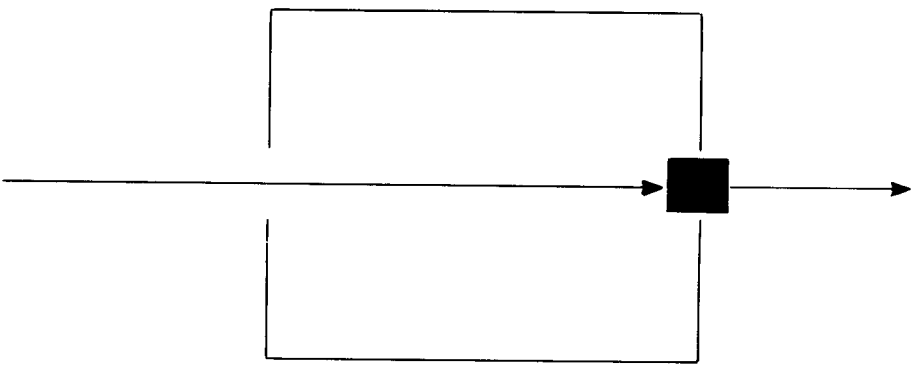
D. APPLICATION

PROGRAMME N° XX

1. Description :

Ce programme est un petit jeu, entièrement réalisé en langage-machine. Une balle (représentée par un carré jaune) traverse l'écran de gauche à droite : nous avons choisi la technique du scroll transversal pour cette réalisation. La rapidité du langage-machine nous oblige à créer une boucle de temporisation.

La balle ou carré jaune traverse, au centre de l'écran, un carré rouge plus grand, qui est la cible. Lors de son passage, la « balle » efface une partie des côtés de cette cible ; ceci est laissé tel quel volontairement et n'a que peu d'importance.



Le jeu consiste à arrêter la « balle » au centre de la cible, en appuyant sur la barre d'espacement.

Un compteur, au bas de l'écran, calcule les coups et les points. Vous n'avez que 9 coups par jeu, pour simplifier la programmation. En effet, l'ordinateur travaillant en binaire, il est nécessaire de convertir en B.C.D. les nombres supérieurs à 9 (le binaire possédant des combinaisons inutilisables pour le codage du décimal). D'autre part, l'inscription d'un chiffre se fait par un sous-programme d'affichage utilisant le code ASCII de ce chiffre. De 0 à 9, il suffit d'ajouter $30 à la valeur du chiffre pour le convertir en ASCII, mais ensuite cela se

complique car pour un nombre de 2 chiffres, il faut faire la conversion de chaque chiffre en code ASCII. Par exemple, pour 10 : $30 + 1 = chiffre des dizaines, et $30 + 0 = chiffre des unités.

Vous trouverez dans le livre de M. BUI MINH DUC des explications sur l'addition des nombres B.C.D., et une routine ou sous-programme de conversion en 2 caractères ASCII de mots de 8 bits.

Ceux qui sont intéressés par ce type d'application trouveront un programme d'affichage de compteur à 4 chiffres, à la fin de cette IV[e] partie, dans le chapitre intitulé : le décimal codé binaire.

2. Réalisation :

a. TO7(70) et TO9 :

Sur TO7-70 et TO9 les instructions PSHU et PULU, encadrant l'appel de la routine DRAWH, ne sont pas obligatoires.

```
            E803    PUTCH  EQU    $E803
            603D    PLOTX  EQU    $603D
            603F    PLOTY  EQU    $603F
            6038    FORME  EQU    $6038
            E80C    DRAWH  EQU    $E80C
            602D    USERAF EQU    $602D
            E809    KTSTH  EQU    $E809
            E806    GETCH  EQU    $E806
            E833    CHPLH  EQU    $E833

                    *CORPS DE PROGRAMME
7D00                       ORG    &32000
7D00 34   7F               PSHS   U,Y,X,DP,D,CC
7D02 86   30       DEBUT   LDA    #$30
7D04 C6   30               LDB    #$30
7D06 CE   7FFA             LDU    #$7FFA
7D09 36   06               PSHU   D
7D0B 4F                    CLRA
7D0C B7   7FFB             STA    $7FFB
7D0F 8D   09               BSR    ECRAN
7D11 8D   17               BSR    CIBLE
7D13 8D   3E               BSR    AFFICH
7D15 8D   4C               BSR    JEU
7D17 35   7F       FIN     PULS   CC,D,DP,X,Y,U
7D19 3F                    SWI

                    *SOUS-PROGRAMMES
7D1A CE   7E44     ECRAN   LDU    #TABLE1
7D1D E6   C0       EC1     LDB    ,U+
7D1F C1   04               CMPB   #4
7D21 1027 011E             LBEQ   RET
7D25 BD   E803             JSR    PUTCH
7D28 20   F3               BRA    EC1
7D2A C6   01       CIBLE   LDB    #1
```

```
7D2C F7   6038          STB    FORME
7D2F CE   7E50          LDU    #TABLE2
7D32 AE   C1            LDX    ,U++
7D34 10AE C1            LDY    ,U++
7D37 BF   603D          STX    PLOTX
7D3A 10BF 603F          STY    PLOTY
7D3E AE   C1      CIB1  LDX    ,U++
7D40 8C   0004          CMPX   #4
7D43 1027 00FC          LBEQ   RET
7D47 10AE C1            LDY    ,U++
7D4A 34   40            PSHS   U
7D4C BD   E80C          JSR    DRAWH
7D4F 35   40            PULS   U
7D51 20   EB            BRA    CIB1
7D53 CE   7E66    AFFICH LDU   #TABLE3
7D56 E6   C0      AFF1  LDB    ,U+
7D58 C1   04            CMPB   #4
7D5A 1027 00E5          LBEQ   RET
7D5E BD   E803          JSR    PUTCH
7D61 20   F3            BRA    AFF1
7D63 CE   7E85    JEU   LDU    #DEFGRO
7D66 FF   602D          STU    USERAF
7D69 CE   7E8D          LDU    #TABLE4
7D6C E6   C0      JE1   LDB    ,U+
7D6E C1   04            CMPB   #4
7D70 27   05            BEQ    SUITE1
7D72 BD   E803          JSR    PUTCH
7D75 20   F5            BRA    JE1
7D77 B6   E7C3    SUITE1 LDA   $E7C3
7D7A 8A   01            ORA    #1
7D7C B7   E7C3          STA    $E7C3
7D7F 8E   4F00    SCROLL LDX   #&20224
7D82 31   89 0140       LEAY   320,X
7D86 10BF 7FFE          STY    $7FFE
7D8A A6   00      SCR1  LDA    0,X
7D8C A7   01            STA    1,X
7D8E 4F                 CLRA
7D8F A7   00            STA    0,X
7D91 30   88 28         LEAX   40,X
7D94 BC   7FFE          CMPX   $7FFE
7D97 27   02            BEQ    SCR2
7D99 20   EF            BRA    SCR1
7D9B 30   89 FEC1 SCR2  LEAX   -319,X
7D9F 31   21            LEAY   1,Y
7DA1 10BF 7FFE          STY    $7FFE
7DA5 8D   2E            BSR    KTST1
7DA7 B6   E7C3          LDA    $E7C3
7DAA 8A   01            ORA    #1
7DAC B7   E7C3          STA    $E7C3
7DAF B6   7FFB          LDA    $7FFB
7DB2 81   39            CMPA   #$39
7DB4 1027 008B          LBEQ   RET
7DB8 8C   4F27          CMPX   #&20263
7DBB 27   02            BEQ    SCR3
7DBD 20   CB            BRA    SCR1
7DBF 30   89 FEC0 SCR3  LEAX   -320,X
```

```
7DC3 A6    00       SCR4   LDA    0,X
7DC5 A7    88 D9           STA    -39,X
7DC8 4F                    CLRA
7DC9 A7    00              STA    0,X
7DCB 30    88 28           LEAX   40,X
7DCE BC    7FFE            CMPX   $7FFE
7DD1 27    AC              BEQ    SCROLL
7DD3 20    EE              BRA    SCR4
7DD5 4F               KTST1  CLRA
7DD6 5F                    CLRB
7DD7 C3    0001     KBIS   ADDD   #1
7DDA 1083  0258            CMPD   #600
7DDE 26    F7              BNE    KBIS
7DE0 BD    E809            JSR    KTSTH
7DE3 25    01              BCS    CLAV
7DE5 39                    RTS
7DE6 BD    E806     CLAV   JSR    GETCH
7DE9 C1    20              CMPB   #$20
7DEB 27    01              BEQ    COMPT
7DED 39                    RTS
7DEE CE    7FF8     COMPT  LDU    #$7FF8
7DF1 37    06              PULU   D
7DF3 4C                    INCA
7DF4 B7    7FFB            STA    $7FFB
7DF7 8C    4F15            CMPX   #&20245
7DFA 22    0A              BHI    RATE
7DFC 8C    4F11            CMPX   #&20241
7DFF 25    05              BLO    RATE
7E01 5C                    INCB
7E02 36    06              PSHU   D
7E04 20    17              BRA    A2
7E06 36    06       RATE   PSHU   D
7E08 CE    7E94     A1     LDU    #TABLE5
7E0B E6    C0       A1BIS  LDB    ,U+
7E0D C1    04              CMPB   #4
7E0F 27    05              BEQ    A1TER
7E11 BD    E803            JSR    PUTCH
7E14 20    F5              BRA    A1BIS
7E16 1F    89       A1TER  TFR    A,B
7E18 BD    E803            JSR    PUTCH
7E1B 20    1A              BRA    TEMP
7E1D 8D    E9       A2     BSR    A1
7E1F CE    7E9A            LDU    #TABLE6
7E22 E6    C0       A2BIS  LDB    ,U+
7E24 C1    04              CMPB   #4
7E26 27    05              BEQ    A2TER
7E28 BD    E803            JSR    PUTCH
7E2B 20    F5              BRA    A2BIS
7E2D CE    7FF8     A2TER  LDU    #$7FF8
7E30 37    06              PULU   D
7E32 BD    E803            JSR    PUTCH
7E35 36    06              PSHU   D
7E37 CC    0000     TEMP   LDD    #0
7E3A C3    0001     T1     ADDD   #1
7E3D 1083  FFFF            CMPD   #$FFFF
7E41 26    F7              BNE    T1
```

```
7E43 39              RET    RTS
7E44 0C 1B 23 20     TABLE1 FCB    $0C,$1B,$23,$20,$43,$1B
7E48 43 1B
7E4A 23 20 50 1B            FCB    $23,$20,$50,$1B,$60,$4
7E4E 60 04
7E50 0088 004F       TABLE2 FDB    136,79,136,119
7E54 0088 0077
7E58 00B0 0077              FDB    176,119,176,79
7E5C 00B0 004F
7E60 0088 004F              FDB    136,79,$4
7E64 0004
7E66 1B 42 1F 55     TABLE3 FCB    $1B,$42,$1F,$55,$41
7E6A 41
7E6B 43 4F 4D 50            FCC    /COMPTEUR:00 POINTS SUR /
7E6F 54 45 55 52
7E73 3A 30 30 20
7E77 50 4F 49 4E
7E7B 54 53 20 53
7E7F 55 52 20
7E82 30 30                  FCC    /00/
7E84 04                     FCB    $4
7E85 FF FF FF FF     DEFGRO FCB    255,255,255,255,255,255
7E89 FF FF
7E8B FF FF                  FCB    255,255
7E8D 1F 4C 41 1B     TABLE4 FCB    $1F,$4C,$41,$1B,$43
7E91 43
7E92 80 04                  FCB    $80,$4
7E94 1B 42 1F 55     TABLE5 FCB    $1B,$42,$1F,$55,$59,$4
7E98 59 04
7E9A 1F 55 4B 04     TABLE6 FCB    $1F,$55,$4B,$4     ,
          0000               END

00000 Total Errors
```

b. MO5 :

```
          0002     PUTCH  EQU    2
          2032     PLOTX  EQU    $2032
          2034     PLOTY  EQU    $2034
          2029     FORME  EQU    $2029
          000E     DRAWH  EQU    $0E
          2070     USERAF EQU    $2070
          000C     KTSTH  EQU    $0C
          000A     GETCH  EQU    $0A
          0012     CHPLH  EQU    $12

                   *CORPS DE PROGRAMME
7D00                        ORG    &32000
7D00 34   7F                PSHS   U,Y,X,DP,D,CC
7D02 86   30        DEBUT   LDA    #$30
7D04 C6   30                LDB    #$30
7D06 CE   7FFA              LDU    #$7FFA
```

```
7D09 36   06          PSHU   D
7D0B 4F               CLRA
7D0C B7   7FFB        STA    $7FFB
7D0F 8D   0B          BSR    ECRAN
7D11 8D   18          BSR    CIBLE
7D13 8D   3A          BSR    AFFICH
7D15 8D   47          BSR    JEU
7D17 35   7F          PULS   CC,D,DP,X,Y,U
7D19 BD   B000        STOP

                      *SOUS-PROGRAMMES
7D1C CE   7E38  ECRAN LDU    #TABLE1
7D1F E6   C0    EC1   LDB    ,U+
7D21 C1   04          CMPB   #4
7D23 1027 0110        LBEQ   RET
7D27 3F   02          CALL   PUTCH
7D29 20   F4          BRA    EC1
7D2B C6   01    CIBLE LDB    #1
7D2D F7   2029        STB    FORME
7D30 CE   7E42        LDU    #TABLE2
7D33 AE   C1          LDX    ,U++
7D35 10AE C1          LDY    ,U++
7D38 BF   2032        STX    PLOTX
7D3B 10BF 2034        STY    PLOTY
7D3F AE   C1    CIB1  LDX    ,U++
7D41 8C   0004        CMPX   #4
7D44 1027 00EF        LBEQ   RET
7D48 10AE C1          LDY    ,U++
7D4B 3F   0E          CALL   DRAWH
7D4D 20   F0          BRA    CIB1
7D4F CE   7E58  AFFICH LDU   #TABLE3
7D52 E6   C0    AFF1  LDB    ,U+
7D54 C1   04          CMPB   #4
7D56 1027 00DD        LBEQ   RET
7D5A 3F   02          CALL   PUTCH
7D5C 20   F4          BRA    AFF1
7D5E CE   7E77  JEU   LDU    #DEFGRO
7D61 FF   2070        STU    USERAF
7D64 CE   7E7F        LDU    #TABLE4
7D67 E6   C0    JE1   LDB    ,U+
7D69 C1   04          CMPB   #4
7D6B 27   04          BEQ    SUITE1
7D6D 3F   02          CALL   PUTCH
7D6F 20   F6          BRA    JE1
7D71 B6   A7C0  SUITE1 LDA   $A7C0
7D74 8A   01          ORA    #1
7D76 B7   A7C0        STA    $A7C0
7D79 8E   0F00  SCROLL LDX   #&3840
7D7C 31   89 0140     LEAY   320,X
7D80 10BF 7FFE        STY    $7FFE
7D84 A6   00    SCR1  LDA    0,X
7D86 A7   01          STA    1,X
7D88 4F               CLRA
7D89 A7   00          STA    0,X
7D8B 30   88 28       LEAX   40,X
7D8E BC   7FFE        CMPX   $7FFE
```

```
7D91 27   02             BEQ    SCR2
7D93 20   EF             BRA    SCR1
7D95 30   89 FEC1 SCR2   LEAX   -319,X
7D99 31   21             LEAY   1,Y
7D9B 10BF 7FFE           STY    $7FFE
7D9F 8D   2E             BSR    KTST1
7DA1 B6   A7C0           LDA    $A7C0
7DA4 8A   01             ORA    #1
7DA6 B7   A7C0           STA    $A7C0
7DA9 B6   7FFB           LDA    $7FFB
7DAC 81   39             CMPA   #$39
7DAE 1027 0085           LBEQ   RET
7DB2 8C   0F27           CMPX   #&3879
7DB5 27   02             BEQ    SCR3
7DB7 20   CB             BRA    SCR1
7DB9 30   89 FEC0 SCR3   LEAX   -320,X
7DBD A6   00      SCR4   LDA    0,X
7DBF A7   88 D9          STA    -39,X
7DC2 4F                  CLRA
7DC3 A7   00             STA    0,X
7DC5 30   88 28          LEAX   40,X
7DC8 BC   7FFE           CMPX   $7FFE
7DCB 27   AC             BEQ    SCROLL
7DCD 20   EE             BRA    SCR4
7DCF 4F           KTST1  CLRA
7DD0 5F                  CLRB
7DD1 C3   0001    KBIS   ADDD   #1
7DD4 1083 0258           CMPD   #600
7DD8 26   F7             BNE    KBIS
7DDA 3F   0C             CALL   KTSTH
7DDC 26   01             BNE    CLAV
7DDE 39                  RTS
7DDF 3F   0A      CLAV   CALL   GETCH
7DE1 C1   20             CMPB   #$20
7DE3 27   01             BEQ    COMPT
7DE5 39                  RTS
7DE6 CE   7FF8    COMPT  LDU    #$7FF8
7DE9 37   06             PULU   D
7DEB 4C                  INCA
7DEC B7   7FFB           STA    $7FFB
7DEF 8C   0F15           CMPX   #&3861
7DF2 22   0A             BHI    RATE
7DF4 8C   0F11           CMPX   #&3857
7DF7 25   05             BLO    RATE
7DF9 5C                  INCB
7DFA 36   06             PSHU   D
7DFC 20   15             BRA    A2
7DFE 36   06      RATE   PSHU   D
7E00 CE   7E86    A1     LDU    #TABLE5
7E03 E6   C0      A1BIS  LDB    ,U+
7E05 C1   04             CMPB   #4
7E07 27   04             BEQ    A1TER
7E09 3F   02             CALL   PUTCH
7E0B 20   F6             BRA    A1BIS
7E0D 1F   89      A1TER  TFR    A,B
7E0F 3F   02             CALL   PUTCH
```

```
7E11 20   18              BRA    TEMP
7E13 8D   EB      A2      BSR    A1
7E15 CE   7E8C            LDU    #TABLE6
7E18 E6   C0      A2BIS   LDB    ,U+
7E1A C1   04              CMPB   #4
7E1C 27   04              BEQ    A2TER
7E1E 3F   02              CALL   PUTCH
7E20 20   F6              BRA    A2BIS
7E22 CE   7FF8    A2TER   LDU    #$7FF8
7E25 37   06              PULU   D
7E27 3F   02              CALL   PUTCH
7E29 36   06              PSHU   D
7E2B CC   0000    TEMP    LDD    #0
7E2E C3   0001    T1      ADDD   #1
7E31 1083 FFFF            CMPD   #$FFFF
7E35 26   F7              BNE    T1
7E37 39           RET     RTS
7E38 0C 1B 20 43  TABLE1  FCB    $0C,$1B,$20,$43,$1B,$20
7E3C 1B 20
7E3E 50 1B 60 04          FCB    $50,$1B,$60,$4
7E42 0088 004F    TABLE2  FDB    136,79,136,119
7E46 0088 0077
7E4A 00B0 0077            FDB    176,119,176,79
7E4E 00B0 004F
7E52 0088 004F            FDB    136,79,$4
7E56 0004
7E58 1B 42 1F 55  TABLE3  FCB    $1B,$42,$1F,$55,$41
7E5C 41
7E5D 43 4F 4D 50          FCC    /COMPTEUR:00 POINTS SUR /
7E61 54 45 55 52
7E65 3A 30 30 20
7E69 50 4F 49 4E
7E6D 54 53 20 53
7E71 55 52 20
7E74 30 30                FCC    /00/
7E76 04                   FCB    $4
7E77 FF FF FF FF  DEFGRO  FCB    255,255,255,255,255,255
7E7B FF FF
7E7D FF FF                FCB    255,255
7E7F 1F 4C 41 1B  TABLE4  FCB    $1F,$4C,$41,$1B,$43
7E83 43
7E84 80 04                FCB    $80,$4
7E86 1B 42 1F 55  TABLE5  FCB    $1B,$42,$1F,$55,$59,$4
7E8A 59 04
7E8C 1F 55 4B 04  TABLE6  FCB    $1F,$55,$4B,$4
          0000            END

00000 Total Errors
```

3. Explications :

Au début du programme, nous plaçons la valeur $30 (= 0 en code ASCII) dans "A et dans "B". "A" représentera le compteur de coups,

et ''B'' le compteur de points. Ces valeurs seront sauvegardées par l'empilement de l'accumulateur D dans la pile U. Il est nécessaire de placer auparavant cette pile, car le système ne le fait pas. Nous avons pour cela choisi l'adresse $7FFA puisque nous n'avons que 2 octets (''D'') à sauvegarder pour tout le programme. Ces octets sont stockés en $7FF9 pour ''B'', et en $7FF8 pour ''A''. Le pointeur de pile se trouve alors en $7FF8 après l'instruction PSHU D. Nous plaçons ensuite zéro en $7FFB. Cette case-mémoire nous sera utile pour détecter la fin du jeu ($39 = 9 coups), sans avoir à dépiler et rempiler trop souvent (en SCR2). A noter que $7FFA est une case éventuellement utilisable car la pile U ne s'en sert pas.

L'utilisation de cases-mémoire, pour la sauvegarde de données, n'est pas très justifiée en assembleur puisque les piles ont normalement cette fonction. Il apparaît cependant que cette méthode, bien qu'elle emploie parfois la R.A.M. de manière mal organisée, est plus facile à manier lorsqu'on ne maîtrise pas encore bien la technique du langage-assembleur.

ÉCRAN : Initialisation des couleurs d'écran. LBEQ doit être utilisé car la longueur du programme, pour atteindre l'étiquette RET, dépasse le nombre d'octets autorisés pour un offset-machine (exprimé en hexadécimal dans le code-objet) de branchement court.

CIBLE : On place la couleur rouge (= 1) dans le registre de couleur graphique FORME. On dessine la cible (CIB1). L'appel à la routine DRAWH doit être précédé d'une sauvegarde du registre U sur T07.

AFFICH : Il s'agit ici d'afficher le compteur.

JEU : Il faut tout d'abord dessiner un carré jaune à gauche de l'écran (DEFGR0), puis réaliser le scroll transversal (SCROLL) après s'être mis en mémoire-caractère (SUITE1) pour translater les octets « écrits » en « forme ».

Détail du scroll :

On charge dans ''X'' l'adresse du 1er octet (&20224 sur TO7(70) et TO9 ou &3840 sur MO5) du carré jaune et on le « recopie » en 1,X , donc à côté. On efface l'ancien emplacement de l'octet déplacé en mettant 0 en 0,X. On prend l'octet suivant, en-dessous (40,X), et ainsi de suite jusqu'à la valeur maximum 320 (= 8*40 adresses, soit une matrice de 8 bits sur 8, donc un carré) représentée par ''Y'' et placée en $7FFE. On prend ensuite (SCR2) l'octet suivant le tout 1er (&20225 sur TO7(70) et TO9 ou &3841 sur MO5), et on recommence, non sans s'assurer que l'utilisateur n'a pas joué (BSR KTST1) et que le jeu n'est pas fini (case-mémoire $7FFB déjà vue). Il faut aussi

s'assurer que l'on n'est pas au bout de l'écran (&20263 sur TO7(70) et TO9 ou &3879 sur MO5), car dans ce cas il est nécessaire de se brancher en SCR3 pour remettre le carré jaune en 1re colonne, et continuer sans cesse.

KTST1 : A chaque fois que l'on va en KTST1 voir si une touche est appuyée, on temporise en faisant compter le processeur jusque 600 (KBIS).

CLAV : Nous ne décodons le clavier (GETCH) que lorsqu'une touche est appuyée (KTSTH). Dans l'affirmative, cette touche est comparée à $20, code ASCII de la barre d'espacement. S'il s'agit d'une autre touche, on continue le scroll (RTS).

COMPT : On incrémente ''A'', compteur de coups, et on vérifie que les coordonnées de la « balle » arrêtée sont bien comprises dans la fourchette des coordonnées (colonnes) de la cible. Le cas échéant, on incrémente ''B'', compteur de points. On sauvegarde ''A'' et ''B'' dans la pile U, dont le pointeur sera remis en place à la fin du programme comme cela est fait régulièrement maintenant.

RATE : Si le coup est raté, on inscrit un coup (A1), mais pas de point (A2).

Lors de l'inscription des coups, on temporise encore un peu (TEMP) pour bien voir l'endroit où la « balle » a été arrêtée.

TABLE 4 : écriture de GR0 (carré jaune).

TABLES 5 et 6 : positionnement des couleurs, et du curseur pour les compteurs.

Note :

Pour se servir d'une pile, il faut la dessiner et connaître régulièrement la position exacte de son pointeur : par exemple, à COMPT, il était nécessaire de le remettre en 7FF8H, adresse de la donnée entrée par ''A''.

9. Routine JOYSH

A. DESCRIPTION

Comme son nom l'indique, cette routine gère les manettes de jeu ou JOYSTICKS.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E827
— Sur MO5 : code n° $1C.

* *Paramètre d'entrée :*

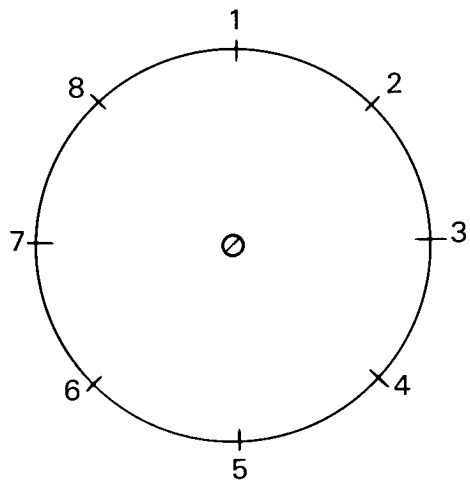
— *Registre A du 6809 :*

Ce registre doit contenir 0 ou 1, selon le numéro de manette utilisée.

* *Paramètres de retour :*

— *Registre B du 6809 :*

Ce registre contient, en retour, le code de la position de la manette (0 à 8) où 0 est le code du centre de la manette, et les autres chiffres, les positions signalées par le schéma ci-dessous :



— *Registre CC du 6809 :*

La retenue C du registre d'état CC sera à 1 si la gâchette (ou bouton « action ») a été enfoncée, et à zéro dans le cas contraire. Il n'y a pas d'anti-rebond.

B. APPLICATION

PROGRAMME N° XXI

1. Description :

Nous allons, avec une manette de jeu, faire de la musique. Lorsque cette manette sera à la position 1, le programme fera exécuter un DO GRAVE ; à 2, un RE ; à 3, un MI, etc., jusque 8 pour un DO AIGU.

La position 0 sera celle du dièse, sauf lorsque celui-ci n'existe pas pour la note en question (MI et SI).

La gâchette servira à arrêter le programme.

2. Réalisation :

a. TO7(70) et TO9 :

```
                6037    OCTAVE  EQU     $6037
                E827    JOYSH   EQU     $E827
                E81E    NOTEH   EQU     $E81E

7D00                            ORG     &32000
7D00 34         7F              PSHS    U,Y,X,DP,D,CC
7D02 5F                         CLRB
7D03 F7         7FFE            STB     $7FFE
7D06 86         04              LDA     #4
7D08 B7         6037            STA     OCTAVE
7D0B CC         0000    DEBUT   LDD     #0
7D0E C3         0001    D1      ADDD    #1
7D11 1083       44FF            CMPD    #$44FF
7D15 26         F7              BNE     D1
7D17 5F                         CLRB
7D18 4F                         CLRA
7D19 BD         E827            JSR     JOYSH
7D1C 25         67              BCS     FIN
7D1E C1         00      DIESE   CMPB    #0
7D20 27         4E              BEQ     PLAY2
7D22 86         04              LDA     #4
7D24 B7         6037            STA     OCTAVE
7D27 C1         01      DO      CMPB    #1
7D29 26         04              BNE     RE
7D2B C6         31              LDB     #$31
7D2D 20         39              BRA     PLAY
7D2F C1         02      RE      CMPB    #2
7D31 26         04              BNE     MI
7D33 C6         33              LDB     #$33
7D35 20         31              BRA     PLAY
7D37 C1         03      MI      CMPB    #3
7D39 26         04              BNE     FA
7D3B C6         35              LDB     #$35
```

```
7D3D 20   29          BRA   PLAY
7D3F C1   04    FA    CMPB  #4
7D41 26   04          BNE   SO
7D43 C6   36          LDB   #$36
7D45 20   21          BRA   PLAY
7D47 C1   05    SO    CMPB  #5
7D49 26   04          BNE   LA
7D4B C6   38          LDB   #$38
7D4D 20   19          BRA   PLAY
7D4F C1   06    LA    CMPB  #6
7D51 26   04          BNE   SI
7D53 C6   3A          LDB   #$3A
7D55 20   11          BRA   PLAY
7D57 C1   07    SI    CMPB  #7
7D59 26   04          BNE   UT
7D5B C6   3C          LDB   #$3C
7D5D 20   09          BRA   PLAY
7D5F C6   02    UT    LDB   #2
7D61 F7   6037        STB   OCTAVE
7D64 C6   31          LDB   #$31
7D66 20   00          BRA   PLAY
7D68 F7   7FFE  PLAY  STB   $7FFE
7D6B BD   E81E  P1    JSR   NOTEH
7D6E 20   9B          BRA   DEBUT
7D70 F6   7FFE  PLAY2 LDB   $7FFE
7D73 C1   00          CMPB  #0
7D75 27   94          BEQ   DEBUT
7D77 C1   35          CMPB  #$35
7D79 27   F0          BEQ   P1
7D7B C1   3C          CMPB  #$3C
7D7D 27   EC          BEQ   P1
7D7F 5C               INCB
7D80 BD   E81E        JSR   NOTEH
7D83 20   86          BRA   DEBUT
7D85 35   7F    FIN   PULS  CC,D,DP,X,Y,U
7D87 3F               SWI
          0000        END

00000 Total Errors
```

b. MO5 :

```
          203F  OCTAVE EQU   $203F
          001C  JOYSH  EQU   $1C
          001E  NOTEH  EQU   $1E

7D00                   ORG   &32000
7D00 34   7F           PSHS  U,Y,X,DP,D,CC
7D02 5F                CLRB
7D03 F7   7FFE         STB   $7FFE
7D06 86   04           LDA   #4
7D08 B7   203F         STA   OCTAVE
7D0B CC   0000  DEBUT  LDD   #0
```

```
7D0E C3   0001     D1      ADDD   #1
7D11 1083 44FF             CMPD   #$44FF
7D15 26   F7               BNE    D1
7D17 5F                    CLRB
7D18 4F                    CLRA
7D19 3F   1C               CALL   JOYSH
7D1B 25   62               BCS    FIN
7D1D C1   00       DIESE   CMPB   #0
7D1F 27   4A               BEQ    PLAY2
7D21 86   04               LDA    #4
7D23 B7   203F             STA    OCTAVE
7D26 C1   01       DO      CMPB   #1
7D28 26   02               BNE    RE
7D2A 20   38               BRA    PLAY
7D2C C1   02       RE      CMPB   #2
7D2E 26   04               BNE    MI
7D30 C6   03               LDB    #3
7D32 20   30               BRA    PLAY
7D34 C1   03       MI      CMPB   #3
7D36 26   04               BNE    FA
7D38 C6   05               LDB    #5
7D3A 20   28               BRA    PLAY
7D3C C1   04       FA      CMPB   #4
7D3E 26   04               BNE    SO
7D40 C6   06               LDB    #6
7D42 20   20               BRA    PLAY
7D44 C1   05       SO      CMPB   #5
7D46 26   04               BNE    LA
7D48 C6   08               LDB    #8
7D4A 20   18               BRA    PLAY
7D4C C1   06       LA      CMPB   #6
7D4E 26   04               BNE    SI
7D50 C6   0A               LDB    #$0A
7D52 20   10               BRA    PLAY
7D54 C1   07       SI      CMPB   #7
7D56 26   04               BNE    UT
7D58 C6   0C               LDB    #$0C
7D5A 20   08               BRA    PLAY
7D5C C6   02       UT      LDB    #2
7D5E F7   203F             STB    OCTAVE
7D61 5A                    DECB
7D62 20   00               BRA    PLAY
7D64 F7   7FFE     PLAY    STB    $7FFE
7D67 3F   1E       P1      CALL   NOTEH
7D69 20   A0               BRA    DEBUT
7D6B F6   7FFE     PLAY2   LDB    $7FFE
7D6E C1   00               CMPB   #0
7D70 27   99               BEQ    DEBUT

7D72 C1   05               CMPB   #5
7D74 27   F1               BEQ    P1
7D76 C1   0C               CMPB   #$0C
7D78 27   ED               BEQ    P1
7D7A 5C                    INCB
7D7B 3F   1E               CALL   NOTEH
7D7D 20   8C               BRA    DEBUT
```

```
7D7F 35   7F       FIN     PULS   CC,D,DP,X,Y,U
7D81 BD   B000             STOP
          0000             END

00000 Total Errors
```

3. Explications :

On place 0 en $7FFE, case-mémoire qui nous sera utile ultérieurement. On introduit ensuite la valeur 4 en $6037 sur TO7(70) et TO9, ou en $203F sur MO5 (poids faible de l'adresse) : c'est le choix de l'octave ; il s'agit ici d'une octave moyenne. On place encore un compteur (''D'', de 0 à $44FF), en début de programme. On « nettoie » ''B'', puis ''A'', et on appelle la routine JOYSH de lecture des manettes de jeu. Si ''C'' = 1 (gâchette appuyée ou manettes non branchées), on va directement à la fin du programme. On cherche à savoir si ''B'' = 0 (dièse) pour se brancher à PLAY2, sinon on cherche la note correspondant à la position de la manette ; on charge le code de cette note, et on appelle la routine NOTEH. On sauvegarde ensuite dans $7FFE la dernière note jouée (PLAY) pour, le cas échéant, incrémenter son code et faire un dièse. Au démarrage, comme $7FFE est vide, aucune note n'est jouée tant que la manette n'est pas déplacée.

10. Routines GETLH et LPINH

Les possibilités supplémentaires pour ces routines, offertes par le TO9, sont répertoriées dans le chapitre P9 en fin d'ouvrage.

A. DESCRIPTION

Ces routines servent à l'utilisation du crayon optique (appelé aussi photostyle ou encore LIGHTPEN). Le crayon optique est muni d'un phototransistor qui émet un petit « courant » vers l'ordinateur lorsqu'une certaine intensité lumineuse lui parvient. Avec certains écrans vous aurez du mal à obtenir une luminosité suffisante (postes de télévision mal réglées ou à écran trop large), de même le rouge et le noir ont un spectre qui ne fait pas réagir le phototransistor. Dans ces cas de mauvaise luminosité, le bit C du registre CC sera mis à 1 au retour de la routine de lecture du crayon optique GETLH, correspondant en basic à INPEN.

Le bouton-interrupteur est à différencier du phototransistor ; il ne s'agit que d'un contacteur. La routine gérant le contacteur est LPINH.

En basic, INPUTPEN ne permet la lecture d'un point à l'écran que lorsque le contact est établi (contacteur fermé, et non ouvert comme on le dit trop souvent par erreur). INPUTPEN utilise donc les deux routines : LPINH, puis GETLH.

Un conseil : comme en basic vous pouvez régler le phototransistor au 1er menu, du moins sur TO7(70) et TO9, avant de programmer avec la cartouche-assembleur. Ceci est toujours une sage précaution.

1. Lecture des coordonnées d'écran avec le crayon optique : GETLH.

Pour le TO9, nous n'étudierons ici que le mode 40 colonnes.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E818
— Sur MO5 : code n° $18

* *Paramètres de retour :*

— Registres X, Y, CC du 6809 :

L'abscisse du point visé par le crayon optique est retournée dans X (0,319) et l'ordonnée dans Y (0,199).

Le registre CC voit son bit de retenue C forcé à 1 en cas de mauvaise lecture (y compris lorsque le crayon est trop éloigné de l'écran !), sinon le bit C est forcé à zéro dans le cas d'une lecture correcte.

2. Le contacteur : LPINH.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E81B
— Sur MO5 : code n° $16

* *Paramètres de retour :*

— Registre CC du 6809 :

Ce paramètre de retour est unique sur TO7(70) et TO9, comme sur MO5. Si le bit C du registre d'état revient à 1, c'est que le bouton a été enfoncé (contacteur fermé). Dans le cas contraire (bit C revenant à zéro), le contacteur est resté ouvert. Sur TO9, un anti-rebond de 10 millisecondes est effectué et le bit Z du registre d'état toujours forcé à zéro.

B. APPLICATION

PROGRAMME N° XXII

1. Description :

Nous allons dessiner un carré blanc, à partir de l'adresse &20243 sur TO7(70) et TO9, ou &3859 sur MO5. Celui-ci sera affiché en mémoire-couleur uniquement, en « pokant » le « fond » de 8 octets verticalement consécutifs.

Nous testerons ensuite :

a) le contacteur du crayon optique,
b) le phototransistor.

Si les coordonnées pointées par le photostyle correspondent à celles du carré blanc, alors s'affichera, en-dessous, le message : TEST CONCLUANT.

2. Réalisation :

a. TO7(70) et TO9 :

```
            E81B    LPINH   EQU     $E81B
            E818    GETLH   EQU     $E818
            E803    PUTCH   EQU     $E803

7D00                        ORG     &32000
7D00 34     7F              PSHS    U,Y,X,DP,D,CC
7D02 CE     4F13            LDU     #&20243
7D05 8D     3B              BSR     COLOR
7D07 E7     40      DEBUT   STB     0,U
7D09 33     C8 28           LEAU    40,U
7D0C 1183   5053            CMPU    #&20243+(40*8)
7D10 27     02              BEQ     TEST
7D12 20     F3              BRA     DEBUT
7D14 BD     E81B    TEST    JSR     LPINH
7D17 24     FB              BCC     TEST
7D19 BD     E818    LECT    JSR     GETLH
7D1C 25     F6              BCS     TEST
7D1E 8C     0098            CMPX    #152
7D21 25     F1              BLO     TEST
7D23 8C     009F            CMPX    #159
7D26 22     EC              BHI     TEST
7D28 108C   0060            CMPY    #96
7D2C 25     E6              BLO     TEST
7D2E 108C   0067            CMPY    #103
7D32 22     E0              BHI     TEST
7D34 CE     7D50            LDU     #TABLE
```

```
7D37 E6    C0        ECRIT  LDB     ,U+
7D39 C1    04               CMPB    #4
7D3B 27    10               BEQ     FIN
7D3D BD    E803             JSR     PUTCH
7D40 20    F5               BRA     ECRIT
7D42 B6    E7C3      COLOR  LDA     $E7C3
7D45 84    FE               ANDA    #254
7D47 B7    E7C3             STA     $E7C3
7D4A C6    FF               LDB     #$FF
7D4C 39                     RTS
7D4D 35    7F        FIN    PULS    CC,D,DP,X,Y,U
7D4F 3F                     SWI
7D50 1B 40 1B 57    TABLE  FCB     $1B,$40,$1B,$57,$1B,$60
7D54 1B 60
7D56 1F 54 4A               FCB     $1F,$54,$4A
7D59 54 45 53 54            FCC     /TEST    CONCLUANT/
7D5D 20 20 20 43
7D61 4F 4E 43 4C
7D65 55 41 4E 54
7D69 04                     FCB     $4
           0000             END

00000 Total Errors
```

b. MO5 :

```
           0016     LPINH  EQU     $16
           0018     GETLH  EQU     $18
           0002     PUTCH  EQU     2

7D00                       ORG     &32000
7D00 34    7F              PSHS    U,Y,X,DP,D,CC
7D02 CE    0F13            LDU     #&3859
7D05 8D    38              BSR     COLOR
7D07 E7    40        DEBUT STB     0,U
7D09 33    C8 28           LEAU    40,U
7D0C 1183 1053             CMPU    #&3859+(40*8)
7D10 27    02              BEQ     TEST
7D12 20    F3              BRA     DEBUT
7D14 3F    16        TEST   CALL    LPINH
7D16 24    FC              BCC     TEST
7D18 3F    18        LECT   CALL    GETLH
7D1A 25    F8              BCS     TEST
7D1C 8C    0098            CMPX    #152
7D1F 25    F3              BLO     TEST
7D21 8C    009F            CMPX    #159
7D24 22    EE              BHI     TEST
7D26 108C 0060             CMPY    #96
7D2A 25    E8              BLO     TEST
7D2C 108C 0067             CMPY    #103
7D30 22    E2              BHI     TEST
7D32 CE    7D4F            LDU     #TABLE
7D35 E6    C0        ECRIT  LDB     ,U+
```

```
7D37 C1    04              CMPB    #4
7D39 27    0F              BEQ     FIN
7D3B 3F    02              CALL    PUTCH
7D3D 20    F6              BRA     ECRIT
7D3F B6    A7C0    COLOR   LDA     $A7C0
7D42 84    FE              ANDA    #254
7D44 B7    A7C0            STA     $A7C0
7D47 C6    77              LDB     #$77
7D49 39                    RTS
7D4A 35    7F      FIN     PULS    CC,D,DP,X,Y,U
7D4C BD    B000            STOP
7D4F 1B 40 1B 57   TABLE   FCB     $1B,$40,$1B,$57,$1B,$60
7D53 1B 60
7D55 1F 54 4A              FCB     $1F,$54,$4A
7D58 54 45 53 54           FCC     /TEST    CONCLUANT/
7D5C 20 20 20 43
7D60 4F 4E 43 4C
7D64 55 41 4E 54
7D68 04                    FCB     $4
          0000             END

00000 Total Errors
```

3. Explications :

Si le programme ne fonctionne pas correctement, augmentez la luminosité de votre moniteur-vidéo, et passez lentement et transversalement devant le carré blanc avec le crayon optique.

Le carré est dessiné en début de programme, directement dans les octets. TEST correspond au test du contacteur du crayon optique. LECT représente la lecture des coordonnées par le phototransistor. Si TEST ne fonctionne pas, "C" = 0 et si LECT ne fonctionne pas, "C" = 1 ; dans ces 2 cas, on retourne à l'étiquette TEST.

ECRIT : On écrit en COLOR 0,7. Le tour est mis en couleur noire. On positionne le curseur en colonne 11, ligne 20, et on inscrit : TEST CONCLUANT.

11. Routines GETSH et GETPH

Nous ne traiterons ici que du mode 40 colonnes. Pour le TO9, voir aussi le paragraphe P9.

A. DESCRIPTION

Ces routines permettent les lectures respectives d'un caractère et de la couleur d'un point (FOND ou FORME). Le code ASCII du caractère, s'il est reconnu par le moniteur-système, sera retourné dans "B".

L'abscisse et l'ordonnée du caractère à lire sont à passer par "X" et "A".

L'abscisse et l'ordonnée de la couleur du point à reconnaître sont à passer par "X" et "Y". Le code de la couleur est retourné par "B" (– 8 à + 7 sur TO7, et – 16 à + 15 sur TO7-70, TO9 et MO5).

1. Routine GETSH :

Il s'agit de la routine qui permet la lecture d'un caractère. Elle ne fonctionne qu'avec l'alphabet standard, sauf sur TO9 où elle gère aussi tous les caractères du G2 (voir P9).

* *Point d'entrée :*

— Dur TO7(70) et TO9 : $E824
— Sur MO5 : code n° $1A.

* *Paramètres d'entrée :*

— Registre A du 6809 : ordonnée à passer à la routine (0,24)
— Registre X du 6809 : abscisse à passer à la routine (1,40)

* *Paramètres de retour :*

— Registre B du 6809 : contient le code ASCII du caractère affichable reconnu, et zéro si le caractère n'est pas reconnu ou s'il n'y a pas de caractère à lire à l'endroit déterminé.

Pour les minuscules accentuées ou le ''ç'', trois appels à la routine sont nécessaires.

1er appel :

Une minuscule accentuée est détectée : ''B'' retourne $16, code ASCII de la séquence ACCENT. Le registre SS3GET (voir tableau n° 6) est chargé avec le code de l'accent (alphabet G2), et SS2GET avec le code ASCII de la minuscule.

2e appel :

''B'' retourne le code de l'accent (déjà dans SS3GET). SS3GET prend alors la valeur $80, le registre SS2GET restant intact.

3e appel :

''B'' retourne le code ASCII de la minuscule (déjà dans SS2GET). Les registres SS3GET et SS2GET sont remis à zéro.

2. Routine GETPH :

Elle permet la lecture de la couleur d'un point.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E821
— Sur MO5 : code n° $14.

* *Paramètres d'entrée :*

— Registres X et Y du 6809 : ''X'' pour l'abscisse à passer à la routine (0, 319), et ''Y'' pour l'ordonnée à passer à cette même routine (0,199).

* *Paramètre de retour :*

— Registre B du 6809 : retourne le code de la couleur.

B. APPLICATION

PROGRAMME N° XXIII

1. Description :

Nous allons écrire le mot INFORMATIQUE en colonne 10, ligne 12. L'écran aura été initialisé en SCREEN 1,4,4 auparavant.

Le mot sera lu ensuite par la routine GETSH, et recopié en colonne 10, ligne 7, dans une couleur de type COLOR 4,1. Celle-ci sera déterminée comme suit : on lit la couleur d'un point (GETPH) de la lettre I du mot INFORMATIQUE, écrite en rouge (code = 1) ; on ajoute, à ce code, la valeur nécessaire pour obtenir une inversion-vidéo manuelle. Rappelons que l'inversion-vidéo peut être, par ailleurs, obtenue par l'envoi à la routine PUTCH du code $5C sur TO7(70) et TO9, ou $7B sur MO5 (tableau n° 7).

2. Réalisation :

a. TO7(70) et TO9 :

```
            E803    PUTCH  EQU    $E803
            6041    CHDRAW EQU    $6041
            603B    COLOUR EQU    $603B
            E833    CHPLH  EQU    $E833
            E824    GETSH  EQU    $E824
            E821    GETPH  EQU    $E821

7D00                       ORG    &32000
7D00 34     7F             PSHS   U,Y,X,DP,D,CC
7D02 CE     7D56           LDU    #TABLE
7D05 E6     C0      DEBUT  LDB    ,U+
7D07 C1     04             CMPB   #4
7D09 27     05             BEQ    SUITE
7D0B BD     E803           JSR    PUTCH
7D0E 20     F5             BRA    DEBUT
7D10 8E     000A    SUITE  LDX    #10
7D13 108E   000C           LDY    #12
7D17 CE     7D61           LDU    #TABLE2
7D1A E6     C0      DEBUT2 LDB    ,U+
7D1C C1     04             CMPB   #4
7D1E 27     0A             BEQ    SUITE2
7D20 F7     6041           STB    CHDRAW
7D23 BD     E833           JSR    CHPLH
7D26 30     01             LEAX   1,X
7D28 20     F0             BRA    DEBUT2
7D2A 8E     004B    SUITE2 LDX    #75     DANS L' OCTET
                    *COLONNE NO 09=LETTRE I (72a79)
```

```
7D2D 108E 0062          LDY    #98      DANS L' OCTET
                     *LIGNE NO 12=LETTRE I (96a103)
7D31 BD   E821          JSR    GETPH
7D34 CB   E0            ADDB   #$E0
7D36 F7   603B          STB    COLOUR
7D39 8E   000A          LDX    #10
7D3C 86   0C            LDA    #12
7D3E 108E 0007          LDY    #7
7D42 BD   E824    SUITE3 JSR    GETSH
7D45 F7   6041          STB    CHDRAW
7D48 BD   E833          JSR    CHPLH
7D4B C1   45            CMPB   #$45
7D4D 27   04            BEQ    FIN
7D4F 30   01            LEAX   1,X
7D51 20   EF            BRA    SUITE3
7D53 35   7F      FIN   PULS   CC,D,DP,X,Y,U
7D55 3F                 SWI
7D56 1B 20 23 41  TABLE FCB    $1B,$20,$23,$41,$1B,$20
7D5A 1B 20
7D5C 23 54 1B 64        FCB    $23,$54,$1B,$64,$4
7D60 04
7D61 49 4E 46 4F  TABLE2 FCC   /INFORMATIQUE/
7D65 52 4D 41 54
7D69 49 51 55 45
7D6D 04                 FCB    $4
          0000          END

00000 Total Errors
```

b. MO5 :

```
          0002    PUTCH  EQU    2
          2036    CHDRAW EQU    $2036
          202B    COLOUR EQU    $202B
          0012    CHPLH  EQU    $12
          001A    GETSH  EQU    $1A
          0014    GETPH  EQU    $14

7D00                     ORG    &32000
7D00 34   7F             PSHS   U,Y,X,DP,D,CC
7D02 CE   7D53           LDU    #TABLE
7D05 E6   C0      DEBUT  LDB    ,U+
7D07 C1   04             CMPB   #4
7D09 27   04             BEQ    SUITE
7D0B 3F   02             CALL   PUTCH
7D0D 20   F6             BRA    DEBUT
7D0F 8E   000A    SUITE  LDX    #10
7D12 108E 000C           LDY    #12
7D16 CE   7D5C           LDU    #TABLE2
7D19 E6   C0      DEBUT2 LDB    ,U+
7D1B C1   04             CMPB   #4
```

```
7D1D 27   09            BEQ    SUITE2
7D1F F7   2036          STB    CHDRAW
7D22 3F   12            CALL   CHPLH
7D24 30   01            LEAX   1,X
7D26 20   F1            BRA    DEBUT2
7D28 8E   004B    SUITE2 LDX    #75      DANS L' OCTET
                  *COLONNE NO 09=LETTRE I (72a79)
7D2B 108E 0062          LDY    #98      DANS L' OCTET
                  *LIGNE NO 12=LETTRE I (96a103)
7D2F 3F   14            CALL   GETPH
7D31 CB   40            ADDB   #$40
7D33 F7   202B          STB    COLOUR
7D36 8E   000A          LDX    #10
7D39 86   0C            LDA    #12
7D3B 108E 0007          LDY    #7
7D3F 3F   1A      SUITE3 CALL   GETSH
7D41 F7   2036          STB    CHDRAW
7D44 3F   12            CALL   CHPLH
7D46 C1   45            CMPB   #$45
7D48 27   04            BEQ    FIN
7D4A 30   01            LEAX   1,X
7D4C 20   F1            BRA    SUITE3
7D4E 35   7F      FIN   PULS   CC,D,DP,X,Y,U
7D50 BD   B000          STOP
7D53 1B 20 41 1B  TABLE FCB    $1B,$20,$41,$1B,$20
7D57 20
7D58 54 1B 64 04        FCB    $54,$1B,$64,$4
7D5C 49 4E 46 4F  TABLE2 FCC   /INFORMATIQUE/
7D60 52 4D 41 54
7D64 49 51 55 45
7D68 04                 FCB    $4
          0000          END

00000 Total Errors
```

3. Explications :

Il suffit de se rappeler que pour écrire avec CHPLH, il faut placer le code ASCII du caractère dans CHDRAW, et les coordonnées dans "X" (1,40) et "Y" (0,39). La couleur sera soit celle de l'écran (comme ici), soit celle proposée par COLOUR.

La valeur à ajouter au code de la couleur rouge de la lettre I (du mot INFORMATIQUE), pour obtenir une inversion-vidéo manuelle, est de $E0 sur TO7(70) et TO9, et de $40 sur MO5. En effet, le registre COLOUR doit contenir un code identique à celui que l'on utilise pour écrire directement dans les octets de R.A.M. d'écran.

La routine GETSH se sert de "A" (= 12) et "X" (= 10), et la routine CHPLH de "Y" (= 7) et "X" (= 10). Ceci nous permet d'éviter des manipulations de coordonnées.

12. Routine K7COH

A. DESCRIPTION

Cette routine gère le lecteur-enregistreur de programmes (L.E.P.). Il faut totalement différencier les TO7(70) et TO9, du MO5.

1. TO7(70) et TO9 :

L'utilisation de cette routine permet de lire ou écrire autant d'octets qu'on le désire, les uns à la suite des autres sur une cassette. Contrairement au MO5, la routine ne différencie pas un BLOC D'EN-TETE ou un BLOC DE FIN, d'un BLOC DE DONNÉES.

Lors de l'enregistrement d'un programme-basic (SAVE), ou binaire (SAVEM en basic, ou SC : en assembleur), l'ordinateur écrit d'abord un BLOC D'EN-TÊTE contenant le code ASCII des caractères du descripteur de fichier, sur la cassette.

Exemple :

''PROG1.BAS'' équivaut à la séquence ASCII suivante : $50, $52, $4F, $47, $31, $20, $20, $20, $42, $41, $53, dans laquelle $20 représente un espace (SP). En effet, un nom de programme comprend 8 lettres ou chiffres, et des espaces comblant les emplacements des caractères manquants. Ce nom de programme, appelé descripteur de fichier, doit toujours commencer par une lettre.

Le BLOC D'EN-TETE est suivi du programme ou BLOC DE DONNÉES, qui se termine par un BLOC DE FIN contenant un test de vérification du passage des données, appelé CHECKSUM.

Sur MO5, il est nécessaire de positionner le nombre de données et le checksum, même pour n'enregistrer que des DONNÉES, mais ce n'est pas aussi absolu sur TO7(70) ou TO9, à moins de vouloir ensuite relire le programme par une instruction en basic ou en assembleur.

Il suffit, sur TO7(70) et sur TO9, de bien retenir qu'entre un BLOC D'EN-TETE et un BLOC DE DONNÉES, il est nécessaire de laisser une seconde de bande vierge pour le recalage de celle-ci, au démarrage du moteur (vous vous êtes rendu compte que le moteur s'arrête un

court instant entre le BLOC D'EN-TETE et le BLOC DE DONNÉES, lors de la lecture d'un programme sur votre magnétophone).

Il reste enfin à signaler qu'une zone en R.A.M. doit être définie pour lire ou recopier les données.

Les lignes qui suivront pour le MO5 peuvent vous être utiles pour bien comprendre tout ceci. Quant au CHECKSUM, nous allons en reparler par la suite.

* *Point d'entrée :*

— $E815

* *Paramètres d'entrée :*

— Registre B du 6809 : dans ce registre sont à placer, une à une, les données à écrire.
— Registre K7OPC ($6029) : ce registre doit contenir le code de l'opération demandée à la routine :

a. Lecture d'une bande :

— Si K7OPC = 1 lors de l'appel à la routine, cela provoquera une mise en route du moteur pour une lecture.

— Si K7OPC = 2 lors de l'appel à la routine, cela permettra la lecture d'un caractère (à répéter autant de fois qu'il y a d'octets à lire).

— Si K7OPC = $10 lors de l'appel à la routine, cela provoquera l'arrêt du moteur.

b. Écriture sur une bande :

— Si K7OPC = 4 lors de l'appel à la routine, cela provoquera la mise en marche du moteur pour une écriture.

— Si K7OPC = 8 lors de l'appel à la routine, cela permettra l'écriture d'un caractère (à répéter autant de fois qu'il y a d'octets à écrire).

— Si K7OPC = $10 lors de l'appel à la routine, cela provoquera l'arrêt du moteur.

* *Paramètres de retour :*

— Registre B du 6809 : dans ce registre sont placées, une à une, les données lues.

— Registre CC du 6809 : en cas d'erreur, le bit "C" de "CC" se positionne à 1, et le code de l'erreur est retourné dans K7STA.

— Registre K7STA ($602A) : retourne le code de l'opération réalisée, ou de l'erreur (dans ce cas, "C" = 1). L'erreur peut être de n'avoir pas raccordé son L.E.P. ! (IO ERROR en basic).

Codes passés par K7STA :

- 1 = ouvert en lecture.
- 4 = ouvert en écriture.
- $10 = moteur arrêté.
- $80 = périphérique non prêt.

2. MO5 :

Si vous avez lu les lignes concernant les TO7(70) et le TO9, vous savez qu'un programme enregistré se compose de 3 BLOCS. Lorsque vous sauvegardez un programme-basic (SAVE) ou binaire (SAVEM en basic ou SC : en assembleur), l'ordinateur écrit d'abord un BLOC D'EN-TETE contenant le code ASCII des caractères du descripteur de fichier, sur la cassette.

Exemple :

"PROG1.BAS" se traduit par la séquence ASCII suivante : $50, $52, $4F, $47, $31, $20, $20, $20, $42, $41, $53, dans laquelle $20 représente un espace (SP). En effet, un nom de programme comprend 8 lettres ou chiffres, et des espaces comblant les emplacements des caractères manquants. Ce nom de programme, appelé descripteur de fichier, doit toujours commencer par une lettre.

Derrière le programme D'EN-TETE, se trouve le BLOC DE DONNÉES qui représente le programme proprement dit ou les octets à sauvegarder. L'écriture ou la lecture d'un bloc d'octets se fait ici d'un seul coup, contrairement aux TO7(70) et TO9. L'utilisation de la routine K7COH vous permettra d'enregistrer ou de lire, sur votre magnétophone, des BLOCS de 253 octets de DONNÉES au maximum.

Le dernier BLOC DE DONNÉES est le BLOC DE FIN.

Chaque type de BLOC, quel qu'il soit, doit être encadré par un octet qui le précède, et un autre qui le suit. L'octet qui le précède doit donner le nombre total d'octets du bloc à écrire, plus deux. L'octet qui le suit est le CHECKSUM. Qu'est-ce que le CHECKSUM ? C'est une valeur, correspondant à une combinaison additive de tous les octets du bloc ; nous allons revoir cela dans un des programmes qui suivent. Ce CHECKSUM est un test de bonne transmission des données.

Il faut aussi rappeler qu'il est nécessaire de définir une zone en R.A.M. dans laquelle vous pourrez loger 255 octets (253 octets de données + 2 octets « d'encadrement »). Le programme de copie, si vous voulez sauvegarder plus de 253 octets de données, devra charger le programme à copier par tranches de 253 octets, les encadrer comme

cela vient d'être expliqué (avec un sous-programme de calcul du CHECKSUM), et les inscrire sur la cassette. Inversement, à la lecture, il faudra débarrasser les données de leurs octets de transmission ou « d'encadrement », et les coller bout à bout, à l'emplacement d'utilisation du programme.

L'ordinateur met obligatoirement sur la bande un petit espace entre le BLOC D'EN-TETE et les autres, et vous n'aurez pas à vous en préoccuper comme sur TO7(70) ou TO9. Chaque type de BLOC est écrit de manière différente sur la bande, au moyen de codes que nous allons étudier. Il est nécessaire de respecter ces codes de transfert des divers types de BLOCS.

ATTENTION : à l'inverse des TO7(70) et TO9, la routine K7COH sur MO5 ne COMMANDE PAS LA MISE EN ROUTE DU MOTEUR du magnétophone. Ceci est le rôle d'une autre petite routine, que nous allons brièvement décrire ensuite, et qui ne mérite pas un chapitre particulier. Décrivons d'abord la routine essentielle : K7COH.

* *Point d'entrée :*

— Code n° $20.

* *Paramètres d'entrée :*

— Registre A du 6809 : Si "A" est nul, la routine effectue une écriture sur la bande, sinon elle effectue une lecture.

— Registre B du 6809 : "B" doit contenir le type de BLOC (en écriture seulement).

Codes :	*Types de BLOC :* (du fichier à écrire)
00	BLOC D'EN-TETE
01	BLOC DE DONNÉES
$FF	BLOC DE FIN

— Registre Y du 6809 : "Y" doit contenir l'adresse du début de la zone-mémoire dans laquelle sera rangé le bloc lu ou, dans le cas d'une écriture, les données à écrire.

"Y" pointe donc sur l'adresse du 1er octet de transmission, égal à n + 2 (n étant le nombre de données).

— Registre K7DATA ($2040) : Ne vous en préoccupez pas, il s'agit du registre dans lequel se trouve l'octet à écrire sur la bande magnétique. Ce registre ne sert pas lors d'une lecture.

* *Paramètres de retour :*

Ils ne servent qu'à la lecture.

— Registres B et A du 6809 :

Lors d'une lecture de la bande, "B" revient avec le type de bloc lu.

Quant à "A", il revient :

1) avec la valeur 0 si l'octet de checksum, inscrit sur la bande, est correct.
2) avec le checksum calculé ET COMPLÉMENTÉ A DEUX, si l'octet de checksum inscrit sur la bande est égal à zéro.

Ceci est dû au fait que le système calcule le checksum d'après les données lues, le complémente à deux, et l'additionne à l'octet de checksum inscrit sur la bande. L'addition de deux nombres opposés produisant zéro, vous retrouverez cette valeur dans "A" chaque fois que le checksum inscrit sur la bande sera correct et que les données seront bien transmises.

Routine de mise en marche du moteur du L.E.P. :

* *Point d'entrée :*

— Code n° $22.

* *Paramètre d'entrée :*

— Registre A du 6809 :
Si le bit n° 0 de "A" est à zéro, le moteur est arrêté après une demi-seconde de délai. Ceci sert aux lecture et écriture.

Si le bit n° 0 et le bit n° 1 de "A" sont tous deux à un, le moteur est mis en route après une seconde de délai : le magnétophone est prêt pour une écriture.

Si le bit n° 0 de "A" est à un, et le bit n° 1 de "A" à zéro, le moteur est mis en route et il n'y a pas de délai : le magnétophone est prêt pour une lecture.

En résumé :

Mise en route du moteur :

Écriture :

```
LDA #3
CALL $22
```

Lecture :

```
LDA #1
CALL $22
```

Arrêt du moteur :

```
LDA #2
CALL $22
```

Tout de suite après une mise en route, vous donnerez vos instructions programmées à la routine K7COH, et à la fin, vous arrêterez le moteur (fermeture).

* *Paramètre de retour :*

Lorsque le L.E.P. n'est pas, ou mal raccordé au MO5, le bit ''C'' de ''CC'' est mis à 1, sinon il est forcé à zéro.

PROGRAMME N° XXIV

1. Description :

Nous allons sauvegarder un petit programme, avec moins de 253 octets de données pour être compatible entre TO7(70), TO9 et MO5, c'est-à-dire pour ne pas avoir à faire de calculs compliqués pour le MO5.

Nous choisirons pour cela le programme n° XVII intitulé : « écriture verticale du mot TABLE ».

2. Réalisation :

a. TO7(70) et TO9 :

```
            6029    K7OPC   EQU     $6029
            E815    K7COH   EQU     $E815

7918                        ORG     &31000
7918 34     7F              PSHS    U,Y,X,DP,D,CC
791A 86     04              LDA     #4          .OUVERTURE
791C B7     6029            STA     K7OPC       .MOTEUR
791F BD     E815            JSR     K7COH       .MODE ECRITURE
7922 25     16              BCS     FIN
7924 8E     7D00            LDX     #&32000
7927 86     08              LDA     #8
7929 B7     6029            STA     K7OPC
792C E6     80      DEBUT   LDB     ,X+
792E BD     E815            JSR     K7COH       ECRITURE
7931 25     07              BCS     FIN
7933 8C     7D4C            CMPX    #&32000+$4C
7936 27     02              BEQ     FIN
7938 20     F2              BRA     DEBUT
793A 86     10      FIN     LDA     #$10        .FERMETURE
793C B7     6029            STA     K7OPC       .
793F BD     E815            JSR     K7COH       .MOTEUR
7942 35     7F              PULS    CC,D,DP,X,Y,U
7944 3F                     SWI
            0000            END

00000 Total Errors
```

b. MO5 :

```
                0020    K7COH   EQU     $20

7918                            ORG     &31000
7918 34     7F                  PSHS    U,Y,X,DP,D,CC
791A 86     4C                  LDA     #$4A+2  LONGUEUR BLOC +2
791C B7     7CFF                STA     &31999
791F 4F                         CLRA            OU CHECKSUM ($E6)
7920 B7     7D4A                STA     $7D49+1 FIN BLOC +1
7923 108E   7CFF                LDY     #31999
7927 C6     01                  LDB     #1
7929 86     03                  LDA     #3      .MOTEUR
792B 3F     22                  CALL    $22     .MODE ECRITURE
792D 25     03                  BCS     FIN
792F 4F                         CLRA
7930 3F     20                  CALL    K7COH   ECRITURE
7932 86     02      FIN         LDA     #2      .ARRET
7934 3F     22                  CALL    $22     .MOTEUR
7936 35     7F                  PULS    CC,D,DP,X,Y,U
7938 BD     B000                STOP
            0000                END

00000 Total Errors
```

3. Explications :

a. T07(70) et TO9 :

Le programme à sauvegarder est au préalable implanté en &32000 (par assemblage ou par des POKE en basic).

Le programme de sauvegarde est implanté un peu plus bas, en &31000.

Nous connaissons le nombre d'octets à sauvegarder : $4C. Lorsque tous seront lus, le registre d'adresse X pointera sur le dernier octet plus un : &32000 + $4C (puisque &32000 est compté comme premier octet).

La 1re phase est la mise en route du moteur, en écriture : #4 dans K7OPC et appel de K7COH. Nous prenons ensuite les octets pour les inscrire un à un : mise de l'octet dans "B" (#8 ayant été mis au préalable dans K7OPC) et appel de K7COH. Lorsque "X" = &32000 + $4C, nous arrêtons le moteur : #$10 dans K7OPC et appel de la routine K7COH. Le programme est alors terminé.

Tout appel à une routine, gérant un périphérique, peut être suivi d'un BCS FIN (condition "C" = 1), pour arrêter le programme en cas d'erreur. Il est possible, par ailleurs, de ne pas sauvegarder "CC" en

début de programme, mais de le faire à la fin de celui-ci. Lors du dépilement final, en cas d'erreur, on retrouve alors "C" = 1, et le code de l'erreur dans le registre de retour approprié.

b. MO5 :

Le programme à sauvegarder est au préalable implanté en &32000 (par assemblage ou par des POKE en basic).

Le programme de sauvegarde est placé un peu plus bas, en &31000. On met en &31999 le premier octet, celui d'encadrement contenant "n données + 2", c'est-à-dire $4A + 2 ($4A = nombre d'octets du programme à sauvegarder).

Si on ne connaît pas le checksum, on peut placer 0 dans le dernier octet (octet d'encadrement), soit en $7D49 + 1 (fin du BLOC DE DONNÉES + 1). A la lecture, le programme reviendra avec le checksum calculé, mais complémenté à deux ($1A). On peut aussi calculer le checksum ($E6) et le placer en $7D49 + 1. Le programme reviendra alors, à la lecture, avec un checksum à zéro signifiant : données bien transmises. Pour le calcul du checksum, voir le programme n° XXVI.

On fait démarrer le moteur pour une écriture : LDA # 3, et CALL $22. En cas d'erreur, BCS FIN (condition "C" = 1) permet d'arrêter le programme*. Nous avons chargé "B" avec la valeur 1 : en effet nous ne voulons ici sauvegarder qu'un BLOC DE DONNÉES, sans EN-TETE ni BLOC DE FIN ; nous le relirons tel quel, et il sera impossible de le charger par LOADM en basic, ou "LC:" en assembleur.

Après avoir mis "A" à zéro (CLRA), pour une écriture, on appelle la routine de sauvegarde (K7COH = $20). Celle-ci prend le 1er octet en &31999 (valeur de "Y").

On arrête le moteur en fin de programme : LDA # 2 et CALL $22.

* Il est possible de ne pas sauvegarder "CC" en début de programme, mais de le faire à la fin de celui-ci. Lors du dépilement final, en cas d'erreur, on retrouve alors la « condition » ("C" = 1) de cette erreur.

PROGRAMME N° XXV

1. Description :

Éteignez l'ordinateur pour vider sa mémoire à coup sûr, et rallumez-le.

Nous allons maintenant relire les données préalablement sauvegardées, ce qui nous permettra de faire fonctionner le magnétophone dans les 2 sens et de vérifier que nos programmes « tournent » bien.

Nous chargerons donc, uniquement par ses données, et en langage-machine car ce ne serait pas possible autrement, le programme n° XVII (écriture verticale du mot TABLE), et nous vérifierons que ce programme fonctionne correctement.

2. Réalisation :

a. TO7(70) et TO9 :

```
            6029    K7OPC   EQU     $6029
            E815    K7COH   EQU     $E815

7918                        ORG     &31000
7918 34     7F              PSHS    U,Y,X,DP,D,CC
791A 86     01              LDA     #1          .OUVERTURE
791C B7     6029            STA     K7OPC       .MOTEUR
791F BD     E815            JSR     K7COH       .MODE LECTURE
7922 25     16              BCS     FIN
7924 8E     7D00            LDX     #&32000
7927 86     02              LDA     #2
7929 B7     6029            STA     K7OPC
792C BD     E815    DEBUT   JSR     K7COH       LECTURE
792F 25     09              BCS     FIN
7931 E7     80              STB     ,X+
7933 8C     7D4C            CMPX    #&32000+$4C
7936 27     02              BEQ     FIN
7938 20     F2              BRA     DEBUT
793A 86     10      FIN     LDA     #$10        .FERMETURE
793C B7     6029            STA     K7OPC       .
793F BD     E815            JSR     K7COH       .MOTEUR
7942 35     7F              PULS    CC,D,DP,X,Y,U
7944 3F                     SWI
            0000            END

00000 Total Errors
```

b. MO5 :

```
          0020     K7COH  EQU     $20

7918                     ORG     &31000
7918 34   7F             PSHS    U,Y,X,DP,D,CC
791A 108E 7CFF           LDY     #&31999
791E 86   01             LDA     #1        .MOTEUR
7920 3F   22             CALL    $22       .MODE LECTURE
7922 25   08             BCS     FIN
7924 3F   20             CALL    K7COH     LECTURE
7926 B7   7FFF           STA     $7FFF     SAUV.CHECKSUM
7929 F7   7FFE           STB     $7FFE     SAUV.TYPE BLOC
792C 86   02       FIN   LDA     #2        .ARRET
792E 3F   22             CALL    $22       .MOTEUR
7930 35   7F             PULS    CC,D,DP,X,Y,U
7932 BD   B000           STOP
          0000           END

00000 Total Errors
```

3. Explications :

a. TO7(70) et TO9 :

On met en marche le moteur du L.E.P. pour une lecture : # 1 dans K7OPC et appel à la routine K7COH. Nous donnons à "X" l'adresse d'implantation du programme lu : &32000. Le programme de traitement est en &31000. On lit un à un les octets par la routine K7COH (la valeur # 2 ayant été auparavant mise dans K7OPC). C'est "B" qui est, en retour, chargé de l'octet lu et le place à l'adresse pointée par "X", autoincrémenté jusqu'au dernier octet du programme + 1 (l'autoincrémentation se faisant après l'instruction, à l'inverse de l'autodécrémentation). L'adresse de fin est &32000 + $4C – 1 ; la lecture s'arrête donc pour "X" = &32000 + $4C. On stoppe alors le moteur : #$10 dans K7OPC et appel de K7COH.

On vérifiera enfin par G&32000 dans le moniteur-binaire, ou par EXEC 32000 en basic, que notre programme lu fonctionne correctement.

b. MO5 :

On charge "Y", qui doit contenir l'adresse d'implantation du programme lu, avec &31999 où sera stocké le 1er octet (d'encadrement) = nombre de données + 2.

On met la valeur 1 dans "A" pour appeler la routine $22 : mise en route du moteur du L.E.P. pour une lecture.

On laisse ''A'' ensuite à 1 (différent de zéro = lecture), pour appeler K7COH (code n° $20). On conserve le checksum calculé et complémenté à 2, se trouvant dans ''A'', en le stockant en $7FFF. On conserve le type de BLOC se trouvant dans ''B'', en le stockant en $7FFE. Ces valeurs, conservées dans ces cases-mémoire, pourront être ainsi relues après la fin du programme. Il est nécessaire de stocker ces valeurs si on désire les relire, car ''A'' va changer par la suite et d'autre part l'instruction PULS D, contenant donc ''A'' et ''B'', va faire reprendre à ces registres les valeurs qu'ils avaient avant le programme.

On arrête enfin le moteur : # 2 dans ''A'' et CALL $22.

PROGRAMME N° XXVI

1. Description :

Il s'agit ici de calculer un CHECKSUM. Tant qu'à faire, calculons le checksum du programme précédent et stockons-le, ainsi que le nombre de données + 2, à sa place (ceci est essentiellement valable pour MO5).

Ce programme permet aussi, bien sûr, un calcul de checksum pour TO7(70) et TO9, mais dans ce cas n'oubliez pas que le nombre de données pour ces 2 appareils est de $4C. Le calcul de checksum pour TO7(70) et TO9 ne vous sera pas utile dans cet ouvrage, ne passez cependant pas au-dessus du programme qui suit.

Pour le matériel THOMSON, le checksum est une combinaison additive de tous les octets de données d'un BLOC, à l'exception des octets dits d'encadrement.

Le checksum sert à vérifier la bonne transmission des données d'un BLOC, quel qu'il soit sur MO5, et lorsque ce checksum a été bien calculé, le registre qui doit le contenir en retour revient avec la valeur zéro lors de la lecture.

2. Réalisation :

a. MO5 :

```
7530                    ORG     &30000
7530 86   4C            LDA     #$4A+2   NOMBRE DONNEES+2
7532 B7   7CFF          STA     &31999
7535 8E   7D00          LDX     #32000
7538 A6   80            LDA     ,X+
753A AB   80     DEBUT  ADDA    ,X+      CALCUL DU CHECKSUM
753C 8C   7D4A          CMPX    #$7D4A
753F 26   F9            BNE     DEBUT
7541 B7   7D4A          STA     $7D4A    OCTET DE CHECKSUM
7544 BD   B000          STOP
          0000          END

00000 Total Errors
```

3. Explications :

Sur TO7(70) et TO9, remplacez $4A par $4C ; le nombre de données + 2 n'est pas à préciser en &31999.

On charge le nombre de données + 2 du programme ($4A + 2) dans

''A'', et on stocke cette valeur en &31999. On place l'adresse du début des données proprement dite (&32000) dans ''X'', et on additionne ces données les unes aux autres, par l'instruction : ADDA ,X+ (ADDA = ajoute à l'accumulateur A la valeur contenue dans le champ-opérande : ,X+). On ne s'occupe pas des retenues (c'est donc une combinaison additive plus qu'une addition réelle). La fin du programme est en &32000 + $4A-1, sur MO5. Il faut comparer ''X'' à $7D4A pour « stopper » l'addition. Voilà, on stocke le checksum calculé ($E6 sur MO5, et $E8 sur TO7(70) et TO9) en $7D4A. Si vous désirez connaître le checksum complémenté à deux, que vous retrouverez dans ''A'' (MO5) à la lecture si vous avez mis l'octet de checksum à zéro sur la bande, vous pouvez ajouter NEGA, juste avant le stockage de ''A'' en $7D4A. L'instruction COMA représenterait, quant à elle, la complémentation à un même registre ''A''.

13. Routine DKCOH

Les programmes d'application de ce chapitre ne sont pas conçus pour fonctionner sur nano-réseau.

A. DESCRIPTION

Cette routine permet la gestion du lecteur-enregistreur de disquettes. Ce lecteur-enregistreur peut être de simple ou de double densité. Après initialisation du contrôleur, placé sous le lecteur ou intégré (TO9), le registre DKSTA contiendra l'information nécessaire à l'ordinateur pour savoir s'il est relié à un lecteur de simple ou de double densité.

Il vous faudra définir une zone (dite zone-tampon) en R.A.M., comme pour le L.E.P., dans laquelle seront lues ou rangées les DONNÉES à transcrire. L'EN-TETE est ici remplacée par le CATALOGUE (DIRECTORY).

Lorsqu'on se sert de la DIRECTORY, en basic ou en assembleur par exemple, on travaille à un niveau LOGIQUE. Avec le langage-machine, il est possible de travailler directement sur la disquette, au niveau PHYSIQUE, ce que nous allons faire. Nous ne mettrons donc pas d'EN-TETE, ou plutôt nous ne remplirons aucune zone du CATALOGUE, comme nous l'avons déjà fait pour le magnétophone. Inutile de dire que nos DONNÉES ne seront lisibles qu'en langage-machine ; vous devinez maintenant quels seront les programmes qui vont suivre.

Les disquettes sont divisées en 40 PISTES sur TO7(70) et MO5 ou 80 PISTES sur TO9, de 16 SECTEURS CHACUNE. Un secteur contient 128 OCTETS en simple densité, et 256 en double densité.

Avec les lecteurs de double densité, vous pouvez opter pour un travail en simple densité (qui peut le plus, peut le moins) ; il vous suffit de le programmer, après l'initialisation du contrôleur qui choisit la densité d'office.

La zone-tampon ne peut excéder 128 octets de données, pour un secteur. Ceci est valable en simple densité ; en double densité, vous aurez

la chance de pouvoir doubler vos possibilités (nouveau lecteur-enregistreur de disquettes ou TO9). Les programmes qui vont suivre ont été réalisés, pour des raisons de compatibilité entre tous, pour le lecteur-enregistreur de simple densité. Ils fonctionnent néanmoins sur les lecteurs-enregistreurs de double densité.

Tout programme important devra être découpé en blocs de 128 octets, dont un dernier égal ou inférieur à ce nombre. Chaque bloc de données sera sauvegardé sur un seul secteur à la fois. Plus question ici, même avec les TO7(70) et TO9, de travailler octet par octet. Les possesseurs de TO7(70) et de TO9 rejoignent donc les possesseurs de MO5. Ces derniers sont ici avantagés, car ils ont déjà utilisé cette technique pour leur magnétophone. Il y a maintenant compatibilité, dans le texte, entre tous les appareils auxquels ce livre est destiné.

* *Point d'entrée :*

— Sur TO7(70) et TO9 : $E82A
— Sur MO5 : code n° $26
— De plus, sur TO9 : $E004. Il existe aussi, sur TO9, un point d'entrée pour le formatage : DKFORMH = $E007.

* *Paramètres d'entrée :*

— *Registre DKOPC :*

- TO7(70) et TO9 : $6048
- MO5 : $2048

Ce registre doit contenir le code de l'opération à réaliser :

— code 0 (sur TO9 seulement) : formatage sans vérification.

— code 1 : initialisation du contrôleur. Si l'initialisation s'est faite sans erreur, le bit C du registre CC revient à zéro, sinon il est mis à 1 et le code d'erreur $40 est retourné par DKSTA (voir tableau des erreurs qui suit). Si l'opération s'est faite correctement, le registre DKSTA revient avec le code ASCII de la lettre C en simple densité, ou le code ASCII de la lettre D en double densité.

— code 2 : lecture d'un secteur. Les erreurs possibles auront les codes suivants : 2, 4, 8, $10, $80 (voir tableau qui suit).

— code 4 : sur un lecteur de double densité, passage en simple densité. Pas d'erreur possible, sauf pour le lecteur de simple densité.

— code 8 : écriture d'un secteur. Les erreurs possibles auront les codes suivants : 1, 2, 4, 8, $10, $80.

— code $10 : passage en double densité sur les lecteurs pré-

vus à cet effet. Pas d'erreur possible, sauf pour le lecteur de simple densité.

— code $20 : recherche de la piste 0. Les codes d'erreurs possibles sont : $10 et $80.

— code $40 : recherche de la piste dont le numéro est passé par le registre DKTRK. Les codes d'erreurs possibles sont : $10 et $80.

— code $80 : option de vérification en écriture. Il est nécessaire de faire un ET LOGIQUE entre ce code et le code de l'opération à vérifier. Les codes d'erreurs possibles sont ceux de l'opération effectuée, augmentés du code $20 pour chacun d'entr'eux.

— *Registre DKDRV :*

- TO7(70) et TO9 : $6049
- MO5 : $2049

— TO7(70) et MO5 : Ce registre doit contenir le numéro de lecteur utilisé (de 0 à 3).

— TO9 : Ce registre doit contenir le numéro de lecteur utilisé (de 0 à 4). Le n° 0 correspond au lecteur interne (le n° 1 ne peut être utilisé qu'avec un lecteur hypothétique de 2 faces), et les n^{os} 2 et 3 aux 2 faces du lecteur externe (5 pouces 1/4). Si le lecteur externe est de type 3 pouces 1/2, seul le n° 2 est actuellement utilisable. Le n° 4 est le R.A.M. DISK : l'extension R.A.M. de 64 k-octets est gérée comme une unité de disque en double densité, ayant 16 pistes (de 16 à 32) de 16 secteurs contenant chacun 256 octets.

— *Registre DKTRK :*

- TO7(70) et TO9 : ($604A-$604B)
- MO5 : ($204A-$204B)

Ce registre doit contenir le numéro de piste à utiliser (de 0 à 39 sur TO7(70) et MO5, ou de 0 à 79 sur TO9).

— *Registre DKSEC :*

- TO7(70) et TO9 : $604C
- MO5 : $204C

Ce registre doit contenir le numéro de secteur à utiliser (de 1 à 16).

— *Registre DKNUM :*

- TO9 seulement : $604D

Ce registre contient l'entrelacement des secteurs logiques sur la disquette lors du formatage, et permet donc d'accélérer les vitesses d'accès au disque. Sur TO9, l'entrelacement utilisé par le basic est

de 7. Une lecture rapide des secteurs peut être optimale avec un entrelacement de 2.

— *Registre DKBUF :*

- TO7(70) et TO9 : ($604F-$6050)
- MO5 : ($204F-$2050).

Ce registre doit contenir l'adresse de la zone-tampon dans laquelle est (ou sera) implanté le programme à sauvegarder (ou à lire). Cette zone doit être de 128 octets maximum en simple densité, et de 256 octets maximum en double densité.

* *Paramètres de retour :*

— *Registre DKSTA :*

- TO7(70) et TO9 : $604E
- MO5 : $204E

Ce registre contient, en sortie de routine, le type du contrôleur après son initialisation, ou le code d'erreur retourné lors d'une opération quelconque.

— *Registre 6809 CC :*

En cas d'erreur, au cours d'une opération, le bit ''C'' de ''CC'' est forcé à 1, sinon il revient à zéro.

— *Registre DKFLG :*

- TO7(70) et TO9 : $6080
- MO5 : $2080

La présence du contrôleur de disquettes est attestée par la mise à $FF de ce registre. Si ce registre contient 0, le contrôleur est absent.

Comme pour les routines qui précèdent, vous trouverez dans le tableau n° 6 d'autres registres non cités ici.

* *Tableau des codes d'erreurs possibles :*

— code n° 1 : disquette protégée. Erreur en écriture seulement bien sûr.

— code n° 2 :

- TO7(70) et MO5 : Erreur de piste. L'identificateur de piste peut être correct, mais ne correspond pas à la piste demandée.
- TO9 : Il existe des problèmes de timing, ou des données ont été perdues.

— code n° 4 : Erreur de secteur. Identificateur de secteur incorrect (soit qu'il ne peut être lu, ou erreur sur le checksum*). La piste peut, elle, être correcte.

— code n° 8 : Erreur sur les données. L'identificateur est correct, mais les données ne peuvent être lues, ou le checksum* est incorrect.

— code n° $10 : Lecteur non prêt : inexistant, ou moteur qui n'est pas en marche.

— code n° $20 : Erreur sur vérification, c'est-à-dire qu'il n'y a pas identité entre la zone-tampon en mémoire et la zone correspondante sur la disquette.

— code n° $40 (sauf TO9) : contrôleur non prêt.

— code n° $80 (sauf TO9) : disquette non formatée. L'identificateur de piste ne peut être lu.

** Le format Basic Microsoft (R) :*

Les fichiers créés par le TO7(70), TO9 et MO5, suivent le standard Basic Microsoft (R).

La piste 20 est réservée à la Directory :
— secteur 1 : réservé.
— secteur 2 : FAT ou table d'allocation des fichiers.
— secteur 3 à 16 : catalogue.

a. La FAT :

Les fichiers sont inscrits à raison de 2 blocs par piste. En simple densité, un bloc est égal à 1K-octet et, en double densité, à 2 K-octets.

Chaque octet de la table d'allocation des fichiers représente un bloc physique et ce, à partir de l'octet n° 1 de cette FAT.

La FAT est organisée comme suit :

— octet 0 : rien = 0
— octet 1 : bloc 0, piste 0, secteurs 1 à 8
— octet 2 : bloc 1, piste 0, secteurs 9 à 16
— octet 3 : bloc 2, piste 1, secteurs 1 à 8
— octet 4 : bloc 3, piste 1, secteurs 9 à 16...
— ... jusqu'à l'octet 80 : bloc 79, piste 39, secteurs 9 à 16 (TO7(70) et MO5).
— ... jusqu'à l'octet 160 : bloc 159, piste 79, secteurs 9 à 16 (TO9).

* checksum du contrôleur, transparent pour le programmeur.

En simple densité, sur T09, la FAT est limitée à 127 blocs.

Un octet de la FAT, représentant un bloc physique, peut avoir comme valeurs :

— $FF : bloc non alloué
— $FE : bloc réservé
— x compris entre 0 et $BF, où x = numéro du bloc logique suivant du même fichier. Le bloc représenté par cet octet est donc alloué.
— y compris entre $C1 et $C8, signifiant dernier bloc alloué d'un fichier. Les 4 bits de poids faible de y indiquent le nombre de secteurs utilisés dans ce bloc.

b. Le catalogue :

Il occupe 14 secteurs (3 à 16) de la piste 20. Ce catalogue contient le nom des fichiers. Chaque nom de fichier occupe 32 octets. Il y a donc 4 fichiers par secteur en simple densité, et 8 en double densité. Vous pouvez répertorier ainsi 56 fichiers en simple densité, et 112 en double densité.

Répartition des 32 octets d'un nom de fichier :

— octets 0 à 7 : nom du fichier, cadré à gauche, complété par des blancs. L'octet n° 0 est le 1er octet d'une entrée dans le catalogue (voir ci-après).

— octets 8 à $A : suffixe du fichier (BAS, BIN, etc.) toujours cadré à gauche et complété par des blancs éventuellement.

— octet $B : type de fichier, dont voici les codes :

0 : programme-basic ASCII ou binaire.
1 : données-basic en ASCII.
2 : programme en langage-machine (binaire).
3 : fichier-assembleur édité en ASCII.

— octet $C : sémaphore où $FF signifie ASCII, et 0 signifie binaire.

— octet $D : numéro du premier bloc logique du fichier (pour le dernier, voir la FAT).

— octets $E-$F : nombre d'octets utilisés dans le dernier secteur du fichier.

— octets $10 à $1F sur TO7(70) et MO5, ou $18 à $1F sur T09 : réservés.

— octets $10 à $17 sur T09 : commentaire associé au fichier.

Premier octet d'une entrée dans le catalogue :

Cet octet peut contenir :

— 0 : l'entrée n'est pas allouée, il n'y a pas de fichier répertorié pour cette entrée.
— $20 à $7F : c'est le code ASCII du premier caractère du nom d'un fichier, l'entrée est donc allouée.
— $FF : signale la fin logique du catalogue.

N.B.

A la création du catalogue, les octets d'entrée sont mis à $FF. A chaque création de fichier, la fin logique du catalogue est déplacée dans la 1re entrée suivante, jusqu'au remplissage du catalogue. Lorsqu'un fichier est détruit, le premier octet de son entrée est mis à zéro. Tout fichier nouvellement créé se verra attribué, en priorité, cette entrée.

PROGRAMME N° XXVII

Ce programme ne fonctionne pas sur nano-réseau.

1. Description :

ATTENTION !! : choisissez une disquette vierge, formatée. Nous allons recopier, sur celle-ci, le programme n° XVII : « écriture verticale du mot TABLE ».

Pour ce faire, il faut positionner en &32000 le programme à sauvegarder, par un assemblage, ou par des POKE en basic.

2. Réalisation :

a. TO7(70) et TO9 :

```
                    *ATTENTION CHANGEZ VOTRE DISQUETTE!!!!!!

            6048    DKOPC   EQU     $6048
            6049    DKDRV   EQU     $6049
            604B    DKTRK   EQU     $604B
            604C    DKSEC   EQU     $604C
            604F    DKBUF   EQU     $604F
            E82A    DKCOH   EQU     $E82A
            604E    DKSTA   EQU     $604E   REGISTRE RETOUR

7918                        ORG     &31000
7918 34     7F              PSHS    U,Y,X,DP,D,CC
791A 4F                     CLRA
791B B7     6049            STA     DKDRV   LECTEUR NO ZERO
791E 4C                     INCA
791F B7     6048            STA     DKOPC
7922 BD     E82A            JSR     DKCOH   INIT.CONTROLEUR
7925 25     22              BCS     FIN
7927 86     27              LDA     #&39
7929 B7     604B            STA     DKTRK   PISTE
792C 86     10              LDA     #&16
792E B7     604C            STA     DKSEC   SECTEUR
7931 86     40              LDA     #$40
7933 B7     6048            STA     DKOPC
7936 BD     E82A            JSR     DKCOH   RECHERCHE DE PISTE
7939 25     0E              BCS     FIN
793B 8E     7D00            LDX     #&32000
793E BF     604F            STX     DKBUF   ADRESSE BUFFER
7941 86     08              LDA     #8
7943 B7     6048            STA     DKOPC
```

```
7946 BD   E82A          JSR    DKCOH   ECRITURE
7949 35   7F      FIN   PULS   CC,D,DP,X,Y,U
794B 3F                 SWI
          0000          END

00000 Total Errors
```

b. MO5 :

```
                *ATTENTION CHANGEZ VOTRE DISQUETTE!!!!!!

          2048  DKOPC  EQU    $2048
          2049  DKDRV  EQU    $2049
          204B  DKTRK  EQU    $204B
          204C  DKSEC  EQU    $204C
          204F  DKBUF  EQU    $204F
          0026  DKCOH  EQU    $26
          204E  DKSTA  EQU    $204E   REGISTRE RETOUR

7918                   ORG    &31000
7918 34   7F           PSHS   U,Y,X,DP,D,CC
791A 4F                CLRA
791B B7   2049         STA    DKDRV   LECTEUR NO ZERO
791E 4C                INCA
791F B7   2048         STA    DKOPC
7922 3F   26           CALL   DKCOH   INIT. CONTROLEUR
7924 25   20           BCS    FIN
7926 86   27           LDA    #&39
7928 B7   204B         STA    DKTRK   PISTE
792B 86   10           LDA    #&16
792D B7   204C         STA    DKSEC   SECTEUR
7930 86   40           LDA    #$40
7932 B7   2048         STA    DKOPC
7935 3F   26           CALL   DKCOH   RECHERCHE DE PISTE
7937 25   0D           BCS    FIN
7939 8E   7D00         LDX    #&32000
793C BF   204F         STX    DKBUF   ADRESSE BUFFER
793F 86   08           LDA    #8
7941 B7   2048         STA    DKOPC
7944 3F   26           CALL   DKCOH   ECRITURE
7946 35   7F      FIN  PULS   CC,D,DP,X,Y,U
7948 BD   B000         STOP
          0000         END

00000 Total Errors
```

3. Explications :

Nous mettons d'abord le numéro du lecteur-enregistreur de disquettes dans le registre DKDRV, puis nous initialisons le contrôleur (valeur 1 dans DKOPC et appel de la routine DKCOH). Cette initialisation, faite à l'allumage des appareils, n'est pas obligatoire. En cas d'erreur à l'initialisation, le programme se branche directement à la fin (BCS = branchement si ''C'' = 1).

On charge ensuite le n° de piste (39 en décimal dans notre exemple) dans le registre DKTRK, et le n° de secteur (16 en décimal dans notre exemple) dans DKSEC.

Avec $40 en DKOPC, et appel à la routine DKCOH, nous positionnons la tête de lecture-écriture sur le début de la piste signalée par DKTRK. En cas d'erreur, le programme ira à la fin.

Il faut charger le registre DKBUF avec l'adresse du BUFFER ou ZONE-TAMPON, dans laquelle sont stockées toutes les données à transcrire.

Enfin, par le code 8 mis dans DKOPC, suivi d'un appel à la routine DKCOH, nous écrivons notre programme sur la disquette.

PROGRAMME N° XXVIII

Ce programme ne fonctionne pas sur nano-réseau.

1. Description :

Nous allons éteindre puis rallumer l'ordinateur, pour être sûr que le programme (n° XVII) précédemment enregistré sur disquette puisse maintenant être bien relu, et qu'il ne s'agisse pas de sa persistance en R.A.M. . Nous le testerons ensuite pour vérifier son bon fonctionnement.

Le programme n° XVII sera copié en &32000. Le programme d'exécution sera implanté en &31000 comme toujours.

2. Réalisation :

a. TO7(70) et TO9 :

```
            6048    DKOPC   EQU     $6048
            6049    DKDRV   EQU     $6049
            604B    DKTRK   EQU     $604B
            604C    DKSEC   EQU     $604C
            604F    DKBUF   EQU     $604F
            E82A    DKCOH   EQU     $E82A

7918                        ORG     &31000
7918 34     7F              PSHS    U,Y,X,DP,D,CC
791A 4F                     CLRA
791B B7     6049            STA     DKDRV   LECTEUR NO ZERO
791E 4C                     INCA
791F B7     6048            STA     DKOPC
7922 BD     E82A            JSR     DKCOH   INIT.CONTROLEUR
7925 25     22              BCS     FIN
7927 86     27              LDA     #&39
7929 B7     604B            STA     DKTRK   PISTE
792C 86     10              LDA     #&16
792E B7     604C            STA     DKSEC   SECTEUR
7931 8E     7D00            LDX     #&32000
7934 BF     604F            STX     DKBUF   ZONE INSCRIPTION
7937 86     40              LDA     #$40
7939 B7     6048            STA     DKOPC
793C BD     E82A            JSR     DKCOH   RECHERCHE DE PISTE
793F 25     08              BCS     FIN
7941 86     02              LDA     #2
7943 B7     6048            STA     DKOPC
7946 BD     E82A            JSR     DKCOH   LECTURE
7949 35     7F      FIN     PULS    CC,D,DP,X,Y,U
794B 3F                     SWI
            0000            END

00000 Total Errors
```

b. MO5 :

```
                2048    DKOPC   EQU     $2048
                2049    DKDRV   EQU     $2049
                204B    DKTRK   EQU     $204B
                204C    DKSEC   EQU     $204C
                204F    DKBUF   EQU     $204F
                0026    DKCOH   EQU     $26

7918                            ORG     &31000
7918 34 7F                      PSHS    U,Y,X,DP,D,CC
791A 4F                         CLRA
791B B7 2049                    STA     DKDRV   LECTEUR NO ZERO
791E 4C                         INCA
791F B7 2048                    STA     DKOPC
7922 3F 26                      CALL    DKCOH   INIT. CONTROLEUR
7924 25 20                      BCS     FIN
7926 86 27                      LDA     #&39
7928 B7 204B                    STA     DKTRK   PISTE
792B 86 10                      LDA     #&16
792D B7 204C                    STA     DKSEC   SECTEUR
7930 8E 7D00                    LDX     #&32000
7933 BF 204F                    STX     DKBUF   ZONE INSCRIPTION
7936 86 40                      LDA     #$40
7938 B7 2048                    STA     DKOPC
793B 3F 26                      CALL    DKCOH   RECHERCHE DE PISTE
793D 25 07                      BCS     FIN
793F 86 02                      LDA     #2
7941 B7 2048                    STA     DKOPC
7944 3F 26                      CALL    DKCOH   LECTURE
7946 35 7F      FIN             PULS    CC,D,DP,X,Y,U
7948 BD B000                    STOP
                0000            END

00000 Total Errors
```

3. Explications :

Comme pour le programme précédent, nous inscrivons le numéro de lecteur-enregistreur dans le registre DKDRV.

Nous initialisons ensuite le contrôleur ; cette opération reste bien sûr facultative.

Il faut positionner les piste et secteur à lire, comme lors de l'écriture.

L'adresse à laquelle le 1er octet sera copié est donc &32000.

Nous cherchons enfin la piste n° 39 comme lors de l'écriture, et donnons à DKOPC le code de LECTURE (2) avant d'appeler la routine DKCOH.

Voilà, essayons : G&32000 dans le moniteur-binaire, ou EXEC 32000 en basic... Si votre programme ne fonctionne pas, vérifiez vos opérations d'ÉCRITURE d'abord, puis de LECTURE.

Si vous n'avez pas nettoyé la R.A.M. avant de lire le programme XVII sur votre disquette, vous pouvez croire avoir réussi, mais faussement peut-être...

14. Routine RSCOH

Le programme d'application de ce chapitre n'est pas conçu pour fonctionner sur nano-réseau.

A. DESCRIPTION

Il s'agit de la dernière routine que nous étudierons ; elle permet la gestion de l'interface de communication. Nous allons la détailler, et étudier ensuite son utilité dans le cadre d'une copie graphique d'écran sur imprimante à impact PR90-080. Cette copie graphique garde toujours son intérêt, malgré la sortie du BASIC 128 ; celui-ci, en effet, ne possède pas l'instruction adéquate pour la « PR90-080 ». Ce ne sera pas peine perdue, pour ceux qui disposent d'un nouveau type d'imprimante, que d'aborder un exemple de programmation avec cette routine, et les difficultés maintenant relatives pour tous d'une copie d'écran en langage-machine.

Il est préférable d'étudier séparément cette routine pour TO7(70) et TO9, ou MO5.

1. TO7(70) et TO9 :

* *Point d'entrée :* $E812

* *Paramètres d'entrée :*

— *Registre B du 6809 :*
Il doit contenir l'octet à passer à l'interface.

— *Registre RSOPC :* $602B

Le contenu de ce registre sélectionne l'opération à effectuer, dont voici les codes :

- code n° 1 : ouverture en lecture-écriture (RS232*).
- code n° 2 : lecture d'un caractère (RS232*)
- code n° 4 : ouverture en écriture seule (RS232*)
- code n° 8 : écriture d'un caractère, en parallèle
- code n° 9 (TO9) : écriture d'un caractère, en série

* RS232 = pour l'interface seulement, et non pour l'imprimante.

— code n° $C (TO9) : écriture d'un caractère, en série
— code n° $10 : fermeture, en parallèle
— code n° $11 (TO9) : fermeture, en série
— code n° $14 (TO9) : fermeture, en série
— code n° $20 : copie graphique d'écran
— code n° $40 : ouverture en écriture, en parallèle

Lorsque la ligne « série » est seule ouverte, une fermeture en parallèle sera considérée comme une fermeture en série.

— *Registre BAUDS :* ($6044-$6045)

• TO7(70) : La vitesse de transmission, qui peut varier de 110 à 4800 BAUDS*, est inscrite dans ce registre sous forme de paramètre. Ce paramètre est pris dans la table BDTAB, située en $E836. Le 1er paramètre (sur 2 octets) représente 110 BAUDS, et les suivants : 300, 600, 1200, 2400 et 4800 BAUDS. Vous pouvez ne pas vous en préoccuper pour l'instant.

• TO9 : voir le chapitre P9.

— *Registre NOMBRE :* $6046

Pour les transmissions en série, le registre NOMBRE doit contenir la valeur $80 pour transmettre 8 bits de données pour un octet, ou $40 pour transmettre 7 bits seulement. Pour le TO9, voir aussi le chapitre P9.

— *Registre GRCODE :* $6047

En cas de copie graphique d'écran, ce registre doit contenir le code, SPECIFIQUE A L'IMPRIMANTE, de mise en mode graphique (consulter la notice de l'imprimante). Ce registre contient la valeur 7 par défaut. Pour la « PR90-080 », le code de mise en mode graphique est 8, comme celui de l'écriture d'un caractère. Ceci nous permettra lors du programme suivant de ne faire qu'un seul appel à la routine, pour ces deux opérations. En appelant la routine RSCOH, avec la valeur 8 dans ''B'', il n'est effectivement pas nécessaire de déposer 8 dans GRCODE. Sur TO9, la copie graphique d'écran s'exécute selon le mode 40 ou 80 colonnes de l'écran.

* *Paramètres de retour :*

— *Registre CC du 6809 :*
Le bit ''C'' de ''CC'' est mis à zéro si tout s'est passé correctement, sinon il est forcé à un, comme d'habitude.

* BAUD : unité de fréquence dans la transmission de données, à rapprocher des bits par seconde.

— *Registre RSTA :* $602C

Ce registre retourne le code de l'opération réalisée ou, en cas d'erreur, un des codes suivants :

- code n° 1 : ouverture en lecture-écriture (RS232)
- code n° 4 : ouvert en écriture seule (RS232)
- code n° $10 : fermé
- code n° $40 : ouvert en écriture, en parallèle
- code n° $80 : périphérique non prêt

2. MO5 :

* *Point d'entrée :*

— code n° $24

* *Paramètres d'entrée :*

— *Registre B du 6809 :*

Ce registre doit contenir l'octet à envoyer à l'interface.

— *Registre PROPC :* ($2042)

Ce registre contient le code de l'opération à effectuer :

- code n° 1 : écriture d'un octet, en parallèle
- code n° 2 : copie graphique d'écran
- code n° 4 : ouverture pour écriture, en parallèle
- code n° $10 : fermeture

— *Registre GRCODE :* ($2077)

Ce registre doit contenir le code de mise en mode graphique, SPECIFIQUE A L'IMPRIMANTE (voir notice de votre imprimante). Il contient la valeur 7 par défaut. Par exemple, pour la « PR90-080 », le code de mise en mode graphique est : 8.

* *Paramètres de retour :*

— *Registre CC du 6809 :*

Le bit C du registre CC est mis à zéro si tout s'est passé correctement, sinon il est forcé à un, comme d'habitude.

— *Registre PRSTA :* ($2043)

Ce registre retourne le code de l'opération réalisée, ou celui de l'erreur :

- code n° 4 : ouvert pour écriture en parallèle
- code n° 8 : périphérique non prêt
- code n° $10 : fermé

B. *APPLICATION*

PROGRAMME N° XXIX

Ce programme ne fonctionne pas sur nano-réseau.

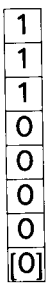
1. Description :

Il s'agit donc de réaliser une copie graphique d'écran, sur imprimante à impact PR90-080. Rappelons qu'il est nécessaire de programmer en langage-machine, pour réaliser une copie graphique d'écran sur « PR90-080 », car ni le BASIC 128, ni le BASIC Microsoft 1.0 ne le permettent par l'instruction SCREENPRINT, réservée à d'autres types d'imprimante.

Étudions tout d'abord la manière dont fonctionne cette imprimante, en mode graphique :

Elle possède 7 aiguilles verticales, et ne peut donc imprimer que 7 bits verticaux de l'écran à la fois (passés par "B"). Le 8e bit passé par l'accumulateur B, à l'imprimante, n'est pas inscrit sur le papier mais doit être mis à 1, soit 128 en décimal.

Pour envoyer à l'imprimante l'octet vertical suivant :



il faut placer dans "B" la valeur décimale : 1+2+4+0+0+0+0+128. Le bit n° 7 vertical (entre crochets), ou 8e bit, sera à passer la fois suivante.

Bien sûr, en mode-caractère, il suffirait d'envoyer le code ASCII de la lettre à imprimer, l'imprimante faisant le reste du travail. Le matriçage de l'imprimante PR90-080 est de 6 bits horizontaux sur 7 bits verticaux, en mode-caractère. Cette imprimante peut écrire sur 40 ou 80 colonnes, c'est-à-dire sur (6bits*40) 240 colonnes de bits, ou au maximum sur (6bits*80) 480 colonnes de bits.

En mode graphique, les 320 points de l'écran seront bien sûr tous recopiés et occuperont 320 colonnes de bits, soit les deux tiers de la largeur du papier qui fait 480 colonnes de bits en tout.

La revue THEOPHILE (voir BIBLIOGRAPHIE) propose dans son numéro 9 d'avril-mai 85, une copie d'écran en doubles hauteur et largeur, mais... la largeur est tournée de 90 degrés sur la hauteur ! (astucieux). Il n'y avait pas d'autre solution à cette réalisation particulière.

Votre imprimante (voir sa notice) vous offre, par des adressages de codes, différentes possibilités d'écriture. A vous de les tester.

2. Réalisation :

a. TO7(70) et TO9 :

```
            602B    RSOPC   EQU     $602B
            E812    RSCOH   EQU     $E812

7D00                        ORG     &32000
7D00 34     7F      P1      PSHS    U,Y,X,DP,D,CC
7D02 C6     40      R1      LDB     #$40     OUVERTURE// ECRIT.
7D04 F7     602B            STB     RSOPC
7D07 BD     E812            JSR     RSCOH
7D0A 1025   008E    R2      LBCS    D11
7D0E B6     E7C3    R3      LDA     $E7C3    MEMOIRE-CARACTERE
7D11 8A     01              ORA     #1
7D13 B7     E7C3            STA     $E7C3
7D16 C6     08      R4      LDB     #8       .CODE COPIE GRAPH.
7D18 F7     602B            STB     RSOPC    .IMP.PR90-080+MISE
7D1B BD     E812            JSR     RSCOH    .EN ECRITURE
7D1E 1025   007A            LBCS    D11
7D22 CE     7D00            LDU     #&32000 POSIT. PILE U
7D25 8E     3FFF    R7      LDX     #$4000-1 DEBUT PROGRAMME
7D28 86     28      D1      LDA     #&40
7D2A 36     02      P2      PSHU    A
7D2C 30     01      D2      LEAX    1,X
7D2E 86     08              LDA     #8
7D30 36     02      P3      PSHU    A
7D32 86     07      D3      LDA     #7
7D34 36     02      P4      PSHU    A
7D36 36     04      P5      PSHU    B
7D38 E6     00      D4      LDB     0,X
7D3A A6     42      P6      LDA     2,U
7D3C 54             D5      LSRB
7D3D 4A                     DECA
7D3E 26     FC              BNE     D5
7D40 37     04      P7      PULU    B
7D42 56                     RORB
7D43 6A     40      P8      DEC     0,U
7D45 27     07              BEQ     D6
```

```
7D47 36   04       P9     PSHU   B
7D49 30   88 28             LEAX   &40,X
7D4C 20   EA               BRA    D4
7D4E 54             D6     LSRB
7D4F CA   80               ORB    #$80
7D51 7D   7DAF             TST    FLAG
7D54 26   37               BNE    D9
7D56 BD   E812     D7      JSR    RSCOH   IMPRESSION
7D59 25   41               BCS    D11
7D5B 30   89 FF10          LEAX   -(6*&40),X
7D5F 37   02       P10     PULU   A
7D61 6A   40       P11     DEC    0,U
7D63 27   02               BEQ    P12
7D65 20   CB               BRA    D3
7D67 37   02       P12     PULU   A
7D69 6A   40       P13     DEC    0,U
7D6B 27   02               BEQ    P14
7D6D 20   BD               BRA    D2
7D6F 37   02       P14     PULU   A
7D71 C6   0A               LDB    #$0A    AVANCE LIGN/PAPIER
7D73 BD   E812             JSR    RSCOH
7D76 25   24               BCS    D11
7D78 8C   5EC7             CMPX   #$5F3F-(&40*3)
7D7B 27   14               BEQ    FIN
7D7D 30   89 00F0          LEAX   (6*&40),X

7D81 8C   5E9F             CMPX   #$5F3F-(&40*4)
7D84 27   02               BEQ    D8
7D86 20   A0               BRA    D1
7D88 7C   7DAF     D8      INC    FLAG
7D8B 20   9B               BRA    D1
7D8D C4   8F       D9      ANDB   #$8F
7D8F 20   C5               BRA    D7
7D91 CC   080A     FIN     LDD    #$80A   3 COPIES/FEUILLE
7D94 BD   E812     D10     JSR    RSCOH
7D97 25   03               BCS    D11
7D99 4A                    DECA
7D9A 26   F8               BNE    D10
7D9C 7F   7DAF     D11     CLR    FLAG    R.A.Z. FLAG
7D9F C6   0F               LDB    #$0F    MIS.MOD.CARACT.IMP
7DA1 BD   E812             JSR    RSCOH
7DA4 C6   10               LDB    #$10    FERMETURE
7DA6 F7   602B             STB    RSOPC
7DA9 BD   E812             JSR    RSCOH
7DAC 35   7F       P15     PULS   CC,D,DP,X,Y,U
7DAE 3F                    SWI
7DAF 00            FLAG    FCB    00
          0000             END

00000 Total Errors
```

b. MO5

```
                2042    PROPC   EQU     $2042
                0024    RSCOH   EQU     $24
                2077    GRCODE  EQU     $2077

7D00                            ORG     &32000
7D00 34   7F            P1      PSHS    U,Y,X,DP,D,CC
7D02 C6   04            R1      LDB     #4      OUVERTURE// ECRIT.
7D04 F7   2042                  STB     PROPC
7D07 3F   24                    CALL    RSCOH
7D09 1025 009A          R2      LBCS    D11
7D0D B6   A7C0          R3      LDA     $A7C0   MEMOIRE-CARACTERE
7D10 8A   01                    ORA     #1
7D12 B7   A7C0                  STA     $A7C0
7D15 C6   08            R4      LDB     #8      .CODE COPIE GRAPH.
7D17 F7   2077                  STB     GRCODE  .IMP.PR90-080
7D1A C6   02            R5      LDB     #2      COPIE GRAPHIQUE
7D1C F7   2042                  STB     PROPC
7D1F 3F   24                    CALL    RSCOH
7D21 1025 0082                  LBCS    D11
7D25 C6   01            R6      LDB     #1      MISE EN ECRITURE
7D27 F7   2042                  STB     PROPC
7D2A 3F   24                    CALL    RSCOH
7D2C 1025 0077                  LBCS    D11
7D30 CE   7D00                  LDU     #&32000 POSIT. PILE U
7D33 8E   FFFF          R7      LDX     #$0000-1 DEBUT PROGRAMME
7D36 86   28            D1      LDA     #&40
7D38 36   02            P2      PSHU    A
7D3A 30   01            D2      LEAX    1,X
7D3C 86   08                    LDA     #8
7D3E 36   02            P3      PSHU    A
7D40 86   07            D3      LDA     #7
7D42 36   02            P4      PSHU    A
7D44 36   04            P5      PSHU    B
7D46 E6   00            D4      LDB     0,X
7D48 A6   42            P6      LDA     2,U
7D4A 54                 D5      LSRB
7D4B 4A                         DECA
7D4C 26   FC                    BNE     D5
7D4E 37   04            P7      PULU    B
7D50 56                         RORB
7D51 6A   40            P8      DEC     0,U
7D53 27   07                    BEQ     D6
7D55 36   04            P9      PSHU    B
7D57 30   88 28                LEAX    &40,X
7D5A 20   EA                    BRA     D4
7D5C 54                 D6      LSRB
7D5D CA   80                    ORB     #$80
7D5F 7D   7DBA                  TST     FLAG
7D62 26   35                    BNE     D9
7D64 3F   24            D7      CALL    RSCOH   IMPRESSION
7D66 25   3F                    BCS     D11
7D68 30   89 FF10               LEAX    -(6*&40),X
7D6C 37   02            P10     PULU    A
```

```
7D6E 6A   40        P11     DEC     0,U
7D70 27   02                BEQ     P12
7D72 20   CC                BRA     D3
7D74 37   02        P12     PULU    A
7D76 6A   40        P13     DEC     0,U
7D78 27   02                BEQ     P14
7D7A 20   BE                BRA     D2

7D7C 37   02        P14     PULU    A
7D7E C6   0A                LDB     #$0A      AVANCE LIGN/PAPIER
7D80 3F   24                CALL    RSCOH
7D82 25   23                BCS     D11
7D84 8C   1EC7              CMPX    #$1F3F-(&40*3)
7D87 27   14                BEQ     FIN
7D89 30   89 00F0           LEAX    (6*&40),X
7D8D 8C   1E9F              CMPX    #$1F3F-(&40*4)
7D90 27   02                BEQ     D8
7D92 20   A2                BRA     D1
7D94 7C   7DBA      D8      INC     FLAG
7D97 20   9D                BRA     D1
7D99 C4   8F        D9      ANDB    #$8F
7D9B 20   C7                BRA     D7
7D9D CC   080A      FIN     LDD     #$80A     3 COPIES/FEUILLE
7DA0 3F   24        D10     CALL    RSCOH
7DA2 25   03                BCS     D11
7DA4 4A                     DECA
7DA5 26   F9                BNE     D10
7DA7 7F   7DBA      D11     CLR     FLAG      R.A.Z. FLAG
7DAA C6   0F                LDB     #$0F      MIS.MOD.CARACT.IMP
7DAC 3F   24                CALL    RSCOH
7DAE C6   10                LDB     #$10      FERMETURE
7DB0 F7   2042              STB     PROPC
7DB3 3F   24                CALL    RSCOH
7DB5 35   7F        P15     PULS    CC,D,DP,X,Y,U
7DB7 BD   B000              STOP
7DBA 00           FLAG      FCB     00
          0000              END

00000 Total Errors
```

3. Explications :

Ce programme est plus complexe que les précédents, et nécessite un développement d'explications assez long.

Pour bien se repérer dans le programme, en plus des étiquettes D(x), nous avons disposé des points de repère : P(x) devant les instructions manipulant les piles, et R(x) devant d'autres instructions.

P1 :

Nous sauvegardons les registres du processeur, comme à l'accoutumée.

R1 :

Ouverture en parallèle pour une écriture. C'est de cette manière qu'il faut travailler avec l'imprimante.

R2 :

Si ''C'' = 1, LBCS branche en D11 vers la fin du programme, car une erreur s'est produite.

R3 :

Mise en mémoire-forme (ou caractère).

R4 :

Envoi du code spécifique de mise en mode graphique à l'imprimante PR90-080 (TO7(70) et TO9), ou mise de celui-ci dans GRCODE (MO5). Sur TO7(70) et TO9, mise en écriture par la même occasion, avec branchement en fin de programme en cas d'erreur (LBCS D11) lors de l'appel à la routine RSCOH. A noter qu'un code d'imprimante s'adresse directement à celle-ci, mais qu'un ordre à la routine doit passer par le registre RSOPC (TO7(70) et TO9) ou PROPC (MO5).

R5 :

Sur MO5, mise en copie graphique. Après appel de la routine RSCOH, on retrouve un branchement en fin de programme en cas d'erreur, par LBCS D11.

R6 :

Sur MO5, mise en écriture.

R7 :

C'est ici que le programme débute réellement. Après avoir mis en place la pile U, on charge l'adresse-1 du 1er octet horizontal de l'écran, dans ''X''.

D1 et P2 :

Première sauvegarde proprement-dite du programme. On empile ''A'' qui contient la valeur &40. Cette valeur correspond à 40 octets horizontaux, à l'écran.

Les explications qui suivent se réfèrent au premier passage dans chaque boucle du programme :

D2

Déplacement de "X" vers l'adresse du 1er octet horizontal d'écran à traiter.

P3 :

Deuxième sauvegarde de "A" dans la pile ; il contient ici la valeur 8, correspondant aux 8 bits d'un octet horizontal.

D3 et P4 :

Sauvegarde de "A" = 7 dans la pile. Ceci correspond à 7 bits verticaux à imprimer. Rappelons qu'il est possible d'empiler plusieurs fois "A", avec des valeurs différentes, et de restituer ensuite chaque donnée entrée par "A", une à une, dans l'ordre inverse de leur entrée dans la pile.

P5 :

Sauvegarde de "B" qui a pour l'instant une valeur quelconque (x).

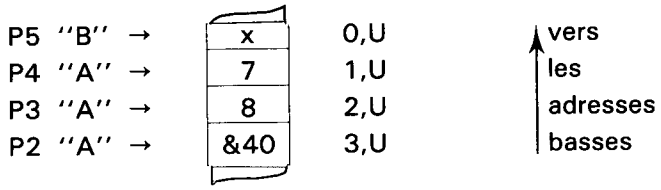
D4 :

Mise du premier octet horizontal d'écran à traiter, dans "B".

P6 :

On charge "A" avec la valeur 8, entrée dans la pile en P3.

Ici il est nécessaire de dessiner la pile U, pour voir où nous en sommes.



D5 :

DÉCALAGE LOGIQUE A DROITE de "B" (LSRB = LOGICAL SHIFT RIGHT B). Examinons un exemple de ce type de décalage :

"C" = 0 ; "B" = | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Avec LSRB, on obtient : | 0 | 1 | 0 | 1 | 0 | 1 | 0- | 1 | → "C"

0 → | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

"C" devient égal à 1 (bit le moins significatif de "B", entrant dans "C").

Le bit le plus significatif est toujours mis à zéro avec UN DÉCALAGE LOGIQUE.

1er N.B :

Il faut signaler qu'en arithmétique non signée, un décalage à droite réalise une division entière par deux. Un décalage à gauche réalise par contre une multiplication par deux (ici 0 prend la place du bit de poids le plus faible, et le bit de poids le plus fort est mis dans "C". Dans ce cas, si "C" devient égal à 1, 8 bits ne suffiront plus à la réalisation de l'opération). Le décalage logique à gauche de "B" se dit LSLB (LOGICAL SHIFT LEFT B). On peut également décaler "A" (LSRA ou LSLA), ou une valeur en mémoire.

Il existe aussi des instructions de décalage arithmétique (ASR ou ASL).

Pour travailler sur 16 bits, on se sert de "D". Il nous faut donc parler ici d'un deuxième type de décalage : le DÉCALAGE CIRCULAIRE.

Le décalage circulaire de "B" à droite s'appelle RORB, et à gauche : ROLB. Il en est de même pour "A" ou pour une valeur en mémoire. RORB signifie : ROTATE RIGHT B, et ROLB : ROTATE LEFT B.

Lors d'un décalage circulaire à droite de "B", la retenue "C" occupe le bit le plus significatif de "B", et le bit le moins significatif de cet accumulateur est mis dans "C". L'opération est symétrique à gauche.

Pour diviser ou multiplier "D" par deux, il faut utiliser les deux types de décalage conjointement.

Exemple :

Soit "D" = $ 0142, à diviser par deux.

"C" = 0

	A								B							
	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0
HEXA	0				1				4				2			

En premier : LSRA. "A" = 01, et devient égal à zéro. "C" devient égal à 1.

En second : RORB. "B" $42, et devient égal à $A1. "C" devient égal à zéro.

Donc $142/2 = $A1.

Autre exemple :

Soit "D" = $0181, à multiplier par deux.

"C" = 0

En premier : "B" = $81. Avec LSLB, il devient égal à 02, et "C" = 1.

En second : "A" = 01. Avec ROLA, il devient égal à 03, et "C" = 0.

Donc $0181 *2 = $0302

Cet aparté terminé, revenons au programme. Il n'est pas question ici de multiplication ou de division, mais de placer les bits à envoyer à l'imprimante. Les exemples qui précèdent n'ont été développés que pour bien comprendre les décalages, et leurs effets.

Après donc, en D5, le décalage logique à droite de "B" (1er octet horizontal d'écran), on décrémente "A" (= 8), et tant que ce dernier n'est pas égal à zéro, on retourne faire un décalage de "B" vers la droite en D5 (BNE D5). Cela fera 8 décalages logiques à droite de "B", et celui-ci deviendra donc égal à zéro. "C" CONTIENDRA ALORS NOTRE PREMIER BIT A IMPRIMER. A noter que lorsqu'après DECA, "A" égale 0, le bit "Z" de "CC" se positionne à 1 et que BNE ne fonctionne plus. Par l'usage d'un compteur à décrémentation nous évitons ici, contrairement à nos habitudes, une instruction de comparaison qui alourdirait le programme.

P7 :

Nous sortons dans "B" la valeur quelconque (x), sauvegardée en P5. Par une rotation circulaire à droite (RORB) nous mettons le premier bit à imprimer (contenu dans "C") à la place du bit n° 7 de "B".

Schéma de "B" à cet instant :

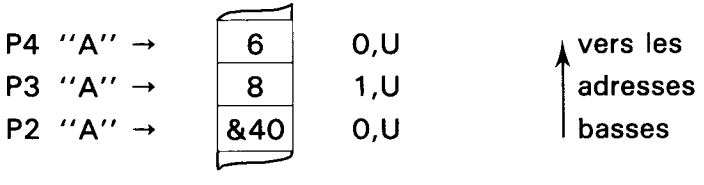
x1	x	x	x	x	x	x	x

où x1 est le 1er bit à imprimer (venant de "C").

P8 :

Nous décrémentons la valeur 7 (mise en P4 dans la pile U). Cette valeur est bien en 0,U , puisque la valeur quelconque x est sortie dans "B".

Schéma de la pile U à cet instant :



Il n'est pas question ici de sortir la valeur mise dans la pile en P4 par "A", mais de la décrémenter directement dans la pile U.

2e N.B. :

Il faut distinguer :

— DEC x,U : cette instruction permet la décrémentation d'une valeur dans la pile, sans enlever la donnée qui s'y trouve, ni déplacer le pointeur de la pile. On peut réaliser une opération d'incrémentation de la même manière, par l'instruction INC x,U. x est un offset quelconque.

— LEAU x,U : cette instruction ne fait que déplacer la position du pointeur de pile, sans toucher aux données qui s'y trouvent. x est un offset quelconque.

— PULU "R" : cette instruction sort la dernière donnée empilée, dans le registre "R", même lorsque celle-ci a été entrée par un autre registre. Le pointeur de la pile est incrémenté de 1 ou 2 octets, selon la longueur de la donnée sortie.

— LDA ou STA x,U : chargement ou stockage d'une valeur (par le registre cité : "A" en l'occurrence) en x,U. x est un offset quelconque.

Revenons au programme :

Nous irons en D6 lorsque la donnée entrée par "A", en P4, sera égale à zéro dans la pile, soit après la mise en place de 7 bits verticaux à imprimer, dans "B".

P9 :

Sauvegarde de "B", dont la valeur reprend sa place dans la pile. Cette valeur n'est maintenant plus quelconque, mais contient le premier bit à imprimer. Nous partons chercher les suivants en D4, un à un, en avançant "X" de 40 adresses d'écran à chaque fois.

D6 :

Lorsque "B" a ses 7 bits à imprimer, voici son schéma :

x 7	x 6	x 5	x 4	x 3	x 2	x 1	x

Seul x ne nous intéresse pas. Un nouvel LSRB le fera sortir. Avec le masquage que réalise ORB #$80, on forcera à un le bit de poids le plus fort de l'octet (dans "B") à imprimer (voir mode de fonctionnement de l'imprimante). Ce bit est en effet, dans tous les cas, égal à zéro après LSRB.

Schéma de "B" à cet instant :

1	x7	x6	x5	x4	x3	x2	x1

ou plutôt :

x1
x2
x3
x4
x5
x6
x7
1

On teste ensuite une case-mémoire définie, comme contenant zéro, en fin de programme (FLAG). L'instruction TST suivie de "A" ou "B" (adressage inhérent), ou d'une adresse-mémoire dans le champ-opérande (TST M), permet de positionner les indicateurs du registre d'état. Si le FLAG est à zéro, "Z" = 1 et BNE ne branche pas le programme en D9. Lorsqu'on met le FLAG à 1, le programme va en D9.

A ce sujet, il faut rappeler que l'imprimante doit recevoir les 7 premiers bits verticaux, puis les suivants sur la même ligne horizontale et ce, jusqu'à la 320e colonne. Il faudra ensuite lui envoyer les 7 bits verticaux situés à gauche de l'écran, sous les 7 premiers, et ainsi de suite. Nous nous apercevons, après un rapide calcul, que 200 lignes divisées par 7 bits font 28,57 retours de chariot, et ceci n'est pas possible. Il faudra donc faire 29 retours de chariot, ce qui correspond à 203 lignes. Nous allons pour cela imaginer 3 lignes supplémentaires, et mettre leurs bits à zéro par un masque (ANDB #$8F) en D9. Le programme ira donc en D9 pour la dernière ligne de septets verticaux d'écran à imprimer (FLAG sera positionné à 1). Il eût été possible également d'inventer 3 lignes blanches en mettant les 120 octets (40 octets * 3 lignes) des adresses qui suivent la mémoire-écran, à zéro. En effet les 192 octets situés juste après la mémoire-écran représentent une partie libre, car le boîtier qui contient la mémoire écran lui réserve 8 K., alors que 8 000 octets seulement sont nécessaires à sa gestion.

D7 :

Envoi à l'imprimante du premier septet vertical d'écran. Tout ceci est bien fastidieux, mais en langage-machine cela va tellement vite !

On recule ensuite de 6 fois 40 octets pour se remettre au niveau de la 1re adresse.

P10 :

On sort dans ''A'' la valeur mise en P4, et devenue égale à zéro. La décrémentation suivante portera donc, en 0,U , sur la valeur 8, mise en P3 par ''A''. Cette valeur correspond aux 8 bits horizontaux d'un octet d'écran à traiter. Lorsque cette valeur, mise en P3, sera égale zéro, il faudra aller en P12 car les 8 premiers septets verticaux de l'écran auront été envoyés au buffer* de l'imprimante.

P12 :

On sort dans ''A'' la valeur mise en P3, et devenue égale à zéro. La pile U ne contient plus alors que la valeur &40 mise par ''A'' en P2. C'est cette valeur qu'il faut décrémenter (DEC 0,U). En effet une ligne à l'écran contient 40 octets. Quand tous ceux-ci auront été traités, nous irons en P14.

P14 :

On vide la pile de sa dernière valeur mise en P2, et devenue égale à zéro, en la sortant dans ''A''. La pile est maintenant vierge.

Après chaque envoi d'une ligne complète de septets verticaux d'écran, il faut vider le buffer en imprimant ses données, et retourner au début de la ligne suivante (code $0A envoyé à l'imprimante, voir sa notice). Si nous sommes à 3 lignes de la dernière adresse de l'écran, la copie est terminée (n'oublions pas qu'il y a 3 lignes supplémentaires « imaginées » après celles de l'écran). Dans le cas contraire, il importe de faire pointer ''X'' sur la nouvelle adresse-1 à traiter (LEAX (6*&40),X), puisqu'en D2 nous repasserons par LEAX 1,X. S'il ne reste plus qu'une ligne de septets verticaux d'écran à traiter (CMPX #$5F3F-(&40*4) ou CMPX #$1F3F-(&40*4)), il faut mettre le FLAG à 1 (D8).

Les instructions en D9 ont déjà été expliquées.

FIN :

LDD #$80A a pour effet de mettre la valeur 8 dans ''A'', et $0A dans ''B''. En appelant la routine RSCOH on fait avancer le papier (le buffer étant vide), par le code $0A mis dans ''B'', 8 fois (DECA et BNE D10).

D11 :

Étiquette de branchement final en cas d'erreur.

* Buffer : zone-tampon ou d'attente, dont la capacité pour la « PR90-080 » est de 480 septets verticaux, soit pour 6 aiguilles horizontales, en mode-caractère et en 80 colonnes, une ligne complète d'écran.

Nettoyage de la case FLAG, pour des copies successives le cas échéant.

Il est nécessaire de remettre l'imprimante en mode-caractère (code $0F, voir sa notice), car sinon vous ne pourriez pas faire un listing de programme sans éteindre et rallumer celle-ci, après la copie d'écran.

Fermeture de l'imprimante.

P15 :

Dépilement habituel.

FLAG :

Définition d'une case-mémoire (FLAG) mise à zéro, pour la copie de la dernière ligne de septets verticaux d'écran.

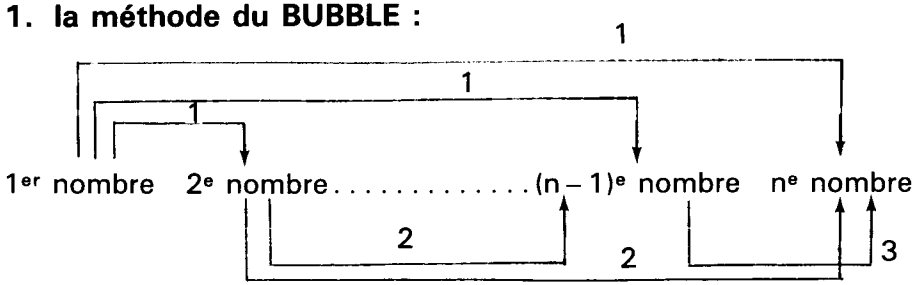
15. Le tri en langage-machine

A. DESCRIPTION

Parmi les nombreuses applications du langage-machine, retenons celle du tri numérique pour sa rapidité d'exécution par rapport au basic.

Il existe des ouvrages très complets sur les tris, et de multiples méthodes. Citons :

1. la méthode du BUBBLE :

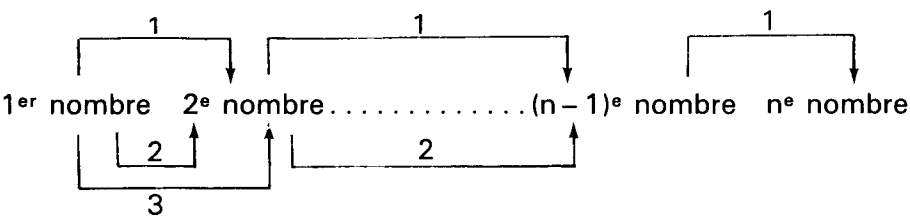


nombre de passages : n – 1, où n est le nombre de données à trier. 1, 2 et 3 représentent les numéros des passages successifs ci-dessus.

nombre de comparaisons = n (n – 1)/2

Lorsque deux nombres ne sont pas à leur place, on les permute.

2. La méthode du RIPPLE :



Les nombres de passages et de comparaisons sont identiques à ceux de la méthode précédente ; cependant il y a possibilité ici de posi-

tionner un flag qui sera égal à 1 en cas de permutation, et à zéro dans le cas contraire. A la fin d'un passage, si le flag est resté à zéro, le tri est terminé. Cette méthode est donc meilleure que la première.

3. La méthode de SHELL-METZNER :

Elle est très connue, complexe, mais c'est la plus performante.

Il faut tout d'abord compter les n données à trier. L'intervalle (entre une donnée et celle qui lui sera comparée) sera égal à n@2 lors du premier passage, puis toujours divisé par 2 à chaque passage suivant (division entière), jusqu'à ce qu'il soit égal à 1.

Lorsqu'il y a permutation entre 2 nombres, un flag est positionné. Ce flag ne devient actif, c'est-à-dire n'influence la suite des opérations, qu'à la condition que la différence entre le plus petit numéro de position des 2 nombres permutés, lui-même considéré comme nombre, et l'intervalle, soit supérieure à zéro.

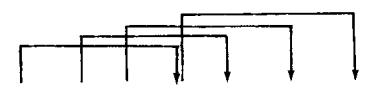
Exemple :

Soit à trier, pour les ranger dans un ordre décroissant, les nombres suivants :

35 6 2 4 33 7 49

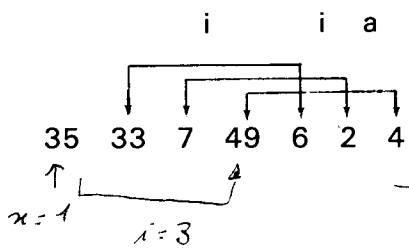
1er passage :

- n = 7
- I (premier intervalle) = 7@2, soit 3
- C (nombre de comparaisons pour le passage déterminé) = n – I, soit 4



Comparaison des nbs 2 à 2
Permutation si nb2 > nb1
Si permutation (soit 2,3,4) → flag

Flags de permutation :



1° 2-3 = -1 i = 7@2 = 3
3° 3-3 = 0
4° 4-3 = 1 : OK flag actif
→ Cas 4 : x = 1
35 > 49 ? non → on permute

- x = condition d'un flag pour devenir actif
- i = flag inactif ($x \leqslant 0$)
- a = flag actif ($x > 0$)

a) Entre 33 et 6 : le numéro de position de 33 est 2, moins l'intervalle I égal à 3 → x = − 1, donc flag inactif.

b) Entre 7 et 2 : le numéro de position de 7 est 3, moins l'intervalle I égal à 3 → x = 0, donc flag inactif.

c) Entre 49 et 4 : le numéro de position de 49 est 4, moins l'intervalle I égal à 3 → x = + 1, donc flag actif. X = 1 va servir lors de l'ultime comparaison due au flag actif, de même que l'intervalle I = 3.

Dernière permutation due au flag actif :

On compare la x^{e} donnée (ici x = 1) avec la x + I^{e} donnée (donc ici la 4^{e}), et on permute si nécessaire :

35 33 7 49 6 2 4

Résultat du 1er passage :

49 33 7 35 6 2 4

2^{e} passage :

- n = 7
- I = 3@2, soit 1
- C (nombre de comparaisons pour le passage déterminé) = n − I, soit 6

49 33 7 35 6 2 4

Flags de permutation :

a a

49 33 35 7 6 4 2

a) Entre 35 et 7 : le numéro de position de 35 est 3, moins l'intervalle I égal à 1 → x = 2, donc flag actif.

b) Entre 4 et 2 : le numéro de position de 4 est 6, moins l'intervalle I égal à 1 → x = 5, donc flag actif.

Dernières permutations dues aux flags actifs :

49 33 35 7 6 4 2

a) *x = 2 :*

On compare la x^e donnée (x = 2) avec la x + I^e donnée (donc la 3^e), et on permute si nécessaire.

b) *x = 5 :*

On compare la x^e donnée (x = 5) avec la x + I^e donnée (donc la 6^e) et on permute si nécessaire.

A noter que cette dernière comparaison est inutile, mais sera effectuée quand même.

Résultat final :

49 35 33 7 6 4 2

Vous pourrez trouver, dans le n° 7 des mois de DÉC.-JANV. 84 de la revue THEOPHILE, un exemple de tri numérique par cette méthode de SHELL-METZNER, en langage-machine (voir bibliographie).

B. APPLICATIONS

PROGRAMME N° XXX

1. Description :

Nous allons exécuter un tri par RIPPLE en langage-machine, avec flag de terminaison, et contrôle du résultat. Les données à classer seront pour l'instant peu nombreuses et placées, dans le désordre, en fin de programme dans une table. On les stockera après le tri dans un ordre décroissant, à partir de l'adresse représentée par l'étiquette DON. Ce programme faisant moins d'une PAGE (255 octets) nous pouvons nous mettre en PAGE DIRECTE ($79).

Lorsque plusieurs PAGES sont nécessaires à un programme, il faut remettre à jour la page directe tous les 255 octets. Le signe "<" n'est pas obligatoire lorsqu'il précède une adresse située dans la page (sauf si cette adresse est définie par une étiquette dans le programme, comme ici : COMP ou FLAG) ; s'il est malgré tout employé, cela ne changera rien au code-objet. Par contre, si vous forcez l'adressage direct en dehors de la page, l'assembleur vous signalera une erreur. Pour passer en adressage étendu, dans une autre page, il est préférable de l'indiquer par ">" car la relecture du programme sera plus facile. Néanmoins, si vous l'omettez, l'assembleur saura reconnaître cet adressage étendu. Enfin, un adressage étendu forcé pour une adresse située dans la page, donnera à l'assemblage un code-objet avec un octet de plus (comme si vous n'étiez pas en page directe). Dans tous les cas vous devez programmer vos adresses sur 2 octets ; c'est l'assembleur qui choisira le code-objet final.

Dans le programme qui suit, nous gagnerons quelques octets par cette mise en page directe. Vous pourrez le constater, en lisant les codes-objet générés après assemblage. Seuls les adressages étendus sont bien sûr transformables en adressages directs, et non les adressages immédiats.

2. Réalisation :

a. TO7(70) et TO9 :

```
7900                        ORG     $7900
7900 34   7F                PSHS    U,Y,X,DP,D,CC
          79                SETDP   $79
7902 86   79                LDA     #$79
7904 1F   8B                TFR     A,DP
```

```
7906 108E 796B          LDY    #DON
790A 8E   7956          LDX    #TABLE
790D EC   81      DEBUT LDD    ,X++
790F 1083 0004          CMPD   #4
7913 27   04            BEQ    SUITE
7915 ED   A1            STD    ,Y++
7917 20   F4            BRA    DEBUT
7919 8E   797B    SUITE LDX    #DON+16
791C 9F   68            STX    <COMP
791E 8E   796B    D1    LDX    #DON
7921 EC   81      D2    LDD    ,X++
7923 10A3 81            CMPD   ,X++
7926 22   02            BHI    SUITE2
7928 8D   1C            BSR    SWAP
792A 9C   68      SUITE2 CMPX  <COMP
792C 27   04            BEQ    D3
792E 30   1E            LEAX   -2,X
7930 20   EF            BRA    D2
7932 0D   6A      D3    TST    <FLAG
7934 27   1D            BEQ    FIN
7936 0F   6A            CLR    <FLAG
7938 DE   68            LDU    <COMP
793A 33   5E            LEAU   -2,U
793C 1183 796D          CMPU   #DON+2
7940 27   11            BEQ    FIN
7942 DF   68            STU    <COMP
7944 20   D8            BRA    D1
7946 0C   6A      SWAP  INC    <FLAG
7948 10AE 83            LDY    ,--X
794B 10AF 83            STY    ,--X
794E 30   02            LEAX   2,X
7950 ED   81            STD    ,X++
7952 39                 RTS
7953 35   7F      FIN   PULS   CC,D,DP,X,Y,U
7955 3F                 SWI
7956 000C 028F    TABLE FDB    12,655,34543,1467,3567
795A 86EF 05BB
795E 0DEF
7960 FB41 B26E          FDB    64321,45678,46,$4
7964 002E 0004
7968 0000         COMP  FDB    0000
796A 00           FLAG  FCB    00
          796B    DON   EQU    *
          0000          END

00000 Total Errors
```

b. MO5 :

Même chose que pour les TO7(70) et TO9, mais SWI est à remplacer par STOP.

3. Explications :

La sauvegarde systématique des registres du processeur, dans la pile-système, a préservé la page de base de l'assembleur ou du basic, dans laquelle nous devrons retourner en fin de programme.

On se met donc en PAGE DIRECTE, ce qui nécessite 2 opérations. La première, SETDP $79, est une directive signifiant à l'assembleur que nous allons travailler en PAGE DIRECTE $79 [$7900-$79FF]. Celui-ci, connaissant nos intentions, générera le code-objet adéquat à l'assemblage. Il faut ensuite mettre le numéro de cette PAGE dans le registre DP, car en « machine » ce sera la seule opération connue du processeur puisque SETDP $79 n'est pas traduite en binaire. On utilise TFR qui permet de TRANSFERER la valeur d'un registre dans un autre de MEME CAPACITE (ici TFR A,DP). EXG A,DP est une instruction d'échange et pouvait aussi convenir, mais à quoi bon mettre la valeur de ''DP'' dans ''A'' ?

On place dans ''Y'' l'adresse à partir de laquelle nous allons mettre les données numériques, dans le désordre (DON).

On charge l'adresse de TABLE dans ''X'', et le registre D avec les nombres à trier sur 16 bits (par exemple : 00 + $ 12 = $0012), implantés par FDB.

Tant que ''D'' n'atteint pas la valeur $4, signifiant E.O.T., nous déposons ces données à l'adresse de ''Y'', autoincrémenté de 2 octets à la fois (16 bits), c'est-à-dire de DON à DON + 16 (pour nos 8 données, chacune sur 16 bits).

''X'' nous permettra de placer l'adresse de DON + 16 dans la case COMP (comparaison).

D1 :

On charge dans ''D'' le premier nombre du tableau (par l'intermédiaire de ''X'') et on le compare au suivant (CMPD ,X++). S'il n'y a pas de permutation on va en SUITE2, sinon on se branche au SWAP.

SUITE2 :

CMPX < COMP : est-on à la fin du passage ? Si oui allons en D3, sinon replaçons ''X'' pour continuer le tri en D2.

D3 :

Si le flag est resté à zéro pour un passage (TST FLAG), le tri est terminé ; sinon il faut le remettre à zéro et décrémenter la case COMP, car la dernière donnée est à sa place définitive. Lorsque ''U'' sera égal à l'adresse de DON + 2, le tri sera de toute façon terminé.

SWAP :

On incrémente le FLAG*, et la 2e donnée est mise à l'adresse de la 1re par l'intermédiaire de ''Y''. On replace ''X'' pour stocker la 1re donnée, située dans ''D'', à l'adresse de la seconde. Au retour, ''X'' pointe de nouveau sur la donnée suivante.

DON est une étiquette définie à la position courante du ''PC'', en fin de programme. COMP et FLAG représentent respectivement des cases-mémoire de 16 et 8 bits, mises à zéro.

En comparant les valeurs à partir des adresses de TABLE et de DON, on peut vérifier le bon fonctionnement du tri dans le moniteur-binaire, ou par des PEEK en basic, après exécution du programme.

** Rappel :*

Les instructions CLR, INC et DEC peuvent fonctionner avec une position-mémoire de la zone-opérande.

PROGRAMME N° XXXI

1. Description :

Ce programme ne fonctionne sur MO5 qu'avec le D.O.S. (pour la fonction USR).

Vous avez compris le principe du tri précédent, il vous faut maintenant pouvoir l'utiliser dans vos programmes, en basic. Nous restons ici fidèles à la ligne de conduite de l'ouvrage : améliorer nos performances en basic, langage de base de notre programmation.

Le problème sera donc d'appeler le langage-machine, au moment précis où nous en aurons besoin. Les nombres à trier seront rangés dans un tableau de variables indicées, en basic, dimensionné par DIM A%(I), où I représentera le nombre de données numériques à trier.

Où seront rangées ces données en basic, à quelle adresse pourrons-nous aller les chercher pour les trier en langage-machine ?

C'est ici qu'intervient VARPTR (A%(I)), où I est l'indice de la variable A%. Selon leur type (entier, réel, de double précision ou chaîne), les variables ont leurs valeurs rangées dans 2, 4, 8 octets ou plus. Les variables qui nous intéressent, dans le programme qui suit, sont entières. Leurs valeurs sont stockées sur 2 octets, le premier représentant les bits de poids fort et le second les bits de poids faible, pour chacune d'entr'elles (nous connaissons déjà ce type de représentation), en mode-complément à 2, c'est-à-dire signé. Ces variables peuvent représenter des valeurs allant de [– 32768 à + 32767]. Dans notre programme, A%(1) contiendra d'ailleurs une valeur maximum : + 32767.

Les adresses des variables étant des nombres entiers, elles sont aussi codées sous ce mode ; une valeur négative représentant une adresse supérieure à 32767, il faudra lui ajouter 65536 pour obtenir la valeur réelle de cette adresse. Ce type de stockage permet de représenter toute valeur d'adresse d'une mémoire adressée sur 16 bits.

Essayons donc, pour connaître l'adresse d'une variable (c'est-à-dire de son octet de poids fort), d'utiliser la fonction VARPTR (< nom de la variable >).

Vous pouvez à ce sujet consulter votre manuel de référence du basic.

Exemple :

```
10 DIM A%(24)
20 FOR I= 0 TO 24
30 X=VARPTR(A%(I))
40 IF X<0 THEN X=X+65536
50 ? I "=" X
60 NEXT I:END
```

L'adresse de la variable, donnée par VARPTR, est un nombre entier et décimal compris entre [– 32768 et + 32767]. On ajoute "+ 65536" aux nombres négatifs pour obtenir la valeur d'adresse leur correspondant.

Il n'est cependant pas possible de lancer un tri-machine par VARPTR(A%(I)). En effet, cette fonction ne permet pas le passage d'un argument à un sous-programme « machine ». A signaler par ailleurs que la moindre modification dans le programme-basic (sauf en BASIC 128) entraîne un changement d'adresse des variables. En BASIC 128, les variables sont rangées en fin de mémoire (dernière banque), et non après le programme, des adresses de poids fort vers les adresses de poids faible.

Pour passer un argument à un sous-programme écrit en langage-machine, il faut utiliser la fonction USR. Cependant, avec celle-ci, vous n'aurez dans L'ACCUMULATEUR FLOTTANT* (qui sert aux calculs faits par l'interpréteur), c'est-à-dire dans FACMO et FACLO pour les « entiers », que le seul nombre passé comme argument (exemple : le 1er élément du tableau pour A%(0)).

Il faut donc, pour avoir l'adresse du 1er élément et trier l'ensemble d'un tableau, utiliser la combinaison de USR et de VARPTR, comme ceci :

Z = USR0 (VARPTR(A%(0)))

Z est ici factice, mais peut éventuellement servir lors du retour d'une donnée, d'un programme-machine vers le basic.

Alors, qu'obtiendrons-nous ?

Si nous avons défini l'adresse de notre programme-machine par DEFUSR, la fonction USR, sous la forme citée plus haut, lancera ce programme-machine, remplaçant ainsi l'EXEC. Nous récupérerons alors, dans le registre X, l'adresse de FAC (et non de FACM0). L'adresse de notre première variable (nombre entier) se trouvera cependant dans FACMO et FACLO ; "X" pointant sur FAC, il faudra

* Le livre de référence du BASIC 128 ne fait plus état de cette accumulateur, mais ces explications restent valables.

le déplacer de 2 octets pour le faire pointer sur FACMO, et y récupérer l'adresse de cette variable (dans ce même registre X, par exemple : LDX 2,X). A ce sujet, relisez votre manuel de référence du basic.

Il suffira de connaître le nombre d'éléments à trier (qu'il est possible d'ailleurs de placer dans le programme-machine par un POKE judicieux, à l'aide d'une variable dont peut se servir l'utilisateur) pour pouvoir lancer le programme.

Essayez, par exemple, pour un programme-machine débutant en &32000 :

DEFUSR0 = 32000 ou DEFUSR = 32000
Z = USR0 (0) Z = USR (0)

Ceci équivaut à un EXEC 32000.

Le <numéro> derrière DEFUSR ou USR est facultatif lorsqu'il s'agit d'un premier appel (0 par défaut), mais bien sûr pas l'argument entre parenthèses (même s'il est fictif comme ici).

2. Réalisation (TO7(70), T09 et MO5) :

a. Partie-basic :

```
10 CLS
20 DIM A%(24):DEFUSR0=&H7F00
30 FOR I=0 TO 24
40 READ A%(I)
50 NEXT I
60 LOADM "PROG.BIN",0
70 Z=USR0(VARPTR(A%(0)))' OU Z=USR0(VARP
TR(A%(24))) EN BASIC 128.
80 FOR I=0 TO 24
90 PRINT"A%(";:PRINTUSING"##";I;:PRINT")
=";A%(I)
100 NEXT
110 DATA 4314,32767,13,1,9,28,42,2456,42
3,2141,3044,42,4567,214,6,87,12345,543,8
,45,345,9876,4567,3,235
120 END
```

b. Partie-machine :

```
                    *EN ASSEMBLEUR:
                    *FAIRE A0 OU AC:PROG.BIN (COMMAND.EDIT.)

7F00                        ORG     $7F00
7F00 34   7F                PSHS    U,Y,X,DP,D,CC
          7F                SETDP   $7F
```

```
7F02 86    7F              LDA     #$7F
7F04 1F    8B              TFR     A,DP
7F06 AE    02              LDX     2,X
7F08 9F    4E              STX     <ADVI
7F0A 30    88 32           LEAX    50,X
7F0D 9F    50              STX     <ADF
7F0F 9E    4E      DEBUT   LDX     <ADVI
7F11 EC    81      D1      LDD     ,X++
7F13 10A3 81               CMPD    ,X++
7F16 22    02              BHI     SUITE
7F18 8D    22              BSR     SWAP
7F1A 9C    50      SUITE   CMPX    <ADF
7F1C 27    04              BEQ     D3
7F1E 30    1E              LEAX    -2,X
7F20 20    EF              BRA     D1
7F22 DE    52      D3      LDU     <FLAG
7F24 1183 0000             CMPU    #0
7F28 27    22              BEQ     FIN
7F2A CE    0000            LDU     #0
7F2D DF    52              STU     <FLAG
7F2F DE    50              LDU     <ADF
7F31 33    5E              LEAU    -2,U
7F33 1193 50               CMPU    <ADVI+2
7F36 27    14              BEQ     FIN
7F38 DF    50              STU     <ADF
7F3A 20    D3              BRA     DEBUT
7F3C CE    0001    SWAP    LDU     #1
7F3F DF    52              STU     <FLAG
7F41 10AE 83               LDY     ,--X
7F44 10AF 83               STY     ,--X
7F47 30    02              LEAX    2,X
7F49 ED    81              STD     ,X++
7F4B 39                    RTS
7F4C 35    FF      FIN     PULS    CC,D,DP,X,Y,U,PC
7F4E 0000          ADVI    FDB     0000
7F50 0000          ADF     FDB     0000
7F52 0000          FLAG    FDB     0000
           0000            END

00000 Total Errors
```

3. Explications :

a. Partie-basic :

On évite sur TO7(70) et TO9 de faire un CLEAR, &H7EFF en ligne 20, car cette adresse est trop proche du programme-basic quand on a implanté le D.O.S.-système, néanmoins le programme fonctionne très bien sans CLEAR.

On dimensionne le tableau, et on positionne l'adresse de départ (&H7F00) avec DEFUSR0.

En ligne 40, on place dans A%(I) les valeurs stockées de manière désordonnée en DATA.

Si vous possédez la cartouche-assembleur, vous pouvez sauvegarder la partie-machine sous le nom par exemple de PROG.BIN . Pour ce faire, procédez de la manière suivante :

— zone commande-éditeur :

- faire A0 : PROG.BIN pour une disquette sur lecteur n° 0.
- faire AC : PROG.BIN pour une cassette.

En basic, il faut implanter ce programme vous-même, comme vous en avez l'habitude.

On appelle en ligne 70 le programme de tri.

En ligne 80, on lit les variables indicées : leurs valeurs sont bien triées, dans un ordre décroissant.

b. Partie-machine :

La page de programmation est ici définie en $7F car il n'est plus nécessaire, du moins sur TO7 dont la mémoire-utilisateur se termine en $7FFF, de réserver de la place en fin de programme pour une table plus ou moins longue.

Rien de nouveau dans l'ensemble par rapport au programme précédent, si ce n'est qu'on met dans "X" l'adresse de la 1re variable à trier, par l'instruction : LDX 2,X ("X" pointe sur FACM0 – 2 octets avant cette instruction).

On sauvegarde l'adresse de début des variables à trier dans ADVI (adresse de la variable initiale).

On déplace "X" de 50 octets, et on stocke l'adresse de fin de tableau + 2 (pour 25 variables) dans ADF (adresse de fin).

Donc, par rapport au programme précédent, la valeur en ADVI remplace DON, et ADF : COMP ; l'étiquette DEBUT est ici mise à la place de D1. Le reste est identique.

Le programme se termine par PULS CC, D, DP, X, Y, U, PC. En incluant le "PC" dans le dépilage, on retourne au basic, sans RTS. Ce programme, même préparé sur assembleur, est en effet conçu pour fonctionner avec le basic.

16. Notes sur les interruptions

A. DESCRIPTION

Une interruption est un avertissement donné au processeur, permettant de faire cesser la tâche qu'il est en train d'exécuter, pour lui en faire réaliser une autre. Cet avertissement, pour une interruption dite matérielle, correspond à un « courant » ou impulsion envoyé sur une des broches du C.P.U., reliées au bus de commande. Ce même bus de commande est par ailleurs connecté aux broches d'extension ou « nez de carte », à l'arrière de votre appareil (tableau n° 1).

Les interruptions sont utiles en informatique lorsqu'il y a nécessité d'arrêt immédiat d'un programme, ou lorsqu'à un signal donné le processeur doit effectuer un travail quelconque. Lors de leur activation, ce dernier sauvegarde plus ou moins l'état de ses registres internes, ce qui lui permet de reprendre, après réalisation du sous-programme d'interruption, le cours normal de son activité.

Toute séquence d'interruption doit se terminer par RTI (RETURN FROM INTERRUPT), instruction examinant le dépilage à effectuer en retour (bit "E" de "CC"). Si "E" = 1, tous les registres* sont à dépiler (dans le même ordre que lors d'un PULS), sinon seuls "CC", puis "PC" le sont.

Le système se sert de ces interruptions pour son propre compte. Vous pourrez faire de même, soit en pratiquant le HARDWARE, soit en détournant l'une d'elles par votre programmation.

Le détournement d'une interruption vers un sous-programme personnel, se pratique au niveau d'un vecteur de 16 bits, situé en R.A.M. . Il faut pour ce faire remplacer l'adresse du programme-moniteur, mise dans ce vecteur (lorsque la routine d'interruption est vectorisée en R.A.M.), par l'adresse de votre propre programme d'interruption. D'autres opérations peuvent être également nécessaires pour détourner une interruption. Nous verrons un exemple de détournement logiciel, dans l'application qui va suivre. Soyez cependant très méfiant,

* du processeur bien sûr. Dans le texte, cette explication sera désormais implicite.

car certaines routines du moniteur sont interruptibles. Vos programmes ne doivent pas modifier leurs paramètres.

Exemples :

La répétition du clavier ou le clignotement du curseur sont gérés par le système d'interruption du 6809.

Un CNT-C, en basic, réalise une interruption de la tâche du processeur.

Les interruptions sont plus ou moins prioritaires les unes par rapport aux autres.

Il est des interruptions dites masquables et d'autres qui ne le sont pas, c'est-à-dire que certaines d'entr'elles peuvent être inhibées par le programmeur, et d'autres non. C'est en forçant à 1 le bit du registre d'état lui correspondant, que l'on peut masquer une interruption.

On distingue les interruptions logicielles ou SOFTWARE, des interruptions matérielles ou HARDWARE. Il faut y ajouter le RESET, et les instructions de synchronisation sur une interruption. Pour faciliter la compréhension de leur étude, elles seront décrites dans l'ordre qui suit :

1. L'interruption manuelle RESET.
2. Les interruptions LOGICIELLES.
3. Les interruptions MATÉRIELLES pures.
4. Les instructions de SYNCHRONISATION sur une interruption.

1. Le RESET :

On différencie l'initialisation à froid, du RESET ou initialisation à chaud.

Le RESET est une interruption matérielle et manuelle.

Lorsque vous appuyez sur le bouton du RESET, le processeur se branche à l'adresse contenue en ($FFFE-$FFFF). Il n'y a pas ici d'empilement des registres, et à la fin de cette interruption vous revenez automatiquement au point de départ du moniteur-système, avec affichage du menu (TO7(70) et TO9), ou de la page d'en-tête (MO5).

2. Les interruptions LOGICIELLES :

L'instruction SWI (SOFTWARE INTERRUPT) n'a pas le même effet sur TO7(70) et TO9, que sur MO5.

a. TO7(70) et TO9 :

Lors d'une interruption de type SWI, le processeur met le bit E du

registre CC à 1, pour signaler que l'ensemble des registres est sauvegardé, puis il charge l'adresse contenue en ($FFFA-$FFFB) dans le "PC". SWI est prioritaire sur IRQ et FIRQ qu'elle masque en mettant les bits "I" et "F" de "CC" à 1. Cette interruption logicielle se termine par RTI (retour d'interruption), comme les autres interruptions, et vous rend la main lorsque vous travaillez en assembleur. Dans ce même langage, l'avantage de SWI est de permettre la correction pas à pas de vos programmes. Il est possible de programmer plusieurs break tout au long d'un programme, par la technique des points d'arrêt. En tapant "K" dans le moniteur-binaire, vous positionnez un break à l'adresse que vous indiquez derrière ce "K". Vous pouvez utiliser 8 points d'arrêt en tout (de 0 à 7) au cours d'un programme. L'interruption SWI est considérée comme le 9e. C'est pour cela que lorsqu'un programme est terminé, vous voyez apparaître le numéro 8 avant le "BRK" du break. Pour supprimer un point d'arrêt et reprendre le cours normal d'un programme, il faut donner, dans le moniteur-binaire, la commande "Y" (YANK) suivie du numéro du point d'arrêt à enlever. Si le numéro du point d'arrêt n'est pas mentionné, tous les break sont annulés. Cette technique exige une certaine prudence pour ne pas mettre des points d'arrêt au milieu des codes d'une même instruction, et « planter » le programme.

b. MO5 :

Sur MO5, SWI permet d'accéder à toutes les routines du moniteur ; elle n'en reste pas moins une interruption logicielle, avec sa priorité, sa sauvegarde de tous les registres, et le chargement du "PC" à l'adresse contenue en ($FFFA-$FFFB).

Le vecteur en R.A.M. ou pointeur sur SWI est en ($205E-$205F) et s'apelle SWIPT où "PT" signifie pointeur. Pour rendre la main au programmeur, on utilise en assembleur la directive STOP (= JSR $B000). Les points d'arrêt existent comme sur TO7(70) et TO9, STOP étant lui-même un dernier point d'arrêt.

- Il existe encore SWI2 et SWI3.

Elles ne sont pas utilisables sur MO5.

Sur TO7(70) et TO9, elles agissent comme SWI, mais sans masquer IRQ et FIRQ. Leurs adresses sont contenues en :

SWI2 : ($FFF4-$FFF5)
SWI3 : ($FFF2-$FFF3)

Le vecteur en R.A.M. ou pointeur sur SWI (appelé SWI1) est en ($602F-$6030). Pour SWI2 il passe en $6800, et pour SWI3 en $7000.

3. Les interruptions MATÉRIELLES :

Elles sont représentées par IRQ, FIRQ et NMI.

a. IRQ (INTERRUPT REQUEST) :

Cette interruption sauvegarde l'ensemble des registres, en mettant le bit "E" de "CC" à 1.

Le flag "I" de "CC" est mis à 1 lors de l'activation de IRQ, interdisant toute autre interruption du même type, mais pas FIRQ car l'indicateur "F" reste à zéro. IRQ est donc une interruption masquable par la mise à 1 du bit du registre d'état lui correspondant. On se sert, pour ce faire, de l'instruction ORCC suivie de l'opérande #$10 (%00010000). Pour remettre le bit "I" du registre de code-condition à zéro, on utilisera l'instruction ANDCC suivie de #$EF (%11101111). Ces deux instructions permettent l'opération booléenne bit à bit du registre CC avec le masque contenu dans la zone-opérande.

L'adresse de IRQ est contenue en ($FFF8-$FFF9).

IRQ sert au clignotement du curseur et au clavier.

Les vecteurs en R.A.M. de IRQ sont IRQPT et TIMEPT. Le premier comme le second contiennent l'adresse d'un branchement vers le moniteur, à exécuter en fin de traitement de l'interruption. TIMEPT sert à la répétition du clavier (TO7(70) et MO5 seulement) et au clignotement du curseur. IRQ est une interruption utilisable en software pur ; nous étudierons dans le programme qui va suivre comment la détourner.

— *Sur TO7(70) et TO9 :*
- IRQPT = ($6021-$6022)
- TIMEPT = ($6027-$6028)

— *Sur MO5 :*
- IRQPT = ($2064-$2065)
- TIMEPT = ($2061-$2062)

b. FIRQ (FAST INTERRUPT REQUEST) :

Cette interruption ne provoque que l'empilement de "PC" et "CC", le programmeur devant sauvegarder au préalable les registres qu'il risque de modifier. De ce fait, elle est plus rapide que IRQ. Lors de son activation, le bit "E" de "CC" reste à zéro, mais les bits "I" et

"F" de "CC" sont mis à 1, interdisant toute autre interruption de type IRQ ou FIRQ. FIRQ est donc masquable par le bit "F" du registre d'état.

L'adresse de cette interruption est située en ($FFF6-$FFF7).

Les vecteurs en R.A.M. de FIRQ sont :

Sur TO7(70) et TO9 :

- FIRQPT = ($6023-$6024).

Sur MO5 :

- FIRQPT = ($2067-$2068).

c. NMI (NON MASQUABLE INTERRUPT) :

Il s'agit ici d'une interruption non masquable et prioritaire. L'ensemble des registres est sauvegardé lors de son activation, avec mise à 1 des bits "E", "I" et "F" de "CC", interdisant donc les interruptions de type IRQ et FIRQ.

L'adresse de NMI est contenue en ($FFFC-$FFFD).

Cette interruption n'est pas utilisable sur TO7, ni sur TO9 ou MO5.

Sur TO7-70, son vecteur en R.A.M. (NMIPT) est en ($6025-$6026).

- Le tableau n° 6 vous donne les vecteurs en R.A.M. des différentes interruptions.

4. Les instructions de synchronisation sur une interruption :

Elles sont représentées par CWAI et SYNC qui sont des instructions en assembleur, comme SWI. Ces instructions permettent la mise en concordance d'un sous-programme avec une tâche principale, par le biais des interruptions matérielles précédemment décrites.

a. CWAI (CLEAR CC BITS AND WAIT FOR INTERRUPT) :

CWAI est une instruction fonctionnant en mode immédiat, et réalisant un ET LOGIQUE entre l'opérande qui lui est associé, et le registre d'état. Cette instruction permet une sauvegarde de l'ensemble des registres, et la mise du processeur en attente d'interruption de type IRQ ou FIRQ, voire NMI. Si le bit "F" de "CC" est à 1, avant CWAI, FIRQ ne peut être prise en compte. Le bit "E" de "CC" est mis à 1 lors de la rencontre de CWAI et, même lorsque l'interruption qui arrive est de type FIRQ, l'ensemble des registres est sauvegardé.

b. SYNC (SYNCHRONIZE TO EXTERNAL EVENT) :

Cette instruction permet également la mise du processeur en attente

d'une interruption matérielle. FIRQ et IRQ peuvent être inhibées, mais pas NMI. Dans le cas d'une inhibition, ou d'une impulsion hardware trop courte, le processeur reprend sa tâche immédiatement après SYNC, sinon il y a réalisation de la séquence d'interruption. Aucune opération logique n'est ici effectuée sur le registre d'état, pas plus que de sauvegarde des registres.

Ces instructions de synchronisation programmée sur une interruption matérielle sont utiles pour gérer des événements externes, et CWAI peut vous aider à synchroniser un sous-programme d'interruption avec un corps de programme. Cette dernière oblige à passer par le sous-programme d'interruption, alors que SYNC peut être utilisée comme simple synchronisation de programme, avec un événement extérieur. Elles n'ont été citées que dans le but d'être assez complet sur ce chapitre.

En résumé :

Cet ouvrage n'étant pas destiné au ''hard.'', voyons ce que vous pouvez faire de manière simple :

- Vous pouvez gérer manuellement un RESET.
- Vous pouvez programmer une interruption de type SWI et la détourner. SWI2 et SWI3 ne sont programmables et ne pourront être détournées que sur TO7(70) et TO9.
- Vous pouvez détourner IRQ, utilisée par le système pour le clignotement du curseur et la répétition des touches. C'est ce que nous allons maintenant étudier.

B. APPLICATION

PROGRAMME N° XXXII

1. Description :

Ce programme pourrait s'intituler : Écriture en musique. Nous allons en effet écrire 10 fois le mot INTERRUPTION par la routine PUTCH et, à chaque interruption du TIMER, gérant le clignotement du curseur et la répétition des touches (sauf sur TO9 pour cette dernière), nous détournerons provisoirement la séquence d'exception pour jouer rapidement quelques notes de musique, au RYTHME du curseur. L'interruption-TIMER est de type IRQ, appelée aussi interruption-utilisateur. Sur TO9, existe une interruption ACIA pour le clavier.

Pour détourner l'interruption-TIMER, il faut 4 conditions :

a. Sur TO7(70) et TO9 :

— mettre l'adresse du sous-programme personnel d'interruption dans le registre TIMEPT de 16 bits.
— mettre le bit n° 5 de STATUS à 1 pour signaler au système que l'interruption est détournée.
— en fin de sous-programme d'exception, faire un JMP vers KBINH pour valider l'interruption (RTI y est programmée), mais ne pas faire un JSR !
— restituer les registres altérés pour le détournement, pour reprendre la main en mode-programmation.

b. Sur MO5 :

— Les mêmes conditions que ci-dessus sont nécessaires, à la différence près qu'au lieu de modifier STATUS, vous devez mettre une valeur non nulle et quelconque en $2063 (sémaphore de IRQ), et qu'au lieu de faire un JMP vers KBINH, vous devez faire un simple RTI.

Par ailleurs, les registres DP et S doivent être sauvegardés, ce qui est habituel pour nous.

2. Réalisation :

a. TO7(70) et TO9 :

```
6031    TEMPO   EQU     $6031
6033    DUREE   EQU     $6033
6035    TIMBRE  EQU     $6035
6036    OCTAVE  EQU     $6036
```

```
                7D00    ADINTH EQU     &32000
                6027    TIMEPT EQU     $6027
                6019    STATUS EQU     $6019
                E803    PUTCH  EQU     $E803
                E844    INTERH EQU     $E844
                E81E    NOTEH  EQU     $E81E
                E830    KBINH  EQU     $E830

                       *CORPS DE PROGRAMME

7918                           ORG     &31000
7918 34 7F                     PSHS    U,Y,X,DP,D,CC
791A CC 0001    MUS    LDD     #1
791D FD 6031           STD     TEMPO
7920 CC 0005           LDD     #5
7923 FD 6033           STD     DUREE
7926 86 01             LDA     #1
7928 B7 6035           STA     TIMBRE
792B CC 0002           LDD     #2
792E FD 6036           STD     OCTAVE
7931 CC 7D00    DEBUT  LDD     #ADINTH
7934 FD 6027           STD     TIMEPT
7937 B6 6019           LDA     STATUS
793A 8A 20             ORA     #$20
793C B7 6019           STA     STATUS
793F 4F                CLRA
7940 8E 7D0D           LDX     #TABLE
7943 3C 00     ATTEN   CWAI    #0
7945 E6 80     ECRIT   LDB     ,X+
7947 C1 04             CMPB    #4
7949 27 05             BEQ     RECRIT
794B BD E803           JSR     PUTCH
794E 20 F3             BRA     ATTEN
7950 4C        RECRIT  INCA
7951 81 0A             CMPA    #10
7953 27 05             BEQ     FIN
7955 8E 7D0D           LDX     #TABLE
7958 20 EB             BRA     ECRIT
795A CC E844    FIN    LDD     #INTERH
795D FD 6027           STD     TIMEPT
7960 4F                CLRA
7961 B7 6019           STA     STATUS
7964 35 7F             PULS    CC,D,DP,X,Y,U
7966 3F                SWI

                       *SOUS-PROGRAMME D'INTERRUPTION

7D00                           ORG     ADINTH
7D00 C6 31             LDB     #$31
7D02 BD E81E    GAM    JSR     NOTEH
7D05 5C                INCB
7D06 C1 3D             CMPB    #$3D
7D08 26 F8             BNE     GAM
7D0A 7E E830    RETI   JMP     KBINH
```

```
7D0D 2E 2E 2E 2E  TABLE  FCC     /.........interruption./
7D11 2E 2E 2E 2E
7D15 2E 69 6E 74
7D19 65 72 72 75
7D1D 70 74 69 6F
7D21 6E 2E
7D23 2E 2E 2E 2E          FCC     /................../
7D27 2E 2E 2E 2E
7D2B 2E 2E 2E 2E
7D2F 2E 2E 2E 2E
7D33 2E
7D34 04                   FCB     $4
          0000            END

00000 Total Errors
```

b. MO5 :

```
          203A    TEMPO   EQU     $203A
          203C    DUREE   EQU     $203C
          203D    TIMBRE  EQU     $203D
          203E    OCTAVE  EQU     $203E
          7D00    ADINTH  EQU     &32000
          2061    TIMEPT  EQU     $2061
          2063    SEMIRQ  EQU     $2063
          0002    PUTCH   EQU     2
          F0AD    INTERH  EQU     $F0AD
          001E    NOTEH   EQU     $1E

                  *CORPS DE PROGRAMME

7918                      ORG     &31000
7918 34   7F              PSHS    U,Y,X,DP,D,CC
791A C6   01      MUS     LDB     #1
791C F7   203A            STB     TEMPO
791F C6   05              LDB     #5
7921 F7   203C            STB     DUREE
7924 C6   01              LDB     #1
7926 F7   203D            STB     TIMBRE
7929 CC   0002            LDD     #2
792C FD   203E            STD     OCTAVE
792F CC   7D00    DEBUT   LDD     #ADINTH
7932 FD   2061            STD     TIMEPT
7935 86   01              LDA     #1
7937 B7   2063            STA     SEMIRQ
793A 4F                   CLRA
793B 8E   7D0A            LDX     #TABLE
793E 3C   00      ATTEN   CWAI    #0
7940 E6   80      ECRIT   LDB     ,X+
7942 C1   04              CMPB    #4
7944 27   04              BEQ     RECRIT
7946 3F   02              CALL    PUTCH
```

```
7948 20   F4              BRA     ATTEN
794A 4C           RECRIT  INCA
794B 81   0A              CMPA    #10
794D 27   05              BEQ     FIN
794F 8E   7D0A            LDX     #TABLE
7952 20   EC              BRA     ECRIT
7954 CC   F0AD    FIN     LDD     #INTERH
7957 FD   2061            STD     TIMEPT
795A 4F                   CLRA
795B B7   2063            STA     SEMIRQ
795E 35   7F              PULS    CC,D,DP,X,Y,U
7960 BD   B000            STOP

                  *SOUS-PROGRAMME D'INTERRUPTION

7D00                      ORG     ADINTH
7D00 C6   01              LDB     #1
7D02 3F   1E      GAM     CALL    NOTEH
7D04 5C                   INCB
7D05 C1   0D              CMPB    #$D
7D07 26   F9              BNE     GAM
7D09 3B           RETI    RTI
7D0A 2E 2E 2E 2E  TABLE   FCC     /.........interruption./
7D0E 2E 2E 2E 2E
7D12 2E 69 6E 74
7D16 65 72 72 75
7D1A 70 74 69 6F
7D1E 6E 2E
7D20 2E 2E 2E 2E          FCC     /................../
7D24 2E 2E 2E 2E
7D28 2E 2E 2E 2E
7D2C 2E 2E 2E 2E
7D30 2E
7D31 04                   FCB     $4
          0000            END

00000 Total Errors
```

3. Explications :

Nous avons défini nos étiquettes, en assembleur, avant le programme. Parmi celles-ci, ''ADINTH'' représente l'adresse de notre sous-programme d'interruption, et ''INTERH'' l'adresse à remettre dans TIMEPT au retour. Sur MO5, ''SEMIRQ'' est le sémaphore de IRQ, et remplace STATUS. Le nom des trois étiquettes entre guillemets est purement inventé. Sur MO5, KBINH n'existe pas.

Le programme débute en &31000 pour des raisons de compatibilité entre les divers appareils.

MUS représente la préparation des registres musicaux. On essaie d'obtenir une musique assez rapide pour ne pas ralentir trop l'écriture.

DEBUT est le détournement d'interruption : chargement de l'adresse de notre sous-programme dans TIMEPT, mise à 1 du bit n° 5 de STATUS par l'opération booléenne adéquate sur TO7(70) et TO9, ou mise à 1 par exemple du sémaphore de IRQ sur MO5.

CLRA vient ensuite et représente la préparation du compteur d'écriture. On charge l'adresse de la TABLE d'écriture dans ''X''.

ATTEN est la mise en attente d'interruption du processeur, par CWAI #0. Ceci ralentit le programme, mais a pour avantage de synchroniser notre sous-programme avec le programme proprement dit, pour CHAQUE INTERRUPTION DU TIMER. L'opérande immédiat de CWAI étant 0, le registre CC ne sera pas affecté par le ET LOGIQUE de cette instruction.

ECRIT représente le chargement de ''B'' avec les caractères à écrire, jusque E.O.T. = $4. En fin de table, on se branche vers RECRIT pour préparer une nouvelle écriture. Celle-ci sera exécutée dix fois en tout (compteur ''A'').

A la FIN, il y a remise en état de TIMEPT, de STATUS ou SEMIRQ, selon votre appareil, suivie de SWI pour les uns et de STOP pour les autres.

Le sous-programme est implanté par l'étiquette ADINTH. Il est séparé du corps de programme par un titre. L'astérisque fait passer ce dernier en mode commentaire (REM en basic). On pouvait aussi mettre ORG &32000, et ADINTH devant la première instruction : LDB #$31 (ou #1), qui représente la première note à jouer (DO).

GAM figure l'adresse de la gamme, à jouer par incrémentation de ''B'' jusque $3C ou $C (SI).

RETI : retour d'interruption, vers KBINH sur TO7(70) et TO9, par RTI sur MO5.

Ce programme réalise donc 10 fois l'inscription du mot INTERRUPTION, encadrée par des points, avec un décalage du mot à chaque ligne si on le désire. L'écriture n'est pas rapide en raison de l'attente d'interruption par CWAI, et parce qu'une note (et a fortiori plusieurs notes à jouer) demande au moins le temps qu'on l'entende. CWAI cependant nous synchronise à merveille sur le TIMER ; quant à la musique, vous pouvez la jouer note par note au cours d'un programme, par une sous-routine qui peut être ou non gérée par une interruption.

17. Commutation de mémoire sur TO7-70 et TO9

Les applications de ce chapitre ne sont pas conçues pour fonctionner en BASIC 128, où elles n'ont guère d'utilité.

A. DESCRIPTION

Les possesseurs de TO7 ou MO5 ne perdront pas leur temps en lisant ce chapitre, qui complète l'étude de la translatabilité des programmes. Si vous travaillez sur un ancien TO7-70, ou sur TO9, vous pourrez avoir besoin des notions qui suivent, pour la programmation en BASIC Microsoft 1.0. Le MO5 dispose quant à lui d'une cartouche de 64 K-octets, réservée au téléchargement sur Nano-réseau.

Sur TO9, ne pas confondre cette commutation avec celle permettant l'accès aux divers programmes d'application intégrés en R.O.M. (routine COM$: voir P9).

L'extension-mémoire de 64 K, sur TO9, n'est pas considérée par les applications comme de la mémoire de programme, mais comme un disque virtuel. Le texte qui suit ne tient donc pas compte de cette extension-mémoire, qui est accédée directement par le contrôleur de disque (voir routine DKCOH, et le tableau de Bank Ram : P9).

Pour la mémoire-utilisateur, le constructeur a prévu des extensions, dont une est déjà dans le TO7-70, et d'autres dans le TO9. Cependant, la mémoire adressable sur 16 bits n'est pas extensible à l'infini ! Seize lignes d'adresses ne peuvent en effet servir qu'à l'adressage de 2^{16} cases-mémoire, soit 65536 octets ou 64 K-octets. Toutes les adresses étant occupées sur TO7-70 et TO9, il a fallu prévoir des COMMUTATIONS.

La R.A.M. de votre appareil se situe dans des boîtiers de 16 K., de $6000 à $DFFF. De $6000 à $9FFF se trouvent 16 K. fixes, et de

$A000 à $DFFF, 16 K. commutables. Lorsque l'on commute 16 K., ceux-ci sont « débranchés », mais conservent leurs données en mémoire. A la place de ces 16 K. « débranchés » du système, on implante 16 K. tout neufs, vierges, dans lesquels on peut de nouveau programmer aux mêmes adresses, soit de $A000 à $DFFF. Pour retrouver les anciennes données, il faut « débrancher » les nouvelles et replacer les autres. Vous comprenez déjà qu'on ne peut pas faire appel au programme « branché » à partir d'un programme « débranché », et vice-versa. Les opérations de « branchement » ou COMMUTATIONS peuvent être faites par la LOGIQUE, donc en mode-programmation. Le programme de commutation, pouvant appeler de un (TO7-70) à plusieurs boîtiers (TO7-70 avec extension, ou TO9), devra nécessairement se trouver en R.A.M. fixe, et pourquoi pas à la fin de celle-ci par exemple, c'est-à-dire juste avant $A000.

Remarques du constructeur : pour des raisons de protection de sélection simultanée de plusieurs banques R.A.M., il a été installé le système d'utilisation suivant. Les 5 bits de données du PIA qui servent à la sélection sont toujours à 0. Pour commuter une banque on n'écrit donc pas dans le registre de données du PIA, mais on change la direction des bits concernés, sachant qu'un bit en entrée génère un 1 et un bit en sortie génère un 0.

Attention : certaines routines du moniteur peuvent modifier temporairement le contenu du PIA, et restaurer plein 0 à la fin.

B. APPLICATIONS

PROGRAMME N° XXXIII

1. Description :

A partir des indications fournies par le constructeur, nous allons donc passer de la banque résidente à la banque commutable n° 1, qui équipe d'office tous les TO7-70. Ceux qui disposent d'une extension ou d'un TO9 pourront s'inspirer du même programme, pour utiliser les divers codes hexadécimaux correspondant aux banques qu'ils désirent commuter, et qui sont fournis ci-après.

Ce programme est destiné au langage-assembleur, il prépare le programme-basic qui suit. Il est implanté en $9FDF, mais peut fonctionner partout ailleurs, avant l'adresse $A000.

2. Réalisation :

a. TO7-70 et TO9 :

```
9FDF                        ORG     $9FDF
9FDF 34   7F       COMMUT   PSHS    U,Y,X,DP,D,CC
9FE1 CE   E7C0              LDU     #$E7C0
9FE4 E6   4B                LDB     11,U
9FE6 C4   FB                ANDB    #$FB
9FE8 E7   4B                STB     11,U
9FEA 8E   9FFA              LDX     #TAB
9FED 86   01                LDA     #1        NO DE BANQUE
9FEF A6   86                LDA     A,X
9FF1 A7   49                STA     9,U
9FF3 CA   04                ORB     #4
9FF5 E7   4B                STB     11,U
9FF7 35   7F                PULS    CC,D,DP,X,Y,U
9FF9 3F                     SWI
                   * NOS DES BANQUES DE 0 a 5

9FFA 0F 17 E7 67   TAB      FCB     $0F,$17,$E7,$67,$A7,$27
9FFE A7 27
           0000             END

00000 Total Errors
```

3. Explications :

COMMUT est donc l'adresse de la sous-routine de commutation. Après un empilement des registres du 6809, systématique, nous char-

geons l'adresse $E7C0 dans ''U''. Cette adresse est celle de début des entrées-sorties ; c'est en effet au niveau des P.I.A. que l'on peut accéder à la commutation de la R.A.M. (tableau n° 13). Avec LDB 11,U , nous mettons dans ''B'' la valeur du registre de contrôle (placé en 11,U , donc en $E7CB) du port B du PIA-système 6821. Par ANDB #$FB, et stockage de cette nouvelle valeur dans le registre de contrôle, nous faisons passer ce P.I.A. en mode-direction, de telle manière qu'il autorise la commutation.

On charge dans ''X'' l'adresse de TAB et on prend le 2e code de la table, correspondant à la banque n° 1. Les codes qui suivent, dans la table, permettent la commutation des autres banques éventuelles, et le code 0F qui précède, le retour à la banque numéro 0 d'origine. En changeant la valeur de l'opérande immédiat de ''A'', vous pouvez ainsi commuter, de 0 à 5, les 6 banques lorsque vous les possédez.

Le numéro de la banque est placé en 9,U , soit en $E7C9, registre de données du PORT B du même système P.I.A.-6821. Ceci permet le passage vers la nouvelle banque. Par un ORB #4, et stockage de cette nouvelle valeur en $E7CB, nous remettons le P.I.A.-6821 en mode-donnée.

Le dépilement vient ensuite, comme d'habitude.

Ce programme fonctionne donc avec la cartouche-assembleur, et à l'adresse donnée par la directive ORG, ou à une adresse plus basse que celle-ci.

PROGRAMME N° XXXIV

1. Description :

Ce programme-ci est écrit en basic, avec une partie-machine. Il est en effet nécessaire de pouvoir commuter la ou les banques supplémentaires, en langage-basic Microsoft 1.0.

2. Réalisation (TO7-70 et TO9) :

a. Partie-machine :

```
          6A00     COMMUT EQU    *
6A00 34   7F               PSHS   U,Y,X,DP,D,CC
6A02 CE   E7C0             LDU    #$E7C0
6A05 E6   4B               LDB    11,U
6A07 C4   FB               ANDB   #$FB
6A09 E7   4B               STB    11,U
6A0B A6   03               LDA    3,X
6A0D 30   8D 000A          LEAX   TAB,PCR
6A11 A6   86               LDA    A,X
6A13 A7   49               STA    9,U
6A15 CA   04               ORB    #4
6A17 E7   4B               STB    11,U
6A19 35   FF               PULS   CC,D,DP,X,Y,U,PC
6A1B 0F 17 E7 67  TAB      FCB    $0F,$17,$E7,$67,$A7,$27
6A1F A7 27
          0000             END

00000 Total Errors
```

b. Partie-basic :

```
10 CLEAR,&H9FDE
20 DEFINT X
30 DEFUSR=&H9FDF
40 FOR I=&H9FDF TO &H9FFF
50 READ A$
60 POKE I,VAL("&H"+A$)
70 NEXT I
80 DATA 34,7F,CE,E7,C0,E6,4B,C4,FB,E7,4B
,A6,03,30,8D,00,0A,A6,86,A7,49,CA,04,E7,
4B,35,FF,0F,17,E7,67,A7,27
90 CLS:PRINT "CHOISISSEZ VOTRE BANQUE (0
 OU 1)":X$=INPUT$(1):X=VAL(X$)
100 IF X<0 OR X>1 THEN 90'pas d'erreur
 possible
110 Z=USR(X)
120 PRINT"VALEUR DE &HDFFF EN BANQUE";X;
"="; PEEK(&HDFFF):PRINT"VOULEZ-VOUS CHAN
```

```
GER?(O/N)":A$=INPUT$(1):IF A$="O" THEN I
NPUT"PAR QUELLE VALEUR(0a255)";A:POKE&HD
FFF,A
130 PRINT"ARRET?(O/N)":A$=INPUT$(1):IF A
$="O" THEN STOP ELSE 90
```

3. Explications :

a. Le sous-programme « machine » :

Alors que nous avons l'habitude de débuter nos programmes-assembleur par la directive ORG, nous nous contentons ici de définir l'étiquette COMMUT par la directive EQU, suivie de ''*''. Un sous-programme défini de cette manière, peut néanmoins être appelé par son étiquette.

Notre sous-programme s'implantera donc à la position courante du ''PC''. Il n'a pour intérêt que de fournir des codes-objet pour une utilisation en basic.

Après un empilement maintenant classique, nous positionnons le registre de contrôle, déjà cité, en mode de commutation.

En basic, USR nous permettra de récupérer l'argument (n° de banque), et non plus son adresse, par l'intermédiaire du registre X du 6809, à 3 octets près. En effet, nous emploierons la fonction USR, sans VARPTR, et l'argument sera passé directement à la sous-routine écrite en langage-machine, par l'accumulateur flottant. Cette variable, entière, aura son octet de poids faible dans FACLO, et ''X'' pointera sur FAC, d'où la nécessité d'un déplacement de 3 octets réalisé par LDA 3,X. ''A'' contiendra alors le numéro de banque.

Pour pouvoir implanter ce sous-programme « machine » à n'importe quelle adresse d'un programme-basic, sans réassemblage, nous demandons à ''X'' de prendre l'adresse de TAB en mode translatable, grâce au PC, programmé par PCR*. Nous chargeons ensuite ''A'' avec le numéro de la banque choisie, pour commuter celle-ci.

Exemples de translatabilité par PCR :

LEAU TAB,PCR se traduit en binaire par ''33'' suivi d'un post-octet, puis d'un offset-machine. ''U'' est chargé avec l'adresse de TAB, en mode translatable. LDU #TAB,PCR est proscrit.

* PCR : PROGRAM COUNTER RELATIVE. L'adressage doit être ici, dans tous les cas, étendu ou direct.

LDU TAB,PCR se traduit en binaire par ''EE'' suivi d'un post-octet, puis d'un offset-machine, et réalise le chargement de ''U'' avec la 1re valeur de la table, sur 2 octets, en mode translatable.

Par contre, LDU #TABLE se traduit en binaire par ''CE'' suivi de l'adresse de la table et n'est pas translatable sans réassemblage, en cas de changement d'adresse d'implantation du programme.

b. La partie-basic :

La variable à passer est définie comme étant une variable entière (DEFINT X).

Le programme de commutation, en langage-machine, est implanté à la fin de la R.A.M. fixe, soit en &H9FDF, et protégé par l'instruction CLEAR.

Vous pouvez aussi commuter les extensions (si vous les possédez) en modifiant les valeurs de X, variable-basic.

Les lignes qui suivent la ligne 110 ne sont utiles qu'à titre de test de bonne commutation. Ce test est ici réalisé par un POKE de la dernière adresse de la banque choisie.

Remarques :

Ce système de commutation vous sera utile pour travailler en basic Microsoft 1.0, avec des sous-programmes « machine » implantés dans les diverses banques. Le programme écrit en langage-basic ne pourra pas dépasser l'adresse &H9FDE ; pensez-y au moment où vous réaliserez ce dernier.

La variable-basic, X, pourra être avantageusement remplacée par la variable « BANK ». Lors de la mise au point du programme, vous pourrez ainsi faire à tout moment : PRINT BANK, et connaître le numéro de la banque en service.

Pour sauvegarder ou charger l'ensemble des programmes basic et binaires, il est nécessaire de le faire en plusieurs fois, avec un petit programme préalable de lancement prévoyant la ou les commutations. Le ou les programmes-machine sont à sauvegarder en dernier, et à charger en premier.

18. Le décimal codé binaire

A. DESCRIPTION

Le B.C.D. (décimal codé binaire) est un code identique à celui de l'hexadécimal, mais sans les chiffres A, B, C, D, E, F. Certaines combinaisons binaires, qui ont un sens pour le codage de l'hexadécimal, n'en ont plus en BCD : il s'agit, par quartet, des états binaires correspondant aux chiffres hexadécimaux de $A à $F. Par exemple, %10101110 = $AE n'a aucun sens en BCD, car un quartet ne peut, dans ce code, contenir que les chiffres de 0 à 9. Ainsi donc, pour coder le nombre décimal 10, il faudra inscrire sur 2 quartets : %00010000.

Lors d'une addition en BCD, le processeur effectue d'abord l'opération en binaire, puis examine la demi-retenue H. Si celle-ci est à 1, il ajoute 6 au quartet de poids faible.

Vient ensuite un deuxième ajustement décimal, permettant d'éviter les combinaisons binaires inutilisables. Il consiste à additionner 6 à tout chiffre de quartet supérieur à 9. Après un tel ajustement, si "C" = 1, il est nécessaire de travailler sur 16 bits.

L'addition en BCD n'est réalisable que par des instructions affectant l'indicateur H, comme ADDA, ou ADCA (celle-ci permettant le report de la retenue C sur un octet de poids fort, par l'intermédiaire du registre A), suivies de DAA. Cette dernière signifie DECIMAL ADDITION ADJUST, et ne fonctionne qu'avec l'accumulateur A.

Attention : le processeur ne travaille qu'en binaire. Le BCD est un artifice obtenu par ajustement. Ce code n'a par conséquent pas de correspondance, en valeur, avec les nombres qu'il représente. Par exemple, si on demandait l'ajustement décimal de $10, le processeur ne changerait rien au code %00010000 ; ce serait en effet au programmeur de savoir qu'il considère ce code comme étant celui d'un nombre décimal.

L'ajustement décimal ne permet donc pas d'effectuer une conversion

binaire-décimal. Il n'a de sens qu'après une addition de nombres considérés au départ, par le programmeur, comme étant des nombres décimaux. Par exemple LDA #$CC, suivi de DAA, donne ''A'' = %00110010 et ''C'' = 1 ; or $CC est égal à 204 en décimal, et non à 132.

L'octet ou les octets écrits en BCD son considérés, par le processeur, comme contenant des valeurs binaires, exprimables en hexadécimal.

B. APPLICATION

PROGRAMME N° XXXV

1. Description :

Réalisation et affichage d'un compteur décimal à 4 chiffres. Ce compteur pourra effectuer des calculs jusqu'à 9999, valeur maximum sur 16 bits en BCD.

L'affichage d'un compteur se fait habituellement à une position définie par des coordonnées. Pour simplifier la programmation le compteur sera néanmoins, dans ce programme, affiché à la position courante du curseur, et un nettoyage régulier de la fenêtre permettra sa remise à jour. Ceci aura pour inconvénient mineur de le faire clignoter.

2. Réalisation :

a. TO7(70) et TO9 :

```
                    *COMPTEUR A 4 CHIFFRES SUR TO7(70)-TO9:

          E806      GETCH   EQU     $E806
          E803      PUTCH   EQU     $E803

                    * CORPS DE PROGRAMME

7D00                        ORG     &32000
7D00 34   79                PSHS    U,Y,X,DP,CC
7D02 CE   7D00              LDU     #&32000 POSIT.PILE U
7D05 4F                     CLRA
7D06 5F                     CLRB
7D07 36   06                PSHU    D       COMPTEUR A ZERO
7D09 BD   E806      CLAV    JSR     GETCH   VOIR CLAVIER
7D0C C1   46                CMPB    #$46    TOUCHE F= FIN
7D0E 27   0E                BEQ     FIN
7D10 8D   11                BSR     COMPT   COMPTEUR
7D12 EC   40                LDD     0,U     RESUT.BCD
7D14 1083 9999              CMPD    #$9999  MAX.BCD AUTORISE?
7D18 27   04                BEQ     FIN     OUI->FIN
7D1A 8D   22                BSR     AFFICH  AFFICHAGE
7D1C 20   EB                BRA     CLAV    ON CONTINUE
7D1E 37   06        FIN     PULU    D       RECUPER.RESULT.BCD
7D20 35   79                PULS    CC,DP,X,Y,U
7D22 3F                     SWI

                    * SOUS-PROGRAMMES

7D23 C6   0C        COMPT   LDB     #$0C    RAZ
7D25 BD   E803              JSR     PUTCH
```

```
7D28 C6   07            LDB    #7       BIP COMPTEUR
7D2A BD   E803          JSR    PUTCH

                        *INCREMENTATION LSB COMPTEUR

7D2D A6   41            LDA    1,U      CHARG. LSB
7D2F 1C   FE            ANDCC  #$FE     MISE DE "C" A ZERO
7D31 8B   01            ADDA   #1       "A"= "A"+1
7D33 19                 DAA             AJUST.DECIMAL "A"
7D34 A7   41            STA    1,U      RESULT.LSB

                        *TRAITEMENT RETENUE D'OCTET:MSB

7D36 86   00            LDA    #0       "A"=0:"C"PRESERVE
7D38 A9   40            ADCA   0,U      "A" ="A"+"C"+MSB
7D3A 19                 DAA             AJUST.DECIMAL "A"
7D3B A7   40            STA    0,U      RESULT.MSB
7D3D 39          RET    RTS

                 *AFFICHAGE

7D3E FD   7D6B   AFFICH STD    VAL      STOCKAGE COMPT.BCD
7D41 F6   7D6B          LDB    VAL      M.S.B. COMPTEUR
7D44 7F   7D6D          CLR    FLAG     NETTOYER FLAG
7D47 54          SUITE  LSRB            4DECAL.LOGI.DROITE
7D48 54                 LSRB
7D49 54                 LSRB
7D4A 54                 LSRB
7D4B CB   30            ADDB   #$30      .ASCII
7D4D BD   E803          JSR    PUTCH    .AFFICH 1E CHIFFRE
7D50 F6   7D6B          LDB    VAL
7D53 C4   0F            ANDB   #$0F     4BITS POIDSFORT->0
7D55 CB   30            ADDB   #$30      .ASCII
7D57 BD   E803          JSR    PUTCH    .AFFICH 2E CHIFFRE
7D5A 7D   7D6D          TST    FLAG     .FIN?
7D5D 26   0B            BNE    RET1     .OUI=RETOUR
7D5F F6   7D6C          LDB    VAL+1    .LSB COMPTEUR
7D62 F7   7D6B          STB    VAL      .DANS CASE VAL
7D65 7C   7D6D          INC    FLAG     MISE FLAG A 1
7D68 20   DD            BRA    SUITE    VERS AFFICH. LSB
7D6A 39          RET1   RTS             FIN AFFICHAGE

7D6B 0000        VAL    FDB    0000     VALEUR COMPTEUR
7D6D 00          FLAG   FCB    00
          0000          END

00000 Total Errors
```

b. MO5 :

Les changements suivants sont à opérer :

- GETCH EQU $0A
- PUTCH EQU 2
- Chaque JSR est à remplacer par CALL, et SWI par STOP.

3. Explications :

Compteur :

Incrémentation du LSB, ajustement décimal de celui-ci, report de la retenue éventuelle sur le MSB, et ajustement décimal de ce dernier.

On utilise l'accumulateur D, sauvegardé dans la pile U, comme compteur sur 16 bits. Après avoir chargé "A" avec le LSB du compteur, on effectue une mise à zéro du bit C du registre d'état par le masque ET LOGIQUE ANDCC #$FE. On additionne 1 à l'accumulateur A, seul utilisable ensuite par DAA. On stocke en 1,U le résultat du LSB.

CLRA affectant la CARRY, il faut ensuite utiliser LDA #0 pour vider "A" de son contenu, tout en préservant "C". Cette retenue, quel que soit son état, sera ajoutée au MSB, et le résultat final stocké dans "A" (ADCA 0,U). Après ajustement décimal (DAA), on range le MSB en 0,U.

Retour au corps de programme :

"D" vient d'être incrémenté, en BCD. Lorsqu'il atteindra $9999 (seule valeur connue du processeur mais que nous considérons, nous, comme étant une valeur décimale), le compteur sera bloqué (il pourrait repartir à zéro si on le désirait). En attendant, on continue l'incrémentation du compteur, après son affichage.

Affichage :

L'affichage se réalise de la manière suivante :

1) mise de "D" dans la case-mémoire VAL définie sur 16 bits en fin de programme.
2) chargement du MSB dans "B".
3) nettoyage du FLAG défini en fin de programme.
4) 4 décalages logiques à droite de "B", permettent de récupérer les 4 bits de poids fort du MSB sur son quartet de poids faible. On convertit ce chiffre des unités de mille, en ASCII (ADDB #$30), et on l'affiche.
5) reprise du MSB (VAL) dans "B". Le masque logique ANDB #$0F permet cette fois la mise à 0 de son quartet de poids fort. Affichage du chiffre des centaines, après conversion ASCII.
6) TST FLAG permet de recommencer la précédente opération, pour le LSB. Si FLAG = 1, l'affichage est terminé, sinon on transfère le LSB de VAL + 1 dans VAL et on positionne cette fois le FLAG à 1. On effectue enfin l'affichage des 2 chiffres du LSB.

Ce programme, comme d'autres dans l'ouvrage, comporte beaucoup de remarques dans la zone des commentaires. Celles-ci permettent de réduire le nombre des explications dans le texte, ou leur rappel lors de la lecture de l'application. Cette zone, en temps normal, ne doit cependant contenir que quelques commentaires concis, qui seront des points de repère pour son auteur ou un programmeur averti.

Conclusion

Ce livre n'avait pas la prétention de faire de vous des professionnels du langage-machine, ni d'étudier dans le détail et dans leur subtilité toutes les instructions du 6809. Il s'est contenté de vous faire découvrir certains aspects intéressants de ce langage, et les routines accessibles de votre appareil.

Les programmes qui y ont été présentés ne sont pas les plus performants, ni les plus polyvalents possibles (l'usage d'un compteur à décrémentation eût été parfois meilleur que celui d'un compteur à incrémentation par exemple, de même les branchements comparatifs n'ont-ils pas été employés en mode signé, etc...). Néanmoins ces programmes ont peut-être l'avantage d'être simplifiés, en raison d'un emploi limité du nombre de mnémoniques et d'astuces techniques, ou plus faciles à comprendre grâce à leur étalement.

L'auteur espère avoir atteint le but qu'il s'était fixé, et accepte toute critique ou remarques par l'intermédiaire de son éditeur. Ces observations pourront d'ailleurs être utiles en cas de réédition de l'ouvrage.

Ceux qui ne possèdent pas de cartouche-assembleur auront quelques difficultés à bâtir, et surtout à translater, leurs programmes. Ils disposent cependant d'un certain nombre d'éléments non négligeable pour pouvoir le faire.

Leurs connaissances, comme celles des autres d'ailleurs, pourront être largement complétées par la lecture d'ouvrages plus spécialisés que celui-ci.

BONNE CONTINUATION... EN LANGAGE-MACHINE.