P. GUEULLE

MAITRISEZ VOTRE ZX81

MICRO SYSTEMES

ETSF

Collection
MICRO-SYSTEMES
dirigée par ALAIN TAILLIAR
directeur de la rédaction de la revue MICRO-SYSTEMES

Maîtrisez votre ZX-81

Patrick GUEULLE
Ingénieur EFREI

•

Maîtrisez votre ZX-81

(2e édition)

Diffusion :
ÉDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 PARIS CEDEX 19

OUVRAGES DU MEME AUTEUR :

- Réalisez vos récepteurs en circuits intégrés.
- Interphone, téléphone, montages périphériques.
- Pilotez votre ZX-81.

Dans la collection Technique Poche :

N° 17 – Réalisez vos circuits imprimés et décors de panneaux.

N° 27 – Réduisez votre consommation d'électricité.

N° 29 – Montages économiseurs d'essence.

N° 32 – Antennes pour CiBistes.

N° 34 – Détecteurs de trésors.

Dans la collection Poche informatique :

N° 2 – Montages périphériques pour ZX-81

En langue allemande :

- Energiesparen (Franzis Verlag, München).

Maquette de couverture : Laurent MARINOT.

ISBN 2-85535-066-2

Sommaire

Index des tables figures

Index des principaux programmes

Avant-propos

Lorsque nous avons entrepris la rédaction de notre premier ouvrage d'informatique « Pilotez votre ZX-81 », notre but avoué était de faire partager à nos lecteurs les découvertes d'un débutant placé en face de cet étonnant petit appareil sans avoir jamais pratiqué la programmation.

Le succès de cette approche « partant de zéro » est venu confirmer, s'il en était besoin, que la plupart des possesseurs de ZX-81 faisaient leurs toutes premières armes sur cette machine promise à la plus large diffusion.

Or il se confirme que l'informatique individuelle, bien abordée, devient vite une passion poussant inexorablement sa victime à aller de l'avant toutes affaires cessantes.

Notre premier ouvrage laisse le lecteur voler de ses propres ailes lorsque sa connaissance du langage BASIC est devenue suffisante pour lui permettre d'écrire ses propres programmes.

Cependant, les possibilités du ZX-81, éventuellement complété par les accessoires désormais commercialisés, s'étendent bien au-delà de la stricte programmation BASIC.

Notre volumineux courrier atteste que nos lecteurs veulent écrire de très longs programmes, accéder directement à la mémoire centrale, brancher des « interfaces » permettant au ZX-81 de composer de la musique, de téléphoner par ses propres moyens, voire de faire office de « robot » à vocation domestique.

Même en mode rapide, les programmes BASIC ne « tournent » pas encore assez vite pour certains, alors que d'autres rêvent de fonctions que ce BASIC « résident » ne permet pas d'obtenir. Bref, le recours à la programmation en langage machine fait l'objet de bien des convoitises !

Seulement voilà, le bon usage des instructions PEEK, POKE et USR, n'est pas une mince affaire pour le débutant, qui doit franchir une marche singulièrement élevée pour passer du BASIC à l'Assembleur.

Cette marche, l'auteur de ces lignes a passé un nombre respectable de soirées à la franchir, mais il peut maintenant vous tendre la main pour vous aider à le suivre de l'autre côté, sur une route qui apparaît beaucoup plus praticable une fois le premier obstacle surmonté !

Chapitre 1

La programmation avec le module 16 K RAM

Tous les programmes proposés dans notre premier ouvrage avaient été mis au point sur un ZX-81 « de base » possédant une mémoire vive (RAM) de 1 K-octet. S'il est certain que cette capacité mémoire réduite suffit largement pour apprendre la programmation, on ne tarde pas à en apercevoir les limites dès que l'on cherche à mettre au point des programmes utilitaires tant soit peu évolués et, surtout, dès que l'on cherche à utiliser des fonctions graphiques. Nous avons donné dans le livre en question des exemples de programmes ne laissant subsister aucun doute à ce sujet.

Aucune hésitation n'est donc permise, le module 16 K est bien le tout premier accessoire à acquérir, avant même la carte d'entrées-sorties, dont les possibilités sont pourtant immenses et, à coup sûr, avant l'imprimante.

Gardons-nous cependant de déduire de cet ordre de priorité que l'on aurait intérêt à investir tout son budget « accessoires » dans un des modules RAM de capacité supérieure à 16 K qui commencent à apparaître sur le marché. Enormément de choses peuvent être faites avec 16 K-octets (l'équivalent de 500 lignes de 32 caractères, soit une vingtaine d'écrans TV ou un listing de près de 1,50 mètre de long !). En pratique, seuls les longs fichiers de données ou les grands tableaux peuvent parvenir à remplir complètement la mémoire, et de tels cas ne se présentent guère que dans le cadre d'applications professionnelles.

Nous allons donner quelques exemples pratiques de programmes exécutant des tâches complexes sans pour autant exiger plus de 16 K-octets de mémoire vive. Précisons que l'entrée au clavier de ces programmes exige un soin extrême, la probabilité d'erreur de

frappe augmentant avec la longueur du programme, surtout en ce qui concerne les caractères graphiques.

Le programme listé à la *figure 1-1* utilise le sous-programme de tracé de la carte de France que nous avions présenté à la fin de notre premier ouvrage. Un jeu éducatif animé vient se greffer à la suite, dont le but est de faire retenir au joueur la position géographique des principales villes du pays.

```
    1 REM "FRANCE"
    2 FOR N=1 TO LEN A$
    3 IF CODE A$(N)>=29 AND CODE
A$(N)<=37 THEN GOTO 10
    4 IF A$(N)="0" THEN PRINT ""
    5 IF A$(N)="0" THEN GOTO 8
    6 PRINT A$(N);
    8 NEXT N
    9 GOTO 15
   10 FOR F=1 TO VAL A$(N)
   11 PRINT " ";
   12 NEXT F
   13 LET N=N+1
   14 GOTO 3
   15 LET L=12
   16 LET C=12
   19 GOSUB 100
   20 PRINT AT L,C;"■"
   21 PRINT AT L,C;" "
   22 IF INKEY$="5" THEN GOTO 36
   23 IF INKEY$="5" THEN LET C=C-
1
   24 IF INKEY$="8" THEN LET C=C+
1
   25 IF INKEY$="7" THEN LET L=L-
1
   28 IF INKEY$="6" THEN LET L=L+
1
   30 GOTO 20
   36 PRINT AT 0,17;"
   "
   38 PRINT AT 1,17;"
   "
   40 IF CT>C THEN PRINT AT 0,17;
"PLUS A L""EST"
   45 IF CT<C THEN PRINT AT 0,17;
"PLUS A L""OUEST"
   50 IF LT<L THEN PRINT AT 1,17;
"PLUS AU NORD"
   55 IF LT>L THEN PRINT AT 1,17;
"PLUS AU SUD"
   60 IF LT=L AND CT=C THEN GOTO
200
```

```
  70 GOTO 20
 100 LET R=INT (RND*20)
 103 LET I=1+(11*R)
 105 LET J=1+(2*R)
 140 PRINT AT 0,0;"                "
 150 PRINT AT 0,0;B$(I TO I+10)
 160 LET LT=VAL L$(J TO J+1)
 170 LET CT=VAL C$(J TO J+1)
 180 RETURN
 200 PRINT AT 0,17;"SE SITUE ICI
"
 205 PRINT AT L,C;CHR$ 8
 210 PAUSE 200
 215 PRINT AT 0,17;"
"
 220 GOTO 15
 250 REM COPYRIGHT 1982
 260 SAVE "FRANCE"
 270 GOTO 1
```

VARIABLES (DONNEES)

```
A$ : 778065 055 3 08 5 05
 7 02 2 77 01 78 01 99
02 88 03 77 05 67 05 66 06 56
6 66 06 66 06 67 06 67 05 77 05
 6 2 06 6 3 07 2 065
```

```
B$ : LE HAVRE    LILLE        ROUEN
       CHERBOURG  BREST        RENN
ES       LE MANS     PARIS       MET
Z         STRASBOURG NANTES       TO
URS         BOURGES     DIJON        P
OITIERS    LYON          BORDEAUX
CLERMONT FDTOULOUSE    AVIGNON
  MARSEILLE
```

```
L$ : 03010404060707060506090910@
911131513161719
```

```
C$ : 08131006010508121820050912 1
80816071310161 7
```

Fig. 1-1.

On remarquera que, pour des raisons de simplification du programme proprement dit, les données sont présentées séparément, et qu'il faudra les frapper, à la suite de *commandes* LET, soit avant, soit après la frappe des instructions du programme. On fera, par exemple :

LET A$ = 778065... puis NEWLINE, puis
LET B$ = LE HAVRE # # # LILLE... puis NEWLINE,
puis de même pour L$ et C$.

Lors de la frappe de A$, on veillera, grâce à de fréquentes vérifications, à ne pas commettre d'erreurs, même minimes, dans l'entrée des symboles graphiques, car notre malheureux hexagone s'en trouverait profondément déformé !

De même, la frappe de B$ exige le respect exact du nombre des espaces séparant les noms de ville les uns des autres. On les comptera donc soigneusement en se guidant sur une ligne voisine contenant des caractères au droit des espaces à compter.

Une fois entrées, ces données n'apparaîtront plus lors des listages mais seront bien sûr stockées sur la cassette lorsque l'on exécutera la commande GOTO 260.

Si l'on désire opérer un contrôle, il suffira, par exemple, de frapper la *commande* PRINT A$, ou PRINT B$, etc.

Pour retrouver exactement la présentation imprimée ici, il conviendrait cependant de faire :

PRINT «B$ # : # »;B$

sans oublier les deux espaces de part et d'autre des deux points.

Ce programme peut servir de prétexte à un court développement consacré à la fonction INKEY$:

Le principal avantage de cette fonction est de permettre d'entrer des données au clavier par la manœuvre d'une seule touche, donc sans validation par NEWLINE. En contrepartie, une fonction INKEY$ ne peut pas vous « attendre » comme une instruction INPUT.

En fait, la variable de chaîne nommée INKEY$ prend la valeur correspondant à la touche enfoncée du clavier *lors de l'exécution* de l'instruction contenant INKEY$. Si aucune touche n'est enfoncée à cet instant, la chaîne INKEY$ reste une chaîne vide. Ce mode de fonctionnement très particulier explique que les fonctions INKEY$

soient presque toujours incorporées dans des boucles de programme qui, grâce à un GOTO judicieusement placé, « tournent en rond » tant que la touche permettant de sortir de cette boucle n'a pas été

```
   5 REM "LABYRINTHE"
  10 PRINT "VOUS ALLEZ POUVOIR T
RACER UN"
  12 PRINT "PARCOURS PLUS OU MOI
NS COMPLIQUE"
  14 PRINT "EN DEPLACANT LE POIN
T BLANC"
  16 PRINT "GRACE AUX TOUCHES 5,
6,7,8"
  18 PRINT
  20 PRINT "CECI FAIT,APPUYEZ SU
R Y ET"
  21 PRINT "ESSAYEZ DE SUIVRE LE
 MEME PAR-"
  22 PRINT "COURS AVEC LE POINT
NOIR..."
  23 PRINT
  24 PRINT "ATTENTION,NE LE PERD
EZ PAS..."
  25 PAUSE 500
  30 CLS
 100 FOR F=1 TO 704
 110 PRINT "■";
 120 NEXT F
 130 LET L=22
 140 LET C=32
 150 GOSUB 500
 160 IF INKEY$="Y" THEN GOTO 300
 165 UNPLOT L,C
 170 GOTO 150
 300 GOSUB 500
 310 PLOT L,C
 320 GOTO 300
 500 IF INKEY$="5" THEN LET L=L-
1
 510 IF INKEY$="8" THEN LET L=L+
1
 520 IF INKEY$="7" THEN LET C=C+
1
 530 IF INKEY$="6" THEN LET C=C-
1
 540 RETURN
 550 REM COPYRIGHT 1982
 560 SAVE "LABYRINTH"
 570 GOTO 10
```

Fig. 1-2. – La partie de programme située entre la ligne 100 et la ligne 550 tient largement dans 1 K-octet, mais son exécution nécessite bien davantage de place mémoire.

pressée. Un avantage de plus de ce procédé est que la touche peut rester enfoncée aussi longtemps qu'on le désire et que, à chaque lecture, la chaîne INKEY$ sera mise à jour. Cela permet, dans le programme qui nous intéresse, de faire déplacer de façon continue un caractère sur l'écran, tant que dure l'appui sur la touche fixant le sens du déplacement. Ce fonctionnement est identique à celui rencontré dans le programme du « dessinateur paresseux » présenté dans « Pilotez votre ZX-81 ».

Une fois sauvegardé sur cassette par GOTO 260, puis rechargé en machine par LOAD « FRANCE », le programme se lance seul et trace donc le contour de la France. Cela achevé, un petit carré clignotant apparaît au milieu de l'écran en même temps qu'un nom de ville. Il vous reste à déplacer le carré au moyen des touches fléchées du ZX-81 puis à enfoncer la touche B dès que vous pensez avoir amené le carré au bon endroit. La suite des opérations se passe de commentaire puisque la machine prend les choses en main !

Nous ne détaillerons pas davantage le fonctionnement de ce programme qui fait appel à des notions développées dans le précédent livre. Contentons-nous de noter que, sans être encore très long, ce programme excède nettement les possibilités d'une mémoire vive de 1 K-octet, même si l'on fait abstraction de la place exigée par le fichier d'affichage une fois la carte tracée.

Les fonctions graphiques, précisément, occupent beaucoup de place en mémoire, même si le programme les mettant en œuvre reste très court. La *figure 1-2* donne un exemple d'un tel programme, tenant largement dans 1 K-octet mais saturant complètement le reste de cette capacité mémoire dès son lancement. Un module 16 K fait bien évidemment tout rentrer dans l'ordre et permet de passer quelques moments distrayants devant son ordinateur : aucune explication n'est nécessaire car la machine imprime la règle du jeu sur l'écran avant le début des opérations !

Le domaine d'élection des fortes capacités mémoire reste cependant le traitement (et non le stockage, les cassettes étant faites pour cela) des gros volumes de données. L'exemple typique, repris par le programme de la *figure 1-3,* est celui de la gestion d'un stock commercial. Ce programme de démonstration permet de traiter jusqu'à 500 articles distincts, tous repérés par un code étiquette à cinq chiffres, un prix modifiable à loisir, et assortis de données supplémentaires telles que : nom abrégé, nombre d'article en stock, nombre minimum à tenir en stock, nombre d'articles à commander, etc.

```
   10 REM "GESTION"
   15 LET X=500
   20 DIM C$(X,5)
   22 DIM P(X)
   24 DIM N$(X,3)
   26 DIM D$(X,7)
   28 DIM M$(X,3)
   29 CLS
   30 PRINT "NATURE D""OPERATION
?"
   31 PRINT "--------------------
"
   32 PRINT
   33 PRINT
   35 PRINT "SORTIE,FRAPPER S"
   36 PRINT
   38 PRINT "ENTREE,FRAPPER E"
   39 PRINT
   40 PRINT "MODIFICATION PRIX,FR
APPER M"
   41 PRINT
   42 PRINT "MODIF ARTICLE,FRAPPE
R A"
   43 PRINT
   44 PRINT "INVENTAIRE,FRAPPER I
"
   45 PRINT
   46 PRINT "COMMANDES,FRAPPER C"
   47 PRINT
   48 PRINT "MISE SUR K7,FRAPPER
K"
   52 IF INKEY$="S" THEN GOTO 100
   55 IF INKEY$="E" THEN GOTO 100
0
   60 IF INKEY$="M" THEN GOTO 200
0
   65 IF INKEY$="I" THEN GOTO 300
0
   70 IF INKEY$="C" THEN GOTO 400
0
   80 IF INKEY$="K" THEN GOTO 700
0
   85 IF INKEY$="A" THEN GOTO 750
0
   90 GOTO 50
  100 GOSUB 8000
  110 LET L=1
  120 IF C$(L)=K$ THEN GOTO 200
  130 LET L=L+1
  140 IF L>=X THEN GOSUB 8000
  150 GOTO 120
  200 SLOW
  205 PRINT "NOMBRE DE ";D$(L);"
SORTIS ?"
  210 INPUT NB
```

(suite au verso)

```
 220 PRINT
 230 LPRINT NB;" ";D$(L);" A ";P
(L);" SOIT ";NB*P(L);" F"
 240 LET N$(L)=STR$ (VAL N$(L)-N
B)
 250 IF VAL N$(L)<=VAL M$(N) THE
N LPRINT "COMMANDER  :  ";D$(L);
"  (";C$(L);")"
 260 GOTO 29
1000 GOSUB 8000
1030 LET L=1
1045 IF C$(L)=K$ THEN GOTO 1100
1050 LET L=L+1
1060 IF C$(L)="       " THEN GOTO
1200
1070 GOTO 1040
1100 SLOW
1102 LET N=L
1105 PRINT
1110 PRINT "NOMBRE D""ARTICLES E
NTRES ?"
1120 INPUT NB
1125 PRINT
1130 PRINT "IL RESTAIT ";N$(N);"
 ";D$(N)
1135 PRINT
1140 PRINT "IL Y EN A MAINTENANT
 ";VAL N$(N)+NB
1145 LPRINT "ENTRE ";NB;" ";D$(N
);" (";C$(N);")"
1150 LET N$(N)=STR$ (VAL N$(N)+N
B)
1154 PRINT
1155 PRINT "LE MINIMUM ETANT ";M
$(N)
1160 PAUSE 200
1170 GOTO 29
1200 SLOW
1205 LET L=1
1210 IF C$(L)="       " THEN GOTO
1260
1220 LET L=L+1
1230 IF L>=X THEN PRINT "FICHIER
 PLEIN"
1235 IF L>=X THEN PAUSE 200
1240 IF L>=X THEN GOTO 29
1250 GOTO 1210
1260 LET N=L
1282 PRINT
1285 LET C$(N)=K$
1290 PRINT "DESIGNATION ?"
1295 PRINT
1300 INPUT S$
1305 LET D$(N)=S$
1310 PRINT "PRIX ?"
```

```
1315 PRINT
1320 INPUT P
1325 LET P(N)=P
1330 PRINT "NOMBRE MINI ?"
1335 PRINT
1340 INPUT V$
1345 LET M$(N)=V$
1350 PRINT "NOMBRE ENTRE ?"
1355 PRINT
1360 INPUT E$
1365 LET N$(N)=E$
1370 LPRINT "ENTRE ";E$;" ";D$(N
);" (";C$(N);")"
1380 GOTO 29
2000 SLOW
2005 GOSUB 8000
2010 LET L=1
2020 IF C$(L)=K$ THEN GOTO 2100
2030 LET L=L+1
2040 IF L>=X THEN GOTO 2000
2050 GOTO 2020
2100 SLOW
2105 PRINT
2110 PRINT "ANCIEN PRIX :",P(L)
2120 PRINT
2130 PRINT "NOUVEAU PRIX ?"
2140 INPUT P(L)
2150 GOTO 29
3000 CLS
3002 FOR F=1 TO X
3005 SCROLL
3007 IF C$(F)<>"      " THEN GOTO
 3020
3010 GOTO 3030
3015 GOTO 3007
3020 PRINT N$(F);" ";D$(F);" ";C
$(F);" A ";P(F);" F"
3025 NEXT F
3030 SCROLL
3032 SCROLL
3035 PRINT "INVENTAIRE TERMINE"
3036 SCROLL
3040 PAUSE 300
3050 GOTO 29
4000 CLS
4001 PRINT
4002 PRINT
4003 PRINT "COMMANDER AU MOINS :
"
4004 PRINT
4005 LET L=1
4010 IF C$(L)="      " THEN GOTO
3035
4015 IF VAL N$(L)<VAL M$(L) THEN
 GOSUB 5000
4020 LET L=L+1
```

(suite au verso)

```
4030 GOTO 4010
5000 PRINT "- ";VAL M$(L)-VAL N$
(L);" ";D$(L);" ";C$(L)
5010 RETURN
7000 CLS
7010 PRINT "DEMARRER LE MAGNETOP
HONE"
7020 PRINT
7030 PRINT "PUIS PRESSER [illegible]
"
7040 INPUT H$
7050 SAVE "GESTIO[illegible]"
7060 CLS
7070 PRINT "ARRETER LE MAGNETOPH
ONE"
7080 PRINT
7090 PRINT "PUIS PRESSER [illegible]
"
7100 INPUT H$
7110 GOTO 29
7500 CLS
7510 PRINT "ARTICLE SUPPRIME :"
7520 PRINT "--------------------"
7530 GOSUB 8002
7540 SLOW
7570 LET L=1
7580 IF C$(L)=K$ THEN GOTO 7700
7590 LET L=L+1
7600 IF L>=X THEN GOTO 7500
7610 GOTO 7580
7700 CLS
7710 PRINT "NOUVEL ARTICLE :"
7720 PRINT "-----------------"
7730 GOSUB 8002
7735 SLOW
7740 GOTO 1260
8000 CLS
8002 PRINT
8005 PRINT "CODE ARTICLE :",
8010 INPUT K$
8020 PRINT K$
8022 PRINT
8025 FAST
8030 RETURN
8100 REM COPYRIGHT 1982
```

Fig. 1-3.

Toutes les opérations d'entrée, sortie, changement de prix, inventaire, commandes, etc., peuvent être facilement exécutées. La sauvegarde des données à jour est prévue sur cassette en fin de journée, mais un enregistrement immédiat sur papier de chaque

opération est prévu en cas d'incident de fonctionnement. Un rechargement manuel des opérations perdues peut alors être effectué.

Le programme lui-même n'occupe que 3 K-octets environ malgré sa longueur, et on arrive à remplir la totalité du module 16 K avec les données relatives à 500 articles. Ce n'est donc qu'au-delà de ce chiffre qu'une capacité supérieure s'imposerait.

Une remarque s'impose à ce sujet : le programme débute par une série d'instructions DIM venant « déclarer » à la machine quel sera l'encombrement maximum des *tableaux* qu'il lui sera demandé de stocker en mémoire vive. L'exécution d'une instruction DIM *réserve* une fois pour toutes cette place maximum afin d'écarter tout risque de double utilisation. Cela signifie que, même si aucune donnée relative à un quelconque article n'a été entrée, la totalité des 16 K-octets de la RAM se trouve occupée dès que le programme est lancé. La conséquence de cela est que, si une sauvegarde sur cassette est entreprise, la durée de l'enregistrement atteindra environ *neuf minutes,* mais que cette durée n'augmentera pas lorsque des informations réelles viendront progressivement prendre la place des zéros de « remplissage » que les instructions DIM ont entassés dans la mémoire.

Les effets de cette « réservation » de place par les instructions DIM apparaissent nettement lorsque l'on entre les désignations des articles : celles-ci sont en effet stockées dans un *tableau* de 500 chaînes de 7 caractères. L'instruction DIM D$ (500,7) *initialise* ces caractères en en faisant des espaces. Si le programme vient remplir une des chaînes du tableau avec un mot de moins de 7 caractères (par exemple VIN), tout se passe bien, et il reste même quelques espaces inutilisés. Si, par contre, nous avons à stocker la désignation *confiture,* celle-ci ne sera restituée par la suite que sous la forme tronquée CONFITU. Cette troncature automatique, connue sous le nom d'affectation de Procruste, facilite le traitement des chaînes de caractères et est utilisée à outrance par les services informatiques des administrations ou grandes entreprises. Voici peut-être l'explication de cet entêtement apparent de la Sécurité sociale ou de votre fournisseur de chaussettes par correspondance à abréger malencontreusement votre prénom !

On remarquera également que les données chiffrées que sont les quantités d'articles en stock sont mémorisées sous forme de chaînes N$ et M$ et non sous forme numérique comme le prix P.

La raison de cette originalité apparente est que, sous réserve que ces quantités ne dépassent pas 999 (3 chiffres), il est beaucoup plus économique, en termes de place mémoire, d'établir une chaîne

de 3 caractères (dont la valeur numérique pourra être calculée à loisir), que de stocker une constante numérique avec mantisse et exposant de 10, sans parler du signe ! Pour la même raison, il est bien plus avantageux d'ajouter la ligne 15 LET X = 500 que d'introduire la constante 500 dans chacune des lignes qui en ont besoin (et elles sont nombreuses !).

Autre fonction spéciale utilisée par ce programme, l'instruction SCROLL (de l'anglais SCreen ROLL, ou déroulement d'écran).

Cette fonction très utile permet de faire défiler de bas en haut sur l'écran un nombre de lignes très supérieur à la capacité normale de l'affichage sans nécessiter un arrêt puis un redémarrage du programme dès que l'écran est plein.

Seulement, le manuel Sinclair reste excessivement discret sur le fonctionnement de cette instruction sur laquelle bien des débutants se sont cassés les dents !

Chaque fois qu'une instruction SCROLL est exécutée, tout le contenu de l'écran, quel qu'il soit (même un écran vide), est déplacé d'une ligne vers le haut (la ligne du haut est donc perdue) et, en même temps, la *position d'écriture* est amenée à la ligne du bas qui vient d'être dégagée par la remontée générale. On est donc libre d'user à volonté de cette ligne pour introduire de nouveaux caractères, mais il faut veiller attentivement à ne pas dépasser, ne fût-ce que d'un caractère, la capacité de cette dernière ligne (32 caractères). Tout dépassement poserait à la machine le problème insoluble consistant à continuer l'écriture sur une ligne n'existant pas, d'où un arrêt du programme sur un compte rendu d'écran plein !

En effet, retenons que la fonction SCROLL *ne sait pas aller à la ligne* et que le programme doit donc surveiller les risques de débordement de la dernière ligne et lancer en temps voulu un ordre SCROLL supplémentaire pour dégager la place nécessaire. Ceci explique que c'est à l'intérieur de boucles FOR NEXT ou GOTO/GOSUB que la fonction SCROLL est la plus facile à mettre en œuvre.

Enfin, on notera que l'effacement d'un écran rempli « à coups de SCROLL » est très lent, qu'il soit commandé par un CLS, un RUN ou par tout autre procédé. Ce phénomène est particulièrement net à la fin d'un inventaire.

Nous ne fournirons pas d'autres explications sur ce programme, dont le dialogue avec l'utilisateur a été étudié pour guider celui-ci à travers les diverses opérations devant être exécutées.

Egalement, malgré sa longueur, ce programme ne fait pas appel à des fonctions particulières autres que celles expliquées plus haut. Par contre, un soin tout particulier doit être apporté à sa frappe au clavier et à son enregistrement sur cassette, comme nous allons le voir à présent.

Enregistrement sur cassette de programmes 16 K

Il est bien tentant de décider une bonne fois pour toutes que le chargement sur cassette de programmes 16 K consiste en une simple extrapolation des principes utilisés avec les programmes 1 K et que, somme toute, il n'y a pas là matière à fouetter un chat !

En fait, il y a parfois assez loin de la théorie à la pratique, et cette vérité profonde s'applique fort bien à notre propos. Il faut savoir que le procédé d'enregistrement choisi par Sinclair présente, certes, des avantages certains mais reste en revanche assez sensible aux perturbations provenant de défauts dans la bande magnétique, de réglages de niveau insuffisamment précis et surtout de problèmes d'*azimutage* des têtes du magnétophone.

Ces dernières difficultés ne se présentent, le plus souvent, que lors de tentatives de lecture de cassettes d'édition achetées tout enregistrées. En effet, les équipements de duplication servant à la fabrication de ces cassettes sont réglés selon les normes « studio » extrêmement sévères. En revanche, les petits magnétophones bon marché, par ailleurs très suffisants pour un usage informatique, sortent trop souvent d'usine ajustés tant bien que mal et plutôt mal que bien ! Compte tenu de leur sonorité souvent déplorable, l'oreille ne trouve guère de différence entre une lecture effectuée avec des têtes réglées de façon parfaite ou avec un alignement douteux. L'ordinateur, lui, est plus exigeant et n'acceptera le programme que si l'alignement tête-bande est exactement le même à l'enregistrement et à la lecture.

En cas de difficultés de cet ordre, on agira avec précaution sur la petite vis de réglage de la tête de lecture, accessible soit à travers un petit trou, soit après démontage d'un cache décoratif.

Le bon réglage est obtenu lorsque la sonorité produite par la lecture d'un programme est la plus sèche possible (maximum d'aiguës).

Ces difficultés, ainsi que toutes celles dues à d'autres défauts, sont beaucoup plus gênantes dans le cas de longs programmes, car

il est clair que, pour un nombre donné de défauts sur une même cassette, les risques sont bien plus importants si le programme dure sept ou huit minutes que s'il n'occupe que vingt secondes de bande.

On peut admettre, *grosso modo,* que la durée de bande nécessaire à un programme donné est d'environ 30 s par K-octet de programme et/ou de données (variables, tableaux, etc.). On comprend donc que le plus grand soin soit de rigueur pour les manipulations autour du programme présenté précédemment.

Toujours dans le domaine des longs programmes, nous voudrions donner à nos lecteurs un conseil d'ami : sachant que la mise au point d'un programme occupant quelques K-octets en mémoire exige des heures, voire des jours de travail, la plus élémentaire prudence consiste à effectuer des sauvegardes partielles sur une cassette dite « de travail », disons toutes les quinze à vingt lignes de programme.

Le temps ainsi investi (nous nous refusons catégoriquement à le qualifier de perdu) sera, à coup sûr, rendu au centuple lorsque surviendra la quasi inévitable perte de données.

On peut avoir une entière confiance dans le réseau électrique et être victime d'une trahison du ZX lui-même ou de son bloc secteur. Sachez en effet que ce matériel n'apprécie guère un fonctionnement continu dépassant deux heures, suite à l'échauffement assez conséquent du régulateur 5 V et du transformateur.

Une sage précaution consiste à refaire systématiquement toutes les soudures du bloc secteur, qui semble beaucoup souffrir lors de sa traversée de la Manche, et, éventuellement, à ménager quelques orifices de ventilation dans la coquille supérieure du boîtier du ZX. Le bloc 16 K vient en effet boucher hermétiquement la seule ouverture qui pouvait permettre à l'air chaud de retourner à l'atmosphère.

Même après ces transformations, des « trous de mémoire » peuvent encore se produire, rendant très recommandables les précautions évoquées plus haut.

Compatibilité entre programmes 1 K et 16 K

Il ne fait aucun doute qu'un programme occupant plus de 1 K-octet en mémoire vive (donc écrit à l'aide d'un module 16 K) ne pourra jamais être chargé sur un ZX-81 dépourvu de cet accessoire.

La réciproque semble absolument évidente, et, pourtant, il faut distinguer deux cas très différents.

Si un programme a été enregistré sur une cassette à partir d'un ZX-81 de base, dépourvu de toute extension mémoire, il pourra être entré de nouveau en machine, même si celle-ci est équipée d'un bloc de RAM, quelle qu'en soit la capacité.

Par contre, il n'est pas possible de faire entrer en machine un programme, même très court, mais écrit en présence d'un bloc d'extension mémoire, si ce bloc n'est plus en service lors de ce rechargement. Cela peut poser de sérieux problèmes à tous ceux qui, craignant avec juste raison une fatigue rapide des connecteurs, laissent raccordé en permanence leur module 16 K, voire leur imprimante.

Il est instructif de tenter d'écrire un programme extrêmement court, par exemple :

10 PRINT « BONJOUR »

alors que le module mémoire de 16 K est en place.

Bien que ce programme n'occupe que quelques octets de RAM, on peut vérifier que sa sauvegarde sur cassette dure beaucoup plus longtemps que celle de ce même programme frappé en l'absence d'extension mémoire. Un tel enregistrement « enrichi » ne peut plus être rechargé sur un ZX dépourvu du module 16 K, bien que le programme tienne très largement dans 1 K-octet et même moins !

Sans en avoir la certitude, car les « petits secrets » du ZX sont très difficiles à percer, nous pensons que le problème provient du fait que le « fichier d'affichage », lorsqu'il est vide, tient beaucoup plus de place en configuration 16 K, et que, peut-être, il serait sauvegardé (en pure perte d'ailleurs) en même temps que le programme et ses données (?).

Il existe cependant un moyen de s'affranchir de cette petite difficulté : lors de la mise sous tension de la machine, on peut « déconnecter » logiciellement le bloc 16 K en frappant la courte *commande* suivante :

POKE 16389,68 (suivie de NEWLINE)

Si toutefois, même en cours d'écriture du programme, la place mémoire venait à manquer, on pourrait « remettre en service » le bloc d'extension en frappant :

POKE 16389,128 (suivie de NEWLINE)

Sans entrer davantage, pour l'instant, dans le détail de l'opération, sachez que ces commandes POKE modifient une *variable,* normalement inaccessible au BASIC, qui indique au microprocesseur Z-80 la capacité mémoire dont il dispose. Une telle variable « interne » est dite *« variable système ».*

Cette façon de procéder est à peu près la seule qui permette de « brancher » et « débrancher » le module 16 K alors que le ZX se trouve sous tension. On rappelle, en effet, que toute manipulation sous tension au niveau du connecteur arrière peut avoir des conséquences tragiques vis-à-vis des composants du système, et doit donc être *proscrite absolument !*

Les « grandes capacités » mémoire

L'un des aspects les plus originaux du « phénomène ZX-81 » est qu'il existe, en Angleterre du moins, une multitude de toutes petites sociétés, dépourvues de tout lien avec Sinclair, qui produisent toute une gamme d'accessoires destinés à venir se connecter au ZX de base afin d'en élargir les possibilités.

Parmi ces accessoires, les modules d'extension mémoire occupent bien sûr une place de choix, et la concurrence est si âpre que, lors de la rédaction de ces lignes, on assiste outre-Manche à une chute vertigineuse des prix des modules 16 K. Certains constructeurs n'hésitent pas à livrer une carte imprimée sans boîtier afin de vendre au prix le plus bas, alors que d'autres se vantent de « faire mieux que Sinclair » en évitant le bruit assez désagréable, voire inquiétant, du module d'origine.

La bataille des prix ne constitue qu'un aspect du phénomène, qui se prolonge en une course aux grandes capacités mémoire. On peut ainsi trouver des modules 32 K, 48 K, voire plus !

Certains de ces modules à grande capacité sont disponibles en France à des prix nettement plus élevés (pour le moment du moins) que l'ordinateur lui-même.

A notre humble avis (peut-être en changerons-nous un jour !), le ZX-81 se prête de façon idéale au fonctionnement avec 16 K de RAM, et devient bien lourd à manier avec des capacités supérieures.

Ce n'est guère que pour des applications professionnelles ou très spécialisées (graphiques haute résolution) que le bloc 16 K atteint ses limites, et l'expérience montre que l'on atteint alors, à peu près en même temps, d'autres limites du ZX-81 lui-même. Il ne fait aucun doute que le ZX-81 est un ordinateur d'amateur, et c'est ce qui en fait le succès. Il ne nous paraît guère judicieux de l'équiper d'un grand nombre d'extensions (d'ailleurs branlantes, le connecteur arrière n'ayant pas été conçu dans ce sens), au risque de dépenser plus que le prix d'un système professionnel finalement plus performant (1).

Un récent voyage au pays natal des ZX nous a convaincu que le ZX-81 pouvait *tout* faire, à condition de lui adapter les compléments nécessaires, tant matériels que logiciels. Nous sommes cependant tout aussi convaincu que là n'est pas sa vocation !

(1) Par exemple, le tout nouveau ZX Spectrum de Sinclair, dont les possibilités sont impressionnantes mais qui se prête moins bien à l'initiation des débutants.

Chapitre 2

Explorez la mémoire de la machine

Le programmeur se limitant à l'utilisation du ZX-81 sous BASIC ne se trouve pratiquement jamais en « prise directe » avec la mémoire centrale. S'il écrit une instruction telle que :

20 LET A$ = « TEXTE »

Il sait bien que plusieurs cellules mémoire seront occupées par des indications permettant à l'ordinateur de conserver la trace de ce texte, de savoir que celui-ci s'appelle A$ et d'exécuter cette affectation à la ligne 20 du programme, mais il ignorera jusqu'au bout quelles sont ces cellules. Il est d'ailleurs fort probable que ces informations « navigueront » dans la mémoire pendant l'exécution du programme, sans que l'utilisateur ne s'en aperçoive le moins du monde.

En fait, le microprocesseur Z-80 exécute un programme en *« langage machine »* délivrant le programmeur du souci permanent de gérer la mémoire et, surtout, lui permettant d'écrire son programme dans un langage proche du langage courant, le BASIC.

Ce programme de « traduction » s'appelle interpréteur BASIC et « réside » dans la mémoire morte (ou ROM) de 8 K-octets dont est muni le ZX-81.

Cependant, nous allons voir qu'il peut être avantageux, dans certains cas, de « marcher sur les brisées » de l'interpréteur en accédant directement aux cellules mémoire de la RAM ou, même, de la ROM.

Pour ce faire, il est indispensable de pouvoir se référer à une sorte de « carte routière » permettant de localiser les cellules mémoires, sur le contenu desquelles le programmeur peut se trouver

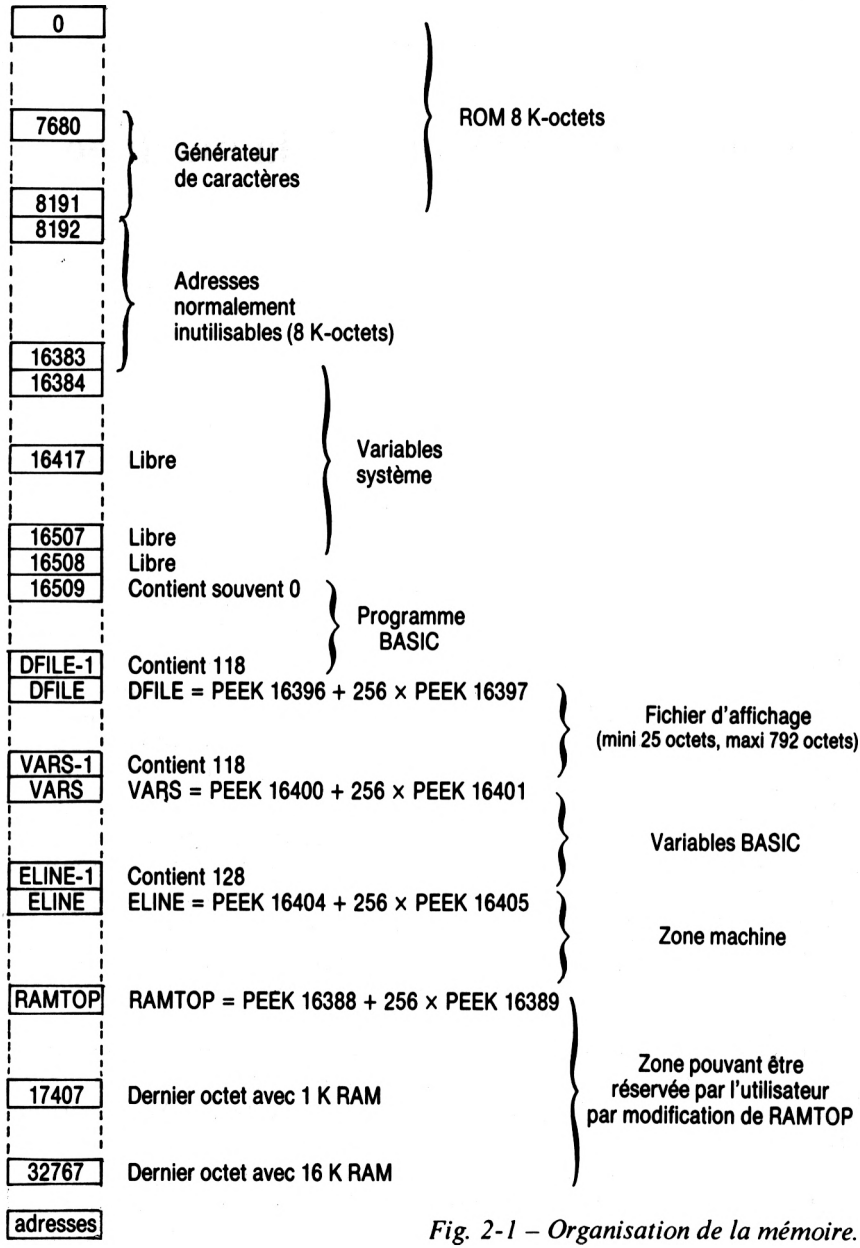


Fig. 2-1 – Organisation de la mémoire.

appelé à intervenir. La *figure 2-1* rassemble ainsi toutes les indications de cette nature nécessaires à la compréhension des notions exposées dans la suite de cet ouvrage.

Il faut savoir que chaque cellule de la mémoire peut contenir un *octet,* ou groupe de huit *éléments binaires,* ou *bits* en notation anglaise. (A ne pas confondre avec byte, traduction du terme français octet.) Chaque élément binaire peut prendre la valeur 0 ou 1, d'où l'existence de 256 octets possibles, de 00000000 à 11111111. En pratique, il est rare de représenter les octets sous cette forme développée, la plus proche cependant de la machine.

Les informaticiens utilisent généralement la notation *hexadécimale* (de ØØ à FF), qui présente certains avantages, mais qui ne se prête guère à une utilisation sur le ZX-81.

En ce qui nous concerne, il nous a fallu prendre le parti, que certains pourront trouver déchirant mais qui convient beaucoup mieux au débutant, de travailler sur la *représentation décimale* (de 0 à 255) des octets.

Un octet ne peut donc prendre que 256 valeurs distinctes, ce qui est loin d'être suffisant en pratique. Par exemple, il n'est pas possible d'utiliser un seul octet pour représenter *l'adresse,* c'est-à-dire le *numéro* d'une cellule mémoire donnée, puisque le microprocesseur Z-80 peut en accepter jusqu'à 64 K, soit exactement 65536 ! Dans ce cas précis, on fait appel à deux octets, le premier ayant un *poids* de 1 (octet le moins significatif, ou OMS) et le second ayant un poids de 256 (octet le plus significatif, ou OPS).

Expliquons-nous :

Avec seulement 1 K-octet de RAM, le ZX-81 doit gérer des adresses de cellules mémoire comprises entre 0 et 17407 (en décimal). Il est clair qu'un seul octet ne peut servir à numéroter que les cellules 0 à 255. Au-delà, on introduit un deuxième octet, qui « comptera pour 256 » tout comme en numération décimale, le chiffre des unités compte pour 1, celui des dizaines pour 10, etc. Ainsi, par exemple, la cellule n° 16514 sera identifiée par les deux octets suivants :

OMS : 130 OPS : 64

On peut vérifier que 130 + (256 × 64) = 16514.

Il est primordial pour la bonne compréhension de ce qui va suivre de bien retenir que si une cellule mémoire donnée ne peut *contenir qu'un seul octet,* il en faut *deux* pour *stocker son numéro.*

Ces deux octets ne sont évidemment pas permutables, et il faut être très vigilant quant à l'ordre dans lequel on range l'OMS et l'OPS (pas toujours le même selon les opérations effectuées par la machine !).

```
16514 : OMS = 130 : OPS = 64
16515 : OMS = 131 : OPS = 64
16516 : OMS = 132 : OPS = 64
16517 : OMS = 133 : OPS = 64
16518 : OMS = 134 : OPS = 64
16519 : OMS = 135 : OPS = 64
16520 : OMS = 136 : OPS = 64
16521 : OMS = 137 : OPS = 64
16522 : OMS = 138 : OPS = 64
16523 : OMS = 139 : OPS = 64
16524 : OMS = 140 : OPS = 64
16525 : OMS = 141 : OPS = 64
16526 : OMS = 142 : OPS = 64
16527 : OMS = 143 : OPS = 64
16528 : OMS = 144 : OPS = 64
16529 : OMS = 145 : OPS = 64
16530 : OMS = 146 : OPS = 64
16531 : OMS = 147 : OPS = 64
16532 : OMS = 148 : OPS = 64
16533 : OMS = 149 : OPS = 64
16534 : OMS = 150 : OPS = 64
16535 : OMS = 151 : OPS = 64
16536 : OMS = 152 : OPS = 64
16537 : OMS = 153 : OPS = 64
16538 : OMS = 154 : OPS = 64
16539 : OMS = 155 : OPS = 64
16540 : OMS = 156 : OPS = 64
16541 : OMS = 157 : OPS = 64
16542 : OMS = 158 : OPS = 64
16543 : OMS = 159 : OPS = 64
16544 : OMS = 160 : OPS = 64
16545 : OMS = 161 : OPS = 64
16546 : OMS = 162 : OPS = 64
16547 : OMS = 163 : OPS = 64
16548 : OMS = 164 : OPS = 64
16549 : OMS = 165 : OPS = 64
16550 : OMS = 166 : OPS = 64
16551 : OMS = 167 : OPS = 64
16552 : OMS = 168 : OPS = 64
16553 : OMS = 169 : OPS = 64
16554 : OMS = 170 : OPS = 64
16555 : OMS = 171 : OPS = 64
16556 : OMS = 172 : OPS = 64
16557 : OMS = 173 : OPS = 64
16558 : OMS = 174 : OPS = 64
16559 : OMS = 175 : OPS = 64
16560 : OMS = 176 : OPS = 64
```

```
16561 : OMS = 177 : OPS = 64
16562 : OMS = 178 : OPS = 64
16563 : OMS = 179 : OPS = 64
16564 : OMS = 180 : OPS = 64
16565 : OMS = 181 : OPS = 64
16566 : OMS = 182 : OPS = 64
16567 : OMS = 183 : OPS = 64
16568 : OMS = 184 : OPS = 64
16569 : OMS = 185 : OPS = 64
16570 : OMS = 186 : OPS = 64
16571 : OMS = 187 : OPS = 64
16572 : OMS = 188 : OPS = 64
16573 : OMS = 189 : OPS = 64
16574 : OMS = 190 : OPS = 64
16575 : OMS = 191 : OPS = 64
16576 : OMS = 192 : OPS = 64
16577 : OMS = 193 : OPS = 64
16578 : OMS = 194 : OPS = 64
16579 : OMS = 195 : OPS = 64
16580 : OMS = 196 : OPS = 64
16581 : OMS = 197 : OPS = 64
16582 : OMS = 198 : OPS = 64
16583 : OMS = 199 : OPS = 64
16584 : OMS = 200 : OPS = 64
16585 : OMS = 201 : OPS = 64
16586 : OMS = 202 : OPS = 64
16587 : OMS = 203 : OPS = 64
16588 : OMS = 204 : OPS = 64
16589 : OMS = 205 : OPS = 64
16590 : OMS = 206 : OPS = 64
16591 : OMS = 207 : OPS = 64
16592 : OMS = 208 : OPS = 64
16593 : OMS = 209 : OPS = 64
16594 : OMS = 210 : OPS = 64
16595 : OMS = 211 : OPS = 64
16596 : OMS = 212 : OPS = 64
16597 : OMS = 213 : OPS = 64
16598 : OMS = 214 : OPS = 64
16599 : OMS = 215 : OPS = 64
16600 : OMS = 216 : OPS = 64
```

Fig. 2-2. – Représentation des adresses 16514 à 16600 sur 2 octets.

Nous utiliserons largement par la suite les *adresses* 16514 à 16600 pour la programmation en langage machine. Nous reproduisons donc à la *figure 2-2* la décomposition de ces adresses en OMS et OPS, alors que la *figure 2-3* donne la table de multiplication par 256 au complet, facilitant de la sorte le calcul des adresses jusqu'à 65536, si nécessaire, à l'aide de toute calculette simple ! Essayons par exemple de déterminer les octets représentant l'adresse 16417 :

```
1*256=  256        2*256=  512
3*256=  768        4*256=  1024
5*256=  1280       6*256=  1536
7*256=  1792       8*256=  2048
9*256=  2304       10*256=  2560
11*256=  2816      12*256=  3072
13*256=  3328      14*256=  3584
15*256=  3840      16*256=  4096
17*256=  4352      18*256=  4608
19*256=  4864      20*256=  5120
21*256=  5376      22*256=  5632
23*256=  5888      24*256=  6144
25*256=  6400      26*256=  6656
27*256=  6912      28*256=  7168
29*256=  7424      30*256=  7680
31*256=  7936      32*256=  8192
33*256=  8448      34*256=  8704
35*256=  8960      36*256=  9216
37*256=  9472      38*256=  9728
39*256=  9984      40*256=  10240
41*256=  10496     42*256=  10752
43*256=  11008     44*256=  11264
45*256=  11520     46*256=  11776
47*256=  12032     48*256=  12288
49*256=  12544     50*256=  12800
51*256=  13056     52*256=  13312
53*256=  13568     54*256=  13824
55*256=  14080     56*256=  14336
57*256=  14592     58*256=  14848
59*256=  15104     60*256=  15360
61*256=  15616     62*256=  15872
63*256=  16128     64*256=  16384
65*256=  16640     66*256=  16896
67*256=  17152     68*256=  17408
69*256=  17664     70*256=  17920
71*256=  18176     72*256=  18432
73*256=  18688     74*256=  18944
75*256=  19200     76*256=  19456
77*256=  19712     78*256=  19968
79*256=  20224     80*256=  20480
81*256=  20736     82*256=  20992
83*256=  21248     84*256=  21504
85*256=  21760     86*256=  22016
87*256=  22272     88*256=  22528
89*256=  22784     90*256=  23040
91*256=  23296     92*256=  23552
93*256=  23808     94*256=  24064
95*256=  24320     96*256=  24576
97*256=  24832     98*256=  25088
99*256=  25344     100*256=  25600
101*256=  25856    102*256=  26112
103*256=  26368    104*256=  26624
105*256=  26880    106*256=  27136
```

```
107*256=  27392   108*256=  27648
109*256=  27904   110*256=  28160
111*256=  28416   112*256=  28672
113*256=  28928   114*256=  29184
115*256=  29440   116*256=  29696
117*256=  29952   118*256=  30208
119*256=  30464   120*256=  30720
121*256=  30976   122*256=  31232
123*256=  31488   124*256=  31744
125*256=  32000   126*256=  32256
127*256=  32512   128*256=  32768
129*256=  33024   130*256=  33280
131*256=  33536   132*256=  33792
133*256=  34048   134*256=  34304
135*256=  34560   136*256=  34816
137*256=  35072   138*256=  35328
139*256=  35584   140*256=  35840
141*256=  36096   142*256=  36352
143*256=  36608   144*256=  36864
145*256=  37120   146*256=  37376
147*256=  37632   148*256=  37888
149*256=  38144   150*256=  38400
151*256=  38656   152*256=  38912
153*256=  39168   154*256=  39424
155*256=  39680   156*256=  39936
157*256=  40192   158*256=  40448
159*256=  40704   160*256=  40960
161*256=  41216   162*256=  41472
163*256=  41728   164*256=  41984
165*256=  42240   166*256=  42496
167*256=  42752   168*256=  43008
169*256=  43264   170*256=  43520
171*256=  43776   172*256=  44032
173*256=  44288   174*256=  44544
175*256=  44800   176*256=  45056
177*256=  45312   178*256=  45568
179*256=  45824   180*256=  46080
181*256=  46336   182*256=  46592
183*256=  46848   184*256=  47104
185*256=  47360   186*256=  47616
187*256=  47872   188*256=  48128
189*256=  48384   190*256=  48640
191*256=  48896   192*256=  49152
193*256=  49408   194*256=  49664
195*256=  49920   196*256=  50176
197*256=  50432   198*256=  50688
199*256=  50944   200*256=  51200
201*256=  51456   202*256=  51712
203*256=  51968   204*256=  52224
205*256=  52480   206*256=  52736
207*256=  52992   208*256=  53248
209*256=  53504   210*256=  53760
211*256=  54016   212*256=  54272
213*256=  54528   214*256=  54784
215*256=  55040   216*256=  55296
```

(suite au verso)

```
217*256=  55552   218*256=  55808
219*256=  56064   220*256=  56320
221*256=  56576   222*256=  56832
223*256=  57088   224*256=  57344
225*256=  57600   226*256=  57856
227*256=  58112   228*256=  58368
229*256=  58624   230*256=  58880
231*256=  59136   232*256=  59392
233*256=  59648   234*256=  59904
235*256=  60160   236*256=  60416
237*256=  60672   238*256=  60928
239*256=  61184   240*256=  61440
241*256=  61696   242*256=  61952
243*256=  62208   244*256=  62464
245*256=  62720   246*256=  62976
247*256=  63232   248*256=  63488
249*256=  63744   250*256=  64000
251*256=  64256   252*256=  64512
253*256=  64768   254*256=  65024
255*256=  65280   256*256=  65536
```

Fig. 2-3. – Table de multiplication par 256 pour toutes opérations sur 2 octets.

Nous cherchons dans la table l'OPS le plus proche *par défaut*, soit 16384 = *64* × 256. Nous déduisons facilement l'OMS en faisant 16417 – 16384 = *33*, d'où le résultat cherché :

16417 # = # OMS : 33 OPS : 64

Vérifions que 16417 = OMS + 256 × OPS = 33 + 256 × 64.

Lorsque nous voudrons *adresser* des cellules mémoire par le jeu d'instructions en *langage machine*, il nous faudra souvent procéder à de telles décompositions.

Fort heureusement, nous pourrons nous dispenser de ce pensum chaque fois que nous passerons par le BASIC pour intervenir directement en mémoire, grâce aux fonctions PEEK et POKE.

La fonction PEEK permet de lire une cellule mémoire sans pour autant affecter son contenu. Nous nous en sommes déjà servi, à la *page 120* de « Pilotez votre ZX-81 », lorsque nous avons entrepris d'inspecter le contenu de la ROM. Le mot-clé PEEK suivi du numéro décimal d'une quelconque cellule mémoire peut être introduit partout dans une instruction ou une commande, exactement comme une constante numérique ou une variable. Quelques exemples :

```
PRINT PEEK 16417
LET A = PEEK 16417
FOR F = PEEK 16417 TO PEEK 16508 STEP PEEK 16507
etc.
```

La fonction POKE est à utiliser avec plus de circonspection, car elle « écrit » en mémoire, en *écrasant* le contenu précédent de la cellule mémoire spécifiée.

Vérifions cela au niveau de la cellule n° 16417, laissée par Sinclair à la discrétion de l'utilisateur :

– faisons PRINT PEEK 16417 puis NEWLINE, ce qui doit imprimer 0, cette cellule étant normalement vide,

– continuons avec : POKE 16417, 222, puis faisons encore :

PRINT PEEK 16417, ce qui doit cette fois imprimer 222.

L'expérience peut être poursuivie avec d'autres valeurs au gré de chacun.

Si nous rééditons la tentative avec l'adresse 1018, nous pourrons constater que tous les POKE du monde ne convaincront pas cette cellule d'adopter un contenu différent de 0. En effet, cette cellule fait partie de la mémoire morte (ROM) qui n'autorise (heureusement pour l'interpréteur !) que la lecture.

Par contre, essayons d'entrer en machine un programme, quel qu'il soit, puis renouvelons les opérations précédentes avec l'adresse mémoire 16509.

Nous constatons que, après avoir mis l'octet 222 dans la cellule n° 16509, le programme semble avoir disparu de l'intérieur de l'ordinateur ! Son listage comme son lancement échouent régulièrement.

Faisons alors : POKE 16509,0, et tout rentrera dans l'ordre.

Deux enseignements sont à tirer de cette expérience : le premier est que la fonction POKE ne doit pas être employée les yeux fermés et le second, qu'il nous reste beaucoup de choses à apprendre sur le fonctionnement profond du ZX-81 !

Mais revenons à notre *figure 2-1,* dont nous sommes à présent capables de comprendre toutes les indications.

Les adresses 0 à 8191 correspondent à la mémoire inaltérable (ROM) qui contient diverses données nécessaires au fonctionnement du ZX-81, et en particulier l'interpréteur BASIC, complété par différents tableaux, dont le plus intéressant est certainement le *générateur de caractères,* occupant les cellules n° 7680 à 8191, et dont nous allons présenter bientôt une application pratique. Suit une zone mystérieuse comprise entre 8192 et 16383, réputée inutilisée, mais sur laquelle des fonctions PEEK donnent tout de même des résultats. Certaines adresses de cette zone sont utilisées par des

accessoires du marché britannique, et les tentatives de POKE ne se traduisent par aucun résultat visible (1).

Le domaine compris entre 16384 et 16508 appartient aux fameuses *variables système* utilisées par le Z-80, mais sur lesquelles des PEEK ou des POKE ne sont pas sans intérêt. Le manuel Sinclair donne la liste complète de ces variables, dont les plus intéressantes sont, à notre avis, DFILE, VARS, ELINE et *RAMTOP*, que nous avons déjà utilisée sans le savoir pour « déconnecter » logiciellement le module 16 K.

Ces quatre variables (contenues chacune dans *deux octets* consécutifs dans l'ordre OMS, OPS) représentent des *adresses mémoire* correspondant aux limites de zones mémoire utilisées à des fins bien précises par la machine.

Par exemple, un programme BASIC chargé en machine se trouve implanté à partir de l'adresse 16509 et occupe forcément une place dépendant de sa longueur, à concurrence de la capacité mémoire disponible. La machine charge automatiquement dans la variable système DFILE (OMS en 16396 et OPS en 16397) *l'adresse* de la première cellule mémoire suivant la fin du programme. Il se trouve que cette cellule correspond au début du fichier d'affichage, lequel se termine par une cellule précédant immédiatement celle dont le numéro est rangé, en deux octets, dans la variable système VARS, et qui constitue le premier octet de la zone dans laquelle la machine stocke les variables BASIC.

Après la case mémoire repérée par ELINE se trouve une zone dans laquelle le Z-80 travaille librement, qu'il serait de peu d'intérêt de détailler à ce stade de notre exploration.

Sachons seulement que cette zone machine peut s'étendre jusqu'au dernier octet disponible compte tenu de la capacité mémoire dont est équipé le ZX-81 (17407 pour 1 K, 32767 pour 16 K).

Ce dernier octet étant repéré grâce à la variable système RAMTOP, qui conserve son adresse sur deux octets (dont l'OMS est souvent 0), il est possible de fixer une limite à l'ardeur du Z-80 quant à l'occupation de cette partie de la mémoire.

---

(1) Les 8 K-octets de cette zone sont effectivement inutilisés, mais la conception particulière du ZX-81 fait que des PEEK lancés dans cette partie de la mémoire viennent lire certaines adresses de la ROM. Les accessoires utilisant ces adresses possèdent un petit circuit corrigeant ce défaut, sans inconvénient en temps normal.

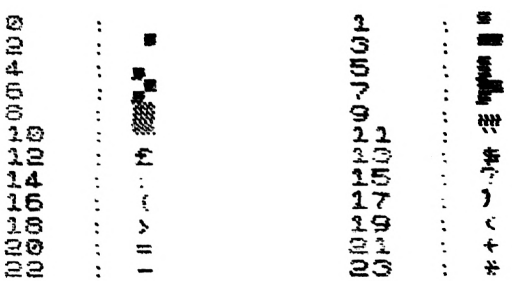
Il suffit pour ce faire de lancer une commande POKE initialisant RAMTOP à une valeur inférieure à celle d'origine. Toute la zone mémoire placée au-delà de l'octet désigné par la nouvelle valeur de RAMTOP ne pourra plus être surchargée par la machine (ni même sauvegardée sur cassette). Seules des fonctions POKE ou PEEK, ou bien sûr des instructions en langage machine, permettront d'accéder à cette zone réservée.

C'est pour cette raison que POKE 16389,68 réserve suffisamment de mémoire pour anéantir complètement le surcroît de capacité introduit par l'adjonction du module 16 K.

Remarquons que la « lecture » de RAMTOP peut servir à vérifier la présence ou l'absence du bloc 16 K. Il existe en effet des programmes pouvant très bien être chargés dans 1 K-octet mais exigeant davantage de capacité mémoire pour fonctionner (ce livre en contient des exemples).

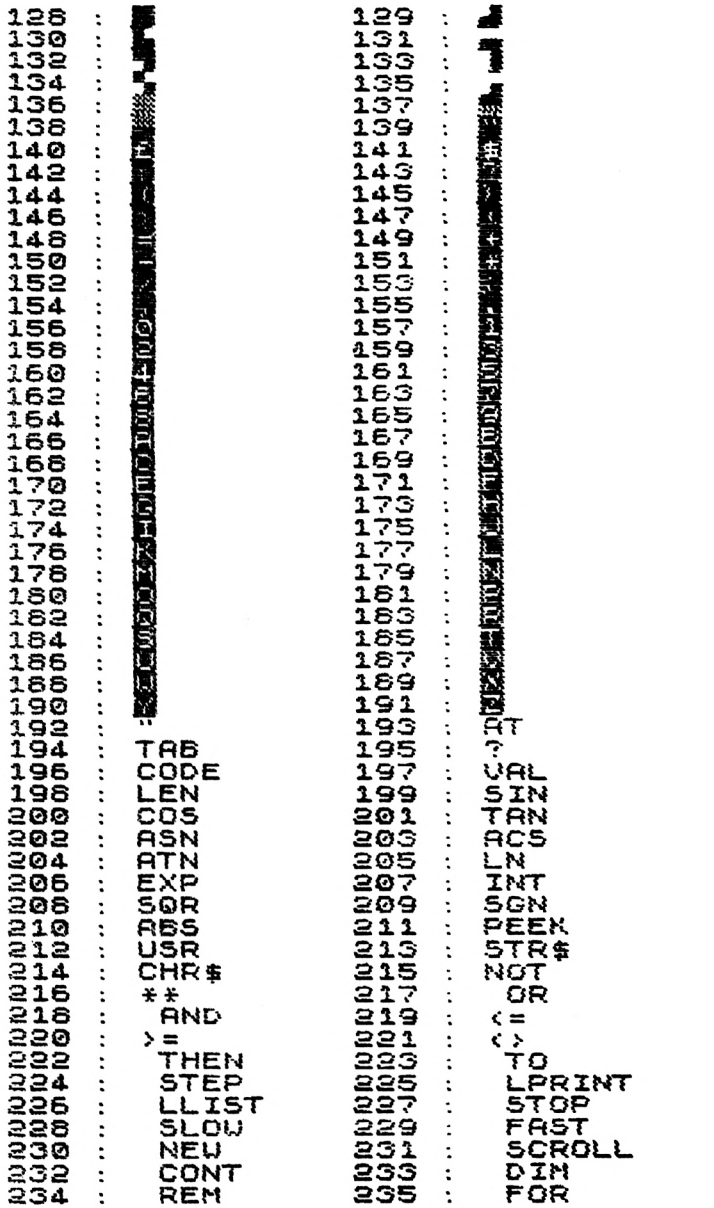
Nous reviendrons ultérieurement sur la manipulation de ces variables système, l'essentiel pour le moment étant de connaître leur existence et surtout de bien cerner les principales subdivisions de la mémoire du ZX-81.

Nous avons jusqu'à maintenant traité les octets en tant que nombres décimaux de 0 à 255, mais bien des significations différentes peuvent leur être attribuées, en particulier celle d'instructions en *langage machine,* qui seront étudiées plus loin, ou encore de *caractères.* Chaque caractère alphabétique, numérique, graphique, signe de ponctuation, ou même mot-clé, est traité sous la forme d'un seul octet. La *figure 2-4* donne les correspondances entre tous les caractères disponibles sur le ZX-81 et les octets correspondants. Signalons que, comme le jeu de caractères du ZX-81 contient moins de 256 caractères différents, les octets non attribués se voient affecter le signe ?.



(suite au verso)

```
INKEY$
RND
PI
```



(suite au verso)

```
236 : GOTO      237 : GOSUB
238 : INPUT     239 : LOAD
240 : LIST      241 : LET
242 : PAUSE     243 : NEXT
244 : POKE      245 : PRINT
246 : PLOT      247 : RUN
248 : SAVE      249 : RAND
250 : IF        251 : CLS
252 : UNPLOT    253 : CLEAR
254 : RETURN    255 : COPY
```

Fig. 2-4. – Correspondance entre codes décimaux et caractères du ZX-81. (N.B. : tous les codes « de contrôle » sont représentés par des ?)

Le ZX-81 permet un passage facile des octets vers les caractères, et vice-versa :

Le mot-clé CODE, suivi d'un caractère présenté sous forme de chaîne *à un seul élément,* permet de faire déterminer à la machine l'octet correspondant à ce caractère.

Par exemple, PRINT CODE « A » donnera le résultat 38.

Inversement, la fonction CHR$, suivie d'un nombre compris entre 0 et 255, construira la chaîne à un seul élément contenant le caractère correspondant à cet octet.

Par exemple, PRINT CHR$ 38 imprimera un A.

Application pratique : un programme générateur de caractères géants

Le programme proposé ici exploite les données contenues dans le tableau « générateur de caractères » de la ROM. Ce tableau sert normalement de modèle à l'interpréteur BASIC pour le tracé des caractères existants sur le ZX-81. C'est uniquement aux données contenues dans ce tableau que le ZX doit de travailler en lettres majuscules et non minuscules. Bref, cette imposante masse d'octets n'est rien d'autre que le « trace-lettres » du ZX-81. Voici, par exemple, comment la forme de la lettre A est définie au niveau du générateur de caractères :

Adresse	Octet	Code décimal
7984	00000000	000
7985	00111100	060
7986	01000010	066
7987	01000010	066
7988	01111110	126
7989	01000010	066
7990	01000010	066
7991	00000000	000

Il est important de noter que chaque caractère occupe huit octets et que les tables correspondant à chaque caractère se suivent, en ROM, *dans l'ordre croissant des codes décimaux* des caractères. Ainsi, une opération très simple permet de déterminer l'adresse du premier octet de la table d'un caractère dont on connaît le code.

Bien que ce tableau soit là uniquement en vue d'une utilisation par l'interpréteur BASIC, rien n'empêche de tenter d'en tirer parti d'une autre façon, par exemple en essayant de tracer des caractères géants ou déformés.

Le principe des caractères géants a été utilisé par Sinclair dans un programme présenté dans le manuel de l'imprimante. Son *extrême* lenteur (essayez-le donc !) est rédhibitoire pour n'importe quelle application pratique. Son auteur ne l'a-t-il pas en effet baptisé avec humour (anglais) « Slowest program in the world » ? Et pourtant les applications d'un programme capable de tracer de très grands caractères sont nombreuses.

Notre programme permet de construire, avec une rapidité raisonnable, des textes composés dans une immense variété de caractères, différant les uns des autres par la taille, la forme plus ou moins allongée ou étirée, et le graphisme. La *figure 2-5* donne quelques exemples, nullement limitatifs, de formes pouvant être obtenues. La longueur des textes pouvant être composés est illimitée, ou presque, puisque le programme propose, au choix, de faire défiler le texte verticalement et sans fin sur l'écran ou d'imprimer tous ses caractères, à la suite, sur le ruban de l'imprimante.

Signalons que le plus grand modèle de caractère pouvant être construit par le programme (dimensions 4 × 4) remplit entièrement l'écran TV ou la largeur du papier de l'imprimante !

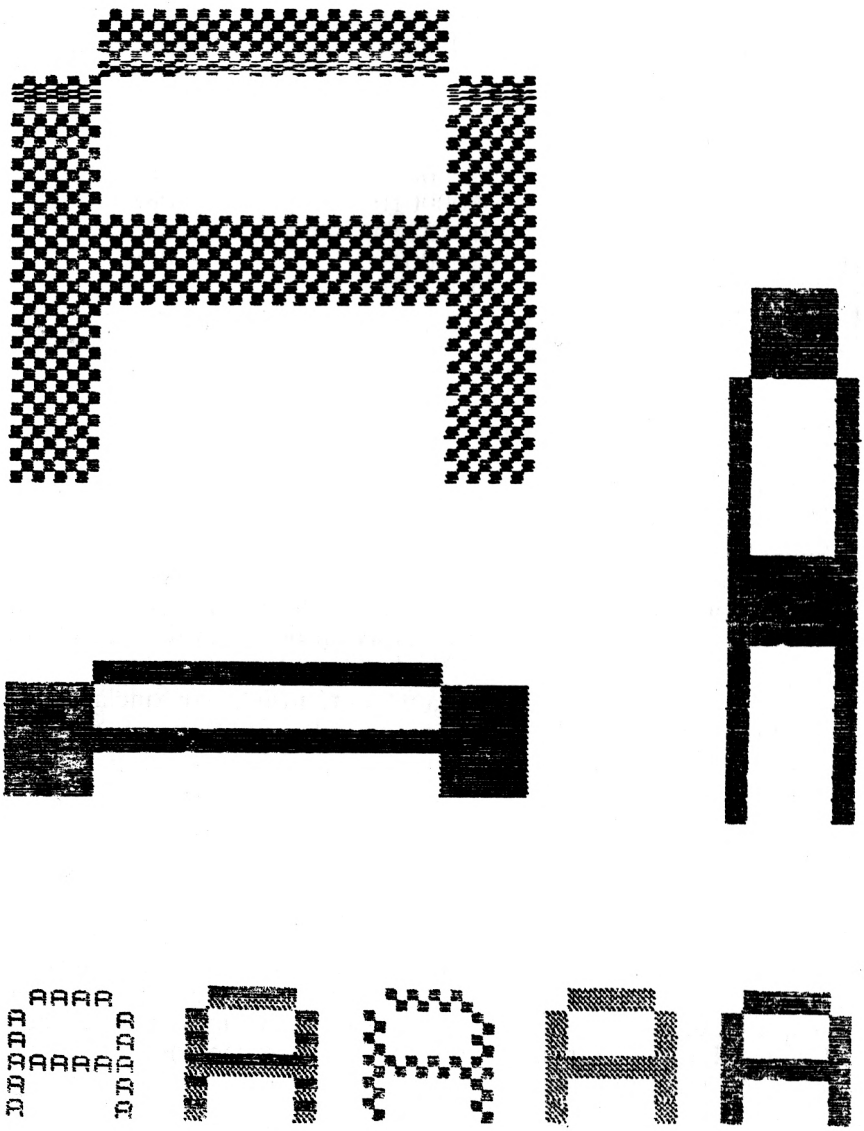


Fig. 2-5.

Nous ne nous étendrons pas sur les détails d'écriture du programme, notre but n'étant plus, dans cet ouvrage, de faire de la programmation BASIC. Son étude détaillée, assez complexe, pourra cependant constituer un bon exercice pour nos lecteurs. Sa compréhension ne réclame aucune notion autre que celles développées dans « Pilotez votre ZX-81 » et dans le début de cet ouvrage.

Tout au plus peut-il être commode de disposer des points de repère suivants (se reporter à la *figure 2-6)* :

```
   5 REM "COMPOSITION"
  10 PRINT "HAUTEUR DES CARACTER
ES ?"
  15 GOSUB 1500
  20 INPUT HA
  22 CLS
  25 PRINT "LARGEUR DES CARACTER
ES ?"
  28 GOSUB 1500
  30 INPUT LA
  35 CLS
  40 PRINT "TYPE DE CARACTERE ?"
  41 PRINT
  42 PRINT
  45 PRINT "▌ FRAPPER 128",
  50 PRINT "▒ FRAPPER 8"
  52 PRINT
  55 PRINT "▐ FRAPPER 134",
  60 PRINT "▗ FRAPPER 6"
  62 PRINT
  65 PRINT "▓ FRAPPER 135",
  67 PRINT "▓ FRAPPER 137"
  90 PRINT
  91 PRINT "LE CARACTERE LUI-MEM
E,FRAPPER 0"
  92 PRINT
  95 PRINT "       PUIS NEWLINE"
 100 INPUT CH
 110 CLS
 120 PRINT "POUR UN DEFILEMENT C
ONTINU"
 125 PRINT "FRAPPER D"
 130 PRINT
 135 PRINT "POUR UNE IMPRESSION,
FRAPPER P"
 140 LET MD=0
 145 IF INKEY$="D" THEN GOTO 160
 150 IF INKEY$="P" THEN GOTO 170
 155 GOTO 145
 160 LET MD=1
 170 CLS
 171 PRINT "FRAPPEZ VOTRE TEXTE"
 172 PRINT
```

```
 175 PRINT "     PUIS NEWLINE"
 180 INPUT T$
 190 CLS
 200 FOR M=1 TO LEN T$
 225 LET ADR=7680+8*CODE T$(M)
 235 LET L=0
 236 SCROLL
 238 LET A$=""
 239 FOR F=1 TO LA
 240 LET A$=A$+"          "
 241 NEXT F
 245 LET N=PEEK (ADR+L)
 250 LET W=128
 260 FOR Q=1 TO (8*LA) STEP LA
 270 LET E=INT (N/W)
 280 IF N>=W THEN LET N=N-W
 290 LET W=W/2
 294 FOR F=1 TO LA
 295 IF E=1 AND CH=0 THEN LET A$
(Q+F-1)=T$(M)
 296 IF E=1 AND NOT CH=0 THEN LE
T A$(Q+F-1)=CHR$ CH
 297 NEXT F
 300 NEXT Q
 310 FOR F=1 TO HA
 312 PRINT A$
 313 IF MD=0 THEN LPRINT A$
 314 SCROLL
 315 NEXT F
 320 LET L=L+1
 330 IF L<8 THEN GOTO 238
 345 NEXT M
 350 IF MD=1 THEN GOTO 1450
1400 STOP
1450 SCROLL
1455 SCROLL
1456 GOTO 200
1500 PRINT
1550 PRINT "(FRAPPER 1,2,3,OU 4,
 ET NEWLINE)"
1600 RETURN
2000 REM COPYRIGHT 1982
```

Fig. 2-6.

- *ligne 245 :* prélèvement en ROM des octets qui représentent le caractère (un octet à la fois) ;
- *lignes 250 à 290 :* conversion en binaire du code décimal de l'octet prélevé ;
- *lignes 295 et 296* (l'une ou l'autre selon les options prises au départ) : insertion dans une chaîne « blanche » de « noirs » (en

fait, tout caractère autorisé) aux emplacements correspondant au « 1 » de l'octet prélevé ;

- *lignes 310 à 315 :* impression de la chaîne achevée, un nombre de fois égal à la hauteur (1 à 4) voulue du caractère (sur écran ou papier selon valeur de MD) ;
- *lignes 320 à 350 :* gestion du « bouclage » du programme sur lui-même, selon que l'édition s'effectue sur écran (bouclage perpétuel) ou sur papier (arrêt à la fin de l'impression du texte).

Ceux de nos lecteurs qui pourraient éprouver des difficultés à dégager la ligne directrice de ce programme aux multiples imbrications pourront se reporter aux *figures 2-7 et 2-9.* Celles-ci reproduisent les programmes intermédiaires que nous avons écrits lors de la mise au point progressive du programme définitif. Le premier se

```
  5 REM "CARACTERES"
  6 PRINT "TEXTE ?"
  7 INPUT T$
  8 FOR M=1 TO LEN T$
 15 LET C$=T$(M)
 25 LET ADR=7680+(CODE C$+8)
 35 LET L=0
 36 SCROLL
 38 LET A$="                "
 40 LET N=PEEK (ADR+L)
 50 LET W=128
 60 FOR Q=1 TO 8
 70 LET E=INT (N/W)
 80 IF N>=W THEN LET N=N-W
 90 LET W=W/2
 95 IF E=1 THEN LET A$(Q)="■"
100 NEXT Q
110 PRINT "          ";A$
120 LET L=L+1
130 IF L<8 THEN GOTO 36
145 NEXT M
150 REM COPYRIGHT 1982
```

Fig. 2-7 – Programme permettant de tracer des caractères de hauteur et de largeur égales à 1 (limité à deux caractères avec 1 K de RAM, sauf utilisation de la commande CONT).

contente de tracer des caractères de dimension 1 × 1, en noir uniquement, alors que le second comprend la modification permettant d'atteindre la taille 2 × 2. Un autre avantage de ces programmes est qu'ils peuvent être chargés dans seulement 1 K-octet de RAM, tout en provoquant rapidement une panne de mémoire lorsqu'ils sont lancés en l'absence de bloc 16 K. Ils constituent donc d'excellents exemples pour les manipulations de capacité mémoire décrites à la fin du *chapitre 1*.

```
  5 REM "CARACTERES"
  6 PRINT AT 0,0;"TEXTE  ?"
  7 INPUT T$
  8 FOR M=1 TO LEN T$
 15 LET C$=T$(M)
 25 LET ADR=7680+(CODE C$+8)
 35 LET L=0
 36 SCROLL
 38 LET A$="                              "
 40 LET N=PEEK (ADR+L)
 50 LET W=128
 60 FOR Q=1 TO 16 STEP 2
 70 LET E=INT (N/W)
 80 IF N>=W THEN LET N=N-W
 90 LET W=W/2
 95 IF E=1 THEN LET A$(Q)="■"
 96 IF E=1 THEN LET A$(Q+1)="■"
100 NEXT Q
110 PRINT "          ";A$
111 SCROLL
112 PRINT "          ";A$
120 LET L=L+1
130 IF L<8 THEN GOTO 36
145 NEXT M
150 REM COPYRIGHT 1982
```

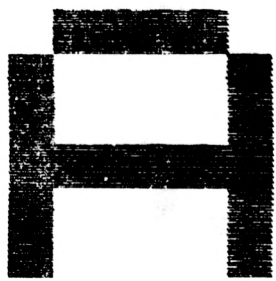


Fig. 2-8. – Programme permettant de tracer des caractères de hauteur et de largeur égales à 2 (s'arrête au milieu du premier caractère avec 1 k de RAM).

Exploitation de la mémoire programme de la machine

Nous avons appris sur la *figure 2-1* que les programmes BASIC entrés en machine sont stockés en mémoire à partir de la cellule n° 16509, et jusqu'à une cellule dépendant de la longueur du programme, mais dont l'adresse peut être obtenue en faisant :

PRINT (PEEK 16396 + 256 × PEEK 16397 (lecture de DFILE)

Cela permet déjà de calculer la longueur (en octets) d'un programme présent en machine, avec une meilleure précision qu'au moyen d'un chronométrage de son enregistrement sur cassette. La machine peut imprimer directement le nombre d'octets occupés par le programme (abstraction faite des données) grâce à la *commande* suivante :

PRINT (PEEK 16396 + 256 × PEEK 16397) – 16509

Cette détermination est intéressante, car elle permet, entre autres, d'évaluer les gains de place mémoire introduits par les diverses variantes d'un même programme (notamment, affectation à une variable d'une constante numérique fréquemment utilisée). Dans certains cas, il est possible, par des économies successives, de rendre un programme, initialement trop long, compatible avec 1 K de RAM.

Il est encore plus instructif d'examiner en détail la façon dont les programmes BASIC sont stockés en mémoire par l'interpréteur. La *figure 2-9* propose un court programme qui examine lui-même la façon dont il est mémorisé !

Une boucle FOR-NEXT, dont l'amplitude est égale à la longueur du programme, fait imprimer à la machine les *caractères* correspondant à tous les octets de la zone programme (fonction CHR$).

On constate que l'essentiel du texte des instructions se retrouve assemblé à la file, mais que des caractères apparemment farfelus (symboles graphiques, chiffres, lettres, espaces et, même, mots-clé) se trouvent incorporés ici ou là.

Ces caractères correspondent à des octets *qui ne sont pas* la traduction de caractères des lignes du programme, mais bien des

```
  10 FOR F=16509 TO PEEK 16396+2
56*PEEK 16397-1
  20 PRINT CHR$ PEEK F;
  30 NEXT F
  40 REM COPYRIGHT 1982
```

```
 "U  FOR F=16509?   IF    TO PEEK
 16396?  /   +256?     *PEEK 1639
7?  ,   -1?         ? =  PRINT CHR$
PEEK F;? 2   NEXT F? C(  REM COP
YRIGHT 1982?
```

Fig. 2-9.

indications de service, telles que : numéro de ligne sur deux octets, longueur de la ligne (sur deux octets également), etc.

A la *figure 2-10,* tous les octets sont représentés sous leur forme décimale, ce qui fait disparaître cet apparent désordre.

```
  10 FOR F=16509 TO PEEK 16396+2
56*PEEK 16397-1
  20 PRINT PEEK F;"/";
  30 NEXT F
  40 REM COPYRIGHT 1982
```

```
0/10/59/0/235/43/20/29/34/33/28/
37/126/143/0/250/0/0/223/211/29/
34/31/37/34/126/143/0/24/0/0/21/
30/33/34/126/137/0/0/0/0/23/211/
29/34/31/37/35/126/143/0/26/0/0/
22/29/126/129/0/0/0/0/118/0/20/5
/0/245/211/43/25/11/24/11/25/118
/0/30/3/0/243/43/118/0/40/16/0/2
34/40/52/53/62/55/46/44/45/57/0/
29/37/36/30/118/
```

Fig. 2-10.

Enfin, sur la *figure 2-11,* le programme proposé imprime à la fois les octets traduits en décimal et, entre parenthèses, le caractère ZX-81 correspondant. Avec ces trois représentations complémentaires, nous sommes armés pour examiner confortablement le détail du traitement des instructions BASIC.

```
  10 FOR F=16509 TO PEEK 16396+2
56*PEEK 16397-1
  20 PRINT PEEK F;"(";CHR$ PEEK
F;")";
  30 NEXT F
  40 REM COPYRIGHT 1982
```

```
0( )10(")59(V)0( )235( FOR )43(F
)20(=)29(1)34(6)33(5)28(0)37(9)1
26(?)143(■)0( )250( IF )0( )0( )
223( TO )211(PEEK )29(1)34(6)31(
3)37(9)34(6)126(?)143(■)0( )24(/
)0( )0( )21(+)30(2)33(5)34(6)126
(?)137(■)0( )0( )0( )0( )23(*)21
1(PEEK )29(1)34(6)31(3)37(9)35(7
)126(?)143(■)0( )26(,)0( )0( )22
(-)29(1)126(?)129(■)0( )0( )0( )
0( )118(?)0( )20(=)17( )0( )245(
 PRINT )211(PEEK )43(F)25(;)11("
)16(()11(")25(;)214(CHR$ )211(PE
EK )43(F)25(;)11(")17( )11(")25(
;)118(?)0( )30(2)3(■)0( )243( NE
XT )43(F)118(?)0( )40(C)16(()0(
)234( REM )40(C)52(O)53(P)62(Y)5
5(R)46(I)44(G)45(H)57(T)0( )29(1
)37(9)36(8)30(2)118(?)
```

Fig. 2-11.

Le premier octet de la zone programme (adresse 16509) est un 0, correspondant donc à un espace. Cet octet est quasiment toujours à zéro, car il représente le poids fort (OPS) du numéro de la première ligne du programme. Ce n'est donc que pour un numéro de première ligne supérieur à 255 (bien peu probable !) que cet octet pourrait prendre une valeur différente de zéro.

Par ailleurs, le numéro de ligne ne devant pas dépasser 9999, l'octet n° 16509 ne doit jamais dépasser la valeur 39 (voir *figure 2-3*). Ainsi, lorsque nous avons forcé cet octet à 222, au *chapitre 2,* en faisant POKE 16509,222, nous avons créé un numéro de ligne invraisemblable, entraînant l'arrêt des programmes machine chargés de lister ou lancer le programme BASIC. Plus loin dans ce chapitre, nous découvrirons une intéressante application de cette particularité du ZX-81.

L'octet suivant contient le poids faible (OMS) du numéro de ligne, soit bien entendu 10, puisque OPS = 0.

Les deux octets qui suivent contiennent, mais cette fois dans l'ordre inverse OMS suivi de OPS, la *longueur* de la ligne de programme, c'est-à-dire le nombre d'octets qu'elle occupe dans la mémoire programme, non compris les quatre octets « de service », dont nous venons de traiter, mais y compris l'octet 118 (code de NEWLINE) qui termine systématiquement toute ligne BASIC.

Dans notre exemple, la ligne 10 occupe 59 octets. Ce chiffre paraît démesuré mais se justifie par le fait que cette ligne contient cinq constantes numériques (16509, 16396, 256, 16397 et 1). Le ZX-81 mémorise deux fois chaque constante : une fois à raison d'un octet par caractère (pour les listages) plus une fois sous forme binaire à virgule flottante monopolisant cinq octets. Ces octets donnent, si on leur applique la fonction CHR$, des symboles absolument quelconques, comme le IF de la troisième ligne de la *figure 2-11* pour le moins inattendu !

Nous laissons à nos lecteurs le soin de poursuivre, selon les mêmes méthodes, l'analyse du contenu de la mémoire programme, mais nous ne leur conseillons pas d'approfondir la question du double stockage des constantes numériques, le décodage de la représentation en virgule flottante (mantisse et exposant) faisant appel à des notions mathématiques passablement complexes.

Retenons surtout que ces constantes occupent énormément de place en mémoire et qu'il convient donc de ne pas en abuser !

Nous allons nous rendre compte qu'il peut être très intéressant de manipuler au moyen de POKE les octets de service placés en tête des lignes BASIC dans la mémoire de programmes.

Comme il est extrêmement fastidieux de se lancer, par le calcul, à la recherche d'une ligne donnée, nous avons écrit un petit programme qui, chargé en machine *avant* la frappe du programme principal, fournit en un tournemain les adresses des quatre octets de service de la ligne spécifiée.

La compréhension profonde du fonctionnement de ce programme, fourni à la *figure 2-12,* est un excellent exercice, parfaitement à la portée de nos lecteurs ayant correctement assimilé ce qui précède.

Comme bien des programmes « utilitaires » (appelés *toolkits* par nos amis anglais), celui-ci occupe les dernières lignes disponibles avant 9999, est séparé des lignes précédentes par une ligne STOP et se fait donc oublier tant qu'il n'est pas appelé par une commande GOTO 9200.

```
9100 STOP
9200 REM RECHERCHE DE LIGNE
9250 CLS
9300 LET W=16509
9500 PRINT "NUMERO DE LIGNE ?"
9510 INPUT NL
9520 CLS
9530 GOSUB 9700
9540 IF A<NL THEN LET W=W+B
9550 IF A<NL THEN GOTO 9530
9555 IF A<>NL THEN GOTO 9200
9560 GOTO 9800
9700 LET A=PEEK (W+1)+256*PEEK W
9710 LET B=4+PEEK (W+2)+256*PEEK
 (W+3)
9720 RETURN
9800 PRINT "LIGNE NO ";NL;" DU P
ROGRAMME"
9810 PRINT
9820 PRINT "ADRESSE DU NO :"
9825 PRINT
9830 PRINT W+1;" (BMS)",W;" (BPS
)"
9840 PRINT
9850 PRINT "ADRESSE DE LA LONGUE
UR :"
9855 PRINT
9860 PRINT W+2;" (BMS)",W+3;" (B
PS)"
9900 REM COPYRIGHT 1982
```

Fig. 2-12.

La suite n'est qu'une formalité, puisqu'un dialogue en langage clair s'instaure entre la machine et l'utilisateur.

Application pratique : protection de programmes par mot de passe

Nous avons vu plus avant comment l'introduction forcée d'un numéro de ligne hors intervalle pouvait, apparemment, faire « disparaître » un programme. En fait, chacun aura compris que, mis à part l'octet modifié, rien n'a changé en mémoire, mais que le programme est seulement devenu « inaccessible ». Il peut encore, cependant, être chargé sur cassette, puis réentré en machine, tout en restant « paralysé » tant que l'octet en cause n'aura pas été ramené à une valeur vraisemblable.

Tout ce processus peut permettre de « protéger » des programmes enregistrés sur cassette ou transmis par téléphone ou radio. Il suffirait à l'utilisateur autorisé de faire POKE 16509,0 pour « redonner la vie » au programme.

Ce « mot de passe » deviendrait cependant vite un « secret de polichinelle », et c'est pourquoi nous avons étudié un programme autorisant une certaine liberté dans le choix du code et dont le mode d'emploi est plus commode.

Le programme de la *figure 2-13* doit être entré en machine avant la frappe du programme à protéger (ou après, mais obligatoirement au clavier et sous réserve qu'il n'existe pas de chevauchements entre les numéros des lignes).

```
    1 PRINT "FRAPPEZ LE CODE"
    2 PRINT
    3 PRINT "PUIS NEWLINE"
    4 INPUT CODE
    5 POKE CODE,0
    6 CLS
    7 REM
   10 REM DEBUT DU PROGRAMME
9995 STOP
9996 CLS
9997 POKE 16599,255
9998 PRINT AT 10,0;"LE CODE SERA
 16598"
9999 REM COPYRIGHT 1982
```

Fig. 2-13.

Le programme à protéger doit commencer à la ligne 10 et finir avant la ligne 9994. La protection est mise en place en lançant une commande GOTO 9996, à laquelle la machine répond en indiquant que le numéro de code sera 16598. A partir de cet instant, tout listage tournera court au niveau de la ligne 7, et un RUN ne pourra lancer que le programme compris entre les lignes 1 et 7. Ce programme réclame le numéro de code, qui est en fait l'adresse de l'OPS du numéro de la ligne 10. Si un mauvais numéro est donné, un 0 viendra écraser le contenu de la cellule d'adresse correspondante, ce qui risque fort de détruire à jamais le programme que l'on cherche à « délivrer ».

Ce n'est que lorsque le 255 de la cellule n° 16598 aura été

remplacé par un 0 que le programme se dévoilera et pourra alors soit être lancé par GOTO 10, soit être listé par LIST 10.

Notons que le numéro de code peut être augmenté d'une unité chaque fois qu'un caractère (quelconque) est inséré dans l'instruction REM de la ligne 7.

Si nous faisons par exemple :

```
7 REM??
```

le numéro de code devient 16600.

Cette méthode de protection ne prétend pas constituer « l'arme absolue » (qui n'existe d'ailleurs pas), mais au moins constitue-t-elle un moyen de compliquer singulièrement l'accès à certains programmes plus ou moins confidentiels, tels que données bancaires, ou déconseillés à certains publics : nous avons en effet pu voir, outre-Manche, des programmes ZX-81 « classés X » et ne devant donc pas être mis entre toutes les mains !!

Interventions sur le fichier d'affichage

Le « fichier d'affichage » est la zone de la mémoire dont les adresses limites sont contenues dans les variables système DFILE et VARS, et dans laquelle la machine construit l'image devant apparaître sur l'écran TV.

Insistons sur le fait que lorsque le ZX-81 travaille sans bloc d'extension mémoire, le fichier d'affichage fait l'objet de réductions d'encombrement automatiques qui rendent fort périlleuses toutes tentatives d'intervention directe par POKE ou même PEEK.

Les idées que nous allons soumettre à nos lecteurs sont donc destinées à être mises en œuvre à l'aide d'un bloc 16 K RAM. Il faut distinguer plusieurs variantes du fichier d'affichage : le fichier le plus commode à manipuler est celui dont l'utilisateur a directement conscience, à savoir un groupe de 22 lignes de 32 caractères, soit $22 \times 32 = 704$ octets.

En fait, chaque ligne de l'écran, mise en mémoire, comprend en outre un code « NEWLINE » dont l'équivalent décimal est 118.

Enfin, il existe encore deux lignes en bas de l'écran, servant à la mise au point des lignes de programme et à l'impression des curseurs ou comptes-rendus.

Application pratique : sauvegarde de l'écran

La création sur l'écran de graphismes évolués représente toujours une somme de travail non négligeable. On peut certes écrire de toutes pièces un programme construisant petit à petit le motif désiré, à l'aide d'instructions PLOT ou PRINT AT (voir *figure 1-1*), mais il s'agit là d'un travail de Romain ! Toutefois, l'avantage certain du procédé est de permettre la reconstitution du dessin autant de fois que nécessaire, dès lors que le programme a été sauvegardé sur cassette.

Il existe une méthode plus rapide pour exécuter des graphismes sur l'écran, nommée « *sketchpad* » par les Anglais, et que nous avons utilisée dans le tout premier programme de « Pilotez votre ZX-81 ». Quatre touches permettent de déplacer à volonté un « crayon électronique » laissant une trace sur l'écran. Seulement, une fois le dessin achevé, il ne reste plus qu'à le copier sur imprimante ou à le photographier, car le programme ne garde aucune trace des ordres qu'il a exécutés.

Pourtant, la mémoire de l'ordinateur possède bien une réplique fidèle de l'écran dans ce fameux fichier d'affichage ! Le jeu de deux programmes proposé par la *figure 2-14* permet de faire construire à la machine une *chaîne* de longueur respectable (704 octets), qui pourra être stockée à loisir sur une cassette et imprimée à volonté sur l'écran ou sur l'imprimante, de façon à reconstituer en une fraction de seconde le graphisme d'origine qu'il aura bien fallu se résoudre à effacer tôt ou tard !

Au niveau de ses lignes 2 et 4, le programme réserve de la place au-dessus de RAMTOP (voir *page 41*), c'est-à-dire à un endroit parfaitement protégé même contre les effacements du programme par NEW.

Le programme reprend ensuite, de la ligne 6 à la ligne 39, les opérations bien connues de construction du graphisme, à ceci près qu'il est prévu une « sortie » de la boucle GOTO 20 en actionnant la touche Z (COPY). Et, précisément, cette sortie mène à la ligne 90, à partir de laquelle on construit une chaîne A$, préalablement déclarée comme devant contenir 704 caractères (ligne 90), en explorant tour à tour les 22 lignes de 32 caractères de l'écran.

Ensuite, et à partir de la ligne 110, les codes de tous les caractères de la chaîne A$ sont mis à l'abri au-dessus de RAMTOP. Toutes ces opérations sont exécutées en mode rapide mais exigent quand même un certain temps : ne pas s'impatienter !

```
    1 REM "GRAPHIQUE"
    2 POKE 16388,0
    4 POKE 16389,125
    6 LET L=22
    7 LET N=9
    8 LET C=32
   20 IF INKEY$="5" THEN LET L=L-
1
   22 IF INKEY$="8" THEN LET L=L+
1
   24 IF INKEY$="7" THEN LET C=C+
1
   26 IF INKEY$="6" THEN LET C=C-
1
   27 IF INKEY$="9" THEN LET N=9
   28 IF INKEY$="0" THEN LET N=0
   30 IF N=9 THEN PLOT L,C
   32 IF N=0 THEN PLOT L,C
   34 IF N=0 THEN UNPLOT L,C
   35 IF INKEY$="Z" THEN GOTO 90
   39 GOTO 20
   90 DIM A$(704)
   97 LET DFILE=PEEK 16396+256*PE
EK 16397
  100 FAST
  101 FOR I=0 TO 21
  102 FOR J=1 TO 32
  103 LET A$(J+32*I)=CHR$ PEEK (D
FILE+J+33*I)
  104 NEXT J
  105 NEXT I
  107 CLS
  110 FOR F=1 TO 704
  120 POKE 32000+F,CODE A$(F)
  130 NEXT F
  140 SLOW
  145 PRINT A$
  147 LOAD "RECONSTITUTION A$"
  150 REM COPYRIGHT 1982



  200 DIM A$(704)
  205 FAST
  210 FOR F=1 TO 704
  220 LET A$(F)=CHR$ PEEK (32000+
F)
  230 NEXT F
  240 PRINT A$
```

Fig. 2-14.

Ce travail effectué, la machine passe en attente du second volet du programme que le magnétophone doit lire.

Une fois lancé par RUN, ce programme passe un certain temps à reconstruire la chaîne A$ à partir des données conservées au-dessus de RAMTOP, puis imprime A$ sur l'écran.

A ce stade, on peut effacer les lignes 200 à 230, donner si nécessaire un nouveau numéro à la ligne 240, ajouter toutes les lignes voulues : la chaîne graphique restera en mémoire et pourra être imprimée à tout instant par PRINT A$.

Cependant, *il est formellement exclu* de lancer ce nouveau programme par RUN, car cette manœuvre efface toutes les données, et donc la chaîne A$ si laborieusement constituée. La solution consiste tout simplement à démarrer par un GOTO.

En résumé, le mode d'emploi du programme est le suivant :

- charger le premier volet du programme et le lancer par RUN ;
- tracer le graphisme puis presser COPY ;
- attendre que la machine passe en chargement K7 ;
- démarrer le magnétophone, qui doit lire le second volet du programme, enregistré sous le titre « RECONSTITUTION A$ » ;
- démarrer par RUN et attendre l'impression du graphisme ;
- frapper le programme utilisant A$, le précédent programme pouvant être effacé *ligne à ligne ;*
- lancer ce nouveau programme exclusivement par GOTO ligne ;
- sauvegarder ce programme sur cassette.

Ce procédé est très pratique lorsqu'il s'agit de créer des jeux mettant en œuvre des motifs graphiques plus ou moins complexes, car il évite d'encombrer le programme avec toutes les lignes nécessaires à la construction de ces dessins. De plus, le gain de temps lors de la programmation est plus que notable !

On se souviendra que le second volet du programme laisse la machine en mode rapide et qu'il peut être nécessaire, selon les applications envisagées, de la commuter de nouveau en mode lent.

Un programme d'animation de vitrine (fig. 2-15)

Ce programme ne fait pas appel à des opérations particulières sur le contenu du fichier d'affichage, mais il en exploite à fond les

possibilités grâce à un emploi massif de la fonction SCROLL (voir *chapitre 1*).

Cela permet de faire « dérouler » lentement sur l'écran un texte même très long, préalablement entré au clavier exactement comme sur une machine à écrire.

En effet, le ZX-81 possède la déplorable habitude, lorsqu'un texte continu lui est soumis, d'en couper les mots n'importe où, au gré de ses retours à la ligne automatique. Dans ce programme, c'est à l'opérateur qu'il incombe de presser NEWLINE à temps, quitte à couper le mot lui-même selon les règles d'usage.

```
    1 REM "JOURNAL"
    2 LET B$=""
    3 PRINT "INSTRUCTIONS :"
    4 PRINT
    5 PRINT "A LA FIN DE CHAQUE L
IGNE,      PRESSEZ NEWLINE"
    6 PRINT
    7 PRINT "POUR SAUTER UNE LIGN
E,PRESSEZ    SHIFT PUIS NEWLINE"
    8 PRINT
    9 PRINT "EN FIN DE TEXTE,PRES
SEZ 2 FOIS    NEWLINE"
   10 PRINT
   11 PRINT "POUR EFFACER UN CARA
CTERE,        PRESSEZ SHIFT ET RU
BOUT"
   12 PRINT
   13 PRINT "NE DEPASSEZ JAMAIS L
A "
   14 PRINT "   LARGEUR DE L'ECRAN
"
   16 PAUSE 300
   17 CLS
   18 PRINT "FRAPPEZ VOTRE TEXTE
:"
   20 INPUT A$
   25 IF A$="" THEN GOTO 2100
   30 LET B$=B$+A$+"£"
   32 PRINT AT 0,0;"
                "
   34 PRINT AT 0,0;A$
   36 GOTO 20
   40 CLS
   41 LET L=1
   45 LET C$=""
   50 GOSUB 1005
   55 IF L>=LEN B$ THEN GOTO 2000
   60 LET C$=C$+B$(L)
```

(suite au verso)

```
  70 LET L=L+1
  80 IF CODE B$(L)=12 THEN GOSUB
1000
  85 GOTO 55
  90 LET C$=""
 100 LET L=L+1
 110 GOTO 60
1000 LET L=L+1
1005 SCROLL
1010 PRINT C$
1020 FOR F=1 TO 100
1030 NEXT F
1035 LET C$=""
1040 RETURN
2000 LET C$="
             "
2010 GOSUB 1005
2020 GOTO 41
2100 CLS
2110 PRINT "POUR STOCKER SUR K7,
PRESSER S"
2120 PRINT
2130 PRINT "POUR DEMARRER DIRECT
EMENT ."
2140 PRINT "PRESSER D"
2150 IF INKEY$="D" THEN GOTO 40
2160 IF INKEY$="S" THEN GOTO 220
0
2170 GOTO 2150
2200 CLS
2210 PRINT "DEMARREZ LE MAGNETOP
HONE"
2220 PRINT
2230 PRINT "PUIS PRESSEZ NEWLINE
"
2240 INPUT Z$
2250 SAVE "JOURNA■"
2260 CLS
2270 PRINT "ARRETEZ LE MAGNETOPH
ONE"
2280 PRINT
2290 PRINT "PUIS PRESSEZ NEWLINE
"
2300 INPUT Z$
2310 GOTO 40
3000 REM COPYRIGHT 1982
```

Fig. 2-15.

La machine construit une très longue chaîne B$ au moyen des caractères du texte frappé, mais en insérant des £ (normalement inutilisés en France) à chaque retour à la ligne. Ainsi, lors de l'impression animée, les coupures seront respectées.

A l'exception de £, tous les caractères peuvent être utilisés, comme en témoigne l'exemple de la *figure 2-16.*

Dès son lancement par RUN, le programme explique à l'opérateur son mode d'emploi détaillé, qui prévoit même une possibilité de stockage du texte composé sur une cassette en vue d'utilisations futures.

Ce programme se prête fort bien à des applications publicitaires (animation de vitrines), puisque le texte se déroule sans fin sur l'écran tant qu'un BREAK n'est pas lancé.

```
VOICI UNE DEMONSTRATION DES
POSSIBILITES DE NOTRE SYSTEME
DE PRESENTATION DE TEXTES
DOCUMENTAIRES OU PUBLICITAIRES
******************************

IL EST BIEN SUR POSSIBLE DE LUI
FAIRE AFFICHER TOUTES SORTES
DE MESSAGES EN CARACTERES TELS
QUE CEUX-CI.

CEPENDANT,IL PEUT ETRE TOUT A
FAIT INTERESSANT DE FAIRE APPEL
A DES TRES BEAUX CARACTERES
NEGATIFS LORSQU"IL S"AVERE
NECESSAIRE D"ATTIRER L"ATTEN-
TION...

BIEN SUR,TOUS LES CHIFFRES
PEUVENT ETRE REPRESENTES (1 2 3
4 5 6 7 8 9 0),ET MEME EN NEGA-
TIF SI NECESSAIRE: 1234567890

ON PEUT AUSSI UTILISER LA PLU-
PART DES SIGNES DE PONCTUATION
OU D"ARITHMETIQUE (:;?/*-+=#
<=<>>=$<>,.)

ENFIN,IL EXISTE TOUTE UNE
GAMME DE SYMBOLES DECORATIFS
NOIRS,BLANCS,OU GRIS,QUI SONT
PRECIEUX POUR LA MISE EN
VALEUR DES PARTIES D"IMAGE
LES PLUS IMPORTANTES:
```

(suite au verso)

CECI POUR SE LIMITER AUX PLUS
FREQUEMMEMT UTILISES...

FAITES DONC UN ESSAI...

Fig. 2-16.

Chapitre 3

Les interfaces
Introduction au langage machine

Les interfaces d'un ordinateur sont les dispositifs lui permettant de communiquer avec l'extérieur.

A l'origine, le ZX-81 ne pouvait communiquer avec l'extérieur que par l'intermédiaire de son clavier, de l'écran TV et des connexions prévues exclusivement pour un magnétophone à cassettes. Or un ordinateur digne de ce nom doit pouvoir communiquer avec des capteurs, des relais, des moteurs, des haut-parleurs, que sais-je encore.

La tentation est grande, pour l'électronicien disposant d'un ZX-81, de mettre à contribution le connecteur arrière qui, normalement destiné à recevoir un module 16K et/ou une imprimante, dispose de tous les bus du microprocesseur Z-80.

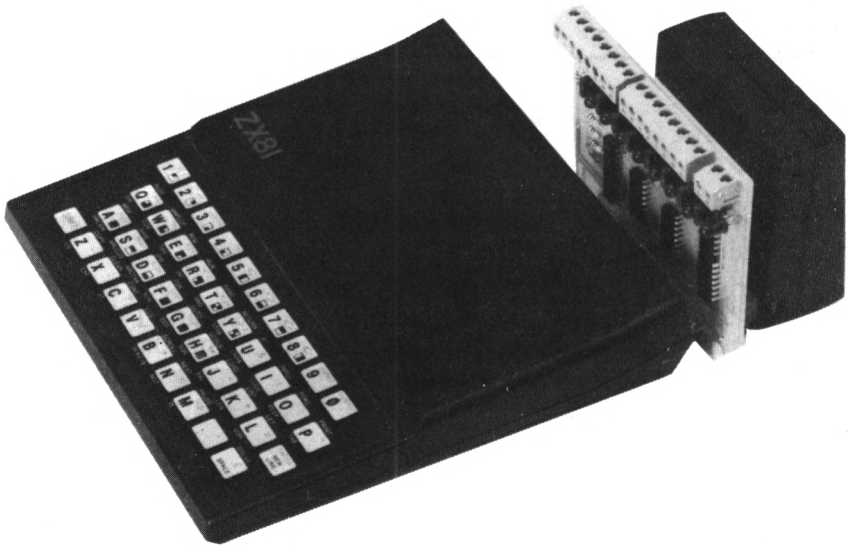
Nous avions effleuré la question à la fin de « Pilotez votre ZX-81 », en précisant toutefois que ce genre d'adaptation ne s'adressait qu'à des bricoleurs plus qu'avertis.

Nous n'avons nullement changé d'avis à ce sujet, car il est prouvé que toute fausse manœuvre au niveau de ce connecteur peut avoir les pires conséquences, allant jusqu'à la destruction quasi complète de l'ordinateur !

On sait que les pires absurdités de programmation n'endommageront jamais un ZX-81 isolé de l'extérieur, mais il faut bien noter que certaines commandes (notamment en langage machine) peuvent faire des ravages si des périphériques douteux sont branchés sur le connecteur arrière.

Faut-il donc renoncer aux innombrables possibilités offertes par les interfaces ? Certes non !

Un module d'entrées-sorties, spécialement conçu pour le ZX-81, a été mis au point en France et est disponible auprès de l'importateur de l'ordinateur pour un prix inférieur à celui de l'imprimante ou du module 16 K.



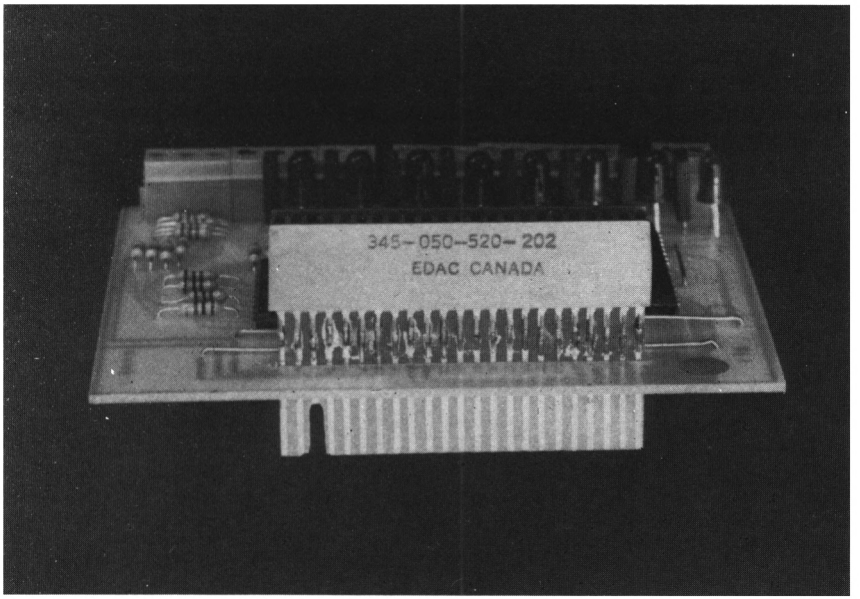
Fabriquée en France, la carte 8ES permet de faire communiquer le ZX-81 avec l'extérieur, ce qui en augmente considérablement les possibilités.

Baptisé 8ES, cet accessoire peut recevoir huit entrées « Tout ou Rien » (contacts), et commander huit sorties (transistors 2 A en collecteur ouvert).

Utilisée seule, cette carte ne permet guère que de faire clignoter des voyants, mais sa conception astucieuse autorise le raccordement de montages extérieurs des plus variés, *sans aucun risque* pour l'ordinateur et ses extensions !

Voilà, à notre avis, le plus grand intérêt de cette carte, à savoir sa vocation de *circuit tampon* protégeant le ZX-81 contre les mauvaises manipulations au niveau des interfaces.

Une fausse manœuvre en aval de la carte 8ES ne pourra guère endommager que quelques-uns des composants fort peu coûteux et très courants qu'elle utilise, sans que l'ordinateur puisse en souffrir.



Un connecteur « gigogne » rend la carte 8ES compatible avec tous les autres accessoires destinés au connecteur arrière. ATTENTION cependant à la rigidité de l'ensemble, qui devient très problématique à partir de deux prises.

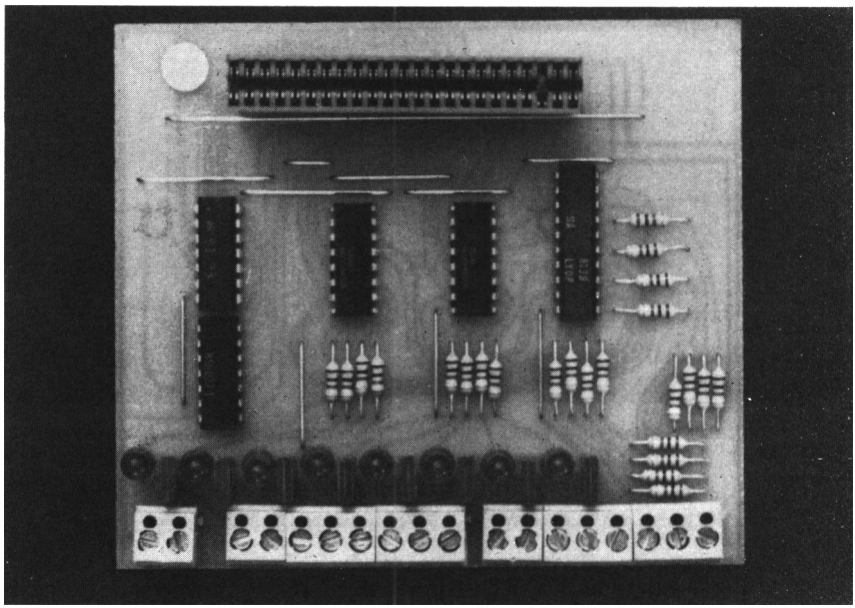
Branchée entre le ZX-81 et le module 16 K, l'imprimante, voire les deux (attention, l'assemblage est branlant et fragile !), la carte 8ES doit se faire totalement oublier tant que l'ordinateur travaille sous BASIC.

En effet, seules des commandes appropriées en *langage machine* peuvent faire sortir la carte 8ES de sa léthargie. Nous butons donc pour la première fois (et certes pas la dernière) sur un problème que le BASIC est impuissant à résoudre. L'un des atouts déterminants du langage machine est précisément d'ouvrir la porte à toute une gamme de fonctions inaccessibles sous BASIC, et en particulier les entrées-sorties.

Fort heureusement, le fabricant de la carte 8ES livre celle-ci accompagnée des *programmes en langage machine* nécessaires pour prendre la relève du BASIC provisoirement défaillant.

Ces programmes sont contenus dans un programme BASIC, appelés par celui-ci, et communiquent avec lui, ce qui fait que l'utilisateur ne se rend pratiquement pas compte que la carte d'entrée-sortie « ne parle pas le BASIC » !

En fait, nous allons voir, au chapitre suivant, qu'il est d'un grand intérêt de partir à la découverte des « routines », en langage machine, utilisées par la carte 8ES...



Un robuste bornier à vis permet toutes sortes de raccordements, sans aucun risque pour le ZX-81 lui-même.

La *figure 3-1* montre que la carte 8ES est munie d'un bornier d'entrée, d'un bornier de sortie et de deux bornes de masse. Il est important de bien noter les numéros des entrées et des sorties, car ceux-ci seront utilisés dans les programmes de commande de la

carte. Les huit entrées sont considérées comme étant au niveau « 1 » lorsqu'elles restent « en l'air », alors que, pour leur appliquer un niveau « 0 », il convient de les réunir à la masse.

Les voyants équipant les huit sorties s'allument en présence d'un niveau « 1 » et restent éteints pour les niveaux « 0 ». Ces voyants fonctionnent sur l'alimentation générale du ZX-81, mais il n'en va pas de même pour les organes extérieurs commandés par les sorties de puissance de la carte 8ES, qui devront disposer de leur propre alimentation (30 V max.). Les voyants permettent, cependant, de tester les programmes en l'absence de tout montage extérieur.

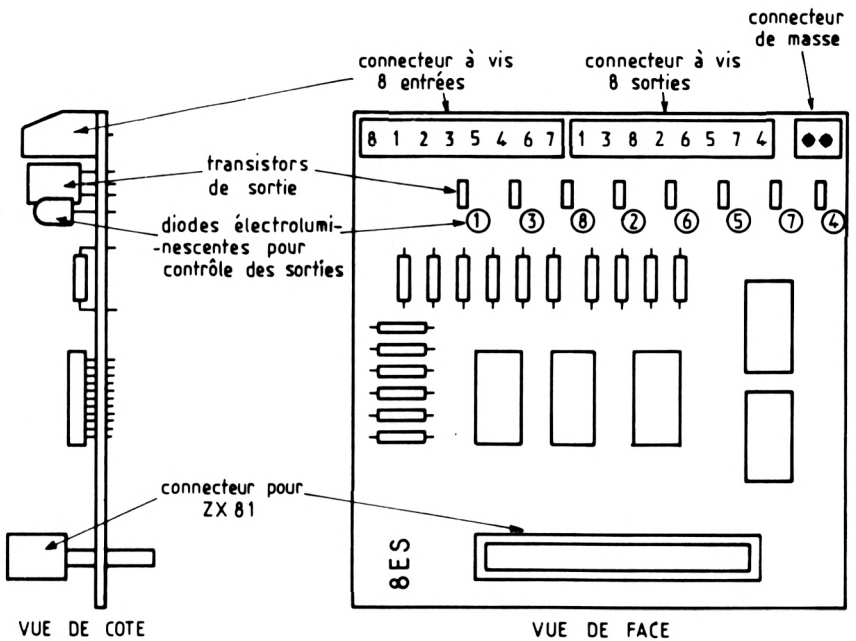


Fig. 3-1.

La *figure 3-2* donne quelques indications sur les branchements extérieurs à la carte, mais il ne s'agit, bien évidemment, que d'exemples nullement limitatifs, compte tenu des immenses possibilités du système.

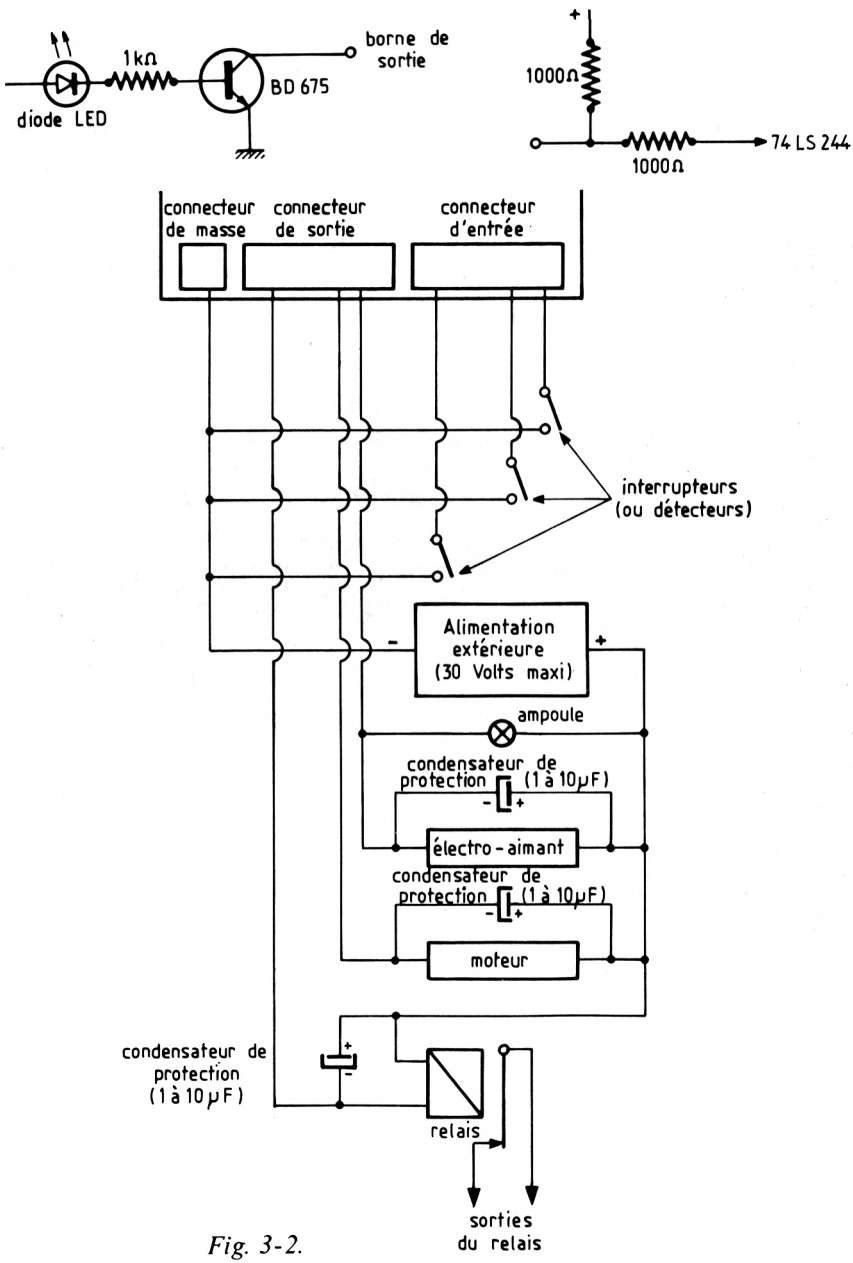

1kΩ
diode LED
BD 675
borne de sortie
+
1000Ω
74 LS 244
1000Ω
connecteur de masse
connecteur de sortie
connecteur d'entrée
interrupteurs (ou détecteurs)
Alimentation extérieure (30 Volts maxi)
-
+
ampoule
condensateur de protection (1 à 10µF)
électro-aimant
condensateur de protection (1 à 10µF)
moteur
condensateur de protection (1 à 10µF)
relais
sorties du relais

Fig. 3-2.

REMARQUE TRES IMPORTANTE

Il a été mis sur le marché différents modèles de cartes 8ES, ce qui peut rendre nécessaires de légères (mais indispensables) adaptations dans les programmes publiés ici.

Tous ces programmes ont été écrits de façon à pouvoir fonctionner sans changement sur les premières versions commercialisées dont **l'adresse** d'accès est 127.

Des cartes plus récentes possèdent quatre adresses commutables au gré de l'utilisateur : 63 (sélectionnée d'origine), 119, 95 et 111 (obtenues par modifications de câblage). Cette amélioration permet de faire fonctionner ensemble plusieurs cartes accessibles de façon indépendante. Lors d'une lecture **attentive** de la notice de la carte, **il est indispensable** que nos lecteurs identifient la version dont ils disposent avant de lancer les programmes publiés ici.

Il est facile de procéder à la modification des programmes BASIC, qui reprennent tous les instructions de la **figure 3-3.** Il suffit de remplacer la valeur 127 de la ligne 11 **et** de la ligne 14 par 63 ou toute autre adresse choisie auparavant.

En ce qui concerne les programmes écrits en **langage machine,** qui sont présentés plus loin, il suffit de remplacer l'octet 127, chaque fois qu'il apparaît dans un listing, par l'adresse choisie. On notera également que les nouvelles cartes 8ES imposent à leurs sorties, lors de la mise sous tension, des états imprévisibles, certains voyants pouvant être allumés, d'autres éteints. Le lancement d'un programme d'entrée-sortie fait tout rentrer dans l'ordre.

Le programme le plus simple pouvant être écrit pour la carte 8ES est donné à la *figure 3-3.* Il permet d'entrer ou de sortir, sous forme binaire, des octets qui, en BASIC, seront représentés en décimal et placés dans les variables IN (à l'entrée) et OUT (à la sortie).

Attention ! Lors de la frappe de ce programme, la ligne 2 devra être dactylographiée sous la forme suivante :

2 REM . . . . . (REM suivi de 14 points)

Ce n'est qu'après la première exécution du programme (après RUN NEWLINE), qu'un listage fera apparaître la ligne 2 sous l'aspect qu'elle revêt sur la figure.

Nos lecteurs ayant correctement assimilé le *chapitre 3* de cet ouvrage auront remarqué que l'adresse 16527 correspond au premier point de l'instruction REM de la ligne 2 (code 27). Comme le programme exécute justement une série de POKE à partir de cette

```
    1 GOTO 3
    2 REM COPY U■RNDPEEK ■TAN (=■
M■RNDTAN .
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q9,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 GOTO 30
   20 POKE Q7,OUT
   21 LET Q6=USR (Q8)
   22 RETURN
   25 LET Q6=USR (Q9)
   26 LET IN=PEEK (Q7)
   27 RETURN
   28 REM COPYRIGHT 1982
   29 REM "SIDENA"
```

Fig. 3-3.

adresse 16527, et ce, avec des codes différents de 27, il est normal que le texte de l'instruction REM s'en trouve modifié. Nous aurons ainsi maintes fois l'occasion, par la suite, d'utiliser des lignes REM pour « accueillir » des suites d'octets constituant nos programmes en langage machine.

Il ne faut pas tenter d'opérer une coïncidence stricte entre les caractères apparaissant après REM et les codes contenus dans les instructions POKE, car certains d'entre eux peuvent se trouver modifiés au cours des opérations d'entrée-sortie (cela explique la présence du mot-clé COPY en tête du texte suivant REM).

Sans entrer pour le moment dans les détails, notons simplement que la ligne 2 contient deux programmes en langage machine, l'un occupant les adresses 16528 à 16533, le second étant logé de 16534 à 16539. Pour les lancer, le programme BASIC exécute respectivement les instructions LET Q6 = USR 16528 ou USR 16534. Nous aurons très bientôt l'occasion de « démonter » entièrement ces « routines » en langage machine, afin d'en assimiler le fonctionnement dans le détail.

Pour l'instant, tentons de mettre en service la carte 8ES à partir du BASIC.

Puisque tous les voyants de la carte sont éteints lors de la mise sous tension de la machine, nous pensons tout naturellement à essayer de les allumer.

Pour allumer tous les voyants, il faut « sortir » l'octet 11111111, soit 255 en décimal. Donnons donc cette valeur à la variable OUT, en exécutant la *commande* suivante :

LET OUT = 255 NEWLINE

puis lançons la « routine » de sortie en faisant :

GOSUB 20 NEWLINE

Une fois tous les voyants allumés, on peut les éteindre à nouveau en renouvelant l'opération avec OUT = 0 au lieu de 255.

L'idée vient alors automatiquement d'enchaîner ces deux procédures afin de faire clignoter les voyants.

Le programme de la *figure 3-4* est écrit à cet effet.

On remarquera que ce programme ne contient pas d'instruction PAUSE, ou équivalente, en vue de fixer les durées d'allumage ou d'extinction des voyants. Ces durées sont donc uniquement déterminées, ici, par le temps nécessaire à la machine pour exécuter le programme. Il est possible d'accélérer un peu le mouvement (dans un rapport de un à quatre environ), en commutant le ZX-81 en mode rapide. Il n'en reste pas moins vrai que la lenteur relative du BASIC le rend totalement inapte à certaines tâches d'entrées-sorties, telles que, par exemple, la commande directe d'un haut-parleur, au moyen de la carte 8ES, pour lui faire émettre des tonalités audibles.

Nous verrons plus loin que l'unique solution à ce problème réside dans le recours au *langage machine* qui permettrait, par exemple, de faire clignoter les voyants de la carte 8ES à une fréquence atteignant 50 kHz !

Il n'en reste pas moins vrai que de très nombreuses applications de la carte 8ES se contentent largement des possibilités du BASIC. Le programme de la *figure 3-5* est le *programme de test* préconisé par le fabricant de la carte lorsqu'un contrôle global s'avère nécessaire. Un transfert immédiat des données d'entrée vers les sorties est exécuté logiciellement par le ZX-81. Lors du lancement du programme, tous les voyants doivent s'allumer puisque toutes les entrées sont « en l'air », mais on doit pouvoir éteindre séparément

```
  1 GOTO 3
  2 REM COPY U?RNDPEEK KTAN <=K
M?RNDTAN .
  3 LET Q7=16527
  4 LET Q8=16528
  5 LET Q9=16534
  6 POKE Q7,0
  7 POKE Q8,58
  8 POKE Q8+1,143
  9 POKE Q8+2,64
 10 POKE Q8+3,211
 11 POKE Q8+4,127
 12 POKE Q8+5,201
 13 POKE Q9,219
 14 POKE Q9+1,127
 15 POKE Q9+2,50
 16 POKE Q9+3,143
 17 POKE Q9+4,64
 18 POKE Q9+5,201
 19 GOTO 30
 20 POKE Q7,OUT
 21 LET Q6=USR (Q8)
 22 RETURN
 25 LET Q6=USR (Q9)
 26 LET IN=PEEK (Q7)
 27 RETURN
 28 REM COPYRIGHT 1982
 29 REM "SIDENA"
 30 LET OUT=255
 40 GOSUB 20
 50 LET OUT=0
 60 GOSUB 20
 70 GOTO 30
```

Fig. 3-4.

chaque voyant en mettant à la masse l'entrée portant le même numéro que la sortie testée.

Ce programme fournit une occasion de découvrir le fonctionnement du sous-programme d'entrée, qui ressemble beaucoup à ce qui a été vu pour la sortie : il faut exécuter l'instruction :

GOSUB 25

et l'octet présent sur les entrées se retrouve dans la variable IN. Les variables IN et OUT sont des variables BASIC et peuvent ainsi être traitées en machine selon les procédures habituelles.

Les applications les plus immédiates d'une carte d'entrée-sortie ne se prêtent cependant guère à la manipulation directe d'octets. On souhaite, par exemple, mettre en service les sorties n° 1, 5 et 6,

```
    1 GOTO 3
    2 REM COPY U?RNDPEEK ?TAN <=?
M?RNDTAN .
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 GOTO 30
   20 POKE Q7,OUT
   21 LET Q6=USR (Q8)
   22 RETURN
   25 LET Q6=USR (Q9)
   26 LET IN=PEEK (Q7)
   27 RETURN
   30 GOSUB 25
   40 LET OUT=IN
   50 GOSUB 20
   60 GOTO 30
   70 REM COPYRIGHT 1982
   80 REM "TEST"
```

Fig. 3-5.

sans pour autant avoir à effectuer les calculs visant à déterminer qu'il faut pour cela envoyer en sortie l'octet 49 !

Inversement, on préférerait savoir que les entrées n° 2, 3, 5 et 8 sont à la masse, plutôt que d'avoir à décoder l'octet 150... Le programme de la *figure 3-6* comprend les routines permettant au ZX-81 de se charger de ces fastidieux calculs.

L'entrée comme la sortie se voient affecter chacune huit variables indicées, respectivement E(1) à E(8) et S(1) à S(8).

Pour activer les sorties 1, 5 et 6, par exemple, il suffit de faire :

```
LET S(1) = 1          LET S(5) = 1
LET S(2) = 0          LET S(6) = 1
LET S(3) = 0          LET S(7) = 0
LET S(4) = 0          LET S(8) = 0
```

```
    1 GOTO 3
    2 REM COPY U■RNDPEEK ■TAN (=■
M■RNDTAN .
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
   22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
   23 POKE Q7,OUT
   24 LET Q6=USR (Q8)
   25 RETURN
   26 LET Q6=USR (Q9)
   33 LET IN=PEEK (Q7)
   34 LET Q5=128
   35 FOR Q=1 TO 8
   36 LET E(9-Q)=INT (IN/Q5)
   37 IF IN>=Q5 THEN LET IN=IN-Q5
   38 LET Q5=Q5/2
   39 NEXT Q
   40 RETURN
   41 REM INTERFACE SIDENA 8E5
   42 REM SORTIE:S(1) A S(8) PUIS
 GOSUB 22
   43 REM ENTREE: GOSUB 26 PUIS
E(1) A E(8)
   45 REM PROGRAMME A PARTIR DE
LA LIGNE 50
   49 REM COPYRIGHT 1982
```

Fig. 3-6.

Etant bien entendu que, pour modifier l'état d'une seule sortie sans s'intéresser aux autres, il suffit d'actualiser la seule variable indicée correspondante. Pour remettre la sortie 6 au repos, il suffirait donc de faire :

LET S(6) = 0

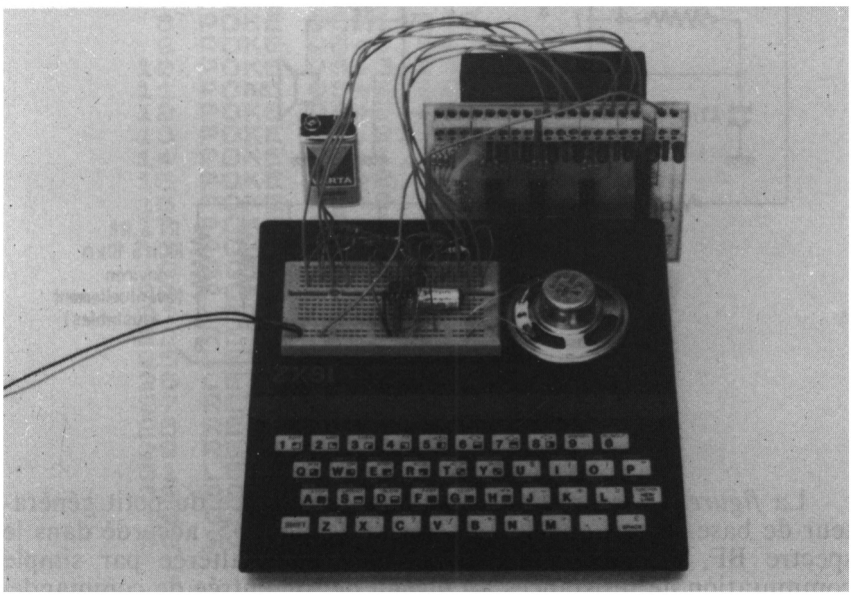
Le principe est exactement le même au niveau des entrées, chaque élément du « tableau » E contenant un 1 ou un 0, selon l'état de l'entrée portant le même numéro.

On notera que les sous-programmes d'accès aux entrées et sorties portent des numéros de lignes différents de ceux du programme précédent : on déclenche une sortie par GOSUB 22, une entrée par GOSUB 26. Le programme principal, quant à lui, devra être écrit à partir de la ligne 50.

Il n'est pas nécessaire d'en savoir davantage pour réaliser des montages intéressants mettant à contribution la carte d'entrée-sortie 8ES, associée, bien sûr, à quelques composants extérieurs.

Le ZX-81 musicien

Les interfaces sonores se rencontrent de plus en plus fréquemment sur les ordinateurs individuels, ne serait-ce que pour permettre de donner une dimension supplémentaire aux innombrables jeux existants ou à venir.



Techniquement parlant, rien n'empêche un ordinateur convenablement équipé de jouer une musique composée par lui-même, bien que les résultats obtenus ne soient pas du goût de tout le monde !

A notre sens, la machine n'a pas à prendre la place de l'homme au niveau de l'inspiration artistique (nous allons avoir l'occasion d'entendre les résultats ainsi obtenus !), mais il est certain qu'un ordinateur muni d'une interface sonore peut contribuer à la création d'effets intéressants.

Le procédé que nous avons retenu pour donner la parole au ZX-81 consiste à commander par les sorties de la carte 8ES les « touches » d'une sorte de petit orgue électronique monodique. Il s'agit là, bien sûr, de l'approche la plus simple possible, et le même principe peut servir de base à des expérimentations plus poussées, pour lesquelles des générateurs sonores plus performants pourront avantageusement remplacer notre circuit de démonstration.

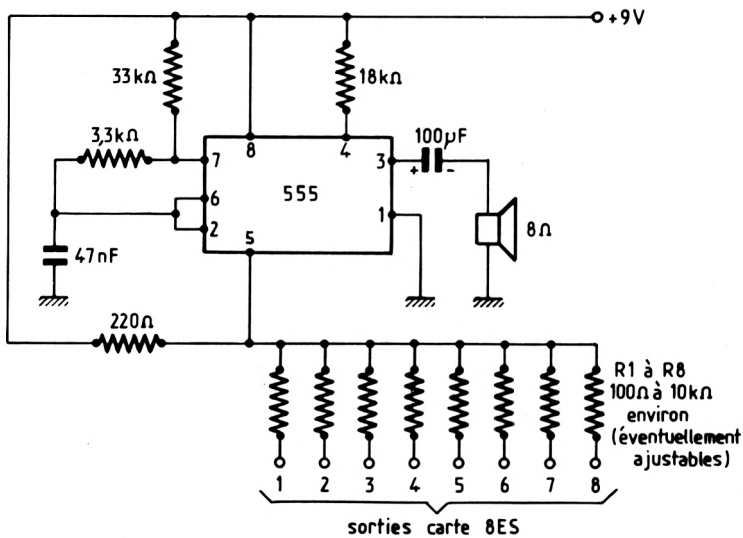


Fig. 3-7.

La *figure 3-7* reproduit le schéma de principe du petit générateur de base. Il fait appel à un très classique 555, accordé dans le spectre BF, et dont la fréquence peut être altérée par simple commutation de résistances au niveau de son entrée de commande.

L'avantage principal de ce choix est que le 555 peut piloter directement un petit haut-parleur, sans amplificateur intermédiaire. Il est commode de réaliser ce montage sur une petite boîte de connexions sans soudures, afin de faciliter le remplacement rapide des résistances R_1 à R_8 lors des manipulations visant à obtenir des effets sonores intéressants. Pour commencer, on pourra choisir pour R_1 une valeur voisine de 100 Ω et adopter pour chaque autre résistance une valeur à peu près double de la précédente. Rien n'empêche, bien sûr, de prévoir huit résistances ajustables, auquel cas l'étalonnage pourra se faire en respectant les huit notes d'une octave, de DO à DO.

Le montage sera alimenté *exclusivement par une pile de 9 V,* mais on n'oubliera pas de relier le pôle négatif de cette dernière à la borne de masse de la carte 8ES. Les huit résistances du générateur rejoindront les huit sorties de la carte 8ES, l'ordre croissant des

```
    1 GOTO 3
    2 REM SU?RNDPEEK ?TAN :=?N?RN
DTAN
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 GOTO 30
   20 POKE Q7,OUT
   21 LET Q6=USR (Q8)
   22 RETURN
   25 LET Q6=USR (Q9)
   26 LET IN=PEEK (Q7)
   27 RETURN
   28 REM COPYRIGHT 1982
   29 REM "SIDENA"
   31 LET OUT=255*RND
   32 PAUSE 5
   40 GOSUB 20
   50 GOTO 31
```

Fig. 3-8.

valeurs suivant la numérotation des sorties. On pourra alors charger et lancer le programme de la *figure 3-8* pour juger de l'aptitude du ZX-81 à jouer les Beethoven ! Il n'est pas nécessaire de posséder le génie du titan de Bonn pour se rendre compte que la fonction aléatoire RND du ZX-81 ne dispose que d'un bien piètre sens artistique !

Rendons donc l'initiative à l'opérateur en utilisant le programme de la *figure 3-9.*

```
    1 GOTO 3
    2 REM * U■RNDPEEK ■TAN ↑=■↑■RN
DTAN
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
   22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
   23 POKE Q7,OUT
   24 LET Q6=USR (Q8)
   25 RETURN
   26 LET Q6=USR (Q9)
   33 LET IN=PEEK (Q7)
   34 LET Q5=128
   35 FOR Q=1 TO 8
   36 LET E(9-Q)=INT (IN/Q5)
   37 IF IN>=Q5 THEN LET IN=IN-Q5
   38 LET Q5=Q5/2
   39 NEXT Q
   40 RETURN
   41 REM INTERFACE SIDENA 8ES
   42 REM SORTIE:S(1) A S(8) PUIS
 GOSUB 22
   43 REM ENTREE: GOSUB 26 PUIS
E(1) A E(8)
```

```
  45 REM PROGRAMME A PARTIR DE
LA LIGNE 50
  49 REM COPYRIGHT 1982
  50 SLOW
  55 CLS
  60 PRINT "POUR PROGRAMMER,APPU
YER SUR P"
  65 PRINT
  70 PRINT "POUR DEMARRER,APPUYE
R SUR R"
  80 IF INKEY$="P" THEN GOTO 100
  85 IF INKEY$="R" THEN GOTO 300
  90 GOTO 72
 100 CLS
 110 PRINT "CADENCE DE DEROULEME
NT EN SEC ?"
 120 INPUT C
 130 LET C=C*50
 135 LET L=1
 136 LET P$=""
 140 CLS
 150 PRINT "NUMERO DE SORTIE POU
R LE PAS "
 160 PRINT "NUMERO ";L;" (1 A 8)
"
 161 PRINT
 162 PRINT "A LA FIN,ENTRER 0"
 165 PRINT
 170 PRINT "PUIS [illegible]"
 180 INPUT N
 190 CLS
 195 IF N=0 THEN GOTO 50
 200 LET P$=P$+STR$ N
 210 LET L=L+1
 220 GOTO 140
 300 FAST
 305 FOR F=1 TO LEN P$
 310 LET OUT=0
 320 GOSUB 23
 330 LET S(VAL P$(F))=1
 340 GOSUB 22
 350 PAUSE C
 355 LET S(VAL P$(F))=0
 360 NEXT F
 370 GOTO 300
 380 REM "PROGRAMME"
```

Fig. 3-9.

Ce programme n'a pas seulement des ambitions musicales, car il peut servir à enregistrer et reproduire les séquences marche-arrêt les plus variées sur pratiquement n'importe quel groupe d'appareils électriques limité à huit éléments, depuis les électrovannes, résis-

tances et moteurs d'une machine à laver, jusqu'aux projecteurs et magnétophones d'un spectacle « son et lumière ». Le cycle simplifié, consistant à n'activer qu'une sortie à la fois pendant un temps fixé une fois pour toutes, qui convient bien à la reproduction de petits airs musicaux, pourrait facilement être compliqué d'après les idées mises en œuvre dans les prochains exemples.

Le programme prévoit un dialogue très clair qui rend inutile toute description de son mode d'emploi, mais signalons qu'il est possible d'enregistrer sur cassette toute séquence jugée suffisamment digne d'intérêt. Lors du rechargement, on se gardera alors de lancer le programme par RUN, mais on utilisera GOTO 50.

Le ZX-81 vous réveille

Le programme de la *figure 3-10* utilise un cycle un peu plus élaboré, mais beaucoup plus court et immuable, pour transformer le ZX-81 en un sympathique « robot domestique » capable de prendre en charge les *« levers du Roy »* avec un maximum d'égards ; mais, jugez plutôt !

```
    1 GOTO 3
    2 REM ■ U?RNDPEEK KTAN <=K^?RN
DTAN .
    3 LET Q7=16527
    4 LET Q8=16526
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
   22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
   23 POKE Q7,OUT
```

```
  24 LET Q6=USR (Q8)
  25 RETURN
  26 LET Q6=USR (Q9)
  33 LET IN=PEEK (Q7)
  34 LET Q5=128
  35 FOR Q=1 TO 8
  36 LET E(9-Q)=INT (IN/Q5)
  37 IF IN>=Q5 THEN LET IN=IN-Q5
  38 LET Q5=Q5/2
  39 NEXT Q
  40 RETURN
  50 PRINT "HEURE DU REVEIL ?"
  55 PRINT " HEURES"
  60 INPUT HR
  65 PRINT " MINUTES"
  67 INPUT MR
  70 CLS
  75 PRINT "QUELLE HEURE EST-IL
?"
  80 PRINT " HEURES"
  85 INPUT H
  90 PRINT " MINUTES"
  95 INPUT M
 100 LET T=MR-M+(60*(24-H+HR))
 110 CLS
 120 LET L=0
 130 PAUSE 2985
 140 LET L=L+1
 150 PRINT AT 0,0;L;"
"
 160 IF L<T-15 THEN GOTO 130
 170 IF L=T-15 THEN GOTO 300
 180 IF L=T-10 THEN GOTO 400
 190 IF L=T THEN GOTO 500
 200 IF L>=T+5 THEN GOTO 558
 210 GOTO 130
 300 LET S(1)=1
 350 GOSUB 22
 360 GOTO 130
 400 LET S(3)=1
 450 GOSUB 22
 460 GOTO 130
 500 LET S(8)=1
 550 GOSUB 22
 555 GOTO 130
 558 LET S(2)=1
 559 GOSUB 22
 560 PAUSE 1500
 600 LET OUT=0
 650 GOSUB 23
 700 REM COPYRIGHT 1982
 800 REM "REVEIL"
```

Fig. 3-10.

Lors du coucher, le programme s'enquiert de l'heure présente, car le ZX-81 ne peut pas garder un synchronisme rigoureux avec le temps pendant plus de quelques heures. On peut alors lui préciser l'heure souhaitée pour le réveil du lendemain avant qu'il ne passe en attente (on peut alors, bien sûr, arrêter le téléviseur !). Un quart d'heure avant l'heure programmée pour le réveil, le ZX met en route la cafetière (ou tout autre appareil similaire).

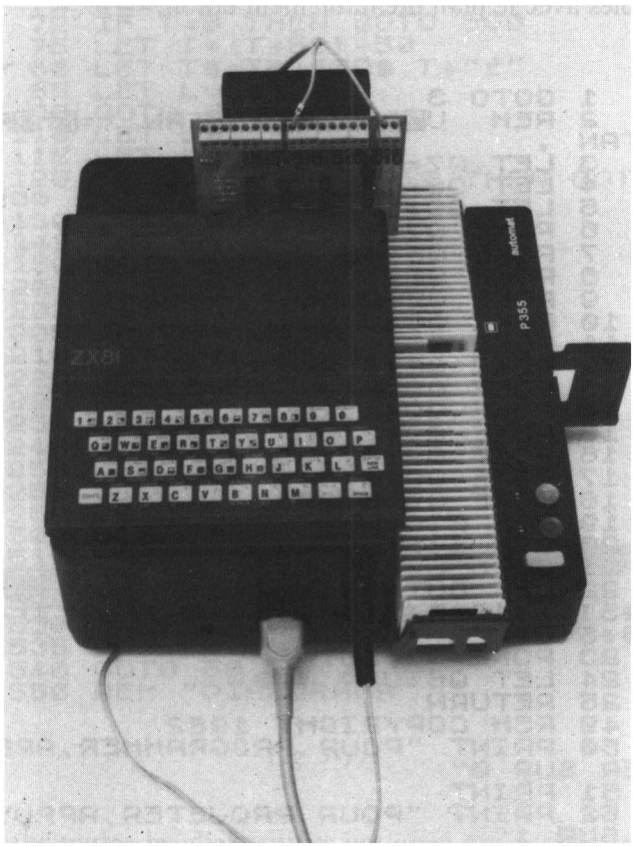
Cinq minutes plus tard, c'est le tour de la radio, suivie à l'heure précise du lever par la lumière de la chambre.

Enfin, cinq minutes plus tard, si le programme n'a pas été interrompu par BREAK, une sirène est actionnée pendant trente secondes environ !

On pourrait envisager une étape suivante consistant à appeler automatiquement les pompiers par téléphone, mais ceci fera l'objet d'un autre programme...

Le ZX-81 passe vos diapositives

Une bonne solution, pour délivrer le photographe amateur de l'esclavage que représente la projection de diapositives en parfait synchronisme avec une bande sonore, consiste à disposer un synchronisateur entre le magnétophone et le projecteur. Cependant, tous



les magnétophones n'acceptent pas de gaieté de cœur cette adaptation, surtout lorsque la stéréophonie doit être conservée. L'informatique offre une très élégante solution de rechange, puisque toutes les informations concernant le passage des vues peuvent être enregistrées sur les premières minutes de la bande, chargées en machine

d'un bloc avant le début de la séance et relues en mémoire au fur et à mesure de la projection, en toute indépendance vis-à-vis du magnétophone.

Le programme de la *figure 3-11*, lors de son premier lancement par RUN NEWLINE, interroge l'opérateur sur la durée pendant laquelle il souhaite projeter chaque diapositive. Le numéro de la vue est toujours rappelé afin de permettre tous les recoupements souhaitables avec le plan préalablement établi.

```
    1 GOTO 3
    2 REM  U■RNDPEEK ■TAN <=■M■RN
DTAN .
    3 LET Q7=16527
    4 LET Q8=16528
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
   22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
   23 POKE Q7,OUT
   24 LET Q6=USR (Q8)
   25 RETURN
   49 REM COPYRIGHT 1982
   50 PRINT "POUR PROGRAMMER,APPU
YER SUR 0"
   51 PRINT
   52 PRINT "POUR PROJETER,APPUYE
R SUR 1"
   53 IF INKEY$="0" THEN GOTO 56
   54 IF INKEY$="1" THEN GOTO 100
   55 GOTO 53
   56 LET L=1
   58 LET T$=""
   59 CLS
   60 PRINT "TEMPS DE PROJECTION
DE LA VUE"
```

```
  65 PRINT "NUMERO ";L;" EN SECO
NDES ?"
  66 PRINT
  67 PRINT "(A LA FIN,REPONDRE 0
)"
  68 PRINT
  69 PRINT "EN VALIDANT PAR [NEW
LINE]"
  70 INPUT T
  71 CLS
  72 IF T=0 THEN GOTO 500
  75 LET T=(T*50)-50
  80 LET T$=T$+STR$ T+"£"
  85 LET L=L+1
  90 GOTO 59
 110 LET L=1
 115 LET P$=""
 120 IF CODE T$(L)=12 THEN GOTO
200
 130 LET P$=P$+T$(L)
 140 LET L=L+1
 150 GOTO 120
 200 CLS
 202 FAST
 205 PAUSE VAL P$
 210 LET S(1)=1
 220 GOSUB 22
 225 PAUSE 15
 230 LET S(1)=0
 240 GOSUB 22
 250 LET L=L+1
 255 SLOW
 260 GOTO 115
 500 CLS
 510 PRINT "[ENREGISTRER SUR CASS
ETTE]"
 520 PAUSE 300
 530 SAVE "DIAPORAM[A]"
 535 CLS
 540 GOTO 50
 550 REM "DIAPORAMA"
```

Fig. 3-11.

Lorsque toutes les diapositives ont ainsi été enregistrées, le fait de répondre 0 à l'interrogation déclenche automatiquement la sauvegarde sur cassette du programme et de ses données. Il est intéressant d'utiliser la suite de la même cassette pour enregistrer la bande sonore, dont la réalisation sera exécutée pendant une projection pilotée par le ZX-81. Ce pilotage est très facilement obtenu en répondant 1 à la question de la ligne 52.

Le branchement du projecteur (muni d'une prise de télécommande) ne soulève généralement aucun problème particulier : il suffit le plus souvent de relier les deux fils prévus pour recevoir le contact de passage de vues à la sortie 1 de la carte 8ES (+) et à la borne de masse de cette carte (–). Il faut cependant vérifier que la tension à commuter n'excède pas 30 V sous 2 A et que la polarité ne s'inverse jamais. Le cas échéant, il pourra être nécessaire de prévoir des protections, et, en cas de doute, on pourra toujours passer par un relais alimenté par sa propre pile.

A titre indicatif, nous avons pu réaliser un branchement direct sans aucun problème sur un projecteur P 355 Rollei (broches 2 et 3 de la prise DIN 5 broches).

On notera que ce programme n'utilise que la sortie n° 1 de la carte 8ES et que des modifications très simples pourraient autoriser la commande d'autres projecteurs (multivision), d'éclairages de salle, rideaux de scène, voire du magnétophone lui-même.

Le ZX-81 standardiste

Ce programme performant, listé à la *figure 3-12,* remplit à la fois les fonctions de répertoire téléphonique et de composeur automatique à mémoire.

```
    1 GOTO 3
    2 REM  U?RNDPEEK KTAN  =K=?RN
DTAN
    3 LET 07=16527
    4 LET 08=16528
    5 LET 09=16534
    6 POKE 07,0
    7 POKE 08,58
    8 POKE 08+1,143
    9 POKE 08+2,64
   10 POKE 08+3,211
   11 POKE 08+4,127
   12 POKE 08+5,201
   13 POKE 09,219
   14 POKE 09+1,127
   15 POKE 09+2,50
   16 POKE 09+3,143
   17 POKE 09+4,64
   18 POKE 09+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
```

```
  22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
  23 POKE 07,OUT
  24 LET Q6=USR (08)
  25 RETURN
  49 REM COPYRIGHT 1982
  50 CLS
  55 PRINT "NOM DE L""ABONNE ?"
  56 PRINT
  58 PRINT "SUIVI DE [redacted]"
  60 INPUT A$
  70 CLS
  75 LET B$=""
  80 IF A$="HORLOGE" THEN LET B$
="4638400"
  85 IF A$="POMPIERS" THEN LET B
$="18"
  90 IF A$="REAGAN" OR A$="RONAL
D" THEN LET B$="19 12024561414"
 100 IF A$="BREJNEV" OR A$="LEON
ID" THEN LET B$="19 7095295905I"
 500 IF B$="" THEN GOTO 50
 505 FAST
 510 LET L=0
 520 LET L=L+1
 530 IF L>LEN B$ THEN GOTO 1000
 535 IF CODE B$(L)=28 THEN LET C
=10
 540 IF CODE B$(L)>28 AND CODE B
$(L)<=37 THEN LET C=VAL B$(L)
 545 IF CODE B$(L)=0 THEN GOTO 2
000
 550 FOR F=1 TO C
 560 LET S(1)=1
 570 GOSUB 22
 580 PAUSE 1
 590 LET S(1)=0
 600 GOSUB 22
 620 NEXT F
 630 PAUSE 20
 640 GOTO 520
1000 SLOW
1100 PRINT "POUR COMPOSER LE MEM
E NUMERO"
1110 PRINT "APPUYER SUR N"
1120 PRINT
1130 PRINT "POUR CHANGER DE NUME
RO"
1140 PRINT "APPUYER SUR C"
1150 IF INKEY$="C" THEN GOTO 50
1160 IF INKEY$="N" THEN GOTO 150
0
1170 GOTO 1150
1500 CLS
```

(suite au verso)

```
1505 LET S(1)=1
1510 GOSUB 22
1520 PAUSE 150
1530 LET S(1)=0
1540 GOSUB 22
1550 PAUSE 500
1560 GOTO 505
2000 PAUSE 500
3000 GOTO 520
5000 REM "COMPOSEUR"
```

Fig. 3-12.

Branché selon le schéma de la *figure 3-13* sur une ligne téléphonique (sous réserve des dispositions réglementaires en vigueur au jour de réalisation de l'installation), ce système ne perturbe en rien le fonctionnement du poste téléphonique d'origine, que le ZX-81 soit ou non sous tension. Mis en service, il peut cependant assurer des services très appréciables ; qu'on en juge !

A condition que la programmation voulue ait été préalablement effectuée selon les besoins propres de chacun, il suffit, pour appeler un correspondant, de frapper au clavier son nom, prénom, voire son surnom, puis NEWLINE, après avoir décroché le combiné et obtenu la tonalité. La machine se charge de tout, y compris de la

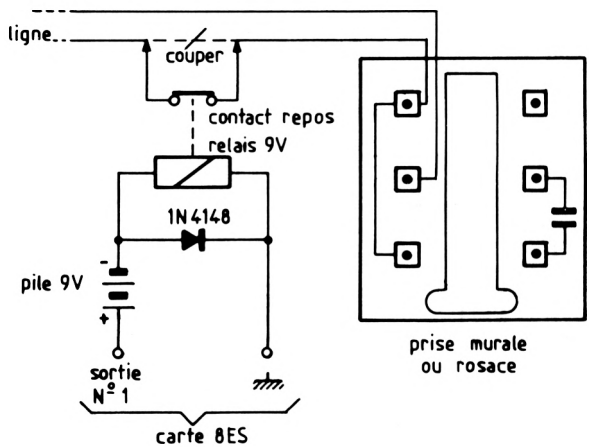
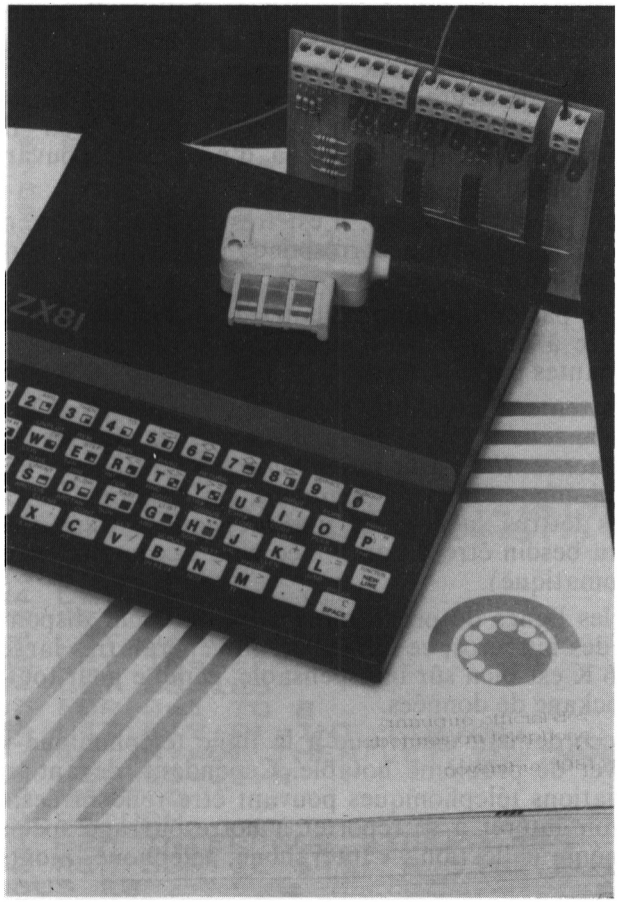


Fig. 3-13.



numérotation sur le réseau local, interurbain ou international, même derrière un standard privé exigeant la composition d'un code d'accès à l'extérieur. La longueur des numéros n'est pas limitée, pas plus d'ailleurs que le nombre de pauses d'attente de tonalités intermédiaires.

Si le numéro demandé est occupé, il suffit de frapper N pour que la machine obtienne seule la tonalité et recompose le même numéro. Pour essayer un autre numéro, il suffit de presser la touche C puis de reprendre la procédure normale, sans pour autant avoir à manipuler le combiné.

Le programme imprimé ici contient, à titre d'exemple, quatre numéros locaux et internationaux que tout homme d'action se doit de connaître : il suffira donc de frapper « REAGAN » ou « RONALD », selon le degré d'intimité atteint, pour se trouver mis en relation, sous quelques secondes, avec la Maison Blanche à Washington. Attention ! il s'agit là d'un plaisir pouvant devenir coûteux à la longue !

A partir de ces exemples, il est très facile d'entrer en machine les lignes de programme correspondant à son cas personnel en notant les points suivants :

– Il n'existe pas de limitation quant au nombre de libellés pouvant correspondre à un numéro donné. Il suffit de séparer les chaînes correspondantes par OR A$ =.

– Toute attente de tonalité (ligne extérieure, 16, 19, etc.) sera programmée sous forme d'un *espace* entre les chiffres du numéro. En conséquence, *il ne faut pas séparer* les chiffres à l'intérieur du numéro lui-même. Bien évidemment, on traduira en chiffres les éventuelles lettres survivantes d'une époque révolue (la machine pourrait au besoin être programmée pour accomplir cette tâche de façon automatique).

– Seules les lignes de programme 80 à 499 sont disponibles pour accueillir des numéros, ce qui suffit, en général, très largement. Le module 16 K est bien sûr indispensable, comme pour toute application de stockage de données.

Le raccordement électrique à la ligne téléphonique ne devrait pas soulever de problème notable. Cependant, devant la diversité des installations téléphoniques pouvant être rencontrées, le lecteur pourra avoir intérêt à se reporter à notre ouvrage spécialisé paru dans la même collection : « Interphone, téléphone, montages périphériques ».

Le ZX-81 veille sur vous

Chargé avec le programme de la *figure 3-14* et branché comme précédemment à une ligne téléphonique, le ZX-81 équipé de la carte 8ES se transforme en centrale d'alarme à transmission à distance. Pour permettre une pleine utilisation des possibilités de ce programme, des circuits électroniques plus élaborés sont nécessaires ; ils pourront être réalisés selon le schéma de la *figure 3-15*. Le fonctionnement de ce programme est en effet le suivant : en mode

```
    1 GOTO 3
    2 REM  U■RNDPEEK ■TAN (=■N■RN
DTAN
    3 LET Q7=16527
    4 LET Q6=16526
    5 LET Q9=16534
    6 POKE Q7,0
    7 POKE Q8,58
    8 POKE Q8+1,143
    9 POKE Q8+2,64
   10 POKE Q8+3,211
   11 POKE Q8+4,127
   12 POKE Q8+5,201
   13 POKE Q9,219
   14 POKE Q9+1,127
   15 POKE Q9+2,50
   16 POKE Q9+3,143
   17 POKE Q9+4,64
   18 POKE Q9+5,201
   19 DIM E(8)
   20 DIM S(8)
   21 GOTO 50
   22 LET OUT=S(1)+2*S(2)+4*S(3)+
8*S(4)+16*S(5)+32*S(6)+64*S(7)+1
28*S(8)
   23 POKE Q7,OUT
   24 LET Q6=USR (Q8)
   25 RETURN
   26 LET Q6=USR (Q9)
   33 LET IN=PEEK (Q7)
   34 LET Q5=128
   35 FOR Q=1 TO 8
   36 LET E(9-Q)=INT (IN/Q5)
   37 IF IN>=Q5 THEN LET IN=IN-Q5
   38 LET Q5=Q5/2
   39 NEXT Q
   40 RETURN
   41 REM INTERFACE SIDENA 8E5
   42 REM SORTIE:S(1) A S(8) PUIS
 GOSUB 22
   43 REM ENTREE: GOSUB 26 PUIS
E(1) A E(8)
   45 REM PROGRAMME A PARTIR DE
LA LIGNE 50
   49 REM COPYRIGHT 1982
   50 FAST
   51 LET S(1)=1
   52 GOSUB 22
   53 LET A=2000
   55 GOSUB 26
   60 FOR F=1 TO 5
   70 IF E(F)=0 THEN GOTO 100
   80 NEXT F
   90 GOTO 50
  100 SLOW
```

(suite au verso)

```
  110 PRINT "ALARME ";
  115 LET A$=""
  120 FOR F=1 TO 5
  130 IF E(F)=0 THEN LET A$=A$+ST
R$ F
  140 NEXT F
  150 PRINT A$
  160 LET B$="16 14638400"
  162 LET S(1)=0
  163 GOSUB 22
  164 PAUSE 300
  179 FAST
  180 LET L=0
  190 LET L=L+1
  200 IF L>LEN B$ THEN GOTO 500
  210 IF CODE B$(L)=28 THEN LET C
=10
  220 IF CODE B$(L)>28 AND CODE B
$(L)<=37 THEN LET C=VAL B$(L)
  230 IF CODE B$(L)=0 THEN GOTO 7
00
  240 FOR F=1 TO C
  260 LET S(1)=1
  270 GOSUB 22
  280 PAUSE 1
  290 LET S(1)=0
  300 GOSUB 22
  320 NEXT F
  330 PAUSE 20
  340 GOTO 190
  500 LET K=0
  505 GOSUB 26
  510 IF E(8)=0 THEN GOTO 1000
  520 LET K=K+1
  530 PAUSE 50
  540 IF K>=70 THEN GOTO 5000
  550 GOTO 505
  700 PAUSE 500
  800 GOTO 190
 1000 FOR F=1 TO LEN A$
 1005 FOR G=1 TO VAL A$(F)
 1010 LET S(4)=1
 1020 GOSUB 22
 1030 PAUSE 10
 1040 LET S(4)=0
 1050 GOSUB 22
 1060 PAUSE 10
 1070 NEXT G
 1075 PAUSE 200
 1080 NEXT F
 1090 CLS
 1095 SLOW
 1100 GOTO 50
 2000 LET A=2100
 2010 LET B$="18"
```

```
2020 GOTO 162
2100 LET A=160
2110 LET B$="222222"
2120 GOTO 162
5000 LET S(1)=1
5010 GOSUB 22
5020 PAUSE 150
5030 GOTO A
9000 REM "TRANSMETTEUR"
```

Fig. 3-14.

veille (à partir de la mise en route du programme), le relais est collé, isolant ainsi la ligne téléphonique du montage proprement dit. Bien que le poste soit décroché, micro disposé devant un petit haut-parleur, la ligne reste donc libre. On notera qu'une alimentation secteur est nécessaire (au besoin secourue par batteries), puisque le relais est collé en permanence.

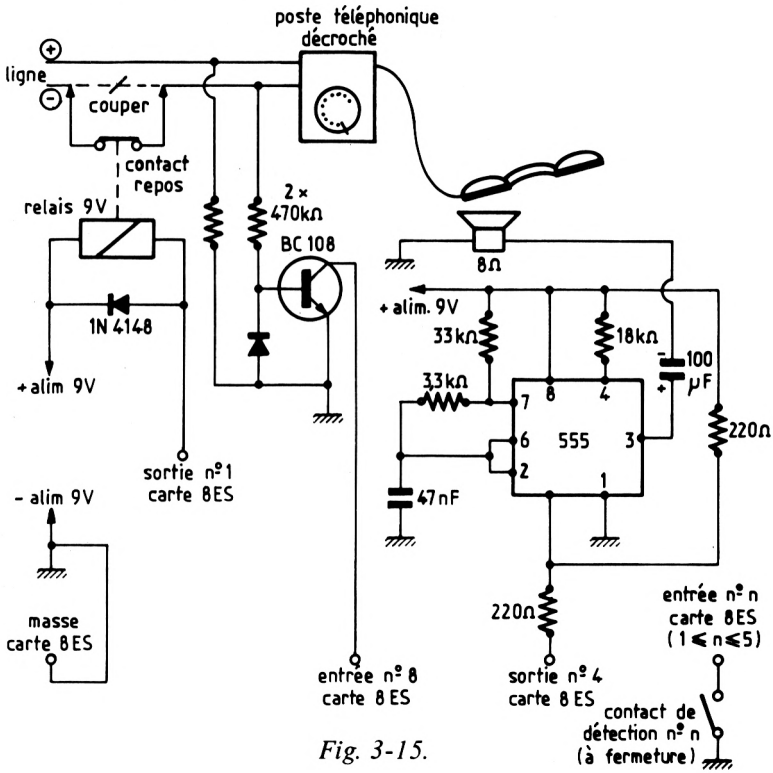


Fig. 3-15.

Si une ou plusieurs des cinq entrées « d'alarme » se trouvent réunies à la masse, même brièvement, les numéros d'alarmes correspondants se trouvent enregistrés dans une chaîne A$ et, à des fins de contrôle uniquement, imprimés sur l'écran TV.

La machine appelle alors le numéro de téléphone programmé à la ligne 160 (mêmes conventions que dans le programme précédent) et attend pendant un temps raisonnable une réponse, matérialisée par l'inversion de la polarité de ligne, transmise à l'entrée n° 8.

L'ordinateur annonce alors, à la suite les uns des autres, les numéros des entrées qui ont été activées, sous forme de trains de « bips » sur un fond sonore de fréquence fixe. La cadence est choisie de façon à permettre un comptage aisé de ces signaux. La non-réponse du numéro appelé entraîne l'appel du numéro programmé à la ligne 2010, lui-même secouru par celui de la ligne 2110. Il serait facile, si nécessaire, d'ajouter des numéros supplémentaires selon les mêmes modalités.

Il faut rester très conscient du fait que ce programme ne peut pas supporter de perte de mémoire intempestive de la part du ZX-81 et que des mesures adéquates devront être prises en vue de rendre irréprochable la fiabilité de celui-ci. Une bonne solution consiste à l'alimenter sur une batterie de 7,5 V montée en tampon avec un chargeur de qualité.

De toute façon, ce programme est surtout fourni en tant qu'exemple des vastes possibilités de la carte 8ES et devrait faire l'objet d'essais serrés avant toute utilisation mettant en jeu la sécurité des biens ou des personnes. Il ne faut pas perdre de vue, en effet, que le ZX-81 est un ordinateur d'amateur et n'a jamais prétendu offrir une fiabilité totale (dans le cas contraire, son prix ne serait pas celui que nous connaissons actuellement !).

Les limites des entrées-sorties sous BASIC

Nous espérons que ces quelques programmes auront contribué à faire découvrir à nos lecteurs la variété des applications que laisse entrevoir le ZX-81 équipé de la carte 8ES. En fait, on pourrait croire que seules les limites de l'imagination du programmeur pourraient freiner l'ardeur de ce système.

La réalité n'est pas exactement aussi réjouissante : il nous a en effet fallu choisir parmi nos nombreux projets ceux qui restaient compatibles avec la lenteur des entrées-sorties sous BASIC.

Avec nos applications téléphoniques, nous atteignons l'extrême limite de ce que permet le BASIC en mode rapide, soit une fréquence de commutation des sorties de 2 Hz environ et une cadence d'interrogation des entrées du même ordre de grandeur.

Cette (relative) lenteur est inacceptable pour une foule d'applications, telles que : les jeux interactifs (avec manches de commande), l'automatisation des réseaux de chemin de fer miniature, la commande d'imprimantes, téléscripteurs et autres, ou le codage

et décodage en temps réel de signaux Morse ou RTTY, pour ne pas oublier nos amis radio-amateurs.

L'augmentation massive de rapidité exigée par bon nombre d'applications (et pas seulement par les opérations d'entrée-sortie) ne peut être obtenue, sur le ZX-81, que par le recours à la programmation en *langage machine* Z-80 dont va traiter la suite de cet ouvrage. Au fil de notre exploration de ce mode de programmation radicalement différent du BASIC, nous découvrirons bien d'autres avantages intéressants à exploiter, au prix bien sûr de difficultés auxquelles le BASIC ne nous avait guère habitués, mais dont il ne faudrait tout de même pas surestimer l'importance !

Chapitre 4

Initiation au langage machine du ZX-81

Notre but n'est pas, en rédigeant cet ouvrage, de publier un livre de plus traitant de la programmation du microprocesseur Z-80. Il en existe d'excellents, auxquels nous avons d'ailleurs dû nous référer bien souvent lors de notre propre approche du langage machine du ZX-81.

L'auteur de ces lignes tient en effet à préciser que, s'il ignorait tout du BASIC en achetant son ZX-81, il imaginait à l'époque le langage machine comme une sorte de monstruosité, à l'égard de laquelle la plus élémentaire prudence s'imposait.

En effet, le lecteur du manuel Sinclair, qui serait tenté d'aborder le chapitre consacré au langage machine, ressent immédiatement la désagréable impression que, s'il ne fait pas partie d'une élite bien délimitée, l'usage de la touche USR lui est définitivement interdit... à moins de s'atteler à la lecture de quelques livres, dont on précise aimablement « qu'il serait présomptueux de les recommander à des débutants ».

Et, en effet, le débutant, même sorti victorieux de l'étude du BASIC, ne peut rien tirer de la littérature consacrée au Z-80, pour la simple raison que le ZX-81 *n'est pas* un système de développement pour le Z-80, mais bien un *ordinateur BASIC* possédant une possibilité de *branchement vers des routines machine.*

Lorsque le programme machine servant à piloter la carte 8ES a éveillé notre curiosité quant à l'art et la manière d'utiliser la fonction USR, nous ne nous attendions pas à passer autant de temps et à brasser autant de papier avant de commencer à comprendre les petits secrets de la programmation du ZX-81 en lan-

gage machine. Nous savons maintenant que, une fois franchie la très haute marche qui sépare le BASIC du langage machine, la situation s'améliore fort et que d'intéressants résultats peuvent très vite être obtenus, à la condition de ne jamais perdre de vue que le ZX-81 est fait pour travailler en BASIC, et que les programmes machine ne doivent être considérés que comme des auxiliaires de ce BASIC, lorsque celui-ci laisse entrevoir ses limites.

Tenter d'utiliser le ZX-81 exclusivement en langage machine reviendrait à employer le TGV pour desservir une ligne de trains de banlieue !

Bref, nous ne chercherons pas à faire de nos lecteurs des programmeurs de Z-80 (ce métier exige des années d'études et de pratique quotidienne), mais nous leur proposons tout simplement de profiter du travail qu'il nous a fallu accomplir avant de parvenir à faire fonctionner sur le ZX-81 des programmes BASIC capables de « sous-traiter » au langage machine les tâches qu'ils s'avèrent incapables de mener à bien.

Rien n'empêchera cependant nos lecteurs les plus passionnés de poursuivre leur découverte du langage machine à l'aide des livres dont nous avons déjà parlé, et vers la lecture desquels nous espérons ouvrir la voie avec cet ouvrage d'initiation.

Définition du langage machine

Le langage machine est le CODE dans lequel il est nécessaire de donner ses ordres aux unités de traitement que sont les circuits intégrés *microprocesseurs*. Le ZX-81 est construit autour d'un microprocesseur de type Z-80 qui ne « comprend » qu'un code bien précis. Le ZX-81 possède dans sa mémoire morte (ROM) un programme « traducteur », nommé *interpréteur BASIC*, et permettant au programmeur de formuler ses ordres dans un langage simple, alors que le Z-80 ne comprend qu'un langage beaucoup plus difficile à manipuler, car très proche de la machine. Bien sûr, les incessantes traductions exécutées par l'interpréteur prennent du temps, et c'est ce qui explique, en partie du moins, la relative lenteur du BASIC.

Pour programmer en langage machine, il faut se mettre « à la place » du Z-80 et gérer directement le contenu de milliers de cellules mémoire, dûment numérotées grâce aux *adresses* machine.

Le microprocesseur Z-80 n'est capable d'exécuter qu'un nombre limité d'opérations, en général très élémentaires. Il semble donc, au premier abord, moins « puissant » que le ZX-81 muni de son BASIC, mais, ne nous y trompons pas, sa véritable puissance réside dans la possibilité qu'ont ses instructions d'être *assemblées* entre elles de façon à *construire* des fonctions aussi complexes que nécessaire. Le BASIC du ZX-81 ne représente qu'un nombre limité, bien que déjà performant, d'assemblages d'instructions Z-80. On pourrait en créer bien d'autres permettant, par exemple, d'obtenir des lettres minuscules sur l'imprimante à la place des habituelles majuscules ou, sans chercher si loin, de transmettre des ordres à la carte 8ES, qui reste absolument insensible à n'importe quelle instruction BASIC tant qu'un programme machine convenable n'a pas été chargé !

Nous aborderons précisément la découverte du langage machine du ZX-81 par une « dissection » du programme de commande de la carte 8ES.

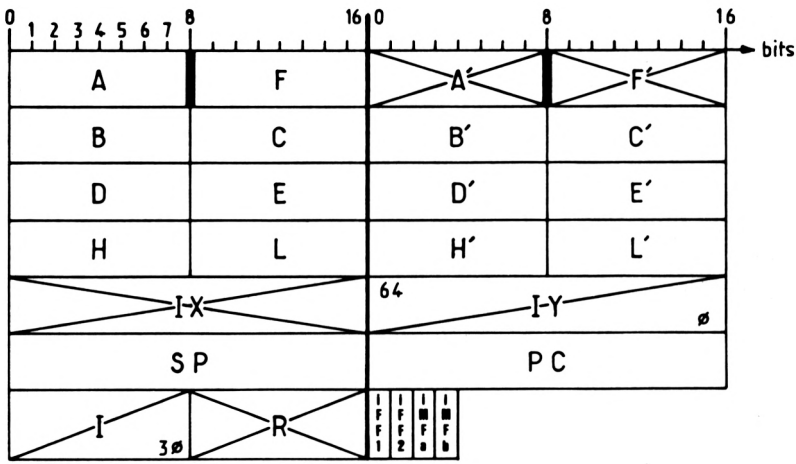
Examen d'un programme en langage machine

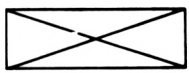
Pour bien comprendre le fonctionnement de programmes machine, il est nécessaire de connaître le nom des « registres » dans lesquels vont s'exécuter la plupart des opérations.

En effet, si le microprocesseur Z-80 peut, bien sûr, échanger des informations avec la totalité des cellules de la mémoire, il possède en propre, dans son boîtier même, quelques cellules bien particulières qu'il peut utiliser (et qu'il utilise souvent !) sans mettre à contribution la mémoire centrale, beaucoup plus lourde (donc plus lente) à manier.

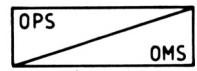
La *figure 4-1* donne la « carte géographique » de ces registres, dont la plupart sont prévus pour accueillir un octet mais peuvent facilement être associés deux à deux pour ranger deux octets qui doivent rester ensemble (par exemple l'OMS et l'OPS d'une adresse). Certains registres (IX, IY, SP, PC) acceptent d'autorité deux octets, mais ils jouent des rôles particuliers. Les registres A, F, A', F' jouent aussi des rôles tout à fait spéciaux.

Le registre A est nommé *accumulateur* et mérite le titre de registre principal, car c'est dedans que s'effectuent la plupart des opérations simples. Par exemple, une des instructions du Z-80





à ne pas utiliser sur le ZX-81 à cause de l'affichage



si utilisé, devra être réinitialisé à OPS, OMS

Fig. 4-1 – Carte des registres du Z-80.
N.B. – Pour éviter tout risque vis-à-vis des registres A' et F', il suffit de s'abstenir d'utiliser l'instruction EX AF, AF' (code décimal 8).
Pour protéger IX, il faut éviter toutes les instructions contenant IX dans leur mnémonique (en remplacement, travailler sur IY, quitte à le réinitialiser à 64,Ø après usage). La protection de R ne pose pas de problème particulier.
Les registres B' à L' sont accessibles grâce à l'instruction EXX (Code décimal 217), qui procède à l'échange pur et simple des contenus des registres B à L avec les registres B' à L'.

permet d'ajouter une quantité N à l'octet contenu dans A, le résultat de l'addition étant rangé dans A à son tour.

Le registre F contient huit bits indépendants, les *flags* ou *drapeaux*. Il s'agit d'*indicateurs* qui peuvent être positionnés par certaines instructions, notamment si un octet donné est nul, positif, négatif, pair, impair, etc.

D'autres instructions s'exécutent, ou non, selon qu'un de ces drapeaux est à 1 ou à 0. On retrouve ici l'équivalent des boucles IF-THEN du BASIC.

Signalons enfin quatre petits registres (des bascules, en fait) qui ne contiennent chacun qu'un bit.

On notera que le ZX-81 utilise bien évidemment certains de ces registres, et le programmeur ne peut pas en disposer à volonté. Sur la *figure 4-1,* les registres barrés une ou deux fois doivent faire l'objet de précautions particulières.

Cet indispensable prélable étant posé, nous invitons nos lecteurs à se reporter à la *figure 3-3,* qui reproduit le programme que nous allons étudier.

Examinons le sous-programme permettant de sortir un octet sur la carte 8ES (il n'est pas nécessaire de posséder cet accessoire pour suivre les explications).

A la ligne 20, le BASIC charge l'octet présent dans la variable OUT dans la cellule mémoire n° 16527. Cette cellule contenait le code correspondant au premier caractère de l'instruction REM de la ligne 2, soit 0 depuis la ligne 6.

Il est très important de bien localiser, sur la « carte » de la mémoire, la place de chacun des octets manipulés par ce programme. En cas de difficultés, on se reportera donc au *chapitre 2* qui donne tous les renseignements souhaitables à ce sujet.

La ligne 21 lance le programme machine proprement dit au moyen de la fameuse fonction USR.

Le mot-clé USR doit toujours être employé suivi d'une adresse mémoire, ici Q8, soit 16528.

Cette adresse est celle à partir de laquelle le ZX-81 ira chercher en mémoire le programme en langage machine à exécuter. Si l'adresse inscrite après USR ne correspond pas à un programme machine, le Z-80 tentera d'exécuter en tant que tel une suite d'octets sans queue ni tête, et nul ne peut prévoir ce qui se passera (neuf fois sur dix, la machine se bloque).

Remarquons que le mot-clé USR suivi d'une adresse ne peut pas être employé seul : il faut l'inclure dans une instruction, exactement au même titre qu'une variable. Ici, il a été choisi d'écrire :

LET Q6 = USR 16528

mais on aurait aussi bien pu prévoir :

PRINT USR 16528 ou, mieux : RAND USR 16528

En effet, la variable USR reçoit, lorsque la machine revient au BASIC en fin de routine machine, un nombre obtenu en considérant les octets contenus dans les registres B et C, comme l'OMS et l'OPS de ce nombre.

En pratique, ce nombre est souvent sans intérêt, et, comme il faut absolument en faire quelque chose, on le place dans une variable inutilisée (ici, Q6 qui ne sert à rien d'autre), mais on pourrait tout aussi bien l'imprimer (quitte à le chasser par un CLS) ou l'affecter à cette fameuse fonction RAND qui ne sert que bien rarement.

Par la suite, lorsque nous aurons à lancer un programme machine à la main, le plus rapide sera de faire :

RAND USR adresse NEWLINE

Dans notre exemple, donc, la machine démarre le programme machine à partir de l'adresse 16528, correspondant au code du second caractère de l'instruction REM, laquelle constitue décidément un lieu bien accueillant pour les octets de toutes sortes.

Cet octet, le premier de notre programme, revêt la valeur décimale 58, qui lui a été affectée par la ligne 7 lors de la première exécution du programme BASIC.

Si nous nous reportons à la table figurant à la fin de l'ouvrage, nous constatons que ce code correspond à une instruction nommée « LD A,(NN) », dont le rôle est de charger dans le registre A le contenu de la cellule mémoire dont l'adresse est contenue, elle, dans les deux otets suivant 58 dans le programme, dans l'ordre OMS suivi de OPS.

Ces deux octets sont respectivement 143 et 64, ce qui correspond à l'adresse 143 + (256 × 64) = 16527.

Après cette première instruction machine, le registre A contient donc l'octet que le BASIC avait placé dans la variable OUT, qui n'a fait que transiter par la cellule mémoire n° 16527. Au sujet de cette première instruction, il convient de remarquer que, dans le cours d'un programme en langage machine, un même octet n'a pas toujours la même signification : 58 en tête de programme représentait le code de l'opération LD A,(NN), mais le même octet répété une seconde fois aurait dû être interprété comme l'OMS de l'adresse sur laquelle agit cette opération. Il faut donc bien se garder de « reconnaître » l'instruction LD A,(NN) à chaque fois que l'octet 58 se présente !

Notons également que cette instruction n'occupe, en tout et pour tout, que trois octets en mémoire. Essayez d'imaginer l'encombrement d'une instruction BASIC exécutant une tâche voisine, par exemple :

```
10 LET ACCU = PEEK 16527
```

Pas moins de 23 octets ! On découvre ici un autre avantage de taille du langage machine, le très faible encombrement mémoire des programmes utilisant ce langage. C'est la raison pour laquelle certains jeux très évolués, écrits en langage machine, arrivent à se contenter de 1 K-octet de RAM.

L'instruction suivante débute par l'octet 211. La table nous indique que le *mnémonique* de l'opération réalisée est OUT (N),A. Il s'agit là d'une instruction que seul le langage machine peut exécuter, à savoir l'envoi sur le *port* n° N (N est contenu dans l'octet qui suit 211) du contenu du registre A.

Autrement dit, cette instruction envoie l'octet venant de la variable BASIC OUT, et stocké dans l'accumulateur, sur le port n° 127. Le terme PORT représente en fait une certaine combinaison de signaux sur les lignes (ou BUS) du connecteur arrière du ZX-81. Il en existe 256 possibles, et la carte 8ES est câblée de façon à utiliser le port n° 127. En même temps que la combinaison de signaux n° 127 informe la carte 8ES qu'un message lui est transmis, l'octet contenu dans A est recopié sur huit autres lignes et atteint ainsi la carte qui se charge de l'appliquer à ses sorties en représentation binaire.

L'instruction suivante possède le code 201, correspondant au mnémonique RET, et, tout comme une instruction RETURN en BASIC, renvoie vers le programme principal. Ici, le langage machine « repasse les commandes » au BASIC.

Nous laissons à nos lecteurs le soin d'étudier de la même façon le second programme machine, implanté à partir de l'adresse 16534, appelé par LET Q6 = USR 16534, et venant chercher un octet sur le port n° 127 pour le mettre dans l'accumulateur, d'où il sera tranféré à l'adresse 16527 avant qu'une instruction PEEK ne vienne le charger dans la variable BASIC IN.

Dans l'exemple que nous venons d'étudier, les différents octets des programmes en langage machine sont contenus dans autant d'instructions POKE d'un programme BASIC. Bien que l'utilisation de variables pré-initialisées, Q7, Q8 et Q9, fasse gagner un peu de temps lors de la frappe et un peu de place en mémoire, cette

façon de procéder est fastidieuse et deviendrait inexploitable pour le chargement de programmes machine plus longs. Certes, on peut effacer les lignes 6 à 18 après la première exécution du programme, puisque tous les codes machine ont émigré dans l'instruction REM. Cependant, il est souhaitable de mettre sur pied un moyen plus commode permettant de charger des programmes machine dans les meilleures conditions.

Nous conserverons le choix consistant à placer nos routines machine dans une instruction REM située en tête du programme BASIC, car ce procédé est le plus simple. Il ne faut cependant pas ignorer qu'il en existe d'autres, basés notamment sur l'emploi de tableaux ou sur l'usage d'espaces mémoire placés au-delà de RAM-TOP.

Le programme de la *figure 4-2* réserve un espace de 87 octets sous la forme d'autant de points dans l'instruction REM de la ligne 10. Ces adresses réservées depuis 16514 jusqu'à 16600 suffisent pour loger des routines machine déjà conséquentes. Pourquoi choisir une instruction REM pour abriter ces routines ? En premier lieu, le contenu d'une telle instruction est absolument libre puisque ignoré par le BASIC. De plus, une instruction REM apparaît clairement au listage et se laisse sauvegarder sur cassette. Enfin, placée en tête d'un programme BASIC, elle occupe une place fixe

```
  10 REM .....................
.....................................
.....................................
  12 LET N=16514
  15 LET L=1
  20 FAST
  25 PRINT "CODE DECIMAL Z 80 EN
";N+L-1;" ?"
  30 INPUT C$
  32 CLS
  35 IF C$="" THEN GOTO 62
  40 POKE N+L-1,VAL C$
  50 LET L=L+1
  60 GOTO 20
  61 LET N=16514
  62 SAVE "CHARGEU■"
  66 SLOW
  70 FOR F=N TO N+L-1
  80 PRINT F,PEEK F
  90 NEXT F
 100 REM COPYRIGHT 1982
```

Fig. 4.2.

en mémoire, ce qui permet de connaître à tout instant l'adresse de n'importe lequel de ses octets.

En effet, si dans un programme BASIC il est facile de déplacer telle ou telle ligne, il n'en va pas de même en langage machine, puisque les instructions ne sont pas numérotées. C'est donc uniquement leurs adresses mémoire qui permettent de les localiser, d'où l'intérêt de leur fixité.

On peut, de toute façon, s'attendre à des difficultés si, d'aventure, il s'avère nécessaire de déplacer ou intercaler une instruction...

Une fois lancé, le programme réclame successivement les octets (en représentation décimale) correspondant aux adresses successives. Ces adresses ne sont imprimées que pour fournir au programmeur des points de repère s'il venait à « perdre le fil » lors de la frappe d'un long programme machine. En pratique, il suffit d'entrer les octets dans leur ordre normal.

En fin de programme, une pression sur NEWLINE, sans qu'un octet n'ait été frappé, entraîne la clôture de la procédure d'acquisition et déclenche une sauvegarde automatique sur cassette. Cette précaution se justifie par le fait que, neuf fois sur dix, un incident au cours de l'exécution d'un programme machine entraîne la perte de tout le contenu de la mémoire, et il vaut mieux, dans ces conditions, assurer ses arrières !

Cela fait, la machine imprime un « listing machine » du programme, chaque octet étant précédé de son adresse, aux fins de vérification. Si la longueur du programme dépassait la capacité de l'écran, la suite pourrait être affichée en faisant : CONT NEWLINE.

Cette vérification achevée, on pourra lancer le programme en frappant : RAND USR 16514, ou en incorporant cette instruction dans un programme BASIC écrit à partir de la ligne 101 et pouvant être lancé par GOTO 100. On pourrait, bien sûr, effacer les lignes 12 à 100, mais en se privant alors de la possibilité de demander un listage en faisant : GOTO 68.

Présentation d'un programme machine

S'il n'existe guère qu'une seule façon de présenter le listing d'un programme BASIC ZX-81, en revanche, un programme en langage machine peut revêtir de nombreuses formes différentes, dont certai-

nes capables de dérouter complètement l'utilisateur du ZX-81. Pour notre part, nous représenterons nos programmes, tout au long de cet ouvrage, sous la forme du listing imprimé par notre programme de la *figure 4-2.* Nous ne chercherons pas à cacher que cette présentation est tout à fait inhabituelle, mais il se trouve qu'elle se prête très bien au chargement de routines machine sur le ZX-81 au moyen du programme qui a été fourni.

Fig. 4-3.

La *figure 4-3* donne un exemple, très court, de programme machine présenté de cette façon. Pour le charger, il suffit, après avoir lancé le programme de la *figure 4-2,* de rentrer dans l'ordre les octets 195, 130 et 64.

La séquence clavier est donc la suivante :

195 NEWLINE 130 NEWLINE 64 NEWLINE NEWLINE

Après la sauvegarde du programme sur cassette, on peut examiner un listage de contrôle, qui doit être une copie conforme de la *figure 4-3,* demander un listage BASIC, afin de constater les modifications intervenues à la ligne 10, puis lancer le programme machine par RAND USR 16514 NEWLINE.

On pourra être surpris par la réaction de la machine qui présente tous les symptômes d'un blocage. L'écran est vide, et aucune touche du clavier, pas même BREAK, ne semble agir.

En fait, la machine « boucle » perpétuellement sur ce programme machine d'une seule instruction 195, soit JP NN. Cette instruction est l'équivalent exact du GOTO en BASIC. Ici, le numéro de ligne habituellement associé à GOTO est remplacé par l'OMS et l'OPS d'une adresse, respectivement 130 et 64, correspondant, d'après la table de la *figure 2-2,* à l'adresse 16514 elle-même !

Ce programme se comporte donc comme le programme BASIC suivant :

10 GOTO 10

Une différence de taille, cependant : si un programme BASIC qui « boucle » peut être arrêté par un BREAK, en revanche, rien ne permet d'arrêter depuis le clavier notre programme machine. La raison tient à ce que le ZX-81, lorsqu'il travaille en BASIC, exécute régulièrement une routine machine de « lecture du clavier » capable de détecter l'appui sur BREAK. D'une façon générale, les programmes machine, dans lesquels n'est pas prévue explicitement une fonction d'interrogation du clavier, déconnectent entièrement celui-ci durant tout le temps de leur exécution.

Ici, nous avons volontairement créé le « bouclage », mais le même phénomène se produit fréquemment en cas d'erreur de programmation machine, et, notamment, lorsqu'on oublie de prévoir le retour au BASIC par une instruction RET ou équivalente.

Le cas se présente aussi quand une adresse de début de routine machine introduite dans la fonction USR ne correspond pas réellement au début d'un programme.

On découvre ici l'intérêt qu'il y a à stocker sur cassette les programmes machine *avant même* de les lancer. En cas d'échec, en effet, on peut les recharger très rapidement, les lister et modifier les octets incorrects par quelques commandes POKE bien choisies. Un exemple : dans le programme de la *figure 4-3,* ajoutons l'octet 201 à la fin en faisant : POKE 16517, 201. On ne constate aucune amélioration puisque le programme boucle *avant* de parvenir à cette instruction. Par contre, si nous faisons en plus POKE 16515, 133, la machine revient instantanément au BASIC puisque la première instruction, au lieu de se boucler sur elle-même, ordonne l'exécution de l'instruction placée en 16517, soit RET.

```
16514    195
16515    133
16516     64
16517    201
```

Fig. 4-4.

Ce nouveau programme, publié à la *figure 4-4,* ne présente vraiment aucun intérêt pratique ! Il va cependant servir d'exemple, permettant de montrer d'autres formes sous lesquelles un programme machine peut être présenté :

– *Liste des mnémoniques*

Les mnémoniques ne sont que des abréviations de mots anglais décrivant le fonctionnement des instructions. Ils facilitent le raison-

nement du programmeur mais ne sont pas compris par le Z-80. Il existe des programmes, dits *Assembleurs,* permettant d'écrire à l'aide des mnémoniques et effectuant la « traduction » en codes machine (octets). Ecrit de cette façon, notre programme prendrait l'allure suivante :

16514 JP 16517
16517 RET

– *Liste hexadécimale*

Le code hexadécimal est largement employé par les professionnels de l'informatique, car il présente certains avantages notables. Le ZX-81, pour sa part, n'accepte pas directement les codes hexadécimaux. Plutôt que de devoir effectuer sans cesse des conversions de code, soit par programme, soit manuellement, nous avons délibérément pris le parti de parler au ZX dans le langage qu'il comprend, afin d'éviter les ambiguïtés qui surgissent à coup sûr lorsque deux codes différents sont utilisés tour à tour. Nous avons beaucoup hésité avant de faire ce choix, car il creuse un fossé entre les utilisateurs de ZX-81 et les adeptes d'autres machines. Finalement, nous avons joué la simplification vis-à-vis du débutant qui, après tout, n'est censé lire ce livre que parce qu'il programme sur ZX-81 !

De toute façon, nous publions à la *figure 4-5* une table de conversion facilitant le passage entre codes décimaux, hexadécimaux et, même, binaires. Egalement, la littérature consacrée au Z-80 abonde de tableaux d'instructions directement numérotés en « hexa », ce qui achève de nous rassurer quant au sort de ceux de nos lecteurs qui, possédés par le démon informatique, en viendront à se tourner vers d'autres machines !

En représentation hexa, notre programme de la *figure 4-4* prendrait l'allure suivante :

16514 C3 85 40
16517 C9

ou encore celle-ci, si les adresses étaient aussi représentées en hexa :

4082 C3 85 40
4085 C9

On remarquera qu'il est fréquent d'enchaîner sur une même ligne les divers octets faisant partie d'une même instruction, ce qui permet de n'écrire que les adresses des octets de tête d'instruction.

2e chiffre

1er chiffre	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Figure 4-5. Equivalence entre codes hexadécimaux et décimaux

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

Correspondance entre chiffres hexadécimaux et binaires

Avec cette convention, notre listing en code décimal de la *figure 4-4* pourrait prendre la forme suivante :

16514 195 133 64
16517 201

On pourra, bien sûr, rencontrer des représentations mixtes faisant apparaître à la fois mnémoniques et codes machine. En voici deux exemples, nullement limitatifs :

4082 C3 85 40 JP 4085 (en hexadécimal)
4085 C9 RET

16514 195 133 64 JP 16517 (en décimal)
16517 201 RET

Nous espérons que ces indications épargneront à nos lecteurs les nombreux casse-tête que nous avons dû résoudre lors de nos tentatives visant à charger sur le ZX-81 des programmes machine extraits de publications non destinées à cet ordinateur.

Lors de telles tentatives, une grande vigilance reste cependant de rigueur, car rien ne prouve que le programme considéré soit prévu pour être implanté à partir de l'adresse 16514. Dans le cas contraire, on pourrait s'attendre à des déboires identiques à ceux qui surviendraient si, d'aventure, on se permettait de changer les numéros de lignes d'un programme BASIC contenant des GOTO ou des GOSUB.

Signalons, cependant, que le langage machine Z-80 possède des instructions spéciales permettant de contourner ces difficultés et rendant les programmes qui les utilisent « *relogeables* ». Un programme relogeable peut être implanté n'importe où en mémoire mais est nettement plus délicat à écrire.

Toutes ces remarques renforcent encore notre conviction selon laquelle le ZX-81, utilisé sous langage machine, constitue un cas un peu à part, et qu'il convient de le traiter comme tel, sans trop chercher à lui adapter des principes s'appliquant à d'autres systèmes à Z-80.

Au risque de nous répéter, nous insisterons sur le fait que la meilleure façon d'utiliser le langage machine nous semble décidément être l'incorporation, dans des programmes BASIC, de routines machine destinées à en augmenter les possibilités.

C'est d'ailleurs cette voie que semblent suivre les très nombreux utilisateurs anglais de ZX-81, qui obtiennent, nous avons pu le vérifier sur place, des résultats tout à fait extraordinaires.

Notre premier programme machine utilitaire

Nos lecteurs se souviennent que c'est lors d'une tentative de mise en « oscillation » de la carte 8ES que le BASIC a révélé son impuissance à travailler à haute vitesse. Il était donc naturel que nous cherchions à résoudre ce problème au moyen du langage machine. Le résultat de ces recherches apparaît à la *figure 4-6,* sous la forme d'un programme « sans retour » capable de faire commuter les huit sorties de la carte 8ES toutes ensemble, à la fréquence de 50 kHz environ ! On notera que, pour obtenir cette performance, il est nécessaire de placer le ZX-81 en mode rapide, faute de quoi l'exécution répétée des routines d'affichage (même pour un écran blanc) viendrait nous faire perdre un temps précieux, limitant ainsi la cadence des commutations à quelques dizaines de hertz.

```
16514   62
16515   0
16516   211
16517   127
16518   62
16519   255
16520   211
16521   127
16522   195
16523   130
16524   54
```

Fig. 4-6.

Bien évidemment, la vérification des chiffres annoncés exige le recours à un oscilloscope, que l'on pourra brancher aux bornes de l'une des LED de la carte 8ES.

Le programme utilise à peu près les mêmes instructions que dans sa version d'origine, la différence essentielle se situant au niveau du chargement du registre A : au lieu de recopier le contenu d'une cellule mémoire dans A, le programme exécute des instructions LD, A,N (code 62), lui imposant tantôt la valeur 0, tantôt la

valeur 255. En fin de programme, une instruction JP 16514, désormais bien connue, permet le « bouclage » du programme sur lui-même.

Une routine de test

Le programme de la *figure 4-7* est une version « machine », donc plus rapide, du programme de test proposé à la *figure 3-5.* Il est dérivé du programme d'origine par simple enchaînement des routines d'entrée et de sortie, par la suppression des instructions RET et par un bouclage par JP 16514.

```
16514   219
16515   127
16516   50
16517   143
16518   64
16519   58
16520   143
16521   64
16522   211
16523   127
16524   195
16525   130
16526   64
```

Fig. 4-7.

La *figure 4-8* propose une variante plus courte, dont l'analyse représentera pour nos lecteurs un bon exercice de lecture de code machine.

Il sera intéressant de comparer la rapidité de réaction des deux programmes machine avec celle du programme BASIC de la *figure 3-5 !*

```
16514   219
16515   127
16516   211
16517   127
16518   195
16519   130
16520   64
```

Fig. 4-8.

Une routine d'aide au BASIC

Nous abordons ici l'un des domaines privilégiés de la programmation du ZX-81 en langage machine : le remplacement de sous-programmes BASIC, manifestement peu à l'aise pour accomplir une tâche donnée. Revenons donc à notre programme de la *figure 1-2* (labyrinthe) qui, à un moment donné de son exécution,

```
10 FOR F=1 TO 704
20 PRINT CHR$ 128;
30 NEXT F
70 REM COPYRIGHT 1982
```

Fig. 4-9.

construit un grand rectangle noir sur l'écran. Les instructions utilisées sont analogues à celles regroupées à la *figure 4-9.* Lançons donc ce court programme BASIC et constatons le temps que met la machine à exécuter ce travail. A titre de comparaison, chargeons le programme machine de la *figure 4-10* et lançons-le par RAND USR 16514 NEWLINE. Cette fois, le rectangle noir apparaît instantanément, même si le ZX-81 est placé en mode lent. Voilà qui en dit long sur l'intérêt qu'il y a à incorporer de semblables routines machine dans des programmes BASIC. On pourrait imaginer, en effet, un programme de jeu exigeant un tel noircissement de l'écran

```
16514    42
16515    12
16516    64
16517    6
16518    23
16519    43
16520    35
16521    126
16522    254
16523    118
16524    32
16525    3
16526    16
16527    248
16528    201
16529    54
16530    128
16531    24
16532    243
```

Fig. 4-10.

de façon répétée. La patience des joueurs se trouverait mise à rude épreuve si seules les ressources du BASIC étaient disponibles.

Avantage supplémentaire, la *figure 4-11* montre à quel point cette routine machine occupe peu de place en mémoire : on pourra comparer avec le programme BASIC de la *figure 4-9 !*

```
   10 REM E£RND■*F7 SAVE TAN ■
NEXT
```

Fig. 4-11.

Faible encombrement mémoire, extrême rapidité d'exécution et possibilités interdites au BASIC d'origine, voilà donc les trois gros avantages de la programmation en langage machine.

Toute médaille ayant son revers, il ne faut pas cacher que l'écriture des programmes machine reste considérablement plus délicate que celle des programmes BASIC.

C'est pourquoi nous pensons que le débutant doit aborder l'écriture de ses programmes machine personnels en se limitant à l'usage d'un nombre raisonnable d'instructions, choisies parmi les plus simples, et surtout celles dont le mode de fonctionnement ressemble à celui d'instructions BASIC qu'il maîtrise bien (1).

Nous allons donc, dans le chapitre suivant, faire un tour d'horizon des principales instructions BASIC, en étudiant les possibilités correspondantes offertes par le langage machine. La table des instructions Z-80 située en fin d'ouvrage, qui contient *toutes* les instructions machine, fournit d'ailleurs chaque fois que la chose est possible, une analogie avec le BASIC.

Lorsque le lecteur se sentira familiarisé avec les instructions les plus simples, il pourra passer progressivement à la mise en œuvre des suivantes, qu'il découvrira peu à peu. A ce niveau, il pourra aborder de plain-pied la lecture des ouvrages traitant de la programmation du Z-80, qui détaillent le fonctionnement de *toutes* les instructions. Le présent ouvrage ne pourrait, en aucun cas, remplir ce rôle sans voir son volume quadrupler, au minimum, et sans faire double emploi avec la littérature existante, que nous n'avons ni le désir, ni les moyens de concurrencer.

(1) Voir figure 6-1.

Chapitre 5

Du BASIC au langage machine

Le précédent chapitre, s'il a atteint son but, devrait avoir donné à nos lecteurs les indications voulues pour leur permettre de charger sur le ZX-81 des programmes machine (ce qui est loin d'être évident lorsque l'on ne dispose que des renseignements accessibles au possesseur moyen de ZX-81 !).

La démonstration a été faite des avantages propres à ce type de programmation, sans pour autant en cacher les inconvénients, et des exemples ont été « démontés » en vue d'expliquer le fonctionnement d'un tout petit nombre d'instructions.

Il reste ici à étudier d'autres instructions, en nombre suffisant pour donner à nos lecteurs les moyens de programmer seuls leur ZX-81 en langage machine, afin de résoudre leurs propres problèmes.

Instructions de type REM

L'instruction BASIC REM joue un rôle particulier puisqu'elle est totalement ignorée lors de l'exécution.

Nous avons vu qu'elle est loin de ne servir qu'à accueillir des « commentaires », ce qui était pourtant sa vocation première.

En langage machine Z-80, la seule instruction comparable s'appelle NOP (code décimal 0), lors de l'exécution de laquelle la machine n'entreprend aucune tâche. On peut utiliser des NOP chaque fois que l'on désire faire perdre un tout petit peu de temps (une micro-seconde environ !) à l'ordinateur, mais surtout lorsque l'on ressent le besoin de *réserver* une ou plusieurs adresses mémoire

dans le cours d'un programme. Une telle précaution permet d'insérer des instructions après coup sans avoir à réécrire entièrement le programme. De même, lorsque la suppression d'instructions s'avère nécessaire, le remplacement par des NOP peut être une solution.

Instructions de type STOP

L'équivalent machine de l'instruction BASIC STOP est HALT (code 118). *Il ne faut pas* l'utiliser dans des programmes machine appelés à partir du ZX-81, car l'exécution de HALT arrête complètement la machine, qui ne peut même plus revenir au BASIC. Seule solution, couper le courant ! Notons que les systèmes travaillant d'origine en langage machine peuvent être relancés après un HALT, mais ce n'est pas le cas du ZX-81, sauf modification.

Instructions de type LET

L'instruction LET est, en BASIC, une instruction *d'affectation.* On l'utilise pour *affecter* à une *variable BASIC* soit une valeur numérique explicite, soit la valeur d'une autre variable, soit le résultat d'une opération portant sur des variables et/ou des constantes numériques. Des applications de LET existent également dans le domaine du traitement des chaînes.

En langage machine, énormément d'instructions peuvent se rattacher à la famille des LET.

En premier lieu, toutes les instructions dont le mnémonique commence par LD (load). Ces instructions de *chargement* affectent en effet un *octet* à un registre ou à une cellule mémoire. Les différences se situent au niveau des *modes d'adressage,* puisque la *destination,* comme d'ailleurs la *provenance* de l'octet (ou parfois *des* octets), peut être contenue dans *l'instruction elle-même,* dans une *cellule mémoire* dont le numéro (l'adresse) fait partie de l'instruction (sous la forme OMS suivi de OPS), dans un *registre* dont le nom est contenu dans l'instruction, ou même dans une cellule mémoire dont l'adresse est contenue dans deux registres, et ce n'est pas tout !

Nous abordons ici le domaine complexe des *modes d'adressage* du Z-80. Ce microprocesseur performant est en effet l'un de ceux qui possèdent le plus de procédures distinctes pour désigner les

octets à traiter. En fait, le Z-80 possède *dix* modes d'adressage différents, et il serait irréaliste de penser qu'un débutant pourrait en manier plus de deux ou trois d'emblée. On retrouve ici notre point de vue général selon lequel il faut délimiter un « langage machine restreint », spécial pour débutants, puis l'étoffer petit à petit au fur et à mesure de l'apparition de nouveaux besoins, tout comme les limites du BASIC entraînent tout naturellement vers la programmation machine.

Les quelques exemples qui vont suivre n'épuiseront donc pas le sujet des instructions de chargement, loin s'en faut, mais donneront aux moins à nos lecteurs *la clé du code* permettant de comprendre le fonctionnement d'une instruction à partir de son mnémonique.

La remarque *fondamentale* qu'il faut faire d'entrée est que, dans un mnémonique Z-80, la localisation annoncée en premier est celle de *destination* de l'information, et que toute localisation placée *entre parenthèses* doit être interprétée comme une *cellule mémoire* dont l'adresse est donnée par le contenu des parenthèses. Voici quelques exemples :

– *Instruction LD A,N* (code décimal 62) : l'octet N, qui suit le code 62, est chargé dans le registre A. Le programme suivant :

```
16514  62
16515 255
```

chargerait la valeur 255 dans l'accumulateur.

– *Instruction LD B,A* (code 71) : l'octet contenu dans l'accumulateur est recopié dans le registre B.

– *Instruction LD A, (NN) (code 58) :* l'octet contenu dans la cellule mémoire, dont l'adresse est donnée par les deux octets qui suivent le code 58, est chargé dans l'accumulateur A. Ce programme :

```
16514  58
16515 130
16516  64
```

chargerait dans le registre A l'octet contenu dans la cellule mémoire n° 130 + (256 × 64) = 16514, autrement dit 58 !

– *Instruction LD (NN), A* (code 50) : l'octet contenu dans le registre A est chargé dans la cellule mémoire dont l'adresse est donnée par les deux octets qui suivent le code 50.

– *Instruction LD A,(BC)* (code 10) : l'octet contenu dans la cellule mémoire, dont l'adresse est donnée par les deux octets contenus dans les registres B et C, est chargé dans le registre A.

Le langage machine est capable d'exécuter des instructions tenant sur un seul octet et remplaçant des instructions BASIC relativement complexes, du genre LET L = L + 1, ou LET L = L – 1. Ces instructions sont dites *d'incrémentation* (INC) ou de *décrémentation* (DEC).

Par exemple, l'instruction INC B (code 4) augmente d'une unité le contenu du registre B, alors que DEC B (code 5) le diminue d'une unité. Mais nous entrons déjà dans le domaine des opérations arithmétiques.

Instructions de type arithmétique ou logique

Le microprocesseur Z-80 possède une *unité arithmétique et logique* qui ne sait faire que des opérations très élémentaires (additions et soustactions, avec ou sans retenue, décalage de bits, etc.). Ce n'est que par le biais de sous-programmes machine que le Z-80 peut exécuter multiplications, divisions et, à plus forte raison, fonctions trigonométriques ou logarithmiques ! On réservera donc au BASIC les calculs d'un certain niveau, pour ne soumettre au Z-80 que les opérations les plus « naïves » que sont les additions et soustractions simples.

Par exemple, l'instruction ADD A,B (code 128) additionne les deux octets contenus respectivement dans les registres A et B. Le résultat est rangé dans le registre A où il « écrase » le précédent contenu. Si le résultat de l'addition est supérieur à 255, il est automatiquement amputé de 256, et le bit indicateur de retenue du registre F (drapeau C) est mis à 1, tenant ainsi lieu de *retenue.* Cependant, cette instruction ne sait pas tenir compte de retenues antérieures. C'est pourquoi il existe aussi l'instruction ADC A,B (code 136) qui ajoute une unité au résultat de l'addition de A et B si le drapeau C est à 1. C'est cette transmission de retenues qui permet les calculs sur des nombres sortant de l'intervalle 0 à 255, notoirement insuffisant !

De telles instructions d'addition existent entre une large variété de registres et sont complétées par autant d'instructions de soustraction, avec ou sans retenue (voir table des instructions).

Les opérations logiques réclament, pour être correctement utilisées, des connaissances mathématiques qui sortiraient nettement du cadre de cet ouvrage d'initiation. Notons simplement que les fonctions de type ET, OU, OU EXCLUSIF, opèrent bit par bit entre

deux octets, formant de la sorte un troisième octet qui est rangé à l'endroit prévu par l'instruction. Des opérations de décalage, avec ou sans retenue, de rotation et permutation de bits, etc., sont aussi disponibles, facilitant la réalisation de fonctions mathématiques complexes.

Instructions de type INPUT

Le langage machine ne prévoit pas directement d'instructions permettant l'entrée de données par le clavier. Il faudrait écrire un programme d'interrogation des touches, qui existe déjà en ROM et qui est très largement utilisé par le BASIC. C'est le plus souvent par l'intermédiaire du BASIC (fonctions POKE) que l'on charge des informations dans certaines cellules mémoire où le programme machine viendra les chercher.

Une possibilité inestimable du langage machine est cependant de prélever des octets sur les *ports* d'entrée-sortie, et tout spécialement sur le port n° 127, dévolu à la carte 8ES.

Il existe plusieurs variantes de l'instruction IN selon le mode d'adressage mis en œuvre, notion que nos lecteurs peuvent désormais comprendre. Signalons cependant une particularité d'écriture des mnémoniques des instructions IN : les parenthèses renferment le *numéro du port* et non une adresse mémoire.

Instructions de type GOTO ou GOSUB et RETURN

Les instructions de branchement sont un outil de programmation tout à fait essentiel.

Nous avons déjà utilisé l'instruction machine de branchement inconditionnel JP NN (code 195), équivalent exact du GOTO en BASIC. L'instruction GOSUB possède également son homologue machine, l'instruction CALL NN (code 205). Elle ne diffère de la précédente que par le fait qu'elle ne se débranche que provisoirement du programme principal, pour en reprendre l'exécution à partir de l'instruction suivant CALL NN, dès qu'une instruction de retour sera détectée.

Une instruction RET (code 201) placée en fin d'un sous-programme machine *appelé par le programme machine principal* ne renvoie donc pas au BASIC comme dans les exemples précédents. D'une façon générale, l'instruction RET renvoie au programme (quel qu'il soit) duquel a été appelé le sous-programme machine auquel elle met un point final.

Nous ne nous sommes intéressés jusqu'à présent qu'aux branchements *inconditionnels,* impérativement exécutés lors du passage de l'instruction correspondante. En BASIC, on utilise largement des *branchements conditionnels* du type IF condition THEN GOTO (ou GOSUB). Cette possibilité existe bien sûr en langage machine, avec même une souplesse supérieure.

Instructions de type IF-THEN

Dans cette famille peuvent être classées toutes les instructions de *comparaison* (et elles sont nombreuses) ainsi que les instructions *conditionnelles,* notamment de branchement. Toutefois, la plupart des instructions « ordinaires » peuvent aussi servir à effectuer des comparaisons de nature à entraîner des branchements. Le « centre nerveux » de toutes les instructions conditionnelles est le *registre F* et les *drapeaux* qu'il renferme. Ces drapeaux sont des bits qui se trouvent positionnés à 1 ou à 0 selon les caractéristiques du résultat de la *dernière* instruction exécutée. On notera donc qu'il faut utiliser sans tarder le résultat de toute comparaison, ou le considérer comme perdu ! En effet, presque toutes les instructions sont susceptibles de modifier tout ou partie du contenu du registre F.

Le drapeau le plus fréquemment utilisé est *l'indicateur de zéro,* tout naturellement nommé Z. Ce drapeau se trouve mis à 1 par toute instruction ramenant à zéro le contenu de l'accumulateur A lors de son exécution.

Les instructions BIT ont le même effet, mettant à 1 le drapeau Z si le bit spécifié par le texte de l'instruction est à zéro. Enfin, les instructions de comparaison CP mettent aussi le drapeau Z à 1 s'il y a égalité entre l'octet spécifié dans l'instruction et l'accumulateur.

Dans tous les cas qui viennent d'être évoqués, si les conditions permettant de mettre le drapeau Z à 1 ne sont pas remplies, celui-ci est mis à zéro *quel que soit son état précédent.* Il n'est donc nullement nécessaire de lancer une comparaison pour que l'état de F puisse changer !

Quelle que soit la cause du changement d'état du drapeau F, celui-ci peut influer sur l'exécution de tout un groupe d'instructions conditionnelles contenant dans leur mnémonique les abréviations Z ou NZ. Les instructions en Z sont exécutées si Z est à 1 et ignorées dans le cas contraire. Inversement, les instructions en NZ ne s'exécutent que si Z est à zéro.

```
16514          58
16515          33
16516          64
16517          61
16518          194
16519          133
16520          64
16521          201


  10 REM USRNDXTAB RNDTAN .....
.....
.....
.........
  11 INPUT D
  12 POKE 16417,D
  13 RAND USR 16514
  14 GOTO 11
  15 REM COPYRIGHT 1982
```

Fig. 5-1.

Le programme de la *figure 5-1* met en œuvre ces instructions, ainsi que celles qui ont été passées en revue depuis le début de ce chapitre. Il s'agit d'un sous-programme de *temporisation,* permettant de faire perdre du temps à la machine dans des proportions très supérieures à ce que permet l'insertion d'instructions NOP. On retrouve donc ici l'équivalent de l'instruction PAUSE du BASIC, bien que les retards mis en œuvre soient considérablement plus courts.

Le principe, très simple, consiste à charger dans le registre A une valeur comprise entre 0 et 255, et qui fixera la durée de la temporisation. Nous avons choisi de stocker cette valeur dans la cellule mémoire n° 16417, ce qui permet de l'y amener soit à partir du BASIC, soit à partir du langage machine. La première instruction du programme est donc : LD A,(16417).

Le programme décrémente ensuite le registre A, et renouvelle cette opération tant que son contenu n'atteint pas zéro. A ce

moment, en effet, l'instruction JP NZ,16517 est ignorée, et le programme, au lieu de boucler, atteint l'instruction RET qui le renvoie au programme principal. Il est clair que plus la valeur mise dans la cellule 16417 sera grande et plus le programme mettra de temps à s'exécuter, puisqu'il boucle un nombre de fois égal au nombre contenu dans cette cellule.

Pour voir ce programme à l'œuvre, il est commode de l'appeler à partir d'un très court programme BASIC, contenu dans les lignes 11 à 15 qui suivent l'instruction REM abritant le code machine.

On pourra frapper ces lignes après le chargement du programme machine, sans même prendre la peine d'effacer le programme chargeur, puisque le programme BASIC est conçu sous forme d'une boucle dont la machine ne pourra sortir seule.

Lançons donc ce programme par RUN NEWLINE et entrons diverses valeurs comprises entre 0 et 255.

On pourra être déçu de constater que le retour au curseur L se fait quasi-instantanément, quelle que soit la valeur frappée avant NEWLINE. En fait, cette expérience est là pour montrer encore une fois à quel point le langage machine est rapide, et que c'est en millionnièmes de secondes qu'il convient de chiffrer ses pauses ! La variante proposée à la *figure 5-2* autorise une expérimentation à l'échelle humaine, grâce à l'imbrication de deux bouclages, l'un réglable comme précédemment, mettant en œuvre le registre A, et l'autre, fixé à 256 exécutions, opérant sur le registre B.

Après avoir lancé le programme BASIC, entrons des valeurs comprises toujours entre 0 et 255 et constatons le retard variable avec lequel le curseur L revient à l'écran. Il convient cependant de méditer le fait que, pour une valeur frappée de 255 (le maximum), le retour au BASIC ne prend guère plus d'une seconde, malgré l'exécution de 256 × 256 = 65536 boucles ! (en mode lent...).

Ce programme pourrait être appelé à partir du langage machine au moyen d'une instruction CALL 16514, et pourrait même être implanté ailleurs en mémoire, à condition que soient modifiées en conséquence les adresses incorporées dans les deux instructions JP NZ.

Nous avons fait fonctionner ici des branchements conditionnels opérant sur l'indicateur de zéro, mais il existe d'autres drapeaux pouvant être utilisés de façon analogue.

L'indicateur C rend compte de l'existence ou de l'absence de retenue dans une opération arithmétique ou logique, mais peut

aussi être forcé à 1 par l'instruction SCF (code 55) ou complémenté par l'instruction CCF (code 63). Notons que le forçage à zéro ne bénéficie pas d'une instruction à part, puisqu'il suffit de faire SCF puis CCF. Les instructions utilisant ce drapeau sont celles dont le mnémonique contient les abréviations C ou NC.

```
16514           58
16515           33
16516           64
16517           6
16518           255
16519           5
16520           194
16521           135
16522           64
16523           61
16524           194
16525           133
16526           64
16527           201
```

```
  10 REM USRND* COPY ▌TAB ▪RNDXT
AB ▌RNDTAN ....................
....................................
.....................
  11 INPUT D
  12 POKE 16417,D
  13 RAND USR 16514
  14 GOTO 11
  15 REM COPYRIGHT 1982
```

Fig. 5-2.

Citons encore le drapeau P, qui est mis à 1 si le résultat d'une instruction est pair ou si un dépassement se produit, mais qui reste à zéro dans le cas contraire. Les instructions contenant l'abréviation PO réagissent si P est à zéro, alors que celles en PE ne se soucient que d'un drapeau P à 1.

L'indicateur de *signe* se nomme S, et passe à zéro dans le cas d'un octet positif et à 1 pour un octet négatif. Les instructions du type « SI positif » ou « IF>0 » contiennent l'abréviation P, la lettre M étant utilisée pour NEGATIF, puisque N sert déjà par ailleurs.

Instructions de type FOR-NEXT

Une application très courante des branchements conditionnels (d'ailleurs reprise dans notre exemple) est l'exécution d'une certaine portion de programme un nombre déterminé de fois. Pour ce faire, tout comme en BASIC, le recours à une incrémentation (ou à une décrémentation) est une solution, mais il s'est souvent avéré, en BASIC, que l'utilisation de *boucles FOR-NEXT* présentait des avantages.

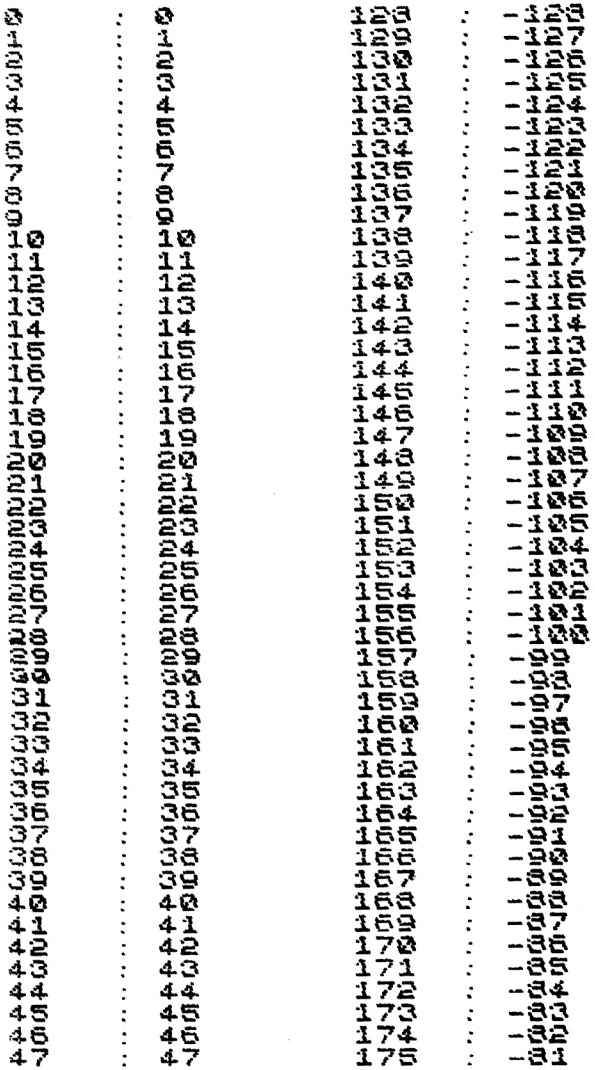
En langage machine, il existe une possibilité similaire grâce à l'instruction DJNZ.

Cette instruction complexe exécute trois tâches successivement : le registre B voit son contenu décrémenté d'une unité, puis est testé. Si le nouveau contenu de B n'est pas nul (dans le cas contraire, l'instruction suivante est exécutée), la machine exécute un *branchement relatif* selon les indications fournies par l'octet qui suit le code opératoire de DJNZ, soit 16.

Nous n'avons envisagé jusqu'à présent que des instructions de branchement *absolu,* l'adresse de l'instruction à exécuter étant contenue de façon explicite soit dans l'instruction de branchement elle-même, soit dans des registres ou cellules mémoires spécifiés. Ici, l'octet D indique un *déplacement,* c'est-à-dire un nombre d'adresses duquel il faut sauter, en avant ou en arrière, pour trouver la prochaine instruction à exécuter. Pour qu'un seul octet, dont la valeur ne peut sortir de l'intervalle 0-255, puisse indiquer des déplacements indifféremment positifs ou négatifs, il fallait définir des conventions, qui ne sont autres que le code dit « complément à deux », dont la *figure 5-3* donne la table complète.

Une remarque s'impose immédiatement : seuls des sauts compris entre + 127 et – 128 adresses pourront être obtenus à partir de cet adressage relatif. Toutefois, cette limite ne devrait pas poser de problème notable dans le cadre de routines machine assez courtes. Nous avons utilisé l'instruction DJNZ (sans l'expliquer) dans notre programme de la *figure 4-10,* à laquelle nous invitons nos lecteurs à se reporter s'ils désirent approfondir cette question. Nous estimons cependant que l'emploi des instructions à adressage relatif (contenant la lettre d dans leur mnémonique) est à déconseiller aux débutants. En effet, le seul avantage de ce mode d'adressage est de rendre *relogeables* les programmes qui l'utilisent de façon massive. Les programmes écrits dans le cadre de cet ouvrage sont suffisam-

ment courts pour pouvoir être facilement relogés à la main et affectionnent tout particulièrement les adresses mémoire suivant 16514, pour lesquelles la *figure 2-2* donne une table facilitant beaucoup l'adressage immédiat.



(suite au verso)

```
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
```

```
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
```

```
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
```

```
-80
-79
-78
-77
-76
-75
-74
-73
-72
-71
-70
-69
-68
-67
-66
-65
-64
-63
-62
-61
-60
-59
-58
-57
-56
-55
-54
-53
-52
-51
-50
-49
-48
-47
-46
-45
-44
-43
-42
-41
-40
-39
-38
-37
-36
-35
-34
-33
-32
-31
-30
-29
-28
-27
-26
```

103	:	103	231	:	-25
104	:	104	232	:	-24
105	:	105	233	:	-23
106	:	106	234	:	-22
107	:	107	235	:	-21
108	:	108	236	:	-20
109	:	109	237	:	-19
110	:	110	238	:	-18
111	:	111	239	:	-17
112	:	112	240	:	-16
113	:	113	241	:	-15
114	:	114	242	:	-14
115	:	115	243	:	-13
116	:	116	244	:	-12
117	:	117	245	:	-11
118	:	118	246	:	-10
119	:	119	247	:	-9
120	:	120	248	:	-8
121	:	121	249	:	-7
122	:	122	250	:	-6
123	:	123	251	:	-5
124	:	124	252	:	-4
125	:	125	253	:	-3
126	:	126	254	:	-2
127	:	127	255	:	-1

Fig. 5-3. – Table des déplacements en « complément à deux ».

Instructions de type PRINT, LPRINT, COPY

Le langage machine ne prévoit pas d'autres instructions de sortie d'information que celles permettant de diriger des octets sur les 256 ports d'entrée-sortie. En BASIC, la gestion de l'écran TV est confiée à un circuit intégré spécialisé, qui dialogue avec le Z-80 par l'intermédiaire de certains de ces ports. Pour ce faire, il existe en ROM des sous-programmes machine chargés de traiter comme il convient les caractères à afficher ou à imprimer.

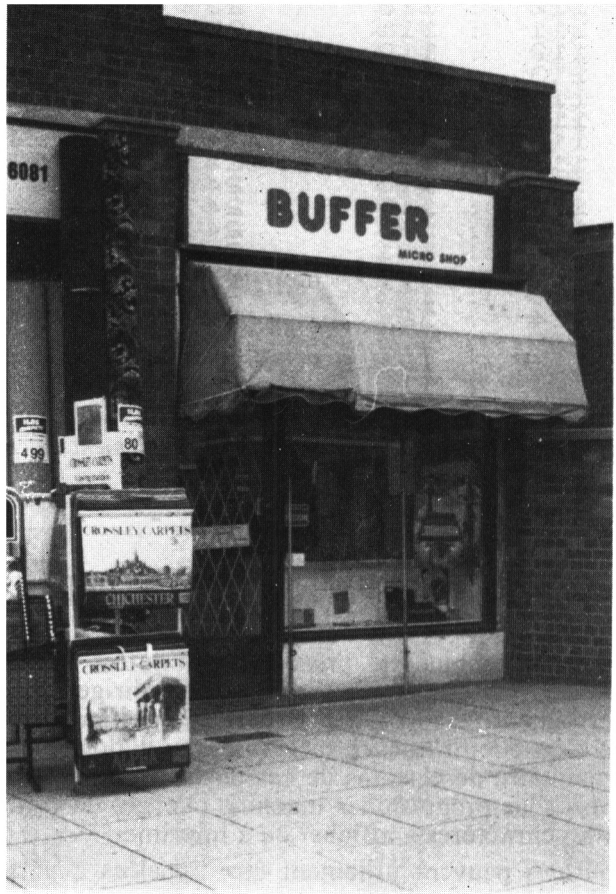
Ces routines peuvent utilement être appelées à partir de programmes BASIC ou machine au moyen d'instructions appropriées.

Le problème majeur consiste à localiser parmi les 8 192 adresses existant en ROM celle, unique, à laquelle démarre chaque routine puis à en étudier le fonctionnement dans le détail.

Il est vraiment dommage que les créateurs du ZX-81 se montrent aussi jaloux des petits secrets de leur ROM, se limitant à

donner très discrètement, dans le manuel de l'imprimante, la localisation du générateur de caractères et de la routine LPRINT.

Nous avons dû nous livrer à un véritable travail de fourmi pour obtenir quelques indications que nous publions ici à l'intention de nos lecteurs.



Ceux de nos lecteurs ayant l'occasion de passer par Londres pourront y trouver une multitude de toutes petites boutiques micro-informatique comme celle-ci, située près du terminus du bus 133 (Streatham Garage). Qu'ils n'espèrent pas réaliser des affaires mirobolantes, car les prix sont pratiquement les mêmes qu'en France. Il est par contre possible de revenir chargé d'une moisson de renseignements de tout premier ordre !

Le procédé d'espionnage (!) que nous avons utilisé au début de nos recherches consistait à faire imprimer par la machine elle-même une longue liste de toutes les instructions de *retour* (RET notamment) contenues dans la ROM. Il est logique de penser que, après un branchement inconditionnel, une nouvelle routine a toutes les chances de commencer. Il reste donc à essayer des RAND USR sur toutes les adresses supposées de débuts de routines. Bien sûr, des résultats extravagants sont très souvent obtenus, mais cette méthode peut s'avérer payante, puisque c'est ainsi que nous avons pu découvrir que RAND USR 2153 déclenche la fonction COPY.

Pour en savoir plus, il nous a cependant fallu traverser la Manche, car, en Grande-Bretagne, même les enfants de quinze ans pratiquent le langage machine comme des programmeurs de carrière...

Peu à peu, nous avons pu obtenir quelques « tuyaux » jalousement gardés au pays des ZX.

Parmi ceux-ci, le plus intéressant est peut-être l'adresse de départ de la routine de visualisation d'un caractère sur l'écran TV. Ce sous-programme, l'un des plus fréquemment appelés par le BASIC, est logé tout au début de la ROM, à partir de l'adresse 16. Pour l'utiliser, il suffit de charger dans le registre A l'octet correspondant au code du prochain caractère à afficher, puis d'appeler le sous-programme de la ROM par un CALL 16 (205,16,0). A titre d'illustration, la *figure 5-4* propose un petit programme construi-

```
16514    62
16515    128
16516    205
16517    16
16518    0
16519    195
16520    130
16521    64
```

```
  10 REM Y■LN ( ?■RND.....
.....
.....
....
  11 RAND USR 16514
  12 REM COPYRIGHT 1982
```

Fig. 5-4.

sant à nouveau un écran noir. Il est moins rapide que celui de la *figure 4-10,* qui intervenait directement dans le fichier d'affichage. En revanche, il permet de suivre de visu la construction du rectangle.

```
10 PRINT CHR$ 128;
20 GOTO 10
```

Fig. 5-5.

On notera que, tout comme son équivalent BASIC donné à la *figure 5-5,* ce programme n'est pas destiné à servir de sous-programme, car il débouche sur un arrêt par débordement d'écran. Une amélioration pourrait consister à compter, grâce à une boucle appropriée, le nombre d'impressions exécutées et à lancer un retour au programme principal dès que les 704 caractères sont atteints.

Utilisation des routines de la ROM

Certaines autres routines peuvent avantageusement être mises à contribution, et tout spécialement le sous-programme de lecture du clavier. Normalement, nous avons vu que le clavier est totalement neutralisé lorsqu'un programme machine est en cours d'exécution. La routine commençant à l'adresse 699 (OMS = 187, OPS = 2) charge la valeur 255 dans chacun des deux registres H et L si, lors de son exécution, le clavier est au repos. Dans le cas contraire, elle charge des codes dont chaque touche possède une paire bien à elle. On notera que, tout comme la fonction INKEY$ en BASIC, cette routine n'attend pas ! Il faut donc la lancer périodiquement pour pouvoir interroger le clavier en temps réel.

Le programme de la *figure 5-6* donne un exemple de l'utilité que peut revêtir cette routine : il s'agit, en effet, d'un programme machine qui « boucle » sur lui-même, et que rien ne permettait jusqu'à présent d'arrêter autrement que par coupure de l'alimentation. A présent, il suffit d'appuyer sur la touche SHIFT pour que la machine revienne au BASIC.

Le programme place en effet l'octet 255 dans l'accumulateur et compare cet octet à celui présent dans le registre H en fin de routine 699. Tant que H contient 255, le programme est bouclé par un JP Z, 16517 (202, 130, 64), mais exécute un RET (code 201) dès que le contenu de H devient autre que 255.

```
16514    205
16515    187
16516    2
16517    62
16518    255
16519    183
16520    202
16521    130
16522    64
16523    201
```

Fig. 5-6.

Cela se produit lorsque la touche SHIFT est enfoncée et aussi lors de l'appui sur d'autres touches, moins pratiques d'emploi. On peut donc incorporer ce petit programme à tout programme machine qu'il est souhaitable de pouvoir arrêter manuellement, mais il est indispensable de le faire appeler régulièrement, par exemple deux à dix fois par seconde.

Beaucoup d'autres routines du moniteur seraient sans nul doute d'un grand intérêt pour l'utilisateur, mais, même lorsque leurs adresses en ROM sont connues, les détails manquent quant à leur fonctionnement qui s'imbrique avec celui d'autres sous-programmes. C'est notamment le cas des routines SAVE et LOAD, dont d'intéressantes applications ont été développées par des radioamateurs anglais en vue du décodage et du codage en temps réel de signaux morse, ou RTTY. Il ne fait aucun doute que les Britanniques disposent des informations intéressantes environ dix mois avant les « continentaux », d'où l'intérêt de fréquentes « expéditions » !

Chapitre 6

Le jeu d'instructions machine du ZX-81

Le lecteur qui nous aura suivi au fil des chapitres précédents dispose désormais des connaissances de base lui permettant de se lancer seul dans l'écriture de petits programmes machine, de les entrer sur le ZX-81, de les lancer et les arrêter, enfin de les sauvegarder sur cassette.

Nous avons fourni des exemples de complexité croissante, mettant en relief les possibilités d'instructions variées.

Il est bien entendu impossible, dans le cadre d'un ouvrage d'initiation comme celui-ci, d'étudier toutes les instructions du Z-80. La table complète située à la fin de ce chapitre montre en effet qu'il en existe des centaines, dont certaines ne font d'ailleurs pas bon ménage avec le ZX-81, puisque celui-ci monopolise certains registres pour les besoins de l'affichage TV.

Après avoir lui-même découvert le langage machine à partir du niveau zéro, l'auteur a pu acquérir la conviction que le débutant peut déjà obtenir des résultats honorables en se limitant à l'emploi d'une trentaine d'instructions regroupées dans le tableau de la *figure 6-1*. A l'inverse de la table complète, ce tableau est présenté dans l'ordre alphabétique et ne mentionne que les *codes* opératoires, sans se soucier des octets qui peuvent les suivre. La forme développée de l'instruction devra donc être cherchée dans la table complète, en regard du code correspondant.

Par contre, la *figure 6-1* donne une équivalence BASIC pour chaque instruction machine citée. Il ne s'agit bien évidemment pas d'une équivalence stricte, qui permettrait un remplacement direct, mais bien d'une analogie destinée à faciliter la compréhension du fonctionnement de cette instruction.

TABLE DES 30 INSTRUCTIONS MACHINE « PREFERENTIELLES »

MNEMONIQUE	CODE	EQUIVALENT BASIC
ADD A,B	128	LET A = A + B
ADD A,N	198	LET A = A + OMS
CALL NN	205	GOSUB (OMS + 256 x OPS)
CALL NZ,NN	196	IF Z=0 THEN GOSUB (OMS+256xOPS)
CALL Z,NN	204	IF Z=1 THEN GOSUB (OMS+256xOPS)
CP B	184	IF B=A THEN LET Z=1
CP N	254	IF A=N THEN LET Z=1
DEC A	061	LET A = A – 1
DEC B	005	LET B = B – 1
IN A,(N)	219	spécial (entrée par port)
INC A	060	LET A = A + 1
INC B	004	LET B = B + 1
JP NN	195	GOTO (OMS + 256 x OPS)
JP NZ,NN	194	IF Z=0 THEN GOTO (OMS+256xOPS)
JP Z,NN	202	IF Z=1 THEN GOTO (OMS+256xOPS)
LD A,B	120	LET A = B
LD A,N	062	LET A = OMS
LD A,(NN)	058	LET A = PEEK (OMS+256xOPS)
LD B,A	071	LET B = A
LD B,N	006	LET B = OMS
LD(NN),A	050	POKE (OMS+256xOPS),A
NOP	000	(REM)
OUT(N),A	211	spécial (sortie sur port)
RET	201	RETURN
RET NZ	192	IF Z=0 THEN RETURN
RET Z	200	IF Z=1 THEN RETURN
SBC A,B	152	LET A = A – B – Cy
SBC A,N	222	LET A = A – OMS – Cy
SUB B	144	LET A = A – B
SUB N	214	LET A = A – OMS

Fig. 6-1.

Le lecteur ayant travaillé quelque temps au moyen de ces instructions « préférentielles » ressentira tôt ou tard le besoin de mettre à contribution d'autres instructions.

Il pourra alors se référer utilement à la table complète donnée à la *figure 6-4*, et aux *figures 6-2* et *6-3* qui traduisent les différentes abréviations anglaises entrant dans la composition des mnémoniques. Là encore, des équivalences, nécessairement plus lointaines, sont données par rapport au BASIC.

SIGNIFICATION DES ABREVIATIONS UTILISEES DANS LES MNEMONIQUES :

Les premières lettres des mnémoniques des instructions Z-80 sont l'abréviation de mots anglais décrivant la fonction réalisée. Il suffit d'en connaître la signification ainsi que les conventions de notation des opérandes (registres, cellules mémoire, ports) pour comprendre l'essentiel du fonctionnement de l'instruction. Pour les détails, il faudra bien sûr consulter un recueil complet des instructions Z-80, publié par les fabricants de ce microprocesseur.

ABREVIATION	SIGNIFICATION	EQUIVALENCE BASIC
ADC	addition avec retenue	+
ADD	addition sans retenue	+
AND	ET logique bit à bit	AND
BIT	lecture d'un bit spécifié d'un octet donné	spécial
CALL	appel sous-programme	GOSUB
CP	comparaison	IF-THEN
C	complémentation	spécial
DAA	conversion en BCD	spécial
DEC	décrémentation	LET L = L – 1
DI	annulation interruptions	spécial
DJNZ	décrémenter B et brancher si B = 0	LET B = B – 1 et GOTO
EI	autorisation interruptions	spécial
EX	échanger deux opérandes	double LET
EXX	échange complexe	spécial
HALT	arrêt machine	STOP
IM	choix mode interruption	spécial
IN	entrée par port	spécial (INPUT)
INC	incrémentation	LET L = L + 1
IND	entrée + décrémentations	spécial
INI	entrée + incrémentations	spécial
JP ou JR	branchement	GOTO
LD	chargement	LET
LDD	chargement + décrémentation	spécial
LDI	chargement + incrémentation	spécial
NEG	complémentation à 2	spécial
NOP	pas d'opération	(REM)
OR	OU logique bit à bit	OR
OTD	sortie + décrémentation	spécial
OTI	sortie + incrémentation	spécial
OUT	sortie sur port	spécial
OUTD	sortie + décrémentation	spécial
OUTI	sortie + incrémentation	spécial
POP	récupérer données pile	spécial

(suite au verso)

ABREVIATION	SIGNIFICATION	EQUIVALENCE BASIC
PUSH	entrer données pile	spécial
RES	mettre bit spécifié à 0	spécial (LET)
RET	retour au prog. principal	RETURN
RL	permutation de bits à gauche	spécial
RR	permutation de bits à droite	spécial
RST	repartir à adresse spécifiée	spécial (RUN)
SBC	soustraction avec retenue	–
SCF	forcer la retenue à 1	spécial (LET)
SET	mettre à 1 bit spécifié	spécial (LET)
SLA	décalage de bits à gauche	spécial
SRA	décalage de bits à droite	spécial
SRL	décalage de bits à droite	spécial
SUB	soustraction sans retenue	–
XOR	OU exclusif bit à bit	spécial

Fig. 6-2.

Lors de l'utilisation de ces instructions, le lecteur aura fréquemment à se reporter à différentes tables destinées à faciliter l'assemblage des codes et des adresses. Ces tables ont été publiées plus avant aux *figures 2-2, 2-3, 2-4, 4-1, 4-5 et 5-3.*

C'est là une des originalités de cet ouvrage que de fournir un jeu complet de tables spécialement établies pour le ZX-81, c'est-à-dire utilisant les représentations décimales des octets, compatibles avec les fonctions PEEK et POKE de la machine.

ABREVIATIONS COMPLEMENTAIRES

Ces abréviations sont employées dans la suite des mnémoniques pour en préciser le sens, ou pour désigner les opérandes (registres ou cellules mémoire sur lesquels agit l'instruction).

ABREVIATION	SIGNIFICATION
( )	cellule mémoire dont l'adresse est stockée, sous la forme OMS OPS, dans les deux registres ou les deux nombres mentionnés entre les parenthèses.
A	registre A (accumulateur)
B	registre B
C	registre C
D	registre D
E	registre E
H	registre H

ABREVIATION	SIGNIFICATION
L	registre L
N	valeur numérique (octet) contenu dans l'instruction après le code opératoire.
IX	registre d'index IX (à deux octets)
IY	registre d'index IY (à deux octets)
HL	paire de registres H et L utilisés ensemble
BC	paire de registres B et C utilisés ensemble
DE	paire de registres D et E utilisés ensemble
SP	registre SP (pointeur de pile)
d	déplacement : quantité contenue, sous forme d'un octet, dans une instruction et intervenant, selon des modalités diverses, dans le calcul d'une adresse (à étudier cas par cas).
NN	paire de valeurs numériques (2 octets OMS et OPS)
Z	si drapeau Z à 1 (résultat nul ou égalité)
NZ	si drapeau Z à 0 (résultat =0 ou inégalité)
C	si drapeau C à 1 (retenue)
NC	si drapeau C à 0 (pas de retenue)
M	si drapeau S à 1 (résultat négatif)
P	si drapeau 5 à 0 (résultat positif)
AF	paire de registres A et F
AF'	paire de registres A' et F'
NC	si drapeau C à 0 (pas de retenue)
PO	si drapeau P à 0 (résulat pair ou pas de dépassement)
PE	si drapeau P à 1 (résultat impair ou dépassement)
0	bit N° 0 de l'octet spécifié (registre)
...	.....
7	bit n° 7 de l'octet spécifié (registre)
10H	adresse 16 (après RST)
18H	adresse 24
20H	adresse 32
28H	adresse 40
30H	adresse 48
38H	adresse 56
8	adresse 8
( )	numéro d'un port d'entrée-sortie
'	indique un registre du second groupe

Fig. 6-3.

Les ouvrages consacrés au Z-80 utilisent en effet la notation hexadécimale dans toutes leurs tables, ce qui oblige l'utilisateur de ZX-81 à pratiquer de constantes conversions soit manuellement, soit par programme.

Les plus hardis de nos lecteurs devront cependant passer tôt ou tard à l'hexadécimal, puisque la maîtrise totale du langage machine Z-80 exige la lecture des ouvrages spécialisés.

Cependant, gageons que ces lecteurs passionnés ressentiront alors l'appel de machines beaucoup plus puissantes encore que le ZX-81 ! Nous espérons donc avoir réussi à créer un ouvrage adapté aux particularités et aux limites du ZX-81, qui reste, sans le plus petit doute, un instrument privilégié d'initiation à la micro-informatique, du BASIC jusqu'au langage machine.

Fig. 6-4.

```
→  000          NOP
   001  OMS OPS LD BC,NN
   002          LD (BC),A
   003          INC BC
→  004          INC B
→→ 005          DEC B
→→ 006  OMS     LD B,N
   007          RLCA
   008          EX AF,AF'
   009          ADD HL,BC
   010          LDA,(BC)
   011          DEC BC
   012          INC C
   013          DEC C
   014  OMS     LD C,N
   015          RRCA
   016  OMS     DJNZ,d
   017  OMS OPS LD DE,NN
   018          LD (DE),A
   019          INC DE
   020          INC D
   021          DEC D
   022  OMS     LD D,N
   023          RLA
   024  OMS     JR,d
   025          ADD HL,DE
   026          LDA,(DE)
   027          DEC DE
   028          INC E
   029          DEC E
   030  OMS     LD E,N
   031          RRA
   032  OMS     JRNZ,d
   033  OMS OPS LD HL,NN
   034  OMS OPS LD(NN),HL
   035          INC HL
   036          INC H
   037          DEC H
   038  OMS     LD H,N
   039          DAA
   040  OMS     JRZ,d
   041          ADD HL,HL
   042  OMS OPS LD HL,(NN)
   043          DEC HL
   044          INC L
   045          DEC L
   046  OMS     LD L,N
   047          CPL
   048  OMS     JRNC,d
   049  OMS OPS LD SP,NN
→  050  OMS OPS LD(NN),A
   051          INC SP
   052          INC(HL)
   053          DEC(HL)
   054  OMS     LD(HL),N
   055          SCF
   056  OMS     JRC,d
   057          ADD HL,SP
→  058  OMS OPS LD A,(NN)
   059          DEC SP
→  060          INC A
```

→	**061**		**DEC A**
→	**062**	**OMS**	**LD A,N**
	063		CCF
	064		LD B,B
	065		LD B,C
	066		LD B,D
	067		LD B,E
	068		LD B,H
	069		LD B,L
	070		LD B,(HL)
→	**071**		**LD B,A**
	072		LD C,B
	073		LD C,C
	074		LD C,D
	075		LD C,E
	076		LD C,H
	077		LD C,L
	078		LD C,(HL)
	079		LD C,A
	080		LD D,B
	081		LD D,C
	082		LD D,D
	083		LD D,E
	084		LD D,H
	085		LD D,L
	086		LD D,(HL)
	087		LD D,A
	088		LD E,B
	089		LD E,C
	090		LD E,D
	091		LD E,E
	092		LD E,H
	093		LD E,L
	094		LD E,(HL)
	095		LD E,A
	096		LD H,B
	097		LD H,C
	098		LD H,D
	099		LD H,E
	100		LD H,H
	101		LD H,L
	102		LD H,(HL)
	103		LD H,A
	104		LD L,B
	105		LD L,C
	106		LD L,D
	107		LD L,E
	108		LD L,H
	109		LD L,L
	110		LD L,(HL)
	111		LD L,A
	112		LD(HL),B
	113		LD(HL),C
	114		LD(HL),D
	115		LD(HL),E
	116		LD(HL),H
	117		LD(HL),L
	118		HALT
	119		LD(HL),A
→	**120**		**LD A,B**

	121	LD A,C
	122	LD A,D
	123	LD A,E
	124	LD A,H
	125	LD A,L
	126	LD A,(HL)
	127	LD A,A
→	**128**	**ADD A,B**
	129	ADD A,C
	130	ADD A,D
	131	ADD A,E
	132	ADD A,H
	133	ADD A,L
	134	ADD A,(HL)
	135	ADD A,A
	136	ADC A,B
	137	ADC A,C
	138	ADC A,D
	139	ADC A,E
	140	ADC A,H
	141	ADC A,L
	142	ADC A,(HL)
	143	ADC A,A
→	**144**	**SUB B**
	145	SUB C
	146	SUB D
	147	SUB E
	148	SUB H
	149	SUB L
	150	SUB (HL)

	151	SUB A
→	**152**	**SBC A,B**
	153	SBC A,C
	154	SBC A,D
	155	SBC A,E
	156	SBC A,H
	157	SBC A,L
	158	SBC A,(HL)
	159	SBC A,A
	160	AND B
	161	AND C
	162	AND D
	163	AND E
	164	AND H
	165	AND L
	166	AND(HL)
	167	AND A
	168	XOR B
	169	XOR C
	170	XOR D
	171	XOR E
	172	XOR H
	173	XOR L
	174	XOR(HL)
	175	XOR A
	176	OR B
	177	OR C
	178	OR D
	179	OR E
	180	OR H

	181		OR L
	182		OR(HL)
	183		OR A
→	**184**		**CP B**
	185		CP C
	186		CP D
	187		CP E
	188		CP H
	189		CP L
	190		CP(HL)
	191		CP A
→	**192**		**RET NZ**
	193		POP BC
→	**194**	**OMS OPS**	**JP NZ,NN**
→	**195**	**OMS OPS**	**JP NN**
→	**196**	**OMS OPS**	**CALL NZ,NN**
	197		PUSH BC
→	**198**	**OMS**	**ADD A,N**
	199		RST 0
→	**200**		**RET Z**
→	**201**		**RET**
→	**202**	**OMS OPS**	**JP Z,NN**
	203	utilisé comme préfixe (voir plus loin)	
→	**204**	**OMS OPS**	**CALL Z,NN**
→	**205**	**OMS OPS**	**CALL NN**
	206	OMS	ADC A,N
	207		RST 8
	208		RET NC
	209		POP DE
	210	OMS OPS	JP NC,NN

→	**211**	**OMS**	**OUT(N),A**
	212	OMS OPS	CALL NC,NN
	213		PUSH DE
→	**214**	**OMS**	**SUB N**
	215		RST 10H
	216		RET C
	217		EXX
	218	OMS OPS	JPC,NN
→	**219**	**OMS**	**IN A,(N)**
	220	OMS OPS	CALL C,NN
	221	utilisé comme préfixe (voir plus loin)	
→	**222**	**OMS**	**SBC A,N**
	223		RST 18H
	224		RET PO
	225		POP HL
	226	OMS OPS	JP PO,NN
	227		EX(SP),HL
	228	OMS OPS	CALL PO,NN
	229		PUSH HL
	230	OMS	AND N
	231		RST 20H
	232		RET PE
	233		JP(HL)
	234	OMS OPS	JP PE,NN
	235		EX DE,HL
	236	OMS OPS	CALL PE,NN
	237	utilisé comme préfixe (voir plus loin)	
	238	OMS	XOR N
	239		RST 28H
	240		RET P

	241		POP AF
	242	OMS OPS	JP P,NN
	243		DI
	244	OMS OPS	CALL P,NN
	245		PUSH AF
	246	OMS	OR N
	247		RST 30H
	248		RET M
	249		LD SP,HL
	250	OMS OPS	JP M,NN
	251		EI
	252	OMS OPS	CALL M,NN
	253	utilisé comme préfixe (voir plus loin)	
→	**254**	**OMS**	**CP N**
	255		RST 38H
203	000		RLC B
	001		RLC C
	002		RLC D
	003		RLC E
	004		RLC H
	005		RLC L
	006		RLC(HL)
	007		RLC A
203	008		RRC B
	009		RRC C
	010		RRC D
	011		RRC E
	012		RRC H
	013		RRC L
	014		RRC(HL)
	015		RRC A
203	016		RL B
	017		RL C
	018		RL D
	019		RL E
	020		RL H
	021		RL L
	022		RL(HL)
	023		RL A
203	024		RR B
	025		RR C
	026		RR D
	027		RR E
	028		RR H
	029		RR L
	030		RR(HL)
	031		RR A
203	032		SLA B
	033		SLA C
	034		SLA D
	035		SLA E
	036		SLA H
	037		SLA L
	038		SLA(HL)
	039		SLA A
203	040		SRA B
	041		SRA C
	042		SRA D
	043		SRA E
	044		SRA H
	045		SRA L
	046		SRA(HL)
	047		SRA A

203	056	SRL B
	057	SRL C
	058	SRL D
	059	SRL E
	060	SRL H
	061	SRL L
	062	SRL(HL)
	063	SRL A
203	064	BIT 0,B
	065	BIT 0,C
	066	BIT 0,D
	067	BIT 0,E
	068	BIT 0,H
	069	BIT 0,L
	070	BIT 0,(HL)
	071	BIT 0,A
203	072	BIT 1,B
	073	BIT 1,C
	074	BIT 1,D
	075	BIT 1,E
	076	BIT 1,H
	077	BIT 1,L
	078	BIT 1,(HL)
	079	BIT 1,A
203	080	BIT 2,B
	081	BIT 2,C
	082	BIT 2,D
	083	BIT 2,E
	084	BIT 2,H
	085	BIT 2,L
	086	BIT 2,(HL)
	087	BIT 2,A
203	088	BIT 3,B
	089	BIT 3,C
	090	BIT 3,D
	091	BIT 3,E
	092	BIT 3,H
	093	BIT 3,L
	094	BIT 3,(HL)
	095	BIT 3,A
203	096	BIT 4,B
	097	BIT 4,C
	098	BIT 4,D
	099	BIT 4,E
	100	BIT 4,H
	101	BIT 4,L
	102	BIT 4,(HL)
	103	BIT 4,A
203	104	BIT 5,B
	105	BIT 5,C
	106	BIT 5,D
	107	BIT 5,E
	108	BIT 5,H
	109	BIT 5,L
	110	BIT 5,(HL)
	111	BIT 5,A
203	112	BIT 6,B
	113	BIT 6,C
	114	BIT 6,D
	115	BIT 6,E
	116	BIT 6,H
	117	BIT 6,L
	118	BIT 6,(HL)
	119	BIT 6,A

203	120	BIT 7,B
	121	BIT 7,C
	122	BIT 7,D
	123	BIT 7,E
	124	BIT 7,H
	125	BIT 7,L
	126	BIT 7,(HL)
	127	BIT 7,A
203	128	RES 0,B
	129	RES 0,C
	130	RES 0,D
	131	RES 0,E
	132	RES 0,H
	133	RES 0,L
	134	RES 0,(HL)
	135	RES 0,A
203	136	RES 1,B
	137	RES 1,C
	138	RES 1,D
	139	RES 1,E
	140	RES 1,H
	141	RES 1,L
	142	RES 1,(HL)
	143	RES 1,A

203	144	RES 2,B
	145	RES 2,C
	146	RES 2,D
	147	RES 2,E
	148	RES 2,H
	149	RES 2,L
	150	RES 2,(HL)
	151	RES 2,A
203	152	RES 3,B
	153	RES 3,C
	154	RES 3,D
	155	RES 3,E
	156	RES 3,H
	157	RES 3,L
	158	RES 3,(HL)
	159	RES 3,A
203	160	RES 4,B
	161	RES 4,C
	162	RES 4,D
	163	RES 4,E
	164	RES 4,H
	165	RES 4,L
	166	RES 4,(HL)
	167	RES 4,A
203	168	RES 5,B
	169	RES 5,C
	170	RES 5,D
	171	RES 5,E
	172	RES 5,H
	173	RES 5,L
	174	RES 5,(HL)
	175	RES 5,A
203	176	RES 6,B
	177	RES 6,C
	178	RES 6,D
	179	RES 6,E
	180	RES 6,H
	181	RES 6,L
	182	RES 6,(HL)
	183	RES 6,A
203	184	RES 7,B
	185	RES 7,C
	186	RES 7,D
	187	RES 7,E
	188	RES 7,H
	189	RES 7,L
	190	RES 7,(HL)
	191	RES 7,A
203	192	SET 0,B
	193	SET 0,C
	194	SET 0,D
	195	SET 0,E
	196	SET 0,H
	197	SET 0,L
	198	SET 0,(HL)
	199	SET 0,A
203	200	SET 1,B
	201	SET 1,C
	202	SET 1,D
	203	SET 1,E

	204	SET 1,H
	205	SET 1,L
	206	SET 1,(HL)
	207	SET 1,A
203	208	SET 2,B
	209	SET 2,C
	210	SET 2,D
	211	SET 2,E
	212	SET 2,H
	213	SET 2,L
	214	SET 2,(HL)
	215	SET 2,A
203	216	SET 3,B
	217	SET 3,C
	218	SET 3,D
	219	SET 3,E
	220	SET 3,H
	221	SET 3,L
	222	SET 3,(HL)
	223	SET 3,A
203	224	SET 4,B
	225	SET 4,C
	226	SET 4,D
	227	SET 4,E
	228	SET 4,H
	229	SET 4,L
	230	SET 4,(HL)
	231	SET 4,A
203	232	SET 5,B
	233	SET 5,C
	234	SET 5,D
	235	SET 5,E
	236	SET 5,H
	237	SET 5,L
	238	SET 5,(HL)
	239	SET 5,A
203	240	SET 6,B
	241	SET 6,C
	242	SET 6,D
	243	SET 6,E
	244	SET 6,H
	245	SET 6,L
	246	SET 6,(HL)
	247	SET 6,A
203	248	SET 7,B
	249	SET 7,C
	250	SET 7,D
	251	SET 7,E
	252	SET 7,H
	253	SET 7,L
	254	SET 7,(HL)
	255	SET 7,A

221	009		ADD IX,BC
221	025		ADD IX',DE
221	033	OMS OPS	LD IX,NN
221	034	OMS OPS	LD(NN),IX
221	035		INC IX
221	041		ADD IX,IX
221	042	OMS OPS	LD IX,(NN)
221	043		DEC IX
221	052	OMS	INC (IX+d)
221	053	OMS	DEC (IX+d)
221	054	OMS OPS	LD (IX+d),N
221	057	OMS	ADD IX,SP
221	070	OMS	LDB,(IX+d)
221	078	OMS	LDC,(IX+d)
221	086	OMS	LDD,(IX+d)
221	094	OMS	LDE,(IX+d)
221	102	OMS	LDH,(IX+d)
221	110	OMS	LDL,(IX+d)
221	112	OMS	LD(IX+d),B
221	113	OMS	LD(IX+d),C
221	114	OMS	LD(IX+d),D
221	115	OMS	LD(IX+d),E
221	116	OMS	LD(IX+d),H
221	117	OMS	LD(IX+d),L
221	119	OMS	LD(IX+d),A
221	126	OMS	LDA,(IX+d)
221	134	OMS	ADD A,(IX+d)
221	142	OMS	ADC A,(IX+d)
221	150	OMS	SUB(IX+d)
221	158	OMS	SBC A,(IX+d)
221	166	OMS	AND(IX+d)
221	174	OMS	XOR(IX+d)

221	182	OMS	OR(IX+d)
221	190	OMS	CP(IX+d)
221	225		POP IX
221	227		EX(SP),IX
221	229		PUSH IX
221	233		JP(IX)
221	249		LD SP,IX
221	203	OMS 006	RLC(IX+d)
221	203	OMS 014	RRC(IX+d)
221	203	OMS 022	RL(IX+d)
221	203	OMS 030	RR(IX+d)
221	203	OMS 038	SLA(IX+d)
221	203	OMS 046	SRA(IX+d)
221	203	OMS 062	SRL(IX+d)
221	203	OMS 070	BIT 0,(IX+d)
221	203	OMS 078	BIT 1,(IX+d)
221	203	OMS 086	BIT 2,(IX+d)
221	203	OMS 094	BIT 3,(IX+d)
221	203	OMS 102	BIT 4,(IX+d)
221	203	OMS 110	BIT 5,(IX+d)
221	203	OMS 118	BIT 6,(IX+d)
221	203	OMS 126	BIT 7,(IX+d)
221	203	OMS 134	RES 0,(IX+d)
221	203	OMS 142	RES 1,(IX+d)
221	203	OMS 150	RES 2,(IX+d)
221	203	OMS 158	RES 3,(IX+d)
221	203	OMS 166	RES 4,(IX+d)
221	203	OMS 174	RES 5,(IX+d)
221	203	OMS 182	RES 6,(IX+d)
221	203	OMS 190	RES 7,(IX+d)
221	203	OMS 198	SET 0,(IX+d)
221	203	OMS 206	SET 1,(IX+d)

221	203	OMS 214	SET 2,(IX+d)
221	203	OMS 222	SET 3,(IX+d)
221	203	OMS 230	SET 4,(IX+d)
221	203	OMS 238	SET 5,(IX+d)
221	203	OMS 246	SET 6,(IX+d)
221	203	OMS 254	SET 7,(IX+d)
237	064		IN B,(C)
237	065		OUT(C),B
237	066		SBC HL,BC
237	067	OMS OPS	LD(NN),BC
237	068		NEG
237	069		RET N
237	070		IM0
237	071		LD I,A
237	072		INC(C)
237	073		OUT(C),C
237	074		ADD HL,BC
237	075	OMS OPS	LD BC,(NN)
237	077		RET I
237	080		IN D,(C)
237	081		OUT(C),D
237	082		SBC HL,DE
237	083	OMS OPS	LD(NN),DE
237	086		IM 1
237	087		LD A,I
237	088		IN E,(C)
237	089		OUT(C),E
237	090		ADC HL,DE
237	091	OMS OPS	LD DE,(NN)
237	094		IM 2
237	096		IN H,(C)
237	097		OUT(C),H
237	098		SBC HL,HL
237	103		RRD
237	104		IN L,(C)
237	105		OUT(C),L
237	106		ADC HL,HL
237	111		RLD
237	114		SBC HL,SP
237	115	OMS OPS	LD(NN),SP
237	120		IN A,(C)
237	121		OUT(C),A
237	122		ADC HL,SP
237	123		LD SP,(NN)
237	160		LDI
237	161		CPI
237	162		INI
237	163		OUTI
237	168		LDD
237	169		CPD
237	170		IND
237	171		OUTD
237	176		LDIR
237	177		CPIR
237	178		INIR
237	179		OTIR
237	184		LDDR
237	185		CPDR
237	186		INDR
237	187		OTDR
253	009		ADD IY,BC
253	025		ADD IY,DE
253	033	OMS OPS	LD IY,NN
253	034	OMS OPS	LD(NN),IY

253	035		INC IY
253	041		ADD IY,IY
253	042	OMS OPS	LD IY,(NN)
253	043		DEC IY
253	052	OMS	INC (IY+d)
253	053	OMS	DEC (IY+d)
253	054	OMS OPS	LD(IY+d),N
253	057		ADD IY,SP
253	070	OMS	LDB,(IY+d)
253	078	OMS	LDC,(IY+d)
253	086	OMS	LDD,(IY+d)
253	094	OMS	LDE,(IY+d)
253	102	OMS	LDH,(IY+d)
253	110	OMS	LDL,(IY+d)
253	112	OMS	LD (IY+d),B
253	113	OMS	LD (IY+d),C
253	114	OMS	LD (IY+d),D
253	115	OMS	LD (IY+d),E
253	116	OMS	LD (IY+d),H
253	117	OMS	LD (IY+d),L
253	119	OMS	LD (IY+d),A
253	126	OMS	LD A,(IY+d)
253	134	OMS	ADD A,(IY+d)
253	142	OMS	ADC A,(IY+d)
253	150	OMS	SUB (IY+d)
253	158	OMS	SBC A,(IY+d)
253	166	OMS	AND (IY+d)
253	174	OMS	XOR (IY+d)
253	182	OMS	OR (IY+d)
253	190	OMS	CP (IY+d)
253	225		POP IY
253	227		EX(SP),IY
253	229		PUSH IY
253	233		JP(IY)
253	249		LD SP,IY

253	203	OMS 006	RLC(IY+d)
253	203	OMS 014	RRC(IY+d)
253	203	OMS 022	RL(IY+d)
253	203	OMS 030	RR(IY+d)
253	203	OMS 038	SLA(IY+d)
253	203	OMS 046	SRA(IY+d)
253	203	OMS 062	SRL(IY+d)
253	203	OMS 070	BIT 0,(IY+d)
253	203	OMS 078	BIT 1,(IY+d)
253	203	OMS 086	BIT 2,(IY+d)
253	203	OMS 094	BIT 3,(IY+d)
253	203	OMS 102	BIT 4,(IY+d)
253	203	OMS 110	BIT 5,(IY+d)
253	203	OMS 118	BIT 6,(IY+d)
253	203	OMS 126	BIT 7,(IY+d)
253	203	OMS 134	RES 0,(IY+d)
253	203	OMS 142	RES 1,(IY+d)
253	203	OMS 150	RES 2,(IY+d)
253	203	OMS 158	RES 3,(IY+d)
253	203	OMS 166	RES 4,(IY+d)
253	203	OMS 174	RES 5,(IY+d)
253	203	OMS 182	RES 6,(IY+d)
253	203	OMS 190	RES 7,(IY+d)
253	203	OMS 198	SET 0,(IY+d)
253	203	OMS 206	SET 1,(IY+d)
253	203	OMS 214	SET 2,(IY+d)
253	203	OMS 222	SET 3,(IY+d)
253	203	OMS 230	SET 4,(IY+d)
253	203	OMS 238	SET 5,(IY+d)
253	203	OMS 246	SET 6,(IY+d)
253	203	OMS 254	SET 7,(IY+d)

Collection Poche-Informatique **G. ISABEL**

Un livre de 128 pages,
format 11,7 × 16,5 cm

50 PROGRAMMES POUR ZX 81

Utiles ou divertissants, les programmes qui sont rassemblés dans cet ouvrage sont originaux et utilisent au mieux toutes les fonctions du ZX 81. Ils sont tous écrits pour la version de base avec **mémoire RAM de 1 K.** Loin d'être limités, ils constituent au contraire un **exercice très profitable** pour apprendre à ne pas dépasser la place mémoire disponible.
Votre propre imagination et quelques idées glanées dans ces lignes vous permettront de créer très rapidement des programmes personnels.

Quelques programmes :

- **Aide à la programmation**
- **Conversion de température**
- **Conversion décimal en binaire**
- **Conversion binaire en décimal**
- **Calcul des intérêts**
- **Conjugaison**
- **Loto**
- **Billard**
- **Le champ de mines**

Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue - 75940 PARIS CEDEX 19

Achevé d'imprimer
sur les presses
de l'Imprimerie Marcel Bon
70001 Vesoul
Dépôt légal Octobre 1983
N° éditeur : 399
N° imprimeur : 2714

MAITRISEZ VOTRE ZX 81

Après vous avoir fait partager son apprentissage du Basic dans « Pilotez votre ZX 81 », Patrick Gueulle vous propose de découvrir la **programmation 16 K** et la **programmation en langage machine.**
L'assembleur Z 80 permet, grâce aux fonctions **PEEK, POKE** et **USR,** d'écrire des programmes extrêmement rapides, très peu encombrants et ouvrant la porte à des fonctions nouvelles telles que **les interfaces** auxquelles un chapitre entier est consacré.

Principaux chapitres :

- Programmation avec le module 16 K
- Explorez la mémoire de la machine
- Les interfaces : introduction au langage machine
- Initiation au langage machine du ZX 81
- Du Basic au langage machine
- Jeu d'instructions machine du ZX 81.

ISBN 2-85535-066-2

TARIF CODE N

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/