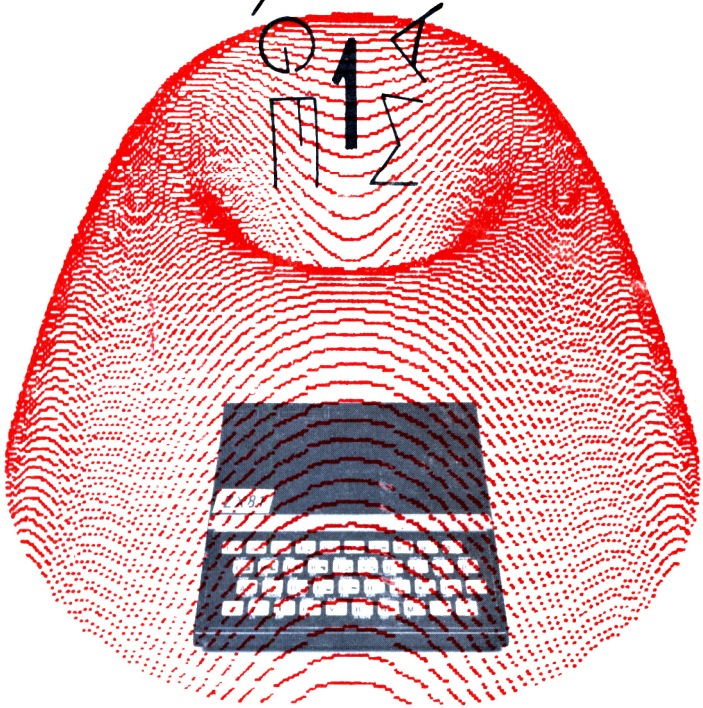
LANGAGE MACHINE 8086



André CHENIÈRE

ÉDITIONS

LANGAGE MACHINE
ZX 81

INTRODUCTION

Alors que l'apprentissage du Basic se fait naturellement, l'aspect rébarbatif et quelque peu ésotérique du langage machine décourage souvent l'amateur.

Le plus difficile est d'en aborder l'étude, de faire les premiers pas.

Le but de ce livre est de vous y aider.

Nous supposerons acquise une bonne connaissance du Basic pour deux raisons :

- la conception même du ZX 81 le rend assez dépendant du Basic;
- les principes élémentaires de programmation sont applicables au langage machine.

Le plus souvent possible nous soulignerons les similitudes entre les deux langages.

Il est souhaitable de disposer au moins de l'extension mémoire 16 K. Si vos programmes atteignent une certaine taille, un assembleur sera indispensable et il faudra lui trouver de la place en mémoire (5 à 6 K).

Notre approche sera très progressive. Ne cédez pas à la tentation de sauter les premiers chapitres. Certaines notions de base doivent être parfaitement assimilées notamment celles concernant la représentation de l'information. Il est nécessaire de connaître la structure du microprocesseur Z 80 A.

Un programme Basic pour l'entrée du code machine vous est proposé au chapitre IV.

Vous vous familiariserez progressivement avec les instructions du microprocesseur en les utilisant. Des exemples, extraits du programme moniteur en mémoire morte, vous aideront à mieux connaître votre ZX 81.

Vous apprendrez à tirer parti de quelques sous-routines du système en ROM.

Nous verrons où et comment implanter le code machine en mémoire et les précautions à prendre.

Vous trouverez en annexes le jeu d'instructions du Z 80 A et d'autres informations utiles.

Ce modeste livre ne prétend pas faire de vous un expert en langage machine. Il faut des années de pratique pour atteindre ce niveau.

Si cet ouvrage vous aide à démarrer dans cette technique et vous encourage à persévérer, l'objectif sera atteint.

En tous cas, sûrement, vous connaîtrez mieux votre ZX 81 et vous tirerez des satisfactions renouvelées de cette remarquable petite machine.

TABLE DES MATIÈRES

CHAPITRE I - BASIC ET LANGAGE MACHINE 5
Langage machine - Inconvénients - Avantages - Différences fondamentales avec le Basic - La perte de contrôle.

CHAPITRE II - REPRÉSENTATION DE L'INFORMATION 9
Le code binaire - Bits et octets - Représentation des nombres, des adresses, des caractères alphanumériques - Complément à deux - Nombres signés - Le code hexadécimal - Le langage d'assemblage.

CHAPITRE III - LE SYSTÈME SINCLAIR ZX 81 19
Les liaisons - L'horloge - Le microprocesseur - Les registres utilisables Organisation de la mémoire.

CHAPITRE IV - PROGRAMME DE CHARGEMENT DU CODE MACHINE ... 27

CHAPITRE V - LE JEU D'INSTRUCTIONS DU Z 80 A 37

CHAPITRE VI - CHARGEMENT DES REGISTRES - TRANSFERTS ET ECHANGES ENTRE REGISTRES - UTILISATION DE LA PILE 45

CHAPITRE VII - TRANSFERTS ENTRE REGISTRES ET MÉMOIRE LES MODES D'ADRESSAGE 53

CHAPITRE VIII - INSTRUCTIONS ARITHMÉTIQUES - COMPARAISONS LES INDICATEURS D'ÉTAT..... 65

CHAPITRE IX - INSTRUCTIONS LOGIQUES 81

CHAPITRE X - INSTRUCTIONS DE BRANCHEMENT 89
Boucles et sous-programmes.

CHAPITRE XI - INSTRUCTIONS SUR BLOCS 109

CHAPITRE XII - ROTATIONS - DÉCALAGES - OPÉRATIONS SUR BITS 119

CHAPITRE XIII - ENTRÉES, SORTIES, INTERRUPTIONS
LE FICHIER D'AFFICHAGE 131

CHAPITRE XIV - PROGRAMME MONITEUR - ROUTINES UTILISABLES ... 141

CHAPITRE XV - IMPLANTATION DES PROGRAMMES 159

ANNEXES :

1. - LE JEU D'INSTRUCTIONS DU Z 80
   - CODAGE 169
   - DÉCODAGE 174

2. - CONVERSION DÉCIMAL - HEXADÉCIMAL 180

3. - TABLE DES VALEURS DES TOUCHES 182

4. - EFFET DES INSTRUCTIONS SUR LES INDICATEURS D'ÉTAT 183

5. - LES VARIABLES SYSTÈME 184

CHAPITRE I

BASIC ET LANGAGE MACHINE.

Quand vous communiquez en Basic avec l'ordinateur, par programme ou en commande directe, vous le faites par l'intermédiaire d'un interprète, d'un traducteur. Ce traducteur est l'interpréteur Basic.

Chaque ligne de programme, chaque commande, doit être examinée en détail par l'interpréteur et traduite dans un langage que la machine peut comprendre; un langage dit de bas niveau, le binaire.

En page 162 de votre manuel SINCLAIR , vous avez une photo de l'intérieur de votre système. Examinez le circuit intégré marqué NEC. C'est l'unité centrale, le microprocesseur Z 80 A. Manifestement cette puce ne peut réagir qu'à des impulsions électriques. Ces signaux lui arrivent simultanément par huit de ses pattes. Leur combinaison forme le code machine, le code binaire.

Par convention, on note O l'absence de courant électrique (O.V.) et 1 la présence de courant (5 V.).

Un chiffre binaire, que ce soit le zéro ou le un est appelé un BIT (contraction de Binary digit). C'est l'unité fondamentale d'information. Le Z 80 A, microprocesseur 8 bits, opère sur huit bits en même temps (sur un octet).

Une instruction ou une donnée pour la machine est de la forme : 11001001.

Un programme en code machine stocké en mémoire est composé d'une série de groupes de 8 bits ou octets de cette forme. Tel est aussi bien sûr, en mémoire inaltérable (ROM) le programme interpréteur Basic.

Avec une instruction en binaire comme 00111010 10000010 01000000, on est bien loin des expressions familières du Basic et pourtant tel est le langage machine, comme son nom l'indique, un langage pour machines.

Puisque nous avons entrepris de le démythifier, voyons ce que nous apporte son utilisation.

LANGAGE MACHINE - INCONVÉNIENTS.

• INSTRUCTIONS NÉCESSAIRES PLUS NOMBREUSES.

Dans un langage évolué comme le Basic les instruction sont globales, synthétiques. C'est l'interpréteur qui les traduit en pas élémentaires pour le microprocesseur.

En langage machine ce travail long et méticuleux est à la charge du programmeur.

• PROGRAMMES DIFFICILES A LIRE ET A METTRE AU POINT.

Le code binaire est difficilement lisible. Nous verrons qu'il existe des équivalents plus proches du langage humain destinés à faciliter la programmation. Malgré cela le langage machine reste complexe parce que très détaillé. Il exige effort d'adaptation et pratique.

• DÉFAUT DE PORTABILITÉ DES PROGRAMMES.

Les langages évolués se veulent universels, mais le langage machine est spécifique de la machine utilisée.

• CALCULS DÉLICATS A PROGRAMMER.

Un langage évolué convient mieux à des applications financières, commerciales ou scientifiques. La manipulation des nombres en virgule flottante ne peut guère être envisagée par un programmeur moyen.

Le langage machine, heureusement n'apporte pas que des inconvénients. Il permet l'utilisation optimale des ressources internes de l'ordinateur et offre des avantages souvent décisifs.

AVANTAGES.

• VITESSE D'EXÉCUTION.

Le fait de passer par l'interpréteur Basic pour communiquer avec la machine est très pénalisant. Entre un programme écrit en code machine et son équivalent en Basic il peut y avoir une différence considérable dans le temps d'exécution (de 1 à 5 au minimum, le rapport peut aller jusqu'à 1 à 100).

C'est un critère déterminant dans de nombreuses applications, notamment pour certains jeux où le Basic s'avère décidément trop lent.

• ÉCONOMIE D'ESPACE MÉMOIRE.

Les programmes écrits en langage machine occupent moins de place que leurs équivalents en Basic. Le programmeur peut tirer meilleur parti de l'espace disponible.

• LIBERTÉ PAR RAPPORT AU SYSTÈME D'EXPLOITATION.

L'utilisateur est affranchi des limites imposées par le programme moniteur. Certaines applications difficiles ou même impossibles à réaliser en Basic peuvent être programmées en langage machine.

• INTÉGRATION FACILE DANS DES PROGRAMMES EN BASIC.

Il est rarement indiqué de composer des programmes exclusivement en code machine. Il peut être réservé pour des sections critiques (où le Basic serait trop lent par exemple). L'appel de ces sous-programmes avec USR et le retour au Basic ne posent pas de problèmes.

DIFFÉRENCES FONDAMENTALES AVEC LE BASIC.

- **PAS DE NUMÉRO DE LIGNE.**

Dans un programme Basic les instructions sont précédées d'un numéro de ligne qui permet à l'ordinateur de savoir dans quel ordre les exécuter.

En langage machine les instructions sont exécutées dans l'ordre croissant de leur position en mémoire.

- **PAS DE COMMANDE DIRECTE.**

Une instruction doit faire partie intégrante d'un programme.

- **PAS DE VARIABLES.**

Le concept de variables tel qu'il existe en Basic ne s'applique pas en langage machine. Les registres internes nommément désignés, que l'on «charge» avec une valeur, ne sont pas des variables au sens «Basic» du terme malgré une certaine analogie.

- **PAS D'ARRÊT AUTOMATIQUE EN FIN DE PROGRAMME.**

Le microprocesseur essaie d'interpréter le contenu de l'adresse suivante. S'il n'y arrive pas, c'est le crash.

LA PERTE DE CONTRÔLE.

Elle se produit lorsqu'une erreur de codage s'est glissée dans le programme ou lorsqu'une instruction de retour au Basic a été omise.

En langage machine, vous n'avez pas de garde-fou, pas de message d'erreur, pas de vérification de la syntaxe. La plus petite erreur peut être fatale.

Vous apprendrez vite à reconnaître les symptômes d'un crash. L'écran devient gris. Vous essayez désespérément de recouvrer le contrôle avec BREAK, STOP, NEW-LINE, etc. et vous découvrez avec horreur que toutes les touches sont inopérantes.

Il ne vous reste plus qu'à débrancher l'ordinateur, à le rebrancher et à recommencer.

Evidemment tout ce qui était en mémoire, Basic, code machine (représentant peut-être de nombreuses heures de travail) est perdu.

Conclusion : si vous procédez à la mise au point d'un programme important en code machine utilisez la cassette ou l'imprimante pour le sauvegarder **avant** de le faire tourner.

CHAPITRE II

REPRÉSENTATION DE L'INFORMATION

LE CODE BINAIRE.

BITS ET OCTETS.

Pour les circuits électroniques de la machine, l'information est matérialisée par l'absence ou la présence de courant, un niveau de tension égal à 0 V. ou à 5 V.. Le BIT unité élémentaire d'information représente l'un de ces deux états possibles, notés 0 ou 1.

Le Z 80 A opère sur 8 bits en même temps. Un groupe de 8 bits est appelé un octet.

Lors de l'exécution d'un programme le microprocesseur décode les octets un par un. Par suite le contenu des cases de la mémoire centrale, ROM ou RAM, chacune dotée d'une adresse différente, est de 8 bits ou un octet.

Le programme est formé d'une suite d'octets commençant à une adresse bien définie.

Une instruction machine du Z 80 A occupera de 1 à 4 octets en mémoire.

Exemple : charger l'accumulateur avec le contenu de l'adresse mémoire 16514 d s'écrit en binaire sur 3 octets :

00111010 10000010 01000000

Le premier octet d'une instruction est toujours un code opératoire. Ce code est différent suivant la nature de l'opération à effectuer et de l'information à traiter. Nous verrons qu'un code opératoire peut occuper jusqu'à 3 octets et qu'il en existe près de 700 dans le jeu d'instructions du Z 80 A.

00111010 signifie : charger l'accumulateur avec le contenu de l'emplacement mémoire dont l'adresse suit sur 2 octets (adressage absolu).

Autre exemple : charger l'accumulateur avec la valeur décimale 10 s'écrit sur 2 octets :

00111110 00001010

Le code opératoire est différent du précédent.

00111110 signifie : charger l'accumulateur avec la valeur représentée par l'octet qui suit (adressage immédiat).

Certaines instructions n'ont qu'un seul octet, le code opératoire.

Exemple : charger l'accumulateur avec le contenu du registre B s'écrit sur 1 octet.

01111000 (adressage implicite)

Nous reviendrons en détail sur la contexture des instructions machine du Z 80 et sur les modes d'adressage. Pour le moment retenez que c'est le code opératoire qui définit l'action spécifique à effectuer. En le décodant le microprocesseur «sait»

- la nature de l'opération à effectuer;
- le nombre d'octets de l'instruction complète;
- comment interpréter l'information qui y est incluse.

Outre l'instruction proprement dite ou code opératoire, des informations de nature très différente peuvent être codées sur un ou plusieurs octets consécutifs : nombre, adresse, caractère alphanumérique, chaîne de caractères...

REPRÉSENTATION DES NOMBRES ENTIERS.

Dans tout système de numération, chaque chiffre d'un nombre donné possède un certain «poids» suivant le rang qu'il occupe. Ainsi dans le système décimal (base 10) on a de la droite vers la gauche le chiffre des unités, puis celui des dizaines, celui des centaines, celui des milliers...

Chaque chiffre d'un nombre décimal vaut entre 0 et 9 fois une certaine puissance de 10.

Exemple : 2605 représente

$$5 \times 10^0 + 0 \times 10^1 + 6 \times 10^2 + 2 \times 10^3$$

De la même façon dans le système binaire (base 2) chaque bit (ou chiffre binaire) vaut 0 ou 1 fois une certaine puissance de 2.

Pour faciliter leur repérage, les 8 bits d'un octet sont numérotés de 0 à 7 de la droite vers la gauche.

bit n°	7	6	5	4	3	2	1	0

Chaque bit possède un poids (croissant de la droite vers la gauche) représenté par une puissance de 2.

bit n° 0	$2^0 = 1$
bit n° 1	$2^1 = 2$
bit n° 1	$2^2 = 4$
bit n° 3	$2^3 = 8$
bit n° 4	$2^4 = 16$
bit n° 5	$2^5 = 32$
bit n° 6	$2^6 = 64$
bit n° 7	$2^7 = 128$

Le bit n° 0 est appelé le bit le moins significatif, le bit n° 7 étant le bit le plus significatif.

Exemple : le nombre binaire

	0	1	1	1	0	0	1	1
bit n°	7	6	5	4	3	2	1	0

représente

$$
\begin{aligned}
1 \times \quad 1 &= \quad 1 \ (2^0) \\
1 \times \quad 2 &= \quad 2 \ (2^1) \\
0 \times \quad 4 &= \quad 0 \ (2^2) \\
0 \times \quad 8 &= \quad 0 \ (2^3) \\
1 \times \ 16 &= \ 16 \ (2^4) \\
1 \times \ 32 &= \ 32 \ (2^5) \\
1 \times \ 64 &= \ 64 \ (2^6) \\
0 \times 128 &= \ 0 \ (2^7)
\end{aligned}
$$

$$= 115 \text{ en décimal}$$

01110011 en binaire et 115 en décimal représentent le même nombre.

Dans cet exemple, les bits n° 0, 1, 4, 5, 6 sont à un, en anglais SET, les autres sont à zéro ou RESET. On retrouvera ces termes dans certaines instructions du microprocesseur (opérations sur BITS).

11111111 est le plus grand nombre binaire sur 8 bits. Son équivalent décimal est $1 \times 1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 1 \times 16 + 1 \times 32 + 1 \times 64 + 1 \times 128 = 255$

Le plus petit est 00000000 = zéro.

Avec un octet on peut donc représenter 256 valeurs différentes (de 0 à 255 décimal) ce qui correspond bien à 2^8 combinaisons.

Certains instructions permettent de manipuler des nombres sur 16 bits (valeur décimale de 0 à 65535). Ces nombres sont représentés sur 2 octets.

L'un est appelé octet de poids faible ou octet le moins significatif, LSB (Least significant byte) ou Low Byte (LO).

L'autre est l'octet de poids fort, octet le plus significatif, MSB (Most significant byte) High Byte (HI).

Dans le programme les octets sont inversés; c'est l'octet de poids faible qui suit immédiatement le code opératoire.

Ainsi, charger le registre double HL avec la valeur décimale 30000 s'écrit :

00100001	00110000	01110101
code opératoire	LO	HI

L'octet de poids fort peut être considéré comme un prolongement de l'octet de poids faible formant ainsi un seul nombre binaire sur 16 bits

	0	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0
bit n°	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	octet de poids fort								octet de poids faible							

$16384\ (2^{14}) + 8192\ (2^{13}) + 4096\ (2^{12}) + 1024\ (2^{10}) + 256\ (2^8) + 32\ (2^5) + 16\ (2^4) = 30000$

En prenant les 2 octets séparément :

HI 01110101 = 64 + 32 + 16 + 4 + 1 = 117
LO 00110000 = 32 + 16 = 48

d'où valeur décimale = 117 x 256 + 48 = 30000
HI LO

Nous n'aborderons pas ici la représentation des nombres en virgule flottante. Sachez que les nombres réels sont représentés sur 5 octets par le Basic 8 K du ZX 81. Des sous-routines de la ROM permettent de les manipuler. Ils sont stockés dans une zone en mémoire vive appelée pile du calculateur (calculator stack).

REPRÉSENTATION DES ADRESSES.

Une adresse est **toujours** spécifiée sur 2 octets (16 bits).

Encore une fois c'est le code opératoire de l'instruction qui précise qu'il s'agit d'une adresse et non pas d'une donnée numérique sur 16 bits. Remarquez qu'une adresse peut toujours être manipulée ou modifiée comme un vulgaire nombre tant que le programme ne s'en sert pas pour désigner nommément un emplacement mémoire. Ne vous inquiétez pas, si cela n'est pas évident pour le moment. Ça ira mieux ensuite.

Reprenons le tout premier exemple du chapitre :

Charger l'accumulateur avec le contenu de l'adresse mémoire 16514 (décimal)

00111010	10000010	01000000
code opératoire	octet de poids faible	octet de poids fort

Ici encore on retrouve en premier l'octet de poids faible puis l'octet de poids fort.

REPRÉSENTATION DES CARACTÈRES ALPHANUMÉRIQUES.

Un caractère alphanumérique ou semi-graphique est représenté par un code sur un octet. Vous trouverez en annexe A, page 181 du manuel SINCLAIR, le jeu de caractères du ZX 81

Le code est différent du code standard ASCII habituellement utilisé sur micro-ordinateur.

Bien entendu, chaque valeur décimale a son équivalent binaire et c'est à charge du programme de faire en sorte que le microprocesseur interprète cette valeur binaire comme le code d'un caractère et non pas comme celui d'une valeur numérique quelconque.

Une chaîne de caractères est représentée par la suite des octets qui codent chacun des caractères (lettres, chiffres, ponctuation).

COMPLÉMENT A DEUX - NOMBRES SIGNÉS.

Tous les nombres manipulés par le microprocesseur sont en valeur absolue. Chaque octet a une valeur binaire comprise entre 00000000 et 11111111, 0 et 255 en équivalent décimal.

A priori, rien ne permet de dire que tel ou tel nombre pourrait être considéré comme négatif. Lorsqu'une opération telle que l'addition affecte le contenu d'un octet au-delà de 255, ce contenu revient à zéro. De même à la suite d'une soustraction le contenu de l'octet revient à 255 après avoir passé zéro. Nous verrons qu'un indicateur spécial permet de tenir compte de la retenue ou de l'emprunt généré par l'opération et de corriger le résultat.

L'addition binaire obéit à des règles analogues à celles de l'arithmétique décimale :

0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = ① 0

avec ① représentant une retenue.

10 binaire est équivalent à 2 décimal.

1 + 1 = 10 je pose 0 et je retiens 1.

Exemple :

125	0 1 1 1 1 1 0 1
+ 63	0 0 1 1 1 1 1 1
+ 188	1 0 1 1 1 1 0 0

(retenues : 1 1 1 1 1 1 1)

COMPLÉMENT A DEUX.

Nous admettrons ce qui suit sans démonstration mathématique.

L'opposé ou complément à deux d'un nombre binaire s'obtient en complémentant ce nombre, c'est-à-dire en inversant les bits (1 → 0 et 0 → 1) et en ajoutant 1.

En ajoutant à un nombre son opposé ou doit trouver zéro. Vérifions :

soit le nombre binaire :	00110010
son complément à deux est	
en complémentant	11001101
en ajoutant 1	1
	11001110

Additionnons 00110010 et son complément à deux 11001110

(retenues : 1 1 1 1 1 1)

0 0 1 1 0 0 1 0

1 1 0 0 1 1 1 0

1 0 0 0 0 0 0 0 0

le résultat est bien zéro sur 8 bits. La retenue doit être ignorée.

L'utilisation du complément à deux est largement répandue sur micro-ordinateur. Elle permet une simplification de l'unité arithmétique et logique du microprocesseur.

La soustraction se ramène à une addition. Elle s'effectue en ajoutant le complément à deux du nombre à soustraire.

Exemple :

Soit à effectuer la soustraction suivante :

	01111101	en décimal	125
—	00111111	—	63

Ce complément à deux de 00111111 est 11000001. La soustraction devient :

	01111101		125
+	11000001	+	(-63)
1	00111110		62

On ne tient pas compte de la retenue.

NOMBRES SIGNÉS.

La représentation binaire signée, ou représentation en complément à deux permet d'interpréter une valeur binaire (absolue) et de lui attribuer une valeur décimale positive ou négative qui, sur un octet, sera comprise entre — 128 et + 127.

Certaines instructions machine font cette interprétation, en particulier les instructions de saut relatif.

Par convention le bit n° 7, le plus à gauche, est le bit de signe. 0 indique un nombre positif. 1 indique un nombre négatif.

Les nombres positifs de 0 à + 127 sont ainsi représentés par les valeurs binaires comprises entre 00000000 et 01111111.

Les nombres négatifs de — 128 à — 1 sont représentés par les valeurs binaires comprises entre 10000000 et 11111111 complément à deux des nombres positifs correspondants (sauf 128).

	BINAIRE	DÉCIMAL	HEXA
nombres positifs	01111111	+ 127	7F
	01111110	+ 126	7E
	00000010	+ 2	02
	00000001	+ 1	01
	00000000	0	00
nombres négatifs	11111111	— 1	FF
	11111110	— 2	FE
	10000001	— 127	81
	10000000	— 128	80

Il est bon de le répéter. Un nombre binaire en mémoire compris entre 00000000 et 11111111 peut avoir un équivalent décimal entre 0 et 255 ou entre — 128 et 127, mais pas les deux en même temps. C'est le programmeur qui décide de l'interprétation choisie, c'est-à-dire si le nombre binaire doit être traité en complément à deux ou non.

Un nombre sur 16 bits peut être traité en complément à deux. Le bit de signe est alors le bit n° 7 de l'octet de poids fort.

Nous verrons plus tard en détail le rôle des indicateurs d'état. Il est cependant utile de dire ici quelques mots sur le débordement et son indicateur en relation avec ce que nous avons vu.

Le terme de débordement prête parfois à confusion. Il ne signifie pas un manque de place (cela est signalé par l'indicateur de retenue). Une condition de débordement existe lorsqu'après une opération sur des nombres traités en complément à deux, le bit de signe du résultat est changé accidentellement. L'indicateur V est alors mis à 1.

Voyons cela sur plusieurs exemples.

```
   65          01000001
+  64        + 01000000
  129          10000001
               ↑
               négatif en complément à 2.
```

Le résultat est correct en arithmétique binaire absolue mais incorrect en complément à deux. L'indicateur de débordement V est mis à 1.

```
   20          00010100
+  90        + 01011010
  110          01101110
```

Pas de débordement

L'indicateur est mis à zéro.

en complément à 2		en valeur absolue
− 125	10000011	131
− 111	10010001	145
20	1 00010100	20 + retenue

Débordement. Le résultat est positif par accident. L'indicateur est mis à 1. Le résultat est cependant correct en valeur absolue (il y a une retenue).

LE CODE HEXADÉCIMAL.

Le code binaire est malcommode à utiliser. Une notation équivalente a été universellement adoptée, c'est l'hexadécimal.

Le système hexadécimal est le système de numération à base 16.

Les chiffres hexadécimaux sont 0 1 2 3 4 5 6 7 8 9 A B C D E F.

Chaque chiffre hexadécimal représente un groupe de 4 bits ou quartet (nibble).

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Un octet sera représenté par 2 chiffres hexadécimaux.

Exemples : 10000100 = 84 = 132 d
11001001 = C9 = 201 d
11111010 = FA = 250 d
00001110 = OE = 14 d

Un nombre sur 2 octets, une adresse par exemple, sera représenté par 4 chiffres hexadécimaux.

Exemples :

64 x 256 + 130 = 16514 déci.

0100 0000 1000 0010

4 0 8 2 4082 hexa.

$4 \times 4096 + 0 \times 256 + 8 \times 16 + 2 = 16514$ déci.

$16^3 \quad 16^2 \quad 16^1 \quad 16^0$

117 x 256 + 48 = 30000 déci.

0111 0101 0011 0000

7 5 3 0 7530 hexa.

$7 \times 4096 + 5 \times 256 + 3 \times 16 + 0 = 30000$ déci.

$16^3 \quad 16^2 \quad 16^1 \quad 16^0$

$$\overbrace{2 \times 256}^{0000\ 0010} + \overbrace{187}^{1011\ 1011} = 699 \text{ déci.}$$

$$\underbrace{0}_{0000}\ \underbrace{2}_{0010}\quad \underbrace{B}_{1011}\ \underbrace{B}_{1011} \qquad 02BB \text{ hexa.}$$

$$0 \times 4096 + 2 \times 256 + 11 \times 16 + 11 = 699 \text{ déci.}$$

$$16^3 \quad 16^2 \quad 16^1 \quad 16^0$$

Conversions.

Pour convertir le binaire en hexadécimal et inversement on utilisera le tableau précédent dont on pourra se passer avec un peu de pratique.

Une table de conversion décimal ↔ hexadécimal est donnée en annexe.

Pour convertir le décimal en binaire et inversement, on peut passer par l'hexadécimal.

• • •

Le format hexadécimal est pratique à utiliser pour plusieurs raisons :

1. - La conversion en binaire est facile.
2. - Au premier coup d'œil, on sait s'il s'agit d'un nombre de 8 ou 16 bits (2 ou 4 chiffres hexadécimaux).
3. - Tous les nombres sont standardisés.
4. - C'est une convention universellement adoptée en micro-informatique.

Reprenons le premier exemple présenté au début de ce chapitre.

Charger l'accumulateur avec le contenu de l'adresse mémoire 16514 d.

0011	1010	1000	0010	0100	0000
3	A	8	2	4	0

Notre instruction sur 3 octets devient : 3A 82 40 ce qui est plus concis que sous la forme binaire.

Malgré la grande simplification qu'il apporte il est exclus de programmer directement en hexadécimal. Un programme doit avoir un sens pour un lecteur humain.

C'est pourquoi un autre niveau de langage est utilisé : l'assembleur symbolique ou langage d'assemblage.

LE LANGAGE D'ASSEMBLAGE.

Le langage d'assemblage est une traduction partielle du langage machine qui le rapproche du langage humain.

Le découpage en actions élémentaires est respecté. Chaque instruction a un équivalent symbolique.

Un code mnémonique rappelle sous une forme abrégée la nature de l'opération à effectuer. Ce code est généralement défini par le fabricant du microprocesseur. Nous utiliserons donc les mnémoniques établis par ZILOG Corporation pour son ZX 80 A.

Nous reviendrons en détail sur les codes mnémoniques en étudiant le jeu d'instructions du Z 80 A et ensuite bien sûr nous les utiliserons.

Pour compléter ce chapitre, reprenons nos trois exemples concernant le chargement de l'accumulateur.

Charger se dit LOAD en Anglais d'où le mnémonique LD.

1. - Charger l'accumulateur avec le contenu de l'adresse mémoire 16514 (décimal).

mnémonique	code hexa	binaire
LD A, (16514 d)	3A 82 40	00111010 10000010 01000000

2. - Charger l'accumulateur avec la valeur décimale 10 (0A en hexa)

mnémonique	code hexa	binaire
LD A, 0A h	3E 0A	00111110 00001010

3. - Charger l'accumulateur avec le contenu du registre B.

mnémonique	code hexa	binaire
LD A, B	78	01111000

Le langage d'assemblage est parfois improprement appelé assembleur.

En réalité l'assembleur est un programme qui traduit les instructions en code mnémonique en code binaire directement utilisable par le microprocesseur.

On entre par exemple LD A, OA h et l'assembleur range à l'adresse spécifiée 00111110 00001010.

Le langage d'assemblage est le langage de programmation. Nous l'utiliserons tout au long de ce livre. En l'absence d'assembleur, nous assemblerons nos programmes à la main, c'est-à-dire que nous traduirons à l'aide d'une table le code mnémonique en code hexadécimal.

Un programme utilitaire nous permettra d'entrer le code machine sous cette forme hexadécimale.

CHAPITRE III

LE SYSTÈME SINCLAIR ZX 81

Tout d'abord, lisez ou relisez les chapitres 25 à 28 de votre manuel SINCLAIR. Dans les pages qui suivent nous insisterons surtout sur la structure du microprocesseur et l'organisation de la mémoire du système ZX 81.

L'architecture d'un système micro-ordinateur est organisée autour de son microprocesseur, l'unité centrale, CPU (Central Processing Unit).

Le microprocesseur Z 80 A, fabriqué par la société californienne ZILOG INC., est le cerveau du SINCLAIR. Parmi les microprocesseurs 8 bits actuels, il est l'un des plus répandus. Il se caractérise par un jeu d'instructions très vaste et de nombreux registres internes.

Chargé du traitement de l'information, le microprocesseur a besoin d'instructions et de données pour opérer. Elle lui seront fournies par programme.

Il ne peut pas travailler isolément et doit être relié aux autres éléments du système.

LES LIAISONS.

Elles sont de trois types suivant le genre d'information à acheminer. Les faisceaux de lignes (fils électriques) portent le nom de bus.

• Le bus de données, bidirectionnel, 8 lignes en parallèle, transporte 8 bits à la fois. On dit qu'il est capable d'acheminer un octet de données à la fois. Remarquez que le terme d'octet est habituellement utilisé pour décrire un emplacement susceptible de recevoir une donnée sur 8 bits plutôt que la donnée elle-même.

Le bus de données est physiquement relié au circuit de mémoire morte (ROM), au circuit de mémoire vive (RAM), au clavier et aux interfaces cassette et téléviseur.

• Le bus d'adresses, unidirectionnel, comporte 16 lignes en parallèle. On peut dire qu'il est large de 16 bits ou 2 octets. Il transporte une adresse générée par le microprocesseur. Cette adresse (ROM ou RAM) est un nombre qui spécifie la source ou la destination des données qui transitent par le bus de données.

Le nombre maximum d'octets adressables devrait être $2^{16} = 65536$ mais sur le ZX 81, 15 seulement des 16 bits sont utilisés ce qui donne $2^{15} = 32768$ adresses possibles (de 0 à 32767 inclus).

Le bus d'adresses relie l'unité centrale aux circuits RAM et ROM, au clavier et à la puce logique.

De tout ceci, retenez qu'une adresse doit **toujours** être spécifiée sur 16 bits, 2 octets. Un octet haut (HI) qui comprend les 8 bits les plus significatifs. Un octet bas (LO) qui comprend les 8 bits les moins significatifs.

• Les lignes de commandes (un seul fil) transportent les différents signaux de synchronisation nécessaires au système.

L'HORLOGE.

La base de temps est fournie par une horloge à 3,25 MHz.

Pour être accomplie une instruction demande un certain temps (recherche, décodage, exécution...) exprimé en cycles d'horloge.

Les instructions les plus «rapides» (4 cycles) seront donc exécutées en :

$$\frac{1}{3,25} \times 4 = 1,23 \text{ microsecondes}$$

et les plus longues (23 cycles) en :

$$\frac{1}{3,25} \times 23 = 7,07 \text{ microsecondes}$$

En une seconde, le microprocesseur exécute ainsi entre 141 300 et 812 500 instructions dès qu'il est mis sous tension.

LE MICROPROCESSEUR.

La structure interne du Z 80 est complexe. Elle peut être schématisée en 5 parties fonctionnelles.

- Unité de commande
- Registre instruction
- Compteur ordinal PC
- Unité arithmétique et logique
- Registres utilisables.

L'unité de commande est chargée de diriger le système et le registre instruction est un registre spécial interne destiné à conserver une copie de l'instruction en cours d'exécution.

Le compteur ordinal PC (Program Counter).

C'est un registre 16 bits très important. Il contient soit l'adresse mémoire de l'instruction en cours d'exécution soit l'adresse de la prochaine instruction à chercher en mémoire.

Il permet une recherche en séquence aussi bien des octets successifs d'une instruction que des instructions successives.

Son rôle est essentiel dans les instructions de branchement, sauts ou appels de sous-programmes.

Son action est similaire à celle de la variable système : PPC (N° de la ligne Basic en cours d'exécution) rangée aux adresses 16391 et 16392.

L'unité arithmétique et logique.

Comme son nom l'indique, l'UAL est chargée d'effectuer les opérations arithmétiques et logiques.

Les opérations arithmétiques sont très limitées. Il s'agit des simples addition et soustraction binaires, de l'incrémentation (ajouter un) et de la décrémentation (soustraire un).

La multiplication, la division, le traitement des nombres réels doivent être programmés. Les routines correspondantes existent en ROM et peuvent éventuellement être utilisées.

Les opérations logiques et de nombreuses opérations sur bits sont réalisées par l'UAL.

L'UAL est une unité interne au Z 80 à ne pas confondre avec la puce logique (Ferranti) dont la description dépasse le cadre de ce livre.

LES REGISTRES UTILISABLES.

Ils peuvent recevoir des informations au gré du programmeur.

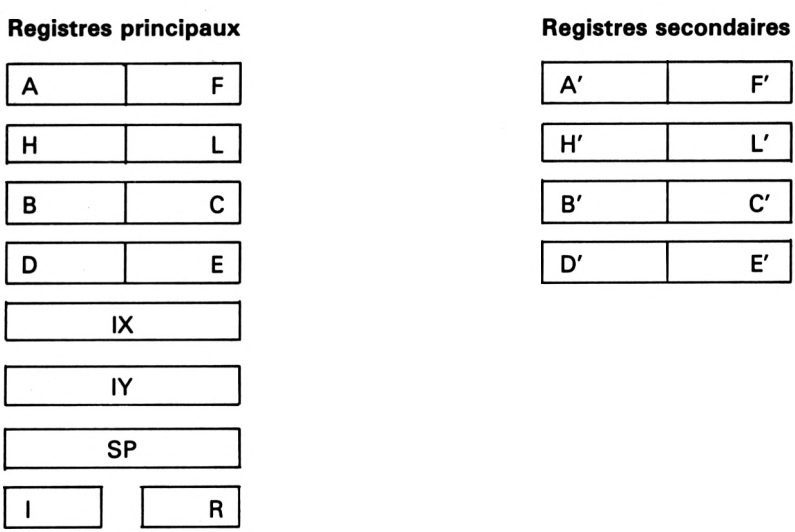
Il convient de définir exactement ce qu'est un registre.

Vous savez que l'information peut être rangée dans les différents octets de la mémoire. Le contenu de ces octets peut être figé (ROM) ou modifiable (RAM).

Les registres internes sont des cases mémoire temporaires de 1 ou 2 octets à l'intérieur du microprocesseur. Ils jouent le rôle de bloc-notes, conservant les informations éphémères à traiter par le microprocesseur.

L'utilisation judicieuse des registres internes est primordiale dans le travail de programmation du Z 80.

Nous allons les passer successivement en revue. Leurs particularités seront précisées, complétées et même répétées au besoin dans la suite du livre.



Le registre A.

C'est le registre simple (8 bits) le plus important du Z 80. Il est souvent appelé accumulateur. La plupart des instructions arithmétiques et logiques opèrent sur le contenu de A. De nombreuses opérations ne peuvent s'effectuer qu'avec ce registre.

Le registre F (Flag Register)

C'est le registre des indicateurs d'état du microprocesseur. Six de ses huit bits sont utilisés pour indiquer un état particulier du Z 80. Le programmeur est réellement concerné par quatre indicateurs principaux, les bits n° 0, 2, 6 et 7 du registre F.

Ils sont appelés respectivement : C (Carry) retenue, P/V (Parity-overflow) parité-débordement, Z (zéro) et S (signe).

Nous reviendrons plus longuement sur le rôle important joué par ces indicateurs.

Le registre F est apparié avec l'accumulateur pour deux instructions concernant la pile : PUSH AF et POP AF.

Les registres H et L - Le registre HL.

Dans les premiers modèles de microprocesseur le registre utilisé pour contenir les adresses était un registre simple octet (8 bits) capable d'adresser 256 emplacements mémoire (2^8).

Quand les registres double octet sont apparus, l'un des deux fut appelé registre haut (High) et l'autre registre bas (Low) d'où le nom des registres H et L.

Ces registres peuvent être utilisés séparément mais ils forment le plus souvent le registre double HL.

HL est surtout utilisé pour pointer une adresse en mémoire. Il est spécialisé pour les opérations arithmétiques sur 16 bits. En cela il a un rôle analogue à celui de l'accumulateur.

Les registres B, C, D et E - BC et DE.

B, C, D et E sont des registres d'usage général. Ils peuvent être utilisés comme simples registres ou former les registres doubles BC et DE.

Le registre B est utilisé comme compteur dans l'instruction de boucle DJNZ.

Le registre double BC est utilisé comme compteur dans les instructions sur blocs, HL et DE servant de pointeurs d'adresse.

Les registres secondaires.

Le Z 80 A possède un deuxième jeu des registres que nous venons de voir. Ils sont appelés registres secondaires et désignés A', F', H', L', B', C', D' et E'.

Deux instructions (EXX et EX AF, A'F') permettent l'échange du contenu des registres principaux avec celui des registres secondaires. Quand l'échange est effectué les registres principaux deviennent registres secondaires et réciproquement.

Le programme moniteur utilise souvent l'échange des registres, aussi ces instructions machine sont proscrites lorsqu'un retour au Basic est prévu.

Les registres doubles IX et IY.

Ces registres 16 bits sont appelés registres index. Un déplacement (positif ou négatif) peut être ajouté à une adresse de base qu'ils détiennent permettant l'entrée dans une liste ou une table.

Il y a certaines restrictions d'emploi de ces registres car le système d'exploitation en mémoire morte les utilise.

Le registre Pointeur de Pile SP - La pile machine.

Le registre 16 bits SP (Stack Pointer) a pour rôle de conserver l'adresse du «sommet de la pile».

Tous les microprocesseurs utilisent une pile. Le nom est évocateur. Pile d'assiettes, pile de plateaux dans une cafeteria. Les derniers empilés sont les premiers réutilisés.

Le rangement et la récupération des informations dans une pile obéit à cette règle du dernier entré, premier sorti LIFO (Last in, First out).

Certains microprocesseurs ont des registres particuliers qui forment la pile (Pile Hardware), mais la plupart, dont le Z 80 A utilisent une pile software. Une zone en mémoire vive lui est spécialement réservée.

Le fond de la pile est à l'adresse la plus haute et les informations sont empilées dans l'ordre des adresses décroissantes comme si, défiant les lois de la pesanteur, la base de la pile était collée au plafond. Le dernier élément empilé, celui qui se trouve au «sommet» de la pile, est à l'adresse la plus faible.

La pile machine, celle qui nous intéresse, est en haut de mémoire vive juste en dessous de la pile de GOSUB. Cette dernière, qui fonctionne selon les mêmes principes, contient le ou les numéros de ligne dont l'interpréteur a besoin pour le retour des sous-programmes Basic.

La pile machine a un rôle important dans les appels de sous-programmes et lors des interruptions car les adresses de retour y sont stockées.

Elle offre au programmeur des emplacements rapidement accessibles pour le stockage temporaire de données et permet ainsi d'accroître l'efficacité des programmes.

Toute la pile machine se déplace en mémoire suivant l'importance de la pile de GOSUB. L'adresse de sa base, c'est-à-dire de la plus ancienne information empilée, est détenue par la variable système ERR-SP rangée aux adresses 16386 et 16387 d.

Le registre pointeur de pile SP contient toujours l'adresse du dernier emplacement occupé. Cette adresse décroît lorsqu'une information est empilée. Elle croît lorsqu'une information est retirée de la pile. En fait, elle augmente ou diminue de deux car tous les transferts avec la pile impliquent la manipulation de deux octets de données.

Une adresse initiale est chargée dans SP par la routine d'initialisation du système. Le contenu de ce registre est modifiable par programme (avec précaution).

Notez que le fait de sortir des données de la pile modifie SP mais n'a pas pour effet de «vider» les emplacements mémoire correspondants. L'information est simplement copiée ailleurs. Elle demeure en place jusqu'à ce qu'une autre information y soit inscrite. Nous aurons l'occasion de rappeler ce principe valable pour toutes les opérations de transfert de données.

Les registres I et R.

Ce sont des registres 8 bits très spécialisés, toujours employés séparément. Ils ne sont normalement pas utilisés par le programmeur.

Le registre I est le registre «table des vecteurs d'interruption». Il a un usage particulier sur SINCLAIR.

R est le registre «rafraîchissement mémoire». Dans le système Z 81 il est simplement utilisé pour compter le nombre de caractères qui doivent former une ligne d'écran.

ORGANISATION DE LA MÉMOIRE.

L'espace adressable du ZX 81 **standard** est de 32.768 octets : 32 K.

Les premiers 8 K sont réservés au système d'exploitation en mémoire morte (ROM) ou programme moniteur. Les 8 K suivants sont une copie des précédents. Les derniers 16 K représentent la capacité de mémoire vive maximum possible.

SYSTÈME D'EXPLOITATION - PROGRAMME MONITEUR.

Pour fonctionner d'une manière organisée, un micro-ordinateur doit disposer d'un programme moniteur même rudimentaire.

Dès la mise en route, le micro-processeur cherche en mémoire des instructions à exécuter. Dans le cas du Z 80, la première instruction doit se trouver à l'adresse zéro.

Lorsque vous mettez votre SINCLAIR sous tension, le microprocesseur commence immédiatement l'exécution du programme moniteur. Après la routine d'initialisation, le K inverse est affiché et l'interpréteur Basic qui fait partie du programme est prêt à intervenir dès que vous touchez au clavier.

Même lorsque vous attendez devant l'écran vide en essayant d'imaginer ce que sera la première ligne de votre programme Basic, le Z 80 n'est pas au repos. Il parcourt inlassablement une boucle de programme qui consiste à scruter le clavier pour détecter une éventuelle entrée, à enregistrer qu'aucune touche n'a été enfoncée et à afficher une nouvelle image (toujours blanche) de l'écran.

Lorsque la fonction USR est utilisée, le microprocesseur est libéré de la tutelle du moniteur. Il est «branché» sur l'adresse spécifiée par USR et exécute les instructions qu'il y trouve.

La taille d'un programme moniteur est déterminée par les possibilités que le fabricant désire donner à son système.

Le ZX 81 standard est doté d'un programme moniteur 8 K. Nous reviendrons plus en détail sur sa composition et nous verrons comment utiliser pour nos propres programmes quelques routines intéressantes qu'il contient.

Pour le moment retenez qu'il permet :

- L'affichage et le maintien à l'écran d'une image qui peut être composée avec 64 caractères différents.
- L'exploration du clavier et le décodage de 78 frappes différentes.
- La sauvegarde ou le chargement de programmes sur cassette.
- L'utilisation du langage Basic.

ZONE DES VARIABLES SYSTÈME.

En mémoire vive, entre les adresses 16384 et 16508 se trouve la zone des variables système. Sa composition est détaillée dans le chapitre 28 de votre manuel ZX 81 SINCLAIR. Nous aurons l'occasion de reparler de ces variables, du moins des plus importantes.

ZONE PROGRAMME EN MÉMOIRE VIVE.

A partir de l'adresse 16509 et jusqu'à :

- 17407 pour le ZX 81 - 1 K
- 32767 pour le ZX 81 - 16 K

la mémoire vive peut être utilisée pour le stockage des programmes Basic ou en code machine.

Au chapitre 27 du manuel SINCLAIR est montrée la place mémoire nécessaire au fonctionnement d'un programme Basic.

L'implantation du code machine exige certaines précautions pour éviter toute interférence. On dit quelques mots des solutions possibles dans votre manuel SINCLAIR, page 167.

Le petit livre du Z 81 (Editions du PSI) en donne une présentation très complète.

La méthode qui consiste à utiliser une instruction REM est la plus satisfaisante malgré l'aspect déplaisant qu'elle donne au listing. Nous en reparlerons.

CHAPITRE IV

PROGRAMME DE CHARGEMENT DU CODE MACHINE - PCCM

Après avoir composé votre programme en langage d'assemblage et l'avoir traduit en code hexadécimal, il vous faut l'implanter en mémoire.

Sur la plupart des micro-ordinateurs individuels, l'accès au moniteur est possible.

Cette facilité minimise les conséquences d'un crash lors de l'exécution et, pour ce qui nous préoccupe actuellement, elle permet l'introduction directe du code hexadécimal.

Malheureusement, le ZX 81 n'offre pas cette facilité. Il faut passer par l'interpréteur Basic et utiliser l'instruction POKE (chapitre 25 de votre manuel).

Le problème est le suivant :

- on dispose du code-machine en hexadécimal;
- le microprocesseur ne comprend que le binaire;
- l'interpréteur peut assurer la traduction mais il faut lui fournir le code en décimal.

Vous pourriez convertir l'hexa en décimal et implanter votre programme par une série de POKE.

C'est une corvée fastitieuse que la machine peut exécuter plus sûrement et incomparablement plus vite que vous.

Ce sera le principal travail du programme utilitaire que je vous propose.

Un tel programme est toujours un compromis entre le confort d'utilisation et l'encombrement acceptable.

Celui-ci n'est pas parfait, il est pourtant assez long. Nous allons nous en contenter.

Voyons ses caractéristiques :

- L'adresse de début est entrée en décimal.
- 200 octets de code machine peuvent être entrés et visualisés en une seule passe. L'adresse du dernier octet entré est affichée.
- L'entrée directe des chaînes de caractères est possible.
- N'importe quelle portion de mémoire ROM ou RAM (200 octets maxi à la fois) peut être affichée.
- Les lignes sont de 10 octets pour faciliter le repérage de l'adresse d'un octet particulier.
- Des modifications peuvent être faites sur le code affiché (un octet à la fois).
- L'option (abusivement appelée) «listing» donne l'affichage sous une forme plus pratique pour la mise au point.
- Principal inconvénient : il n'y a pas de routine d'insertion ou de suppression.

  Si du code doit être inséré, il faut réentrer le programme à partir de l'adresse correspondant à l'insertion (attention aux branchements).

  Pour la suppression on peut remplacer le code par 00 (NOP). Si c'est dans une chaîne de caractères en Data, 00 sera interprété comme un espace.

Armez-vous de courage et tapez le programme suivant. Surtout faites immédiatement une ou plusieurs copies de sauvegarde, car vous le perdrez souvent dans des crashs inévitables.

Vous pouvez l'appeler «PCCM» programme de chargement du code machine ou lui trouver un nom plus original.

Si vous n'êtes pas en forme aujourd'hui vous pouvez vous arrêter à la ligne 2020. Les options MODIF et LISTING ne sont pas indispensables pour commencer.

SOUS-ROUTINES

```
199 GOTO 1000

200 IF INKEY$="" THEN GOTO 200
210 LET H$=INKEY$
220 IF CODE H$<28 OR CODE H$>43 THEN GOTO 200
230 PRINT AT X,Y;H$;
240 IF INKEY$<>"" THEN GOTO 240
250 IF INKEY$="" THEN GOTO 250
260 LET U$=INKEY$
270 IF CODE U$<28 OR CODE U$>43 THEN GOTO 250
280 PRINT U$
290 POKE A,16*CODE H$+CODE U$-476
300 RETURN

400 PRINT AT X,Y;" ";A$(1);
410 POKE A,CODE A$
420 IF CODE A$=23 THEN POKE A,118
430 LET A$=A$(2 TO )
440 RETURN

500 LET A$=""
510 IF A$="" THEN INPUT A$
520 IF LEN A$<>N THEN PRINT AT 0,29;"    "
530 LET N=LEN A$
540 PRINT AT 0,29;N
550 RETURN

600 LET F=PEEK A
610 LET H=INT (F/16)
620 LET U=F-16*H
630 PRINT AT X,Y;CHR$ (H+28);CHR$ (U+28);
640 RETURN
```

PROGRAMME PRINCIPAL

```
1000 CLEAR
1010 CLS
1020 PRINT "1)ENTREE CODE HEXA"
1030 PRINT "2)ENTREE CARACTERES"
1040 PRINT "3)AFFICHAGE"
1050 PRINT "4)LISTING"
1060 INPUT R
1070 CLS
1080 PRINT "      =DEBUT (12 espaces) N.OCT=    "
1090 INPUT S
1100 PRINT AT 0,0;S
1110 INPUT N
1120 IF N<1 OR N>200 THEN GOTO 1110
1130 PRINT AT 0,29;N
1140 IF R=4 THEN GOTO 3000
1150 IF R=2 THEN GOSUB 500
1200 LET A=S
1210 LET X=1
1220 FOR Y=3 TO 30 STEP 3
1230 GOSUB 200*R
1240 LET A=A+1
1250 IF A=S+N THEN GOTO 1290
1260 NEXT Y
1270 LET X=X+1
1280 GOTO 1220
1290 PRINT AT 0,15;A-1
1300 IF R=2 THEN GOTO 1350
1310 PRINT AT 21,0;"1)MENU  2)EXEC  3)LIST
     4)MODIF"
1320 INPUT R
1330 IF R<1 OR >4 THEN GOTO 1320
1340 GOTO 1000*R
1350 PRINT AT 21,0;"PAUSE (26 espaces)"
1360 PAUSE 500
1370 LET R=3
1380 GOTO 1200

2000 CLS
2010 PRINT"ENTREZ CDE USR SOUS LA FORME
     -LET K=USR D (D=ADR.DEBUT PROGR.)"
2020 STOP
```

CONVERSION DÉCIMAL-HEXADÉCIMAL

```
2900 DIM H$(4)
2910 LET D1=B
2920 FOR I=4 TO 1 STEP -1
2930 LET D2=INT (D1/16)
2940 LET H$(I)=CHR$ (D1-16*D2+28)
2950 LET D1=D2
2960 NEXT I
2970 RETURN
```

LISTING

```
3000 CLS
3010 LET E=0
3020 LET B=S+E
3030 PRINT TAB 4;B;
3040 GOSUB 2900
3050 PRINT TAB 10;H$(1);H$(2);H$(3);H$(4);
3060 LET B=PEEK B
3070 GOSUB 2900
3080 PRINT TAB 15;H$(3);H$(4);
3090 PRINT TAB 18;CHR$ B
3100 LET E=E+1
3110 IF E=N THEN GOTO 3130
3120 GOTO 3020
3130 PRINT"1)MENU  2)EXEC  3)RECOM"
3140 INPUT R
3150 IF R=0 OR R)3 THEN GOTO 3140
3160 GOTO 1000*R
```

MODIFICATIONS

```
4000 PRINT AT 21,0;"(11 espaces)ENTREZ ADRESSE "
4010 INPUT AD
4020 IF AD(S OR AD)S+N-1 THEN GOTO 4010
4030 LET X=INT ((AD-S)/10)+1
4040 LET Y=(AD-(S+(X-1)*10))*3+3
4050 PRINT AT X,Y-1;")":REM ) INVERSE
4060 LET A=AD
4070 PRINT AT 21,0;" (25 espaces) "
4080 GOSUB 200
4090 PRINT AT X,Y-1;" "
4100 GOTO 1310
```

Le programme principal commence à la ligne 1000. Une des quatre options d'un menu peut être choisie à l'aide de son numéro. Puis le programme vous demande l'adresse de début S et le nombre d'octets N. En 1200 est la routine d'affichage à l'écran. A prend la valeur de S et servira de variable de travail. En 1240 on passe à l'adresse suivante et en 1250 on sort éventuellement de la boucle. Suivant l'option R choisie les sous-routines 200 (500 puis) 400 ou 600 sont appelées.

Option 1. - Entrée code Hexa.

L'entrée est directe au clavier (fonction INKEY$). Les lignes 220 et 270 refusent les caractères «non hexadécimaux». Ici une explication s'impose. En annexe A, page 181 de votre manuel SINCLAIR, est donné le jeu de caractères avec leur code. Deux remarques en passant :

- Le code utilisé par SINCLAIR est différent de l'ASCII (American Standard Code for Information Interchange) universellement adopté sur micro-ordinateur.
- Il n'y a pas de relation entre le code des caractères et l'interprétation binaire d'une combinaison de 8 bits. Ainsi 00011100 sera lu 1 x 16 + 1 x 8 + 1 x 4 = 28 par la commande PEEK mais le même octet signifiera 0 pour la commande CHR$.

En page 181, donc, vous remarquerez que les caractères qui peuvent être des digits hexadécimaux : 0, 1, 2... F ont un code (décimal) compris entre 28 et 43.

Les valeurs inférieures à 28 ou supérieures à 43 seront dédaignées par notre programme.

La ligne 290 est la cheville ouvrière de la sous-routine. C'est elle qui convertit nos deux digits hexadécimaux en une valeur décimale et la «poke» à l'adresse ad hoc.

Voyons comment se passe cette conversion. L'expression (16 x code H$) + code U$ -476 est en réalité 16 x (code H$ - 28) + code U$ - 28. Dans un nombre hexadécimal de deux chiffres, le premier chiffre est le chiffre des «seizaines» et le deuxième le chiffre des unités. Ainsi F 7, par exemple, vaudra en décimal 16 x 15 + 7 = 247.

Or (merci M. SINCLAIR) les caractères de 0 à F ont un code qui forme une suite continue de 28 à 43 (ce n'est pas le cas en ASCII). Il suffit de retrancher 28 de ce code pour retrouver la valeur décimale.

Option 2. - Entrée caractères.

Cette option vous servira à entrer des chaînes de caractères qui seront intégrées comme Data dans vos programmes, c'est-à-dire assez rarement en fait, mais il m'a semblé intéressant de l'inclure. Nous l'utiliseront pour l'étude des routines PRINT. L'idée est simple. Il est inutile de rechercher dans une table le code hexadécimal des caractères d'un texte, ou d'un graphique alors que la fonction CODE donne directement la valeur décimale qui nous intéresse pour POKE.

C'est ce qui est réalisé à la ligne 410.

Voyons le mode d'emploi de cette option qui est particulier.

La ligne 1150 du programme principal envoie d'abord à la sous-routine 500. Là vous composez votre chaîne de caractères **en entier**, vous la corrigez au besoin et vous **l'entrez en bloc** avec NEW-LINE. Les lignes 520 et 530 et 540 corrigent éventuellement le nombre d'octets. Non pas par indulgence envers vos erreurs de calcul mais pour ne pas perturber l'affichage des caractères en 400 (LEN A $ doit être respectée).

Si vous avez besoin d'un «à la ligne» dans votre chaîne de caractères vous devez utiliser le signe *. Ce signe est baptisé NEWLINE pour l'occasion (ligne 420).

Pour le signe de la multiplication, vous pourrez toujours utiliser x comme au bon vieux temps.

Attention, la sous-routine en 400 affiche la chaîne de caractères telle quelle et implante le code.

Ensuite a lieu une pause (que vous pouvez abréger avec NEWLINE) puis l'affichage automatique du code en hexadécimal.

Vous suivrez aisément le processus en examinant les lignes 1300, 1350 et les suivantes.

Option 3. - Affichage.

Cette option utilise la fonction PEEK. La valeur décimale obtenue en 600 est convertie en deux digits hexadécimaux qui sont affichés (ligne 630). L'opération est exactement l'inverse de celle que nous avons vue pour l'option 1.

La sous-routine en 600 est appelée automatiquement lorsque l'option 2 est choisie.

Option 4. - Listing.

On peut se passer de cette option. La présentation à l'écran est différente et les adresses sont affichées en décimal et en hexadécimal, ce qui peut faciliter les mises au point.

C'est la même sous-routine en 2900 qui assure la conversion de l'adresse (ligne 3040) et du code (ligne 3070).

L'algorithme utilisé est basé sur une méthode manuelle de conversion. Le reste des divisions successives par 16 forme le digit hexadécimal.

Soit par exemple à convertir 16509 décimal :

```
16509/16  quotient = 1031 - reste 13 soit D hexa
 1031/16  quotient =   64 - reste  7 soit 7 hexa
   64/16  quotient =    4 - reste  0 soit 0 hexa
    4/16  quotient =    0 - reste  4 soit 4 hexa
```

d'où 16509 d donne en hexa 4 0 7 D

Il faut remarquer que cette option peut servir à convertir tout nombre décimal compris entre 0 et 65535. Il suffit d'entrer le nombre en guise d'adresse et 1 pour le nombre d'octets.

L'option modifications.

Cette facilité est offerte chaque fois qu'un bloc de code vient d'être entré ou après un affichage.

On ne peut modifier le code que d'un octet à la fois.

Le repérage de l'adresse correspondante est facilité par la longueur des lignes (10 octets). Si vous faites une erreur de pointage, vous devez retaper la valeur initiale pour pouvoir continuer.

On sort du programme par BREAK pendant l'entrée du code ou par STOP.

• • •

Nous verrons plus tard où et comment placer en mémoire un programme en langage machine.

Pour expérimenter les instructions du Z 80 nous nous contenterons de choisir une adresse où l'on ne risque pas d'interférence avec le programme utilitaire.

La variable système E. LINE en 16404 et 16405 détient l'adresse de la fin d'un programme Basic, fichier d'affichage et variables compris.

Tapez PRINT PEEK 16404 + 256 * PEEK 16405

Résultat : 19750 - Puisqu'il commence en 16509, notre programme utilitaire occupe un peu plus de 3 K.

L'adresse 30000 conviendra pour la plupart de nos essais. Nous ne serons pas gênés par le Basic ni par les piles logées sous RAMTOP (32768).

Pour éviter d'avoir à entrer cette adresse à chaque fois, modifions légèrement le PCCM.

Remplacez la ligne 1090 par :

1090 LET S = 30000

Lancez le programme.

Choisissez option 1 entrée code hexa.

Vous obtenez :

30000 = début N.OCT =

entrez 10 pour le nombre d'octets
et maintenant entrez le code machine suivant :

C9 00 00 00 00 00 00 00 00 C9

Choisissez EXEC puis tapez PRINT USR 30000

30000 apparaît dans le coin supérieur gauche et le compte-rendu d'exécution 0/0 est affiché.

Tapez maintenant PRINT USR 30001. Le nombre 30001 apparaît ainsi que le compte-rendu d'exécution. Que s'est-il passé ?

1. - Vous venez de faire tourner vos premiers programmes complets en langage machine. Le prèmier (d'un octet) à l'adresse 30000. Le second (de neuf octets) commençant à l'adresse 30001.

2. - Vous avez utilisé deux instructions importantes : RET (retour) C9 et NOP (no operation) 00.

3. - Vos programmes n'ont pas fait grand chose. Pour le premier il s'agissait d'un simple aller et retour. Pour le second l'excursion était plus longue mais aussi sans but car l'instruction NOP signifie précisément pour le microprocesseur : Ne faites rien - passez à l'instruction suivante.

Retenez qu'une instruction RET est indispensable pour le retour au Basic.

Pourquoi les adresses 30000 et 30001 ont-elles été affichées après l'exécution des programmes ?

Cela est dû à la façon dont le moniteur traite la fonction USR.

Quand cette fonction est rencontrée le registre double BC est chargé avec l'adresse spécifiée avant exécution.

Au retour au Basic, le nouveau contenu de BC est affiché en décimal à condition que la commande fut PRINT USR...

Dans le cas précédent c'est l'adresse de départ qui a été affichée puisque le programme n'a pas modifié le contenu de BC.

C'est une particularité qui sera utile pour vérifier les résultats de nos manipulations.

Il suffira de les transférer dans BC avant le retour au Basic.

Nous allons utiliser la commande PRINT USR pendant quelque temps. Aussi, pour éviter des frappes superflues, modifions encore notre programme.

Remplacez la ligne 2010 par :

```
2010  PRINT USR S
```

CHAPITRE V

LE JEU D'INSTRUCTIONS DU Z 80

Il existe près de 700 instructions différentes pour le ZX 80. Vous les trouverez en annexe, classées par ordre alphabétique ou numérique en deux parties : code mnémonique et code hexadécimal - codage et décodage.

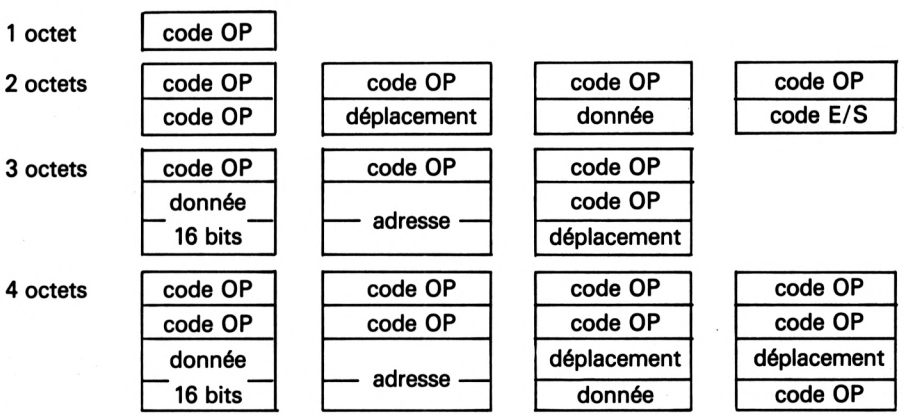
CONTEXTURE DES INSTRUCTIONS.

Une instruction complète peut être de 1, 2, 3 ou 4 octets.

Le premier octet est toujours un code opératoire. Certaines instructions commencent avec un code opératoire sur 2 octets. Le premier étant CB, DD, ED ou FD en hexadécimal.

Le code opératoire (1 ou 2 octets) peut être seul ou suivi sur un octet, d'une donnée 8 bits, d'un code d'entrée/sortie ou d'un déplacement ou suivi sur deux octets d'une donnée 16 bits, d'une adresse, ou d'un déplacement + une donnée 8 bits.

Certaines instructions (rotations, opérations sur bits) mettant en jeu les registres index ont une contexture particulière. Les 2 premiers et le 4e octet forment le code opératoire, le 3e octet désigne un déplacement.



Dans le code mnémonique, données, déplacements, adresses seront symbolisés de la façon suivante :

n = donnée (nombre, valeur, constante) sur un octet (8 bits) comprise entre 0 et 255 d.

nn = donnée (nombre, valeur, constante) sur deux octets (16 bits) comprise entre 0 et 65535 d.

d = déplacement (sur un octet) compris entre — 128 et + 127.

ADR = adresse sur 2 octets.

Un code d'entrée-sortie est une constante sur un octet qui désigne un périphérique.

Pour le ZX 81 cela se résume à :

FE en hexadécimal pour l'entrée clavier ou cassette.
FF pour la sortie TV ou cassette.
FB pour la sortie imprimante.

CLASSEMENT DES INSTRUCTIONS.

Le classement, en vue de leur étude, des 700 instructions du Z 80 n'est pas évident.

Faut-il, dans un intérêt pédagogique, les regrouper par ordre de complexité croissante ou bien, dans un but pratique, étudier d'abord les plus couramment utilisées ?

La logique demande qu'elles soient regroupées par fonctions. Par souci de clarté, c'est ce principe qui a été retenu pour le découpage proposé, tout à fait classique.

Les instructions sont divisées en huit groupes fonctionnels qui feront l'objet des huit prochains chapitres.

Nous allons examiner le contenu de ces huit groupes et ainsi faire connaissance avec les codes mnémoniques.

Les deux premiers groupes concernent exclusivement le transfert d'informations.

1er GROUPE : INSTRUCTIONS DE TRANSFERT SANS RÉFÉRENCE A UNE ADRESSE MÉMOIRE.

Il s'agit de transfert interne : transfert ou échange entre registres mais aussi d'instructions permettant de charger un registre ou d'utiliser la pile. Les codes mnémoniques rencontrés dans ce groupes sont les suivants :

LD **Load - Charger**

C'est l'instruction la plus utilisée de tout le langage machine. Elle sert à transférer les informations d'un endroit à un autre et prend de nombreuses formes.

Certaines opérations de transfert sont possibles avec LD, d'autres ne le sont pas. Cette instruction est employée si souvent que vous deviendrez rapidement familier avec elle.

EX **EXX** **Exchange - Echanger.**

Ces instructions permettent l'échange du contenu de registres ou d'emplacements mémoire. Ce groupe comprend les échanges entre registres dont l'instruction très utile EX DE, HL.

PUSH **«Pousser»** empilage **POP** **«Faire sauter»** désempilage

PUSH place le contenu d'un registre double sur la pile (2 octets).

POP tranfère deux octets de données de la pile dans un registre double.

2e GROUPE : INSTRUCTIONS DE TRANSFERT FAISANT RÉFÉRENCE A UN EMPLACEMENT MÉMOIRE.

La notion de mode d'adressage est introduite en début de chapitre.

Les codes mnémoniques sont :

LD

On retrouve l'instruction LD. Vous noterez la signification des parenthèses autour de la désignation de l'emplacement mémoire. Suivant le cas : contenu de, contenu de l'emplacement mémoire pointé par ou emplacement mémoire désigné par.

EX

L'instruction EX dans ce groupe concerne l'échange des deux octets du sommet de la pile avec le contenu d'un des registres doubles HL, IX ou IY.

3e GROUPE : INSTRUCTIONS ARITHMÉTIQUES - COMPARAISONS

Avec ce groupe commencent à apparaître les instructions de traitement de l'information. Le rôle important des indicateurs d'état sera précisé notamment celui de l'indicateur de retenue.

ADD **Additionner.**

Cette instruction permet sur 8 bits d'ajouter au contenu de A une constante 8 bits ou le contenu d'un registre simple ou encore le contenu d'un emplacement mémoire pointé par HL ou à l'aide d'un registre index. Le résultat reste dans l'accumulateur.

Sur 16 bits elle permet d'ajouter au contenu de HL ou d'un registre index le contenu d'un registre double.

INC **Increment - Incrémenter**

Cette instruction signifie simplement augmenter de un.

Les principaux registres simples ou doubles peuvent être incrémentés ainsi que le contenu d'un emplacement mémoire pointé par HL ou indexé par IX ou IY.

ADC **Add with Carry - Additionner avec retenue**

Les opérations possibles sont les mêmes que pour ADD sauf que pour les additions sur 16 bits les registres index ne sont plus concernés.

Cette fois l'état actuel de l'indicateur de retenue est pris en compte dans le résultat.

SUB **Subtract - Soustraire.**

Les instructions SUB permettent de soustraire du contenu de l'accumulateur une valeur sur 8 bits.

Il n'existe pas d'instruction SUB sur 16 bits.

Dans le mnémonique A est généralement omis ainsi SUB n signifie SUB A, n.

SUB A signifie soustraire A de A ou SUB A, A.

DEC **Décrément - Décrémenter - Diminuer de un.**

Ces instructions portent sur les mêmes registres ou valeurs que pour l'incrémentation.

SBC **Subtract with Carry - Soustaire avec retenue.**

Ces instructions sont symétriques des instructions ADC. Elles portent sur les mêmes registres. La valeur du bit de retenue est soustraite du résultat.

CP **Compare - Comparer.**

La comparaison consiste à soustraire de l'accumulateur une valeur spécifiée ou le contenu d'un registre simple ou d'un emplacement mémoire pointé par HL ou indexé par IX ou IY.

Le résultat n'est pas conservé et le contenu de A n'est pas altéré mais l'opération affecte les indicateurs d'état aussi CP est toujours suivie d'une instruction de branchement conditionnelle basée sur la position d'un indicateur.

4e GROUPE : INSTRUCTIONS LOGIQUES.

Les instructions de ce groupe utilisent les opérateurs logiques (booléens) AND, OR et XOR.

Elles permettent des opérations bit à bit sur le contenu de l'accumulateur et une valeur 8 bits spécifiée ou contenue dans un registre simple ou dans un emplacement mémoire pointé par HL ou indexé par IX ou IY.

On a inclus dans ce groupe certaines opérations sur l'accumulateur ou l'indicateur de retenue.

AND **et logique**

OR **ou logique**

XOR **ou exclusif logique**

CPL **Complément - Complémenter (l'accumulateur)**

NEG **Negate - Rendre négatif (l'accumulateur)**

CCF **Complement Carry Flag - Complémenter l'indicateur de retenue**

SCF **Set Carry Flag - Mettre à un l'indicateur de retenue.**

5e GROUPE : INSTRUCTIONS DE BRANCHEMENT.

Ce sont elles qui donnent son aspect dynamique à un programme. On distingue les instructions de saut : JP, JR, DJNZ; les appels de sous-programmes : CALL et RST; les instructions de retour : RET.

JP **Jump - Sauter.**

Cette instruction est le GOTO du langage machine. La destination n'est pas un numéro de ligne, c'est une adresse en mémoire mais le principe est le même. L'adresse peut être indiquée en clair ou pointée par HL ou indexée par IX ou IY.

Les sauts peuvent être conditionnels, dépendants de la position d'un des quatre indicateurs d'état C, Z, S ou P/V.

JR **Jump Relative - Saut relatif.**

L'adresse de destination est obtenue en ajoutant un déplacement d positif ou négatif (+ 127 à — 128) à l'adresse de l'instruction suivante.

Les sauts relatifs peuvent être conditionnels sur la position d'un des deux indicateurs C ou Z.

DJNZ **Decrement B and Jump relative if Not Zero**
Décrémenter B et Saut relatif si Non Zéro.

Cette instruction de saut relatif conditionnel sert à composer des boucles. Le registre B sert de compteur (variable de contrôle). Il est décrémenté à chaque itération.

B = 0 est la condition de fin de boucle.

L'instruction occupe deux octets et s'écrit DJNZ d où d est le déplacement signé (normalement négatif).

CALL **Appeler.**

L'adresse du sous-programme appelé est toujours donnée en clair (adressage absolu).

L'appel peut être conditionnel sur le test d'un des quatre indicateurs d'état C, Z, S ou P/V.

RST **Restart - Recommencer.**

C'est aussi un appel de sous-programme mais dont l'adresse est prédéterminée en page zéro (adresse inférieure à 256 d -).

Il n'y a pas d'appel conditionnel. 8 sous-programmes différents peuvent être appelés par RST. Ces sous-programmes font partie du programme moniteur en ROM.

RET **Return - Retour.**

Outre le simple retour (C9 en hexa). Il existe 8 instructions de retour conditionnel basé sur la position des indicateurs d'état C, Z, S et P/V.

Les deux instructions d'usage général NOP et HALT ont été incluses dans ce groupe.

NOP **No Operation - Pas d'opération.**

HALT **Halte.**

6e GROUPE : INSTRUCTIONS SUR BLOCS.

Ce sont des instructions composées destinées à déplacer ou à rechercher des données en mémoire.

LDIR — **Load, Increment and Repeat**
Charger, Incrémenter et recommencer

LDDR — **Load, Decrement and Repeat**
Charger, Décrémenter et Recommencer.

LDI — **Load and Increment - Charger et Incrémenter.**

LDD — **Load and Decrement - Charger et Décrémenter.**

CPIR — **Compare, Increment and Repeat**
Comparer, Incrémenter et Recommencer

CPDR — **Compare, Decrement and Repeat**
Comparer, Décrémenter et Recommencer

CPI — **Compare and Increment - Comparer et Incrémenter.**

CPD — **Compare and Decrement - Comparer et Décrémenter.**

7e GROUPE : ROTATIONS - DÉCALAGES - OPÉRATIONS SUR BITS

Les instructions de rotation ou de décalage déplacent vers la gauche ou vers la droite l'ensemble des bits d'un octet spécifié.

Elles utilisent ou affectent le bit de retenue d'une certaine manière.

RL — **Rotate left - Rotation à gauche.**

RR — **Rotate Right - Rotation à droite.**

RLC — **Rotate Left Circular. - Rotation circulaire à gauche.**

RRC — **Rotate Right Circular. - Rotation circulaire à droite.**

SLA — **Shift left arithmetic. - Décalage arithmétique à gauche.**

SRA — **Shift right arithmetic. - Décalage arithmétique à droite.**

SRL — **Shift right logical - Décalage logique à droite.**

Les instructions sur bits permettent soit de tester la valeur d'un bit particulier soit de la modifier.

BIT — **Test d'un bit (positionne l'indicateur zéro).**

SET — **Mettre à un.**

RES — **Reset - Mettre à zéro (ou remettre à zéro).**

Les instructions suivantes de ce chapitre sont surtout utilisées dans les programmes où la représentation décimal codé binaire (DCB) est employée.

DAA — **Decimal Adjust Accumulator**
Ajustement décimal de l'accumulateur

RLD — **Rotate left decimal**

RRD — **Rotate right decimal**

8e GROUPE : ENTRÉES - SORTIES - INTERRUPTIONS.

Les instructions d'entrées/sorties permettent au microprocesseur de communiquer avec un périphérique.

Il existe des instructions simples.

IN — **Input - Entrée.**

OUT — **Sortie.**

et des instructions composées analogues aux instructions sur blocs.

INI **INIR** **IND** **INDR**

OUTI **OUTIR** **OUTD** **OUTDR**

Ces dernières ne sont pas utilisées sur ZX 81.

Vous n'utiliserez pas non plus les instructions d'interruption :

EI — **Enable interrupt - Interruptions autorisées.**

DI — **Disable interrupt - Interruptions interdites.**

IM 0
IM 1
IM 2

Interrupt Mode 0, 1 ou 2
mode d'interruption.
suppose une autorisation EI.

RETI — **Return from maskable interrupt**
Retour d'une sous-routine d'interruption masquable.

RETN — **Return from non-maskable interrupt**
Retour d'une sous-routine d'interruption non masquable.

CHAPITRE VI

CHARGEMENT DE REGISTRES TRANSFERTS ET ÉCHANGES ENTRE REGISTRES UTILISATION DE LA PILE

Ces instructions de transfert d'information ne font pas référence à une adresse-mémoire. Le mode d'adressage est immédiat ou implicite.

1. - CHARGEMENT D'UN REGISTRE SIMPLE OU DOUBLE.

Ces instructions sont parmi les plus utilisées. Elles ont une certaine analogie avec l'instruction Basic. LET : affectation d'une valeur à une variable.

CHARGEMENT D'UN REGISTRE SIMPLE.

Ces instructions sont de deux octets. Un octet pour le code opératoire (instruction proprement dite) et un octet pour la constante (n). Valeur comprise entre 0 et 255 décimal. (00 et FF hexadécimal).

mnémonique	code hexa
LD A, n	3E XX
LD H, n	26 XX
LD L, n	2E XX
LD B, n	06 XX
LD C, n	OE XX
LD D, n	16 XX
LD E, n	1E XX

CHARGEMENT D'UN REGISTRE DOUBLE.

Ces instructions sont de trois ou quatre octets. Elles ont pour effet de charger le registre double avec une copie de la constante représentée par les 2 octets qui suivent le code opératoire (lui-même sur un ou deux octets).

La constante peut désigner un nombre décimal entre 0 et 65535 (0000 - FFFF hexa) ou n'importe quelle adresse en mémoire.

mnémonique	code hexa
LD HL, nn	21 XX XX
LD BC, nn	01 XX XX
LD DE, nn	11 XX XX
LD IX, nn	DD 21 XX XX
LD IY, nn	FD 21 XX XX
LD SP, nn	31 XX XX

Dans l'instruction codée, les octets représentant la constante sont inversés.

Exemple : LD BC, 4082 h est codée 01 82 40. L'octet de poids faible 82 ira dans le registre faible C; l'octet de poids fort 40 ira dans le registre fort B.

Les registres 16 bits (insécables) IX, IY et SP sont spécialisés. Leur utilisation fait l'objet des remarques suivantes :

- Le registre index IX est utilisé par le système pour l'affichage à l'écran. Ce registre est interdit d'emploi en mode SLOW.
- Le registre index IY est chargé avec 4000 hexa adresse du début de la zone des variables système par la routine d'initialisation. Le passage concerné est en 1016 d - 03F8

  LD IY, 4000 FD 21 00 40

  Le contenu de IY peut être changé dans un programme. Il retrouvera automatiquement sa valeur initiale 4000 h dès le retour au Basic.
- Toute instruction modifiant SP le registre pointeur de pile est à manipuler avec précaution.

APPLICATION :

- **Chargement d'un registre simple.**

Soit à charger la valeur décimale 50 (32 hexa) dans le registre C.

adresse	mnémonique	code hexa
30000	LD C, 32 h	0E 32
	LD B, 00	06 00
	RET	C9

Entrez le nombre d'octets N.OCT = 5
puis le code hexadécimal dans l'ordre des instructions 0E 32 06 00 C9

Notez que l'on doit charger B avec 00, sinon après exécution l'affichage donnerait une valeur aberrante (B contenant l'octet de poids fort de l'adresse USR).

Faites EXEC - la valeur décimale 50 s'affiche, ce qui est le contenu de C puisque B = 0.

BC contient 0032 en hexa - soit en décimal (4096 x 0) + (256 x 0) + (16 x 3) + (1 x 2) = 50.

- **Chargement d'un registre double.**

On peut charger BC directement avec la valeur précédente 50 d - 32 hexa.

La constante doit être sur 2 octets, soit 0032 h.

mnémonique	code hexa
LD BC, 0032 h	01 32 00 (LO) (HI)
RET	C9

Il est impératif que vous soyez dès maintenant familier avec l'inversion des octets.

En mémoire ou dans un programme l'octet le moins significatif (octet de poids faible, octet bas, low byte, LO) est stocké en premier, c'est-à-dire à l'adresse la plus faible.

Entraînez-vous à changer BC avec des valeurs décimales supérieures à 255.

2. - TRANSFERT DU CONTENU D'UN REGISTRE DANS UN AUTRE.

Notez que le fait de transférer des données d'un registre dans un autre ne «vide» pas le registre source. Son contenu est simplement copié dans l'autre, il reste intact jusqu'à ce qu'une autre instruction vienne le modifier.

TRANSFERT ENTRE REGISTRES SIMPLES.

Les instructions qui permettent de copier le contenu d'un registre simple dans un autre sont fréquemment utilisées.

Elles occupent un seul octet et sont exécutées très rapidement (1,23 microsec.).

Elles ont une certaine similitude avec LET A = B du Basic.

LD avec → / ↓	A	B	C	D	E	H	L
A	7F	78	79	7A	7B	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	6B	6C	6D

Exemple : charger le registre B avec le contenu du registre H : LD B, H → 44.

Les instructions suivantes qui mettent en jeu les registres I et R ne sont pas utilisées par le programmeur. Elles sont citées pour mémoire :

mnémonique	code hexa
LD A, I	ED 57
LD A, R	ED 5F
LD I, A	ED 47
LD R, A	ED 4F

TRANSFERT ENTRE REGISTRES DOUBLES.

Trois instructions permettent de copier le contenu des registres 16 bits HL, IX et IY dans le registre pointeur de pile SP.

mnémonique	code hexa
LD SP, HL	F9
LD SP, IX	DD F9
LD SP, IY	FD F9

Ces instructions sont très spécialisées.

L'instruction LD SP, HL est utilisée par le programme moniteur dans la routine d'initialisation (adresse 1004 d - 03EC h). Elle a pour but de faire pointer SP vers un endroit de la mémoire vive où la pile peut être installée.

APPLICATION :

Soit à copier dans C le contenu de L.

Chargeons L avec une constante, prenons n = 63 d (3F h)

LD L, 3F	2E 3F

Transférons le contenu de L dans C.

LD C, L	4D

Il faut annuler le contenu de B.

LD B, 00	06 00

mnémonique	code hexa
LD L, 3F	2E 3F
LD C, L	4D
LD B, 00	06 00
RET	C9

Une variante :

LD HL, 003F	21 3F 00
LD C, L	4D
LD B, H	44
RET	C9

Essayez avec d'autres registres et d'autres valeurs n.

3. - ÉCHANGES ENTRE REGISTRES.

Trois instructions permettent l'échange du contenu de registres, ce sont :

mnémonique	code hexa
EX DE, HL	EB
EXX	D9
EX AF, A'F'	08

L'instruction EX DE, HL est très utile. En effet, certaines opérations peuvent être effectuées sur un registre double et non pas sur un autre. C'est le cas de l'addition de deux nombres sur 16 bits qui nécessite l'utilisation de HL. Ainsi, si un nombre contenu dans DE doit être pris en compte dans une addition, DE et HL sont échangés, l'addition est effectuée et les registres sont échangés à nouveau.

Cette technique est fréquemment employée. On la trouve dans le programme moniteur à l'adresse 2496 d - (0960 hexa) - routine de changement des pointeurs. Pour pouvoir additionner BC et DE (il n'existe pas d'instruction pour cela) on échange DE, HL on fait ADD HL, BC puis on échange à nouveau → EB 09 EB.

L'instruction EXX réalise l'échange des registres principaux HL, B, C, D, et E avec les registres secondaires H', L', B', D', C' et E'.

L'instruction EX AF, AF' échange AF avec A'F'.

L'utilisation des registres secondaires est délicate. Elle est fréquente dans le programme moniteur. Notamment EX AF, A'F' est utilisée dans la routine d'affichage en mode lent (adresse 529 d) et doit être alors proscrite (voir page 167 de votre manuel SINCLAIR).

APPLICATION :

L'instruction EX DE, HL.

Dans cet exercice, vous chargez HL avec une valeur comprise entre 0 et 65535 décimal.

A vous de choisir cette valeur nn.

Pour la conversion décimal → hexa vous ne pouvez pas utiliser l'option 4 du PCCM car S est fixé à 30000. Pour vous éviter de perdre du temps dans une table, voici quelques nombres convertis en hexadécimal.

255 = FF	256 = 100	768 = 300
1024 = 400	2048 = 800	4096 = 1000
16384 = 4000	16388 = 4004	16404 = 4014
16417 = 4021	16507 = 407B	16514 = 4082
30000 = 7530		

N'oubliez pas l'inversion des octets en codant l'instruction de chargement de HL.

Assemblez le programme suivant. Il doit être de 12 octets.

mnémonique	code hexa
LD HL, nn	
EX DE, HL	
LD C, D	
LD B, E	
LD D, C	
LD E, B	
EX DE, HL	
LD B, H	
LD C, L	
RET	

La valeur chargée en HL fait un petit tour entre les registres (chaque octet séparément). Vous devriez la retrouver affichée à l'écran (espérons-le). Essayez différentes valeurs pour nn. Il n'est pas nécessaire de retaper tout le programme. Choisissez option 3, affichage 12 octets et modifiez le contenu des adresses 30001 et 30002.

4. - UTILISATION DE LA PILE.

Rappelons brièvement la définition de la pile. C'est un espace réservé en mémoire vive, utilisé par le microprocesseur ou par le programmeur pour le stockage temporaire d'adresses ou de données.

Elle fonctionne suivant le principe LIFO (last in, first out) dernier entré, premier sorti.

Le registre pointeur de pile SP (stack pointer) contient l'adresse du dernier emplacement mémoire rempli.

Les instructions à la disposition du programmeur ont été incluses dans ce chapitre car, bien que des emplacements mémoire soient concernés, ces derniers n'ont pas à être spécifiés.

PUSH - Empilage **POP** - Désempilage.

Ces instructions permettent de mettre en pile (PUSH) ou d'en retirer (POP) deux octets de données.

Pour l'instruction PUSH ces deux octets proviennent d'un registre double spécifié et pour POP ils sont transférés dans un registre double spécifié.

Quand une instruction PUSH est exécutée le registre SP est décrémenté (diminué de 1). L'octet contenu dans le registre fort est copié dans l'emplacement mémoire pointé par SP. Ce dernier est décrémenté à nouveau, puis l'octet du registre faible est copié dans le nouvel emplacement mémoire désigné par SP.

L'opération inverse a lieu pour l'instruction POP.

mnémonique	code hexa	mnémonique	code hexa
PUSH AF	F5	POP AF	F1
PUSH HL	E5	POP HL	E1
PUSH BC	C5	POP BC	C1
PUSH DE	D5	POP DE	D1
PUSH IX	DD E5	POP IX	DD E1
PUSH IY	FD E5	POP IY	FD E1

Il est important de souligner qu'il n'est pas gardé trace de la provenance des 2 octets empilés par PUSH. Ainsi, deux octets provenant de HL peuvent être transférés dans BC ou un autre registre double par POP.

Ces instructions permettent l'échange de registres doubles.

Exemple : C5 D5 C1 D1. BC et DE sont échangés.

PUSH AF et POP AF sont utilisées principalement pour sauvegarder le contenu de A et le recouvrer le moment voulu. F est apparié avec A pour l'occasion. Cela peut être utile dans le cas où l'on a besoin de A temporairement pour des calculs.

Les instructions PUSH et POP sont utilisées par le programme moniteur dans la routine d'affichage en mode SLOW. Au moment de l'interruption les valeurs contenues dans les 4 principaux registres doubles AF, BC, DE et HL sont sauvegardées dans la pile (adresses, 544 à 547 d) et recouvrées ensuite (adresses 676 à 679 d) dans l'ordre inverse.

APPLICATION :

Le programme suivant occupe 6 octets. Assemblez-le et faites-le tourner.

Il vous montre comment un nombre peut être transféré d'un registre double à un autre avec PUSH et POP.

mnémonique

```
LD HL, nn
PUSH HL
POP BC
RET
```

PUSH HL, POP BC est l'équivalent de LD B, H et LD C, L (LD BC, HL n'existe pas).

Conclusion : Dans ce chapitre nous avons vu des instructions très importantes, toutes axées sur le transfert d'informations : chargement d'un registre, copie du contenu d'un registre dans un autre, échange de registres, empilage, désempilage.

Nous allons étudier d'autres instructions de transfert. Il s'agira de transfert entre registres et mémoire et vice-versa.

Avant de continuer, je vous encourage vivement à expérimenter les instructions citées. Laissez de côté celles où les registres IX, SP, I et R sont impliqués.

De nombreuses combinaisons sont possibles. N'oubliez pas de transférer en BC le résultat de vos manipulations avant le retour au Basic.

Ne craignez pas les erreurs, elles sont pleines d'enseignements. Vous ne risquez en aucune façon d'endommager votre ordinateur.

Ecrivez toujours le programme en assembleur avant d'entrer le code machine en mémoire.

CHAPITRE VII

TRANSFERTS ENTRE REGISTRES ET MÉMOIRE

Jusqu'ici les instructions utilisées avaient pour but de manipuler des données directement accessibles soit qu'il s'agissait d'une constante incluse dans l'instruction, soit d'information à faire passer d'un registre à un autre, d'un registre vers la pile ou de la pile vers un registre.

A partir de maintenant nous allons rencontrer des instructions qui font référence à un emplacement mémoire pour le transfert de données ou leur traitement ou pour indiquer l'emplacement de la prochaine instruction à exécuter.

Il est utile de préciser ce qu'est un mode d'adressage, c'est-à-dire la façon dont est spécifiée dans une instruction l'adresse d'un emplacement mémoire.

LES MODES D'ADRESSAGE.

ADRESSAGE IMMÉDIAT.

Dans ce mode le code opératoire est suivi de la donnée elle-même. Ce n'est pas à proprement parler un mode d'adressage puisque l'indication de l'adresse est inutile.

Les instructions de chargement d'un registre avec une constante, vues au chapitre précédent, utilisent ce mode. Quand la constante est de 16 bits on l'appelle parfois adressage immédiat étendu.

ADRESSAGE IMPLICITE (OU INHÉRENT).

Ce n'est pas non plus vraiment un mode d'adressage. Nous avons rencontré un nombre d'instructions qui l'utilisent : transfert ou échanges entre registres (ce mode est d'ailleurs quelquefois appelé adressage par registre dans ce cas particulier).

Il concerne aussi les instructions d'utilisation de la pile ou les retours de sous-programme. Là, bien que les informations soient à chercher en mémoire, le microprocesseur sait «implicitement» où les trouver grâce au pointeur de pile.

On peut aussi rattacher à ce mode certaines opérations sur l'accumulateur comme CPL, NEG, DAA.

Nous arrivons maintenant aux modes d'adressage proprement dits utilisés par le Z 80 A.

ADRESSAGE ABSOLU.

C'est le mode d'adressage le plus naturel. L'adresse sur 2 octets figure en clair dans l'instruction.

Exemple :	LD A, (ADR)	3A XX XX
	LD (ADR), A	32 XX XX
	JP ADR	C3 XX XX

Notez que les parenthèses autour de l'adresse signifient : contenu de.

ADRESSAGE INDIRECT PAR REGISTRE.

Dans l'adressage indirect ce n'est pas l'adresse qui est spécifiée mais l'adresse de l'adresse. Le Z 80 ne possède qu'une variante limitée de ce mode que l'on appelle adressage indirect par registre.

Les registres doubles HL, BC ou DE servent de «pointeurs» d'adresse. Ainsi l'instruction LD A, (HL) signifie charger A avec le contenu de l'adresse désignée par HL; LD (HL), A signifie stocker le contenu de A dans l'emplacement mémoire pointé par HL.

ADRESSAGE INDEXÉ.

L'adressage indexé est utile pour accéder rapidement aux éléments d'un bloc ou d'une table. Le principe de ce mode d'adressage est que l'instruction spécifie à la fois un registre index et une adresse.

Pour le Z 80 il s'agit d'un adressage indexé très limité.

Les registres index 16 bits IX et IY contiennent une «adresse de base» à laquelle on peut ajouter un déplacement d compris entre — 128 et + 127.

ADRESSAGE RELATIF.

Ce mode est utilisé uniquement par certaines instructions de saut (JR).

Il offre deux avantages :

- Il permet d'économiser un octet. En effet ce n'est pas une adresse sur deux octets qui est spécifiée, mais un déplacement par rapport à l'adresse actuelle pointée par PC.

  Ce déplacement, spécifié par un octet signé, est limité à 127 octets en avant et 128 octets en arrière.

- Il permet de composer des programmes **relogeables** (ne comportant pas d'adresse nommément désignée).

ADRESSAGE PAGE ZÉRO.

La mémoire d'un système peut être divisée en «pages» de 256 octets. La page zéro comprend les adresses de 0 à 255 d qui peuvent être désignées sur un octet à condition de disposer d'instructions particulières connues du microprocesseur.

Sur ZX 81 la page zéro est occupée par le programme moniteur du système et n'est donc pas directement accessible au programmeur. Cependant des sous-routines importantes du système d'exploitation implantées en page zéro peuvent être appelées par des instructions sur un octet (RST).

Nous en resterons là en ce qui concerne les modes d'adressage pour ne pas apporter de confusion, car la terminologie employée est souvent imprécise. Ainsi l'adressage page zéro est quelquefois appelé adressage court, quelquefois adressage direct. L'adressage absolu est parfois appelé adressage direct, ce qui est assez logique.

Sachez quand même que dans l'élaboration des modes d'adressage, le but des concepteurs a été la diminution de l'encombrement mémoire et l'accroissement de la rapidité d'exécution.

CHARGEMENT DE REGISTRES AVEC DES DONNÉES PROVENANT D'EMPLACEMENTS MÉMOIRE.

Le Z 80 possède de nombreuses instructions qui «vont chercher» des données en mémoire pour en charger un registre.

L'adresse où trouver l'information et le nom du registre simple ou double dans lequel l'information doit être placée sont indiqués.

L'instruction est de la forme
LD registre, (emplacement mémoire).

Rappelez-vous. Les parenthèses signifient : contenu de.

Suivant le mode d'adressage, l'emplacement mémoire sera :

- désigné directement - adressage absolu;
- pointé par un registre double - adressage indirect;
- formé par l'addition d'une «adresse de base» détenue par un registre index et d'un déplacement - adressage indexé.

Il y a une grande analogie entre ces instructions et le PEEK du Basic. Pour la souligner une ligne de pseudo-Basic figurera en face de chaque instruction.

1. - INSTRUCTIONS UTILISANT L'ADRESSAGE ABSOLU.

Dans ces instructions longues de 3 ou 4 octets, l'adresse sur deux octets suit le code opératoire

mnémonique	code hexa	
LD A, (ADR)	3A XX XX	LET A = PEEK ADR
LD HL, (ADR)	2A XX XX	LET L = PEEK ADR
		LET H = PEEK (ADR + 1)
LD BC, (ADR)	ED 4B XX XX	LET C = PEEK ADR
		LET B = PEEK (ADR + 1)
LD DE, (ADR)	ED 5B XX XX	LET E = PEEK ADR
		LET D = PEEK (ADR + 1)
LD IX, (ADR)	DD 2A XX XX	LET IX = PEEK ADR + 256 * PEEK (ADR + 1)
LD IY, (ADR)	FD 2A XX XX	LET IY = PEEK ADR + 256 * PEEK (ADR + 1)
LD SP, (ADR)	ED 7B XX XX	LET SP = PEEK ADR + 256 * PEEK (ADR + 1)

une autre forme pour LD HL, (ADR) est ED 6B XX XX.

Vous remarquerez que le registre A est le seul registre concerné parmi les instructions de ce groupe qui peut être chargé avec un simple octet copié d'une adresse mémoire.

Cette instruction LD A, (ADR) est utilisée par le programme moniteur dans la routine d'initialisation (adresse 1207 d) afin de mesurer la taille mémoire disponible et de déterminer si oui ou non la mémoire vive peut contenir un fichier d'affichage complet. L'accumulateur est chargé avec l'octet de poids fort de RAMTOP en 16389 d - 4005 h. Cette valeur sera ensuite comparée avec la constante 4D pour savoir si 3 1/4 K sont disponibles ou pas (voir page 172 manuel SINCLAIR)

1207 04B7 LD A, (4005) 3A 05 40

Notez bien encore que les octets sont inversés. L'octet de poids faible est en premier.

Les autres instructions permettent le chargement d'un registre double avec **deux octets de données provenant de deux adresses consécutives.**

En réalité le mnémonique de ces instructions est abrégé, par exemple LD DE, (ADR) signifie LD E, (ADR) et LD D, (ADR + 1) comme le montre l'analogie avec le Basic.

L'octet adressé est copié dans le registre bas et l'octet suivant est copié dans le registre haut.

Ces instructions sont fréquemment utilisées.

APPLICATION :

Cherchons par exemple à connaître le numéro de ligne de la position de PRINT. C'est une variable système sur un octet à l'adresse 16442 d - 403A h (manuel SINCLAIR, page 179).

On peut utiliser A.

```
LD A, (16442 d)          3A 3A 40
LD C, A                  4F
LD B, 0                  06 00
RET                      C9
    ou BC
LD BC, (16442 d)         ED 4B 3A 40
LD B, 0                  06 00
RET                      C9
```

Vous rappelez-vous comment nous avions trouvé l'adresse de la fin de notre programme utilitaire ?

C'est la variable système E-LINE qui la détient en 16404 d (4014 h)

Deux lignes d'instructions en code machine doivent vous l'afficher à l'écran (5 octets). Essayez.

Ne soyez pas déçu si vous trouvez un résultat différent de celui que nous avions trouvé précédemment. C'est normal. Nous avons modifié le programme et il a tourné.

La zone des variables système est une zone mémoire essentiellement dynamique, en perpétuel changement.

2. - INSTRUCTIONS UTILISANT L'ADRESSAGE INDIRECT REGISTRE.

Les trois registres doubles HL, BC ou DE sont utilisés pour indiquer l'adresse mémoire où se trouve la donnée. Cela suppose évidemment que ces registres contiennent au préalable l'adresse requise.

L'accumulateur peut être chargé avec le contenu d'une adresse pointée par un quiconque des trois registres doubles, mais les autres registres simples ne peuvent être chargés qu'avec le contenu de l'adresse pointée par HL.

mnémonique	code hexa	
LD A, (HL)	7E	LET A = PEEK HL
LD A, (BC)	0A	LET A = PEEK BC
LD A, (DE)	1A	LET A = PEEK DE
LD H, (HL)	66	LET H = PEEK HL
LD L, (HL)	6E	LET L = PEEK HL
LD B, (HL)	46	LET B = PEEK HL
LD C, (HL)	4E	LET C = PEEK HL
LD D, (HL)	56	LET D = PEEK HL
LD E, (HL)	5E	LET E = PEEK HL

Toutes ces instructions sont sur un octet. Il faut rappeler qu'elles ont pour résultat de charger un registre simple avec une valeur sur 8 bits (un octet).

L'exemple suivant tiré du programme moniteur montre comment un registre double DE peut être chargé avec 2 octets consécutifs grâce à HL utilisé comme pointeur d'adresse. Nous venons de dire que la zone des variables système était une zone très active. Ainsi chaque fois que des caractères doivent être ajoutés ou supprimés du programme Basic, les pointeurs sont modifiés. Ces 9 pointeurs contiennent les adresses des limites entre les différentes zones. Ils s'appellent D. FILE, DF CC...STKEND et sont logés entre 16396 et 16413 d. (voir manuel SINCLAIR, pages 171 et 178).

La routine de «changement des pointeurs» est très souvent utilisée. Le début de la routine, ci-dessous, concerne «l'extraction» de l'adresse détenue par D. FILE en 16396 et 16397.

adresse	mnémonique	code hexa	
2479 - 09AF	LD HL, 400C	21 0C 40	LET HL = 16396
	LD A, 09	3E 09	LET A = 9 (pointeurs)
	LD E, (HL)	5E	LET E = PEEK 16396
	INC HL	23	LET HL = HL + 1
	LD D, (HL)	56	LET D = PEEK 16397

HL est chargé avec l'adresse de base de la liste. INC HL (augmenter de 1) permet de passer à l'adresse suivante.

L'octet de poids faible va en E, l'octet de poids fort va en D. Ainsi DE est chargé avec la valeur de D. FILE.

APPLICATION :

Le PCCM étant chargé, faites en commande directe : POKE 32000, n (n compris entre 0 et 255).

Maintenant assemblez, entrez puis faites tourner les programmes suivants l'un après l'autre. 32000 d = 7D00 h.

```
LD HL, 32000 d          LD DE, 32000 d
LD C, (HL)              EX DE, HL
LD B, 00                LD C, (HL)
RET                     LD B, 00
                        RET
```

```
LD BC, 32000 d          LD DE, (32000 d)
LD A, (BC)              LD A, E
LD E, A                 PUSH AF
LD D, 00                POP BC
EX DE, HL               LD C, B
PUSH HL                 LD B, 0
POP BC                  RET
RET
```

A vous de trouver d'autres combinaisons.

3. - INSTRUCTIONS UTILISANT L'ADRESSAGE INDEXÉ.

Ces instructions permettent le chargement des registres simples avec un octet de donnée dont l'adresse est spécifiée par l'addition d'un déplacement d à une adresse de base détenue dans un registre index.

Le code opératoire est de deux octets. Il est suivi de l'octet indiquant le déplacement traité en complément à deux.

Le diagramme ci-dessous, valable également pour IX montre comment les différentes valeurs (de — 128 à + 127) du déplacement sont utilisées pour former l'adresse d'entrée dans une table.

	IY + d	déplacement en décimal	
	IY + 7F	+ 127	haut
	IY + 0F	+ 15	
	IY + 09	+ 9	Table
	IY + 01	+ 1	liste ou
IY détient ⟶	IY + 00	0	suite de
cette adresse	IY + FF	— 1	code
	IY + FC	— 4	
	IY + 60	— 32	
	IY + 80	— 128	bas

mnémonique	code hexa	
LD A, (IX + d)	DD 73 XX	LET A = PEEK (IX + d)
LD H, (IX + d)	DD 66 XX	LET H = PEEK (IX + d)
LD L, (IX + d)	DD 6E XX	LET L = PEEK (IX + d)
LD B, (IX + d)	DD 46 XX	LET B = PEEK (IX + d)
LD C, (IX + d)	DD 4E XX	LET C = PEEK (IX + d)
LD D, (IX + d)	DD 56 XX	LET D = PEEK (IX + d)
LD E, (IX + d)	DD 5E XX	LET E = PEEK (IX + d)

Pour les instructions concernant IY changer DD en FD.

Il est important de noter que l'utilisation de ces instructions ne change pas le contenu du registre index.

N'oubliez pas les particularités du système ZX 81 en ce qui concerne les registres index.

IX ne peut pas être utilisé en mode SLOW.

IY contient 4000 h et le système d'exploitation s'en sert pour pointer ses variables (IY + d)

APPLICATION :

L'exercice suivant consiste à «récupérer» un octet de donnée dans la zone des variables système grâce à l'adresse de base 4000 h inscrite dans IY par le programme moniteur.

Le PCCM étant chargé, faites POKE 16417, n (n compris entre 0 et 255).

16417 est une adresse inutilisée dans la zone des variables système (voir manuel ZX 81, page 178).

16417 décimal = 4021 hexa.

d = 4021 — 4000 = 21 h.

d'où :

LD C, (IY + 21)	FD 4E 21
LD B, 00	06 00
RET	C9

CHARGEMENT D'EMPLACEMENTS MÉMOIRE AVEC DES DONNÉES PROVENANT DE REGISTRES OU AVEC DES CONSTANTES.

Les opérations réalisées par les instructions qui suivent sont généralement l'inverse de celles décrites précédemment. Au lieu de «lecture» il s'agit cette fois «d'écriture» dans une case mémoire. Le résultat est tout à fait analogue à celui de la fonction POKE du Basic. Bien entendu, on ne peut «écrire» qu'en mémoire vive.

On peut aussi classer ces instructions en trois catégories suivant leur mode d'adressage.

1. - INSTRUCTIONS UTILISANT L'ADRESSAGE ABSOLU.

Ces instructions sont de 3 ou 4 octets. Elles permettent de copier le contenu d'un registre dans un emplacement mémoire nommément désigné. L'adresse sur 2 octets suit le code opératoire.

mnémonique	code hexa	
LD (ADR), A	32 XX XX	POKE ADR, A
LD (ADR), HL	22 XX XX	POKE ADR, L
		POKE ADR + 1, H
LD (ADR), BC	ED 43 XX XX	POKE ADR, C
		POKE ADR + 1, B
LD (ADR), DE	ED 53 XX XX	POKE ADR, E
		POKE ADR + 1, D
LD (ADR), IX	DD 22 XX XX	POKE ADR, IX — INT (IX/256) * 256
		POKE ADR + 1, INT (IX/256)
LD (ADR), IY	FD 22 XX XX	POKE ADR, IY — INT (IY/256) * 256
		POKE ADR + 1, INT (IY/256)
LD (ADR), SP	ED 73 XX XX	POKE ADR, SP — INT (SP/256) * 256
		POKE ADR + 1, INT (SP/256)

Une autre forme de LD (ADR), HL est ED 63 XX XX.

Une fois encore le seul registre simple concerné est l'accumulateur (A). Seule la première instruction de la liste ci-dessus permet le transfert d'un seul octet d'un registre simple dans une case mémoire.

Toutes les autres instructions impliquent le transfert de deux octets de données contenus dans un registre double.

C'est le contenu du registre faible qui est copié à l'adresse spécifiée, le contenu du registre fort étant copié à l'adresse suivante.

Par exemple LD (ADR), HL signifie
LD (ADR), L et LD (ADR + 1), H comme le montre l'analogie avec le Basic.

Il n'existe pas d'instruction pour écrire directement une constante dans un emplacement mémoire nommément désigné.

L'équivalent de POKE ADR, n n'existe pas en langage machine, il faut passer par un registre.

APPLICATION :

Faites les exercices suivants :

Choisissez n entre 0 et 255 d
nn entre 0 et 65535 d
ADR entre 20000 et 29999 d (une seule adresse).

```
LD A, n
LD (ADR), A
LD BC, (ADR)
LD B, 00
RET
```

```
LD HL, nn
LD (ADR), HL
LD BC, (ADR)
RET
```

```
LD DE, nn
PUSH DE
POP HL
LD (ADR), HL
LD DE, (ADR)
EX DE, HL
PUSH HL
POP BC
RET
```

```
LD BC, nn
LD A, B
PUSH AF
POP HL
LD (ADR), HL
LD DE, (ADR)
EX DE, HL
PUSH HL
POP AF
LD B, A
RET
```

2. - INSTRUCTIONS UTILISANT L'ADRESSAGE INDIRECT REGISTRE.

Ces instructions permettent de copier le contenu du registre A dans un emplacement mémoire pointé par les registres doubles HL, BC ou DE. Le contenu des autres registres simples peut être copié dans un emplacement mémoire pointé par HL uniquement.

Une constante sur un octet peut aussi être inscrite dans un emplacement mémoire pointé par HL.

Il paraît judicieux de classer ici trois instructions plutôt spécialisées qui permettent l'échange du contenu des registres doubles HL, IX ou IY avec les deux derniers octets de données de la pile (SP).

mnémonique	code hexa
LD (HL), A	77
LD (BC), A	02
LD (DE), A	12
LD (HL), H	74
LD (HL), L	75
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), n	36 XX
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3

Ces instructions sont souvent utilisées pour ranger en mémoire des valeurs résultant d'une manipulation.

Les lignes suivantes extraites de la routine de changement des pointeurs vue précédemment montrent comment, après manipulation, les nouvelles valeurs des pointeurs sont rangées en mémoire.

adresse	mnémonique	code hexa
2499 - 09C3	LD (HL), D	72
09C4	DEC HL	2B
09C5	LD (HL), E	73

Le contenu du registre fort est rangé à l'adresse pointée par HL - HL est diminué de 1. Ainsi le contenu du registre faible peut être rangé à l'adresse immédiatement inférieure.

APPLICATION :

```
LD HL, nn          LD HL, 5A00
EX (SP), HL        LD DE, nn
POP BC             LD (HL), E
PUSH HL            LD HL, 5A01
RET                LD (HL), D
                   LD BC, (5A00)
                   RET
```

```
LD E, n
LD HL, nn      choisissez nn entre 20000 et 29999 d. C'est une valeur quelconque
LD (HL), E     qui devient une adresse «sensible» à la 3e ligne.
LD C, (HL)
LD B, 00
RET
```

Choisissez ADR entre 20000 et 29999 d

```
LD (ADR), IY        EX (SP), IY
LD HL, (ADR)        POP DE
EX (SP), HL         PUSH IY
LD D, H             LD B, D
LD E, L             LD C, E
POP BC              RET
PUSH DE
RET
```

A quoi correspond
la valeur affichée ?

Remarque : dans les exercices proposés, ne vous contentez pas de faire de l'assemblage. Saisissez bien le mécanisme de ces petits programmes, même s'ils sont farfelus. Ce n'est pas plus idiot que de faire des gammes au piano.

3. - INSTRUCTIONS UTILISANT L'ADRESSAGE INDEXÉ.

Ces instructions permettent de copier le contenu d'un registre simple dans un emplacement mémoire indexé par IX ou IY ou d'y inscrire une constante.

mnémonique	code hexa
LD (IX + d), A	DD 77 XX
LD (IX + d), H	DD 74 XX
LD (IX + d), L	DD 75 XX
LD (IX + d), B	DD 70 XX
LD (IX + d), C	DD 71 XX
LD (IX + d), D	DD 72 XX
LD (IX + d), E	DD 73 XX
LD (IX + d), n	DD 36 XX XX (d) (n)

Pour les instructions concernant IY changer DD en FD.

L'exemple suivant tiré du programme moniteur montre comment une constante peut être chargée dans un emplacement indexé à condition que le registre index contienne l'adresse de base et que le déplacement soit spécifié.

adresse	code hexa	mnémonique
1020 - 03FC	FD 36 3B 40	LD (403B), 40

Cette ligne de la routine d'initialisation permet l'entrée en mode «SLOW».

IY contient l'adresse 4000 h (chargée en 03F8)

IY + d = 4000 + 3B = 403B adresse de la variable système CD - FLAG - (16443) - qui est chargée avec 40 hexa (01000000)

Le bit 6 étant à 1 le système est en mode SLOW jusqu'à ce qu'une modification soit faite par le programmeur.

Notez une erreur dans le manuel ZX 81, page 179, il faut lire BIT 6 et non BIT 7.

APPLICATION :

```
LD (IY + 21 h), n
LD C, (IY + 21 h)
LD B, 00
RET
```

```
LD HL, 16417 d
LD (HL), n
LD C, (IY + 21 h)
LD B, 00
RET
```

```
LD IY, 5000 h
LD (IY + d), n
LD C, (IY + d)
LD B, 00
RET
```

```
LD HL, 5000 h
PUSH HL
EX (SP), IY
POP DE
LD (IY + d), n
LD E, (IY + d)
LD C, E
LD B, 00
RET
```

Dans les 2 exercices ci-dessus n et d peuvent être choisis entre 0 et 255 décimal sans précaution particulière. N'oubliez pas cependant, d est traité en complément à deux dans l'instruction utilisant l'adressage indexé.

Pour retrouver, affichée, la valeur choisie dans l'exercice ci-dessous il faut que le déplacement soit négatif.

Au besoin, allez dans la table en annexe.

```
LD HL, 4021 h
LD (HL), n
LD IY, 16500 d
LD C, (IY + d)
LD B, 00
RET
```

CHAPITRE VIII

INSTRUCTIONS ARITHMÉTIQUES COMPARAISONS

Les indicateurs d'état jouent un rôle important dans le fonctionnement du microprocesseur. A partir de maintenant nous allons rencontrer des instructions où ils sont impliqués soit que l'opération effectuée les modifie, soit que l'effet de l'instruction dépende de la position d'un indicateur particulier.

LES INDICATEURS D'ÉTAT.

Le registre F ou registre d'état est différent des autres en ce sens que six de ses huit bits sont utilisés comme indicateurs (en anglais : Flag = drapeau) pour signaler un état particulier.

La fonction de ces bits d'état est analogue à celle des lampes vertes ou rouges d'un tableau de contrôle. Ils peuvent prendre la valeur 0 ou 1.

Désignés C, N, P/V, H, Z et S ils occupent une position précise dans le registre F.

7	6	5	4	3	2	1	0
S	Z	—	H	—	P/V	N	C
signe	zéro		demi retenue		parité/ débordement	soustraction	retenue

Les bits n° 3 et n° 5 ne sont pas utilisés. Ils prennent aléatoirement la valeur 0 ou 1, non significative.

Les quatre indicateurs principaux C, P/V, Z et S doivent être connus du programmeur car ils sont testés par les instructions conditionnelles.

L'INDICATEUR DE RETENUE.

C'est le bit n° 0 du registre d'état. Désigné C (Carry Flag) il a un rôle essentiel dans les opérations arithmétiques.

L'arithmétique binaire absolue opère sur des nombres d'un format de 8 ou 16 bits, c'est-à-dire ayant une valeur décimale comprise entre 0 et 255 ou entre 0 et 65535.

Il est nécessaire de garder trace de la retenue ou de l'emprunt générés éventuellement par une opération. C'est le rôle de l'indicateur de retenue «C».

Exemple : si on additionne en binaire sur 8 bits les valeurs 128 et 153 le résultat ne «tient» plus sur 8 bits. L'accumulateur contient 25 d. Il y a une retenue signalée par «C» qui est positionné à un. Le résultat réel est 25 + 256.

Le bit de retenue peut être considéré comme un 9e bit pour l'accumulateur.

La soustraction binaire 200 — 201 donne 255 dans l'accumulateur. On a dépassé la valeur zéro. Le résultat a été augmenté de 256. L'«emprunt» est signalé par l'indicateur C qui est positionné à un.

Le même phénomène se produit avec les opérations sur 16 bits.

Exemple : si le contenu du registre double HL est 65535 et que l'on y ajoute un, le contenu de HL devient zéro et le bit de retenue C est positionné à un.

La retenue interne générée par le calcul passe de l'octet de poids faible (dans L) à l'octet de poids fort (dans H) et «sort» de ce dernier pour positionner «C».

Toutes les instructions arithmétiques sauf INC et DEC ainsi que les instructions de comparaison CP affectent l'indicateur de retenue.

Cet indicateur est aussi positionné par les opérations de rotation ou de décalage dans lesquelles un bit sort d'un registre et devient le bit de retenue, ainsi que par DAA et NEG.

Deux instructions permettent de forcer l'indicateur de retenue.

SCF Set Carry Flag - «C» est mis à un.

CCF Complement Carry Flag - Sa valeur est inversée.

L'INDICATEUR DE PARITÉ/DÉBORDEMENT.

C'est le bit n° 2 du registre F désigné P/V. Comme son nom le laisse supposer cet indicateur a une double fonction.

Certaines instructions l'utilisent pour indiquer le résultat d'un test de parité P, d'autres pour signaler un débordement V. Il ne peut pas y avoir confusion car ces opérations ont rarement lieu en même temps.

Il y a parité lorsque, dans l'octet résultant d'une opération, le nombre de bits à un est pair, exemple : 10101010. L'indicateur est alors mis à un. Il est mis à zéro lorsque le nombre de bits à un est impair, exemple 00101010.

Les instructions suivantes positionnent cet indicateur : AND, OR, XOR, RL, RLC, RR, RRC, SLA, SRA, SRL, RLD, RRD, DAA et In r, (C).

Le concept de débordement (ou dépassement de capacité) s'applique aux nombres interprétés en complément à deux, dans lesquels le bit le plus à gauche, le plus significatif, est le bit de signe.

Il y a débordement lorsque le résultat d'une opération est erroné par suite du changement accidentel du bit de signe.

Rappelons que toute soustraction peut être ramenée à une addition. Il suffit d'ajouter au premier nombre le complément à deux du nombre à soustraire.

D'où la règle générale :

Le débordement existe quand la somme de deux nombres positifs est négative ou quand la somme de deux nombres négatifs est positive.

L'indicateur est positionné par les instructions arithmétiques (sauf ADD HL, ADD IY et ADD IX, INC et DEC s'appliquant à un registre double) et par les instructions CP, NEG, DAA.

L'INDICATEUR DE ZÉRO.

Bit n° 6 du registre F, il est désigné Z. Il est mis à un si le résultat d'une opération est zéro, il est mis à zéro autrement.

Attention à l'utilisation de cet indicateur d'abord pour la façon dont il intervient.

Z = 1 lorsque le résultat est égal à zéro

Z = 0 lorsque le résultat est différent de zéro

et aussi parce qu'il n'y a pas de règle générale indiquant les instructions qui le positionnent.

Un résumé est donné ci-dessous. Si vous n'êtes pas sûr, reportez-vous à la table en annexe 4.

Z est positionné par :

- les instructions ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR et XOR utilisant un registre **simple**;
- les instructions ADC et SBC utilisant un registre double;
- les instructions de rotation et de décalage;
- les instructions BIT (test d'un bit);
- les instructions LD A, I et LD A, R (seules parmi les instructions LD).

Il n'existe pas d'instructions pour modifier directement l'indicateur Z.

L'INDICATEUR DE SIGNE.

C'est le bit n° 7 du registre F, désigné S. Son but est d'indiquer que le résultat en complément à deux d'une opération est positif ou négatif. Il est en fait une copie du bit le plus significatif du résultat.

S = 1 lorsque le résultat est négatif

S = 0 lorsque le résultat est positif.

Les mêmes remarques que celles concernant le positionnement de Z, énumérées au paragraphe précédent, s'appliquent au positionnement de l'indicateur S, sauf bien sûr en ce qui concerne le test des bits (BIT). (Un bit n'a pas de signe).

L'indicateur S ne peut pas être modifié directement.

• • •

L'indicateur N est mis à un si la dernière opération était une soustraction et H (Half Carry) demi-retenue est mis à un pour signaler une retenue affectant le bit n° 4 ou le bit n° 12 dans le cas d'un registre double.

Ces indicateurs sont de peu d'intérêt pour le programmeur; ils ne sont pas testés par des instructions de branchement conditionnelles. Le Z 80 les réserve à son usage interne pour l'exécution d'instructions relatives à l'arithmétique décimale codée binaire (DCB).

• • •

Dorénavant vous trouverez, pour chaque catégorie d'instructions, un cartouche rappelant l'effet de l'opération sur les indicateurs d'état.

Les symboles utilisés, les mêmes que ceux de la table en annexe 4, sont les suivants :

+ l'indicateur est affecté par l'opération
— l'indicateur n'est pas affecté
0 l'indicateur est mis à zéro
1 l'indicateur est mis à un
? l'indicateur prend une valeur quelconque
X une explication suit.

APPLICATION :

Il est possible d'examiner le contenu du registre d'état F grâce aux instructions PUSH AF et POP BC qui transfèrent F dans C.

```
LD A, 64 d          3E 40
ADD A, n            C6 XX
PUSH AF             F5
POP BC              C1
LD B, 00            06 00
RET                 C9
```

L'instruction ADD A, n signifie ajouter la valeur n au contenu de l'accumulateur, le résultat reste dans A.

Nous donnerons à n les valeurs successives : 63 (3F hexa), 65 (41 h), 140 (8C), 192 (CO) et 200 (C8). Enfin c'est la valeur chargée dans A que nous modifierons en 128 (80 h).

Pour les modifications, utilisez l'option affichage, adresse modif 30003 puis 30001.

Auparavant, nous allons transformer un peu notre PCCM de façon à obtenir la valeur affichée en hexadécimal, à partir duquel il est plus facile de trouver l'équivalent binaire.

```
2000 CLS
2010 PRINT USR S
2020 LET B = USR S
2030 GOSUB 2900
2040 PRINT H$ (3); H$ (4)
2050 STOP
```

Vous supprimerez les 3 lignes 2020 - 2030 et 2040 après l'exercice.

Le tableau ci-dessous donne le détail des opérations et leur effet prévisible sur les principaux indicateurs.

		Registre A avant	n	Registre A après	indicateurs
1	décimal hexa compl. à 2	64 40 + 64	63 3F + 63	127 7F + 147	
2	décimal hexa compl. à 2	64 40 + 64	65 41 + 65	129 81 — 127	débordement signe
3	décimal hexa compl. à 2	64 40 + 64	140 8C — 116	204 CC — 52	 signe
4	décimal hexa compl. à 2	64 40 + 64	192 C0 — 64	0 00 + 0	retenue zéro
5	décimal hexa compl. à 2	64 40 + 64	200 C8 — 56	8 08 + 8	retenue
6	décimal hexa compl. à 2	128 80 — 128	200 C8 — 56	72 48 + 72	retenue débordement

Le contenu de F affiché à l'écran révèle la position des indicateurs.

		Déc.	hexa	S	Z	—	H	—	P/V	N	C
	1.	40	28	0	0	1	0	1	0	0	0
	2.	132	84	1	0	0	0	0	1	0	0
Valeur affichée	3.	136	88	1	0	0	0	1	0	0	0
contenu de F	4.	65	41	0	1	0	0	0	0	0	1
	5.	9	09	0	0	0	0	1	0	0	1
	6.	13	0D	0	0	0	0	1	1	0	1

Remarquez bien dans quelles conditions se produit le débordement.

LES INSTRUCTIONS D'ADDITION.

Les instructions d'addition opèrent en arithmétique binaire absolue sur 8 ou 16 bits. On distingue les instructions ADD, INC et ADC.

ADD

Les instructions ADD permettent :

- sur 8 bits, d'ajouter au contenu de l'accumulateur une valeur spécifiée, ou le contenu d'un registre simple ou d'un emplacement mémoire pointé par HL ou indexé par IX ou IY;
- sur 16 bits, d'ajouter au contenu de HL ou d'un registre index, le contenu d'un autre registre double.

Vous remarquerez qu'il n'existe pas d'instruction ADD de la forme ADD A, (ADR) ni ADD HL, nn

mnémonique	code hexa
ADD A, n	C6 XX
ADD A, A	87
ADD A, H	84
ADD A, L	85
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, (HL)	86
ADD A, (IX + d)	DD 86 XX
ADD A, (IY + d)	FD 86 XX

mnémonique	code hexa
ADD HL, HL	29
ADD HL, BC	09
ADD HL, DE	19
ADD HL, SP	39
ADD IX, IX	DD 29
ADD IX, BC	DD 09
ADD IX, DE	DD 19
ADD IX, SP	DD 39
ADD IY, IY	FD 29
ADD IY, BC	FD 09
ADD IY, DE	FD 19
ADD IY, SP	FD 39

S	Z	N	P/V	N	C
+	+	+	+	0	+

S	Z	H	P/V	N	C
—	—	+	—	0	+

Les quatre indicateurs principaux sont affectés par ADD sur 8 bits alors que sur 16 bits, seul l'indicateur de retenue est concerné.

Un exemple d'utilisation de l'instruction ADD HL, DE tiré du programme moniteur est donné ci-dessous.

D'abord il est nécessaire de dire quelques mots des routines intéressantes de la ROM que sont la routine de scrutation du clavier et la routine de décodage des caractères. Nous les verrons en détail au chapitre XIV.

Quand une touche est enfoncée, une certaine valeur sur 2 octets est chargée dans HL par la routine de scrutation du clavier. Cette valeur transférée dans BC est prise en compte par la routine de décodage qui commence en 1981 d - 07BD h. Elle est manipulée et transformée en une valeur codée comprise entre 1 et 78 d (01 - 4D hexa) qui est chargée dans l'accumulateur.

Cette valeur comprise entre 1 et 78 correspond à la place occupée par un des 78 caractères du clavier dans la table de caractères (de 126 d - 007E à 203 - 00CB).

Par exemple, la touche D donnera la valeur 7 dans l'accumulateur. Le 7e caractère de la table est 29 hexa qui correspond au code ZX 81 pour la lettre D.

Le résultat de la routine de décodage est de fournir l'adresse du caractère concerné dans le registre HL. Ce dernier est d'abord chargé avec l'adresse de base de la table. Le contenu de l'accumulateur est transféré dans E puis ADD HL, DE donne l'adresse cherchée. (D contenant 0 depuis le début de la routine).

adresse	mnémonique

2005 - 07D5	LD HL, 007D
	LD E, A
	ADD HL, DE

INC

Ces instructions ajoutent une unité à un nombre de 8 bits contenu dans un registre simple ou dans un emplacement mémoire désigné par HL ou indexé par IX ou IY ou à un nombre de 16 bits contenu dans un registre double.

L'indicateur C n'est pas affecté. Les autres ne le sont que dans le cas de l'incrémentation d'une valeur sur 8 bits.

mnémonique	code hexa	mnémonique	code hexa
INC A	3C	INC HL	23
INC H	24	INC BC	03
INC L	2C	INC DE	13
INC B	04	INC SP	33
INC C	0C	INC IX	DD 23
INC D	14	INC IY	FD 23
INC E	1C		
INC (HL)	34		
INC (IX + d)	DD 34 XX		
INC (IY + d)	FD 34 XX		

S	Z	H	P/V	N	C
+	+	+	+	0	—

S	Z	H	P/V	N	C
non affectés					

ADC

Les instructions ADC permettent l'addition de nombres sur 8 ou 16 bits prenant en compte la position actuelle de l'indicateur de retenue. Le résultat de l'addition n'est pas changé si «C» est à zéro. Il est incrémenté si «C» est à un.

Les instructions ADC repositionnent l'indicateur de retenue suivant le résultat de l'opération. Toutes affectent les 4 principaux indicateurs.

mnémonique	code hexa	mnémonique	code hexa
ADC A, n	CE XX	ADC HL, HL	ED 6A
ADC A, A	8F	ADC HL, BC	ED 4A
ADC A, H	8C	ADC HL, DE	ED 5A
ADC A, L	8D	ADC HL, SP	ED 7A
ADC A, B	88		
ADC A, C	89		
ADC A, D	8A		
ADC A, E	8B		
ADC A, (HL)	8E		
ADC A, (IX + d)	DD 8E XX		
ADC 1, (IY + d)	FD 8E XX		

S	Z	H	P/V	N	C
+	+	+	+	0	+

S	Z	H	P/V	N	C
+	+	+	+	0	+

Les instructions ADC ne sont pas très fréquentes. Elles peuvent être utilisées en arithmétique multiple précision (portant sur des nombres de 2 octets ou plus) où il est important en additionnant les octets de poids successif d'ajouter la retenue générée par l'addition des octets de poids inférieur.

Exemple : pour additionner les nombres hexadécimaux GH, IJ, KL et MN, OP, QR

il faudra faire ADD KL, QR
ADC IJ, OP
ADC GH, MN

APPLICATION :

```
LD A, n1
ADD A, n2
LD C, A
LD B, 00
RET
```

```
LD BC, nn
INC BC
RET
```

Ecrivez un programme qui permet d'additionner 1000 à 2000 (en décimal) en utilisant les instructions d'addition sur 8 bits. Rangez le résultat dans BC pour l'affichage.

Le programme suivant affiche la valeur d'une variable système. Laquelle ?

```
LD DE, 0004 h
ADD IY, DE
PUSH IY
POP HL
LD C, (HL)
INC HL
LD B, (HL)
RET
```

LES INSTRUCTIONS DE SOUSTRACTION.

Elles comprennent les instructions SUB, DEC et SBC.

SUB

Les instructions SUB ne portent que sur des valeurs 8 bits. Elles permettent de soustraire du contenu de l'accumulateur une constante ou le contenu d'un registre simple ou d'un emplacement mémoire pointé par HL ou indexé par IX ou IY.

Le nom de l'accumulateur est généralement omis dans l'instruction.

SUB n signifie SUB A, n.

mnémonique	code hexa
SUB n	D6 XX
SUB A	97
SUB H	94
SUB L	95
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB (HL)	96
SUB (IX + d)	DD 96 XX
SUB (IY + d)	FD 96 XX

S	Z	H	P/V	N	C
+	+	+	+	1	+

Tous les indicateurs sont affectés. L'indicateur N (soustraction) est mis à un.

Les instructions SUB sont fréquemment utilisées.

DEC

Les instructions DEC sont symétriques des instructions INC. Elles ont pour effet de soustraire un d'une valeur contenue dans un registre (simple ou double) ou un emplacement mémoire pointé par HL ou indexé par IX ou IY. Les indicateurs sont affectés de la même façon que par INC. L'indicateur N est mis à un au lieu de 0.

mnémonique	code hexa	mnémonique	code hexa
DEC A	3D	DEC HL	2B
DEC H	25	DEC BC	0B
DEC L	2D	DEC DE	1B
DEC B	05	DEC SP	3B
DEC C	0D	DEC IX	DD 2B
DEC D	15	DEC IY	FD 2B
DEC E	1D		
DEC (HL)	35		
DEC (IX + d)	DD 35 XX		
DEC (IY + d)	FD 35 XX		

S	Z	H	P/V	N	C
+	+	+	+	1	–

S	Z	H	P/V	N	C
non affectés					

Les instructions DEC (et INC) sont souvent employées avec les registres servant de compteurs ou de pointeurs d'adresses.

Les lignes ci-dessous sont extraites de la routine d'initialisation du système. Elles montrent comment, HL étant chargé avec la variable système RAMTOP, DEC HL est utilisée plusieurs fois de façon à charger la valeur 3E dans le premier emplacement puis à initialiser le registre pointeur de pile SP et la variable système ERR - SP.

adresse	mnémonique	code hexa
197 - 03E5	LD HL, (4004)	2A 04 40
	DEC HL	2B
	LD (HL), 3E	36 3E
	DEC HL	2B
	LD SP, HL	F9
	DEC HL	2B
	DEC HL	2B
	LD (4002), HL	22 02 40

La valeur 3E sert de marqueur. Elle indique le «fond» de la pile de GOSUB et permet à l'interpréteur Basic de détecter l'absence de numéro de ligne de retour et de lancer l'erreur 7 «RETURN WITHOUT GOSUB».

SBC

Les instructions SBC opèrent sur des valeurs de 8 bits ou de 16 bits.

La valeur de l'indicateur de retenue est ajoutée au nombre à soustraire.

Les mêmes registres sont concernés que pour l'instruction ADC. Les indicateurs sont aussi affectés. N est mis à un (soustraction).

Vous remarquerez que l'accumulateur est spécifié dans le mnémonique de l'instruction SBC sur 8 bits, SBC A, n.

mnémonique	code hexa	mnémonique	code hexa
SBC A, n	DE XX	SBC HL, HL	ED 62
SBC A, A	9F	SBC HL, BC	ED 42
SBC A, H	9C	SBC HL, DE	ED 52
SBC A, L	9D	SBC HL, SP	ED 72
SBC A, B	98		
SBC A, C	99		
SBC A, D	9A		
SBC A, E	9B		
SBC A, (HL)	9E		
SBC A, (IX + d)	DD 9E XX		
SBC A, (IY + d)	FD 9E XX		

S	Z	H	P/V	N	C
+	+	+	+	1	+

S	Z	H	P/V	N	C
+	+	+	+	1	+

Il n'existe pas d'instruction SUB pour effectuer une simple soustraction sur 16 bits.

C'est SBC qui est utilisée mais il faut prendre la précaution de forcer l'indicateur de retenue à zéro avant l'opération.

L'instruction AND A (chapitre IX) est habituellement utilisée pour cela.

Contrairement à ADC, les instructions SBC sont couramment employées, notamment pour comparer des nombres sur 16 bits ou des adresses comme le montre l'exemple suivant extrait du programme moniteur.

La routine d'affichage d'une ligne de Basic (en 1861 d - 0745) décode les «jetons» (entrés avec une seule touche) et affiche le mot complet.

Exemple : le code caractère 245 est converti en PRINT pour l'affichage.

Cette même routine assure l'affichage éventuel du curseur d'erreur de syntaxe (S inversé).

Quand une erreur de syntaxe est rencontrée l'adresse correspondante est stockée en 16408 - 16409 - variable système X.PTR

Avant l'affichage d'un caractère, X.PTR est comparée avec CH.ADD adresse du caractère suivant à interpréter, variable système en 16406 - 4016 h.

SBC HL, BC précédée de AND A, (mise à zéro de C) est utilisée pour la comparaison.

Si le résultat est zéro (mêmes adresses) le S inversé (B8) est chargé dans A et envoyé à l'affichage (RST 10) pour signaler l'erreur de syntaxe.

Sinon, JR NZ envoie à l'adresse 077C.

1901 - 076D	LD BC, (4018)	ED 4B 18 40	X.PTR
	LD HL, (4016)	2A 16 40	CH.ADD
	AND A	A7	C = 0
	SBC HL, BC	ED 42	comparaison
	JR NZ, 077C	20 03	OK
	LD A, B8	3E B8	S inversé
	RST 10	D7	affiché
077C	...		

APPLICATION :

Ajoutez la ligne suivante au PCCM. Vous la supprimerez après les exercices

2015 PRINT "RETENUE ="; PEEK 16417

```
LD A, n1
SUB n2          choisissez différentes
LD C, A         valeurs pour n1 et n2
LD B, 00        (entre 0 et 255)
LD A, 00        option affichage pour
ADC A, 00       les modifier - Adresses
LD (4021 h), A  30001 et 30003
RET
```

Remarque : Après le transfert dans le registre C le contenu de A est annulé (LD A, 00) ADC A, 00 (addition avec retenue) permet de recueillir la valeur de l'indicateur C qui est «pokée» en 16417.

Mêmes exercices avec des nombres sur 16 bits. Choisissez nn1 et nn2 entre 0 et 65535 d.

```
LD HL, nn1
LD DE, nn2
ADD HL, DE
LD B, H
LD C, L
LD A, 00
ADC A, 00
LD (4021 h), A
RET
```

```
LD HL, nn1
LD DE, nn2
AND A (code hexa = A7)
SBC HL,DE
LD B, H
LD C, L
LD A, 00
ADC A, 00
LD (4021 h), A
RET
```

LES INSTRUCTIONS DE COMPARAISON.

Malgré leur nombre restreint ces instructions sont parmi les plus utilisées.

Elles permettent de comparer le contenu du registre A avec une constante sur un octet ou avec le contenu d'un registre simple ou d'un emplacement mémoire (pointé par HL ou indexé par IX ou IY).

Nous avons déjà les instructions de soustraction qui permettent de faire des comparaisons. Il suffit d'interpréter la position des indicateurs d'état Z et C. En effet, si on fait A—B

si A = B	Z = 1	C = 0
si A > B	Z = 0	C = 0
si A < B	Z = 0	C = 1

Les instructions CP opèrent en fait une soustraction, exactement comme les instructions SUB avec cette différence que le résultat n'est pas conservé. Le contenu de l'accumulateur reste inchangé. Le seul effet de CP est de positionner les indicateurs.

C'est intéressant, car on peut faire des comparaisons successives :

```
LD A, n0
CP n1
si Z = 1 aller à Paris
CP n2
si Z = 1 aller à Marseille
...
```

mnémonique	code hexa
CP n	FE XX
CP A	BF
CP H	BC
CP L	BD
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP (HL)	BE
CP (IX + d)	DD BE XX
CP (IY + d)	FD BE XX

S	Z	H	P/V	N	C
+	+	+	+	1	+

CP est intégrée dans les instructions de recherche par blocs que nous verrons au chapitre X.

On trouve d'innombrables comparaisons dans le programme moniteur. Ainsi, au cours de la routine d'initialisation, CP H compare le contenu de l'accumulateur à celui du registre H, octet de poids fort d'une adresse.

Le passage concerné est situé tout au début de la routine. Il mérite d'être reproduit en entier puis commenté à titre de révision et d'introduction aux instructions conditionnelles de branchement. Vous pouvez passer au chapitre suivant si vous le désirez.

Après la mise sous tension de l'ordinateur, le registre BC est chargé avec 7FFF - 32767 d adresse maximum possible du système, puis un saut est effectué en 03CB début de l'initialisation, première routine à être exécutée ce qui n'a rien de surprenant.

Les lignes qui suivent ont pour objet de déterminer la quantité de mémoire vive disponible et de fixer la valeur de la variable système RAMTOP.

```
        0000    D3 FD       OUT (FD), A
        0002    01 FF 7F    LD BC, 7FFF
        0005    C3 CB 03    JP 03CB
        ...
971 -   03CB    60          LD H, B
        03CC    69          LD L, C
        03CD    3E 3F       LD A, 3F
        03CF    36 02       LD (HL), 02
        03D1    2B          DEC HL
        03D2    BC          CP H
        03D3    20 FA       JR NZ, 03CF
        03D5    A7          AND A
        03D6    ED 42       SBC HL BC
        03D8    09          ADD HL, BC
        03D9    23          INC HL
        03DA    30 06       JR NC, 03E2
        03DC    35          DEC (HL)
        03DD    28 03       JR Z, 03E2
        03DF    35          DEC (HL)
        03E0    28 F3       JR Z, 03D5
        03E2    22 04 40    LD (4004), HL
```

En 03CB le contenu de BC, adresse maximum possible, est transféré dans HL. En 03CD, A est chargé avec la valeur 3F.

De 03CF à 03D4 il y a tentative de chargement de tous les emplacements mémoire entre 07FF et 4000. Evidemment ce chargement ne sera effectif que si l'emplacement mémoire existe. Tous les emplacements seront chargés si l'extension 16 K est installée, sinon le chargement ne commencera qu'à l'adresse 17407 (43FF). Le chargement se poursuit jusqu'à l'adresse 4000 incluse (16384) grâce à l'instruction de saut conditionnel en 03D3.

Le retour en 03CF est assuré tant que l'indicateur de zéro n'est pas mis à un (NZ) c'est-à-dire tant que la comparaison entre l'accumulateur et H ne le positionne pas à un. Cela est réalisé dès que HL passe en-dessous de 4000 (à 3FFF). L'octet de poids fort de l'adresse (3F) est alors égal au contenu de l'accumulateur.

Après le chargement de 02 en mémoire vive la routine continue par la lecture dans l'ordre des adresses croissantes à partir de 4000 h de tous les emplacements mémoire. L'adresse du premier emplacement qui ne contient pas 02 devient RAMTOP.

L'opération consiste en une boucle (de 03D5 à 03EO) dont les éléments principaux sont en 03D9. INC HL en 03DC et 03DF DEC (HL) et bien sûr l'instruction de saut en 03EO qui signifie : recommencer si zéro (si l'indicateur Z est mis à un).

Rappelez-vous, l'adresse contenue dans HL était 3FFF à la fin de la boucle précédente INC HL la porte à 4000. Le contenu de cette adresse est décrémenté deux fois DEC (HL) s'il valait 02 il vaut maintenant 0. L'indicateur Z est positionné à 1 et on recommence à l'adresse suivante.

Les autres instructions permettent de sortir de la boucle. Un saut conditionnel envoie en 03E2 ou sera inscrite RAMTOP (la valeur actuelle de HL).

La première sortie peut avoir lieu en 03DA si l'indicateur de retenue est à zéro.

JR NC signifie Jump Relative if Not Carry - saut relatif si pas de retenue. Par quelle instruction, cet indicateur a-t-il été positionné ? Ce n'est pas par INC HL. Vous savez que l'incrémentation est sans effet sur l'indicateur de retenue, d'ailleurs quand il s'agit d'un registre double, aucun des indicateurs n'est affecté. C'est donc ADD HL, BC qui positionne l'indicateur C. Oui, mais pas toute seule. En réalité, vous vous doutez bien qu'il s'agit d'une comparaison entre HL et BC et que l'instruction déterminante est SBC HL, BC.

Les trois instructions AND A - SBC HL, BC - ADD HL, BC laissent la valeur de HL inchangée elles ont simplement pour but de positionner l'indicateur de retenue pour l'instruction conditionnelle JR NC. Voyons comment; au commencement, HL = 3FFF et BC = 7FFF (après un départ normal) donc nous avons HL < BC. L'indicateur C initialisé à zéro par AND A, est mis à un par l'instruction SBC HL, BC qui provoque un «emprunt». Le contenu de HL devient HL—BC + 65536 soit au tout début de la routine :

$$16383 - 32767 + 65536 = 49152$$

ADD HL, BC donne 49152 + 32767 = 81919

Le registre HL contient 16383 (81919 — 65536), sa valeur initiale, et il y a une retenue.

On ne sort pas de la boucle tant que l'indicateur C est à un, c'est-à-dire tant que HL reste inférieur à BC sauf, bien sûr, si le test des emplacements mémoire révèle une valeur différente de 02 (capacité de mémoire vive inférieure à 16 K).

Dès que HL = BC, HL + 1 devient RAMTOP et la routine se poursuit avec l'initialisation de ERR SP du registre pointeur de pile SP, et d'autres variables système.

La commande Basic NEW provoque l'exécution de cette routine qui débute alors à l'adresse 03C3. La valeur actuelle de RAMTOP est chargée dans BC, diminuée de 1, c'est cette valeur qui représente l'adresse maximum disponible.

Si après la mise en route du système, RAMTOP est modifiée par le programmeur, NEW aura pour effet non seulement de vider la mémoire, mais de procéder à une réinitialisation des variables système qui la régissent.

Un exemple concret montrera la nécessité de faire NEW pour «officialiser» la modification de RAMTOP (voir page 168 de votre manuel ZX 81). Attention cet essai va vous faire perdre votre programme.

Faites POKE 16388, 48 POKE 16389, 117

Nouvelle valeur de RAMTOP = 30000 d.

Faites PRINT PEEK 16386 + (256 * PEEK 16387)

Résultat ERR-SP = 32764

Maintenant faites NEW, puis à nouveau

PRINT PEEK 16386 + (256 * PEEK 16387)

Résultat ERR-SP = 29994.

CHAPITRE IX

INSTRUCTIONS LOGIQUES

Les fonctions logiques expriment une relation entre des variables susceptibles de prendre une des deux valeurs 0 ou 1 (état logique).

Ces fonctions suivent les règles de l'algèbre de BOOLE, du nom du mathématicien anglais du 19e siècle, Georges BOOLE, dont les théories ont été redécouvertes pour leurs applications à la technique électronique digitale.

Les opérateurs logiques, ou booléens, sont : AND, OR, XOR et NOT.

- AND, ET, produit logique symbole•

  Le produit logique de variables booléennes est égal à leur produit au sens habituel du terme. Ainsi le produit logique de A et B est égal à un, si A et B sont tous deux égaux à un. La table de vérité est la suivante :

  0 **AND** 0 = 0
  0 **AND** 1 = 0
  1 **AND** 0 = 0
  1 **AND** 1 = 1

- OR, OU, somme logique symbole +

  La somme logique de A et B est égale à un si A ou B est égal à un ou si les deux sont égaux à un conformément à la table de vérité :

  0 **OR** 0 = 0
  0 **OR** 1 = 1
  1 **OR** 0 = 1
  1 **OR** 1 = 1

- XOR, OU, exclusif symbole ⊕

  Le «OU» exclusif diffère du «OU» précédent par un aspect.

  La somme logique exclusive de A et B n'est égale à un que si A ou B est égal à un. Le résultat est zéro si A et B sont égaux à un. La table de vérité est :

  0 **XOR** 0 = 0
  0 **XOR** 1 = 1
  1 **XOR** 0 = 1
  1 **XOR** 1 = 0

- NOT, NON, complémentation logique symbole $\overline{A}$

  $\overline{0} = 1$
  $\overline{1} = 0$

Les instructions AND, OR et XOR du Z 80 permettent d'effectuer des opérations logiques sur 8 bits.

Ces opérations sont effectuées bit à bit. Chaque bit de l'octet contenu dans l'accumulateur est combiné avec le bit correspondant d'un octet spécifié pour former un résultat logique.

L'instruction CPL effectue la complémentation logique (NOT) de l'accumulateur.

AND

Une instruction AND permet d'effectuer l'opération ET logique entre les différents bits du contenu de l'accumulateur et les bits correspondants d'un octet spécifié.

Le résultat de ces huit opérations est conservé dans l'accumulateur.

Exemple :

contenu de A	11011000	D8
	AND	**AND**
octet spécifié	00001111	0F
résultat	00001000	08

Une application de cette instruction est le masquage.

Le masquage permet de conserver certains bits dans l'accumulateur (ou d'annuler les autres selon le résultat recherché).

Dans l'exemple ci-dessus le masque 00001111 permet de conserver les bits 0 à 3, il annule les bits 4 à 7.

L'instruction AND affecte les trois indicateurs P/V, Z et S. Elle met à zéro l'indicateur de retenue. C'est pourquoi, AND A, qui laisse le contenu de l'accumulateur intact, est souvent employée à deux fins différentes :

- pour annuler l'indicateur C avant une opération;
- pour tester à zéro l'accumulateur (CP 00 fait 2 octets) si A = 0 Z = 1

mnémonique	code hexa
AND n	E6 XX
AND A	A7
AND H	A4
AND L	A5
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND (HL)	A6
AND (IX + d)	DD A6 XX
AND (IY + d)	FD A6 XX

S	Z	H	P/V	N	C
+	+	1	+	0	0

Vous remarquerez que le nom de l'accumulateur est omis dans le mnémonique. Le deuxième octet est spécifié de la façon classique déjà rencontrée avec les instructions arithmétiques et de comparaison sur 8 bits.

L'extrait du programme moniteur qui va suivre concerne l'affichage d'un mot-clé. Il montre l'utilisation de AND n pour le masquage de deux bits d'un octet.

Il existe quatre tables des caractères du clavier entre les adresses 007E et 01FB (126 à 507 d).

La quatrième table en 0111, est celle des mots-clés (mode K).

Les mots-clés y figurent en entier, leur dernière lettre étant inversée.

Vous pouvez consulter les quatre tables avec le programme Basic.

```
10  FOR A = 126 TO 507
20  IF A = 204 OR A = 243 OR A = 273
    THEN PRINT,,,
30  PRINT CHR$ PEEK A;
40  NEXT A
50  STOP
```

La commande PRINT par exemple est à l'adresse 01C3 hexa.

451-01C3	code hexa 35	lettre P
452-01C4	code hexa 37	lettre R
453-01C5	code hexa 2E	lettre I
454-01C6	code hexa 33	lettre N
455-01C7	code hexa B9	lettre T

Pour l'affichage de chaque lettre l'accumulateur est chargé avec le code caractère dont l'adresse est pointée par BC.

L'instruction AND 3F forme un masque qui rend «normale» une lettre inversée.

Exemple : pour la lettre T

T inversé	code caractère	10111001	B9
		AND	**AND**
masque	3F	00111111	3F
T normal		00111001	39

Chaque lettre est ainsi envoyée à l'affichage en caractère normal par RST 10.

Un test est effectué ensuite pour déterminer s'il s'agissait de la dernière lettre du mot. Le code initial est rechargé dans l'accumulateur et multiplié par 2 par ADD A, A.

Si une retenue est générée il s'agissait de la dernière lettre, sinon on passe à la lettre suivante.

Voici l'extrait en question :

adresse	code hexa	mnémonique	
2393 - 0959 h	0A	LD A, (BC)	code caractère en A
	E6 3F	AND 3F	masquage bit 6 et 7
	D7	RST 10	affichage
	0A	LD A (BC)	code caractère en A
	03	INC BC	déplacement pointeur
	87	ADD A, A	multiplié par 2
	30 F7	JR NC, 0959	suivant si pas de retenue

OR

Une instruction OR permet d'effectuer l'opération OU logique entre les différents bits du contenu de l'accumulateur et les bits correspondants d'un octet spécifié.

Le résultat de ces huit opérations est conservé dans l'accumulateur.

Alors que l'effet d'une instruction AND était d'annuler certains bits, celui d'une instruction OR est d'en mettre certains à un.

Exemple :

contenu de A	11011000	D8
	OR	**OR**
octet spécifié	00001111	0F
résultat	11011111	DF

mnémonique	code hexa
OR n	F6 XX
OR A	B7
OR H	B4
OR L	B5
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR (HL)	B6
OR (IX + d)	DD B6 XX
OR (IY + d)	FD B6 XX

S	Z	N	P/V	N	C
+	+	0	+	0	0

Les instructions OR ne sont pas souvent employées. Elles ne sont pas indispensables.

Elles peuvent cependant être utiles dans certains cas. Par exemple pour déterminer si le contenu d'un registre double est nul.

Après les deux instructions suivantes : LD A, D - OR E, l'indicateur de zéro sera mis à un si, et seulement si, D et E sont nuls.

Si un registre double est utilisé comme compteur dans une boucle c'est généralement un bon moyen de détecter l'annulation de son contenu, compteur à zéro, condition de fin de boucle, car les instructions INC ou DEC sur 16 bits n'affectent aucun indicateur.

XOR

Une instruction XOR permet d'effectuer l'opération OU EXCLUSIF logique entre les différents bits du contenu de l'accumulateur et les bits correspondants d'un octet spécifié.

Le résultat de ces huit opérations est conservé dans l'accumulateur.

XOR n	EE XX
XOR A	AF
XOR H	AC
XOR L	AD
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR (HL)	AE
XOR (IX + d)	DD AE XX
XOR (IY + d)	FD AE XX

S	Z	H	P/V	N	C
+	+	0	+	0	0

Les instructions XOR sont rarement employées. On peut s'en passer complètement.

XOR A peut être utilisée pour annuler le contenu de l'accumulateur avec une instruction sur un octet au lieu de deux avec LD A, 00. XOR A a cependant pour effet de mettre l'indicateur de retenue à zéro, ce qui n'est pas toujours souhaitable.

L'extrait du programme moniteur ci-dessous montre comment une variable Basic «chaîne de caractères» est marquée d'une façon spéciale grâce à l'instruction XOR.

Une variable chaîne de caractères est reconnue comme telle si les 3 bits les plus significatifs de l'octet qui la désigne ont les valeurs suivantes :

Bit n° 5 = 0 Bit n° 6 = 1 Bit n° 7 = 0. (page 174, manuel SINCLAIR).

Le positionnement de ces 3 bits est réalisé par l'opération OU EXCLUSIF entre le code caractère de la lettre désignant la variable et la valeur 60 hexa. Le code caractère est trouvé à l'adresse contenue dans la variable système DEST - en 16402-16403 - adresse de la variable en affectation.

Exemple :

lettre A	00100110	26
	XOR	**XOR**
60	01100000	60
variable «marquée»	01000110	46

adresse	code hexa	mnémonique	
5064 - 13C8	3E 60	LD A, 60	
	2A 12 40	LD HL, (4012)	DEST
	AE	XOR (HL)	

Sauf CPL complémentation logique de l'accumulateur les instructions qui suivent ne sont pas des instructions logiques à proprement parler. Elles ont été classées dans ce groupe par commodité.

CPL

L'instruction effectue la complémentation logique du contenu de l'accumulateur, opération NOT. Tous les bits sont inversés.

Le code hexa est 2F.

Les indicateurs principaux ne sont pas affectés. H et N sont mis à un.

S	Z	H	P/V	N	C
—	—	1	—	1	—

Rappelons que le complément à deux d'un nombre est obtenu en complémentant ce nombre et en ajoutant 1 au résultat.

Le complément à deux d'un nombre sur 8 bits peut être obtenu directement par l'instruction NEG portant sur le contenu de A.

NEG négation de l'accumulateur, code hexa ED 44.

Cette instruction affecte tous les indicateurs.

S	Z	N	P/V	N	C
+	+	+	+	1	+

Pour obtenir le complément à deux d'un nombre sur 16 bits, il faut complémenter les deux octets séparément puis ajouter 1 au résultat sur 16 bits.

Exemple : soit à trouver le complément à deux du nombre 16 bits contenu dans BC.

LD A, B	78
CPL	2F
LD B, A	47
LD A, C	79
CPL	2F
LD C,A	4F
INC BC	03

Cette procédure sur 7 octets est utilisée dans le programme moniteur - adresse 2657 d - 0A61 h.

Nous avons déjà parlé des instructions qui permettent de modifier l'indicateur de retenue. Elles sont rappelées ici.

CCF complémentation du Bit C, code hexa : 3F

Cette instruction change la valeur de l'indicateur de retenue qui passe de 0 à 1 ou vice versa. Rappelons que AND A est l'instruction usuelle de mise à zéro de cet indicateur.

SCF code hexa 37

L'indicateur de retenue est forcé à un par cette instruction.

APPLICATION :

L'exercice proposé doit vous montrer le résultat d'une instruction logique AND, OR ou XOR.

Il est préférable d'afficher le résultat en hexadécimal. Modifiez le PCCM comme suit :

```
2000 CLS
2010 LET B = USR S
2020 GOSUB 2900
2030 PRINT H$ (3) ; H$ (4)
2040 STOP
```

Après avoir choisi une valeur entre 00 et FF pour n1 et n2 et indiqué le code opératoire de l'instruction logique (sur 2 octets dans ce programme) entrez le code machine.

LD A, n1	3E XX
LD (IY + 21), n2	FD 36 21 XX
instruction logique (IY + 21)	XX XX 21
LD B, 00	06 00
LD C, A	4F
RET	C9

Confrontez les données et le résultat sur une feuille de papier avec un tableau sous la forme.

Exemple :

	code hexa	code binaire
n1	B3	10110011
instruction	**OR**	**OR**
n2	37	00110111
résultat	B7	10110111

CHAPITRE X

INSTRUCTIONS DE BRANCHEMENT

Le compteur ordinal PC (Program Counter) a un rôle capital dans le déroulement d'un programme. C'est dans ce registre 16 bits que le microprocesseur dépose l'adresse de la prochaine instruction à exécuter.

Le premier octet, contenu de l'adresse pointée par PC, est un code opératoire qui détermine la taille de l'instruction 1, 2, 3 ou 4 octets.

Chaque octet est chargé dans un registre interne du Z 80, le registre instruction, puis décodé. A chaque fois PC est incrémenté jusqu'à ce que l'instruction complète ait été lue.

Arrive le stade de l'exécution, PC pointe alors sur l'adresse du premier octet de l'instruction suivante.

Toutes les instructions de branchement, sauts, appels et retours ont pour effet de modifier le contenu actuel du registre PC et ainsi de rompre le déroulement séquentiel du programme.

La plupart de ces instructions utilisent les indicateurs d'état mais aucune ne les affecte. Le cartouche habituel ne figurera pas sous les mnémoniques.

LES INSTRUCTIONS DE SAUT.

Les instructions de saut branchent le programme définitivement (sans retour) sur un emplacement mémoire nommément désigné (adressage absolu) ou désigné par adressage indirect ou par adressage relatif.

Le registre PC est chargé avec l'adresse ainsi spécifiée.

Les sauts peuvent être inconditionnels ou conditionnels. Dans le second cas, leur exécution dépend de la position d'un indicateur d'état désigné.

Les instructions sont similaires au GOTO du Basic.

SAUTS INCONDITIONNELS.

L'adresse de destination est spécifiée en clair ou indirectement au moyen des registres HL, IX ou IY dans les instructions JP. Avec l'instruction de saut relatif JR, elle est obtenue en ajoutant un déplacement d signé, positif ou négatif, à l'adresse détenue par le registre PC qui est, rappelons-le, l'adresse de l'instruction suivante.

mnémonique	code hexa
JP ADR	C3 XX XX
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9
JR d	18 XX

Remarquez l'utilisation inhabituelle des registres 16 bits IX et IY pour l'adressage indirect au lieu de BC et DE.

L'instruction JR d est utile pour les sauts relativements courts (— 128 à + 127).

Elle occupe deux octets au lieu de trois comme dans JP ADR et permet de «reloger» les programmes en mémoire.

Le déplacement d est toujours interprété en complément à deux (voir table en annexe 2).

Sa valeur doit être ajoutée à l'adresse suivant l'instruction JR d. Il faut faire attention pour les sauts en arrière (valeur négative).

	...				JR	d				...	
—128	—127		— 5	— 4	—3	—2	—1	0	1	2	127
80	81		FB	FC	FD	FE	FF	00	01	02	7F

On voit que JR 00 ne sert à rien et que JR FE conduit à une boucle infinie.

Les sauts inconditionnels sont nombreux dans le programme moniteur. On les trouve dès les premières lignes où ils servent à «sauter» divers sous-programmes ou tables de caractères. JP 03CB en 0005 - JR 46 en 000E.

APPLICATION :

Assemblez le programme suivant. Il doit occuper 29 octets. Donnez successivement à d en 30018 les valeurs qui conviennent pour obtenir à l'écran en décimal le nombre chargé dans le registre C.

Pour vous aider, utilisez une étiquette. Exemple en 30008 : VA 29 (valeur affichée = 29). Une étiquette ou label est utilisée par un programme assembleur pour déterminer une adresse de branchement. Dans le cas présent, l'instruction JR, VA 29 en 30017 serait assemblée automatiquement en 18 F5 (d = — 11).

adresse décimal	étiquette (label)	mnémonique	code hexa
30000	début	LD B, 00	
		LD HL, 7541	
		JP 754C	
30008	VA 29	LD C, 1D	
		RET	
		LD C, AF	
		RET	
		LD C, 0D	
		RET	
30017	saut	JR d	18 XX
		LD C, BD	
		RET	
		LD C, 3F	
		RET	
		LD C, 5C	
		RET	
30028		JP (HL)	

Utilisez l'option affichage pour modifier d en 30018.

SAUTS CONDITIONNELS.

Chacun des quatre indicateurs principaux peut être testé pour l'exécution d'un saut conditionnel.

L'adressage absolu et l'adressage relatif sont utilisés pour les sauts conditionnels sur la position des indicateurs C et Z.

Seul l'adressage absolu est employé avec les indicateurs S et P/V.

Les mnémoniques JP ou JR sont suivis d'une abréviation rappelant la condition à remplir, c'est-à-dire la position de l'indicateur.

Exemple : JP NC, ADR Jump if Not Carry, saut absolu à ADR si pas de retenue (indicateur C à zéro).

SUR L'INDICATEUR DE RETENUE.

Quatre instructions permettent d'effectuer des sauts conditionnels sur la position de l'indicateur C.

Elles sont analogues aux lignes Basic IF C THEN GOTO et IF NOT C THEN GOTO

mnémonique	code hexa
JP NC, ADR	D2 XX XX
JP C, ADR	DA XX XX
JR NC, d	30 XX
JR C, d	38 XX

Ces instructions sont souvent employées. Les lignes qui suivent, extraites de la routine de test des paramètres PRINT AT du programme moniteur, montrent trois instructions de saut, conditionnel sur l'indicateur C, successives.

Les numéros de ligne et de colonne contenus respectivement dans les registres B et C sont comparés par soustraction aux valeurs limites chargées dans l'accumulateur 23 et 31 d, 17 et 1F hexa.

Si la valeur est hors limite, la soustraction met l'indicateur C à un et un saut conditionnel lance le message d'erreur B (en 08F8 et 0905).

Si la soustraction en 8F7 donne un résultat positif, c'est-à-dire un nombre de lignes restant en bas de l'écran, la valeur obtenue est comparée à la variable système DF. SZ, nombre de lignes réservées en bas de l'écran, (adresse 4022 - 16418). Là encore le positionnement à 1 de l'indicateur de retenue renvoie à un message de compte-rendu, cette fois le C/R 5.

adresse	code hexa	mnémonique	
2293 - 08F5	3E 17	LD A, 17 h	23 d
	90	SUB B	n° de ligne
08F8	38 0B	JR C, 0905	erreur B
	FD BE 22	CP (4022)	IY + 22
	DA 35 08	JP C, 0835	C/R 5
	3C	INC A	
	47	LD B, A	
	3E 1F	LD A, 1F	31 d
	91	SUB C	n° colonne
0905	DA AD 0E	JP C, 0EAD	erreur B
	C6 02	ADD A, 02	
	4F	LD C, A	

SUR L'INDICATEUR DE ZÉRO.

Cette fois encore, quatre instructions peuvent être utilisées pour effectuer des sauts conditionnels sur la position de l'indicateur de zéro Z.

mnémonique	code hexa
JP NZ, ADR	C2 XX XX
JP Z, ADR	CA XX XX
JR NZ, d	20 XX
JR Z, d	28 XX

Ces instructions sont aussi très communément employées. Nous en avons vu des exemples dans la routine de vérification de la mémoire vive disponible.

SUR L'INDICATEUR DE SIGNE.

Deux instructions utilisant l'adressage absolu permettent d'effectuer des sauts conditionnels sur la position de l'indicateur de signe S.

Elles sont peu employées.

mnémonique	code hexa
JP P, ADR	F2 XX XX
JP M, ADR	FA XX XX

P signifie plus ou positif
M signifie minus ou négatif

SUR L'INDICATEUR DE PARITÉ/DÉBORDEMENT.

Deux instructions permettent d'effectuer des sauts conditionnels sur la position de l'indicateur de parité/débordement P/V. Leur mode d'adressage est aussi l'adressage absolu. Elles sont rarement utilisées.

mnémonique	code hexa
JP PO, ADR	E2 XX XX
JP PE, ADR	EA XX XX

Jump if parity odd or no overflow
si parité impaire ou pas de débordement
indicateur à zéro.

Jump if parity even or overflow
si parité paire ou débordement
indicateur à un.

La routine de «tri des codes caractères» du programme moniteur utilise une instruction JP PE pour séparer les codes des simples caractères, compris entre 00 et 3F hexa, de ceux des commandes d'édition, compris entre 70 et 79 hexa.

adresse	code hexa	mnémonique
1301 - 0515	7E	LD A, (HL)
	FE F0	CP F0
	EA 2D 05	JP PE,052D

Le code caractère est comparé à la constante FO. Il n'y a pas de débordement s'il s'agit d'un simple caractère.

Exemple : la lettre Y, code hexa 3E, donnera 3E — F0 = 4E; en décimal 62 — (—16) = 78 ce qui est correct. Il n'y a pas de débordement, l'indicateur est mis à zéro.

Par contre, les codes des commandes d'édition provoqueront un débordement.

Exemple : EDIT code hexa 75 donnera 75 — F0 = 85, en décimal 117 —(—16) = — 123.

Le résultat est incorrect. L'indicateur de débordement est mis à un et le saut conditionnel est effectué.

L'INSTRUCTION DJNZ d

L'instruction DJNZ d est une instruction puissante et très utile que le Z 80 est l'un des rares microprocesseurs à offrir.

C'est une instruction composée qui décrémente le registre B et effectue un saut relatif si la valeur résultante dans B n'est pas zéro.

Elle présente une grande similitude avec la commande Basic NEXT.

Dans une boucle FOR - NEXT l'instruction FOR sert à définir une variable de contrôle. C'est le registre B chargé avec une certaine valeur qui joue le rôle de variable de contrôle dans l'instruction DJNZ d.

L'exemple suivant montre l'analogie avec le Basic. Remarquez que l'on a donné un pas négatif STEP — 1 à la variable I. Le contenu du registre B est toujours décrémenté dans l'instruction DJNZ.

langage machine	Basic	
LD B, 20 décimal	FOR I = 20 TO 1 STEP-1	variable de contrôle
LD...	LET...	↱ partie active
LD...	LET...	⎡
DJNZ d	NEXT I	⎣ saut à

L'instruction occupe deux octets

mnémonique	code hexa
DJNZ d	10 XX

d représente un déplacement signé, normalement négatif. La valeur de ce déplacement doit être calculée de façon à ce que le saut relatif aboutisse à la partie active de la boucle. Une erreur fréquente consiste à revenir à la variable de contrôle et à créer ainsi une boucle sans fin.

L'instruction DJNZ est utilisée dans la routine d'initialisation pour construire un fichier d'affichage de base constitué de 25 caractères new line (code 76 h). Ce fichier correspond à un écran vide sur une machine 1 K. Sur une machine 16 K, le fichier est rempli ensuite d'espaces (code 00) entre les new lines de façon à former un fichier d'affichage complet (793 octets).

Le registre HL est initialisé avec l'adresse 16509 qui est la valeur actuelle de D.FILE en l'absence de programme.

adresse	code hexa	mnémonique	
1024 - 0400	21 7D 40	LD HL, 407D	(16509)
	22 0C 40	LD (400C), HL	D.FILE
	06 19	LD B, 19	25 N/L
0408	36 76	LD (HL), 76	New line
	23	INC HL	
	10 FB	DJNZ 0408	Next

Encore une fois, regardez bien comment est calculée la valeur du déplacement négatif : FB = — 5.

APPLICATION :

• Sauts conditionnels.

Le programme proposé permet de constater l'exécution éventuelle d'un saut conditionnel.

Ajoutez la ligne suivante au PCCM

2015 IF PEEK 16417 THEN PRINT "SAUT"

Choisissez pour n1 et n2 une valeur comprise entre 0 et 255 décimal (00 et FF hexa).

Nous commencerons par les sauts conditionnels absolus qui doivent brancher le programme, si la condition spécifiée est remplie, à l'adresse 30015 (753F hexa). Vous choisissez un des codes opératoires rappelés ci-contre :

JP NC	D2	JP P	F2
JP C	DA	JP M	FA
JP NZ	C2	JP PO	E2
JP Z	CA	JP PE	EA

adresse	mnémonique	code hexa
30000	LD B, 00	06 00
	LD A, B	78
	LD (4021), A	32 21 40
	LD A, n1	3E XX
30008	ADD A, n2	C6 XX
	LD C, A	4F
30011	JP cond, 753F	XX 3F 75
30014	RET	C9
30015	LD A, 01	3E 01
	LD (4021), A	32 21 40
30020	RET	C9

Pour les sauts conditionnels relatifs, remplacez l'instruction en 30011 par JR cond, 02→ XX 02 00. Pour brancher en 30015, le déplacement est + 2. L'instruction faisant un octet de moins, on la fait suivre de 00, octet inutile (NOP), pour éviter d'avoir à déplacer le reste du programme.

Choisissez un des codes opératoires :

JR NC	30	JR NZ	20
JR C	38	JR Z	28

En 30008 on peut remplacer l'addition par une soustraction SUB n2 (D6 XX).

• Instruction DJNZ.

Cet exercice montre la différence de temps d'exécution entre un programme Basic et son équivalent en langage machine notamment lorsqu'il s'agit de traiter des boucles.

Le programme calcule la somme des 255 premiers nombres entiers.

Entrez le code machine suivant - adresse 30000, 13 octets :

21 00 00 55 06 FF 58 19 10 FC 44 4D C9 faites STOP

puis tapez le programme Basic :

```
                10  LET Z = 0
                20  FOR I = 255 TO 1 STEP-1
calcul          30  LET Z = Z + I
en basic        40  NEXT I
                50  PRINT Z
                60  PRINT "APPUYEZ SUR UNE TOUCHE"

calcul          70  IF INKEY$ = " " THEN GOTO 70
en langage      80  PRINT USR 30000
machine         90  STOP
```

Faites RUN, attendez puis comparez.

Maintenant, désassemblez le code machine à l'aide de la table de décodage en annexe.

Soyez certain de bien comprendre le mécanisme du programme. Utilisation des registres et fonctionnement de l'instruction DJNZ.

Supprimez les lignes 10 à 90 avant de continuer.

LES INSTRUCTIONS D'APPEL.

Ce sont des instructions de saut qui «se rappellent» de l'origine du branchement.

Utilisées pour appeler des sous-programmes, elles sont équivalentes à l'instruction GOSUB du Basic.

L'adresse de retour doit être sauvegardée. Avant d'être remplacé par l'adresse de destination, spécifiée dans l'instruction d'appel, le contenu du compteur ordinal PC est «poussé» dans la pile. Ce contenu est alors, rappelez-vous, l'adresse de la prochaine instruction, c'est-à-dire de celle qui suit l'instruction d'appel. Ce sera l'adresse de retour du sous-programme appelé.

L'octet de poids fort est empilé en premier. Le registre pointeur de pile SP est décrémenté avant l'empilage de chaque octet.

Les instructions CALL utilisent le mode d'adressage absolu, les instructions RST le mode d'adressage page zéro.

CALL

Une instruction CALL permet l'appel inconditionnel d'un sous-programme. Pour les huit autres, l'appel est conditionnel sur la position des 4 principaux indicateurs d'état.

mnémonique	code hexa
CALL ADR	CD XX XX
CALL NC, ADR	D4 XX XX
CALL C, ADR	DC XX XX
CALL NZ, ADR	C4 XX XX
CALL Z, ADR	CC XX XX
CALL P, ADR	F4 XX XX
CALL M, ADR	FC XX XX
CALL PO, ADR	E4 XX XX
CALL PE, ADR	EC XX XX

L'appel inconditionnel est souvent employé alors que l'appel conditionnel est peu fréquent

RST

Les huit instructions RST sont de véritables instructions d'appel (CALL) mais l'adresse du sous-programme n'a pas à être spécifiée car elle est prédéterminée en page zéro. Ces instructions n'occupent qu'un octet.

Exemple : l'instruction simple octet D7 dont le mnémonique est RST 10 appelle le sous-programme du programme moniteur dont l'adresse est 0010 hexa.

mnémonique	code hexa	
RST 00	C7	mise en route
RST 08	CF	compte rendu
RST 10	D7	affichage caractère
RST 18	DF	recueil caractère
RST 20	E7	d'une ligne Basic
RST 28	EF	Entrée calculateur
RST 30	F7	fait de la place en mémoire
RST 38	FF	interruption affichage

Les sous-programmes qui peuvent être appelés par RST font partie intégrante du programme moniteur. Nous en reparlerons au chapitre XIV. Il est classique de trouver les sous-routines souvent utilisées en tête d'un programme.

LES INSTRUCTIONS DE RETOUR.

Il existe une instruction de retour inconditionnel, le simple RET bien connu et huit instructions de retour conditionnel. Les instructions RET sont équivalentes au RETURN du Basic. Cependant les retours conditionnels n'ont pas d'équivalent en Basic.

Il est important de souligner qu'une instruction RET ne fait rien d'autre que de retirer une adresse de la pile et de la placer dans le registre PC. Le premier octet retiré de la pile formant l'octet de poids faible du compteur ordinal et le second l'octet de poids fort. Le registre pointeur de pile SP est incrémenté deux fois.

mnémonique	code hexa
RET	C9
RET NC	D0
RET C	D8
RET NZ	C0
RET Z	C8
RET P	F0
RET M	F8
RET PO	E0
RET PE	E8

Il faut aussi remarquer que l'adresse recueillie dans la pile par une instruction RET n'y aura pas été obligatoirement placée par une instruction CALL correspondante.

NOP et HALT

L'instruction NOP, no operation, code hexa 00, ne sert qu'à perdre du temps. Il faut au microprocesseur 4 cycles soit 1,23 microsecondes pour «l'exécuter».

Elle est utilisée pour les temporisations et aussi pour remplacer le code indésirable au cours de la mise au point d'un programme.

L'instruction HALT arrête le déroulement du programme. Le microprocesseur exécute des NOP internes jusqu'à l'occurence d'une interruption. Cette instruction n'est pas utilisée par le programmeur sur ZX 81.

APPLICATION :

Un programme presque identique à celui proposé pour la constatation des sauts conditionnels peut être utilisé pour les instructions CALL.

Ajoutez la ligne suivante au PCCM

2015 IF PEEK 16417 THEN PRINT "APPEL"

Puis entrez le programme suivant : 21 octets

30000	LD B, 00	06 00
	LD A, B	78
	LD (4021), A	32 21 40
	LD A, n1	3E XX
	ADD A, n2	C6 XX
	LD C, A	4F
30011	CALL cond, 753F	XX 3F 75
30014	RET	C9
30015	LD A, 01	3E 01
	LD (4021), A	32 21 40
30020	RET	C9

Donnez une valeur à n1 et n2 et pour l'instruction CALL choisissez un des codes opératoires rappelés ci-dessous, puis 2> EXEC.

CALL inconditionnel	CD		
CALL NC	D4	CALL P	F4
CALL C	DC	CALL M	FC
CALL NZ	C4	CALL PO	E4
CALL Z	CC	CALL PE	EC

Vous remarquerez que l'instruction RET en 30020 marque la fin du sous-programme commençant en 30015. Elle ne renvoie plus au Basic mais à l'adresse 30014.

Une modification du programme vous permettra d'essayer les retours conditionnels.

Changez la ligne 2015 comme ci-dessous :

2015 IF NOT PEEK 16417 THEN PRINT "RET COND"

Puis entrez le programme suivant : 22 octets

30000	LD B, 00	06 00	
	LD A, B	78	
	LD (4021), A	32 21 40	
30006	CALL 753A	CD 3A 75	
30009	RET	C9	retour au Basic
30010	LD A, n1	3E XX	
30012	SUB n2	D6 XX	
	LD C, A	4F	
30015	RET cond.	XX	
	LD A, 01	3E 01	
	LD (4021), A	32 21 40	
30021	RET	C9	

Codes des retours conditionnels :

RET NC	D0	RET P	F0
RET C	D8	RET M	F8
RET NZ	C9	RET PO	E0
RET Z	C8	RET PE	E8

Remarquez qu'en 30021 en l'absence de retour conditionnel l'instruction RET ne renvoie pas au Basic mais elle est utilisée pour le retour du sous-programme en 30009.

Supprimez la ligne 2015 du PCCM.

BOUCLES ET SOUS-PROGRAMMES.

La possibilité de leur faire effectuer des travaux répétitifs est l'un des principaux avantages offerts par les machines électroniques.

Comme dans les autres langages les boucles de programme sont largement employées en langage machine. Nous avons vu, avec l'instruction DJNZ, qu'elles s'exécutaient bien plus vite qu'en Basic.

Les instructions de branchement confèrent à la machine toute sa puissance mais leur utilisation exige du programmeur beaucoup de méthode.

Il est nécessaire au préalable de définir le but et la structure du programme. L'emploi d'un ordinogramme est à conseiller.

Certaines parties répétitives peuvent être constituées en sous-programmes. La seule modification obligatoire consiste à ajouter une instruction RET pour le retour au programme principal. Les avantages des sous-programmes sont :

- économie de mémoire si le sous-programme est souvent appelé;
- programmes mieux structurés. Possibilité de constitution d'une bibliothèque de sous-programmes utiles.

Cependant, il faut savoir que l'utilisation d'un sous-programme au lieu de la répétition des instructions sans branchement ralentit toujours l'exécution d'un programme.

Les instructions CALL + RET prennent 28 cycles d'horloge pour être exécutées.

Nous allons mettre en pratique des instructions de branchement dans quelques exemples simples.

• Boucle d'attente.

Dans un programme il est parfois nécessaire de «perdre du temps».

Voici deux exemples de temporisation constitués de boucles imbriquées.

La temporisation désirée est obtenue en donnant à B une certaine valeur.

Les résultats sont à peu près les mêmes pour les deux programmes :

Pour B = 1 = 01 h : 1,32 seconde
B = 10 = 0A : 13,2 secondes
B = 100 = 64 : 132 secondes
etc.

```
30000                LD B, n                  06 XX
          ATTENTE    LD C, 00                 0E 00
          Boucle 2   LD D, 00                 16 00
          Boucle 3   DEC D                    15
                     JR NZ, Boucle 3          20 FD
                     DEC C                    0D
                     JR NZ, Boucle 2          20 F8
                     DJNZ ATTENTE             10 F4
                     RET                      C9
```

```
30000                LD B, n                  06 XX
          ATTENTE    PUSH BC                  C5
                     LD B, 00                 06 00
          Boucle 2   LD C, 00                 0E 00
          Boucle 3   DEC C                    0D
                     JR NZ, Boucle 3          20 FD
                     DJNZ, Boucle 2           10 F9
                     POP BC                   C1
                     DJNZ ATTENTE             10 F3
                     RET                      C9
```

Ces boucles d'attente peuvent être utilisées sous forme de sous-programmes. La valeur de la temporisation sera chargée dans le registre B avant l'appel.

Exemple :
```
LD B, 10
CALL ATTENTE
...
```

• Affichage d'un texte à l'écran.

Certains sous-programmes du programme moniteur en ROM sont utilisables par le programmeur. Nous les étudierons au chapitre XIV.

Deux d'entre eux permettent l'affichage à l'écran. Il s'agit des routines.

- affichage d'un caractère, appelée par RST 10
- affichage d'une chaîne de caractères, appelée par CALL 0B6B

Nous allons utiliser l'option 2 du PCCM, entrée d'une chaîne de caractères, pour la première fois.

Procédons pas à pas.

D'abord modifiez le programme comme suit :

```
ligne  1090  INPUT S
       2010  LET K = USR S.
```

Faites RUN - option 2
adresse 30000 N.OCT = 200

Le nombre d'octets exact est recalculé par le programme.

Entrez le texte suivant, **en bloc**

```
30000    IL EXISTE DIFFÉRENTES MÉTHODES *
         D INTRODUCTION DE PARAMÈTRES *
         DANS UN SOUS-PROGRAMME •
```

Les symboles* remplacent un espace, ils sont interprétés comme un retour à la ligne (code 76 h) par le PCCM.

Le texte doit occuper 83 octets. La dernière adresse occupée est 30082.

Retournez au menu.

Choisissez option 1, entrée code hexa.
adresse 30083 N.OCT = 11
Entrez le programme suivant :

30083	LD HL, 7530	21 30 75	ADR début texte
	LD B, 53	06 53	83 caractères
suivant	LD A, (HL)	7E	affichage
	RST 10	D7	un caractère
	INC HL	23	adresse suivante
	DJNZ suivant	10 FB	NEXT
	RET	C9	

HL est chargé avec l'adresse du premier caractère.

B est utilisé comme compteur pour l'instruction DJNZ.

La routine RST 10, affichage d'un caractère, est très simple d'emploi. Il suffit de charger l'accumulateur avec le code caractère avant d'appeler la routine.

Lancez le programme avec 2 > EXEC. Remarquez que cette option envoie à la ligne Basic LET K = USR S où S = 30083, adresse de début de programme.

Le bloc de données en 30000 n'est évidemment pas «exécutable» par le microprocesseur. Il serait désigné à l'intention d'un programme assembleur par la pseudo-instruction

DEFM - Define Message

Il est possible de stocker des messages différents les uns à la suite des autres.

Pour les afficher le moment voulu il est peut-être intéressant de disposer d'un sous-programme d'affichage.

Entrez ce qui suit, adresse 30083 - 15 octets

30083	LD HL, 753A	21 3A 75	
	LD B, 49	06 49	
	CALL AFFICHAGE	CD 8C 75	
	RET	C9	
	sous-programme		
30092	AFFICHAGE	LD A, (HL)	7E
		RST 10	D7
		INC HL	23
		DJNZ - 5	10 FB
		RET	C9

On dispose avec la routine d'affichage d'une chaîne de caractères (adresse 0B6B) d'un sous-programme tout fait.

30083	LD DE, 7551	11 51 75
	LD BC, 001A	01 1A 00
	CALL 0B6B	CD 6B 0B
	RET	C9

Les registres de travail de ce sous-programme en ROM sont DE et BC.

DE doit être chargé avec l'adresse du premier caractère de la chaîne et BC avec le nombre de caractères, préalablement à l'appel de la routine.

Remarquez que ce sous-programme n'est pas adapté à l'affichage d'un texte car il ignore le retour à la ligne NEWLINE (76 h) qu'il remplace par un point d'interrogation.

INTRODUCTION DE PARAMÈTRES DANS UN SOUS-PROGRAMME.

Nous venons de voir une première méthode qui consiste à charger les registres avant l'appel du sous-programme.

C'est la seule utilisable pour transmettre des paramètres à une sous-routine en ROM.

Le programmeur peut en utiliser d'autres pour ses propres sous-programmes.

- **Les paramètres peuvent être rangés après l'instruction d'appel.**

Le sous-programme obtient par POP l'adresse du 1er paramètre qui correspond à l'adresse de retour.

Conservez les 83 octets de texte et entrez le code suivant :

adresse 30083 21 octets

30083	CALL AFFICHAGE	CD 8A 75	
	DEFB	75	
	DEFB	46	paramètres
	DEFB	25	
30089	RET	C9	retour au Basic
30090 AFFICHAGE	POP HL	E1	adresse 1er paramètre
	LD D, (HL)	56	adresse 1er caractère
	INC HL	23	chargée dans DE
	LD E, (HL)	5E	
	INC HL	23	Nb caractères
	LD B, (HL)	46	chargé dans B
	INC HL	23	nouvelle adresse retour
	PUSH HL	E5	remise en pile (30089)
	LD A, (DE)	1A	
	RST 10	D7	boucle d'affichage
	INC DE	13	
	DJNZ - 5	10 FB	
	RET	C9	retour de S/P (30089)

Remarquez que cette fois, c'est DE qui contient l'adresse du caractère à afficher. Il est préférable d'utiliser HL pour pointer l'adresse des paramètres car un registre simple quelconque ne peut être chargé qu'avec le contenu de l'adresse pointée par HL. Avec un autre registre double il faudrait passer par A.

DEFB signifie Define Byte. C'est un pseudo-opérateur qui permet d'entrer un octet de données avec un programme assembleur.

Par rapport à la méthode précédente, l'appel demande deux octets de moins, mais le sous-programme occupe huit octets de plus.

Il y aura économie de mémoire si le programme comporte au moins quatre instructions d'appel.

• Les paramètres peuvent être passés par l'intermédiaire de la pile.

```
30083                LD HL, 7530          21 30 75
                     LD B, 53             06 53
                     PUSH HL              E5
                     PUSH BC              C5
                     CALL AFFICHAGE       CD 8E 75
                     RET                  C9

30094 AFFICHAGE      POP DE               D1      adresse retour
                     POP BC               C1      paramètres
                     POP HL               E1
                     PUSH DE              D5      adresse retour remise en pile
                     LD A, (HL)           7E
                     RST 10               D7      affichage
                     INC HL               23
                     DJNZ - 5             10 FB
                     RET                  C9
```

• Les paramètres peuvent être rangés à des emplacements mémoire réservés.

Adresse = 30083 31 octets

```
Déci   Hexa

30083  7583                 LD HL, 7551          21 51 75
                            LD A, 32             3E 32
                            LD (7592), HL        22 92 75
                            LD (7594), A         32 94 75
                            CALL AFFICHAGE       CD 95 75
                            RET                  C9
30098  7592                 DEFS 3               00
                                                 00     octets réservés
30100  7594                                      00

30101  7595 AFFICHAGE       LD HL, (7592)        2A 92 75
                            LD A, (7594)         3A 94 75
                            LD B, A              47
                            LD A, (HL)           7E
                            RST 10               D7
                            INC HL               23
                            DJNZ - 5             10 FB
                            RET                  C9
```

DEFS suivi d'un nombre signifie Define Storage. Cette directive indique à l'assembleur de réserver un certain nombre d'octets, ici 3. Entrez simplement trois fois 00 pour réserver cet espace.

Cette méthode est utilisée pour passer des paramètres d'un programme Basic à une sous-routine en langage machine.

Soit par exemple à afficher le message situé en 30022 - 37 caractères.

Faites en commande directe :

```
POKE 30098, 70
POKE 30099, 117
POKE 30100, 37
LET K = USR 30101
```

Nous allons voir maintenant un autre procédé d'affichage d'un texte ou de caractères graphiques.

Ensuite, l'utilisation d'une table d'adresses sera illustrée par deux exemples.

Si vous êtes saturé, passez au chapitre suivant. Vous reviendrez plus tard.

Pour afficher un message, notre sous-programme a besoin de l'adresse du 1er caractère.

Si le message est placé après l'instruction d'appel cette adresse est rapidement accessible puisque c'est l'adresse de retour qui se trouve au sommet de la pile.

La fin de texte peut être signalée au moyen d'un marqueur (ici ce sera FF).

Entre le code machine suivant. Pour le message en clair, utilisez l'option 2, entrée chaîne de caractères. Attention, vous n'utiliserez pas EXEC.

```
30000 AFFICHAGE   POP HL              E1          adresse caractère
                  LD A, (HL)          7E
                  INC HL              23          adresse suivante
                  PUSH HL             E5          mise en pile
                  CP FF               FE FF       marqueur fin texte
                  RET Z               C8
                  RST 10              D7
                  JR AFFICHAGE        18 F6

30010   DEBUT     CALL AFFICHAGE      CD 30 75
30013             DEFM - (17 octets)  SALUT LES COPAINS
                  DEFB                FF
                  RET                 C9

30032   DEBUT     CALL AFFICHAGE      CD 30 75
30035             DEFM - (17 octets)  28 26 00 35 3A
                                      2A00 31 2A38
                                      00 28 34 31 2E
                                      33 38
                  DEFB                FF
                  RET                 C9

30054   DEBUT     CALL AFFICHAGE      CD 30 75
                  DEFM (41 octets)    85 03 01 07 84 85
                                      00 00 05 85 02 84
                                      03 76 02 03 05 82
                                      81 85 00 00 05 85
                                      00 85 00 76 02 03
                                      01 01 02 02 03 01
                                      03 03 00 02 00
                  DEFB                FF
                  RET                 C9
```

Faites successivement en commande directe :

```
LET K = USR 30010
LET K = USR 30032
LET K = USR 30054
```

Au chapitre XIV nous verrons comment modifier la position d'écriture avec la sous-routine "PRINT AT" en ROM.

LES TABLES D'ADRESSES.

Les tables sont souvent utilisées en langage machine. Un paramètre correspondant à une certaine valeur d'entrée peut être passé à un programme par ce moyen.

Ainsi un branchement peut être effectué à une adresse donnée suivant le résultat obtenu par un traitement antérieur.

Un exemple concret vaut mieux que de longues explications.

Notre sous-programme d'affichage servira à matérialiser un branchement obtenu de cette façon.

Imaginez que dans un programme de jeu vous vouliez afficher une appréciation sur le score obtenu selon le barême suivant :

de 0 à 7 insuffisant, de 8 à 12 moyen
de 13 à 16 bien, de 17 à 20 très bien.

Le programme se présente comme suit :

```
          adresse

décimal   hexadécimal

30000     7530          AFFICHAGE POP HL            E1
                                  LD A, (HL)        7E
                                  INC HL            23
                                  PUSH HL           E5
                                  CP FF             FE FF
                                  RET Z             C8
                                  RST 10            D7
                                  JR AFFICHAGE      18 F6

30010     753A     ADR    CALL AFFICHAGE  CD 30 75
30103                     DEFM (11 octets) INSUFFISANT
30024                     DEFB             FF
                          RET              C9

30026     754A     ADR    CALL AFFICHAGE  CD 30 75
30029                     DEFM (5 octets)  MOYEN
30034                     DEFB             FF
                          RET              C9

30036     7554     ADR    CALL AFFICHAGE  CD 30 75
30039                     DEFM (4 octets)  BIEN
30043                     DEFB             FF
                          RET              C9
```

```
30045   755D    ADR   CALL AFFICHAGE   CD 30 75
30048                 DEFM (9 octets)  TRES BIEN
30057                 DEFB             FF
                      RET              C9

30059   756B           DEFS 01         00          stockage résultat

30060                  DEFB            07          limite 7
                       DEFW            3A 75       insuffisant
                       DEFB            0C          limite 12
        table d'adresses DEFW          4A 75       moyen
                       DEFB            10          limite 16
                       DEFW            54 75       bien
                       DEFB            14          limite 20
                       DEFW            5D 75       très bien

30072   USR            LD A, (756B)    3A 6B 75    résultat
                       LD C, A         4F          chargé en C
                       LD HL, 7569     21 69 75    init. HL
        limite suiv.   INC HL          23
                       INC HL          23
                       INC HL          23          HL pointe val limite
                       LD A, (HL)      7E
                       CP C            B9          le résultat est comparé
                       JR C limite suiv. 38 F9     R > LIMITE
                       INC HL          23          octet faible ADR
                       LE E, (HL)      5E          chargé en E
                       INC HL          23          octet fort ADR
                       LD D, (HL)      56          chargé en D
                       EX DE, HL       EB          HL contient ADR
                       JP (HL)         E9          saut à ADR
```

```
Partie Basic  10  PRINT "RESULTAT ? (0 A 20)"
              20  INPUT R
              30  PRINT
              40  IF R > 20 THEN GOTO 20
              50  PRINT R
              60  POKE 30059, R
              70  LET K = USR 30072
              80  GOTO 20
              90  STOP
RUN
```

La première partie du programme vous est familière. En 30059 est stocké le résultat (ligne 60 du Basic). En 30060 est la table d'adresses. Chaque valeur limite est suivie par l'adresse de branchement qui convient.

Le programme démarre en 30072. Après chargement des registres de travail, le résultat est comparé aux valeurs limites successives. Si la comparaison ne donne pas de retenue, l'adresse qui suit devient l'adresse de branchement.

DEFW signifie Define Word, c'est l'équivalent de DEFB pour une donnée sur 16 bits, généralement une adresse.

D'autres méthodes existent. La plus classique consiste à séparer les deux tables : celle des valeurs et celle des adresses. Essayez-la. Le code machine est à modifier à partir de 30060. Vous pouvez conserver le reste.

Dans l'exemple qui suit, les tables sont séparées. La table d'adresses est utilisée pour fournir au sous-programme d'affichage l'adresse du début du message.

Remarquez la solution différente pour l'affichage.

Conservez la partie Basic, modifiée comme suit :

```
ligne 60  POKE 30040, R
      70  LET K = USR 30053
```

Faites RUN 199 puis entrez le code machine.

```
30000 AFFICHAGE     LD A, (HL)          7E
                    INC HL              23
                    AND A               A7          Test A = 00
                    RET Z               C8          retour si espace
                    RST 10              D7
                    JR AFFICHAGE        18 F8

30007               DEFM                INSUFFISANT espace MOYEN espace
                    (33 octets)         BIEN espace TRES-BIEN espace

30040               DEFS-1              00

30041 - 7559        DEFB                07
TABLE DES VALEURS   DEFB                0C
(LIMITES)           DEFB                10
                    DEFB                14
                (1)
30045 - 755D        DEFW                37 75
TABLE DES           DEFW                43 75
ADRESSES            DEFW                49 75
                    DEFW                4E 75

30053   USR         LD DE, 7558         11 58 75    initialisation
                    LD HL, 755B         21 5B 75
                    LD A, (7558)        3A 58 75    résultat
                    LD C, A             4F          chargé en C
VAL. SUIVANTE       INC DE              13
                    INC HL              23
                    INC HL              23
                (2) LD A, (DE)          1A
                    CP C                B9
                    JR C, VAL. SUIVANTE 38 F9
                    LD E, (HL)          5E
                    INC HL              23
                    LD D, (HL)          56
                    EX DE, HL           EB
                    CALL AFFICHAGE      CD 30 75
                    RET                 C9
```

Bien entendu, on pourrait modifier le programme de façon à lui faire afficher un message pour une valeur précise.

Il faudrait alors ajouter un marqueur (00 par exemple) en fin de table des valeurs pour le cas où la valeur entrée n'existerait pas dans la table (1).

Les lignes ci-dessus deviendraient : (2)

```
LD A, (DE)          1A
AND A               A7
RET Z               C8      A = marqueur 00, pas trouvé
CP C                B9
JR NZ VAL. SUIVANTE 20 F7
etc...
```

Supprimez les lignes 10 à 90 du PCCM.

CHAPITRE XI

INSTRUCTIONS SUR BLOCS

Le Z 80 possède deux ensembles d'instructions originales qui permettent d'opérer sur des blocs d'octets consécutifs, c'est-à-dire sur des zones déterminées de la mémoire.

Ce sont des instructions composées dont la fonction principale, transfert ou comparaison (LD ou CP) est rappelée dans le mnémonique. Elles assurent en plus l'incrémentation ou la décrémentation des registres doubles qu'elles utilisent comme compteur ou pointeur d'adresse.

Certaines peuvent être qualifiées d'automatiques. Le traitement du bloc entier est assuré par l'instruction elle-même grâce à une fonction «Repeat» incluse dans l'instruction.

Les autres «non automatiques», traitent un octet à la fois. Une «boucle» doit être installée par le programmeur si le traitement concerne un certain nombre d'octets.

Les instructions sur blocs sont puissantes et utiles mais plutôt compliquées. Leur mécanisme doit être bien compris.

Ce chapitre complète l'ensemble des instructions du Z 80 qu'il est indispensable de connaître. Nous commencerons à aborder les aspects pratiques de la programmation en langage machine du système SINCLAIR. Ne négligez pas les applications qui seront proposées.

LES INSTRUCTIONS DE TRANSFERT PAR BLOC.

Ces instructions permettent de déplacer un ensemble d'octets consécutifs d'une zone mémoire à une autre. Rappelons que, comme pour toutes les instructions LD, il s'agit d'une copie. Les emplacements mémoire source ne sont pas «vidés».

Auparavant les registres doubles HL, DE et BC doivent être initialisés de la manière suivante :

HL : Adresse du 1er octet à transférer - Source
DE : Adresse du 1er emplacement mémoire à occuper - Destination
BC : Nombre d'octets à déplacer.

Recette mnémonique DE = Destination.

On trouve quatre instructions dans ce sous-groupe.

mnémonique	code hexa
LDIR	ED B0
LDDR	ED B8
LDI	ED A0
LDD	ED A8

S	Z	H	P/V	N	C
–	–	0	x	0	–

LDIR - LDDR

L'instruction LDIR effectue la séquence d'opérations suivante :

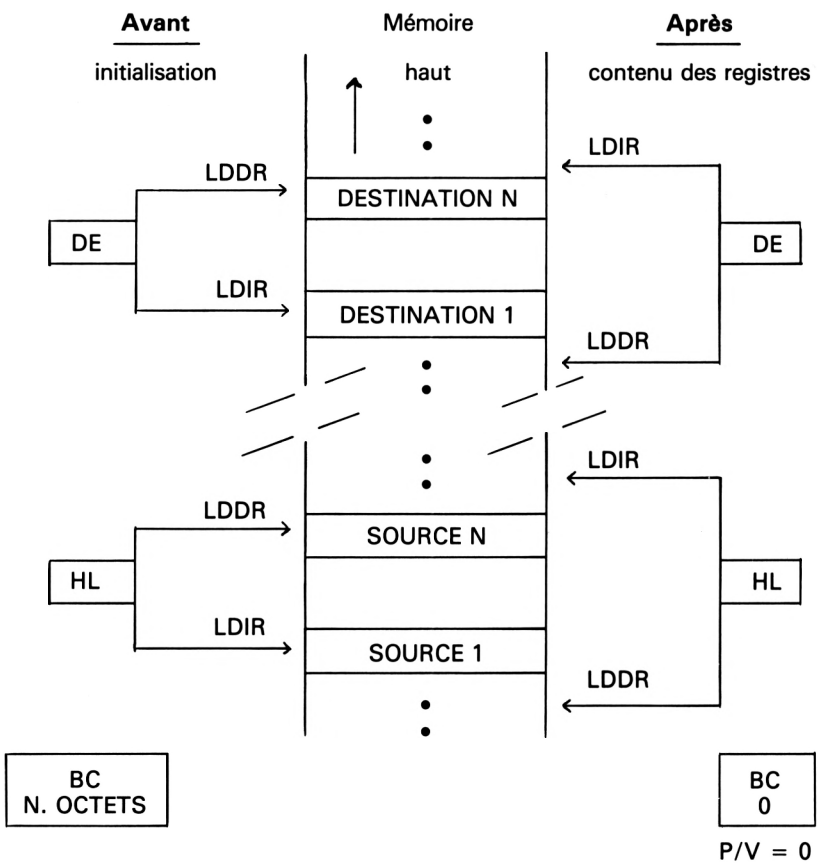
1. - L'octet situé à l'adresse contenue dans le registre double HL est chargé dans l'emplacement mémoire pointé par DE.
2. - Les contenus des registres HL et DE sont incrémentés.
3. - Le contenu du registre BC est décrémenté.
4. - Le contenu de BC est testé. S'il est différent de zéro les opérations 1, 2 et 3 sont répétées. S'il est égal à zéro, le programme passe à l'instruction suivante.

Pendant le transfert l'indicateur de parité/débordement P/V est à un. Il est mis à zéro dès que BC = 0 (transfert terminé). Les autres indicateurs principaux ne sont pas affectés.

L'instruction LDDR est similaire. La seule différence étant qu'à l'étape 2 les contenus des registres HL et DE sont décrémentés.

Suivant l'instruction utilisée les registres HL et DE seront initialisés avec l'adresse du premier ou du dernier octet des zones mémoire concernées.

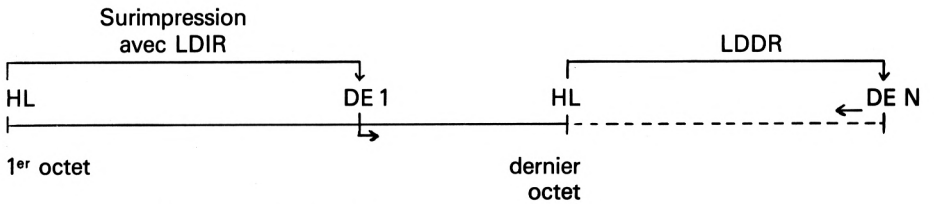
Le schéma ci-après montre l'effet des instructions LDIR et LDDR sur les registres et sur la mémoire.



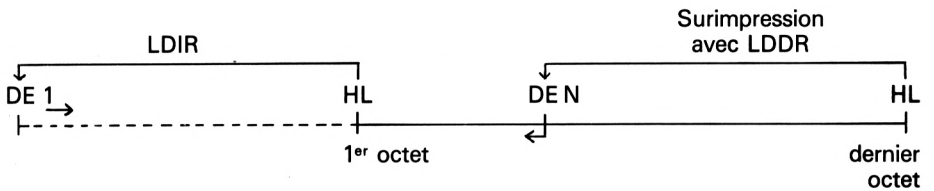
Le bloc d'octets peut être transféré à des adresses plus élevées ou moins élevées en mémoire. (DE) peut être supérieur ou inférieur à (HL).

Quand les deux zones sont séparées (cas du schéma) LDIR ou LDDR peuvent être utilisées, suivant les nécessités du programme, sans précaution particulière.

Quand les deux zones se chevauchent, une seule des deux instructions peut être employée sans risque de surimpression. Ce sera LDDR si (DE) > (HL) et LDIR si (DE) < (HL).



Bloc N. octets à transférer



Bloc à transférer

Il est dit à la page 171 de votre manuel SINCLAIR ZX 81 que la taille des différentes zones (programme, fichier d'affichage, variables) concorde exactement avec l'information qu'elles contiennent. Par exemple, si une variable simple est ajoutée, le bloc de code situé entre l'octet contenant 80 h et STK-END doit être décalé de 6 octets vers le haut.

Le programme moniteur se sert de LDIR ou de LDDR pour ces transferts de blocs.

C'est LDDR qui est utilisée pour «faire de la place» en mémoire comme le montre l'extrait ci-dessous.

Après la routine de changement des pointeurs, BC contient le nombre d'octets à déplacer, HL est chargé avec la nouvelle adresse STK END.

Après EX DE, HL, HL contient l'ancienne STK END et DE la nouvelle et LDDR est exécutée.

adresse	code hexa	mnémonique	
2467-09A3	CD AD 09	CALL 09AD	changement pointeurs
	2A 1C 40	LD HL, (401C)	STK-END
	EB	EX DE, HL	
	ED B8	LDDR	

LDI - LDD

Les instructions LDI et LDD transfèrent un seul octet de (HL) à (DE).

Le contenu de BC est décrémenté. Celui de HL et de DE est incrémenté avec LDI, décrémenté avec LDD.

L'indicateur de parité est mis à zéro si BC = 0. Il est mis à un si BC est différent de zéro.

LES INSTRUCTIONS DE RECHERCHE PAR BLOC.

Ces instructions sont basées sur la comparaison d'un ou de plusieurs octets consécutifs d'une zone mémoire déterminée avec une valeur clé contenue dans l'accumulateur.

Comme pour les instructions de transfert par bloc certains registres doivent être initialisés.

HL : Adresse du 1er octet à comparer
BC : Nombre d'octets du bloc sur lequel s'effectue la recherche.
A : Valeur clé.

mnémonique	code hexa
CPIR	ED B1
CPDR	ED B9
CPI	ED A1
CPD	ED A9

S	Z	H	P/V	N	C
+	x	+	x	1	—

CPIR - CPDR

L'instruction CPIR effectue la séquence d'opération suivante :

1. - L'octet d'adresse pointée par HL est comparé avec le contenu de l'accumulateur. Les indicateurs S, Z et H sont positionnés en fonction du résultat de la comparaison. L'indicateur de retenue n'est pas affecté contrairement à ce qui se passe avec la simple instruction CP (HL).

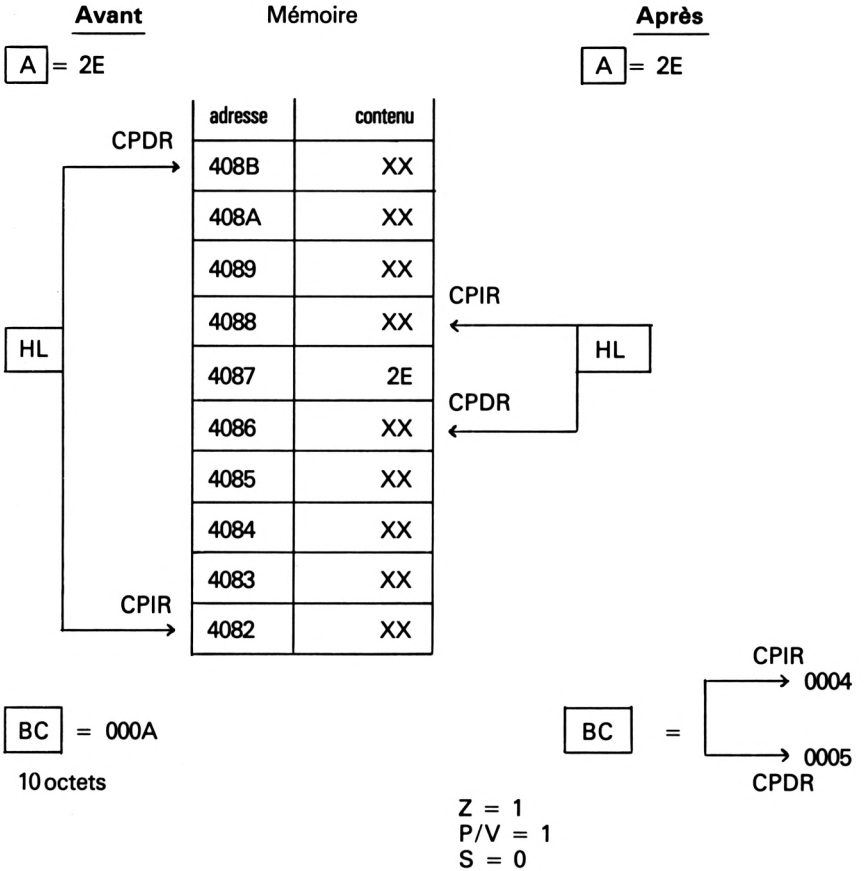
2. - Le contenu du registre HL est incrémenté.

3. - Le contenu du registre BC est décrémenté.
   L'indicateur Z est mis à un si (HL) = A
   L'indicateur P/V est mis à zéro si BC = 0

4. - Si BC = 0 ou (HL) = A l'indicateur de signe S est mis à zéro et le programme passe à l'instruction suivante, sinon les étapes 1, 2 et 3 sont répétées.

Dans l'instruction CPDR à l'étape 2, le registre HL est décrémenté au lieu d'être incrémenté. Il aura été initialisé avec l'adresse du dernier octet du bloc sur lequel s'effectue la recherche.

Le schéma ci-dessous montre l'effet des instructions CPIR et CPDR dans un exemple concret. Le bloc d'octets est compris entre les adresses 4082 et 408B. L'adresse 4087 contient la valeur cherchée.



CPI - CPD

Avec les instructions CPI ou CPD, seules les étapes 1, 2 et 3 de la séquence décrite précédemment sont effectuées.

L'octet pointé par HL est comparé avec le contenu de l'accumulateur.

Si les deux valeurs sont égales, l'indicateur de zéro est mis à un, sinon il est mis à zéro.

HL est incrémenté ou décrémenté.

BC est décrémenté. Si son contenu égale zéro l'indicateur de parité P/V est mis à zéro.

L'indicateur de signe S est mis à zéro et le programme passe à l'instruction suivante.

APPLICATION :

Le premier problème qui se pose lorsqu'on écrit un programme en langage machine est de choisir l'endroit de la mémoire où il sera implanté.

L'utilisation d'une instruction REM en début de programme Basic est probablement la solution la plus satisfaisante.

Nous mettrons cette méthode en pratique pour l'expérimentation des instructions sur bloc.

Modifiez le programme de chargement comme suit :

ligne 1090 INPUT S

Vous pouvez conserver la ligne 2010 PRINT USR S

Ajoutez la ligne suivante :

1 REM 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

Les caractères seront remplacés par le code machine. Ils peuvent être quelconques mais leur nombre doit être au moins égal au nombre d'octets à entrer. L'utilisation de chiffres facilite le comptage. Si le nombre d'octets est important comptez par lignes. Une ligne = 32 caractères sauf la première (après REM) qui en contient 26.

Dans le cas présent, 17 octets sont réservés.

Faites RUN - option 2 - entrée chaîne de caractères.

adresse début = 16514 N. octets = 17

Tapez en entier le message "SALUT LES COPINES" puis entrez-le en bloc avec New-line.

Le message s'affiche puis après la pause (que vous pouvez abréger avec New-line) apparaissent les codes caractères.

Faites STOP puis LIST.

Le message est inscrit en clair après l'instruction REM.

Les 17 octets que nous venons d'entrer en mémoire ne sont pas un programme mais des données (DATA en anglais). Nous avons vu qu'il existe une pseudo-instruction DEFM pour désigner un tel bloc d'octets à l'assembleur.

L'interpréteur considère le contenu d'une instruction REM comme une suite de codes caractères d'où l'aspect généralement biscornu du listing. Ce n'est pas le cas ici puisqu'il ne s'agit pas d'un programme mais d'une suite cohérente de caractères.

Entrez maintenant une seconde ligne REM identique à la première.

2 REM 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

Faites RUN - option 3 affichage

adresse début 16509 N. octets = 48

Voyons comment sont stockées en mémoire nos deux lignes REM. La première commence en 16509.

16509	16510	16511	16512	16513
00	01	13	00	EA
N° de ligne		Nombre d'octets qui suivent		REM

16514	16515	16516	...	16531
38	26	31		76
S	A	L	...	NEW-LINE

La seconde ligne commence en 16532

16532	16533	16534	16535	16536
00	02	13	00	EA
n° de ligne		Nb. d'octets		REM

16537	16538	16539	...	16554
1D	1E	1F	...	76
1	2	3		NEW-LINE

La ligne suivante en 16555 est 00C7 → 199

Remarques :

- Le nombre d'octets utilisés pour le stockage d'une ligne Basic ne correspond pas au nombre de caractères affichés à l'écran.
- Les deux premiers octets indiquent le numéro de ligne. Leur disposition ne suit pas la règle habituelle d'inversion des octets. Ils sont dans l'ordre normal, octet de poids fort en premier.
- Les octets 3 et 4 (inversés) précisent le nombre d'octets qui les suivent. Dans le cas présent 19 octets (13 hexa) soit un octet pour REM, 17 octets pour le message et un octet pour NEW-LINE.
- La fin de la ligne est toujours marquée par un caractère NEW-LINE (76 hexa).

INSTRUCTIONS LDIR ET LDDR.

Le transfert de notre message de la ligne 1 à la ligne 2 ne présente pas de difficulté avec LDIR. HL est chargé avec l'adresse du 1er octet source : 16514 d. DE est chargé avec l'adresse du 1er octet destination 16537 d et BC est chargé avec le nombre d'octets à transférer 17.

mnémonique	code hexa
LD HL, 16514 d	21 82 40
LD DE, 16537 d	11 99 40
LD BC, 17 d	01 11 00
LDIR	ED B0
RET	C9

Retournez au menu.

Entrez ce programme à l'adresse 30000 et faites 2 > EXEC.

Le nouveau contenu de BC, zéro, apparaît à l'écran indiquant que LDIR a été exécutée.

Vérifiez avec LIST.

On aurait pu utiliser LDDR, HL et DE étant initialisés avec les adresses finales.

Retapez la ligne 2 REM qui sert de témoin

2 REM 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 et essayez.

Attention, l'adresse finale d'un bloc est celle du 17e octet. Le code de LDDR est ED B8.

Utilisez l'option 4, listing pour la conversion décimal-hexa.

Examinez le programme suivant :

```
LD HL, 4099        21 99 40
XOR A              AF
LD (HL), A         77
LD DE, 409A        11 9A 40
LD BC, 0010        01 10 00
LDIR               ED B0
RET                C9
```

Avant de l'entrer en mémoire (adresse 30000) et de le faire tourner, assurez-vous que les deux lignes REM contenant 17 caractères sont bien en place et répondez aux questions ci-dessous :

- Quel sera le résultat de ce programme ?
- Quel sera le contenu de la ligne 2 au listing ?

LDI - LDD

Après l'exercice précédent, la première ligne REM doit contenir le message initial et la seconde doit être blanche (mais non vide). Si tel n'est pas le cas, retapez les deux lignes comme suit :

```
1 REM SALUT LES COPAINS
2 REM          17 espaces
```

Entrez et faites tourner le programme suivant : adresse 30000 20 octets

```
30000          LD HL, 4082       21 82 40     initialisation
               LD DE, 4099       11 99 40     registres
               LD BC, 0011       01 11 00
   COMPAR.     LD A, (HL)        7E           «marqueur» fin
30010          CP "T"            FE 39        de bloc à transférer
30012          LDI               ED A0        transfert
30014          JP PO, FIN        E2 43 75     Pas trouvé BC = 0
30017          JR NZ, COMPAR.    20 F6        comparer octet suivant
30019 FIN      RET               C9
```

Après exécution, le contenu de BC = 12 est affiché. Cinq octets ont été transférés (le premier mot du message). Vérifiez avec LIST.

Maintenant faites RUN, option affichage, adresse 30000, 20 octets. Remplacez le code caractère en 30011 (lettre T) par celui d'une lettre qui n'existe pas dans le message, Z par exemple, 3F.

Faites 2 EXEC.

BC = 0 Les 17 octets ont été transférés. Le branchement conditionnel (P/V = 0) a joué. Sans cette instruction en 30014 le tranfert aurait continué jusqu'au prochain octet 3F détruisant notre programme Basic (crash).

Remarquez que c'est la comparaison en 30010 qui positionne l'indicateur de zéro utilisé pour le branchement relatif en 30017 (l'instruction LDI n'affecte pas l'indicateur Z).

INSTRUCTIONS DE RECHERCHE PAR BLOC - CPIR-CPI.

Pour expérimenter ces instructions nous logerons le programme dans une instruction REM. Vérifiez ligne 1090 INPUT S. Supprimez la ligne 2010.

Tapez les lignes suivantes :

```
 1.  REM 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
10  INPUT A$
20  POKE 16417, CODE A$
30  PRINT USR 16514
40  STOP
```

Vous savez que la première partie de la table des caractères en ROM, des adresses 007E à 00CB (126 à 203 décimal), contient les codes des caractères normaux ou avec Shift (mode L).

Le programme proposé a pour but de retrouver dans la table grâce à CPIR l'adresse d'un caractère frappé au clavier.

Le programme basic inscrit le code caractère à l'adresse disponible 16417. Cette valeur est «recueillie» par le programme en langage machine et chargée dans l'accumulateur. L'instruction CPIR compare le contenu des adresses successives de la table avec l'accumulateur. Quand (HL) = A l'instruction DEC HL redonne l'adresse correspondante qui est transférée dans BC pour affichage par PRINT USR 16514.

Faites RUN 199, puis

Entrez le programme, adresse 16514, 15 octets

LD HL, 007E	21 7E 00
LD BC, 004E	01 4E 00
LD A, (4021)	3A 21 40
CPIR	ED B1
DEC HL	2B
LD B, H	44
LD C, L	4D
RET	C9

Faites STOP puis RUN.

Tapez un caractère quelconque.

L'adresse du caractère dans la table est affichée.

Vérifiez avec PRINT CHR$ PEEK adresse.

Le programme que nous allons essayer maintenant vous donnera une idée de la façon dont peut être constitué un répertoire téléphonique.

Tapez les lignes Basic suivantes :

```
 1  REM  38 caractères
10  DIM A$(2)
20  INPUT A$
30  IF CODE A$ = 28 THEN GOTO 80
40  POKE 16507, CODE A$ (1)
50  POKE 16508, CODE A$ (2)
60  LET K = USR 16514
70  STOP
80  LET K = USR 16545
90  STOP
```

Faites RUN 199 et entrez le programme suivant : adresse 16514, 38 octets.

16514	INIT.	LD HL,7530	21 30 75	début table
		LD BC, 0053	01 53 00	83 octets
	RECHERCHE	LD A, (407B)	3A 7B 40	1e lettre
		CPIR	ED B1	
		JP PO FIN	E2 A0 40	fin du bloc
		LD A, (407C)	3A 7C 40	2e lettre
		CPI	ED A1	
		DEC HL	2B	
		JR NZ, RECHERCHE	20 F0	2e lettre incorrecte
		DEC HL	2B	adresse début ligne
		LD B, 0D	06 0D	13 octets
	AFFICH.	LD A, (HL)	7E	pour affichage
		RST 10	D7	
		INC HL	23	
		DJNZ - 5	10 FB	
16544	FIN	RET	C9	retour Basic
16545	REPERTOIRE	LD HL, 7530	21 30 75	début table
		LD B, 53	06 53	83 octets
		JR AFFICH.	18 F3	à afficher

Retournez au menu et choisissez option 2.

Entrez la chaîne de caractères suivante : 83 octets, adresse 30000

TOTO 11 22 33 * MIMI 22 33 44 * PEPE 33 44 55 * LULU 44 55 66 * BIBI 55 66 77 * TATA 66 77 88

Faites STOP puis RUN.

Entrez les 2 premières lettres du nom choisi, exemple PE ou 0 pour le répertoire.

CHAPITRE XII

ROTATIONS, DÉCALAGES, OPÉRATIONS SUR BITS

Les instructions de ce groupe composent près de la moitié du jeu d'instructions du Z 80.

Elles agissent directement sur la structure d'un octet. L'octet concerné peut être le contenu d'un des registres simples A, B, C, D, E, H ou L ou d'un emplacement mémoire pointé par HL ou indexé par IX ou IY.

Rappelons que les 8 bits d'un octet sont numérotés de 0 à 7 de la droite vers la gauche. Le bit 0 étant le moins significatif et le bit 7 le plus significatif.

B7	B6	B5	B4	B3	B2	B1	B0

Rotations et décalages déplacent d'un rang les 8 bits dans un sens ou dans l'autre. Les instructions se différencient par la façon dont sont pris en compte le bit sortant, le bit entrant et l'indicateur de retenue.

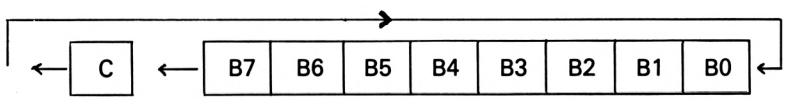
Les instructions sur bit opèrent sur un bit particulier.

Les deux instructions spéciales de rotation décimale par demi-octet seront étudiées en fin de chapitre avec l'instruction DAA.

LES INSTRUCTIONS DE ROTATION.

Elles se divisent en quatre catégories.

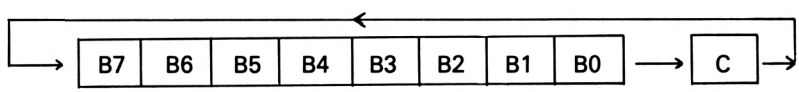
RL **Rotate left - Rotation à gauche**



L'indicateur de retenue C est inclus dans la rotation qui se fait ainsi sur 9 bits.

Les bits sont déplacés d'un rang à gauche.
Le bit C va en B0
Le bit 7 va en C

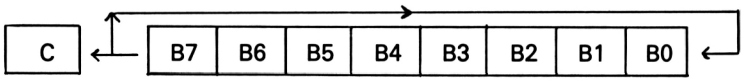
RR **Rotate right - Rotation à droite**



C'est aussi une rotation sur 9 bits.

Les bits sont déplacés d'un rang à droite.
Le bit C va en B7
Le bit 0 va en C.

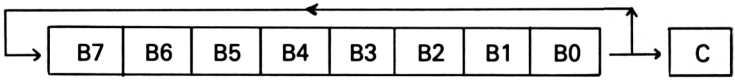
RLC **Rotate left circular - Rotation circulaire à gauche.**



La rotation ne porte que sur les 8 bits de l'octet.

Le bit sortant B7 est copié à la fois dans C et en B0.

RRC **Rotate right circular. - Rotation circulaire à droite.**



Rotation sur 8 bits. Le bit sortant B0 est copié en C et en B7.

Les codes hexadécimaux des instructions de rotation figurent ci-dessous.

Les quatre instructions sur un octet RLA, RRA, RLCA et RRCA qui opèrent sur le contenu de l'accumulateur sont héritées du microprocesseur 8080. Elles ont le même effet que les instructions de la première ligne du tableau sauf en ce qui concerne l'action sur les indicateurs. Elles n'affectent que l'indicateur de retenue.

Les quatre indicateurs principaux sont positionnés par les instructions sur deux ou quatre octets.

L'indicateur C est positionné par la rotation elle-même.

L'indicateur S est une recopie du bit de signe de l'octet résultant.

L'indicateur Z est mis à un si l'octet résultant vaut zéro.

L'indicateur P/V est mis à un s'il y a parité.

RLA	RRA	RLCA	RRCA
17	1F	07	0F

S	Z	H	P/V	N	C
–	–	0	–	0	+

	RL	RR	RLC	RRC
A	CB 17	CB 1F	CB 07	CB 0F
H	CB 14	CB 1C	CB 04	CB 0C
L	CB 15	CB 1D	CB 05	CB 0D
B	CB 10	CB 18	CB 00	CB 08
C	CB 11	CB 19	CB 01	CB 09
D	CB 12	CB 1A	CB 02	CB 0A
E	CB 13	CB 1B	CB 03	CB 0B
(HL)	CB 16	CB 1E	CB 06	CB 0E
(IX + d)	DD CB	DD CB	DD CB	DD CB
	XX 16	XX 1E	XX 06	XX 0E
(IY + d)	FD CB	FD CB	FD CB	FD CB
	XX 16	XX 1E	XX 06	XX 0E

S	Z	H	P/V	N	C
+	+	0	+	0	+

APPLICATION :

CONVERSION DÉCIMAL-HEXADÉCIMAL

```
Basic   1  REM 28 caractères
       10  INPUT A
       20  PRINT A
       30  LET B = INT (A/256)
       40  POKE 16508, B
       50  POKE 16507, A - (B * 256)
       60  LET K = USR 16531
       70  STOP
```

Code machine : adresse 16514 - 28 octets

16514	CONV.	PUSH AF	F5	sauvegarde de A
		AND F0	E6 F0	masque
		RRA	1F	décalage
		RRA	1F	1/2 octet
		RRA	1F	
		RRA	1F	
		ADD A, 1C	C6 1C	+ 28 d pour code caract.
		RST 10	D7	affichage
		POP AF	F1	on récupère A
		AND 0F	E6 0F	masque
		ADD A, 1C	C6 1C	+ 28 d pour code caract.
		RST 10	D7	affichage
		RET	C9	retour Basic ou 2e octet
16531		LD HL, (407B)	2A 7B 40	N chargé dans HL
		LD A, H	7C	octet fort
		OR A	B7	conv. si
		CALL NZ, CONV	C4 82 40	A non nul
		LD A, L	7D	octet faible
		JR CONV	18 E4	conv.

Faites RUN et entrez un nombre entre 0 et 65535.

Le programme démarre en 16531. Le nombre inscrit en 16507 et 16508 est chargé dans HL. Chaque octet est chargé dans A pour traitement par le sous-programme en 16514.

Si l'octet de poids fort n'est pas nul (N > 255) on obtient 4 digits hexadécimaux. S'il est nul, seul l'octet de poids faible est converti.

Voyons sur un exemple comment fonctionne le sous-programme de conversion.

Soit à convertir 181 d = 10110101 = B5 h.

Cette valeur est sauvegardée par PUSH AF.

Après le masquage par AND F0
10110000

Après les 4 décalages RRA.
00001011 = 11 d

Après l'addition de 28 d
00100111 = 39 d

C'est le code caractère du chiffre hexadécimal B. Ce chiffre est affiché par RST 10.

La valeur originale de l'octet est recouvrée par POP AF.

Après le masquage par AND 0F
00000101 = 5 d

Après l'addition de 28 d
00100001 = 33 d

C'est le code caractère du chiffre 5 affiché par RST 10.

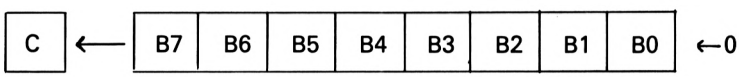
LES INSTRUCTIONS DE DÉCALAGE

Ces instructions sont utilisées principalement dans la programmation d'opérations arithmétiques.

Un décalage à gauche a pour effet de doubler la valeur de l'octet, un décalage à droite de la diviser par deux.

On distingue trois catégories.

SLA **Shift left arithmetic - Décalage arithmétique à gauche**



B7 va en C - B0 est mis à zéro.

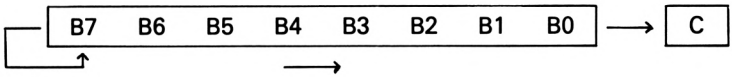
Voyons quelques exemples avec SLA A

	Indicateur C	Accumulateur
1. - avant	?	00101011 = 43 d
après	0	01010110 = 86 d
2. - avant	?	01111111 = 127 d
après	0	11111110 = 254 d
3. - avant	?	10100001 = 161 d
3. - après	1	01000010 = 66 d

Dans ce dernier exemple C indique une retenue, le résultat réel est 256 + 66 = 322 (161 * 2).

SRA **Shift right arithmetic - Décalage arithmétique à droite**

Cette instruction a pour effet de diviser par deux le contenu du registre ou de l'emplacement mémoire concerné. L'opération est réalisée suivant les règles de l'arithmétique en complément à deux. Le bit de signe du quotient est conservé. Le reste de la division va en C.



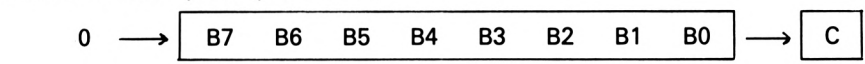
B0 va en C
B7 est inchangé.

Exemples : SRA A.

	Indicateur C	Accumulateur
1. - avant	?	01111111 = 127 d
après	1	00111111 = 63 d
	(reste)	
2. - avant	?	10110101 = — 75 d
après	1	11011010 = — 38 d
	(reste)	

SRL **Shift right logical - Décalage logique à droite.**

L'instruction est symétrique de SLA.



B0 va en C
B7 est mis à zéro.

Exemple : SRL A

	Indicateur C	Accumulateur
1. - avant	?	11101111 = 239 d
après	1	01110111 = 119 d
	(reste)	

Le tableau ci-dessous donne les codes hexadécimaux des instructions de décalage.

	SLA	SRA	SRL
A	CB 27	CB 2F	CB 3F
H	CB 24	CB 2C	CB 3C
L	CB 25	CB 2D	CB 3D
B	CB 20	CB 28	CB 38
C	CB 21	CB 29	CB 39
D	CB 22	CB 2A	CB 3A
E	CB 23	CB 2B	CB 3B
(HL)	CB 26	CB 2E	CB 3E
(IX + d)	DD CB	DD CB	DD CB
	XX 26	XX 2E	XX 3E
(IY + d)	FD CB	FD CB	FD CB
	XX 26	XX 2E	XX 3E

S	Z	H	P/V	N	C
+	+	0	+	0	+

Les indicateurs principaux sont affectés de la même façon que par les instructions de rotation multi-octets.

Les instructions de décalage peuvent être utilisées avec RL pour effectuer des décalages sur plusieurs octets.

Exemple : décalage à gauche du contenu du registre double BC.

```
SLA C
RL B
```

APPLICATION :

CONVERSION DÉCIMAL - BINAIRE

d'un nombre de 0 à 255.

Basic

```
 1 REM 13 caractères
10 INPUT X
20 PRINT X
30 POKE 16417, X
40 LET K = USR 16514
50 STOP
```

Code machine : adresse 16514, 13 octets

16514	LD B, 08	06 08	compteur 8 bits
	XOR A	AF	A = 0 C = 0
	SLA (IY + 21)	FD CB 21 26	décalage B7 dans C
	ADC A, 1C	CE 1C	code caract. 28 ou 29 d
	RST 10	D7	affichage
	DJNZ - 10	10 F6	bit suivant
	RET	C9	

Le décalage à gauche SLA (IY + d) transfère chacun des 8 bits dans l'indicateur de retenue qui se retrouve dans A (ADC).

L'addition avec retenue ADC A, 1C donne 28 ou 29 d, code caractère de 0 ou 1.

Voici maintenant un programme sommaire de multiplication sur 8 bits.

Etudiez-en le mécanisme.

Donnez différentes valeurs à n1 et n2 (00 à FF).

	LD HL, 0000	21 00 00	INIT. HL
	LD DE, 00n1	11 XX 00	multiplicande
	LD A, n2	3E XX	multiplicateur
MULT.	SRL A	CB 3F	Test bit
	JR NC + 2	30 02	multiplicateur
	ADD HL, DE	19	Total partiel
	OR A	B7	Test si A = 0
	JR Z + 6 FIN	28 06	bits restants à zéro
	SLA E	CB 23	décalage
	RL D	CB 12	multiplicande
	JR MULT.	18 F2	bit suivant
FIN	LD B, H	44	Transfert BC
	LD C, L	4D	affichage par
	RET	C9	PRINT USR

Vous pouvez entrer ce programme à l'adresse 30000 Faites en commande directe : PRINT USR 30000

LES INSTRUCTIONS SUR BITS.

Les instructions de ce sous-groupe se divisent en trois catégories : test, mise à un ou mise à zéro d'un bit.

Il existe une instruction différente pour chacun des bits de l'octet concerné, ce qui donne au total 240 instructions

80 instructions BIT test d'un bit
80 instructions SET mise à un d'un bit
80 instructions RES mise à zéro d'un bit.

BIT

Ces instructions permettent de tester la valeur d'un bit particulier. L'octet concerné n'est pas altéré. Le résultat du test est recueilli par l'indicateur de zéro.

Si le bit est à zéro - Z est mis à un
Si le bit est à un - Z est mis à zéro

S	Z	H	P/V	N	C
?	+	1	?	0	—

SET RES

Les instructions SET permettent de mettre à 1 un bit spécifié.

Les instructions RES le mettent à zéro.

Si le bit a déjà la valeur désirée, l'instruction ne fait que confirmer la situation.

Les indicateurs d'état ne sont pas affectés.

Le tableau ci-après regroupe les 240 instructions sur bits.

Notez que tous les codes opératoires indiqués doivent être précédés par CB. Les instructions utilisant l'adressage indexé comportent en plus le préfixe DD ou FD et la valeur du déplacement.

Voici quelques exemples en langage d'assemblage avec le code hexadécimal correspondant.

mnémonique	code hexa
BIT 3, A	CB 5F
SET 0, E	CB C3
RES 6, (HL)	CB B6
BIT 5, (IX + d)	DD CB XX 6E
RES 0, (IY + d)	FD CB XX 86

Les instructions sur bits sont rarement utilisées. Il faut remarquer que le microprocesseur met au moins autant de temps pour modifier un simple bit que pour changer la valeur de l'octet entier.

La plupart des opérations sur bits peuvent être effectuées avec les instructions logiques.

L'instruction BIT b, A peut cependant être préférée à l'instruction AND pour déterminer la valeur d'un bit car elle ne modifie pas le contenu de l'accumulateur, ce que fait un octet de masquage.

		bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
A	BIT	47	4F	57	5F	67	6F	77	7F
(CB)	SET	C7	CF	D7	DF	E7	EF	F7	FF
	RES	87	8F	97	9F	A7	AF	B7	BF
H	BIT	44	4C	54	5C	64	6C	74	7C
(CB)	SET	C4	CC	D4	DC	E4	EC	F4	FC
	RES	84	8C	94	9C	A4	AC	B4	BC
L	BIT	45	4D	55	5D	65	6D	75	7D
(CB)	SET	C5	CD	D5	DD	E5	ED	F5	FD
	RES	85	8D	95	9D	A5	AD	B5	BD
B	BIT	40	48	50	58	60	68	70	78
(CB)	SET	C0	C8	D0	D8	E0	E8	F0	F8
	RES	80	88	90	98	A0	A8	B0	B8
C	BIT	41	49	51	59	61	69	71	79
(CB)	SET	C1	C9	D1	D9	E1	E9	F1	F9
	RES	81	89	91	99	A1	A9	B1	B9
D	BIT	42	4A	52	5A	62	6A	72	7A
(CB)	SET	C2	CA	D2	DA	E2	EA	F2	FA
	RES	82	8A	92	9A	A2	AA	B2	BA
E	BIT	43	4B	53	5B	63	6B	73	7B
(CB)	SET	C3	CB	D3	DB	E3	EB	F3	FB
	RES	83	8B	93	9B	A3	AB	B3	BB
(HL)	BIT	46	4E	56	5E	66	6E	76	7E
CB	SET	C6	CE	D6	DE	E6	EE	F6	FE
	RES	86	8E	96	9E	A6	AE	B6	BE
(IX + d)	BIT	46	4E	56	5E	66	6E	76	7E
DD CB XX	SET	C6	CE	D6	DE	E6	EE	F6	FE
	RES	86	8E	96	9E	A6	AE	B6	BE
(IY + d)	BIT	46	4E	56	5E	66	6E	76	7E
FD CB XX	SET	C6	CE	D6	DE	E6	EE	F6	FE
	RES	86	8E	96	9E	A6	AE	B6	BE

APPLICATION :

Voici une variante du programme de conversion décimal-binaire vu précédemment qui utilise l'instruction BIT.

Elle est moins efficace que la première.

Vous pouvez conserver le même programme Basic avec 1 REM 18 caractères.

Code machine : adresse 16514, 18 octets.

LD HL, 4021	21 21 40	adresse nombre dans HL
LD B, 08	06 08	compteur 8 bits
BIT 7, (HL)	CB 7E	Test B7
LD A, 1C	3E 1C	Code caract. 0
JR Z + 1	28 01	Affichage 0 si B7 = 0
INC A	3C	code caract. 1
RST 10	D7	affichage
SLA (HL)	CB 26	bit suivant en B7
DJNZ - 12	10 F4	Next
RET	C9	

LE DÉCIMAL CODÉ BINAIRE ET LES INSTRUCTIONS SPÉCIALISÉES

Le fonctionnement des routines arithmétiques d'un système exige que les nombres soient mémorisés sur un format (nombre d'octets) déterminé quelle que soit leur valeur.

Si le résultat ne tient pas dans le format fixé le programme conserve généralement les bits les plus à gauche, les plus significatifs. Il y a une perte de précision par «troncation» du résultat.

Ainsi les nombres en virgule flottante manipulés par la ROM 8 K du ZX 81 sont mémorisés sur 5 octets. Un octet pour l'exposant et 4 octets (32 bits, dont le premier est le bit de signe) pour la mantisse. La précision est de 9 1/2 chiffres décimaux.

Dans certaines applications de gestion, aucune perte de précision n'est tolérable. Une solution consiste à représenter les nombres en DCB Décimal codé binaire (en anglais BCD Binary coded decimal).

Le principe est de coder chaque chiffre séparément et d'utiliser autant de bits que nécessaire. Chaque chiffre de 0 à 9 est codé sur un quartet (4 bits) ou 1/2 octet.

Les codes binaires correspondent à ceux des dix premiers chiffres hexadécimaux, les autres ne sont pas utilisés.

DCB	binaire	DCB	binaire
0	0000	8	1000
1	0001	9	1001
2	0010	inutilisé	1010
3	0011	inutilisé	1011
4	0100	inutilisé	1100
5	0101	inutilisé	1101
6	0110	inutilisé	1110
7	0111	inutilisé	1111

Sur un octet les chiffres décimaux codés binaire peuvent représenter un nombre de :

0 00000000
à 99 10011001

sur deux octets par exemple :

	0111	0101	1000	0001
Représente	7	5	8	1

Par convention, dans la désignation d'un nombre complet, le nombre lui-même sera précédé de plusieurs quartets précisant le nombre de chiffres utilisés, le signe, la position de la virgule.

DAA

Certains microprocesseurs possèdent un mode arithmétique DCB. Ce n'est pas le cas du Z 80 où toutes les opérations arithmétiques sont effectuées en binaire naturel.

Une instruction spéciale d'ajustement décimal de l'accumulateur, DAA (Decimal adjust accumulator) permet de corriger le résultat d'une opération sur deux nombres BCD.

Cette instruction utilise les indicateurs H et C. Nous n'entrerons pas dans le détail de son fonctionnement illustré par les quelques exemples suivants.

Soit à additionner deux nombres DCB contenus dans les registres A et B.

```
ADD A, B
DAA
```

	1	2	3	4
Avant addition				
registre B	00010011	00101001	01010001	10011000
DCB	1 3	2 9	5 1	9 8
accumulateur	00100011	00101000	01110010	10001001
DCB	2 3	2 8	7 2	8 9
Après addition				
accumulateur	00110110	01010001	11000011	00100001
hexa	3 6	5 1	C 3	2 1
indicateur H	0	1	0	1
indicateur C	0	0	1	1
Après DAA				
accumulateur	00110110	01010111	00100011	10000111
DCB	3 6	5 7	2 3	8 7

Le code opératoire de DAA est 27 hexa.

Tous les indicateurs d'état sont affectés sauf N.

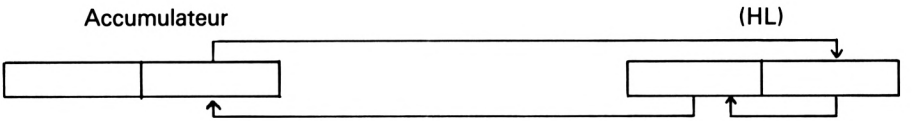
S	Z	H	P/V	N	C
+	+	+	+	–	+

RLD RRD Rotate left decimal, Rotate right decimal

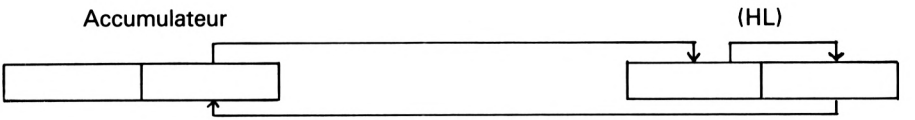
Ces instructions de rotation agissent sur des demi-octets ou quartets. Elles sont donc particulièrement utilisées en arithmétique DCB.

Les quartets concernés sont le quartet le moins significatif de l'accumulateur et les 2 quartets de l'emplacement mémoire pointé par HL.

- Rotation à gauche d'un demi-octet (RLD)



- Rotation à droite d'un demi-octet (RRD)



Les codes hexadécimaux sont les suivants :

RLD ED 6F
RRD ED 67

S	Z	H	P/V	N	C
+	+	0	+	0	—

Tous les indicateurs sont affectés sauf C.

CHAPITRE XIII

ENTRÉES/SORTIES - INTERRUPTIONS

Après ce chapitre, vous aurez parcouru le jeu complet des instructions du Z 80 A.

A moins de vous lancer dans des applications particulières vous n'emploierez guère les instructions de ce groupe. La plupart, d'ailleurs, ne sont pas utilisables sur SINCLAIR.

LES INSTRUCTIONS D'ENTRÉES/SORTIES.

Ces instructions permettent au microprocesseur de dialoguer avec l'extérieur.

Le Z 80 possède de nombreuses instructions pour recueillir (IN) ou envoyer (OUT) des données en provenance ou à destination d'organes périphériques.

En ce qui concerne le système ZX 81, l'information entrante provient du clavier ou du lecteur de cassettes. L'information sortante peut aller vers le téléviseur ou l'enregistreur de cassettes via la puce logique ou vers l'imprimante.

L'information est transmise un octet à la fois. Dans le cas d'une instruction IN le microprocesseur recueille l'octet sur le bus de données et en charge un registre simple spécifié. Dans le cas d'une instruction OUT il place une copie du contenu d'un registre simple sur le bus de données.

Dans le même temps l'adresse du port d'entrée/sortie est placée sur le bus d'adresses et certains signaux de commande sont émis.

Le port d'entrée/sortie est l'accès, la porte, par où doit passer l'information en provenance ou à destination d'un périphérique donné.

L'octet bas d'adresse du port E/S est déterminé par le programmeur. Il peut être chargé dans le registre C.

L'octet haut est formé par le contenu du registre A ou B.

mnémonique	code hexa	registre	adresse Port E/S	
			octet haut	octet bas
IN A, (n)	DB XX	A	A	n
IN A, (C)	ED 78	A	B	C
IN H, (C)	ED 60	H	B	C
IN L, (C)	ED 68	L	B	C
IN B, (C)	ED 40	B	B	C
IN C, (C)	ED 48	C	B	C
IN D, (C)	ED 50	D	B	C
IN E, (C)	ED 58	E	B	C

OUT (n), A	D3 XX	A	A	n
OUT (C), A	ED 79	A	B	C
OUT (C), H	ED 61	H	B	C
OUT (C), L	ED 69	L	B	C
OUT (C), B	ED 41	B	B	C
OUT (C), C	ED 49	C	B	C
OUT (C), D	ED 51	D	B	C
OUT (C), E	ED 59	E	B	C

Pour le SINCLAIR les valeurs de (n) ou (C) sont, en hexadécimal, FE pour l'entrée clavier ou cassette, FF pour la sortie TV ou cassette, FB pour la sortie imprimante.

Il existe des instructions d'entrées/sorties par blocs, analogues à celles que nous avons vues pour le transport ou la recherche.

Le contenu du registre C indique le périphérique concerné.

Ces instructions sont citées pour mémoire car leur utilisation est liée principalement à l'emploi de disques.

mnémonique	code hexa
INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA
OUTI	ED A3
OUTIR	ED B3
OUTD	ED AB
OUTDR	ED BB

Au prochain chapitre nous verrons comment est utilisée l'instruction IN A, (C) dans la routine de scrutation du clavier du programme moniteur.

Pendant le chargement d'un programme en mémoire des bigarrures apparaissent à l'écran attestant que le chargement est effectif (en principe).

Cela est réalisé par l'ajout d'une instruction de sortie après l'instruction d'entrée.

Le signal est renvoyé «en écho» à l'écran par OUT (FF), A.

Voici le passage concerné extrait de la routine «LOAD».

adresse	mnémonique	code hexa
848 - 0350	LD A,7F	3E 7F
	DB FE	IN A,(FE)
	D3 FF	OUT (FF), A

APPLICATION :

Voici un exemple d'utilisation (peu orthodoxe) des instructions d'entrées/sorties.

Ce programme joue un air connu sur votre téléviseur. Vous aurez probablement à modifier l'accord UHF.

Si vous avez l'oreille musicale vous pouvez modifier la valeur des notes en 30045. Diminuez cette valeur pour un son plus aigu. La valeur 00 correspond à «l'attaque» de la note suivante.

		adresse 30000	90 octets	
30000	FREQ.	LD A, E	7B	Fréquence du son
		DEC A	3D	suivant valeur note
		JR NZ	20 FD	
		RET	C9	
30005-7535	JEU	LD B, EO	06 E0	Compteur durée
	NOTE	LD E, (HL)	5E	valeur note
		XOR A	AF	Test si 00
		CP E	BB	
		JR Z + 12	28 0C	si oui pause
		IN A, (FE)	DB FE	si non
		CALL FREQ	CD 30 75	jouer
		OUT (FF), A	D3 FF	la note
		CALL FREQ	CD 30 75	
		JR + 5	18 05	
		LD C, 05	0E 05	pause
		DEC C	0D	
		JR NZ - 3	20 FD	
		DJNZ NOTE	10 E8	boucle durée
		RET	C9	
30032	USR	LD HL, 755D	21 5D 75	adresse 1e note
		CALL JEU	CD 35 75	
		INC HL	23	note suivante
		LD A, (HL)	7E	
		CP FF	FE FF	Test fin
		JR NZ - 9	20 F7	
		RET	C9	

30045-755D DEFM. NOTES

A3 00 A3 00 A3 77 77 00 77
77 00 69 69 00 69 69 00 50
50 50 50 00 60 77 00 00 00
00 00 77 00 60 00 77 90 90
58 58 58 00 69 81 77 77 FF

Partie Basic - ATTENTION, ce programme ne tourne qu'en mode «FAST».

```
10 FAST
20 LET K = USR 30032
30 SLOW
40 STOP
```

LES INSTRUCTIONS D'INTERRUPTION.

Le Z 80 est susceptible d'être interrompu. Cela signifie que le microprocesseur peut être arrêté au cours de l'exécution d'un programme pour accomplir une certaine tâche au profit d'un organe périphérique.

En ce qui concerne le système ZX 81, les interruptions n'ont lieu que pour l'affichage à l'écran.

Deux liaisons de commande peuvent acheminer un signal d'interruption.

La première est la ligne d'interruption non masquable. - NMI non maskable interrupt. - Quand cette ligne est activée le Z 80 termine l'instruction en cours puis stoppe l'exécution du programme.

Le contenu du registre PC, compteur ordinal, est sauvegardé dans la pile pour servir d'adresse de retour.

Puis PC est chargé avec l'adresse 0066 h et le microprocesseur exécute les instructions qui s'y trouvent.

Le SINCLAIR utilise cette caractéristique pour l'affichage en mode normal (SLOW).

Quand un affichage est requis (tous les $1/50^{e}$ de seconde environ) la ligne NMI est activée et le microprocesseur exécute la routine d'affichage du mode Slow en 0066-102 d.

Quand cette routine est terminée, l'adresse de retour est désempilée et le programme principal continue.

La seconde ligne, INT, est masquable, c'est-à-dire que les interruptions peuvent être autorisées ou non par le programmeur.

A la mise sous tension, le système est désactivé (disabled) les interruptions ne sont pas possibles. Il faut une instruction EI - Enable Interrupt - pour les autoriser.

L'instruction opposée DI - Disable Interrupt - les interdit.

Quand les interruptions sont autorisées, elles sont gérées suivant l'un de trois modes programmables : Mode 0, 1 ou 2. Les instructions correspondantes sont :

IM0 - Dans ce mode le périphérique concerné lui-même doit placer une information sur le but de données pour signifier quelle adresse RST est à charger dans le compteur ordinal.

IM1 - L'adresse 0038 est placée dans le compteur ordinal et la routine RST correspondante est exécutée.

IM2 - Le périphérique doit fournir un octet de données, qui est utilisé comme octet de poids faible d'une adresse. Le contenu du registre I formant l'octet de poids fort. Cette adresse permet d'entrer dans une table et de déterminer une autre adresse qui elle, va dans le registre PC

Le retour d'une routine d'interruption nécessite une instruction RET.

Deux instructions spéciales de retour peuvent être utilisées. Elles ont pour effet d'autoriser à nouveau les interruptions.

RETI return from maskable interrupt.
RETN return from non maskable interrupt.

Dans tout le programme moniteur, le mode d'interruption utilisé est le mode 1 qui est spécifié dans la routine d'initialisation à l'adresse 03F6 - 1014 d.

Les codes des instructions d'interruption sont :

mnémonique	code hexa
EI	FB
DI	F3
IM0	ED 46
IM1	ED 56
IM2	ED 5E
RETI	ED 4D
RETN	ED 45

Il est utile de reparler de l'instruction HALT, code hexa 76, en raison du rôle particulier qu'elle joue en relation avec les interruptions.

Lorsque cette instruction est rencontrée, le microprocesseur stoppe l'exécution du programme et effectue des NOP internes jusqu'à l'occurence d'une interruption.

Sur ZX 81 cette interruption ne peut avoir lieu que pour l'affichage en mode normal.

Entrez le programme Basic suivant :

```
10  LET A = 30000
20  POKE A, 118                    HALT    76
30  POKE A + 1, 201                RET     C9
40  LET K = USR 30000
50  PRINT "INTERRUPTION"
60  STOP
```

Faites RUN. Le retour au Basic s'effectue.

Maintenant faites FAST puis RUN.

C'est le crash. Le microprocesseur exécute indéfiniment ses NOP. Il n'y a pas d'interruption pour affichage en mode rapide.

Vous avez sans doute remarqué que le code caractère de NEWLINE était aussi 76 hexa. Ce n'est pas une coïncidence. Quand un caractère NEWLINE est sorti du fichier d'affichage, le microprocesseur exécute l'instruction HALT. L'interruption survient quand le registre R a fini de compter les 32 caractères de la ligne. C'est une particularité du système.

L'étude des instructions d'entrées/sorties et d'interruption conduit naturellement à parler du fichier d'affichage.

Vous lirez les pages qui suivent en liaison avec le chapitre 27 du manuel ZX 81.

LE FICHIER D'AFFICHAGE.

Le clavier incorporé pour l'entrée et le téléviseur pour la sortie sont les principaux périphériques du système.

Pour former une page d'écran, le modulateur UHF reçoit les données du microprocesseur et les incorpore au signal vidéo.

Le fichier d'affichage (DISPLAY-FILE) est la zone réservée en mémoire vive, après la zone Programme Basic, où l'information est rangée avant d'être envoyée à l'écran.

L'adresse du premier octet est détenue par la variable système D.FILE (16396 - 16397). Le dernier est situé avant la zone des variables dont l'adresse est contenue dans VARS (16400 - 16401).

Le premier octet est toujours un caractère NEW-LINE. Chacune des 24 lignes se termine par ce même caractère.

Dans un système 1 K, la taille du fichier est adaptée à l'information à afficher. Un écran vide correspond à 25 caractères NEWLINE.

Lorsque la mémoire disponible est supérieure à 3, 1/4 K le fichier est à sa taille maximale soit 793 octets - (32 x 24) caractères + 25 NEWLINE - les emplacements non occupés sont remplis avec des espaces (code 00 hexa).

Pour exécuter la commande LIST (ou en listing automatique), l'interpréteur copie tout ou partie de la zone programme dans le fichier d'affichage. Le fichier d'affichage est la copie en mémoire de l'image présentée sur le téléviseur. L'information y est inscrite sous une forme appropriée. Ainsi on n'y trouve que les codes caractères compris entre 00 et 3F et entre 80 et BF et le code de New Line.

Entrez le programme Basic suivant :

```
10  LET A = PEEK 16400 + 256 * PEEK 16401
20  PRINT A
30  LET B = PEEK 16396 + 256 * PEEK 16397
40  PRINT B
50  PRINT A — B                                   (VARS — D.FILE)
60  STOP
RUN
```

Ajoutez la ligne 55 PRINT
Faites RUN
Supprimez les lignes 10 à 60.

Nous allons examiner à nouveau le stockage d'une ligne Basic en zone programme. Il s'agira en l'occurence de la première ligne de notre programme utilitaire - 199 GOTO 1000.

Vérifiez ligne 1090 INPUT S.

Faites RUN, option affichage, adresse 16509, 18 octets. On obtient :

00	C7	0C	00	EC	1D	1C	1C	1C	7E
ligne 199		Nb. caract.		GOTO	1	0	0	0	séparateur

8A	7A	00	00	00	76	00	C8
1000 en virgule flottante					NEW LINE	ligne 200	

On constate que les dix octets après EC (le «jeton» pour GOTO) contiennent deux représentations différentes du nombre 1000.

La première est sous forme de codes caractères, la seconde sous la forme «virgule flottante».

Le code 7E qui les sépare signifie pour la machine «les cinq octets qui suivent représentent un nombre en virgule flottante». Nous en reparlerons.

En vue du listing cette même ligne Basic est inscrite ainsi dans le fichier d'affichage.

1D	25	25	00	2C	34	39	34	00
1D	1C	1C	1C	00	...	...	00	76

Le numéro de ligne est en code caractères - GOTO est reproduit en entier. Le séparateur 7E et les cinq octets qui suivent n'apparaissent pas.

Lorsqu'un programme Basic est exécuté, le fichier d'affichage contient l'information spécifiée par les commandes PRINT, PRINT AT, PLOT, UNPLOT, ou CLS (espaces).

Modifiez comme suit la ligne 1090 du PCCM.

```
1090  LET S = PEEK 16396 + PEEK 16397 * 256
```

Faites RUN, option affichage.

C'est la valeur de D.FILE qui s'affiche.

(D.FILE) = DEBUT 34 octets

on obtient :

76	?	?	?	?	14	29	2A	27
NL			(D.FILE) ?		=	D	E	B
3A	39	00	12 espaces		00	33	1B	
U	T	—	—		—	N	•	
34	28	39	14	1F	20	00	76	
O'	C	T	=	3	4	espace	NL	

Le premier caractère newline marque le début du fichier d'affichage, l'autre indique la fin de la ligne. Entre les deux on retrouve les 32 caractères affichés à l'écran, sauf l'adresse du dernier caractère dont l'affichage est postérieur à l'exécution de cette partie du programme.

On peut écrire dans le fichier d'affichage en faisant attention de ne pas écraser un des caractères NEWLINE.

Faites LIST 1090, EDIT et modifiez la ligne comme suit :

```
1090  LET S = 1 + PEEK 16396 + PEEK 16397 * 256
```

Faites RUN, option 2, entrée chaîne de caractères et entrez le message "DERNIER OCTET =" (14 octets). Voyez ce qui se passe.

Retournez au menu, option 3, affichage 34 octets. Les lignes 1000 à 1300 ont rétabli le fichier d'affichage à son état normal.

Supprimez la dernière modification à la ligne 1090 et entrez le même message à l'adresse affichée. C'est le crash. Vous avez écrasé le caractère New Line de début du fichier.

La manipulation du fichier d'affichage offre au programmeur des possibilités intéressantes notamment pour les applications graphiques.

Il faut prendre garde, lorsqu'on remplit le fichier, de ne pas écraser un des caractères Newline dont le système a besoin pour son fonctionnement.

Deux exemples d'utilisation du fichier d'affichage vous sont proposés.

Je vous engage à essayer des programmes de votre cru après avoir expérimenté ceux-ci.

APPLICATION :

Retapez la ligne 1090 INPUT S.

Ce programme est simple, il remplit l'écran de caractères graphiques.

	adresse 30000	35 octets
début	LD (DF.SZ), 00	FD 36 22 00
	LD HL, (D.FILE)	2A 0C 40
	LD C, 06	0E 06
	CALL MOTIF	CD 47 75
	LD C, 09	0E 09
	CALL MOTIF	CD 47 75
	LD C, 85	0E 85
	CALL MOTIF	CD 47 75
	RET	C9
Motif	LD B, 00	06 00
	INC HL	23
	LD A, (HL)	7E
	CP 76	FE 76
	JR Z - 6	28 FA
	LD (HL), C	71
	DJNZ - 9	10 F7
	RET	C9

Faites STOP puis en commande directe LET K = USR 30000.

La première instruction modifie la variable système DF SZ - Nombre de lignes réservées en bas de l'écran. Ce nombre étant maintenant égal à zéro, tout l'écran, soit 24 lignes, est à notre disposition.

Remarquez que cette instruction utilise l'adressage indexé LD (IY + 22), 00. Le registre IY contient l'adresse de base des variables sytème 4000 hexa et DF.SZ est en 4022 hexa.

L'instruction suivante charge HL avec l'adresse de début du fichier d'affichage contenue dans D.FILE.

Le registre C est chargé avec un code caractère graphique avant l'appel du sous-programme de «remplissage» du fichier d'affichage.

Remarquez les précautions qui sont prises pour ne pas «écraser» les new-lines.

Tout d'abord le premier emplacement rempli n'est pas le premier octet qui est, rappelez-vous, un newline.

La comparaison CP 76 fait passer à l'emplacement suivant s'il y a égalité.

Le registre B compteur de boucle est chargé avec zéro, ce qui équivaut à 256. Dans l'instruction DJNZ, le test de zéro est effectué après la décrémentation de B - Decrement and Jump if Not Zero

256 = 8 lignes de 32 caractères pour chacun des 3 motifs.

Cette méthode convient pour des graphiques simples. Les figures plus complexes, en particulier celles qui font appel à des fonctions mathématiques peuvent être réalisées par un programme Basic.

Le fichier d'affichage peut être sauvegardé en mémoire et restitué au moment voulu.

Plusieurs images différentes peuvent être stockées de cette façon, la seule limite étant la capacité de mémoire vive du système.

Rappelez-vous, une page d'écran occupe 793 octets.

Le programme que je vous propose est volontairement très simple. Il est destiné à vous mettre sur la voie de réalisations plus perfectionnées.

Réservez une ligne 1 REM, 18 caractères.

Faites RUN, option 1 et entrez le code machine.

adresse 16514	18 octets	
LD HL, (D.FILE)	2A 0C 40	Transfert
LD DE, 7200	11 00 72	fichier en 7200 h
LD (RAMSTOP), DE	ED 53 04 40	et
LD BC, 0319	01 19 03	réinitialisation
LDIR	ED B0	avec
JP NEW	C3 C3 03	nouveau RAMTOP (7200)

Retournez au menu, option 1.

adresse 30000	13 octets	
LD HL, 7200	21 00 72	remise en
LD DE, (D.FILE)	ED 5B 0C 40	place
LD BC, 0319	01 19 03	fichier
LDIR	ED B0	d'affichage
RET	C9	

Faites STOP. Entrez le programme Basic suivant. C'est un programme de démonstration de l'imprimante ZX.

```
10  FOR J = 1 TO 10
20  FOR I = 0 TO J * 12
30  PLOT 32 + J * 2 * SIN (I/(J * 6) * PI), 22 + J * 2 * COS (I/(J * 6) * PI)
40  NEXT I
50  NEXT J
60  LET K = USR 16514
```

Faites RUN.

Vous avez le temps d'aller prendre une bière. Le mode rapide est moins lent mais personnellement je préfère voir ce qui se passe.

Lorsque le dessin est terminé, l'écran se vide, puis le curseur réapparaît.

Faites LIST. Plus rien. La mémoire est vide.

Rassurez-vous, tout est en sécurité au-dessus de RAMTOP.

Entrez la ligne Basic :

10 LET K = USR 30000

puis faites RUN.

Le transfert du fichier d'affichage à l'adresse choisie 7200 hexa (29184 d), ni le retour n'appellent de remarque particulière.

Vous connaissez bien le fonctionnement de l'instruction LDIR.

Notez que BC est chargé avec 0319 hexa = 793 d.

Le saut à l'adresse 03C3 permet une réinitialisation avec la nouvelle valeur de RAMTOP (adresse de l'image). Si ça ne vous paraît pas évident, retournez à la fin du chapitre VIII.

Rappelez-vous que pour pouvoir utiliser les 24 lignes de l'écran le programme doit commencer par la mise à zéro de la variable système DF-SZ soit :

en langage machine : LD (IY + 22), 00 FD 36 22 00

en Basic : POKE 16418, 0

CHAPITRE XIV

PROGRAMME MONITEUR ROUTINES UTILISABLES

Ce chapitre a un double objectif.

Le premier est de donner un aperçu d'ensemble du programme moniteur avec lequel vous avez eu de brèves rencontres au cours des pages précédentes.

Les différentes parties fonctionnelles seront présentées. Si vous disposez d'un désassembleur les adresses citées pourront vous aider dans vos recherches. N'espérez pas, cependant, percer tous les secrets du programme.

Le second objectif est essentiellement d'intérêt pratique.

Vous apprendrez à utiliser certains sous-programmes du sytème. C'est avantageux pour deux raisons :

- économie de mémoire. Les routines, parfois encombrantes, sont déjà logées en mémoire morte;
- économie de temps de programmation. Ce qui est fait n'est plus à faire.

Dans la plupart des cas, il ne suffit pas de connaître l'adresse d'appel de la routine, il faut aussi savoir comment elle fonctionne. Quels sont ses registres de travail, les paramètres dont elle a besoin et comment recueillir éventuellement les résultats.

Les exercices d'application seront utiles.

Le programme moniteur se compose de 14 parties principales. Les adresses sont en hexadécimal.

1.	Routines RST	0000
2.	Tables des caractères du clavier	007E
3.	Routines d'affichage - lecture clavier	0207
4.	Routines SAVE et LOAD	02F6
5.	Routine d'initialisation	03CB
6.	Routines d'édition Basic	0419
7.	Tables des commandes Basic	0C29
8.	Interpréteur Basic	0CBA
9.	Routines des commandes Basic	0DAB
10.	Evaluateur d'expressions Basic	0F55
11.	Manipulation nombres virgule flottante	14CE
12.	Table des fonctions (calculateur)	1915
13.	Calculateur virgule flottante	199D
14.	Générateur de caractères	1E00

Les six premières parties contiennent toutes les routines intéressantes à l'exception de FAST et SLOW.

Nous verrons successivement :

RST 08 - compte-rendu	0008
RST 10 - affichage d'un caractère	0010
Routine principale d'affichage (PAUSE)	0229
Scrutation du clavier	02BB
SAVE	02F6
LOAD	0340
NEW	03C3
Décodage du clavier	07BD
PRINT AT	08F5
CLS	0A2A
Affichage nombre décimal	0AAB
Affichage chaîne de caractères	0B6B
PLOT et UN PLOT	0BB2
FAST	0F23
SLOW	0F2B

LES ROUTINES RST.

Ces sous-programmes en page zéro ont été évoqués au chapitre X. Leur utilisation est rappelée ci-dessous :

RST 00 - Mise en route.

Le fait d'appeler ce sous-programme a le même effet que de débrancher puis rebrancher la prise de courant.

RST 08 - Compte-rendu.

Cette routine est utilisée par le système pour afficher les comptes-rendus.

Le code hexadécimal de l'instruction d'appel est de la forme CF n où n représente une constante sur un octet qui détermine le code du compte-rendu. Ainsi si n = 00 h le compte rendu d'erreur 1 est affiché. Si n = 09 c'est l'erreur A, si n = 0E, c'est l'erreur F.

Le programmeur peut trouver intéressant d'utiliser cette routine pour afficher ses propres codes d'erreur.

Le fonctionnement de RST 08 est montré par le programme Basic ci-dessous :

```
10  POKE 16507, 207          CF
20  POKE 16508, 19           13
30  LET K = USR 16507
40  PRINT "HELLO"
```

On s'aperçoit que la ligne 40 n'est pas exécutée car la routine a affiché le compte-rendu K à la ligne 30.

RST 10.

Nous avons abondamment utilisé cette routine d'affichage d'un caractère dans les chapitres précédents.

Rappelons que l'accumulateur doit être chargé au préalable avec le code du caractère à afficher. Le code de l'instruction d'appel est D7.

Le caractère est toujours placé au prochain emplacement disponible du fichier d'affichage. Si le fichier est vierge, ce sera à (D.FILE) + 1 c'est-à-dire à l'adresse détenue par la variable système DF.CC (16398 - 16399 - 400E hexa).

(DF.CC) peut être modifiée comme le montre le programme suivant (41 octets) :

```
30000          LD HL, (D.FILE)        2A 0C 40
               LD DE, 0195            11 95 01
               ADD HL, DE             19
               LD (DF.CC), HL         22 0E 40
30010          CALL AFFICHAGE         CD 4F 75
               DEFM. 16 octets        BONJOUR LES AMIS
               DEFB                   FF
               RET                    C9
30031 AFF.     POP HL                 E1
               LD A, (HL)             7E
               INC HL                 23
               PUSH HL                E5
               CP FF                  FE FF
               RET Z                  C8
               RST 10                 D7
               JR. AFF.               18 F6
```

Faites en commande directe LET K = USR 30010
puis maintenant LET K = USR 30000

Cette fois encore, attention de ne pas écraser un newline sous peine de blocage.

Les autres routines RST sont de peu d'intérêt pour le programmeur bien que très importantes pour le système.

RST 18 recueil d'un caractère d'une ligne Basic

RST 20 même chose caractère suivant

RST 28 point d'entrée pour le calculateur

RST 30 fait de la place en mémoire

RST 38 interruption mode 1

RST 66 interruption «non masquable» pour l'affichage en mode SLOW.

LES TABLES DES CARACTÈRES DU CLAVIER.

Nous avons déjà parlé de ces tables au chapitre IX où vous retrouverez le programme Basic qui permet de les consulter.

Remarquez que les commandes d'édition ainsi que les touches de changement de mode sont affichées sous forme de points d'interrogation.

007E - 00CB caractères normaux ou avec Shift (mode L)

00CC - 00F2 fonctions (mode F)

00F3 - 0110 caractères graphiques (mode G)

0111 - 01FB mots-clés (mode K)

LES ROUTINES D'AFFICHAGE.

Les routines comprises entre les adresses 0207 et 02BA sont utilisées pour l'affichage à l'écran.

• La routine en 0207 permet de déterminer quel mode (lent ou rapide) est en fonctionnement.

• La routine en 0229 est la routine principale d'affichage. Elle peut être appelée à partir d'un programme basic (USR 553) ou en langage machine (Call 0229). Elle n'est pas utilisable en mode SLOW. En Basic elle est sans effet, en code machine elle peut causer un crash.

Utilisée en mode FAST, elle maintient l'affichage indéfiniment. Le fait d'appuyer sur une touche fait sortir de la routine.

La commande PAUSE l'utilise pour maintenir l'affichage pendant une période déterminée.

En langage machine on peut aussi régler la durée de la PAUSE en donnant une certaine valeur à la variable système FRAMES (en 16436 - 16437) compteur de trames TV.

Le programme suivant montre l'utilisation de cette routine.

adresse 30000	32 octets	
LD A, (4021)	3A 21 40	recueil valeur pokée 16417
LD C, A	4F	transférée dans C
LD HL, (D.FILE)	2A 0C 40	adresse fichier
LD DE, 02C0	11 C0 02	Nombre de caractères
INC HL	23	emplacement suivant
LD A, (HL)	7E	Test Newline
CP 76	FE 76	
JRZ - 6	28 FA	au suivant si N/L
LD (HL), C	71	écriture
DEC DE	1B	déc compteur DE
LD A, D	7A	
OR E	B3	Test si DE = 0
JR NZ - 12	20 F4	suivant
LD HL, 0040	21 40 00	
LD (FRAMES), HL	22 34 40	compteur trames 4034
CALL PAUSE	CD 29 02	routine affichage
RET	C9	

Partie Basic :

```
10  FAST
20  LET C = 1
30  POKE 16417,C
40  LET K = USR 30000
50  LET C = C + 1
60  IF C = 11 THEN LET C = 128
70  IF C = 138 THEN STOP
80  GOTO 30
```

Faites RUN. Ne touchez pas au clavier avant le compte-rendu de fin de programme.

Maintenant supprimez les deux instructions LD HL, 0040 et LD (FRAMES), HL (6 octets). Vous pouvez faire SLOW puis RUN 199, option affichage 30000, 32 octets et remplacer le code par 00 aux adresses 30022 à 30027.

Faites STOP puis RUN.

L'affichage est maintenu indéfiniment. Il faut appuyer sur une touche pour sortir de la routine et changer d'image.

Deux autres routines, en 0292 et en 02B5, sont utilisées par le système en relation avec l'affichage. Nous n'en parlerons pas.

LA ROUTINE DE SCRUTATION DU CLAVIER.

La routine principale d'affichage accomplit trois tâches :

- Décrémentation et test du compteur de pause
- Lecture du clavier
- Affichage

La routine de scrutation du clavier qu'elle utilise est très importante. Elle commence en 02BB et elle est accessible directement en appelant cette adresse : CALL 02BB - CD BB 02

Nous allons étudier le principe de fonctionnement de cette routine.

Vous savez qu'elle a pour effet de charger le registre HL avec une certaine valeur, différente suivant la touche enfoncée.

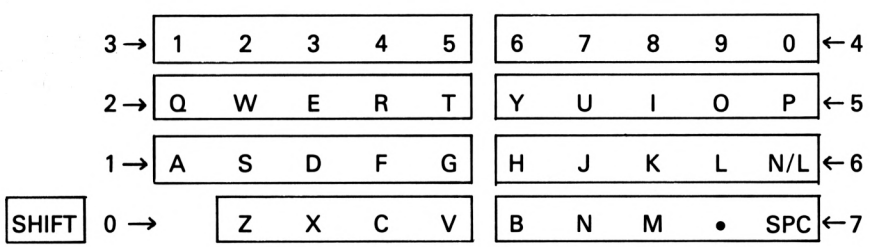
HL étant initialisé avec FFFF en début de routine, il conservera cette valeur si aucune touche n'est enfoncée.

En fait deux valeurs séparées sont élaborées : l'une formera le contenu de H, l'autre celui de L.

Voyons d'abord comment la valeur de L est déterminée.

Imaginez l'ensemble du clavier, à l'exception de SHIFT, divisé en huit groupes horizontaux contenant chacun cinq touches sauf un qui n'en comporte que quatre (SHIFT, étant exclus).

Chacun des groupes est numéroté de 0 à 7 conformément au schéma ci-dessous :



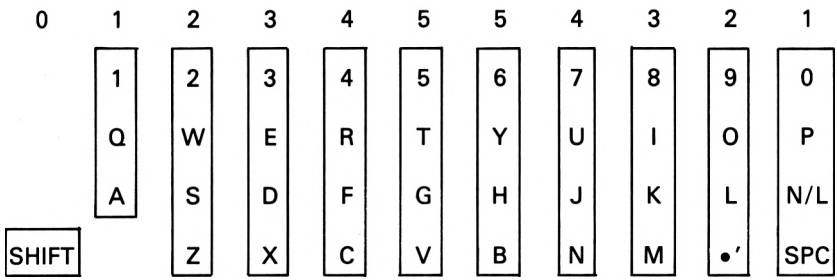
Si aucune touche n'est enfoncée L contient la valeur FF. Si une touche est enfoncée le bit correspondant au numéro du groupe dont fait partie la touche est mis à zéro. L peut ainsi contenir une des valeurs suivantes :

Si aucune touche n'est enfoncée	11111111	FF
Si une touche du groupe 0 est enfoncée	11111110	FE
Si une touche du groupe 1 est enfoncée	11111101	FD
Si une touche du groupe 2 est enfoncée	11111011	FB
Si une touche du groupe 3 est enfoncée	11110111	F7
Si une touche du groupe 4 est enfoncée	11101111	EF
Si une touche du groupe 5 est enfoncée	11011111	DF
Si une touche du groupe 6 est enfoncée	10111111	BF
Si une touche du groupe 7 est enfoncée	01111111	7F

Si la touche Y est enfoncée par exemple, la valeur en L sera DF = 11011111

La valeur inscrite dans le registre H est déterminée d'une manière analogue. Cette fois le clavier est divisé en 11 groupes verticaux numérotés de 0 à 5 de la gauche vers la droite et de 1 à 5 de la droite vers la gauche.

La touche SHIFT formant le groupe 0 à elle seule.



Si aucune touche n'est enfoncée le registre H contient FF.

Si une touche est enfoncée le bit correspondant au groupe dont fait partie la touche est mis à zéro.

Si SHIFT est enfoncée le bit zéro de H est mis à zéro.

SHIFT 0 (RUBOUT) par exemple, donnera 11111100 soit FC.

Le registre H peut ainsi contenir une des valeurs suivantes :

	sans shift		avec shift	
aucune touche	11111111	FF		
groupe zéro (SHIFT seul)			11111110	FE
groupe un	11111101	FD	11111100	FC
groupe deux	11111011	FB	11111010	FA
groupe trois	11110111	F7	11110110	F6
groupe quatre	11101111	EF	11101110	EE
groupe cinq	11011111	DF	11011110	DE

Vous conviendrez aisément que le fait d'appuyer sur une touche ne peut résulter qu'en une seule et unique valeur dans HL. Si deux touches sont dans le même groupe horizontal, elles seront forcément dans deux groupes verticaux différents.

Le programme qui suit utilise la routine de scrutation du clavier. Il vous permettra de vérifier ce que nous venons de voir.

Les contenus des registres L d'abord puis H sont affiché en binaire puis la valeur décimale de la touche (HL).

Vous trouverez une table des valeurs des touches en annexe.

Entrez le programme machine :

adresse 30000	25 octets	
CALL CLAVIER	CD BB 02	
LD C, L	4D	L chargé en C
LD D, 02	16 02	2 octets
LD B, 08	06 08	8 bits
XOR A	AF	
SLA C	CB 21	affichage
ADC A, 1C	CE 1C	8 bits
RST 10	D7	
DJNZ - 8	10 F8	
XOR A	AF	espace
RST 10	D7	
LD C, H	4C	H chargé en C
DEC D	15	
JR NZ - 17	20 EF	2e octet
LD B, H	44	transfert dans
LD C, L	4D	BC pour affichage
RET	C9	par PRINT USR

Partie Basic :

```
10  PRINT "APPUYEZ SUR UNE TOUCHE"
20  PAUSE 200
30  PRINT USR 30000
40  GOTO 20
```

Remarques :

La touche SHIFT n'arrête pas la pause mais sa valeur s'affiche à la fin de la pause si vous la maintenez enfoncée.

La valeur de la touche SPC ne peut pas être affichée. Il y a arrêt du programme Basic par BREAK (compte rendu D).

Si aucune touche n'est enfoncée, la valeur 65535 est affichée à la fin de la pause.

Voici le listing complet de la routine de scrutation du clavier :

02BB	21 FF FF	LD HL, FFFF	HL initialisé
02BE	01 FE FE	LD BC, FEFE	Port d'entrée
02C1	ED 78	IN A, (C)	lecture
02C3	F6 01	OR 01	
02C5	F6 E0	OR E0	élaboration
02C7	57	LD D, A	valeur touche
02C8	2F	CPL	dans
02C9	FE 01	CP 01	HL
02CB	9F	SBC A, A	
02CC	B0	OR B	
02CD	A5	AND L	

```
02CE    6F          LD L, A
02CF    7C          LD A, H
02D0    A2          AND D
02D1    67          LD H, A
02D2    CB 00       RLC B
02D4    ED 78       IN A, (C)       lecture
02D6    38 ED       JR C, 02C5      8 fois
02D8    1F          RRA
02D9    CB 14       RL H            valeur touche = HL
02DB    17          RLA
02DC    17          RLA             variable
02DD    17          RLA             système
02DE    9F          SBC A, A        MARGIN
02DF    E6 18       AND 18          (standard TV)
02E1    C6 1F       ADD A, 1F
02E3    32 28 40    LD (4028),A
02E6    C9          RET
```

Il faut insister sur l'intérêt que présente l'utilisation de cette routine. Le contenu du registre HL peut être comparé à la valeur d'une touche particulière. Il est ainsi possible d'intervenir dans le déroulement d'un programme en langage machine à partir du clavier. Attention, cependant, le contenu de tous les registres est perdu. On peut le sauvegarder dans la pile avant l'appel (PUSH) et le récupérer au retour par POP si nécessaire.

LES ROUTINES SAVE ET LOAD.

La routine SAVE commence à l'adresse 02F6.

Les premières lignes sont :

```
   02F6     DC A8 03    CALL 03A8     Vérification nom
   02F9     38 F9       JR C, 02F4    si pas de nom
   02FB     EB          EX DE, HL     erreur 9
763-02FC    11 CB 12    LD DE, 12CB   silence 5 sec.
```

La routine LOAD commence en 0340

```
   0340     CD A8 03    CALL 03A8     Vérification nom
   0343     CB 12       RL D
   0345     CB 0A       RRC D
839-0347    CD 4C 03    CALL 034C     début lecture
```

Les deux routines ont un sous-programme commun (en 01FC) où elles récupèrent la valeur de E.LINE - Fin de la zone à sauvegarder ou à charger.

Les routines SAVE et LOAD sont compliquées. Nous nous contenterons de souligner une particularité.

Toutes deux commencent par une vérification portant sur le nom du programme. On peut sauter ces premières lignes.

Si cela vous amuse faites l'expérimentation suivante :

Le PCCM étant en mémoire, mettez votre magnétophone en position enregistrement.

Faites FAST puis LET K = USR 763.

Le programme est sauvegardé sans nom.

Faites NEW - Rembobinez.

Position lecture.

Faites FAST puis LET K = USR 839.

Si le système n'y met pas de la mauvaise volonté, le programme devrait être rechargé correctement.

LA ROUTINE D'INITIALISATION.

Les différentes parties de la routine d'initialisation sont :

1. La vérification de la mémoire vive disponible.
2. L'installation de la pile.
3. Le chargement du registre I.
4. La sélection du mode d'interruption IM1
5. Le chargement du registre index IY.
6. La mise en mode «SLOW»
7. La mise en place d'un fichier d'affichage minimum.

Nous avons évoqué ces différentes parties, ici ou là, au cours des pages précédentes.

Il faut cependant préciser le rôle du registre I dans le système ZX 81.

Au chapitre précédent nous avons vu que ce registre du microprocesseur Z 80 A servait normalement lors des interruptions en mode 2.

Ce mode n'est pas utilisé sur SINCLAIR et le registre I a un autre usage. Il contient l'octet de poids fort de l'adresse des caractères dans le générateur de caractères en 1E00.

Le registre I est ainsi chargé avec l'octet de poids fort de cette adresse de base au cours de la routine d'initialisation.

```
03F2        3E 1E       LD A, 1E
03F4        ED 47       LD I, A
```

NEW

La commande Basic NEW entre dans la routine d'initialisation à l'adresse 03C3. En langage machine on obtient le même résultat par un simple saut à cette adresse JP 03 C3. La valeur de RAMTOP peut être modifiée auparavant pour préserver des données en haut de la mémoire vive.

C'est évidemment d'un intérêt limité car le programme s'arrête. Il faut le relancer à partir du Basic, en commande directe ou à l'aide d'un nouveau programme.

Cette facilité ne peut être utile que pour des manipulations.

LES ROUTINES D'ÉDITION BASIC.

Cette partie du programme moniteur s'étend des adresses 0454 à 0C28.

On y trouve un nombre important de routines système. Il serait fastidieux de les citer toutes. Seules les routines utilisables seront présentées.

LA ROUTINE DE DÉCODAGE DU CLAVIER.

Le mécanisme sommaire de cette routine a été expliqué au chapitre VIII dans le paragraphe concernant l'instruction ADD. Elle peut être appelée par CALL 07BD.

Disons pour résumer qu'elle accomplit la tâche suivante. La valeur d'une certaine touche lui étant fournie, dans le registre BC, elle détermine (dans la table de caractères) l'adresse du caractère correspondant. Le résultat se trouve dans HL.

Le listing complet de cette routine est le suivant :

```
07BD    16 00       LD D, 00
07BF    CB 28       SRA B
07C1    9F          SBC A, A
07C2    F6 26       OR 26
07C4    2E 05       LD L,05
07C6    95          SUB L
07C7    85          ADD A, L
07C8    37          SCF
07C9    CB 19       RR C
07CB    38 FA       JR C, 07C7
07CD    0C          INC C
07CE    C0          RET NZ
07CF    48          LD C, B
07D0    2D          DEC L
07D1    23 01       LD L, 01
07D3    20 F2       JR NZ, 07C7
07D5    21 7D 00    LD HL, 007D
07D8    5F          LD E, A
07D9    19          ADD HL, DE
07DA    37          SCF
07DB    C9          RET
```

Le programme ci-dessous montre l'utilisation de la routine de décodage associée à la routine de scrutation du clavier.

```
                adresse 30000       25 octets

DEBUT           CALL CLAVIER        CD BB 02
                INC L               2C
                JR NZ - 6           20 FA
ATTENTE         CALL CLAVIER        CD BB 02
                LD B, H             44
                LD C, L             4D
                LD D, C             51          Test touche
                INC D               14          enfoncée
                JR Z - 9 ATTENTE    28 F7       non, attente
                CALL DECODE         CD BD 07
                LD A, (HL)          7E
                CP 40               FE 40       63 premiers codes caractères
                RET NC              D0
                RST 10              D7
                JR DEBUT            18 E7
```

Faites en commande directe : LET K = USR 30000.

Vous pouvez utiliser le clavier comme celui d'une machine à écrire.

Ce programme fonctionne selon le même principe que la fonction INKEY$.

Les trois premières lignes et les instructions LD D, C et INC D demandent quelques explications.

Avant que nous n'appuyiez sur une touche le programme est en attente à la 4e ligne. En effet, si aucune touche n'est enfoncée le contenu de L est FF. INC L le fait passer à zéro. INC D en fait de même pour D.

Dès qu'une touche est enfoncée, D devient différent de zéro et la routine de décodage intervient. La dernière ligne renvoie à la première. Là, la situation ne peut se débloquer que si vous cessez d'appuyer sur la touche de façon à ce que L = FF.

L'instruction CP 40 provoque le retour au Basic si le code caractère (dont l'adresse dans la table est pointée par HL) est supérieur à 3F. Retour si pas de retenue.

LA ROUTINE PRINT AT.

Cette routine, située en 08F5, permet de modifier la position d'affichage à l'écran.

Avant l'appel - CALL 08F5 - CD F5 08 - les registres B et C doivent contenir respectivement les numéros de ligne et de colonne désirés. Retenez C = colonne. Ces paramètres sont testés en début de routine (voir chapitre X - sauts conditionnels sur indicateur C).

```
30000   LD BC, 0A0C       01 0C 0A        colonne 12, ligne 10
        CALL PRINT AT     CD F5 08
        RET               C9
```

```
10 LET K = USR 30000
20 PRINT "HELLO"
30 STOP
```

On peut reprendre le programme de RST 10 :

```
        30000   LD BC, 0A08       01 08 0A
                CALL PRINT AT     CD F5 08
        30006   CALL AFFICHAGE    CD 4B 75
                DEFM.             BONJOUR LES AMIS
                DEFB              FF
                RET               C9
AFFICH. 30027   POP HL            E1
                LD A, (HL)        7E
                INC HL            23
                PUSH HL           E5
                CP FF             FE FF
                RET Z             C8
                RST 10            D7
                JR AFF.           18 F6
```

Faites LET K = USR 30006
puis LET K = USR 30000

LA ROUTINE CLS.

Quand l'interpréteur Basic exécute la commande CLS il se contente d'appeler cette routine en 0A2A.

Elle est aussi facile à utiliser en langage machine. La simple instruction CALL 0A2A a pour effet de vider l'écran.

LA ROUTINE D'AFFICHAGE D'UN NOMBRE DÉCIMAL.

Cette routine est à l'adresse 0AAB. Elle est utilisée par le système pour l'affichage des numéros de lignes Basic dans un listing.

Les nombres décimaux compris entre 0 et 9999 peuvent être affichés.

Le registre HL doit être chargé avec le nombre à afficher avant l'appel par CALL 0AAB.

L'intérêt de cette routine est assez limité car le retour au Basic est automatique.

adresse 3000	10 octets
LD HL, 0700	21 00 07
LD DE, 0F00	11 00 0F
ADD HL, DE	19
CALL 0AAB	CD AB 0A

Basic :

```
10  LET K = USR 30000
20  STOP
```

Remarquez l'absence d'instruction de retour au Basic.

Faites RUN.

Supprimez maintenant la ligne 20.

Faites RUN.

Faites STOP puis en commande directe LET K = USR 30000.

C'est le crash. Le programme ne sait plus où aller.

LA ROUTINE D'AFFICHAGE D'UNE CHAINE DE CARACTÈRES.

Nous avons déjà utilisé cette routine située en 0B6B.

Les registres de travail sont DE et BC qui doivent être chargés avant l'appel par CALL 0B6B.

DE doit contenir l'adresse de début de la chaîne de caractères à afficher et BC le nombre de caractères.

adresse 30000

```
        LD BC, 0A06          01 06 0A
        CALL PRINT AT        CD F5 08
        LD DE, DEBUT         11 40 75
        LD BC, NB CARAC.     01 11 00
        CALL CHAINE          CD 6B 0B
        RET                  C9
30016   DEFM (17 octets)     VIVE LES VACANCES
```

Faites LET K = USR 30000

LA ROUTINE PLOT ET UNPLOT.

Cette routine fonctionne suivant le même principe que la routine PRINT AT.

Les coordonnées du point sont chargées dans le registre BC.

X (de 0 à 63) est chargé dans C.
Y (de 0 à 43) est chargé dans B.

Notez que ces paramètres ne sont pas déterminés de la même façon que pour PRINT AT (revoyez le chapitre 18 du manuel SINCLAIR ZX 81).

Ils sont aussi testés pour validité en début de routine.

La même routine sert pour les deux commandes. Une comparaison permet de déterminer quelle commande doit être exécutée. La comparaison est faite entre la valeur de la variable système T.ADDR (en 4030) qui détient l'adresse de la rubrique suivante dans la table de syntaxe et l'adresse dans cette même table de la commande UNPLOT.

```
0BDA    11 9E 0C      LD DE, 0C9E adresse UNPLOT
0BDD    3A 30 40      LD A, (T.ADDR)
0BE0    93            SUB E
0BE1    FA E9 0B      JP M, 0BE9 saut à PLOT
0BE4    UNPLOT
...
0BE9    PLOT
```

Seuls les octets de poids faible sont comparés, les octets de poids fort étant les mêmes.

Pour résumer, avant l'appel de la routine par CALL 0BB2 :

- le registre C doit contenir l'abscisse du point
- le registre B doit contenir l'ordonnée du point.
- L'octet de poids faible (en 4030 - 16432 d) de la variable système T.ADDR doit avoir une valeur inférieure à 9E pour PLOT, supérieure à 9E pour UNPLOT.

Notez que cette routine détruit le contenu des registres. Si des modifications sont à faire aux paramètres en cours de programme, il convient de les stocker dans des emplacements mémoire ou d'utiliser la pile.

L'exercice proposé vous montre les résultats du même programme traité en Basic ou en langage machine.

Le programme Basic occupe 64 octets en mémoire (ligne 40 incluse).

Le programme en langage machine fait 29 octets. Une autre version, qui stocke les paramètres dans des emplacement mémoire occupe 41 octets. Celle-ci serait à retenir si X et Y devaient varier en même temps et si leur valeur devait s'inverser (balle rebondissant dans une cage).

	adresse 30000	29 octets	
	LD A, 01	3E 01	
NOUVEL X	LD B, 16	06 16	Y = 20
	LD C, A	4F	X
	PUSH AF	F5	sauvegarde A
	PUSH BC	C5	sauvegarde BC
	LD (IY + 30), 90	FD 36 30 90	T.ADDR < 9E
	CALL PLOT	CD B2 0B	PLOT
	POP BC	C1	récup. BC
	LD (IY + 30), A0	FD 36 30 A0	T.ADDR > 9E
	CALL PLOT	CD B2 0B	UNPLOT
	POP AF	F1	récup. A
	CP 3F	FE 3F	bord droit écran
	RET Z	C8	Retour Basic
	INC A	3C	x = x + 1
	JR. NOUVEL X	18 E5	

Programme Basic :

```
10  FOR X = 0 TO 63
20  PLOT X, 20
30  UNPLOT X, 20
40  NEXT X
50  PRINT "APPUYEZ SUR UNE TOUCHE"
60  PAUSE 1000
70  LET K = USR 30000
80  CLS
90  STOP
```

Voici l'autre version :

	LD A, 16	3E 16
	LD (407C), A	32 7C 40
	LD A, 01	3E 01
NOUVEL X	LD (407B), A	32 7B 40
	LD BC, (407B)	ED 4B 7B 40
	LD (IY + 30),90	FD 36 30 90
	CALL PLOT	CD B2 0B
	LD BC, (407B)	ED 4B 7B 40
	LD (IY + 30), A0	FD 36 30 A0
	CALL PLOT	CD B2 0B
	LD A, (407B)	3A 7B 40
	CP 3F	FE 3F
	RET Z	C8
	INC A	3C
	JR NOUVEL X	18 DE

LES TABLES DES COMMANDES BASIC.

Ces tables commencent en 0C29.

La première est une table index qui permet d'entrer dans la table de syntaxe.

La table de syntaxe est utilisée par l'interpréteur. Elle donne pour chaque commande :

- la classe de la commande qui détermine si la commande doit ou non être suivie de paramètres et leur nature;
- le code caractère des signes de ponctuation;
- l'adresse de la routine à exécuter.

L'INTERPRÉTEUR BASIC.

Cette partie importante du programme moniteur débute en 0CBA par la routine de vérification des lignes Basic.

Chaque commande est identifiée, les renseignements nécessaires à son exécution sont trouvés dans la table de syntaxe et un appel est fait à la routine appropriée.

LES ROUTINES DES COMMANDES BASIC.

On trouve dans cette partie du programme la plupart des routines des commandes.

La plupart mais pas toutes. C'est le cas des commandes NEW, SAVE et LOAD que nous avons vues.

Les routines FAST et SLOW s'y trouvent et sont utilisables par le programmeur.

LES ROUTINES FAST et SLOW.

Elles ont pour effet de positionner certains bits de la variable système CD.FLAG (en 403B - 16443) permettant l'entrée dans l'un ou l'autre des deux modes.

La routine FAST met à zéro les bits 6 et 7.

La routine SLOW met à un le bit 6.

Une simple instruction d'appel permet le choix du mode par un programme en langage machine.

CALL 0F23 pour FAST
CALL 0F2B pour SLOW

L'ÉVALUATEUR D'EXPRESSIONS - LE CALCULATEUR.

Ces dernières parties du programme sont compliquées. Les routines arithmétiques en virgule flottante peuvent être utilisées par le programmeur mais leur présentation entraînerait des développements qui dépassent le cadre de ce livre.

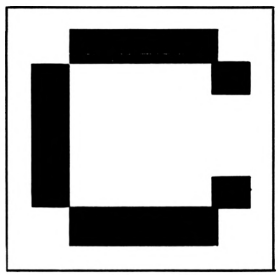
LE GÉNÉRATEUR DE CARACTÈRES.

Il occupe les 512 derniers octets de la ROM des adresses 1E00 à 1FFF.

Ce n'est pas un sous-programme mais une table, en réalité la plus longue table de la mémoire morte.

On y trouve la forme de tous les caractères affichables (64). Chaque symbole est stocké sur 8 octets. Voici par exemple le caractère C tel qu'il est stocké en binaire et tel qu'il apparaît à l'écran.

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0



Les caractères sont stockés dans l'ordre indiqué page 181 de votre manuel SINCLAIR (Annexe A). Ainsi l'espace (00) est à l'adresse 1E00 soit 7680 d. Chaque caractère occupant 8 octets dans le générateur, on peut retrouver l'adresse d'un caractère particulier.

Ainsi la lettre A, 38e caractère aura comme adresse 7680 + (38 x 8) = 7984.

On s'aperçoit qu'il suffit d'ajouter à l'adresse de base du générateur de caractères, le code caractère choisi multiplié par 8 pour trouver son adresse.

Le programme suivant est basé sur cette remarque. Vous donnez à n la valeur d'un code caractère comprise entre 00 et 3F. Ce code est multiplié par 8 à l'aide de trois instructions RLA. Le résultat est ajouté à l'adresse de base de la table 1E00.

Le caractère est ensuite agrandi 8 fois. Chaque bit de l'octet traité est remplacé par un caractère, un blanc pour un bit égal à zéro, un carré plein (code hexa 80) pour un bit égal à un. Ainsi pour les huit bits des huits octets représentant le caractère.

adresse 30000	42 octets	
LD A, code carac.	3E n	(entre 00 et 3F)
AND A	A7	indicateur C = 0
RLA	17	X 2
RLA	17	X 4
RET C	D8	retour Basic si n > 3F
RLA	17	X 8
LD D, 00	16 00	code caractère
RL D	CB 12	multiplié par 8
LD E, A	5F	dans DE
LD HL, 1E00	21 00 1E	adresse de base
ADD HL, DE	19	addition
LD D,08	16 08	8 octets
LD B, 08	06 08	8 bits
LD E, (HL)	5E	octet dans E
AND A	A7	C = 0

SLA E	CB 23	décalage
JR C+ 3	38 03	Si bit = 1
XOR A	AF	A = 0 pour
JR + 2	18 02	affichage espace
LD A, 80	3E 80	curseur
RST 10	D7	affiché
DJNZ - 13	10 F3	bit suivant
LD A, 76	3E 76	Newline
RST 10	D7	
INC HL	23	octet suivant
DEC D	15	D = compteur octets
JR NZ - 23	20 E9	
RET	C9	retour Basic

Vous allez sans doute vous dire et vous aurez raison : Comment ! on m'a dit que l'instruction SLA servait à une multiplication par deux et c'est RLA qui est utilisée.

C'est un cas particulier. Comme le nombre à décaler est inférieur à 3F (00111111) la retenue ne risque pas d'être ajoutée au résultat. RLA, un octet, est plus économique que SLA A qui en fait deux. AND A annule la retenue éventuelle.

Remarquez l'instruction RET C qui provoque l'arrêt du programme si le code caractère est supérieur à 3F (64 x 4 = 256 (retenue)).

A la première ligne du programme, vous chargez l'accumulateur à la main. Rien de plus facile que de le faire à partir du clavier grâce aux routines que nous avons étudiées dans les pages précédentes.

Nous en resterons là après le programme qui suit. La voie vous est ouverte. La programmation est assurément aussi passionnante que les mots croisés. Dites-vous qu'il y a toujours une solution.

adresse 30000	59 octets
CALL CLAVIER	CD BB 02
INC L	2C
JR NZ - 6	20 FA
CALL CLAVIER	CD BB 02
LD B, H	44
LD C, L	4D
LD D, C	51
INC D	14
JR Z, - 9	28 F7
CALL DECODE	CD BD 07
LD A, (HL)	7E
AND A	A7
RLA	17
RLA	17
RET C	D8
RLA	17
LD D, 00	16 00
RL D	CB 12
LD E, A	5F
LD HL, 1E00	21 00 1E
ADD HL, DE	19

LD D, 08	16 08	
LD B, 08	06 08	
LD E, (HL)	5E	on pourrait aussi
AND A	A7	bien utiliser
SLA E	CB 23	RL E CB 13 ou
JR C + 3	38 03	RLC E CB 03
XOR A	AF	
JR + 2	18 02	
LD A, 80	3E 80	
RST 10	D7	
DJNZ - 13	10 F3	
LD A, 76	3E 76	
RST 10	D7	
INC HL	23	
DEC D	15	
JR NZ, - 23	20 E9	
RET	C9	

Partie Basic :

```
10  LET K = USR 30000
20  PAUSE 100
30  CLS
40  GOTO 10
```

CHAPITRE XV

IMPLANTATION DES PROGRAMMES

Il faut bien l'admettre. Le ZX 81 n'a pas été conçu en vue de la programmation en langage machine. Il lui manque la plupart des facilités indispensables.

La première difficulté est due à l'absence d'un programme de chargement en hexadécimal à partir du moniteur. Elle oblige à s'équiper d'un utilitaire encombrant.

La seconde, plus importante, est liée à l'impossibilité de sauvegarder le code machine placé en dehors de la zone Basic.

Nous avons utilisé le haut de la mémoire vive pour la quasi-totalité des exercices de ce livre. C'est un endroit paisible. On peut même le protéger en abaissant la valeur de RAMTOP. Malheureusement les programmes qui y sont implantés ne peuvent pas être sauvegardés sur cassette.

Cela est dû aux caractéristiques des routines SAVE et LOAD qui n'agissent que sur une zone déterminée de la mémoire réservée au programme Basic, au fichier d'affichage et aux variables.

Il faut donc absolument loger un programme en langage machine dans cette zone.

Il y a trois méthodes que nous étudierons tour à tour.

Le moment est peut-être venu d'élaguer le PCCM si vous avez l'intention de le conserver pour un usage opérationnel.

Vous pouvez supprimer les lignes 2900 à 3290 et aménager les autres lignes en conséquence.

Un court programme témoin (12 octets) permettra de vérifier le résultat de nos manipulations qui seront nombreuses.

Le voici :

LD BC, 02C0	01 C0 02
LD A, 97	3E 97
RST 10	D7
DEC BC	0B
LD A, B	78
OR C	B1
JR NZ, - 8	20 F8
RET	C9

Vous remarquerez que c'est un programme relogeable. Il ne comporte pas de référence à une adresse mémoire.

Essayez-le à l'adresse familière 30000 que nous serons obligés de quitter (avec regret).

UTILISATION D'UNE INSTRUCTION REM.

Cette méthode consiste à réserver en début de zone du programme Basic un certain nombre d'octets correspondant à la longueur du programme à implanter. La ligne REM peut être composée de caractères quelconques. Si vous n'êtes pas court en mémoire, un conseil. Pendant la phase de mise au point, prévoyez large, mettez-y quelques dizaines d'octets supplémentaires. Vous pourrez toujours ajuster plus tard, grâce à une manipulation.

Puisque vous connaissez bien cette méthode, nous allons passer tout de suite à la pratique. Notre mémoire vive va être constellée d'astérisques inversés. C'est joli - Non !

Préparez une ligne REM - 20 caractères.

1 REM 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

Entrez la bannière étoilée, adresse 16514 - 12 octets

Faites 2 EXEC - Ça marche ? OK !

Maintenant faites RUN et entrez le code suivant :

adresse 31000 24 octets

(en 30000, il y a déjà quelque chose)

31000	LD HL, 4082	21 82 40
	LD DE, 7600	11 00 76
	LD BC, 000C	01 0C 00
	LDIR	ED B0
	RET	C9
31012	LD HL, 7600	21 00 76
	LD DE, 4082	11 82 40
	LD BC, 000C	01 0C 00
	LDIR	ED B0
	RET	C9

Ces deux programmes serviront à transférer le code en REM à l'adresse 7600 (30208 d) et vive-versa.

Faites STOP puis en commande directe LET K = USR 31000

Faites LIST

Supprimez la ligne REM.

Remplacez-la par une ligne vierge, 12 octets cette fois :

1 REM 1 2 3 4 5 6 7 8 9 0 1 2

Faites LET K = USR 31012
puis LET K = USR 16514.

Nous avons pu ajuster la longueur de la ligne REM grâce à cette manipulation.

Nous aurions aussi bien pu l'allonger. Le code étant ajouté au programme alors qu'il était en haut de mémoire.

Pourquoi ne pas utiliser EDIT direz-vous et rajouter des caractères à la ligne REM ?

C'est impossible dans la plupart des cas, à cause de 7E, ce séparateur indicateur de nombres dont nous avons déjà parlé.

C'est une coïncidence fâcheuse que ce séparateur ait le même code que LD A, (HL) une instruction très fréquemment utilisée.

Le code 7E n'apparaît pas au listing; s'il est présent dans un programme placé dans une instruction REM et que la commande EDIT est utilisée, 7E et les cinq octets qui suivent disparaîtront irrémédiablement.

Ce n'est pas le cas ici, vous pouvez utiliser EDIT sans risque.

Si une ligne REM doit être allongée et que EDIT ne peut pas être utilisée, il y a une autre méthode plus technique. Elle consiste à ajouter une deuxième ligne REM qui sera intégrée à la première. Pour cela il faut tromper l'interpréteur.

Ajoutez une deuxième ligne REM blanche : 2 REM

Faites RUN, option affichage.

adresse 16509 26 octets

On trouve en 16509 et 16510 le numéro de la ligne 1, en 16527 et 16528 le numéro de la ligne 2, en 16533 et 16534 le numéro de la ligne 199.

Chaque ligne se termine par un caractère newline.

Pour tromper l'interpréteur il suffit de modifier le nombre de caractères de la ligne en 16511 et 16512, de façon à lui faire croire que la première ligne se termine au deuxième caractère newline.

Ici il faut ajouter 6 à 0E en 16511, on obtient 14 + 6 = 20 = 14 hexa.

Faites la modification. Remplacez 0E par 14 en 16511.

La ligne 2 est toujours apparente au listing mais on peut la surcharger sans problème, ainsi que, et c'est préférable, le caractère Newline de la première ligne en 16526.

Retournez au menu, option entrée code hexa.

adresse 16526 6 octets

Entrez C9 C9 C9 C9 C9 C9

Retournez au menu, affichage 16509, 26 octets

La ligne 2 a disparu. Elle n'apparaît plus au listing.

Une ligne REM peut ainsi être allongée de six octets minimum. Si vous utilisez cette méthode, attention aux caractères Newline, le second surtout qui doit être préservé.

Pour raccourcir une ligne REM - six octets minimum - on effectue l'opération inverse.

Une ligne 2 est reconstituée. On peut l'effacer au clavier. Si cela vous amuse.

Un autre inconvénient peut se produire si le code machine stocké dans une instruction REM est trop important (un peu plus de 400 octets).

Vous savez que la routine d'affichage d'une ligne de Basic interprète le code caractère des «jetons» pour afficher le mot complet.

Il peut arriver qu'une ligne REM ainsi «gonflée» ne tienne plus sur l'écran au listing.

Une partie seulement est affichée et l'erreur 4 est indiquée. Dans ce cas il ne faut pas effacer la première ligne du programme, sous peine de perdre tout ce qui est en mémoire, la machine entrant dans une boucle sans fin.

MODIFICATION DU POINTEUR E.LINE.

Les routines SAVE et LOAD utilisent le sous-programme en 01FC pour recueillir la valeur de la variable système E.LINE.

Ce pointeur peut être modifié au moment de la sauvegarde sur cassette.

Le programme témoin est reproduit ci-dessous. Entrez-le aux adresses 20000, 21000 et 22000. Retournez au menu après chaque entrée

LD BC, 02C0	01 C0 02
LD A, 97	3E 97
RST 10	D7
DEC BC	0B
LD A, B	78
OR C	B1
JR NZ - 8	20 F8
RET	C9

Faites STOP. Conservez ou remettez en place la ligne 1 REM avec le programme témoin.

Entrez les lignes Basic suivantes. Elles auront pour effet de donner la valeur 24576 - 6000 hexa à la variable système E.LINE.

Pour la routine SAVE la zone à sauvegarder ira jusqu'à cette adresse.

```
10  POKE 16404, 0
20  POKE 16405, 96
30  SAVE "PCCM"
    (ou tout autre nom de votre choix)
```

Mettez votre magnétophone en position enregistrement puis faites RUN.

Il faut un certain temps pour sauvegarder les 8 K. Quand c'est terminé, débranchez votre ordinateur puis rebranchez-le.

Rembobinez le magnétophone, position lecture.

Faites LOAD "PCCM" en commande directe.

Votre programme est de retour prêt à l'emploi. Choisissez option 3 affichage, adresse 21000, 12 octets.

Le code machine est toujours en place.

Faites STOP puis successivement

```
LET K = USR 16514
LET K = USR 20000
LET K = USR 21000
LET K = USR 22000
```

Surtout ne faites pas LET K = USR 30000, sous peine de vous perdre dans l'espace. Il n'y a plus rien là-haut.

Cette méthode est intéressante pour la mise au point des programmes.

Pour éviter une fausse manœuvre vous pouvez supprimer les lignes 10, 20 et 30 ou mieux ajouter une ligne 5 GOTO 199; vous l'enlèverez avant une nouvelle sauvegarde.

Faites RUN 199 et entrez le programme témoin à l'adresse 23000.

Faites STOP puis LIST.

Supprimez la ligne 5. Assurez-vous que les lignes 10, 20 et 30 sont en place.

Ajoutez les lignes

```
40  LET K = USR 23000
50  STOP
```

Répétez la même procédure. Sauvegarde puis chargement.

Chargement terminé notre dessin apparaît attestant que le code machine ajouté a bien été sauvegardé.

Confirmez avec LET K = USR 23000.

Après mise au point le programme peut être transféré dans une instruction REM ou même laissé en place.

Je vous laisse découvrir les possibilités et les limites de cette méthode.

UTILISATION DE LA ZONE DES VARIABLES.

Le code machine peut être stocké sous forme d'un tableau de chaîne de caractères dans la zone des variables Basic dont l'adresse de début est détenue par la variable système VARS en 16400 - 16401.

L'espace requis doit être réservé en dimensionnant le tableau avec une instruction Basic de la forme DIM S$ (n) où n est le nombre d'octets de code machine à stocker.

Il faut six octets pour créer un tableau à une seule dimension.

L'adresse du premier octet sera :

6 + PEEK 16400 + 256 * PEEK 16401 à condition que 0$ soit le premier élément de la zone des variables. Ce sera le cas si DIM S$ (n) est la première des instructions DIM, FOR, INPUT ou LET à être exécutée depuis le dernier RUN ou CLEAR.

L'utilisation de cette méthode impose des programmes relogeables car la moindre modification du programme Basic provoque le déplacement de tout le code machine.

Toute utilisation de RUN ou CLEAR le fait disparaître de la mémoire.

Il nous faut un PCCM tout neuf. Sortez votre version originale, non trafiquée, et chargez-la.

Entrez les lignes suivantes :

```
10  LET S = 6 + PEEK 16400 + 256 * PEEK 16401
20  LET K = USR S
30  STOP
```

Ajoutez après la ligne 1000 CLEAR

```
1005 DIM S$ (12)
```

Modifiez la ligne 1090 comme suit :

```
1090 LET S = 6 + PEEK 16400 + 256 * PEEK 16401
```

La ligne 2010 doit être

```
2010 LET K = USR S
```

Faites RUN 199, option 1, entrée code hexa.

Une adresse s'affiche, c'est VARS + 6

Entrez le programme témoin - N.OCT = 12

```
01  C0  02  3E  97  D7  0B  78  B1  20  F8  C9
```

Vous pouvez faire EXEC, ça marche maintenant; GOTO 10 ça marche encore et maintenant RUN, rien.

Notre programme a disparu de la mémoire.

Faites RUN 199, option affichage, 12 octets.

La chaîne de caractères est vide.

Le fait que le code machine ne soit pas stable en mémoire et qu'il risque de s'envoler au moindre courant d'air rend cette méthode peu enthousiasmante.

Néammoins, elle permet une sauvegarde sur cassette. Elle est souvent utilisée dans ce seul but. Dès que le chargement en mémoire est effectué, le code machine est transféré à un endroit fixe déterminé.

C'est ce que devrait vous montrer le petit montage que je vous propose.

Entrez les lignes suivantes :

```
 1 REM 16 caractères
10 SAVE "PCCM"
20 LET K = USR 16514
30 PRINT "APPUYEZ SUR UNE TOUCHE"
40 PAUSE 10000
50 LET K = USR 30000
60 STOP
```

Le programme en zone des variables sera transféré par la sous-routine en 16514 à l'adresse 30000 d'où il sera exécuté.

Vérifiez ligne 1005 DIM S$ (12)
ligne 1090 INPUT S.

Faites RUN 199

adresse 16514 16 octets

Entrez le programme de transfert

LD HL, (VARS)	2A 10 40	
LD DE, 0006	11 06 00	
ADD HL,DE	19	début = VARS + 6
LD DE, 7530	11 30 75	
LD BC, 000C	01 0C 00	12 octets
LDIR	ED B0	
RET	C9	

Faites STOP et maintenant :

PRINT 6 + PEEK 16400 + 256 * PEEK 16401

Notez la valeur affichée, c'est VARS + 6 adresse où doit être implanté le code machine compte tenu de la dimension de la zone programme Basic.

Faites RUN 199.

A partir de maintenant, les RUN sont interdits.

Entrez le programme témoin

adresse VARS + 6 12 octets

01 C0 02 3E 97 D7 0B 78 B1 20 F8 C9

Faites STOP

Lancez l'enregistreur - GOTO 10

A la fin de l'enregistrement, appuyez sur la touche. Si le dessin apparaît, vous avez des chances.

Débranchez l'ordinateur. Rebranchez-le.

Rembobinez la cassette.

Faites LOAD "PCCM"

Ça marche. Bravo !

MISE EN PLACE DÉFINITIVE.

Il est rare qu'un programme tourne correctement au premier essai. La phase de mise au point peut être plus ou moins longue et difficile suivant la cause de non fonctionnement : erreur de frappe, erreur de codage, erreur de programmation ou erreur d'analyse.

Les deux premières sont relativement faciles à détecter et à corriger. Les autres demandent une recherche approfondie et imposent des modifications.

Une instruction de retour au Basic, judicieusement placée - quand c'est possible - peut vous permettre de déterminer l'endroit où se produit le crash.

Vérifiez systématiquement la validité des instructions de branchement avant et après modification.

Quand vous faites une sauvegarde, notez quelque part la nature de la dernière modification effectuée.

La zone mémoire utilisée pour la sauvegarde et la mise au point peut être différente de celle où sera placée le programme pour exécution. Attention aux instructions de branchement utilisant l'adressage absolu, ou indirect registre.

L'instruction LDIR est particulièrement utile pour les transferts entre zones et la mise en place définitive.

De nombreuses manipulations sont possibles, je n'en donnerai qu'un exemple.

La version définitive du programme est à l'adresse 25000. On désire la placer dans une instruction REM pour l'exécution.

La procédure peut être la suivante :

Modifiez toutes les instructions de branchement nécessaires.

Placez en haut de mémoire un programme de transfert

Exemple : adresse 30000

```
LD HL, 25000 d
LD DE, 16514 d
LD BC, nb. d'octets
LDIR
RET
```

Si on veut se débarrasser du programme utilitaire, abaisser RAMTOP à 25000

```
POKE 16388, 184
POKE 16389, 89
NEW
```

Préparez la ligne 1 REM nb. octets

Lancez le programme de transfert LET K = USR 30000.

Ajoutez les lignes Basic nécessaires.

Sauvegarder le programme complet. Après sauvegarde et nouveau chargement RAMTOP retrouve évidemment sa valeur initiale.

Afin de pouvoir vous en débarrasser plus rapidement le moment venu, vous trouverez sans doute utile d'ajouter ces quelques lignes à votre PCCM (voir paragraphe : utilisation d'une instruction REM).

```
PCCM SUPPLEMENT

5000 LET F=PEEK 16396+256*PEEK 16397
5010 PRINT"NOMBRE D OCTETS RESERVES EN REM?"
5020 INPUT N
5030 LET P=N+6
5040 IF N=0 THEN LET P=0
5050 LET L=F-(16513+P)
5060 LET H=INT (L/256)
5070 LET B=L-H*256
5080 PRINT AT 6,0;"NOTEZ QUELQUE PART"
5090 PRINT
5100 PRINT "POKE ";16511+P;",";B
5110 PRINT "POKE ";16512+P;",";H
5120 PAUSE 500
5130 PRINT AT 12,0;"C EST FAIT?"
5140 PRINT AT 16,0;"FAITES LE EN COMMANDE DIRECTE"
5150 PRINT "PUIS TAPEZ LE NUMERO DE
     LA  PREMIERE LIGNE A SUPPRIMER"
```

Faites RUN 5000

Si vous n'avez pas de ligne 1 REM ou que vous voulez la supprimer aussi. Entrez 0 pour le nombre d'octets. Ne faites pas d'erreur. Le PCCM disparaît entièrement. Vous ne pouvez pas en n'éliminer qu'une partie.

Le code machine que vous avez implanté est préservé mais attention s'il se trouve en zone des variables, il faut faire GOTO 5000.

Rappelez-vous, dans ce dernier cas le programme Basic adapté sera de la forme :

```
10  LET S = 6 + PEEK 16400 + 256 * PEEK 16401
20  LET K = USR S
```

et il faudra, bien sûr, utiliser GOTO 10

Entrez le programme suivant :

adresse 30000	50 octets
CALL 753E	CD 3E 75
DEFM	26 3A 00 37 2A 3B 34 2E 37
DEFB	FF
RET	C9
LD C, 14	0E 14
LD HL, (D.FILE)	2A 0C 40
LD DE, 018B	11 8B 01
ADD HL, DE	19
POP DE	D1
LD B, C	41
PUSH HL	E5
LD HL, 0200	21 00 02
DEC HL	2B
LD A, H	7C
OR L	B5
JR NZ, - 5	20 FB
POP HL	E1
LD (HL), 00	36 00
DEC HL	2B
LD A, (DE)	1A
CP FF	FE FF
RET Z	C8
LD (HL), A	77
DJNZ - 20	10 EC
INC DE	13
PUSH DE	D5
DEC C	0D
JR - 34	18 DE

LET K = USR 30000.

ANNEXE 1 - Z 80 - INSTRUCTIONS - CODAGE

mnémonique	hexadécimal
ADC A,n	CE XX
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC A,(HL)	8E
ADC A,(IX+d)	DD 8E XX
ADC A,(IY+d)	FD 8E XX
ADC HL,BC	ED 4A
ADC HL,DE	ED 5A
ADC HL,HL	ED 6A
ADC HL,SP	ED 7A
ADD A,n	C6 XX
ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85
ADD A,(HL)	86
ADD A,(IX+d)	DD 86 XX
ADD A,(IY+d)	FD 86 XX
ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29
ADD HL,SP	39
ADD IX,BC	DD 09
ADD IX,DE	DD 19
ADD IX,IX	DD 29
ADD IX,SP	DD 39
ADD IY,BC	FD 09
ADD IY,DE	FD 19
ADD IY,IY	FD 29
ADD IY,SP	FD 39

mnémonique	hexadécimal
AND n	E6 XX
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND (HL)	A6
AND (IX+d)	DD A6 XX
AND (IY+d)	FD A6 XX
BIT 0,A	CB 47
BIT 0,B	CB 40
BIT 0,C	CB 41
BIT 0,D	CB 42
BIT 0,E	CB 43
BIT 0,H	CB 44
BIT 0,L	CB 45
BIT 0,(HL)	CB 46
BIT 0,(IX+d)	DD CB XX 46
BIT 0,(IY+d)	FD CB XX 46
BIT 1,A	CB 4F
BIT 1,B	CB 48
BIT 1,C	CB 49
BIT 1,D	CB 4A
BIT 1,E	CB 4B
BIT 1,H	CB 4C
BIT 1,L	CB 4D
BIT 1,(HL)	CB 4E
BIT 1,(IX+d)	DD CB XX 4E
BIT 1,(IY+d)	FD CB XX 4E
BIT 2,A	CB 57
BIT 2,B	CB 50
BIT 2,C	CB 51
BIT 2,D	CB 52
BIT 2,E	CB 53
BIT 2,H	CB 54
BIT 2,L	CB 55

mnémonique	hexadécimal
BIT 2,(HL)	CB 56
BIT 2,(IX+d)	DD CB XX 56
BIT 2,(IY+d)	FD CB XX 56
BIT 3,A	CB 5F
BIT 3,B	CB 58
BIT 3,C	CB 59
BIT 3,D	CB 5A
BIT 3,E	CB 5B
BIT 3,H	CB 5C
BIT 3,L	CB 5D
BIT 3,(HL)	CB 5E
BIT 3,(IX+d)	DD CB XX 5E
BIT 3,(IY+d)	FD CB XX 5E
BIT 4,A	CB 67
BIT 4,B	CB 60
BIT 4,C	CB 61
BIT 4,D	CB 62
BIT 4,E	CB 63
BIT 4,H	CB 64
BIT 4,L	CB 65
BIT 4,(HL)	CB 66
BIT 4,(IX+d)	DD CB XX 66
BIT 4,(IY+d)	FD CB XX 66
BIT 5,A	CB 6F
BIT 5,B	CB 68
BIT 5,C	CB 69
BIT 5,D	CB 6A
BIT 5,E	CB 6B
BIT 5,H	CB 6C
BIT 5,L	CB 6D
BIT 5,(HL)	CB 6E
BIT 5,(IX+d)	DD CB XX 6E
BIT 5,(IY+d)	FD CB XX 6E
BIT 6,A	CB 77
BIT 6,B	CB 70
BIT 6,C	CB 71
BIT 6,D	CB 72
BIT 6,E	CB 73
BIT 6,H	CB 74

mnémonique	hexadécimal
BIT 6,L	CB 75
BIT 6,(HL)	CB 76
BIT 6,(IX+d)	DD CB XX 76
BIT 6,(IY+d)	FD CB XX 76
BIT 7,A	CB 7F
BIT 7,B	CB 78
BIT 7,C	CB 79
BIT 7,D	CB 7A
BIT 7,E	CB 7B
BIT 7,H	CB 7C
BIT 7,L	CB 7D
BIT 7,(HL)	CB 7E
BIT 7,(IX+d)	DD CB XX 7E
BIT 7,(IY+d)	FD CB XX 7E
CALL ADR	CD XX XX
CALL C,ADR	DC XX XX
CALL NC,ADR	D4 XX XX
CALL Z,ADR	CC XX XX
CALL NZ,ADR	C4 XX XX
CALL M,ADR	FC XX XX
CALL P,ADR	F4 XX XX
CALL PE,ADR	EC XX XX
CALL PO,ADR	E4 XX XX
CCF	3F
CP n	FE XX
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP (HL)	BE
CP (IX+d)	DD BE XX
CP (IY+d)	FD BE XX

mnémonique	hexadécimal
CPD	ED A9
CPDR	ED B9
CPI	ED A1
CPIR	ED B1
CPL	2F
DAA	27
DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D
DEC (HL)	35
DEC (IX+d)	DD 35 XX
DEC (IY+d)	FD 35 XX
DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC SP	3B
DEC IX	DD 2B
DEC IY	FD 2B
DI	F3
DJNZ d	10 XX
EI	FB
EX (SP),HL	E3
EX (SP),IX	DD E3
EX (SP),IY	FD E3
EX AF,AF'	08
EX DE,HL	EB
EXX	D9
HALT	76

mnémonique	hexadécimal
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E
IN A,(n)	DB XX
IN A,(C)	ED 78
IN B,(C)	ED 40
IN C,(C)	ED 48
IN D,(C)	ED 50
IN E,(C)	ED 58
IN H,(C)	ED 60
IN L,(C)	ED 68
INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C
INC (HL)	34
INC (IX+d)	DD 34 XX
INC (IY+d)	FD 34 XX
INC BC	03
INC DE	13
INC HL	23
INC SP	33
INC IX	DD 23
INC IY	FD 23
IND	ED AA
INDR	ED BA
INI	ED A2
INIR	ED B2
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD 29
JP C,ADR	DA XX XX
JP NC,ADR	D2 XX XX
JP Z,ADR	CA XX XX

mnémonique	hexadécimal
JP NZ,ADR	C2 XX XX
JP M,ADR	FA XX XX
JP P,ADR	F2 XX XX
JP PE,ADR	EA XX XX
JP PO,ADR	E2 XX XX
JR d	18 XX
JR C,d	38 XX
JR NC,d	30 XX
JR Z,d	28 XX
JR NZ,d	20 XX
LD A,n	3E XX
LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,L	7D
LD A,I	ED 57
LD A,R	ED 5F
LD A,(ADR)	3A XX XX
LD A,(BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD A,(IX+d)	DD 7E XX
LD A,(IY+d)	FD 7E XX
LD B,n	06 XX
LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,E	43
LD B,H	44
LD B,L	45
LD B,(HL)	46
LD B,(IX+d)	DD 46 XX
LD B,(IY+d)	FD 46 XX
LD C,n	0E XX
LD C,A	4F

mnémonique	hexadécimal
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,E	4B
LD C,H	4C
LD C,L	4D
LD C,(HL)	4E
LD C,(IX+d)	DD 4E XX
LD C,(IY+d)	FD 4E XX
LD D,n	16 XX
LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,E	53
LD D,H	54
LD D,L	55
LD D,(HL)	56
LD D,(IX+d)	DD 56 XX
LD D,(IY+d)	FD 56 XX
LD E,n	1E XX
LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,E	5B
LD E,H	5C
LD E,L	5D
LD E,(HL)	5E
LD E,(IX+d)	DD 5E XX
LD E,(IY+d)	FD 5E XX
LD H,n	26 XX
LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,E	63
LD H,H	64
LD H,L	65
LD H,(HL)	66
LD H,(IX+d)	DD 66 XX

mnémonique	hexadécimal
LD H,(IY+d)	FD 66 XX
LD L,n	2E XX
LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,E	6B
LD L,H	6C
LD L,L	6D
LD L,(HL)	6E
LD L,(IX+d)	DD 6E XX
LD L,(IY+d)	FD 6E XX
LD I,A	ED 47
LD R,A	ED 4F
LD BC,nn	01 XX XX
LD BC,(ADR)	ED 4B XX XX
LD DE,nn	11 XX XX
LD DE,(ADR)	ED 5B XX XX
LD HL,nn	21 XX XX
LD HL,(ADR)	2A XX XX
LD HL,(ADR)	ED 6B XX XX
LD SP,nn	31 XX XX
LD SP,HL	F9
LD SP,IX	DD F9
LD SP IY	FD F9
LD SP,(ADR)	ED 7B XX XX
LD IX,nn	DD 21 XX XX
LD IX,(ADR)	DD 2A XX XX
LD IY,nn	FD 21 XX XX
LD IY,(ADR)	FD 2A XX XX
LD (ADR),A	32 XX XX
LD (ADR),BC	ED 43 XX XX
LD (ADR),DE	ED 53 XX XX
LD (ADR),HL	22 XX XX
LD (ADR),HL	ED 63 XX XX
LD (ADR),IX	DD 22 XX XX
LD (ADR),IY	FD 22 XX XX
LD (ADR),SP	ED 73 XX XX
LD (BC),A	02
LD (DE),A	12
LD (HL),n	36 XX

mnémonique	hexadécimal
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75
LD (IX+d),n	DD 36 XX XX
LD (IX+d),A	DD 77 XX
LD (IX+d),B	DD 70 XX
LD (IX+d),C	DD 71 XX
LD (IX+d),D	DD 72 XX
LD (IX+d),E	DD 73 XX
LD (IX+d),H	DD 74 XX
LD (IX+d),L	DD 75 XX
LD (IY+d),n	FD 36 XX XX
LD (IY+d),A	FD 77 XX
LD (IY+d),B	FD 70 XX
LD (IY+d),C	FD 71 XX
LD (IY+d),D	FD 72 XX
LD (IY+d),E	FD 73 XX
LD (IY+d),H	FD 74 XX
LD (IY+d),L	FD 75 XX
LDD	ED A8
LDDR	ED B8
LDI	ED A0
LDIR	ED B0
NEG	ED 44
NOP	00
OR n	F6 XX
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5

mnémonique	hexadécimal
OR (HL)	B6
OR (IX+d)	DD B6 XX
OR (IX+d)	FD B6 XX
OUT (n),A	D3 XX
OUT (C),A	ED 79
OUT (C),B	ED 41
OUT (C),C	ED 49
OUT (C),D	ED 51
OUT (C),E	ED 59
OUT (C),H	ED 61
OUT (C),L	ED 69
OUTD	ED AB
OUTDR	ED BB
OUTI	ED A3
OUTIR	ED B3
POP AF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DD E1
POP IY	FD E1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DD E5
PUSH IY	FD E5
RES 0,A	CB 87
RES 0,B	CB 80
RES 0,C	CB 81
RES 0,D	CB 82
RES 0,E	CB 83
RES 0,H	CB 84
RES 0,L	CB 85
RES 0,(HL)	CB 86
RES 0,(IX+d)	DD CB XX 86

mnémonique	hexadécimal
RES 0,(IY+d)	FD CB XX 86
RES 1,A	CB 8F
RES 1,B	CB 88
RES 1,C	CB 89
RES 1,D	CB 8A
RES 1,D	CB 8B
RES 1,H	CB 8C
RES 1,L	CB 8D
RES 1,(HL)	CB 8E
RES 1,(IX+d)	DD CB XX 8E
RES 1,(IY+d)	FD CB XX 8E
RES 2,A	CB 97
RES 2,B	CB 90
RES 2,C	CB 91
RES 2,D	CB 92
RES 2,E	CB 93
RES 2,H	CB 94
RES 2,L	CB 95
RES 2,(HL)	CB 96
RES 2,(IX+d)	DD CB XX 96
RES 2,(IY+d)	FD CB XX 96
RES 3,A	CB 9F
RES 3,B	CB 98
RES 3,C	CB 99
RES 3,D	CB 9A
RES 3,E	CB 9B
RES 3,H	CB 9C
RES 3,L	CB 9D
RES 3,(HL)	CB 9E
RES 3,(IX+d)	DD CB XX 9E
RES 3,(IY+d)	FD CB XX 9E
RES 4,A	CB A7
RES 4,B	CB A0
RES 4,C	CB A1
RES 4,D	CB A2
RES 4,E	CB A3
RES 4,H	CB A4
RES 4,L	CB A5
RES 4,(HL)	CB A6
RES 4,(IX+d)	DD CB XX A6
RES 4,(IY+d)	FD CB XX A6

mnémonique	hexadécimal
RES 5,A	CB AF
RES 5,B	CB A8
RES 5,C	CB A9
RES 5,D	CB AA
RES 5,E	CB AB
RES 5,H	CB AC
RES 5,L	CB AD
RES 5,(HL)	CB AE
RES 5,(IX+d)	DD CB XX AE
RES 5,(IY+d)	FD CB XX AE
RES 6,A	CB B7
RES 6,B	CB B0
RES 6,C	CB B1
RES 6,D	CB B2
RES 6,E	CB B3
RES 6,H	CB B4
RES 6,L	CB B5
RES 6,(HL)	CB B6
RES 6,(IX+d)	DD CB XX B6
RES 6,(IY+d)	FD CB XX B6
RES 7,A	CB BF
RES 7,B	CB B8
RES 7,C	CB B9
RES 7,D	CB BA
RES 7,E	CB BB
RES 7,H	CB BC
RES 7,L	CB BD
RES 7,(HL)	CB BE
RES 7,(IX+d)	DD CB XX BE
RES 7,(IY+d)	FD CB XX BE
RET	C9
RET C	D8
RET NC	D0
RET Z	C8
RET NZ	C0
RET M	F8
RET P	F0
RET PE	E8
RET PO	E0

mnémonique	hexadécimal
RETI	ED 4D
RETN	ED 45
RLA	17
RL A	CB 17
RL B	CB 10
RL C	CB 11
RL D	CB 12
RL E	CB 13
RL H	CB 14
RL L	CB 15
RL (HL)	CB 16
RL (IX+d)	DD CB XX 16
RL (IY+d)	FD CB XX 16
RLCA	07
RLC A	CB 07
RLC B	CB 00
RLC C	CB 01
RLC D	CB 02
RLC E	CB 03
RLC H	CB 04
RLC L	CB 05
RLC (HL)	CB 06
RLC (IX+d)	DD CB XX 06
RLC (IY+d)	FD CB XX 06
RRA	1F
RR A	CB 1F
RR B	CB 18
RR C	CB 19
RR D	CB 1A
RR E	CB 1B
RR H	CB 1C
RR L	CB 1D
RR (HL)	CB 1E
RR (IX+d)	DD CB XX 1E
RR (IY+d)	FD CB XX 1E
RRCA	0F

mnémonique	hexadécimal
RRC A	CB 0F
RRC B	CB 08
RRC C	CB 09
RRC D	CB 0A
RRC E	CB 0B
RRC H	CB 0C
RRC L	CB 0D
RRC (HL)	CB 0E
RRC (IX+d)	DD CB XX 0E
RRC (IY+d)	FD CB XX 0E
RLD	ED 6F
RRD	ED 67
RST 00	C7
RST 08	CF
RST 10	D7
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF
SBC A,n	DE XX
SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D
SBC A,(HL)	9E
SBC A,(IX+d)	DD 9E XX
SBC A,(IY+d)	FD 9E XX
SBC HL,BC	ED 42
SBC HL,DE	ED 52
SBC HL,HL	ED 62
SBC HL,SP	ED 72
SCF	37

mnémonique	hexadécimal
SET 0,A	CB C7
SET 0,B	CB C0
SET 0,C	CB C1
SET 0,D	CB C2
SET 0,E	CB C3
SET 0,H	CB C4
SET 0,L	CB C5
SET 0,(HL)	CB C6
SET 0,(IX+d)	DD CB XX C6
SET 0,(IY+d)	FD CB XX C6
SET 1,A	CB CF
SET 1,B	CB C8
SET 1,C	CB C9
SET 1,D	CB CA
SET 1,E	CB CB
SET 1,H	CB CC
SET 1,L	CB CD
SET 1,(HL)	CB CE
SET 1,(IX+d)	DD CB XX CE
SET 1,(IY+d)	FD CB XX CE
SET 2,A	CB D7
SET 2,B	CB D0
SET 2,C	CB D1
SET 2,D	CB D2
SET 2,E	CB D3
SET 2,H	CB D4
SET 2,L	CB D5
SET 2,(HL)	CB D6
SET 2,(IX+d)	DD CB XX D6
SET 2,(IY+d)	FD CB XX D6
SET 3,A	CB DF
SET 3,B	CB D8
SET 3,C	CB D9
SET 3,D	CB DA
SET 3,E	CB DB
SET 3,H	CB DC
SET 3,L	CB DD
SET 3,(HL)	CB DE
SET 3,(IX+d)	DD CB XX DE
SET 3,(IY+d)	FD CB XX DE
SET 4,A	CB E7

mnémonique	hexadécimal
SET 4,B	CB E0
SET 4,C	CB E1
SET 4,D	CB E2
SET 4,E	CB E3
SET 4,H	CB E4
SET 4,L	CB E5
SET 4,(HL)	CB E6
SET 4,(IX+d)	DD CB XX E6
SET 4,(IY+d)	FD CB XX C6
SET 5,A	CB EF
SET 5,B	CB E8
SET 5,C	CB E9
SET 5,D	CB EA
SET 5,E	CB EB
SET 5,H	CB EC
SET 5,L	CB ED
SET 5,(HL)	CB EE
SET 5,(IX+d)	DD CB XX EE
SET 5,(IY+d)	FD CB XX EE
SET 6,A	CB F7
SET 6,B	CB F0
SET 6,C	CB F1
SET 6,D	CB F2
SET 6,E	CB F3
SET 6,H	CB F4
SET 6,L	CB F5
SET 6,(HL)	CB F6
SET 6,(IX+d)	DD CB XX F6
SET 6,(IY+d)	FD CB XX F6
SET 7,A	CB FF
SET 7,B	CB F8
SET 7,C	CB F9
SET 7,D	CB FA
SET 7,E	CB FB
SET 7,H	CB FC
SET 7,L	CB FD
SET 7,(HL)	CB FE
SET 7,(IX+d)	DD CB XX FE
SET 7,(IY+d)	FD CB XX FE

mnémonique	hexadécimal
SLA A	CB 27
SLA B	CB 20
SLA C	CB 21
SLA D	CB 22
SLA E	CB 23
SLA H	CB 24
SLA L	CB 25
SLA (HL)	CB 26
SLA (IX+d)	DD CB XX 26
SLA (IY+d)	FD CB XX 26
SRA A	CB 2F
SRA B	CB 28
SRA C	CB 29
SRA D	CB 2A
SRA E	CB 2B
SRA H	CB 2C
SRA L	CB 2D
SRA (HL)	CB 2E
SRA (IX+d)	DD CB XX 2E
SRA (IY+d)	FD CB XX 2E
SRL A	CB 3F
SRL B	CB 38
SRL C	CB 39
SRL D	CB 3A
SRL E	CB 3B
SRL H	CB 3C
SRL L	CB 3D
SRL (HL)	CB 3E
SRL (IX+d)	DD CB XX 3E
SRL (IY+d)	FD CB XX 3E
SUB n	D6 XX
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93

mnémonique	hexadécimal
SUB H	94
SUB L	95
SUB (HL)	96
SUB (IX+d)	DD 96 XX
SUB (IY+d)	FD 96 XX
XOR n	EE XX
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR (HL)	AE
XOR (IX+d)	DD AE XX
XOR (IY+d)	FD AE XX

ANNEXE 1 - Z 80 - INSTRUCTIONS - DÉCODAGE

hexadécimal	mnémonique
00	NOP
01 XX XX	LD BC,nn
02	LD (BC),A
03	INC BC
04	INC B
05	DEC B
06 XX	LD B,n
07	RLCA
08	EX AF,AF'
09	ADD HL,BC
0A	LD A,(BC)
0B	DEC BC
0C	INC C
0D	DEC C
0E XX	LD C,n
0F	RRCA
10 XX	DJNZ d
11 XX XX	LD DE,nn
12	LD (DE),A
13	INC DE
14	INC D
15	DEC D
16 XX	LD D,n
17	RLA
18 XX	JR d
19	ADD HL,DE
1A	LD A,(DE)
1B	DEC DE
1C	INC E
1D	DEC E
1E XX	LD E,n
1F	RRA
20 XX	JR NZ,d
21 XX XX	LD HL,nn
22 XX XX	LD (ADR),HL
23	INC HL
24	INC H
25	DEC H
26 XX	LD H,n
27	DAA
28 XX	JR Z,d
29	ADD HL,HL
2A XX XX	LD HL,(ADR)
2B	DEC HL
2C	INC L
2D	DEC L
2E XX	LD L,n
2F	CPL
30 XX	JR NC,d
31 XX XX	LD SP,nn
32 XX XX	LD (ADR),A
33	INC SP
34	INC (HL)
35	DEC (HL)
36 XX	LD (HL),n
37	SCF
38 XX	JR C,d
39	ADD HL,SP
3A XX XX	LD A,(ADR)
3B	DEC SP
3C	INC A
3D	DEC A
3E XX	LD A,n
3F	CCF
40	LD B,B
41	LD B,C
42	LD B,D
43	LD B,E
44	LD B,H
45	LD B,L
46	LD B,(HL)
47	LD B,A
48	LD C,B
49	LD C,C
4A	LD C,D
4B	LD C,E
4C	LD C,H
4D	LD C,L
4E	LD C,(HL)
4F	LD C,A
50	LD D,B
51	LD D,C
52	LD D,D
53	LD D,E
54	LD D,H
55	LD D,L
56	LD D,(HL)
57	LD D,A
58	LD E,B
59	LD E,C
5A	LD E,D
5B	LD E,E
5C	LD E,H
5D	LD E,L
5E	LD E,(HL)
5F	LD E,A
60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
66	LD H,(HL)
67	LD H,A
68	LD L,B
69	LD L,C
6A	LD L,D
6B	LD L,E
6C	LD L,H
6D	LD L,L
6E	LD L,(HL)
6F	LD L,A
70	LD (HL),B
71	LD (HL),C
72	LD (HL),D
73	LD (HL),E
74	LD (HL),H
75	LD (HL),L
76	HALT
77	LD (HL),A
78	LD A,B
79	LD A,C
7A	LD A,D
7B	LD A,E
7C	LD A,H
7D	LD A,L
7E	LD A,(HL)
7F	LD A,A
80	ADD A,B
81	ADD A,C
82	ADD A,D
83	ADD A,E
84	ADD A,H
85	ADD A,L
86	ADD A,(HL)
87	ADD A,A
88	ADC A,B
89	ADC A,C
8A	ADC A,D
8B	ADC A,E
8C	ADC A,H
8D	ADC A,L
8E	ADC A,(HL)
8F	ADC A,A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
96	SUB (HL)
97	SUB A

hexadécimal	mnémonique
98	SBC A,B
99	SBC A,C
9A	SBC A,D
9B	SBC A,E
9C	SBC A,H
9D	SBC A,L
9E	SBC A,(HL)
9F	SBC A,A
A0	AND B
A1	AND C
A2	AND D
A3	AND E
A4	AND H
A5	AND L
A6	AND (HL)
A7	AND A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
AE	XOR (HL)
AF	XOR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
B6	OR (HL)
B7	OR A
B8	CP B
B9	CP C
BA	CP D
BB	CP E
BC	CP H
BD	CP L
BE	CP (HL)
BF	CP A
C0	RET NZ
C1	POP BC
C2 XX XX	JP NZ,ADR
C3 XX XX	JP ADR
C4 XX XX	CALL NZ,ADR
C5	PUSH BC
C6 XX	ADD A,n
C7	RST 00
C8	RET Z
C9	RET
CA XX XX	JP Z,ADR
CC XX XX	CALL Z,ADR
CD XX XX	CALL ADR
CE XX	ADC A,n
CF	RST 08
D0	RET NC
D1	POP DE
D2 XX XX	JP NC,ADR
D3 XX	OUT (n),A
D4 XX XX	CALL NC,ADR
D5	PUSH DE
D6 XX	SUB n
D7	RST 10
D8	RET C
D9	EXX
DA XX XX	JP C,ADR
DB XX	IN A,(n)
DC XX XX	CALL C,ADR
DE XX	SBC A,n
DF	RST 18
E0	RET PO
E1	POP HL
E2 XX XX	JP PO,ADR
E3	EX (SP),HL
E4 XX XX	CALL PO,ADR
E5	PUSH HL
E6 XX	AND n
E7	RST 20
E8	RET PE
E9	JP (HL)
EA XX XX	JP PE,ADR
EB	EX DE,HL
EC XX XX	CALL PE,ADR
EE XX	XOR n
EF	RST 28
F0	RET P
F1	POP AF
F2 XX XX	JP P,ADR
F3	DI
F4 XX XX	CALL P,ADR
F5	PUSH AF
F6 XX	OR n
F7	RST 30
F8	RET M
F9	LD SP,HL
FA XX XX	JP M,ADR
FB	EI
FC XX XX	CALL M,ADR
FE XX	CP n
FF	RST 38
CB 00	RLC B
CB 01	RLC C
CB 02	RLC D
CB 03	RLC E
CB 04	RLC H
CB 05	RLC L
CB 06	RLC (HL)
CB 07	RLC A
CB 08	RRC B
CB 09	RRC C
CB 0A	RRC D
CB 0B	RRC E
CB 0C	RRC H
CB 0D	RRC L
CB 0E	RRC (HL)
CB 0F	RRC A
CB 10	RL B
CB 11	RL C
CB 12	RL D
CB 13	RL E
CB 14	RL H
CB 15	RL L
CB 16	RL (HL)
CB 17	RL A
CB 18	RR B
CB 19	RR C
CB 1A	RR D
CB 1B	RR E
CB 1C	RR H
CB 1D	RR L
CB 1E	RR (HL)
CB 1F	RR A
CB 20	SLA B
CB 21	SLA C
CB 22	SLA D
CB 23	SLA E
CB 24	SLA H
CB 25	SLA L
CB 26	SLA (HL)
CB 27	SLA A
CB 28	SRA B
CB 29	SRA C
CB 2A	SRA D
CB 2B	SRA E
CB 2C	SRA H
CB 2D	SRA L
CB 2E	SRA (HL)
CB 2F	SRA A
CB 38	SRL B
CB 39	SRL C
CB 3A	SRL D
CB 3B	SRL E
CB 3C	SRL H
CB 3D	SRL L
CB 3E	SRL (HL)
CB 3F	SRL A
CB 40	BIT 0,B
CB 41	BIT 0,C
CB 42	BIT 0,D
CB 43	BIT 0,E
CB 44	BIT 0,H
CB 45	BIT 0,L
CB 46	BIT 0,(HL)
CB 47	BIT 0,A

hexadécimal	mnémonique	hexadécimal	mnémonique	hexadécimal	mnémonique	hexadécimal	mnémonique
CB 48	BIT 1,B	CB 71	BIT 6,C	CB 9A	RES 3,D	CB C3	SET 0,E
CB 49	BIT 1,C	CB 72	BIT 6,D	CB 9B	RES 3,E	CB C4	SET 0,H
CB 4A	BIT 1,D	CB 73	BIT 6,E	CB 9C	RES 3,H	CB C5	SET 0,L
CB 4B	BIT 1,E	CB 74	BIT 6,H	CB 9D	RES 3,L	CB C6	SET 0,(HL)
CB 4C	BIT 1,H	CB 75	BIT 6,L	CB 9E	RES 3,(HL)	CB C7	SET 0,A
CB 4D	BIT 1,L	CB 76	BIT 6,(HL)	CB 9F	RES 3,A	CB C8	SET 1,B
CB 4E	BIT 1,(HL)	CB 77	BIT 6,A	CB A0	RES 4,B	CB C9	SET 1,C
CB 4F	BIT 1,A	CB 78	BIT 7,B	CB A1	RES 4,C	CB CA	SET 1,D
CB 50	BIT 2,B	CB 79	BIT 7,C	CB A2	RES 4,D	CB CB	SET 1,E
CB 51	BIT 2,C	CB 7A	BIT 7,D	CB A3	RES 4,E	CB CC	SET 1,H
CB 52	BIT 2,D	CB 7B	BIT 7,E	CB A4	RES 4,H	CB CD	SET 1,L
CB 53	BIT 2,E	CB 7C	BIT 7,H	CB A5	RES 4,L	CB CE	SET 1,(HL)
CB 54	BIT 2,H	CB 7D	BIT 7,L	CB A6	RES 4,(HL)	CB CF	SET 1,A
CB 55	BIT 2,L	CB 7E	BIT 7,(HL)	CB A7	RES 4,A	CB D0	SET 2,B
CB 56	BIT 2,(HL)	CB 7F	BIT 7,A	CB A8	RES 5,B	CB D1	SET 2,C
CB 57	BIT 2,A	CB 80	RES 0,B	CB A9	RES 5,C	CB D2	SET 2,D
CB 58	BIT 3,B	CB 81	RES 0,C	CB AA	RES 5,D	CB D3	SET 2,E
CB 59	BIT 3,C	CB 82	RES 0,D	CB AB	RES 5,E	CB D4	SET 2,H
CB 5A	BIT 3,D	CB 83	RES 0,E	CB AC	RES 5,H	CB D5	SET 2,L
CB 5B	BIT 3,E	CB 84	RES 0,H	CB AD	RES 5,L	CB D6	SET 2,(HL)
CB 5C	BIT 3,H	CB 85	RES 0,L	CB AE	RES 5,(HL)	CB D7	SET 2,A
CB 5D	BIT 3,L	CB 86	RES 0,(HL)	CB AF	RES 5,A	CB D8	SET 3,B
CB 5E	BIT 3,(HL)	CB 87	RES 0,A	CB B0	RES 6,B	CB D9	SET 3,C
CB 5F	BIT 3,A	CB 88	RES 1,B	CB B1	RES 6,C	CB DA	SET 3,D
CB 60	BIT 4,B	CB 89	RES 1,C	CB B2	RES 6,D	CB DB	SET 3,E
CB 61	BIT 4,C	CB 8A	RES 1,D	CB B3	RES 6,E	CB DC	SET 3,H
CB 62	BIT 4,D	CB 8B	RES 1,E	CB B4	RES 6,H	CB DD	SET 3,L
CB 63	BIT 4,E	CB 8C	RES 1,H	CB B5	RES 6,L	CB DE	SET 3,(HL)
CB 64	BIT 4,H	CB 8D	RES 1,L	CB B6	RES 6,(HL)	CB DF	SET 3,A
CB 65	BIT 4,L	CB 8E	RES 1,(HL)	CB B7	RES 6,A	CB E0	SET 4,B
CB 66	BIT 4,(HL)	CB 8F	RES 1,A	CB B8	RES 7,B	CB E1	SET 4,C
CB 67	BIT 4,A	CB 90	RES 2,B	CB B9	RES 7,C	CB E2	SET 4,D
CB 68	BIT 5,B	CB 91	RES 2,C	CB BA	RES 7,D	CB E3	SET 4,E
CB 69	BIT 5,C	CB 92	RES 2,D	CB BB	RES 7,E	CB E4	SET 4,H
CB 6A	BIT 5,D	CB 93	RES 2,E	CB BC	RES 7,H	CB E5	SET 4,L
CB 6B	BIT 5,E	CB 94	RES 2,H	CB BD	RES 7,L	CB E6	SET 4,(HL)
CB 6C	BIT 5,H	CB 95	RES 2,L	CB BE	RES 7,(HL)	CB E7	SET 4,A
CB 6D	BIT 5,L	CB 96	RES 2,(HL)	CB BF	RES 7,A	CB E8	SET 5,B
CB 6E	BIT 5,(HL)	CB 97	RES 2,A	CB C0	SET 0,B	CB E9	SET 5,C
CB 6F	BIT 5,A	CB 98	RES 3,B	CB C1	SET 0,C	CB EA	SET 5,D
CB 70	BIT 6,B	CB 99	RES 3,C	CB C2	SET 0,D	CB EB	SET 5,E

hexadécimal	mnémonique
CB EC	SET 5,H
CB ED	SET 5,L
CB EE	SET 5,(HL)
CB EF	SET 5,A
CB F0	SET 6,B
CB F1	SET 6,C
CB F2	SET 6,D
CB F3	SET 6,E
CB F4	SET 6,H
CB F5	SET 6,L
CB F6	SET 6,(HL)
CB F7	SET 6,A
CB F8	SET 7,B
CB F9	SET 7,C
CB FA	SET 7,D
CB FB	SET 7,E
CB FC	SET 7,H
CB FD	SET 7,L
CB FE	SET 7,(HL)
CB FF	SET 7,A
DD 09	ADD IX,BC
DD 19	ADD IX,DE
DD 21 XX XX	LD IX,nn
DD 22 XX XX	LD (ADR),IX
DD 23	INC IX
DD 29	ADD IX,IX
DD 2A XX XX	LD IX,(ADR)
DD 2B	DEC IX
DD 34 XX	INC (IX+d)
DD 35 XX	DEC (IX+d)
DD 36 XX XX	LD (IX+d),n
DD 39	ADD IX,SP
DD 46 XX	LD B,(IX+d)
DD 4E XX	LD C,(IX+d)
DD 56 XX	LD D,(IX+d)
DD 5E XX	LD E,(IX+d)
DD 66 XX	LD H,(IX+d)
DD 6E XX	LD L,(IX+d)
DD 70 XX	LD (IX+d),B
DD 71 XX	LD (IX+d),C
DD 72 XX	LD (IX+d),D

hexadécimal	mnémonique
DD 73 XX	LD (IX+d),E
DD 74 XX	LD (IX+d),H
DD 75 XX	LD (IX+d),L
DD 77 XX	LD (IX+d),A
DD 7E XX	LD A,(IX+d)
DD 86 XX	ADD A,(IX+d)
DD 8E XX	ADC A,(IX+d)
DD 96 XX	SUB (IX+d)
DD 9E XX	SBC A,(IX+d)
DD A6 XX	AND (IX+d)
DD AE XX	XOR (IX+d)
DD B6 XX	OR (IX+d)
DD BE XX	CP (IX+d)
DD E1	POP IX
DD E3	EX (SP),IX
DD E5	PUSH IX
DD E9	JP (IX)
DD F9	LD SP,IX
DD CB XX 06	RLC (IX+d)
DD CB XX 0E	RRC (IX+d)
DD CB XX 16	RL (IX+d)
DD CB XX 1E	RR (IX+d)
DD CB XX 26	SLA (IX+d)
DD CB XX 2E	SRA (IX+d)
DD CB XX 3E	SRL (IX+d)
DD CB XX 46	BIT 0,(IX+d)
DD CB XX 4E	BIT 1,(IX+d)
DD CB XX 56	BIT 2,(IX+d)
DD CB XX 5E	BIT 3,(IX+d)
DD CB XX 66	BIT 4,(IX+d)
DD CB XX 6E	BIT 5,(IX+d)
DD CB XX 76	BIT 6,(IX+d)
DD CB XX 7E	BIT 7,(IX+d)
DD CB XX 86	RES 0,(IX+d)
DD CB XX 8E	RES 1,(IX+d)
DD CB XX 96	RES 2,(IX+d)
DD CB XX 9E	RES 3,(IX+d)
DD CB XX A6	RES 4,(IX+d)
DD CB XX AE	RES 5,(IX+d)
DD CB XX B6	RES 6,(IX+d)
DD CB XX BE	RES 7,(IX+d)

hexadécimal	mnémonique
DD CB XX C6	SET 0,(IX+d)
DD CB XX CE	SET 1,(IX+d)
DD CB XX D6	SET 2,(IX+d)
DD CB XX DE	SET 3,(IX+d)
DD CB XX E6	SET 4,(IX+d)
DD CB XX EE	SET 5,(IX+d)
DD CB XX F6	SET 6,(IX+d)
DD CB XX FE	SET 7,(IX+d)
ED 40	IN B,(C)
ED 41	OUT (C),B
ED 42	SBC HL,BC
ED 43 XX XX	LD (ADR),BC
ED 44	NEG
ED 45	RETN
ED 46	IM 0
ED 47	LD I,A
ED 48	IN C,(C)
ED 49	OUT (C),C
ED 4A	ADC HL,BC
ED 4B XX XX	LD BC,(ADR)
ED 4D	RETI
ED 50	IN D,(C)
ED 51	OUT (C),D
ED 52	SBC HL,DE
ED 53 XX XX	LD (ADR),DE
ED 56	IM 1
ED 57	LD A,I
ED 58	IN E,(C)
ED 59	OUT (C),,E
ED 5A	ADC HL,DE
ED 5B XX XX	LD DE,(ADR)
ED 5E	IM 2
ED 60	IN H,(C)
ED 61	OUT (C),H
ED 62	SBC HL, HL
ED 63 XX XX	LD (ADR),HL
ED 67	RRD
ED 68	IN L,(C)
ED 69	OUT (C),L
ED 6A	ADC HL,HL
ED 6B XX XX	LD HL,(ADR)

hexadécimal	mnémonique
ED 6F	RLD
ED 72	SBC HL,SP
ED 73 XX XX	LD (ADR),SP
ED 78	IN A,(C)
ED 79	OUT (C),A
ED 7A	ADC HL,SP
ED 7B XX XX	LD SP,(ADR)
ED A0	LDI
ED A1	CPI
ED A2	INI
ED A3	OUTI
ED A8	LDD
ED A9	CPD
ED AA	IND
ED AB	OUTD
ED B0	LDIR
ED B1	CPIR
ED B2	INIR
ED B3	OUTIR
ED B8	LDDR
ED B9	CPDR
ED BA	INDR
ED BB	OUTDR
FD 09	ADD IY,BC
FD 19	ADD IY,DE
FD 21 XX XX	LD IY,nn
FD 22 XX XX	LD (ADR),IY
FD 23	INC IY
FD 29	ADD IY,IY
FD 2A XX XX	LD IY,(ADR)
FD 2B	DEC IY
FD 34 XX	INC (IY+d)
FD 35 XX	DEC (IY+d)
FD 36 XX XX	LD (IY+d),n
FD 39	ADD IY,SP
FD 46 XX	LD B,(IY+d)
FD 4E XX	LD C,(IY+d)
FD 56 XX	LD D,(IY+d)
FD 5E XX	LD E,(IY+d)
FD 66 XX	LD H,(IY+d)
FD 6E XX	LD L,(IY+d)

hexadécimal	mnémonique
FD 70 XX	LD (IY+d),B
FD 71 XX	LD (IY+d),C
FD 72 XX	LD (IY+d),D
FD 73 XX	LD (IY+d),E
FD 74 XX	LD (IY+d),H
FD 75 XX	LD (IY+d),L
FD 77 XX	LD (IY+d),A
FD 7E XX	LD A,(IY+d)
FD 86 XX	ADD A,(IY+d)
FD 8E XX	ADC A,(IY+d)
FD 96 XX	SUB (IY+d)
FD 9E XX	SBC A,(IY+d)
FD A6 XX	AND (IY+d)
FD AE XX	XOR (IY+d)
FD B6 XX	OR (IY+d)
FD BE XX	CP (IY+d)
FD E1	POP IY
FD E3	EX (SP),IY
FD E5	PUSH IY
FD E9	JP (IY)
FD F9	LD SP,IY
FD CB XX 06	RLC (IY+d)
FD CB XX 0E	RRC (IY+d)
FD CB XX 16	RL (IY+d)
FD CB XX 1E	RR (IY+d)
FD CB XX 26	SLA (IY+d)
FD CB XX 2E	SRA (IY+d)
FD CB XX 3E	SRL (IY+d)
FD CB XX 46	BIT 0,(IY+d)
FD CB XX 4E	BIT 1,(IY+d)
FD CB XX 56	BIT 2,(IY+d)
FD CB XX 5E	BIT 3,(IY+d)
FD CB XX 66	BIT 4,(IY+d)
FD CB XX 6E	BIT 5,(IY+d)
FD CB XX 76	BIT 6,(IY+d)
FD CB XX 7E	BIT 7,(IY+d)
FD CB XX 86	RES 0,(IY+d)
FD CB XX 8E	RES 1,(IY+d)
FD CB XX 96	RES 2,(IY+d)
FD CB XX 9E	RES 3,(IY+d)
FD CB XX A6	RES 4,(IY+d)
FD CB XX AE	RES 5,(IY+d)
FD CB XX B6	RES 6,(IY+d)
FD CB XX BE	RES 7,(IY+d)
FD CB XX C6	SET 0,(IY+d)
FD CB XX CE	SET 1,(IY+d)
FD CB XX D6	SET 2,(IY+d)
FD CB XX DE	SET 3,(IY+d)
FD CB XX E6	SET 4,(IY+d)
FD CB XX EE	SET 5,(IY+d)
FD CB XX F6	SET 6,(IY+d)
FD CB XX FE	SET 7,(IY+d)

ANNEXE 2 - CONVERSION DÉCIMAL -HEXADÉCIMAL

DECIMAL 0-255				HEXA 00-FF OCTET BAS					
DEC.	HEXA	DEC.	HEXA	DEC.	HEXA	CPL2	DEC.	HEXA	CPL2
0	00	64	40	128	80	-128	192	C0	-64
1	01	65	41	129	81	-127	193	C1	-63
2	02	66	42	130	82	-126	194	C2	-62
3	03	67	43	131	83	-125	195	C3	-61
4	04	68	44	132	84	-124	196	C4	-60
5	05	69	45	133	85	-123	197	C5	-59
6	06	70	46	134	86	-122	198	C6	-58
7	07	71	47	135	87	-121	199	C7	-57
8	08	72	48	136	88	-120	200	C8	-56
9	09	73	49	137	89	-119	201	C9	-55
10	0A	74	4A	138	8A	-118	202	CA	-54
11	0B	75	4B	139	8B	-117	203	CB	-53
12	0C	76	4C	140	8C	-116	204	CC	-52
13	0D	77	4D	141	8D	-115	205	CD	-51
14	0E	78	4E	142	8E	-114	206	CE	-50
15	0F	79	4F	143	8F	-113	207	CF	-49
16	10	80	50	144	90	-112	208	D0	-48
17	11	81	51	145	91	-111	209	D1	-47
18	12	82	52	146	92	-110	210	D2	-46
19	13	83	53	147	93	-109	211	D3	-45
20	14	84	54	148	94	-108	212	D4	-44
21	15	85	55	149	95	-107	213	D5	-43
22	16	86	56	150	96	-106	214	D6	-42
23	17	87	57	151	97	-105	215	D7	-41
24	18	88	58	152	98	-104	216	D8	-40
25	19	89	59	153	99	-103	217	D9	-39
26	1A	90	5A	154	9A	-102	218	DA	-38
27	1B	91	5B	155	9B	-101	219	DB	-37
28	1C	92	5C	156	9C	-100	220	DC	-36
29	1D	93	5D	157	9D	- 99	221	DD	-35
30	1E	94	5E	158	9E	- 98	222	DE	-34
31	1F	95	5F	159	9F	- 97	223	DF	-33
32	20	96	60	160	A0	- 96	224	E0	-32
33	21	97	61	161	A1	- 95	225	E1	-31
34	22	98	62	162	A2	- 94	226	E2	-30
35	23	99	63	163	A3	- 93	227	E3	-29
36	24	100	64	164	A4	- 92	228	E4	-28
37	25	101	65	165	A5	- 91	229	E5	-27
38	26	102	66	166	A6	- 90	230	E6	-26
39	27	103	67	167	A7	- 89	231	E7	-25
40	28	104	68	168	A8	- 88	232	E8	-24
41	29	105	69	169	A9	- 87	233	E9	-23
42	2A	106	6A	170	AA	- 86	234	EA	-22
43	2B	107	6B	171	AB	- 85	235	EB	-21
44	2C	108	6C	172	AC	- 84	236	EC	-20
45	2D	109	6D	173	AD	- 83	237	ED	-19
46	2E	110	6E	174	AE	- 82	238	EE	-18
47	2F	111	6F	175	AF	- 81	239	EF	-17
48	30	112	70	176	B0	- 80	240	F0	-16
49	31	113	71	177	B1	- 79	241	F1	-15
50	32	114	72	178	B2	- 78	242	F2	-14
51	33	115	73	179	B3	- 77	243	F3	-13
52	34	116	74	180	B4	- 76	244	F4	-12
53	35	117	75	181	B5	- 75	245	F5	-11
54	36	118	76	182	B6	- 74	246	F6	-10
55	37	119	77	183	B7	- 73	247	F7	- 9
56	38	120	78	184	B8	- 72	248	F8	- 8
57	39	121	79	185	B9	- 71	249	F9	- 7
58	3A	122	7A	186	BA	- 70	250	FA	- 6
59	3B	123	7B	187	BB	- 69	251	FB	- 5
60	3C	124	7C	188	BC	- 68	252	FC	- 4
61	3D	125	7D	189	BD	- 67	253	FD	- 3
62	3E	126	7E	190	BE	- 66	254	FE	- 2
63	3F	127	7F	191	BF	- 65	255	FF	- 1

DECIMAL 0-65280				HEXA 00-FF OCTET HAUT			
DECIMAL	HEXA	DECIMAL	HEXA	DECIMAL	HEXA	DECIMAL	HEXA
0	00	16384	40	32768	80	49152	C0
256	01	16640	41	33024	81	49408	C1
512	02	16896	42	33280	82	49664	C2
768	03	17152	43	33536	83	49920	C3
1024	04	17408	44	33792	84	50176	C4
1280	05	17664	45	34048	85	50432	C5
1536	06	17920	46	34304	86	50688	C6
1792	07	18176	47	34560	87	50944	C7
2048	08	18432	48	34816	88	51200	C8
2304	09	18688	49	35072	89	51456	C9
2560	0A	18944	4A	35328	8A	51712	CA
2816	0B	19200	4B	35584	8B	51968	CB
3072	0C	19456	4C	35840	8C	52224	CC
3328	0D	19712	4D	36096	8D	52480	CD
3584	0E	19968	4E	36352	8E	52736	CE
3840	0F	20224	4F	36608	8F	52992	CF
4096	10	20480	50	36864	90	53248	D0
4352	11	20736	51	37120	91	53504	D1
4608	12	20992	52	37376	92	53760	D2
4864	13	21248	53	37632	93	54016	D3
5120	14	21504	54	37888	94	54272	D4
5376	15	21760	55	38144	95	54528	D5
5632	16	22016	56	38400	96	54784	D6
5888	17	22272	57	38656	97	55040	D7
6144	18	22528	58	38912	98	55296	D8
6400	19	22784	59	39168	99	55552	D9
6656	1A	23040	5A	39424	9A	55808	DA
6912	1B	23296	5B	39680	9B	56064	DB
7168	1C	23552	5C	39936	9C	56320	DC
7424	1D	23808	5D	40192	9D	56576	DD
7680	1E	24064	5E	40448	9E	56832	DE
7936	1F	24320	5F	40704	9F	57088	DF
8192	20	24576	60	40960	A0	57344	E0
8448	21	24832	61	41216	A1	57600	E1
8704	22	25088	62	41472	A2	57856	E2
8960	23	25344	63	41728	A3	58112	E3
9216	24	25600	64	41984	A4	58368	E4
9472	25	25856	65	42240	A5	58624	E5
9728	26	26112	66	42496	A6	58880	E6
9984	27	26368	67	42752	A7	59136	E7
10240	28	26624	68	43008	A8	59392	E8
10496	29	26880	69	43264	A9	59648	E9
10752	2A	27136	6A	43520	AA	59904	EA
11008	2B	27392	6B	43776	AB	60160	EB
11264	2C	27648	6C	44032	AC	60416	EC
11520	2D	27904	6D	44288	AD	60672	ED
11776	2E	28160	6E	44544	AE	60928	EE
12032	2F	28416	6F	44800	AF	61184	EF
12288	30	28672	70	45056	B0	61440	F0
12544	31	28928	71	45312	B1	61696	F1
12800	32	29184	72	45568	B2	61952	F2
13056	33	29440	73	45824	B3	62208	F3
13312	34	29696	74	46080	B4	62464	F4
13568	35	29952	75	46336	B5	62720	F5
13824	36	30208	76	46592	B6	62976	F6
14080	37	30464	77	46848	B7	63232	F7
14336	38	30720	78	47104	B8	63488	F8
14592	39	30976	79	47360	B9	63744	F9
14848	3A	31232	7A	47616	BA	64000	FA
15104	3B	31488	7B	47872	BB	64256	FB
15360	3C	31744	7C	48128	BC	64512	FC
15616	3D	32000	7D	48384	BD	64768	FD
15872	3E	32256	7E	48640	BE	65024	FE
16128	3F	32512	7F	48896	BF	65280	FF

ANNEXE 3 -VALEURS DES TOUCHES

TOUCHE	HEXA	DECIMAL	TOUCHE	HEXA	DECIMAL
1	FDF7	65015	EDIT	FCF7	64759
2	FBF7	64503	AND	FAF7	64247
3	F7F7	63479	THEN	F6F7	63223
4	EFF7	61431	TO	EEF7	61175
5	DFF7	57335	(-	DEF7	57079
6	DFEF	57327	↓	DEEF	57071
7	EFEF	61423	↑	EEEF	61167
8	F7EF	63471	-)	F6EF	63215
9	FBEF	64495	GRAPHICS	FAEF	64239
0	FDEF	65007	RUBOUT	FCEF	64751
Q	FDFB	65019	""	FCFB	64763
W	FBFB	64507			
E	F7FB	63438	OR	FAFB	64251
R	EFFB	61435	STEP	F6FB	63227
T	DFFB	57339	(=	EEFB	61179
Y	DFDF	57311	<>	DEFB	57083
U	EFDF	61407	)=	DEDF	57055
I	F7DF	63455	$	EEDF	61151
O	FBDF	64479	(	F6DF	63199
P	FDDF	64991	)	FADF	64223
A	FDFD	65021	"	FCDF	64735
S	FBFD	64509	STOP	FCFD	64765
D	F7FD	63485	LPRINT	FAFD	64253
F	EFFD	61437	SLOW	F6FD	63229
G	DFFD	57341	FAST	EEFD	61181
H	DFBF	57279	LLIST	DEFD	57085
J	EFBF	61375	**	DEBF	57023
K	F7BF	63423	-	EEBF	61119
L	FBBF	64447	+	F6BF	63167
NEWLINE	FDBF	64959	=	FABF	64191
Z	FBFE	64510	FONCTION	FCBF	64959
X	F7FE	63486	:	FAFE	64254
C	EFFE	61438	;	F6FE	63230
V	DFFE	57342	?	EEFE	61182
B	DF7F	57215	/	DEFE	57086
N	EF7F	61311	*	DE7F	56959
M	F77F	63359	<	EE7F	61055
.	FB7F	64383	>	F67F	63103
ESPACE	FD7F	64895	,	FA7F	64127
neant	FFFF	65535	£	FC7F	64639

ANNEXE 4 - EFFET DES INSTRUCTIONS SUR LES INDICATEURS D'ÉTAT

r = registre simple ou constante 8 bits
rr = registre double

\+ affecté
— non affecté
0 mise à zéro

1 mis à un
? valeur aléatoire
x cas spécial

INSTRUCTIONS	INDICATEURS					
mnemonique	S	Z	H	V	N	C
ADC	+	+	+	+	0	+
ADD A,r	+	+	+	+	0	+
ADD rr,rr	-	-	+	-	0	+
AND	+	+	1	+	0	0
BIT	?	+	1	?	0	-
CALL	-	-	-	-	-	-
CCF (H prend la valeur de C)	-	-	X	-	0	+
CP	+	+	+	+	1	+
CPI CPD						
CPIR CPDR (pour ces 4 instructions P/V=0 si BC=0 et Z=1 si A=(HL))	+	X	+	X	1	-
CPL	-	-	1	-	1	-
DAA	+	+	+	+	-	+
DEC r	+	+	+	+	1	-
DEC rr	-	-	-	-	-	-
DI	-	-	-	-	-	-
DJNZ	-	-	-	-	-	-
EI	-	-	-	-	-	-
EX	-	-	-	-	-	-
EXX	-	-	-	-	-	-
HALT	-	-	-	-	-	-
IM 0,1,2	-	-	-	-	-	-
INC r	+	+	+	+	0	-
INC rr	-	-	-	-	-	-
IN A,(n)	-	-	-	-	-	-
IN r,(C)	+	+	+	+	0	-
INI IND (Z= si B=0)	?	X	?	?	1	-
INIR INDR	?	1	?	?	1	-
JP	-	-	-	-	-	-
JR	-	-	-	-	-	-
LD (seules les instructions LD suivantes affectent les indicateurs)	-	-	-	-	-	-

INSTRUCTIONS	INDICATEURS					
mnemonique	S	Z	H	V	N	C
LD A,I LD A,R	+	+	0	+	0	-
LDI LDD (P/V=0 si BC=0)	-	-	0	X	0	-
LDIR LDDR	-	-	0	0	0	-
NEG	+	+	+	+	1	+
NOP	-	-	-	-	-	-
OR	+	+	0	+	0	0
OUT	-	-	-	-	-	-
OUTI OUTD (Z=1 si B=0)	?	X	?	?	1	-
OUTIR OUTDR	?	1	?	?	1	-
POP	-	-	-	-	-	-
POP AF (valeur bits octet du sommet de la pile)	X	X	X	X	X	X
PUSH	-	-	-	-	-	-
RES	-	-	-	-	-	-
RET RETN RETI	-	-	-	-	-	-
RLA RLCA RRA RRCA	-	-	0	-	0	+
RL RLC RR RRC	+	+	0	+	0	+
RLD RRD	+	+	0	+	0	-
RST	-	-	-	-	-	-
SBC	+	+	+	+	1	+
SCF	-	-	0	-	0	1
SET	-	-	-	-	-	-
SLA SRA SRL	+	+	0	+	0	+
SUB	+	+	+	+	1	+
XOR	+	+	0	+	0	0

ANNEXE 5 - LES VARIABLES SYSTÈME

VARIABLES SYSTEME

deci.	hexa	nom
16384	4000	ERR.NR
16385	4001	FLAGS
16386	4002	ERR.SP
16388	4004	RAMTOP
16390	4006	MODE
16391	4007	PPC
16393	4009	VERSN
16394	400A	E.PPC
16396	400C	D.FILE
16398	400E	DF.CC
16400	4010	VARS
16402	4012	DEST
16404	4014	E.LINE
16406	4016	CH.ADD
16408	4018	X.PTR
16410	401A	STKBOT
16412	401C	STKEND
16414	401E	BERG
16415	401F	MEM
16417	4021	libre
16418	4022	DF.SZ
16419	4023	S.TOP
16421	4025	LAST.K
16423	4027	DB.ST
16424	4028	MARGIN
16425	4029	NXTLIN
16427	402B	OLDPPC
16429	402D	FLAGX
16430	402E	STRLEN
16432	4030	T.ADDR
16434	4032	SEED
16436	4034	FRAMES
16438	4036	COORDS
16440	4038	PR.CC
16441	4039	S.POSN
16443	403B	CDFLAG
16444	403C	PRBUFF
16477	405D	MEMBOT
16507	407B	libre
16508	407C	libre

BIBLIOGRAPHIE

- Manuel de programmation ZX 81, Steven Vickers, Sinclair Research Ltd. Direco International.
- Programmation du Z 80. Rodnay Laks - Sybex.
- Le petit livre du ZX 81. Trevor Toms. PSI.
- Programmer en assembleur. Alain Pinaud. PSI.
- La pratique du ZX 81. Xavier Linant de Bellefonds. PSI.
- Understanding your ZX 81 ROM. Ian Logan. Melbourne House Ltd. England.
- Mastering machine code on your ZX 81. Tony Baker. Interface. London. England.

Achevé d'imprimer
le 20 octobre 1982
sur les presses de l'Imprimerie Mollé
49000 Angers
Dépôt légal : 3e trimestre 1982
Réédition - Dépôt légal : octobre 1982

192 pages
96 F.F.
ISBN 2-904266-00-3

Comme de nombreux possesseurs du ZX 81 vous attendiez ce livre qui aborde le langage machine d'une manière résolument pratique.

L'étude des instructions du microprocesseur est menée en vue d'une utilisation immédiate sur le micro ordinateur SINCLAIR ZX 81 dont les particularités sont largement commentées.

Grâce à cet ouvrage, le ZX 81 devient un moyen didactique original et efficace pour l'initiation à la programmation du Z 80, le microprocesseur 8 bits le plus répandu actuellement.

INFORMATIQUE SERVICE
30, Rue Parcheminerie
49000 ANGERS

LANGAGE MACHINE ZX 81

André CHENIÈRE

AMSTRAD CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.
[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.
[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.