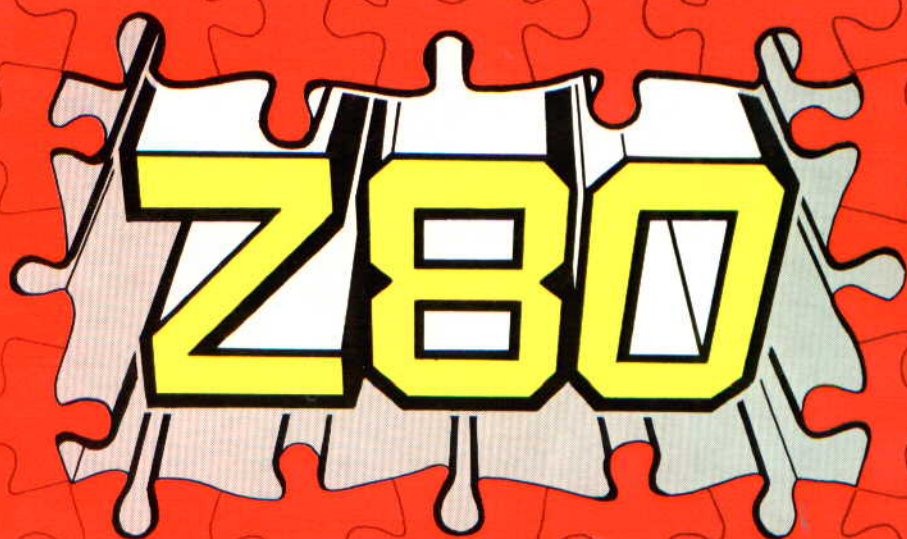


MICRO-ORDINATEURS



L'ASSEMBLEUR FACILE DU



Olivier LEPAPE



EYROLLES

Au bord du précipice

Ce livre veut être une introduction à la programmation en langage machine pour tous les possesseurs ou futurs possesseurs d'un micro-ordinateur conçu autour du microprocesseur Z80. Nous vous proposons donc de faire avec vous vos premiers pas à la découverte de ce nouveau langage qui enrichira beaucoup les possibilités de votre machine.

Il est possible que vous n'ayez actuellement aucune idée sur le langage machine; ne vous en effrayez pas, ceci n'est qu'une impression. En effet, rayons provisoirement le mot "machine", il ne reste plus que le mot "langage" et là nous retombons dans un domaine déjà connu de vous puisque vous utilisez tous un langage de programmation qui est le BASIC. Alors commençons par ce que nous connaissons et essayons de voir en quoi le langage machine appelé aussi "assembleur" peut ressembler au BASIC malgré les apparences. Il n'y a donc pas lieu de s'effrayer, nous n'allons pas vous plonger tout de suite dans la fosse aux "LD A, (8A4 H)", "DEC HL", "CALL SUBR" au risque de vous engloutir. Utilisons plutôt une échelle pour y descendre progressivement, vous verrez qu'en fin de compte les petites bêtes énumérées plus haut ne sont pas aussi antipathiques qu'elles en ont l'air.

Table des matières

Au bord du précipice : descente vers l'assembleur	V
1. Assembleur et BASIC	1
2. Le microprocesseur	9
3. Comment le microprocesseur calcule-t-il ?	14
3.1. Arithmétique 4 bits	16
3.2. Arithmétique 8 bits	18
3.3. L'addition	20
3.4. Les nombres négatifs	20
4. Les registres du Z 80	24
4.1. Registres d'usage général	25
4.2. Registres spécialisés	26
4.3. Définition de la pile	28
4.4. Registre F	28
5. Les modes d'adressage du Z 80	38
5.1. Adressage indirect par registre	39
5.2. Adressage indexé	39
5.3. Adressage par registre	40
5.4. Adressage implicite	41
5.5. Adressage immédiat	41
5.6. Adressage immédiat étendu	41
5.7. Adressage relatif PC	42
5.8. Adressage page zéro modifié	43

6. L'assembleur du Z 80	44
6.1. La syntaxe de l'assembleur	45
6.2. Fonctionnement d'un assembleur	51
6.3. Directives des assembleurs	53
7. Le jeu d'instruction du Z 80	59
7.1. Instructions de chargement 8 bits	59
7.2. Instructions de chargement 16 bits	64
7.3. Instructions de chargement immédiat	67
7.4. Instruction d'échange	68
7.5. Instructions arithmétiques 8 bits	70
7.6. Instruction d'ajustement décimal	74
7.7. Instructions logiques 8 bits	75
7.8. Instructions arithmétiques 16 bits	77
7.9. Instructions de saut	80
7.10. Sous-programmes	84
7.11. Instructions de manipulation de la pile	88
7.12. Instructions sur les bits	90
7.13. Instructions de décalage	91
7.14. Instructions d'entrée-sortie	95
7.15. Instructions de chaînes	96
7.16. Instructions d'usage général	99
7.17. Instructions sur les interruptions	100
7.18. Instructions de contrôle	100
ANNEXE 1. Liste des codes opérations par ordre numérique	101
ANNEXE 2. Liste des codes opérations par ordre alphabétique	107

1

Assembleur et BASIC

Aussi curieux que cela puisse paraître pour un livre sur l'assembleur, nous allons commencer par faire du BASIC. Ceci n'est pas absurde si l'on considère que BASIC et assembleur sont deux langages de programmation et que de ce fait il doit bien exister quelque chose de commun. Pour découvrir ces points communs essayons de résoudre le même problème en BASIC et en assembleur.

Réalisons un programme pour calculer la quantité :

$$1 + 2 + 4 + 8 + 16 + 32 + 64 \dots = \sum_{i=0}^N 2^i$$

Le petit programme suivant résoud ce problème :

```

10 R = 0
20 X = 1
30 N = 5
40 I = 0
50 IF I = N + 1 THEN GOTO 90
60 GOSUB 100
70 I = I + 1
80 GOTO 50
90 END
100 R = R + X
110 X = X + X
120 RETURN

```

Cette solution n'est pas forcément la meilleure, mais elle va nous aider à comprendre ce qu'est un langage de programmation.

Le programme se présente comme une succession de lignes écrites avec un langage particulier. On peut classer ces lignes en différents groupes. Au début il y a toute une série de définition et d'initialisation des variables nécessaires pour résoudre le problème. Dans le cas présent ces variables sont des nombres entiers. Il vient ensuite des lignes qui réalisent le traitement demandé. On peut séparer les lignes qui effectuent un calcul et les lignes qui sont des ordres de branchement et qui composent le "squelette" du programme.

On peut donc résumer le BASIC en trois groupes fondamentaux d'instructions :

Les instructions de branchement ou de décision	GOTO GOSUB RETURN IF THEN ELSE FOR NEXT
Les instructions arithmétiques et logiques	+ - * / SIN COS LOG EXP AND OR NOT etc...
Les instructions d'entrée-sortie	INPUT PRINT

Ces trois types d'instructions sont la base d'un langage de programmation. Les instructions de branchement forment le "squelette" du programme, les instructions arithmétiques et logiques réalisent les "fonctions organiques" du programme et les entrées-sorties sont l'ouïe et la parole du programme.

Il est temps maintenant de poser le pied sur notre premier échelon en disant que ces trois types d'instructions se retrouvent dans le langage machine ou l'assembleur avec un vocabulaire différent. Ainsi le GOSUB s'appellera CALL, le GOTO, JP, les opérations + et - seront ADD et SUB, etc... Cela signifie que l'analyse et la conception d'un programme informatique est la même, que ce soit en BASIC ou en assembleur, les divergences apparaîtront au niveau de la réalisation finale, c'est-à-dire l'écriture des lignes. L'exemple que nous avons écrit en BASIC peut s'écrire en assembleur de la façon suivante :

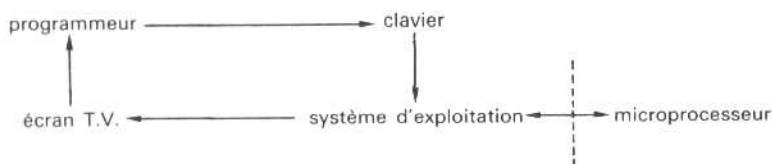
	LD A, 0	10 R = 0	
	LD (R), A		
	LD A, 1	20 X = 1	
	LD (X), A		
	LD A, 5	30 N = 5	
	LD (N), A		
	LD A, 0	40 I = 0	
	LD (I), A		
TEST	LD A, (N)	50 IF I = N + 1	
	INC A		
	LD HL, I		
	CP (HL)		
	JP Z, FIN		THEN GOTO 90
	CALL CALCUL	60 GOSUB 100	
	INC (HL)	70 I = I + 1	
	JP TEST	80 GOTO 50	
	JP END	90 END	
CALCUL	LD A, (R)	100 R = R + X	
	ADD A, (X)		
	LD (R), A		
	LD A, (X)	110 X = X + X	
	ADD A, (X)		
	LD (X), A		
	RET	120 RETURN	

La mise en parallèle de ces deux programmes écrits dans deux langages différents permet de faire plusieurs remarques. La première chose qui nous frappe est que la version assembleur nécessite beaucoup plus de lignes. En général les programmes écrits en assembleurs sont plus longs, ceci n'est pas forcément un handicap car le programme assembleur se décompose souvent sous forme de petits modules regroupant une série d'instructions. Par exemple :

$R = R + X$	$\Leftrightarrow$	LD A, (R)
		ADD A, (X)
		LD (R), A

Avec l'expérience chacun se forme sa propre bibliothèque de modules adaptés à chaque problème. Si la longueur des programmes est à l'avantage du BASIC, le temps d'exécution est par contre nettement à l'avantage du langage machine. L'instruction BASIC $R = R + X$ nécessite environ 1 ms pour être exécutée alors que les trois instructions langage machine équivalentes nécessitent seulement une dizaine de microsecondes, soit un rapport 100 sur la vitesse d'exécution. Ceci est très appréciable pour les programmes utilisant les possibilités graphiques de votre ordinateur.

Maintenant que nous avons mis en évidence un air de famille en ce qui concerne l'aspect purement langage informatique entre le BASIC et l'assembleur, essayons de voir si il n'y aurait pas aussi quelques bases communes du côté machine. L'ordinateur fonctionne de la manière suivante :



Il ressort de ce diagramme qu'il existe une barrière entre le programmeur et le microprocesseur. Il est impossible au programmeur de dire exactement au microprocesseur ce qu'il doit faire. Le microprocesseur est relié au monde extérieur par des fils électriques, il est donc évi-

dent que celui-ci est incapable de comprendre seul un mot BASIC. L'intermédiaire entre le clavier manipulé par le programmeur et le microprocesseur est le "système d'exploitation".

Le système d'exploitation est un programme en langage machine qui réside physiquement en ROM dans l'ordinateur. Ce programme interprète les signaux électriques provenant du clavier, renvoie des messages intelligibles sur l'écran et surtout traduit une instruction BASIC en une succession d'instructions machines matérialisées dans l'ordinateur par des signaux électriques à deux états 1 et 0. Une instruction machine ressemble donc à ceci :

1 0 1 1 0 0 1 1

Il y a donc une énorme différence entre ceci et :

$R = R + X$ ou LD A, (R)

Dans les deux cas le passage "texte" → "code machine" nécessite un programme spécial qui s'appelle un assembleur dans le cas de la programmation en assembleur ou un interpréteur dans le cas du BASIC. La différence réside dans le fait que l'assembleur génère un seul code machine par ligne alors que l'interpréteur génère une suite de codes machines réalisant l'instruction (environ 1 000).

On peut se demander alors pourquoi programmer en assembleur et ne pas continuer de profiter des avantages du BASIC qui condense une succession de codes machine en une seule instruction. Si l'on regarde l'exemple montré précédemment, l'assembleur va générer exactement un code machine par ligne soit 24 codes ; le BASIC quand à lui va effectuer sur chaque ligne les opérations suivantes :

- a) lire l'instruction,
- b) vérifier la syntaxe,
- c) transformer cette instruction en une suite de codes machine,
- d) exécuter l'instruction.

L'ensemble des opérations a, b, c, d nécessite environ 1 000 codes machine soit un total de 12 000 codes machine. Cette énorme différence entre le BASIC et l'assembleur provient du fait que dans le cas d'un programme assembleur les opérations a et b n'existent pas et l'opé-

ration c est pensée et réalisée par le programmeur. Il ne reste plus que la phase d. En BASIC le programmeur écrira $R = R + X$ et ne se posera plus de problèmes. En assembleur au stade $R = R + X$ le programmeur doit encore se demander comment réaliser ceci, il effectue donc l'opération c et écrit :

```
LD  A, (R)
ADD A, (X)
LD  (R), A
```

Les principaux avantages du langage machine sont :

- *Exécution plus rapide du programme.*
- *Utilisation plus efficace de l'espace mémoire.*
- *Programmes moins imposants en mémoire.*
- *Liberté par rapport au système d'exploitation.*

Les principaux désavantages du langage machine sont :

- *Programmes difficiles à lire et à mettre au point.*
- *Programmes non adaptables à d'autres ordinateurs.*
- *Programmes nécessitant plus de lignes.*
- *Difficultés d'effectuer des calculs arithmétiques compliqués.*

Il faut donc faire un choix sur le langage employé pour résoudre un problème.

Le BASIC se prête bien à la résolution de problèmes mathématiques ou de gestion simple.

L'assembleur est très adapté aux problèmes de jeux avec animation graphique.

Jusqu'à maintenant nous avons maintenu une confusion entre langage machine et langage assembleur. En toute rigueur le langage machine n'est composé que de 1 et de 0. Il n'est pas facile à comprendre :

```
10000010
```

Pour cela on a été amené à créer un intermédiaire entre l'homme et la machine, cet intermédiaire est l'assembleur. Chaque code machine est remplacé par un mnémonique. Il est déjà plus facile de lire :

```
ADD A, D
```


La suite du livre est là pour vous expliquer que ADD signifie "addition" et que A et D sont deux variables dont nous reparlerons. L'assembleur va se charger d'effectuer la translation.

ADD A, D → 10000010

Le nombre binaire 10000010 peut aussi être représenté par un équivalent hexadécimal :

82

Cette représentation est certes plus lisible mais elle ne décrit pas plus ce que fait l'instruction. Toutefois certains KITS microprocesseur utilisent uniquement cet intermédiaire hexadécimal pour programmer en langage machine.

1.1. ASSEMBLEUR CONTRE BASIC

Maintenant que nous avons posé nos deux pieds sur le premier échelon, regardons le chemin parcouru. Nous voulons écrire un programme pour notre ordinateur ; la démarche à suivre est la suivante :

— *analyse du problème :*

Cette phase entraîne la dissection du gros problème primaire en une multitude de petits problèmes secondaires que nous appellerons "modules".

— *agencement des différents modules :*

Il s'agit maintenant de recoller ces modules entre eux pour former le squelette du programme. Chaque module est une fonction bien précise dont le schéma est :

variables d'entrées → traitement → variables de sorties

— *réalisation du programme :*

A ce stade il faut faire un choix entre BASIC et assembleur en suivant les critères énoncés plus haut :

1.1.1. Assembleur

traitement nécessitant peu de calcul,
traitement manipulant beaucoup de données graphiques,
traitement rapide.

1.1.2. BASIC

beaucoup de calculs mathématiques,
vitesse d'exécution peu critique,
données uniquement numériques.

A ce propos un programme BASIC rempli de PEEK et de POKE est généralement plus facilement réalisable en assembleur.

Vous voyez donc que vous savez déjà réaliser les deux premières étapes grâce au BASIC, alors, laissez-vous tenter par la solution assembleur de la troisième étape. Au début cela demande un petit effort qui sera pleinement récompensé lorsque vous découvrirez les possibilités illimitées qu'offrent l'assembleur.

2

Le microprocesseur

Avant d'apprendre en détail le langage de ce curieux individu il est important de le connaître pour savoir ce qu'il fait et comment il le fait.

On vous a peut être présenté le microprocesseur comme le cerveau, le "petit génie" qui anime votre ordinateur. Il faut dès maintenant vous débarrasser de cette idée et considérer le microprocesseur comme un individu fort simple à qui l'on demande d'accomplir un grand nombre de tâches le plus rapidement possible. Si votre ordinateur joue aux échecs ce n'est pas à cause de l'intelligence du microprocesseur mais à cause de celle du programmeur qui a su diviser le problème en un grand nombre de tâches élémentaires qu'il fait exécuter ensuite par le microprocesseur. Ces tâches sont essentiellement des calculs.

Notre microprocesseur est donc un individu qui peut exécuter un certain nombre de calculs successifs rapidement sans se soucier de savoir à quoi cela va servir. Ce problème là est celui du programmeur. Prenons un exemple pour entrer plus dans le détail.

Vous voulez faire calculer par votre microprocesseur la somme qu'il vous reste sachant que vous possédiez 57 F et que vous avez dépensé 23 F. Vous décomposez le problème, pour le mettre à la portée du microprocesseur :

- 1°) mettre le nombre 57 dans la case 1,
- 2°) mettre le nombre 23 dans la case 2,

- 3°) prendre le contenu de la case 1,
- 4°) soustraire le contenu de la case 2,
- 5°) mettre le résultat dans la case 3.

Maintenant le microprocesseur peut effectuer les tâches 1, 2, 3, 4, 5 et donner le résultat 34 dans la case 3. Toutes ces manipulations de nombres, de cases, d'opérations arithmétiques seraient très pénibles si le microprocesseur devait les exécuter de tête. C'est pourquoi celui-ci va utiliser ses doigts pour compter. Un microprocesseur compte avec ses mains !!!

Comme nous le verrons plus tard le Z 80 possède un grand nombre de mains que l'on appelle plus scientifiquement des registres. Le microprocesseur possède une main particulière avec laquelle il effectue les calculs arithmétiques et logiques : la "main A". Il interprète les lignes 3, 4, 5 de la façon suivante :

- 3°) compter sur la main A la valeur inscrite dans la case 1,
- 4°) soustraire sur les doigts de la main A la valeur inscrite dans la case 2,
- 5°) ranger dans la case 32 la valeur enregistrée par la main A.

On peut déjà faire quelques remarques sur le microprocesseur à partir de cet exemple :

- Le microprocesseur ne peut pas faire de calculs avec des nombres décimaux comme 13,25. Les mains ne peuvent compter qu'avec des nombres entiers.
- Le microprocesseur est limité par le nombre des doigts d'une main sur la taille des entiers à traiter. Le microprocesseur possède des mains de 8 doigts, ce qui lui permet de compter jusqu'à 255 comme nous le verrons plus tard.
- Le microprocesseur ne sait pas que le nombre 57 représente une somme d'argent. Il ne se soucie absolument pas de savoir ce que représente la donnée qu'il manipule.

Le concept de variable que vous avez appris avec le BASIC est très différent en assembleur. En BASIC une variable est uniquement un

nombre entier ou décimal ou un caractère alphanumérique. Chaque variable possède un nom et le programmeur n'a plus à se soucier de savoir comment l'ordinateur va traiter ces variables. La grosse différence avec l'assembleur réside dans le fait que maintenant le programmeur doit se soucier de la façon dont il va *coder* les données du problème afin de pouvoir les traiter par le microprocesseur. Comme nous l'avons dit le microprocesseur calcule avec ses doigts, il faut donc adapter la représentation des variables à cette méthode de calcul.

L'une des grandes différences entre le BASIC et l'assembleur est cette absence de variables définies à l'avance.

L'assembleur va donc poser un petit problème supplémentaire qui est le codage des données. Pour l'exemple énoncé plus haut ce codage était évident. En effet, sachant que le microprocesseur peut compter jusqu'à 255 avec une main, les sommes 57 F et 23 F ont été codées par les nombres 57 et 23. Il aurait fallu trouver une autre solution si l'on avait 450 F car le nombre 450 ne tient pas sur une seule main du microprocesseur. D'autres problèmes peuvent se présenter ; par exemple comment représenter l'alphabet en utilisant les doigts du microprocesseur ? Ainsi la case 1 qui contient au début le nombre 57 représente une somme d'argent. Plus tard on peut très bien décider que cette case contiendra un code représentant la lettre F et encore plus tard cette case représentera un code indiquant qu'une touche a été enfoncée sur le clavier, etc... Seul le programmeur décide de ce que représente le contenu de la case 1 au cours du déroulement du programme.

Le concept de variable est donc plus compliqué en assembleur, *mais aussi beaucoup plus riche*. On peut très bien personnaliser les doigts d'une main. Par exemple on peut imaginer que le premier doigt représente le signe + ou - et les 7 autres doigts représentent une valeur numérique, ou encore que les 4 premiers représentent la partie entière d'un nombre décimal et les 4 derniers doigts représentent la partie fractionnaire. Tiens tiens... il est donc possible de travailler avec des nombres négatifs et des nombres décimaux ? Oui, mais il ne faut pas oublier que seul le programmeur est au courant de son codage, il doit donc adapter le traitement en conséquence. Le microprocesseur quand à lui ne voit que des nombres compris entre 0 et 255 qu'il peut représenter sur une de ses mains.

De même qu'il a été possible de fabriquer la tour Eiffel avec seulement des morceaux de métal et des rivets, il est possible de faire jouer aux échecs un microprocesseur qui a la seule faculté de savoir compter sur ses mains de 8 doigts. Vous vous trouvez en face d'une personne qui ne sait faire que des additions.

Problème : vous voulez lui faire réaliser l'opération 10×25 .

Solution : faites lui additionner 10 fois le nombre 25.

La personne en question aura effectué une multiplication sans le savoir et vous donnera le bon résultat. L'intérêt devient grand si l'on considère qu'il vous faut 0,1 s pour réaliser cette multiplication alors que la personne à qui vous vous êtes adressé effectue l'addition en 3 μ s. Celle-ci mettra donc 30 μ s pour effectuer la multiplication. Ceci explique pourquoi le microprocesseur peut jouer aux échecs : *il est rapide et ne fait pas d'erreurs de calcul.*

Maintenant que vous connaissez un peu mieux les possibilités d'un microprocesseur vous comprenez pourquoi les programmes assembleurs sont longs mais rapides. Pour en revenir au langage. L'exemple donné plus haut s'écrit :

INITIALISATION :

```
LD   A, 57
LD   (CASE 1), A
LD   A, 23
LD   (CASE 2), A
```

CALCUL :

```
LD   A, (CASE 1)
SUB  A, (CASE 2)
LD   (CASE 3), A
```

A est le nom attribué à la main A privilégiée du Z 80. LD est un mnémonique qui signifie LOAD (charger). Les mots mis entre parenthèses signifient "contenu de".

(CASE 1) = contenu de CASE 1

Vous voyez donc que l'assembleur n'est pas si compliqué. Il reste un petit point troublant :

Nous avons 10 doigts et nous comptons donc jusqu'à 10 sur nos deux mains. Alors comment le microprocesseur peut-il compter jusqu'à 255 avec seulement 8 doigts ? La réponse se trouve un échelon plus bas.

3

Comment le microprocesseur calcule-t-il ?

Le microprocesseur compte sur ses doigts d'une façon beaucoup plus organisée que nous. Si l'on représente par 1 un doigt déplié et par 0 un doigt plié nous ne faisons pas la différence entre les deux représentations :

100 001

Le microprocesseur par contre fait une différence entre son pouce et l'index lorsqu'il compte sur ses doigts. Il ordonne ses huit doigts et avec seulement deux doigts il peut compter jusqu'à trois



00 = 0



01 = 1



10 = 2



11 = 3

La règle de comptage est la suivante :

à chaque incrémentation le premier doigt change d'état (plié, déplié) chaque doigt change d'état si le doigt précédent passe de l'état déplié à l'état plié. De cette façon on obtient 256 configurations possible des 8 doigts de chaque main du microprocesseur, ce qui lui permet de compter de 0 à 255. Essayez avec 3 doigts, vous pourrez compter de 0 à 7.

Cette façon de compter sur ses doigts est en relation directe avec la numération binaire. Une main du Z 80 est matérialisée par un ensemble de 8 cases pouvant enregistrer deux valeurs 0 ou 1.

La position d'une main est représentée par une séquence de huit chiffres binaires (0 ou 1) appelés bits :



⇔

10011010

Pour simplifier cette représentation on coupe la main en deux demi mains de 4 doigts. Étant donné que l'on peut compter de 0 à 15 avec 4 doigts les 16 positions possibles sont représentées par un chiffre :

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

ainsi :

$$\begin{array}{c|c} 1001 & 1010 \\ \hline \end{array} \quad \begin{array}{l} 1001 \rightarrow 9 \\ 1010 \rightarrow A \end{array}$$

est représenté par le nombre hexadécimal

9A

La représentation hexadécimale est très employée en assembleur du fait que l'unité de base qui sert à tout calcul est la main de 8 doigts. Ainsi le programme assembleur se chargera de convertir les nombres hexadécimaux en binaire. La représentation du binaire par des chiffres hexadécimaux est la convention la plus employée pour plusieurs raisons :

1. Conversion hexadécimale binaire simple.
2. Distinction des nombres de 8 et 16 bits.
8 bits = 2 chiffres hexadécimaux,
16 bits = 4 chiffres hexadécimaux.

Cette convention permet de représenter n'importe quel registre ou n'importe quelle case mémoire par deux chiffres hexadécimaux. Tout l'environnement du Z 80 est organisé en blocs de 8 bits ou 16 bits :

registre : 8 bits
mémoire : 8 bits
adresse : 16 bits

3.1. L'ARITHMÉTIQUE 4 BITS

Il est important de se familiariser avec cette arithmétique pour comprendre comment calcule un microprocesseur tel que le Z 80. Il faut donc commencer par savoir passer de la représentation binaire à la représentation décimale ou hexadécimale.

$$1111 = 8 + 4 + 2 + 1 = 15 \quad \text{décimal}$$

F hexadécimal



On peut remarquer que chaque doigt représente les puissances de 2 croissantes. Pour obtenir la valeur décimale il faut sommer les puissances de 2 correspondant à chaque doigt déplié :

$$\text{doigt 1 : } 2^0 = 1$$

$$\text{doigt 2 : } 2^1 = 2$$

$$\text{doigt 3 : } 2^2 = 4$$

$$\text{doigt 4 : } 2^3 = 8$$

ainsi la main suivante représente la valeur 10



$$= 1010$$

binaire

$$= 8 + 0 + 2 + 0 = 10$$

décimal

$$= A$$

hexadécimal

Pour éviter les confusions entre les nombres décimaux et les nombres hexadécimaux, on fait suivre ceux-ci par la lettre H :

10, 38 décimal

AH, 38 H hexadécimal

Attention!!! 38 H ne représente pas le même nombre que 38.

Il est très important de savoir jongler entre ces différentes représentations numériques, c'est pourquoi nous vous proposons ce petit exercice avant de passer à la suite :

Faites la conversion "binaire → décimal" et "binaire → hexadécimal"

décimal

hexadécimal

0010:

0110:

1001:

1010:

1100:

par exemple :

$$1101 = 8 + 4 + 0 + 1 = 13 \text{ (décimal)}$$

DH (hexadécimal)

3.2. L'ARITHMÉTIQUE 8 BITS

L'arithmétique sur 4 bits ne peut traiter que des nombres compris entre 0 et 15. Si nous voulons représenter le nombre 16 il faut rajouter des doigts à notre main. Ce qui donne sur la main de 8 doigts du Z 80:



= 16 décimal

= 10 H hexadécimal

On remarque que $16 = 2^4$ ce qui élargit la méthode exposée sur 4 bits:

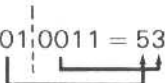
$$0001\ 000 = 0 + 0 + 0 + 16 + 0 + 0 + 0 + 0 = 16$$

$$0001;000 = 10\ H$$

En hexadécimal il suffit de couper la main en deux et de représenter chaque demi-main de 4 doigts par son équivalent hexadécimal. Par exemple :

$$0101\ 0011 = 0 + 2^5 + 0 + 2^4 + 0 + 0 + 2^1 + 2^0$$

$$0 + 64 + 0 + 16 + 0 + 0 + 2 + 1 = 83 \quad \text{décimal}$$

$$0101\ 0011 = 53\ H$$


hexadécimal

On comprend mieux maintenant pourquoi l'hexadécimal est plus facile à manier que le décimal. Il reste à se poser maintenant le problème de la conversion hexadécimal en décimal.

Pour nous qui sommes habitués au système décimal lorsque nous voyons le nombre 83 on effectue immédiatement l'opération

$$83 = (8 * 10) + 3$$

Il se produit exactement la même chose en hexadécimal ainsi la valeur 53 H représente :

$$53\ H = (5 * 16) + 3 = 80 + 3 = 83$$

Il n'y a donc aucun problème pour convertir de l'hexadécimal en décimal. Si l'on regarde la capacité d'une main de huit doigts :



$$= 1111\ 1111$$

$$= FF\ H$$

$$= (15 * 16) + 15 = 255$$

Entraînez-vous maintenant à convertir n'importe quel nombre compris entre 0 et 255 en hexadécimal, en binaire, en décimal. Il est important de bien savoir jongler entre ces différents modes de numération.

décimal 27 28 29 30

hexadécimal 27 H 28 H 29 H 2 AH 2 BH 2 CH 2 DH 2 EH 2 FH 30 H

La valeur suivant 29 H est 2 AH et non 30 H. D'autre part 28 H ne représente pas le même nombre que 28.

3.3. L'ADDITION

L'addition binaire est extrêmement simple; il suffit de retenir les formules suivantes :

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

table d'addition binaire

L'addition s'effectue de la même manière qu'en numération décimale en employant la table d'addition binaire. Le cas $1 + 1 = 10$ entraîne une retenue au rang supérieur exactement comme $8 + 9 = 17$ ($7 + 10$) en numération décimale. L'addition de deux nombres de 8 bits se fait ainsi :

00101101	=	45	2DH
01110100	=	116	74H
<u>10100001</u>	=	<u>161</u>	<u>A1H</u>

3.4. NOMBRES NÉGATIFS

Nous avons déjà donné une idée sur la représentation des nombres négatifs en disant que le bit le plus à gauche pouvait représenter le signe et les 7 bits restant la valeur absolue du nombre. Avec ce système il y a 128 positions des doigts pour lesquelles le doigt de gauche est déplié et 128 positions pour lesquelles il est plié. Ce qui donne 128 nombres négatifs et 128 nombres positifs. Étant donné que 0 est positif (bit 7 = 0) cela permet de compter dans l'intervalle :

$$-128 \text{ à } 127$$

Comme nous l'avons déjà dit, le microprocesseur ne sait pas si le doigt de gauche représente le signe du nombre ou pas. C'est donc au programmeur de décider s'il considère un certain code binaire comme un

nombre signé (bit 7 = signe + ou -) ou comme une valeur positive sur 8 bits. Toutes les instructions du Z 80 sont compatibles avec une représentation signée à condition de choisir correctement une représentation des nombres négatifs.

Il est logique de coder la valeur 0 par :

00000000

La représentation des nombres négatifs doit être telle que l'addition d'un nombre et de son opposé donne un résultat nul. Pour cela il ne suffit pas d'inverser le bit 7 :

$$\begin{array}{rcl}
 0\ 0100100 & = & +\ 36 \\
 1\ 0100100 & = & -\ 36\ ??? \\
 \hline
 1\ 1001000 & = & -\ 72\ !!!
 \end{array}
 \qquad
 \begin{array}{l}
 0 = \text{signe } + \\
 1 = \text{signe } -
 \end{array}$$

Cette représentation n'est pas bonne. On représente un nombre négatif en effectuant les étapes suivantes :

$$1^{\circ})\ 36 = 0\ 0100100$$

2^o) on inverse chaque bit ce qui donne :

11011011

3^o) on additionne 1 :

$$\begin{array}{r}
 11011011 \\
 00000001 \\
 \hline
 11011100 \\
 -36 = 1\ 11011100
 \end{array}$$

Si maintenant on effectue $(+ 36) + (- 36)$ on obtient :

$$\begin{array}{r}
 00100100 \\
 11011100 \\
 \hline
 \text{(retenue)} \quad \swarrow \quad 00000000 \\
 \text{(perdue)} \quad \times
 \end{array}$$

La retenue au rang 8 est perdue car les mains du microprocesseur n'ont que 8 doigts. Le résultat final est donc 0. Avec cette représentation le nombre positif le plus grand est :

$$\underline{01111111 = +127}$$

Le nombre négatif le plus petit est :

$$\underline{10000000 = -128}$$

La règle énoncée plus haut est aussi valable pour trouver l'opposé d'un nombre négatif :

$$1^{\circ}) -36 = \begin{array}{c} | \\ 11011100 \\ | \end{array}$$

2°) on inverse chaque bit :

$$00100011$$

3°) on additionne 1 :

$$\begin{array}{r} 00100011 \\ 00000001 \\ \hline 00100100 \\ 36 = 0 \begin{array}{c} | \\ 0100100 \\ | \end{array} \end{array}$$

Cette représentation des nombres négatifs s'appelle le complément à deux. La méthode énoncée est valable quel que soit le nombre de bits choisis. Dans tous les cas le bit le plus à gauche représente le signe :

$$0 = +$$

$$1 = -$$

N'oubliez pas que ceci n'est qu'une convention et que le programmeur décide de lui-même quand le nombre codé représente une valeur positive (0, 255) ou une valeur signée (-128, +127). Avec la convention choisie (complément à 2) le microprocesseur donnera toujours le résultat correct :

$$\left\{ \begin{array}{l} 00101101 = 45 \\ 10001000 = 136 \end{array} \right.$$

$$45 + 136 = 181$$

ou

$$\left\{ \begin{array}{l} 00101101 = +45 \\ 10001000 = -120 \end{array} \right.$$

$$(+45) + (-120) = -75$$

Dans les deux cas, lors d'une addition le microprocesseur effectue :

$$\begin{array}{r} 00101101 \\ 10001000 \\ \hline 10110101 \end{array}$$

$$10110101 = 181$$

ou

$$\begin{array}{c} | \\ 1|0110101 = -75 \\ | \end{array}$$

Pour exercice, effectuez le complément à 2 de -75 et vérifiez que l'on obtient 75.

Ceci montre donc que le programmeur a le libre choix de la convention adoptée pour représenter les nombres. Dans les deux cas le microprocesseur calcule de la même façon et donne un résultat correct.

Il est important d'avoir les idées claires sur ces conventions. Nous vous conseillons beaucoup de prendre un papier un crayon et d'effectuer à la main tous les calculs que nous vous avons exposés. Ceci n'est pas très drôle mais reste très formateur.

4

Les registres du Z 80

La configuration des registres du Z 80 est la suivante :

A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

IX
IY
SP
PC

I	R
---	---

On peut classer ces registres en deux catégories. Les registres d'usage général et les registres spécialisés.

4.1. REGISTRES D'USAGE GÉNÉRAL

Ce sont les registres B, C, D, E, H, L, B', C', D', E', H', L' qui peuvent contenir chacun un octet. On peut considérer ces registres comme de simples cases mémoires directement accessibles par le microprocesseur sans passer par l'intermédiaire d'une adresse. Ces cases ne possédant pas d'adresse, on les distingue les unes des autres en leur attribuant un nom. Ainsi une instruction du type :

"faire l'opération X sur le contenu de la case mémoire d'adresse Y" devient :

"faire l'opération X sur le contenu du registre B" par exemple.

Ces registres peuvent être associés deux à deux pour former un seul registre de 16 bits. Dans ce cas on ne peut plus considérer une paire de registres comme une case mémoire ordinaire. Cette association de deux registres permet au Z 80 de posséder un certain nombre d'instructions opérant sur des données de 16 bits. Ces paires de registres ne sont pas quelconques. On peut associer deux à deux les registres "BC", "DE", "HL", "B'C'", "D'E'", "H'L'". L'octet de poids fort est contenu dans le premier registre (B, D, H, B', D', H'), l'octet de poids faible est dans le second registre (C, E, L, C', E', L').

Le microprocesseur ne peut pas accéder directement à l'ensemble de ces registres. Il faut considérer que l'on possède deux jeux interchangeables de registres. On sélectionne un jeu, l'autre est mis de côté et conserve évidemment son information. On peut comparer ces deux jeux de registres à ces tableaux noirs à deux faces pivotant autour d'un axe central. La face active sur laquelle on écrit correspond au jeu de registre (B, C, D, E, H, L), la face cachée par contre correspond au jeu mis de côté (B', C', D', E', H', L'). Le microprocesseur possède une instruction du style "faire pivoter le tableau de 180°". La face précédemment écrite est mise au repos en conservant ce que l'on y avait inscrit et l'on travaille maintenant sur l'ancienne face cachée. Ce type de tableau est très pratique pour un professeur qui par exemple conserverait les résultats principaux sur une face et effectuerait les calculs intermédiaires et secondaires sur l'autre face. Cette démarche est très souvent employée en

programmation, nous reprendrons ce point sous le nom plus savant de "sauvegarde du contexte". Pour l'instant mettons-nous dans la tête que le Z 80 possède deux jeux de registres, l'un est actif l'autre est au repos et à tout moment on peut échanger ces deux jeux par une instruction appropriée.

4.2. LES REGISTRES SPÉCIALISÉS

Ce sont les registres A, F, A', F', IX, IY, SP, PC, I, R. On peut noter que certains de ces registres contiennent des mots de 8 bits (ceux dont le nom ne possède qu'une lettre) et d'autres des mots de 16 bits (ceux dont le nom s'écrit avec deux lettres).

4.2.1. Registres A et A'

A comme accumulateur. Le microprocesseur possède deux accumulateurs mais n'en utilise qu'un à la fois, c'est l'accumulateur A. A' est mis au repos. L'accumulateur étant utilisé pour effectuer des opérations combinatoires ($A = A \square \text{donnée}$; $\square$ représente une opération quelconque) on lui associe un registre spécial appelé "Flag" ou F (et F'). Ce "Flag" contient des données binaires concernant le résultat de l'opération effectuée. En aucun cas les registres A et F (A' et F') ne peuvent être associés pour effectuer un unique registre de 16 bits. Les données contenues dans A et F sont corrélées mais de natures très différentes. Les deux jeux "accumulateur" "flag" AF et A'F' forment aussi un petit tableau noir à deux faces que le microprocesseur peut faire pivoter indépendamment des registres d'usage général (B, C, D, E, H, L, B', C', D', E', H', L').

4.2.2. Registre PC

Ce registre est le compteur ordinal. Il contient un mot de 16 bits qui représente l'adresse de l'instruction suivante. A chaque instruction il est incrémenté autant de fois que le nombre d'octets nécessaire au codage de l'instruction avec ses opérandes. On peut modifier par programme la

valeur du PC ce qui permet de modifier l'adresse de l'instruction suivante et de faire des sauts dans le programme. Ceci est l'équivalent des GOTO et GOSUB du BASIC.

4.2.3. Registre IX

Ce registre de longueur 16 bits s'appelle le registre d'index. Il est utilisé par le microprocesseur lors d'un mode d'adressage spécial appelé adressage indexé (ou basé). La donnée contenue par IX est l'adresse d'une case mémoire particulière, choisie par le programmeur. Le contenu de IX peut pointer sur toutes les cases de la mémoire centrale grâce à sa longueur de 16 bits. Ce registre n'est pas indispensable comme A, F, PC mais il facilite beaucoup de problèmes lors du traitement de tables par exemple.

4.2.4. Registre IY

Ce registre est un autre index d'usage identique à IX.

4.2.5. Registres I et R

Ces deux registres de 8 bits ne sont d'aucune utilité pour le programmeur. Ils sont utilisés uniquement pour certaines configurations "matériels". A titre de culture générale le registre R est utilisé pour un certain type de mémoire appelée RAM dynamique.

4.2.5. Registre SP

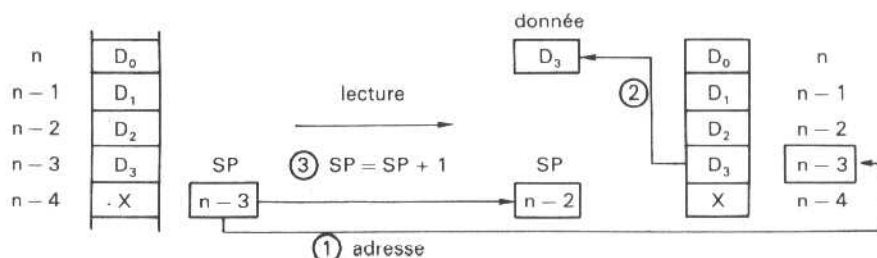
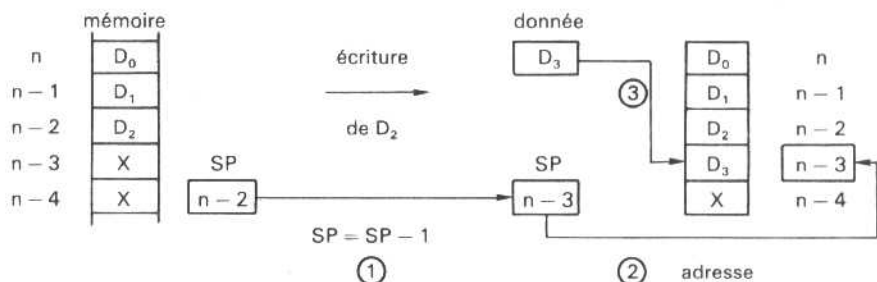
Ce registre est le pointeur de pile. Il est très important de bien comprendre le fonctionnement de ce registre que nous allons décrire. Nous verrons ensuite son utilisation par le microprocesseur et par le programmeur. Le registre SP contient un mot de 16 bits qui représente une adresse de la mémoire centrale. Comme son nom peut le laisser sous-entendre le pointeur de pile sert à gérer une pile.

4.3. DÉFINITION DE LA PILE

Une pile est une succession de cases mémoires contiguës dont l'accès est géré par le registre SP. Cet accès d'un type spécial peut se comparer à l'empilement d'assiettes dans une armoire. Chaque donnée à écrire dans la pile serait l'équivalent d'une nouvelle assiette placée au sommet de la pile. Chaque écriture par le microprocesseur dans la pile ne fait qu'augmenter le tas de données empilées en les conservant. Par contre chaque lecture de la pile équivaut à retirer l'assiette supérieure qui est la seule accessible sans risque de casse. On conçoit ainsi que la première donnée lue dans la pile sera celle qui a été écrite en dernier. Et à chaque lecture on désempile pour accéder aux données plus profondes. Ce mode d'accès s'appelle pile "dernier entré premier sorti" ou "Last In First Out" ou plus couramment pile L I F O.

La gestion de cette pile par le registre SP est très simple. Au départ lorsque la pile est vide SP contient l'adresse d'une case mémoire. A chaque écriture dans la pile le registre SP est décrémenté de 1 puis la donnée est écrite dans la case mémoire dont l'adresse est contenue dans le registre SP. Les données de la pile sont ainsi rangées de façon contiguës par adresses décroissantes. A chaque lecture dans la pile il se produit la démarche inverse. C'est-à-dire qu'on lit la case mémoire pointée par SP puis on incrémente de 1 ce registre.

Ces opérations sont résumées dans le schéma suivant : Les différentes opérations sont notées chronologiquement 1 , 2 , 3.



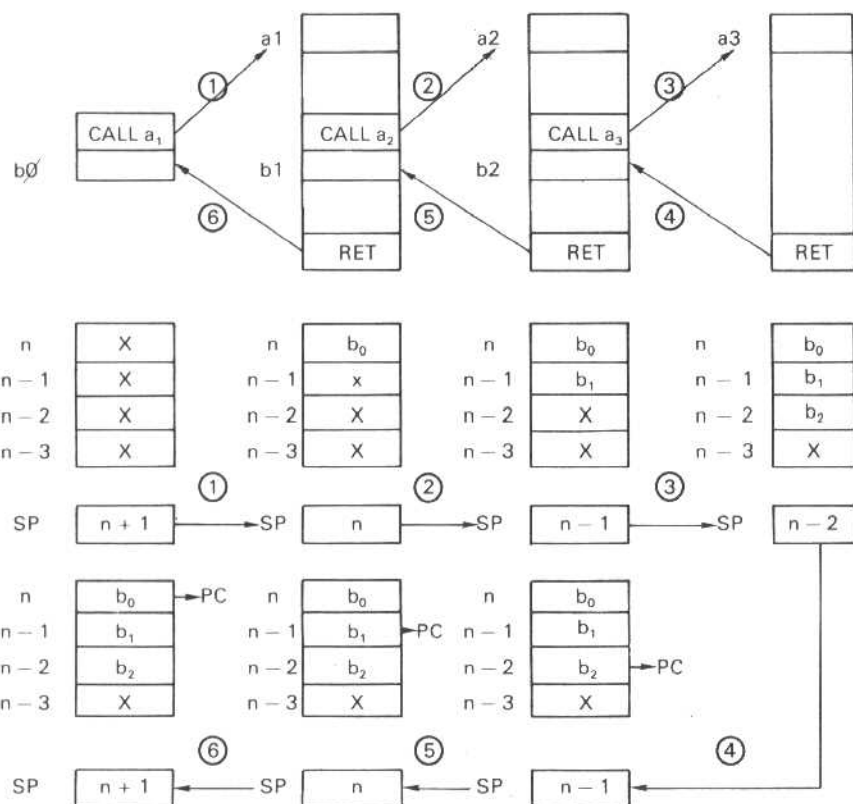
On remarque que la lecture n'affecte pas les valeurs rangées dans la pile. Seul le registre SP est modifié.

La pile est un élément essentiel au fonctionnement du microprocesseur. Il est donc primordial en début de chaque programme d'avoir pensé à réserver une zone mémoire pour la pile et d'initialiser le registre SP avec l'adresse supérieure de cette zone mémoire. Je ne vous décris pas le désastre si à la suite de cet oubli le registre SP pointe sur la zone mémoire où réside le programme ou des données.

Utilisation de la pile par le microprocesseur :

La pile est utilisée lors de l'appel de sous-programmes. Vous connaissez tous le GOSUB du BASIC et son associé RETURN. Ces deux instructions ont un équivalent en langage machine qui sont CALL nn et RET. Lorsque le microprocesseur rencontre CALL nn il se branche à l'adresse nn et poursuit son programme jusqu'à l'encontre de l'instruction RET. Dans ce cas il faut revenir à l'instruction suivant CALL nn.

Cela signifie que le microprocesseur a conservé quelque part cette adresse. Ce quelque part est la pile. Le fonctionnement de la pile permet d'imbriquer les sous-programmes de façon presque illimitée. En effet, la dernière adresse de retour empilée est celle correspondant au premier RET rencontré. On se rappelle ici du fameux LIFO et le schéma suivant devient clair :



L'adresse de l'instruction suivant le CALL est dans le PC. Chaque CALL empile le PC et le charge ensuite avec l'adresse de début de sous-programme. Chaque RET désempile le PC. On remarque que le PC faisant 16 bits de large il nécessite deux empilements pour le ranger ainsi lors d'un CALL.

$$(SP - 1) = PC_H \quad \text{et} \quad (SP - 2) = PC_L \quad SP = SP - 2$$

lors d'un RET :

$$PC_L = (SP) \quad \text{et} \quad PC_H = (SP + 1) \quad SP = SP + 2$$

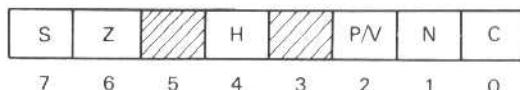
La notation (SP) signifie "contenu de la case mémoire dont l'adresse est contenue dans SP".

Utilisation de la pile par le programmeur :

La pile est souvent pratique pour ranger des données temporaires. Plusieurs instructions sont à la disposition du programmeur pour empiler ou dépiler les données contenues dans les registres du Z 80. Nous verrons comment cette pile peut être utilisée pour sauvegarder le contexte et pour passer des paramètres lors d'appels de sous-programmes.

4.4. REGISTRE F

Ce registre de 8 bits contient des indicateurs (flag) binaires dont la valeur (0 ou 1) dépend de la dernière instruction effectuée par le micro-processeur. Chaque bit de ce registre a une signification bien particulière que nous allons décrire. La structure de ce registre est la suivante :



4.4.1. Bit 0 ou C

Ce bit est l'indicateur de retenu (carry). Il est positionné par les opérations arithmétiques de décalage et de comparaison. Il peut être mis à 1 par une instruction spéciale (SCF). Les opérations logiques (AND, OR, XOR) mettent ce bit à 0.

Addition :

Le Z 80 peut effectuer l'addition de deux octets. Le résultat est lui-même un octet. Cela signifie que les données et le résultat ont une valeur comprise entre 0 et 255. Raisonnons en numération décimale pour simplifier la compréhension. Nous possédons un microprocesseur fictif travaillant sur des nombres compris entre 0 et 99 en numération décimale. Si nous essayons d'effectuer :

$$\begin{array}{r} 70 \\ + 80 \\ \hline 150 \end{array}$$

Le résultat 150 est supérieur à 99. Dans ce cas le microprocesseur donne un résultat égal à 50 et une retenue aux centaines, donc $C = 1$. Par contre l'addition de 60 et 25 donne 85. Dans ce cas il n'y a pas de retenue aux centaines et $C = 0$. Il se produit exactement le même phénomène pour le Z 80 qui ne connaît que le binaire :

00100011	23 H	35
01001001	49 H	73
<u>01101100</u>	<u>6 CH</u>	<u>108</u>

Le résultat tient sur 8 bits (< 256) il n'y a donc pas de retenue et $C = 0$:

	10100011	A3H	163
	10101110	AEH	174
	<u>01010001</u>	<u>151 H</u>	<u>337</u>
C ← 1			

Le résultat tient sur 9 bits (≥ 256) il y a donc une retenue au rang 2^8 et $C = 1$.

Soustraction :

Le cas de la soustraction est plus complexe et fait appel à la notion de complément à 2. La soustraction est traitée par le microprocesseur comme une addition entre le premier nombre et le complément à 2 du nombre à soustraire. Dans ce cas $C = 0$ signifie que le résultat est positif. Le cas $C = 1$ signifie que le résultat négatif exprimé par son complément à 2. Étant donné que l'on additionne le complément à 2 la retenue C enregistre le complément de la retenue réelle de l'addition.

Effectuons l'opération $30 - 25$. Le microprocesseur effectue en fait : $30 + \text{comp}(25)$ que l'on peut décomposer :

$$(30 - 25) + (25 + \text{comp}(25))$$

Or $25 + \text{comp}(25) = 256 = 100H$:

$$(30 - 25) + (25 + \text{comp}(25)) = 5 + 256 = 5H + 100H + 105H$$

La retenue réelle est 1, mais comme on enregistre le complément de cette retenue $C = 0$, et le résultat de $30 - 25$ est bien positif. Effectuons maintenant $30 - 35$. Le microprocesseur effectue l'addition :

$$30 + \text{comp}(35)$$

$$(30 - 35) + (35 + \text{comp}(35))$$

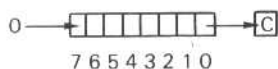
$$-5 + 256 = 251$$

Le résultat inférieur à 256 n'entraîne pas de retenue, mais comme on enregistre le complément de cette retenue $C = 1$. Le résultat trouvé est effectivement le complément à 2 de 5 ($\text{comp}(5) = 256 - 5$), ce qui représente (-5) .

Décalage :

De nombreux types de décalages affectent la retenue C . Ce sont les instructions RLA, RLCA, RRA, RRCA, RL, RLC, RR, RRC, SLA,

SRA, SRL qui seront détaillées plus tard. Citons par exemple le décalage logique à droite SRL. Le bit 0 de l'octet décalé passe dans l'indicateur C :



Toutes ces rotations effectuent des circuits différents passant par l'indicateur C.

Comparaisons :

En fait lorsque le microprocesseur compare deux nombres A et B il effectue de façon fictive $A - B$ sans donner le résultat. Ceci a seulement pour effet de positionner divers flags dont l'indicateur C. La valeur A est toujours contenue dans l'accumulateur. La valeur B est soit le contenu d'un registre, soit le contenu d'une case mémoire définie par l'opérande de l'instruction de comparaison. En reprenant ce qui a été dit sur la soustraction :

$A \geq B$	résultat $A - B \geq 0$	donc $C = 0$
$A < B$	résultat $A - B < 0$	donc $C = 1$

4.4.2. Bit 1 ou N

Le flag N indique si la dernière opération effectuée par le Z 80 est une soustraction. Dans ce cas $N = 1$. Cet indicateur est utilisé avec l'indicateur H et l'opération d'ajustement décimal.

4.4.3. Bit 2 ou P/V

Ce flag a un double sens dépendant de l'opération qui vient d'être effectuée. Il sert d'indicateur de dépassement de capacité (overflow) V dans le cas d'une opération arithmétique, d'incrément et de décrément. Dans le cas d'opérations logiques ou de décalage il sert d'indicateur de parité P.

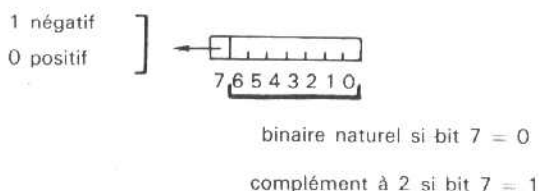
Parité :

La parité d'un octet dépend du nombre de 1 contenus dans l'octet. Si ce nombre est pair la parité est paire et $P = 1$ dans le cas contraire la parité est impaire et $P = 0$.

A =	01100100	3 (1)	impair	$P = 0$
A =	10011010	4 (1)	pair	$P = 1$

Dépassement :

Le bit V n'a d'utilité que lorsque l'on travaille sur un octet signé. Un octet signé permet de représenter les nombres positifs et négatifs. Les nombres positifs sont codés en binaire naturel, les nombres négatifs sont codés par le complément à 2 de leur valeur absolue. (Pour un octet, le complément à 2 de 11 est la valeur $256 - 11 = 245$). Dans ce cas un octet peut coder des nombres variant de -128 à $+127$. Le bit 7 de l'octet représente le signe du nombre. Pour les nombres positifs bit 7 = 0 (ce qui est normal car ils sont ≤ 127), pour les nombres négatifs bit 7 = 1 (-1 à -128 est représenté par $256 - 1 = 255$ à $256 - 128 = 128$, donc bit 7 = 1).



La valeur du nombre est donc codée sur 7 bits. Lors d'opérations arithmétiques si le résultat nécessite plus de 7 bits pour son codage il y a dépassement. Ce dépassement ne peut avoir lieu que pour l'addition de deux nombres de mêmes signes ou la soustraction de deux nombres de signes opposés. D'une manière pratique $V = 1$ lorsque le résultat est incohérent.

Par exemple :

$$\begin{array}{r}
 79 \qquad 01001111 \\
 + 119 \quad 01110111 \\
 \hline
 11000110
 \end{array}$$

le résultat est négatif ??? $\rightarrow V = 1$

L'addition de deux nombres positifs n'a jamais donné un nombre négatif. Mais pour le microprocesseur $119 + 79 = 198 \geq 127$ ne peut pas être codé sur 7 bits.

Une méthode simple pour prédire le dépassement est le suivant. Il y a dépassement lorsque le bit 7 du résultat et le bit de retenue sont complémentaires :

bit 7 = 0 et C = 1

bit 7 = 1 et C = 0

4.4.4. Bit 4 ou H

Ce flag est l'indicateur de retenue au rang 2^4 . C'est-à-dire qu'il fonctionne exactement comme C en se limitant au 4 premiers bits 0, 1, 2, 3.

$$\begin{array}{r}
 \boxed{1} \mid \\
 0101 \mid 0111 \\
 \mid \\
 0010 \mid 1100 \\
 \hline
 1000 \mid 0011
 \end{array}$$

Il y a une retenue au rang 2^4 car :

$$\begin{array}{r}
 0111 \\
 1100 \\
 \hline
 \boxed{H} \leftarrow \boxed{1} \mid 0011
 \end{array}$$

et $H = 1$

4.4.5. Bit 6 ou Z

Cet indicateur signale que le résultat de la dernière opération effectuée est nul. Dans ce cas $Z = 1$.

Pour les opérations arithmétiques, logiques $Z = 1$ si le résultat est nul. Pour les opérations de comparaisons $Z = 1$ si les deux valeurs comparées sont égales. Lors d'un test de bit $Z = 1$ si le bit testé est 0. Ce flag est aussi affecté par certaines instructions d'entrée-sortie en bloc lorsque le décrémentation de B donne un résultat nul.

4.4.6. Bit 7 ou S

Ce flag donne le signe de l'octet contenu dans l'accumulateur. C'est-à-dire qu'il prend la valeur du bit 7 de l'octet contenu dans l'accumulateur.

5

Les modes d'adressage du Z 80

Les modes d'adressage concernent tous les moyens mis à la disposition du microprocesseur pour accéder aux données à partir des opérandes d'une instruction. Le microprocesseur accède toujours à la mémoire centrale en générant une adresse, il n'y a donc rien de changé de ce côté-ci. Par contre les opérandes d'une instruction ne sont pas forcément une adresse réelle mais un nombre, une clé à partir duquel le microprocesseur calcule l'adresse réelle de la case mémoire où se trouve la donnée à traiter. L'exemple suivant montre l'intérêt de posséder plusieurs modes d'adressage.

Soient deux variables A et B, A est un nombre codé sur un octet, B est un tableau de dix nombres codés chacun sur un octet. Ces dix nombres sont rangés contiguement en mémoire centrale. Les instructions traitant la variable A auront de préférence pour opérande l'adresse réelle de A par contre pour la variable B il serait peu pratique de donner explicitement une des dix adresses affectées aux éléments du tableau. Il est beaucoup plus naturel de dire au microprocesseur : "L'opération en cours affecte le 8^e élément du tableau B". Ceci est rendu possible en utilisant un mode d'adressage approprié pour lequel l'opérande de l'instruction n'est pas l'adresse réelle du 8^e élément mais le chiffre 8. Cela nécessite que l'adresse du 1^{er} élément du tableau B soit connue du microprocesseur en la rangeant dans un registre spécial.

Le Z 80 possède huit modes d'adressage que nous allons expliciter. Chacun de ces modes sera illustré par une instruction du Z 80.

Adressage étendu :

Ce mode d'adressage est le plus simple car l'opérande est l'adresse réelle de la case mémoire concernée. Le mot "étendu" provient du fait que ce mode d'adressage permet d'accéder à la totalité de l'espace mémoire. L'opérande est donc un mot de 16 bits (2 octets).

L'instruction :

"charger l'accumulateur avec le contenu de la case mémoire d'adresse 0C745H" s'écrit :

LD A, (0C745H)

On a donc :

adresse effective (AE) = opérande

5.1. ADRESSAGE INDIRECT PAR REGISTRE

Ce mode d'adressage ne nécessite pas d'opérande. L'adresse de la case mémoire est le contenu de l'une des paires de registre du Z 80 HL, DE, BC. L'adresse étant codée sur 16 bits on peut accéder à la totalité de l'espace mémoire. Ce mode d'adressage est très utile pour passer des paramètres aux sous-programmes.

L'instruction :

"charger l'accumulateur avec le contenu de la case mémoire dont l'adresse est dans le registre DE" s'écrit :

LD A, (DE)

On a donc :

AE = contenu de HL ou BC ou DE

5.2. ADRESSAGE INDEXÉ

Ce mode d'adressage s'appelle aussi "relatif à une base", il correspond à l'exemple d'introduction concernant le tableau B. L'opérande est

un octet signé (-128 à $+127$). Le microprocesseur calcule l'adresse réelle en faisant la somme de l'opérande (appelé déplacement) et du contenu de l'un des deux registres d'index IX ou IY. On ne peut accéder qu'à 256 cases mémoires situées de part et d'autre de la case pointée par IX ou IY.

L'instruction :

"charger l'accumulateur avec le contenu de la case mémoire élément 8 du tableau B pointé par le registre IX" s'écrit :

LD A, (IX + 8)

On a donc :

AE = contenu de IX ou IY + déplacement

Ces trois modes d'adressage que nous venons de voir sont les seules façons que possède le Z 80 pour accéder aux données en mémoire centrale.

5.3. ADRESSAGE PAR REGISTRE

Ce mode d'adressage est réservé aux opérations internes au microprocesseur ne nécessitant aucun accès à la mémoire. Les manipulations sur les registres utilisent ce mode d'adressage car un registre est une case mémoire directement accessible donc sans adresse.

L'instruction :

"charger le registre B avec le contenu du registre D" s'écrit :

LD B, D

opérande = nom du registre

5.4. ADRESSAGE IMPLICITE

Certaines instructions du Z 80 ne peuvent s'effectuer que sur l'accumulateur. Dans ce cas il n'est pas besoin de préciser explicitement par un opérande que l'accumulateur est mis en jeu.

L'instruction :

"rotation à gauche de l'accumulateur à travers la retenue" s'écrit :

RLCA

pas d'opérande

5.5. ADRESSAGE IMMÉDIAT

Il ne s'agit pas d'un véritable mode d'adressage mais par abus de langage on le traite comme tel. En effet l'opérande n'est ni une adresse, ni un déplacement, ni un nom de registre mais la donnée numérique à traiter par l'instruction. Dans un véritable mode d'adressage l'opérande indique où se trouve la donnée.

L'instruction :

"charger l'accumulateur avec la valeur 133" s'écrit :

LD A, 133D

opérande = donnée (8 bits)

5.6. ADRESSAGE IMMÉDIAT ÉTENDU

Ce mode d'adressage est identique au précédent sauf que la donnée est codée sur deux octets. Il est utilisé pour charger les registres 16 bits HL, BC, DE, SP, IX, IY.

L'instruction :

"initialiser le pointeur de pile à l'adresse 8000H" s'écrit :

LD SP, 8000H

opérande = donnée (16 bits)

Il y a un autre registre 16 bits que nous n'avons pas nommé, c'est le compteur ordinal PC. En effet, un branchement n'est rien d'autre que le chargement immédiat de PC avec une donnée de 16 bits. Plutôt que d'écrire :

LD PC, XXXXH

on utilise l'écriture plus parlante :

JP XXXXH

JP comme JumP ou saut à l'adresse XXXXH.

5.7. ADRESSAGE RELATIF PC

Puisque nous sommes dans les branchements, restons-y. L'opérande est un déplacement (octet signé). Ce mode d'adressage n'est utilisé que pour le compteur ordinal et permet donc d'effectuer des branchements en modifiant la valeur du PC. Le microprocesseur effectue l'opération $PC = PC + \text{déplacement}$.

Le branchement relatif PC est intéressant car il est indépendant de l'endroit où se situe le programme en mémoire centrale.

L'instruction :

"revenir 30 octets en arrière" s'écrit :

JR - 30D

opérande = déplacement

5.8. ADRESSAGE PAGE ZÉRO MODIFIÉ

Ce mode d'adressage est utilisé par le Z 80 pour un appel de sous-programme spécial appelé RESTART. Cette instruction force la valeur de PC une adresse en page zéro (8 bits de poids forts nuls). Le Z 80 possède 8 restarts différents se branchant sur des sous-programmes débutant aux adresses 0, 8H, 10H, 18H, 20H, 28H, 30H, 38H. Ainsi les instructions :

CALL 10H (appel de sous-programme)

et RST 10H (RESTART)

sont strictement équivalentes à ceci près que le CALL et son opérande sont codés sur 3 octets alors que le RST n'en nécessite qu'un seul.

Le RESTART est utilisé pour des sous-programmes appelés très fréquemment à cause du gain en encombrement qu'il procure.

Il faut noter que l'emploi de l'instruction RST est souvent réservée au système d'exploitation du micro-ordinateur et correspond donc à des sous-programmes particuliers en ROM. Son usage est donc peu aisé et déconseillé pour débiter.

6

L'assembleur du Z 80

Lors du chapitre d'introduction nous avons montré l'utilité de posséder un intermédiaire entre le programmeur en langage machine et le microprocesseur qui ne comprend que des 1 ou des 0. Cet intermédiaire est l'assembleur. Un assembleur est un programme qui effectue une traduction d'un texte appelé source en une succession d'octets appelée code objet directement compréhensible par le microprocesseur, ceci dans le cas des petits ordinateurs individuels. Nous verrons plus tard qu'il peut entrer d'autres intermédiaires dans cette traduction. En réalité l'assembleur est plus qu'un traducteur qui se contenterait de transformer la ligne :

LD A, B

en un octet 78H. L'assembleur offre en plus des possibilités permettant de simplifier grandement l'écriture des programmes. Cette possibilité se résume dans le fait que l'on peut assigner une valeur numérique à un nom. De là découle la possibilité de nommer une adresse de variable ou d'instruction, une constante numérique. Il est plus agréable de relire la ligne :

CALL CLAVIER

plutôt que :

CALL 3BF7H

Dans le premier cas on se rappelle immédiatement qu'il s'agit d'un appel du sous-programme d'acquisition d'une touche d'un clavier alphanumérique. Le second cas nous laisse plus perplexe. Mais reprenons les choses au début et découvrons la "grammaire" fort simple de l'assembleur.

Deux règles majeures :

— *Les règles absolues* : Ce sont celles imposées par le programme assembleur qui ne comprend que des lignes formatées et de syntaxe correcte. Le même phénomène se retrouve en BASIC. Si ces règles sont transgressées l'assembleur génère un code d'erreur et indique la ligne incriminée exactement comme le fait votre interpréteur BASIC.

— *Les règles conseillées* : Ce sont les règles que doit s'imposer le programmeur pour une programmation rapide, minimisant le plus le temps de mise au point des programmes qui fonctionnent rarement du premier coup. Par exemple on utilisera de préférence des noms de variable explicites du style : COMPTEUR, BOUCLE, INDICE, etc... plutôt que X, Y, Z... Les programmes seront aérés, les sous-programmes séparés et bien définis. Ces règles sont très importantes car l'utilisation de mnémoniques ne clarifie pas les programmes.

6.1. LA SYNTAXE DE L'ASSEMBLEUR

Un programme assembleur se présente sous la forme d'une succession de lignes. Chaque ligne représente une instruction exécutable par le microprocesseur ou une directive dont nous parlerons plus tard. La ligne assembleur a une structure bien définie. Elle est composée de plusieurs zones appelées "champs", séparées par un délimiteur qui est le plus souvent un espace. Chacun de ces champs a un rôle bien défini, ils sont au nombre de 4 :

ligne assembleur :

<u>champ 1</u>	↑	<u>champ 2</u>		<u>champ 3</u>		<u>champ 4</u>
		espace				

6.1.1. Champ 1 ou zone étiquette

Une étiquette est un nom particulier attribué à la ligne. Ce nom ne peut se trouver qu'une seule fois dans une zone étiquette sinon l'assembleur génère un code d'erreur (Étiquette attribuée deux fois). La présence d'étiquette n'est pas obligatoire. Dans ce cas le champ 1 est remplacé par une tabulation (CTL I) ou TAB.

La présence d'étiquette est très utile pour les instructions de branchement ou d'appels de sous-programmes. Il est plus simple d'écrire :

JP SUITE

qui est un branchement à la ligne dont l'étiquette est SUITE plutôt que :

JP 034AFH

Cette deuxième écriture sous-entendrait que le programmeur sait déjà que la ligne SUITE définit une instruction qui sera rangée à l'adresse 034AFH, ce qui n'est jamais le cas.

Ici aussi il faut utiliser des étiquettes parlantes. Profitez de cette possibilité qui est un avantage sur le BASIC qui ne connaît que les étiquettes numériques (GOSUB 380). Évitez surtout les étiquettes A, B, C... qui sont les noms réservés aux différents registres du Z 80.

Le champ étiquette est souvent limité à un nombre maximum de caractères. Six caractères pour l'assembleur Z 80 MOSTEK que l'on rencontre le plus souvent.

6.1.2. Champ 2 ou zone opération

Ce champ est réservé à l'écriture du mnémonique d'une instruction ou d'une directive. Les mnémoniques du Z 80 sont tous définis plus loin. Un mnémonique est un assemblage de 2, 3, 4 lettres définissant une instruction exécutable par le microprocesseur. Cet assemblage est une contraction d'un mot ou d'une phrase écrit en Anglais :

LoaD LD chargement

Decrement and Jump relative if Not Zero DJNZ

Les mnémoniques des directives dépendent de l'assembleur utilisé contrairement à ceux des instructions qui sont fixés par le constructeur du Z 80. Nous décrivons les mnémoniques de directives les plus souvent rencontrées.

6.1.3. Champ 3 ou zone opérande

Ce champ est le plus compliqué. Il comprend l'opérande de l'instruction écrite dans le champ 2. Cet opérande peut être simple ou double. Dans le second cas on parle d'opérande source et d'opérande destination. On distingue parmi les opérandes :

- des nombres,
- des noms de variables,
- des noms d'étiquettes,
- des noms de registres,
- des expressions arithmétiques ou logiques.

Un opérande écrit entre parenthèses indique qu'il représente une adresse ainsi :

LD A, 037 H

signifie: charger l'accumulateur avec la *valeur 037H*.

LD A, (037H)

signifie: charger l'accumulateur avec le *contenu de la case mémoire d'adresse 037H*.

(expression) signifie toujours: contenu de la case mémoire d'adresse "expression".

Le cas de l'instruction de chargement est un cas d'opérande double. Les deux opérandes source et destination sont séparés par une virgule (,). Le premier opérande est la destination, le second opérande est la source. Pour les instructions nécessitant un opérande double, on a toujours :

mnémonique
(champ 2)

destination, source
(champ 3)

ainsi les instructions :

LD A, (03FCH)



LD (03FCH), A



ne sont pas équivalentes. On peut définir source et destination par :

destination : endroit où le résultat de l'opération exécutée est rangé.

source : endroit où se trouve la donnée à traiter par l'instruction.

Lorsque l'opérande est un registre on désigne celui-ci par son nom :

LD B, D



LD HL, OFCDEH



La façon dont on écrit l'opérande est liée au mode d'adressage employé, on distingue ainsi :

adressage étendu :

opérande = (expression numérique)

LD A, (03FCH)

adressage indirect par registre :

opérande = (paire de registre)

LD A, (BC)

adressage indexé :

opérande = (IX ou IY + expression numérique)

LD A, (IX + 10)

adressage par registre :

opérande = nom de registre

LD A, C

adressage immédiat :

opérande = expression numérique

LD A, 30

LD HL, 0D034H

adressage relatif PC :

opérande = étiquette de branchement

JR BOUCLE

dans ce cas l'assembleur calcule tout seul la valeur du déplacement.

Expressions arithmétiques et logiques

La partie numérique d'un opérande peut être une expression contenant des additions, des soustractions, le ET logique ou le décalage logique. Les expressions suivantes sont possibles :

LD A, 30 + 40

LD A, 25 - 6

LD A, - 10H

LD A, 11H & 01H

Le décalage logique s'exprime par la combinaison :

valeur < nombre de décalage

3COOH < + 2 la valeur 3COOH est décalée 2 fois à gauche → EDOOH

3COOH < - 2 la valeur 3COOH est décalée 2 fois à droite → OEDOH

On peut avoir par exemple :

LD HL, 3COOH < + 2

Représentation des nombres

L'assembleur accepte plusieurs bases de numération qui sont généralement :

décimal	base 10
hexadécimal	base 16
octal	base 8

Un nombre exprimé dans l'une de ces bases est succédé d'une lettre caractéristique de la base :

décimal	lettre D ou rien	34D ou 34
hexadécimal	lettre H	0A3F4H
octal	lettre O	43O

Il n'est pas besoin de décrire le décimal. Lorsque l'assembleur rencontre un nombre tel que 12 ou 12D il l'interprète comme une constante décimale. L'assembleur n'accepte que des nombres entiers.

L'hexadécimal (base 16) nécessite 16 symboles pour son écriture : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Une constante numérique doit nécessairement commencer par un chiffre (0 à 9). L'écriture F2H est fautive, par contre 0F2H qui représente la même valeur est correcte.

L'octal (base 8) nécessite 8 symboles pour son écriture : 0, 1, 2, 3, 4, 5, 6, 7. Une constante octale s'écrit par exemple 67O.

Nous rappelons la formule de transformation d'une base quelconque en décimal :

$$\begin{aligned} 3C2FH &= F \times 1 + 2 \times 16 + C \times (16)^2 + 3 \times (16)^3 \\ &= 15 \times 1 + 2 \times 16 + 12 \times (16)^2 + 3 \times (16)^3 = 15407 \\ 432O &= 2 \times 1 + 3 \times 8 + 4 \times (8)^2 = 282 \end{aligned}$$

En pratique on emploie souvent le décimal et l'hexadécimal. L'octal est utilisé pour exprimer un code opération. Il permet de distinguer trois champs dans un code opération. Le décimal s'emploie souvent pour

initialiser des compteurs, pour un déplacement lors de recherche en table. L'hexadécimal est pratique pour exprimer des adresses de variables ou de sous-programmes externes en mémoire centrale.

6.1.4. Champ 4 ou zone commentaire

Ce champ est séparé du reste de la ligne par un espace et un ;. Après le ; on peut écrire ce que l'on veut. Ceci permet de documenter les programmes en les rendant de ce fait plus clairs. Un bon commentaire ne doit pas remettre en clair le mnémonique employé dans la ligne. Au contraire il doit apporter une information supplémentaire sur le rôle de cette ligne dans le programme.

LD B, 10	;	charger B avec la valeur 10	est mauvais
LD B, 10	;	faire 10 fois la boucle	est bon

si par exemple le registre B est utilisé en compteur de boucle.

Certaines lignes ne peuvent comporter qu'un commentaire. Dans ce cas elle commence par ;. Ceci permet d'aérer les listings de programmes, de mettre des titres, etc;..

```
; COMMENTAIRE  
BOUCLE LD B, 10 ; faire 10 fois la boucle.
```

6.2. FONCTIONNEMENT D'UN ASSEMBLEUR

Comme nous l'avons dit l'assembleur est un programme qui effectue un traitement sur un texte écrit en assembleur appelé source et qui génère un code objet exécutable par le microprocesseur.

La ligne assembleur est composée de plusieurs champs. La zone étiquette comporte des mots inconnus à priori de l'assembleur. La zone opération par contre utilise un mnémonique que l'assembleur connaît puisqu'il correspond à un code opération du Z 80 ou à une directive. La zone opérande contient un certain nombre de symboles connus de l'assembleur qui sont les parenthèses, les noms de registres, les lettres D, H,

O, les symboles arithmétiques et logiques (+, —, &, <) et des symboles inconnus à priori qui sont les étiquettes utilisées par le programmeur. La zone commentaire est ignorée de l'assembleur.

Un label n'est rien d'autre que l'assignation d'un mot à une valeur numérique, par conséquent l'assembleur doit dans un premier temps faire un récapitulatif de ces assignations. Il dresse ce que l'on appelle la table des symboles. Il constitue son propre dictionnaire lui permettant par la suite de déchiffrer complètement le programme. Cela nécessite une première lecture complète du texte source pour ne laisser passer aucune étiquette. Cette opération s'appelle la première passe. Au cours de cette première passe l'assembleur vérifie aussi la syntaxe des lignes et affiche les erreurs qu'il trouve. Par exemple dans le champ opération l'assembleur doit reconnaître le mnémonique sinon il affiche un code d'erreur (opération inconnue) et la ligne incriminée. Lorsque l'assembleur dresse la table des étiquettes il ne doit rencontrer qu'une seule assignation par étiquette.

Mais quelle valeur numérique correspond à une étiquette donnée ? Il faut rentrer un peu dans la "cuisine" de l'assembleur, ceci est nécessaire pour comprendre plus tard le rôle de certaines directives.

Lorsque l'assembleur rencontre une étiquette dans le champ 1 il lui assigne la valeur de l'adresse qu'occupera le code opération de l'instruction définie par le mnémonique de la ligne lorsque le programme sera assemblé et exécutable par le microprocesseur. L'assembleur doit donc se débrouiller pour connaître d'avance cette adresse. Pour cela il utilise un pointeur d'adresse initialisé à une valeur fixée par le programmeur en début de programme (directive ORG). A chaque fois que l'assembleur rencontre une instruction Z 80 il incrémente le pointeur d'adresse d'un nombre égal au nombre d'octets nécessaire pour coder l'instruction et son opérande (1, 2, 3 ou 4). Ce pointeur d'adresse traduit la valeur du compteur ordinal lorsque le programme sera exécuté. Ce compteur d'adresse contient l'adresse du code opération de l'instruction suivante. Lorsque l'assembleur rencontre une étiquette il lui assigne donc la valeur du pointeur d'adresse.

Lorsque cette première passe est achevée l'assembleur exécute une seconde passe au cours de laquelle il effectue réellement l'assemblage. Supposons que l'assembleur rencontre la ligne :

ETIQU LD A, (VARIAB)

Il reconnaît LD : instruction de chargement.

Il reconnaît A : charger l'accumulateur.

Il reconnaît () et VARIAB n'est pas un registre : adressage étendu.

L'assembleur cherche VARIAB dans la table des symboles et trouve la valeur 3C45 qui lui est assignée, d'autre part il connaît le code opération du "chargement de l'accumulateur avec adressage étendu" qui est 3A. La ligne assemblée donne donc :

3A 45 3C

On remarque que l'octet faible de l'adresse vient en premier. Ceci est imposé par le microprocesseur.

Au cours de cette seconde passe certaines erreurs sont détectées. Par exemple : le calcul d'un déplacement (saut relatif PC) > 127 ou < -128 ou étiquette non définie.

6.3. DIRECTIVES DES ASSEMBLEURS

Une directive (ou pseudo op.) est une instruction spéciale qui n'est utilisée que par l'assembleur et qui ne donne pas lieu à la génération d'un code opération du Z 80. Nous allons décrire les directives les plus courantes des assembleurs Z 80.

ORG nn:

Cette directive doit se situer en début du programme et définit l'adresse de la première instruction rencontrée en initialisant le pointeur d'adresse :

	ORG	3C00H	
DEBUT	LD	SP, 3000H	; première instruction
	⋮		

Le code opération de cette première instruction est rangé à l'adresse 3C00H. Après assemblage on doit trouver en mémoire centrale :

3BFF	XX	chargement immédiat de SP
3C00	31	
3C01	00	
3C02	30	

END nn

Cette directive doit obligatoirement être à la fin du programme. Elle indique à l'assembleur la fin du texte source. Si l'on écrit :

```
END DEBUT
```

L'étiquette DEBUT sera l'adresse où débutera le programme lorsque celui-ci sera chargé. Cette adresse n'est pas toujours celle de la première instruction du programme.

étiq. EQU nn

Cette directive assigne explicitement une valeur numérique nn à l'étiquette. Ceci permet de donner un nom aux variables ou aux sous-programmes externes en assignant leur adresse à leur nom :

```
VARIAB    EQU    4000H    ; variable
SOUSPR    EQU    27BCH    ; sous-programme en ROM
```

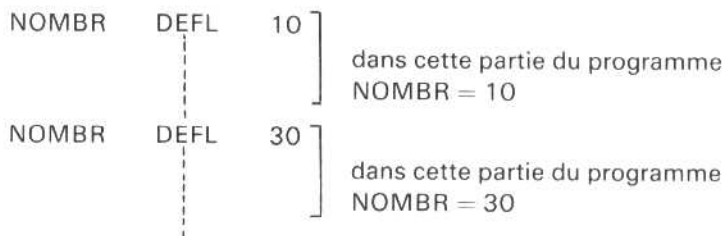
On pourra ensuite trouver les instructions :

```
LD    A, (VARIAB)
CALL  SOUSPR
```


Une étiquette définie par EQU ne peut être redéfinie plus tard. Cette directive sert à définir des références externes au programme ce qui explique que la valeur assignée à l'étiquette n'est pas celle du pointeur d'adresse comme pour une ligne instruction. Dans le cas présenté le sous-programme SOUPR qui se situe dans la ROM ne peut pas être défini par une ligne du programme avec le pointeur d'adresse, on utilise donc la directive EQU.

étiq. DEFL nn

Cette directive a la même caractéristique que EQU mais cette fois-ci l'étiquette peut être redéfinie :



6.3.1. Directives de réservation mémoire

Les directives qui suivent, à la différence des précédentes, ont une influence sur l'espace mémoire où sera rangé le programme. Ces directives modifient le pointeur d'adresse.

étiq DEFB n

Cette directive a pour effet d'initialiser le contenu de la case mémoire dont l'adresse est la valeur du pointeur d'adresse. Le pointeur d'adresse est incrémenté de 1. Pour montrer la différence avec EQU nous allons analyser ces deux exemples :

1)

```

      ORG      5000H
LABEL  DEFB    10H
      LD      A, (LABEL)
      .
      .
      .
  
```

L'assembleur donnera :

```

4FFF
5000
5001
5002
5003
  
```

XX
10
3A
00
50

LD A, (LABEL)
adresse de la variable

Une case mémoire à l'adresse 5000H est réservée pour LABEL et initialisée avec 10H.

2)

```

      ORG      5000H
LABEL  EQU     5000H
      LD      A, (LABEL)
      .
      .
      .
  
```

L'assemblage donnera :

```

4FFF
5000
5001
5002
  
```

XX
3A
00
50

Cette fois-ci il n'y a pas eu de réservation en mémoire centrale.

étiqu. *DEFB "S"*

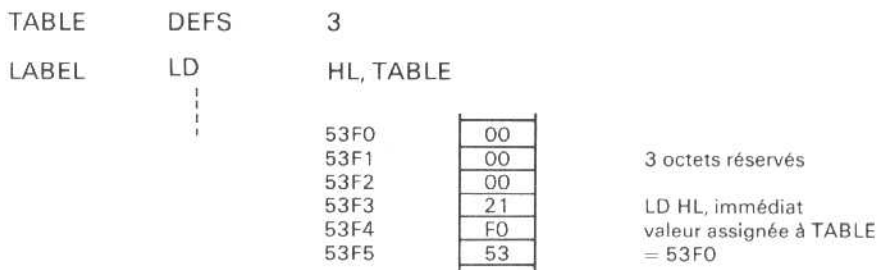
Cette directive identique à la précédente initialise la case mémoire avec le code ASCII du caractère entre " ".



étiqu. *DEFS nn*

Cette directive réserve *nn* cases mémoire à partir de la valeur du pointeur d'adresse. Ceci permet de réserver une zone mémoire pour des tableaux par exemple :

Supposons que le pointeur d'adresse $PA = 53F0$:



étiq *DEFM "ssss"*

Cette directive permet d'initialiser des chaînes de caractères ASCII débutant à l'adresse du PA :

CHAINE	DEFM	"TEXTE"	53F0	54	T
			53F1	45	E
	LD	HL, CHAINE	53F2	58	X
			53F3	54	T
			53F4	43	E
			53F5	21	LD HL, nn
			53F6	F0	}
			53F7	53	
					nn

étiq. *DEFW nn*

Cette directive réserve deux octets pour y écrire le mot nn. L'octet faible est écrit en premier :

MOT	DEFW	3C4F	53F0	4F	}	M01
			53F1	3C		
	LD	HL, (MOT)	53F2	2A	}	LD HL, (nn)
			53F3	F0		
			53F4	53	}	nn

Il existe d'autre directives qui influent sur la présentation du listing de l'assemblage :

TITLE	TITRE	donne un nom au listing
PAGE		saut de page
NOLIST		interrompt le listing (mais pas l'assemblage)
LIST		reprend le listing

7

Le jeu d'instruction du Z 80

7.1. INSTRUCTION DE CHARGEMENT 8 bits

L'ensemble de ces instructions a pour seul effet de recopier la donnée numérique définie par l'opérande source dans l'emplacement défini par l'opérande destination sans en affecter la valeur. Ces instructions n'effectuant aucune opération arithmétique, logique ou décalage sur la donnée manipulée, les indicateurs du registre F (flags) ne sont pas affectés.

L'écriture de ces instructions est :

LD destination, source

Le schéma de l'instruction est le suivant :



On peut diviser cet ensemble d'instructions en trois sous-ensembles suivant la nature de la source et de la destination :

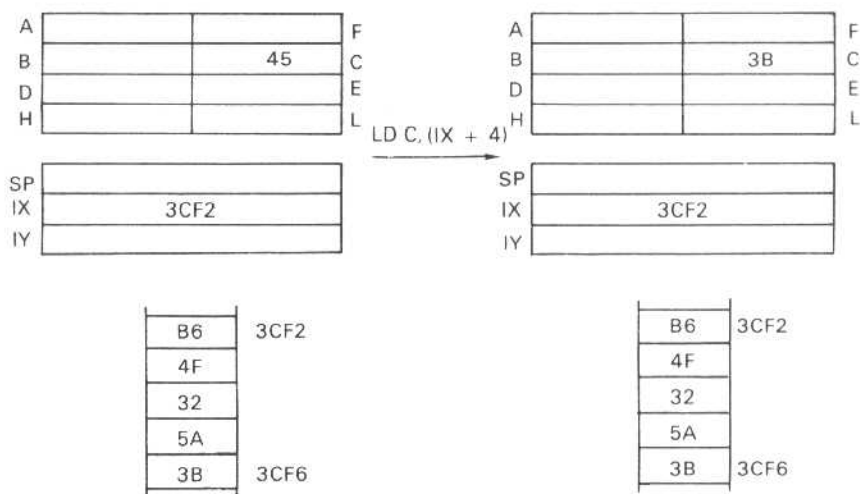
chargement registre	: mémoire	→	registre
stockage registre	: registre	→	mémoire
transfert	: registre	→	registre

La source et la destination sont définies par un des modes d'adressage définis précédemment. Nous allons donner maintenant la liste de toutes les instructions de chargement illustrée par des exemples.

Chargement registre :

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
LD A, (nn)	étendu	aucun
LD A, (BC)	indirect par registre	"
LD A, (DE)	"	"
LD A, (HL)	"	"
LD B, (HL)	"	"
LD C, (HL)	"	"
LD D, (HL)	"	"
LD E, (HL)	"	"
LD H, (HL)	"	"
LD L, (HL)	"	"
LD A, (IX + d)	indexé par IX	"
LD B, (IX + d)	"	"
LD C, (IX + d)	"	"
LD D, (IX + d)	"	"
LD E, (IX + d)	"	"
LD H, (IX + d)	"	"
LD L, (IX + d)	"	"
LD A, (IY + d)	indexé par IY	"
LD B, (IY + d)	"	"
LD C, (IY + d)	"	"
LD D, (IY + d)	"	"
LD E, (IY + d)	"	"
LD H, (IY + d)	"	"
LD L, (IY + d)	"	"

L'utilisation de ces instructions ne pose aucune difficulté. Nous allons rappeler par un exemple le fonctionnement d'un adressage indexé :



Le déplacement est 4 donc l'adresse du mot à charger dans C est :

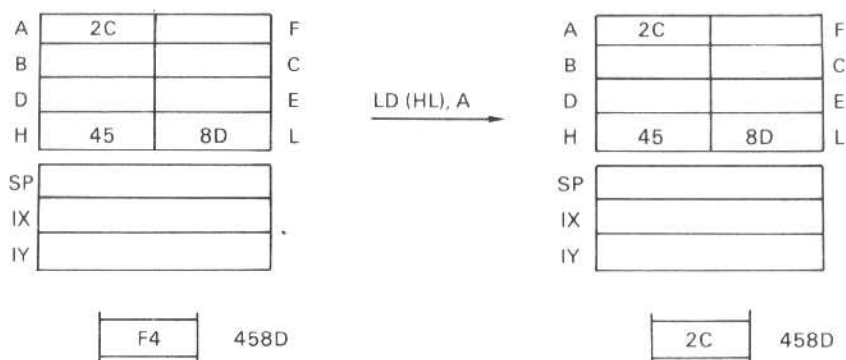
$$3CF2 + 4 = 3CF6$$

Le contenu de cette case mémoire est 3B que l'on recopie dans le registre C.

Stockage registre :

<i>Instruction</i>	<i>Mode d'adressage de la destination</i>	<i>Flags affectés</i>
LD (nn), A	étendu	aucun
LD (BC), A	indirect par registre	"
LD (DE), A	"	"
LD (HL), A	"	"
LD (HL), B	"	"
LD (HL), C	"	"
LD (HL), D	"	"
LD (HL), E	"	"
LD (HL), H	"	"
LD (HL), L	"	"
LD (IX + d), A	indexé par IX	"
LD (IX + d), B	"	"
LD (IX + D), C	"	"
LD (IX + d), D	"	"
LD (IX + d), E	"	"
LD (IX + d), H	"	"
LD (IX + d), L	"	"
LD (IY + d), A	indexé par IY	"
LD (IY + d), B	"	"
LD (IY + d), C	"	"
LD (IY + d), D	"	"
LD (IY + d), E	"	"
LD (IY + d), H	"	"
LD (IY + d), L	"	"

Cet ensemble d'instructions est identique au chargement registre, seuls sources et destinations sont inversés. Nous rappelons dans cet exemple le fonctionnement et l'adressage indirect par registre :



L'adresse de stockage du registre A est le contenu de la paire de registre HL :

458D

Le contenu de A est recopié à cette adresse.

Transfert :

Pour cet ensemble d'instructions la source et la destination sont un nom de registre (adressage par registre).

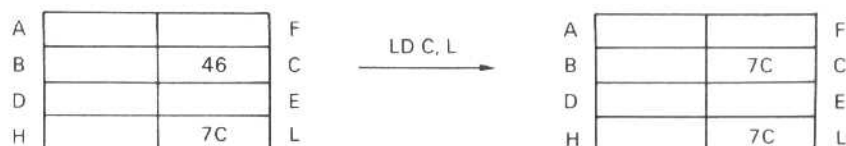
LD R1, R2

aucun flag affecté

R1 = A, B, C, D, E, H, L

R2 = A, B, C, D, E, H, L

Par exemple :



Le contenu du registre L est recopié dans le registre C.

Il existe quatre autres instructions de transfert qui modifient cette fois-ci les flags. Nous les citons pour mémoire car en pratique vous ne les utiliserez jamais :

LD A, R Laissez ces deux instructions au fond du tiroir !!
 LD R, A
 LD A, I
 LD I, A

7.2. INSTRUCTIONS DE CHARGEMENT 16 BITS

Le Z 80 possède la possibilité de coupler deux registres de 8 bits pour former un seul registre 16 bits, il possède aussi des registres de 16 bits (IX, IY, SP). Il existe des instructions de chargement sur ces registres de 16 bits. La mémoire étant organisée en mots de 8 bits, un tel chargement nécessite deux octets de mémoire. Un mot de 16 bits rangé en mémoire contient l'octet de poids faible à l'adresse du mot et l'octet de poids fort à l'adresse suivante. Par exemple :

47CBH rangé à l'adresse 3FF0

CB	3FF0
47	3FF1

Ceci explique pourquoi la directive DEFW range l'octet de poids faible en premier.

L'ensemble de ces instructions a une structure identique à celui des chargements 8 bits. On distingue

chargement registre de 16 bits: mémoire	→	registre
stockage registre de 16 bits: registre	→	mémoire
transfert : registre	→	registre

Chargement registre 16 bits

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
LD HL, (nn)	étendu	aucun
LD BC, (nn)	"	"
LD DE, (nn)	"	"
LD IX, (nn)	"	"
LD IY, (nn)	"	"
LD SP, (nn)	"	"

Stockage registre 16 bits

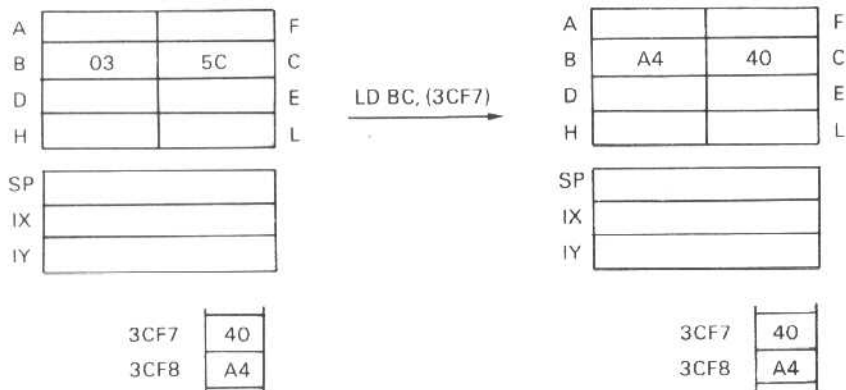
<i>Instruction</i>	<i>Mode d'adressage de la destination</i>	<i>Flags affectés</i>
LD (nn), HL	étendu	aucun
LD (nn), BC	"	"
LD (nn), DE	"	"
LD (nn), IX	"	"
LD (nn), IY	"	"
LD (nn), SP	"	"

Transfert :

<i>Instruction</i>	<i>Mode d'adressage de la destination</i>	<i>Flags affectés</i>
LD SP, HL	par registre	aucun
LD SP, IX	"	"
LD SP, IY	"	"

Seul le registre SP (pointeur de pile) peut être mis en destination.

Exemple de chargement de BC :



L'octet rangé à l'adresse 3CF7 est recopié dans C (poids faible), l'octet rangé à l'adresse suivante est rangé dans B (poids fort).

7.3. INSTRUCTIONS DE CHARGEMENT IMMÉDIAT

Il serait plus juste d'appeler cet ensemble d'instructions "chargement d'une constante". La donnée chargée dans un registre ou en mémoire n'est pas une variable mais une constante imposée par le programme et qui fait donc partie de celui-ci. Cette constante ne peut donc qu'être lue, c'est-à-dire recopiée dans un registre ou dans une case mémoire (voir adressage immédiat). Le schéma de ces instructions est le suivant :

constante → destination

Son écriture est :

LD destination, constante

l'écriture de "constante" correspond au mode d'adressage dit immédiat.

La destination peut être un registre de 8 bits, un octet en mémoire centrale ou un registre 16 bits.

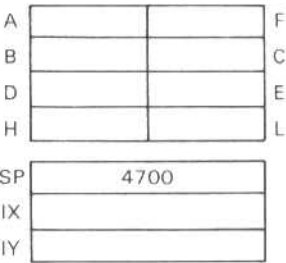
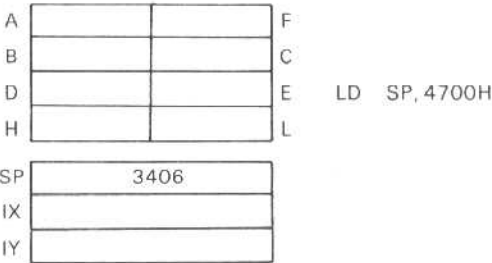
Chargement immédiat 8 bits :

Instruction	Mode d'adressage de la destination	Flags affectés
LD A, n	par registre	aucun
LD B, n	"	"
LD C, n	"	"
LD D, n	"	"
LD E, n	"	"
LD H, n	"	"
LD L, n	"	"
LD (HL), n	indirect par registre	"
LD (IX + d), n	indexé par IX	"
LD (IY + d), n	indexé par IY	"

Chargement immédiat 16 bits

Instruction	Mode d'adressage de la destination	Flags affectés
LD BC, nn	par registre	aucun
LD DE, nn	"	"
LD HL, nn	"	"
LD SP, nn	"	"
LD IX, nn	"	"
LD IY, nn	"	"

Par exemple l'initialisation de SP utilise un chargement immédiat :



Nous en avons enfin terminé avec le LD. Il faut retenir que le LD ne fait que recopier une donnée sans la modifier. C'est l'instruction de base, pour faire un parallèle avec le BASIC le LD serait l'équivalent de LET — = —

LD destination, source

LET destination = source

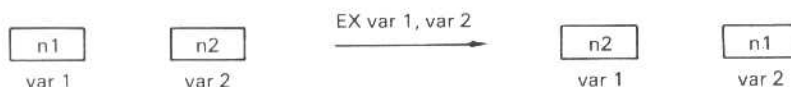
Mettez-vous maintenant devant votre clavier et testez ces différents LD, cela vous familiarisera avec les modes d'adressage en regardant quel chemin la donnée suit.

7.4. INSTRUCTION D'ÉCHANGE

Ces instructions effectuent un déplacement de données, mais contrairement au LD, le déplacement est bidirectionnel, on ne peut donc plus parler de source ou de destination, mais de deux variables qui échangent leurs contenus. L'écriture de ces instructions est :

EX variable 1, variable 2

Le schéma est le suivant :



Les instructions d'échange sont :

<i>Instruction</i>	<i>Mode d'adressage de var 1</i>	<i>Flags affectés</i>
EX DE, HL	par registre	aucun
EX AF, AF'	"	F échangé avec F'
EX (SP), HL	indirect par registre	aucun
EX (SP), IX	"	"
EX (SP), IY	"	"

Rappelez-vous des deux jeux de registre du Z 80, nous avons enfin une instruction pour échanger A et F avec A' et F'.

L'autre instruction d'échange entre les deux jeux de registres d'usage général est :

EXX

qui échange :

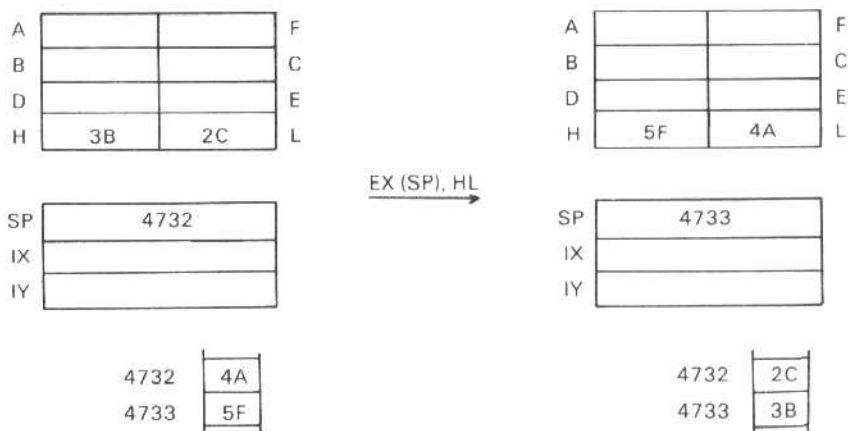
BC avec B'C'

DE avec D'E'

HL avec H'L'

Pour reprendre notre image, faire tourner le tableau de 180° s'écrit EXX.

Revenons le temps d'un exemple sur EX (SP), HL



Un mot de 16 bits est toujours rangé avec l'octet faible en premier.

7.5. INSTRUCTIONS ARITHMÉTIQUE 8 BITS

L'arithmétique du Z 80 est pauvre puisqu'il ne connaît que l'addition et la soustraction. Le Z 80 effectue ces opérations sur des mots de 8 bits. L'indicateur P/V indique ici un débordement et non la parité.

Toutes ces opérations utilisent l'accumulateur, c'est-à-dire que le schéma d'une opération arithmétique est :

$$A = A \Delta \text{ source}$$

Δ représente une opération réalisable par le Z 80

"source" est défini par l'opérande.

L'écriture d'une opération arithmétique est :

OPR A, source

OPR est un mnémonique fictif désignant l'addition ou la soustraction. La source est définie par l'un des modes d'adressage disponible exactement comme pour un chargement.

Addition 8 bits : $A = A + s$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
ADD A, γ^*	par registre	S, Z, H, V, C, N = 0
ADD A, n	immédiat	"
ADD A, (HL)	indirect par registre	"
AED A, (IX + d)	indexé par IX	"
ADD A, (IY + d)	indexé par IY	"

* la lettre γ représente un des registres A, B, C, D, E, H, L

Addition 8 bits avec retenue : $A = A + s + CY$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
ADC A, γ	par registre	S, Z, H, V, C, N = 0
ADC A, n	immédiat	"
ADC A, (HL)	indirect par registre	"
ADC A, (IX + d)	indexé par IX	"
ADC A, (IY + d)	indexé par IY	"

Le bit contenu dans le flag CY est additionné, ceci permet de propager la retenue lors d'additions successives de plusieurs octets.

Soustraction 8 bits : $A = A - s$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
SUB A, γ	par registre	S, Z, H, V, C, N = 1
SUB A, n	immédiat	"
SUB A, (HL)	indirect par registre	"
SUB A, (IX + d)	indexé par IX	"
SUB A, (IY + d)	indexé par IY	"

Soustraction 8 bits avec retenu : $A = A - s - CY$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
SBC A, γ	par registre	S, Z, H, V, C, N = 1
SBC A, n	immédiat	"
SBC A, (HL)	indirect par registre	"
SBC A, (IX + d)	indexé par IX	"
SBC A, (IY + d)	indexé par IY	"

Incrément 8 bits : $s = s + 1$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
INC γ	par registre	S, Z, H, V, N = 0
INC (HL)	indirect par registre	"
INC (IX + d)	indexé par IX	"
INC (IY + d)	indexé par IY	"

Cette série d'instructions n'utilise pas l'accumulateur ce qui explique la syntaxe un peu différente, toutefois cette instruction peut être considérée comme le condensé de trois instructions utilisant l'accumulateur :

```
INC (HL) ⇔ LD A, (HL)
           ADD A, 1
           LD (HL), A
```

à la seule différence que l'instruction *INC* ne modifie pas le contenu de l'accumulateur (mis à part *INC A*).

Décrement 8 bits : $s = s - 1$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
DEC γ	par registre	S, Z, H, V, N = 1
DEC (HL)	indirect par registre	"
DEC (IX + d)	indexé par IX	"
DEC (IY + d)	indexé par IY	"

Comparaison : $A - s$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
CP γ	par registre	S, Z, H, V, N = 1, C
CP n	immédiat	"
CP (HL)	indirect par registre	"
CP (IX + d)	indexé par IX	"
CP (IY + d)	indexé par IY	"

La comparaison est une soustraction dont on ne conserve pas le résultat. Seuls les flags restent positionnés.

CY	= 0	$A \geq$	opérande
CY	= 1	$A <$	opérande
Z	= 0	$A \neq$	opérande
Z	= 1	$A =$	opérande

Opposé : $A = -A$

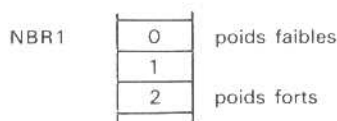
<i>Instruction</i>	<i>Mode d'adressage</i>	<i>Flags affectés</i>
NEG	implicite	S, Z, H, V, N = 1, C

Cette instruction calcule l'opposé d'un octet avec la convention citée en début de livre pour représenter les nombres négatifs.

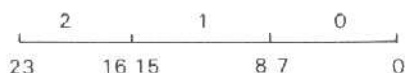
Addition de deux nombres codés sur 24 bits

LD	IX, NBR1	chargement des adresses
LD	IY, NBR2	dans les pointeurs
LD	A, (IX + 0)	addition des poids faibles
ADD	A, (IY + 0)	sans tenir compte de la retenue
LD	(IY + 0), A	rangement du résultat
LD	A, (IX + 1)	addition des poids forts
ADC	A, (IY + 1)	avec la retenue propagée
LD	(IY + 1), A	
LD	A, (IX + 2)	
ADC	A, (IY + 2)	
LD	(IY + 2), A	

Ce petit programme permet de montrer la différence entre les instructions ADD et ADC. Les variables sont codées sur 24 bits, elle occupent donc 3 octets en mémoire rangés de la façon suivante :



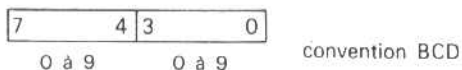
Le nombre se lit ainsi :



L'addition doit se faire en commençant par l'octet de poids faible (0) et en continuant par les octets (1) et (2) en propageant la retenue. Ceci explique le choix de l'instruction ADD pour l'octet (0) : Il ne faut pas tenir compte du bit C pour cette première addition car une éventuelle retenue fausserait le résultat. Pour la suite, la retenue se propage avec l'instruction ADC.

7.6. INSTRUCTION D'AJUSTEMENT DÉCIMAL

Cette inscription permet de faire de l'arithmétique décimale avec le Z 80. L'octet est séparé en deux mots de 4 bits. Chaque mot code l'un des chiffres 0 à 9 ce qui permet de compter de 0 à 99 avec un seul octet :



Cette représentation n'est pas compatible avec l'arithmétique binaire du Z 80. L'instruction d'ajustement décimal (DAA) permet de retourner à la convention BCD à partir du binaire :

Supposons que l'on additionne 8 et 6 :

$$\begin{array}{rcl}
 8: & 00001000 & = \quad 08 \\
 6: & 00000110 & = \quad 06 \\
 \hline
 14: & 00001110 & = \quad 0EH \quad \text{non compatible avec BCD}
 \end{array}$$

Le résultat est 0EH. Si l'on effectue l'instruction DAA sur ce résultat on obtient :

$$\begin{array}{r}
 00010100 \\
 \hline
 1 \quad 4
 \end{array}$$

On retrouve le résultat codé en BCD.

L'instruction DAA fonctionne uniquement sur l'accumulateur (adressage implicite).

Lorsque les bits 0 à 3 ont une valeur supérieure à 9 ou que le bit H est 1, DAA additionne 6 à l'accumulateur. Ensuite si les bits 4 à 7 ont une valeur supérieure à 9 ou que le bit C est 1, ces 4 bits sont incrémentés de 6.

<i>Instruction</i>	<i>Mode d'adressage</i>	<i>Flags affectés</i>
DAA	implicite	S, Z, H, P, C

7.7. INSTRUCTIONS LOGIQUES 8 BITS

L'ensemble de ces instructions s'apparente aux instructions arithmétiques. Les opérations effectuées font parties d'une "arithmétique" spéciale qui n'existe qu'avec le binaire (algèbre de Boole).

Dans ce cas l'indicateur P/V indique la parité du résultat. Toutes ces instructions utilisent l'accumulateur et s'écrivent :

OPR A, source

Avant de vous donner l'ensemble des instructions nous allons rappeler ce que sont les opérations logiques. Pour cet ensemble d'instructions il faut considérer l'octet comme un ensemble de 8 bits indépendants. Les opérations logiques s'effectuent entre deux bits de même poids.

ET logique (notation $\wedge$)

$0 \wedge 0 = 0$	01001110
$0 \wedge 1 = 0$	<u>00111010</u>
$1 \wedge 0 = 0$	00001010
$1 \wedge 1 = 1$	

OU logique (notation $\vee$)

$0 \vee 0 = 0$	01001110
$0 \vee 1 = 1$	<u>00111010</u>
$1 \vee 0 = 1$	01111110
$1 \vee 1 = 1$	

OU exclusif logique (notation $\oplus$)

$0 + 0 = 0$	01001110
$0 + 1 = 1$	<u>00111010</u>
$1 + 0 = 1$	01110100
$1 + 1 = 0$	

NON logique (notation $\overline{}$)

$\overline{1} = 0$	$\overline{01001110} = 10110001$
$\overline{0} = 1$	

on peut remarquer que $A \oplus B = (\overline{A} \wedge \overline{B}) \vee (A \wedge \overline{B})$

ET logique: $A = A \wedge s$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
AND A, γ	par registre	S, Z, H = 1, P, N = 0, C = 0
AND A, n	immédiat	"
AND A, (HL)	indirect par registre	"
AND A, (IX + d)	indexé par IX	"
AND A, (IY + d)	indexé par IY	"

OV logique: $A = A \vee s$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
OR A, γ	par registre	S, Z, H = 0, P, N = 0, C = 0
OR A, n	immédiat	"
OR A, (HL)	indirect par registre	"
OR A, (IX + d)	indexé par IX	"
OR A, (IY + d)	indexé par IY	"

OV exclusif logique: $A = A \oplus s$

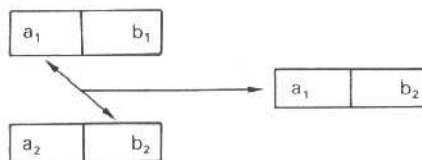
<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
XOR A, γ	par registre	S, Z, H = 0, P, N = 0, C = 0
XOR A, n	immédiat	"
XOR A, (HL)	indirect par registre	"
XOR A, (IX + d)	indexé par IX	"
XOR A, (IY + d)	indexé par IY	"

NON logique: $A = \overline{A}$

<i>Instruction</i>	<i>Mode d'adressage</i>	<i>Flags affectés</i>
CPL	implicite	H = 1, N = 1

Cette instruction ne se fait que sur l'accumulateur.

Concaténation de deux mots de 4 bits



LD	BC, VAR 1	chargement des adresses
LD	DE, VAR 2	des deux variables
LD	HL, VAR 3	adresse du résultat
LD	A, (BC)	masquage des 4 bits de poids faibles
AND	A, OFOH	de VAR 1
LD	(HL), A	rangement provisoire
LD	A, (DE)	masquage des 4 bits de poids forts
AND	A, OFH	de VAR 2
OR	A, (HL)	raccord des deux demi-octets
LD	(HL), A	rangement définitif du résultat

Ce petit programme permet de créer un octet à partir de deux moitiés d'octets. L'utilisation du AND permet de faire des masquages de bits alors que le OR au contraire permet de mettre à 1 certains bits initialement mis à 0.

Les instructions AND et OR utilisent ici un mode d'adressage indirect par registre, il est donc nécessaire d'initialiser les registres BC, DE, HL avec les adresses de 3 variables traitées par le programme.

7.8. INSTRUCTIONS ARITHMÉTIQUES 16 BITS

Certaines instructions arithmétiques du Z 80 sont réalisables en utilisant les registres 16 bits ou les paires de registres 8 bits. Les opérands ont donc une capacité de 16 bits. Cet ensemble d'instruction res-

semble donc aux instructions arithmétiques 8 bits. L'indicateur P/V indique un débordement.

L'accumulateur ne possédant que 8 bits, il n'est pas utilisé par ces instructions, c'est-à-dire que l'arithmétique 16 bits définit dans les opérandes à la fois la source et la destination.

OPR source, destination

Addition 16 bits: $PP = PP + SS$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
ADD HL, BC	par registre	N = 0, C
ADD HL, DE	"	"
ADD HL, HL	"	"
ADD HL, SP	"	"
ADD IX, BC	"	"
ADD IX, DE	"	"
ADD IX, IX	"	"
ADD IX, SP	"	"
ADD IY, BC	"	"
ADD IY, DE	"	"
ADD IY, IY	"	"
ADD IY, SP	"	"

Addition 16 bits avec retenue: $HL = HL + ss + CY$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
ADC HL, BC	par registre	S, Z, V, N = 0, C
ADC HL, DE	"	"
ADC HL, HL	"	"
ADC HL, SP	"	"

Soustraction 16 bits avec retenue: $HL = HL - ss - CY$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
SBC HL, BC	par registre	S, Z, V, N = 1, C
SBC HL, DE	"	"
SBC HL, HL	"	"
SBC HL, SP	"	"

Incrément 16 bits: $ss = ss + 1$

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
INC BC	par registre	aucun
INC DE	"	"
INC HL	"	"
INC SP	"	"
INC IX	"	"
INC IY	"	"

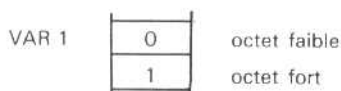
Décrément 16 bits: $ss = ss - 1$

<i>Instruction</i>	<i>Mode d'adresage de la source</i>	<i>Flags affectés</i>
DEC BC	par registre	aucun
DEC DE	"	"
DEC HL	"	"
DEC SP	"	"
DEC IX	"	"
DEC IY	"	"

Soustraction 16 bits

LD	HL, (VAR 1)	initialisation
LD	DE, (VAR 1)	des registres
OR	A, A	remise à 0 du flag C
SBC	HL, DE	soustraction: VAR 1 - VAR 2
LD	(RESULT), HL	rangement du résultat

Les opérations 16 bits ne s'effectuent qu'entre registres. Il est donc nécessaire de charger au préalable ces registres avec le contenu des variables en mémoire centrale. Ces variables codées sur 16 bits sont rangées ainsi :



Il est intéressant de noter que l'instruction OR A, A n'affecte pas l'accumulateur et met le bit C à 0.

7.9. INSTRUCTIONS DE SAUT

Ces instructions vous sont déjà familières car elles sont l'équivalent des GOTO et IF THEN GOTO du BASIC. Comme nous l'avons déjà dit, ces instructions vont former le squelette du programme, il est donc important de bien connaître les possibilités offertes par cet ensemble d'instructions.

Ces instructions utilisent trois modes de branchements que nous avons déjà détaillés :

- relatif,
- indirect,
- page zéro.

Le saut peut être soit conditionnel (équivalent de IF THEN GOTO) ou inconditionnel (GOTO). Les sauts conditionnels utilisent la position des flags pour décider si le branchement doit être effectué ou non. Il y a huit conditions possibles repérées par un mnémonique :

NZ	:	flag Z = 0	résultat non nul
Z	:	flag Z = 1	résultat nul
NC	:	flag C = 0	pas de retenue
C	:	flag C = 1	retenue présente
PO	:	flag P/V = 0	parité impaire

PE : flag P/V = 1 parité paire
P : flag S = 0 signe positif
M : flag S = 1 signe négatif

La syntaxe d'un branchement conditionnel est la suivante :

SAUT CONDITION, ÉTIQUETTE

SAUT est soit un branchement direct soit un branchement relatif.

La différence entre les deux est :

direct : instruction 3 octets
branchement sur toute la zone mémoire

relatif : instruction 2 octets
branchement dans un espace limité à 256 adresses : + 127,
- 128

CONDITION est l'un des mnémoniques cités.

ÉTIQUETTE est l'une des étiquettes de branchement du programme située dans la zone étiquette ou définie par une directive.

Branchement direct

<i>Instruction</i>	<i>Condition</i>
JP ETIQ	inconditionnel
JP NZ, ETIQ	flag Z = 0
JP Z, ETIQ	flag Z = 1
JP NC, ETIQ	flag C = 0
JP C, ETIQ	flag C = 1
JP PO, ETIQ	flag P/V = 0
JP PE, ETIQ	flag P/V = 1
JP P, ETIQ	flag S = 0
JP M, ETIQ	flag S = 1

Branchement relatif

Instruction	Condition
JR ETIQ	inconditionnel
JR C, ETIQ	flag C = 1
JR NC, ETIQ	flag C = 0
JR Z, ETIQ	flag Z = 1
JR NZ, ETIQ	flag Z = 0

Branchement indirect :

L'adresse de branchement est le contenu de l'un des registres 16 bits HL, IX ou IY :

Instruction	Condition
JP (HL)	inconditionnel
JP (IX)	"
JP (IY)	"

Branchement et décrétement :

Le Z 80 possède une instruction spéciale pour faire des boucles. Cette instruction est l'association d'un saut et d'un décrétement du registre B. On peut la résumer ainsi :

- décrétement de B,
- si B = 0 continuer
- sinon se brancher à l'étiquette spécifiée.

Le branchement est relatif et s'effectue donc sur une plage de 256 adresses.

L'instruction qui effectue ceci est :

DJNZ ETIQ

Cette instruction fait penser à la boucle FOR ... NEXT du BASIC :

```
FOR I = 1 TO 10
  ...
NEXT I
```

```
BOUCLE
LD B, 10
...
DJNZ BOUCLE
```

dans les deux cas la séquence est effectuée 10 fois. La version assembleur utilise le registre B comme compteur de boucle.

```
50 IF VAR 1 = VAR 2 THEN GOTO 60 ELSE VAR 1 = VAR 1 + 1
```

```
60 ...
```

```
LD    A, (VAR 1)    chargement du contenu de VAR 1
```

```
LD    HL, VAR 2     chargement de l'adresse  
de VAR 2
```

```
CP    (HL)          IF : comparaison
```

```
JR    Z, SUITE      THEN : test positif
```

```
INC    A            ELSE : test négatif
```

```
LD    (VAR 1), A
```

SUITE

L'emploi de l'instruction CP associée aux branchements conditionnels permet de faire l'équivalent en assembleur de l'instruction BASIC IF... THEN... ELSE...

Addition de deux nombres codés sur 32 bits

```
LD    B, 4          faire 4 fois la boucle
```

```
LD    DE, VAR 1     chargement des adresses
```

```
LD    HL, VAR 2
```

```
OR    A, A          mise à 0 du flag C
```

```
BOUCLE LD    A, (DE)  addition des octets
```

```
ADC    A, (HL)
```

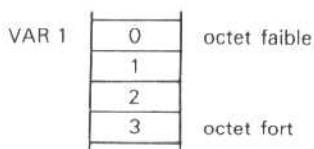
```
LD    (HL), A       rangement du résultat
```

```
INC    HL           octet suivant
```

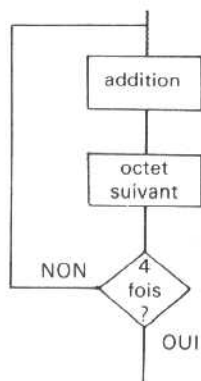
```
INC    DE
```

```
DJNZ   BOUCLE       recommencer 4 fois
```

Ce programme effectue l'addition de deux nombres codés sur 32 bits :



Contrairement à l'exemple précédent, l'addition successive des octets se fait en utilisant une boucle :



Le compteur de boucle est le registre B chargé avec la valeur 4.
Ceci permet d'utiliser l'instruction DJNZ.

7.10. SOUS-PROGRAMMES

7.10.1. Appels de sous-programmes

Ces instructions fonctionnent exactement comme le GOSUB du BASIC. Il existe donc deux instructions jumelées :

appel : CALL	(GOSUB)
retour : RET	(RETURN)

L'appel de sous-programmes peut être conditionnel, (IF... THEN GOSUB). Les conditions sont dans ce cas les mêmes que pour les instructions de saut.

De même le retour de sous-programme peut aussi être conditionnel (IF... THEN RETURN).

La syntaxe de ces instructions est :

CALL	CONDITION, ÉTIQUETTE	appel
RET	CONDITION	retour

CONDITION est l'un des mnémoniques cités dans les instructions de saut.

ÉTIQUETTE est le label attribué au sous-programme appelé :

<i>Instruction</i>	<i>Condition</i>
CALL ETIQ	inconditionnel
CALL NZ, ETIQ	flag Z = 0
CALL Z, ETIQ	flag Z = 1
CALL NC, ETIQ	flag C = 0
CALL C, ETIQ	flag C = 1
CALLPO, ETIQ	flag P/V = 0
CALL PE, ETIQ	flag P/V = 1
CALL P, ETIQ	flag S = 0
CALL M, ETIQ	flag S = 1

7.10.2. Instruction de retour sous-programme

<i>Instruction</i>	<i>Condition</i>
RET	inconditionnel
RET NZ	flag Z = 0
RET Z	flag Z = 1
RET NC	flag C = 0
RET C	flag C = 1
RET PO	flag P/V = 0
RET PE	flag P/V = 1
RET P	flag S = 0
RET M	flag S = 1

Appel de sous-programme en page zéro :

Le Z 80 possède une instruction spéciale appelée RESTART qui permet d'effectuer un branchement sur un sous-programme résidant à une adresse spéciale en page zéro. L'intérêt de ce RESTART est qu'il n'utilise qu'un seul octet contrairement au CALL qui en nécessite trois. Les adresses de branchement en page zéro sont au nombre de 8 :

00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H

La syntaxe est la suivante :

RST p

ceci produit exactement la même chose que CALL p

<i>Instruction</i>	<i>Condition</i>
RST 00H	inconditionnel
RST 08H	" "
RST 10H	"
RST 18H	"
RST 20H	"
RST 28H	"
RST 30H	"
RST 38H	"

Attention !!

Les instructions RST p sont très souvent utilisées par le système d'exploitation et sont donc délicates à manier.

Appel de sous-programmes d'addition 32 bits

LD	DE, VAR 1	chargement des adresses
LD	HL, VAR 2	des variables à additionner
CALL	ADDI	appel du programme addition

ADDI	OR	A, A	programme d'addition
	LD	B, 4	
BOUCLE	LD	A, (DE)	
	ADC	A, (HL)	
	LD	(HL), A	
	INC	HL	
	INC	DE	
	DJNZ	BOUCLE	
	RET		

Le fait d'utiliser un sous-programme permet de traiter plusieurs variables par le même programme d'addition. Dans l'exemple précédent ceci n'était pas possible, il fallait répéter la séquence d'addition à chaque fois. Maintenant il suffit de charger les deux registres DE et HL avec les adresses des deux variables à additionner et d'appeler les sous-programmes ADDI :

LD	DE, X1	il faut trois instructions
LD	HL, X2	pour faire l'addition
CALL	ADDI	avec le sous-programme

7.11. INSTRUCTIONS DE MANIPULATION DE LA PILE

Nous ne reviendrons pas sur le fonctionnement de la pile qui a déjà été vu. Le Z 80 possède un ensemble d'instructions qui permet de placer et de retirer ses registres de la pile. Cette sauvegarde sur la pile se fait toujours par paire de registre ou par registre 16 bits.

L'action d'empiler un registre s'appelle **PUSH**, l'action de désem-piler s'appelle **POP**.

<i>Instruction</i>		<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
PUSH	BC	par registre	aucun
PUSH	DE	"	"
PUSH	HL	"	"
PUSH	AF	"	"
PUSH	IX	"	"
PUSH	IY	"	"
POP	BC	"	"
POP	DE	"	"
POP	HL	"	"
POP	AF	"	"
POP	IX	"	"
POP	IY	"	"

Appel de sous-programmes avec sauvegarde du contexte :

	LD	DE, VAR 1	
	LD	HL, VAR 2	
	CALL	ADDI	
ADDI	PUSH	AF	sauvegarde des
	PUSH	BC	registres AF et BC
	OR	A, A	
	LD	B, 4	
BOUCLE	LD	A, (DE)	
	ADC	A, (HL)	
	LD	(HL), A	
	INC	HL	
	INC	DE	
	DJNZ	BOUCLE	
	POP	BC	restitution des registres
	POP	AF	AF et BC
	RET		

Cet exemple est identique au précédent mais maintenant, au retour de l'addition les registres BC, AF ne sont pas modifiés. On remarque l'ordre des PUSH et du POP. Il faut respecter la règle :

Le premier PUSH correspond au dernier POP.

7.12. INSTRUCTIONS SUR LES BITS : SET RESET

Jusqu'à maintenant toutes les instructions que nous vous avons présentées manipulent des octets (8 bits). Cet ensemble d'instruction SET, RESET permet de manipuler un seul bit sur un registre ou une case mémoire.

Les seules opérations que l'on peut effectuer sur un bit sont sa mise à 1 (SET) ou sa mise à 0 (RESET) ou son test.

Le bit sur lequel opère l'instruction est repéré par un numéro de 0 à 7 :

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

La syntaxe de ces instructions est :

OPR numéro du bit, source

Test de bit : Z = bit testé

<i>Instruction</i>	<i>Mode d'adressage de la source</i>	<i>Flags affectés</i>
* BIT b, γ	par registre	Z, H = 1, N = 0
BIT b, (HL)	indirect par registre	"
BIT b, (IX + d)	indexé par IX	"
BIT b, (IY + d)	indexé par IY	"

(* b = nombre de 0 à 7, par exemple BIT 6, C)

le test de bit affecte uniquement le flag Z de façon significative :

Z = 0 si le bit testé = 1

Z = 1 si le bit testé = 0

Mise à 1 de bit: bit = 1

Instruction	Mode d'adressage de la source	Flags affectés
SET b, γ	par registre	aucun
SET b, (HL)	indirect par registre	"
SET b, (IX + d)	indexé par IX	"
SET b, (IY + d)	indexé par IY	"

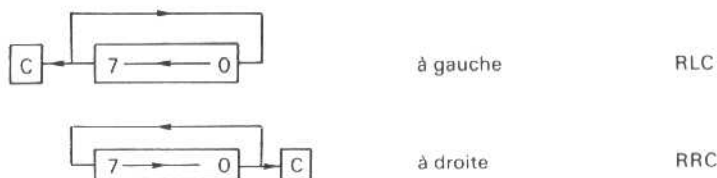
Mise à 0 de bit: bit = 1

Instruction	Mode d'adressage de la source	Flags affectés
RES b, γ	par registre	aucun
RES b, (HL)	indirect par registre	"
RES b, (IX + d)	indexé par IX	"
RES b, (IY + d)	indexé par IY	"

7.13. INSTRUCTIONS DE DÉCALAGE

Il existe un très grand nombre de décalages possibles. Il est difficile de ne pas s'y perdre. La plupart de ces décalages utilisent le bit de retenue C qui de ce fait peut être considéré comme un 9^e bit participant au décalage.

7.13.1. Rotation circulaire



Le bit sortant d'un côté rentre par l'autre côté et se trouve recopié dans le flag C.

Instruction		Mode d'adressage de la source	Flags affectés
RLC	γ	par registre	S, Z, H = 0, P, N = 0, C
RLC	(HL)	indirect par registre	"
RLC	(IX + d)	indexé par IX	"
RLC	(IY + d)	indexé par IY	"
RRC	γ	par registre	"
RRC	(HL)	indirect par registre	"
RRC	(IX + d)	indexé par IX	"
RRC	(IY + d)	indexé par IY	"

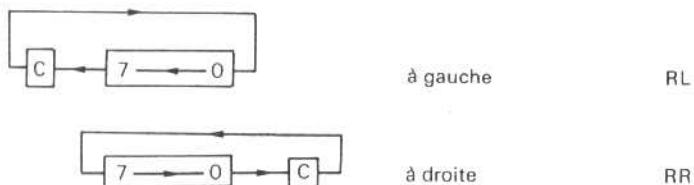
Il existe deux instructions différentes qui pourtant effectuent exactement la même chose. Il s'agit des rotations circulaires sur l'accumulateur :

RLC A = RLCA

RRC A = RRCA

Ceci est une réminiscence du jeu d'instruction du 8080. Les instructions RRCA et RLCA occupent un seul octet et n'affectent pas les flags S, Z, P. Les instructions RRC A et RLC A occupent deux octets.

7.13.2. Rotation circulaire à travers la retenue

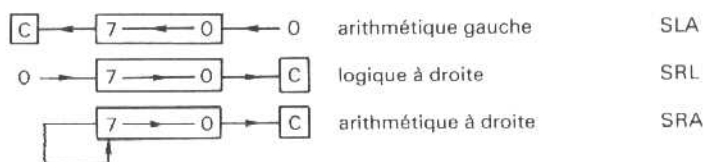


Le bit sortant est recopié dans l'indicateur C. Le bit contenu dans l'indicateur C rentre dans l'octet.

Instruction		Mode d'adressage de la source	Flags affectés
RL	γ	par registre	S, Z, H = 0, P, N = 0, C
RL	(HL)	indirect par registre	"
RL	(IX + d)	indexé par IX	"
RL	(IY + d)	indexé par IY	"
RR	γ	par registre	"
RR	(HL)	indirect par registre	"
RR	(IX + d)	indexé par IX	"
RR	(IY + d)	indexé par IY	"

Il existe ici aussi les deux instructions du 8080 RLA et RRA qui sont identiques à RL A et RR A mis à part qu'elles n'affectent pas les flags S, Z, P et n'occupent qu'un octet.

7.13.3. Décalages logiques et arithmétiques



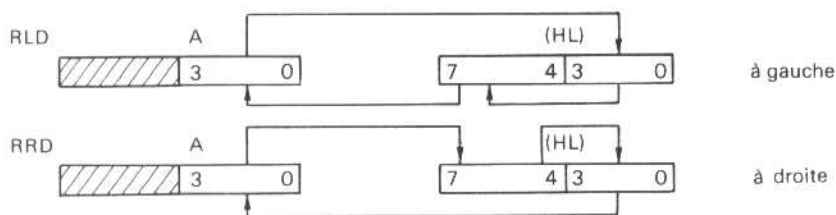
Pour le décalage arithmétique à droite le bit 7 est recopié dans lui-même et non mis à 0. Le terme arithmétique provient du fait que ce décalage conserve le signe de l'octet (bit 7).

Instruction		Mode d'adressage de la source	Flags affectés
SLA	γ	par registre	S, Z, H = 0, P, N = 0, C
SLA	(HL)	indirect par registre	"
SLA	(IX + d)	indexé par IX	"
SLA	(IX + d)	indexé par IY	"
SRL	γ	par registre	"
SRL	(HL)	indirect par registre	"
SRL	(IX + d)	indexé par IX	"
SRL	(IY + d)	index par IY	"
SRA	γ	par registre	"
SRA	(HL)	indirect par registre	"
SRA	(IX + d)	indexé par IX	"
SRA	(IY + d)	indexé par IY	"

7.13.4. Rotation circulaire BCD

Cette rotation est adaptée à l'arithmétique décimale. En arithmétique décimale l'octet est séparé en deux mots de 4 bits. Chacun de ces mots représente le code binaire des chiffres 0 à 9. Les rotations BCD opèrent donc sur ces mots de 4 bits.

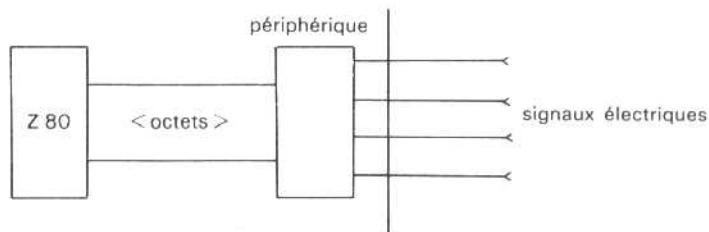
La destination est l'accumulateur, le second mot source est désigné par un adressage indirect par registre :



Instruction		Mode d'adressage de la source	Flags affectés
RLD		indirect par registre HL	S, Z, H = 0, P, N = 0, C
RRD		indirect par registre HL	"

7.14. INSTRUCTIONS D'ENTRÉE-SORTIE

Le Z 80 possède deux instructions qui lui permettent de dialoguer avec les périphériques : IN et OUT. Un périphérique est un composant électronique qui permet de mettre en forme les octets envoyés par le microprocesseur vers l'extérieur ou de recevoir des signaux électriques externes en les mettant sous formes d'octets accessibles pour le Z 80 :



De cette façon le Z 80 possède une ouverture vers l'extérieur qui lui permet par exemple d'actionner une imprimante, de gérer un magnétophone à cassettes, ou de commander n'importe quelle machine. Avec ces périphériques le microprocesseur peut effectuer plusieurs tâches différentes ; chaque tâche possède son périphérique d'entrée-sortie. Pour que le Z 80 se retrouve au milieu de tous ces "bras" tendus vers l'extérieur il va leur attribuer à chacun une adresse exactement comme pour la mémoire centrale. Cette adresse est contenue dans un octet ce qui donne 256 "bras" possibles au microprocesseur.

Il ne faut pas confondre une adresse de périphérique et une adresse mémoire. Les deux sont complètement séparés. Le dialogue microprocesseur périphérique se limite au transfert d'octets. Le Z 80 possède deux modes d'adressage pour communiquer avec ses périphériques :

- direct,
- indirect par le registre C.

Les deux instructions sont :

- | | |
|-----|---|
| IN | transfert d'octet périphérique vers Z 80 |
| OUT | transfert d'octet Z 80 vers périphérique. |

En adressage direct le transfert s'effectue entre l'accumulateur A et le périphérique. L'opérande est l'adresse du périphérique appelé. En adressage indirect le transfert s'effectue entre l'un des registres A, B, C, D, E, H, L et le périphérique.

Instruction		Mode d'adressage des périphériques	Flags affectés
IN	A, (n)	direct	aucun
IN	γ , (C)	indirect par registre C	S, Z, H, P, N = 0
OUT	(n), A	direct	aucun
OUT	(C), γ	indirect par registre C	S, Z, H, P, N = 0

par exemple :

IN A, (ADRP)

L'accumulateur est chargé par un octet provenant du périphérique dont l'adresse est ADRP définie sur 8 bits.

7.15. INSTRUCTIONS DE CHAÎNES

Ces instructions très puissantes du Z 80 travaillent sur un espace mémoire de plusieurs octets contigus. Elles se scindent en trois groupes :

- instructions de transfert de chaîne,
- instructions de recherche d'octet dans une chaîne,
- instructions d'entrée-sortie de chaîne.

7.15.1. Transfert de chaîne

LDI : $\left\{ \begin{array}{l} (DE) = (HL) \\ DE = DE + 1 \\ HL = HL + 1 \\ BC = BC - 1 \end{array} \right.$

La case mémoire dont l'adresse est contenue dans HL est recopiée à l'adresse contenue dans DE puis les deux paires de registres DE et HL sont incrémentées. La paire de registre BC est décrémentée.

LDIR : $\left\{ \begin{array}{l} (DE) = (HL) \\ DE = DE + 1 \\ HL = HL + 1 \\ BC = BC - 1 \end{array} \right.$ l'instruction se répète jusqu'à ce que BC = 0

La lettre R qui suit LDI signifie répétition. Les instructions se terminant par R sont donc des instructions opérant sur une chaîne d'octets.

LDD : $\left\{ \begin{array}{l} (DE) = (HL) \\ DE = DE - 1 \\ HL = HL - 1 \\ BC = BC - 1 \end{array} \right.$

LDDR : $\left\{ \begin{array}{l} (DE) (HL) \\ DE = DE - 1 \\ HL = HL - 1 \\ BC = BC - 1 \end{array} \right.$ l'instruction se répète jusqu'à ce que BC = 0

7.15.2. Recherche d'octet dans une chaîne

CPI : $\left\{ \begin{array}{l} A - (HL) \\ HL = HL + 1 \\ BC = BC - 1 \end{array} \right.$

Le Z 80 effectue une comparaison entre le contenu de l'accumulateur et la case mémoire (HL).

CPIR : $\left\{ \begin{array}{l} A - (HL) \\ HL = HL + 1 \\ BC = BC - 1 \end{array} \right.$ l'instruction se répète jusqu'à ce que BC = 0 ou A = (HL)

$$\text{CPD : } \begin{cases} A = (HL) \\ HL = HL - 1 \\ BC = BC - 1 \end{cases}$$

$$\text{CPDR : } \begin{cases} A = (HL) \\ HL = HL - 1 \\ BC = BC - 1 \end{cases}$$

l'instruction se répète
jusqu'à ce que $BC = 0$

7.15.3. Entrée-sortie de chaîne

$$\text{INI : } \begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL + 1 \end{cases}$$

$$\text{INIR : } \begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL + 1 \end{cases}$$

l'instruction se répète
jusqu'à ce que $B = 0$

$$\text{IND : } \begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL - 1 \end{cases}$$

$$\text{INDR : } \begin{cases} (HL) = (C) \\ B = B - 1 \\ HL = HL - 1 \end{cases}$$

l'instruction se répète
jusqu'à ce que $B = 0$

$$\text{OUTI : } \begin{cases} (C) = (HL) \\ HL = HL + 1 \\ B = B - 1 \end{cases}$$

$$\text{OTIR : } \begin{cases} (C) = (HL) \\ HL = HL + 1 \\ B = B - 1 \end{cases}$$

l'instruction se répète
jusqu'à ce que $B = 0$

OUTD : $\begin{cases} (C) = (HL) \\ HL = HL - 1 \\ B = B - 1 \end{cases}$

OTDR : $\begin{cases} (C) = (HL) \\ HL = HL - 1 \\ B = B - 1 \end{cases}$ l'instruction se répète jusqu'à ce que $B = 0$

C contient l'adresse du périphérique ou s'effectue le transfert.

<i>Instruction</i>	<i>Flags affectés</i>
LDI	$H = 0, N = 0, P/V = 0$ si $BC - 1 = 0$
LDIR	$H = 0, P/V = 0, N = 0$
LDD	$H = 0, N = 0, P/V = 0$ si $BC - 1 = 0$
LDDR	$H = 0, P/V = 0, N = 0$
CPI	$S, H, N = 1, Z = 1$ si $A = (HL), P/V = 0$ si $BC - 1 = 0$
CPIR	"
CPD	"
CPDR	"
INI	$N = 1, Z = 1$ si $B - 1 = 0$
INIR	$N = 1, Z = 1$
IND	$N = 1, Z = 1$ si $B - 1 = 0$
INDR	$N = 1, Z = 1$
OUTI	$N = 1, Z = 1$ si $B - 1 = 0$
OTIR	$N = 1, Z = 1$
OUTD	$N = 1, Z = 1$ si $B - 1 = 0$
OTDR	$N = 1, Z = 1$

7.16. INSTRUCTIONS D'USAGE GÉNÉRAL

Les instructions que nous donnons ici sont d'un usage très particulier et ne sont pas toujours facilement utilisables. Nous en donnons la liste sans les détailler.

7.17. INSTRUCTIONS SUR LES INTERRUPTIONS

DI	defend les interruptions
EI	autorise les interruptions
IM0	met le Z 80 en mode interruption 0
IM1	met le Z 80 en mode interruption 1
IM2	met le Z 80 en mode interruption 2
RETI	retour d'interruption
RETN	retour d'interruption non marquable.

7.18. INSTRUCTIONS DE CONTRÔLE

Le Z 80 possède deux instructions qui lui permettent de "se reposer".

NOP	ne rien faire
HALT	arrêt du microprocesseur

L'instruction NOP n'exécute aucune opération et ne modifie strictement rien. Son intérêt n'est quand même pas négligeable. D'une part le NOP permet de supprimer des instructions directement dans le programme objet sans repasser par un nouvel assemblage. Il suffit dans ce cas de remplacer l'instruction à supprimer par une série de NOP. D'autre part le NOP peut faire perdre du temps au Z 80. Ceci peut être utile lorsque le Z 80 dialogue avec des périphériques plus lents que lui. Le NOP permet au Z 80 d'attendre un peu sans rien faire que le périphérique réponde.

L'instruction HALT quant à elle interrompt complètement le Z 80. Celui-ci ne peut repartir que par une action matérielle venant de l'extérieur : RESET ou interruption. En pratique le HALT n'est jamais utilisé sauf pour des configurations matérielles spéciales, ce qui n'est pas le cas pour les micro-ordinateurs.

ANNEXE 1

Liste des codes opérations par ordre numérique

0000	00	1	NOP	0065	47	72	LD B,A
0001	018405	2	LD BC,NN	0066	48	73	LD C,B
0004	02	3	LD (BC),A	0067	49	74	LD C,C
0005	03	4	INC BC	0068	4A	75	LD C,D
0006	04	5	INC B	0069	4B	76	LD C,E
0007	05	6	DEC B	006A	4C	77	LD C,H
0008	0620	7	LD B,N	006B	4D	78	LD C,I
000A	07	8	RLCA	006C	4E	79	LD C,(HL)
000B	08	9	EX AF,AF	006D	4F	80	LD C,A
000C	09	10	ADD HL,BC	006E	50	81	LD D,B
000D	0A	11	LD A,(BC)	006F	51	82	LD D,C
000E	0B	12	DEC BC	0070	52	83	LD D,D
000F	0C	13	INC C	0071	53	84	LD D,E
0010	0D	14	DEC C	0072	54	85	LD D,H
0011	0E20	15	LD C,N	0073	55	86	LD D,L
0013	0F	16	RRCA	0074	56	87	LD D,(HL)
0014	102E	17	DJNZ DIS	0075	57	88	LD D,A
0016	118405	18	LD DE,NN	0076	58	89	LD E,B
0019	12	19	LD (DE),A	0077	59	90	LD E,C
001A	13	20	INC DE	0078	5A	91	LD E,D
001B	14	21	INC D	0079	5B	92	LD E,E
001C	15	22	DEC D	007A	5C	93	LD E,H
001D	1620	23	LD D,N	007B	5D	94	LD E,L
001F	17	24	RLA	007C	5E	95	LD E,(HL)
0020	182E	25	JR DIS	007D	5F	96	LD E,A
0022	19	26	ADD HL,DE	007E	60	97	LD H,B
0023	1A	27	LD A,(DE)	007F	61	98	LD H,C
0024	1B	28	DEC DE	0080	62	99	LD H,D
0025	1C	29	INC E	0081	63	100	LD H,E
0026	1D	30	DEC E	0082	64	101	LD H,H
0027	1E20	31	LD E,N	0083	65	102	LD H,L
0029	1F	32	RRR	0084	66	103	LD H,(HL)
002A	202E	33	JR NZ,DIS	0085	67	104	LD H,A
002C	218405	34	LD HL,NN	0086	68	105	LD L,B
002F	228405	35	LD (NN),HL	0087	69	106	LD L,C
0032	23	36	INC HL	0088	6A	107	LD L,D
0033	24	37	INC H	0089	6B	108	LD L,E
0034	25	38	DEC H	008A	6C	109	LD L,H
0035	2620	39	LD H,N	008B	6D	110	LD L,L
0037	27	40	DAA	008C	6E	111	LD L,(HL)
0038	282E	41	JR Z,DIS	008D	6F	112	LD L,A
003A	29	42	ADD HL,HL	008E	70	113	LD (HL),B
003B	2A8405	43	LD HL,(NN)	008F	71	114	LD (HL),C
003E	2B	44	DEC HL	0090	72	115	LD (HL),D
003F	2C	45	INC L	0091	73	116	LD (HL),E
0040	2D	46	DEC L	0092	74	117	LD (HL),H
0041	2E20	47	LD L,N	0093	75	118	LD (HL),L
0043	2F	48	CPL	0094	76	119	HALT
0044	302E	49	JR NC,DIS	0095	77	120	LD (HL),A
0046	318405	50	LD SP,NN	0096	78	121	LD A,B
0049	328405	51	LD (NN),A	0097	79	122	LD A,C
004C	33	52	INC SP	0098	7A	123	LD A,D
004D	34	53	INC (HL)	0099	7B	124	LD A,E
004E	35	54	DEC (HL)	009A	7C	125	LD A,H
004F	3620	55	LD (HL),N	009B	7D	126	LD A,L
0051	37	56	SCF	009C	7E	127	LD A,(HL)
0052	382E	57	JR C,DIS	009D	7F	128	LD A,A
0054	39	58	ADD HL,SP	009E	80	129	ADD A,B
0055	3A8405	59	LD A,(NN)	009F	81	130	ADD A,C
0058	3B	60	DEC SP	00A0	82	131	ADD A,D
0059	3C	61	INC A	00A1	83	132	ADD A,E
005A	3D	62	DEC A	00A2	84	133	ADD A,H
005B	3E20	63	LD A,N	00A3	85	134	ADD A,L
005D	3F	64	CCF	00A4	86	135	ADD A,(HL)
005E	40	65	LD B,B	00A5	87	136	ADD A,A
005F	41	66	LD B,C	00A6	88	137	ADC A,B
0060	42	67	LD B,D	00A7	89	138	ADC A,C
0061	43	68	LD B,E	00A8	8A	139	ADC A,D
0062	44	69	LD B,H,NN	00A9	8B	140	ADC A,E
0063	45	70	LD B,L	00AA	8C	141	ADC A,H
0064	46	71	LD B,(HL)	00AB	8D	142	ADC A,L

00AC	8E	143	ADC A,(HL)	010B	DA8405	218	JPC,NN
00AD	8F	144	ADC A,A	010E	DB20	219	IN A,N
00AE	90	145	SUB B	0110	DC8405	220	CALL C,NN
00AF	91	146	SUB C	0113	DE20	221	SBC A,N
00B0	92	147	SUB D	0115	DF	222	RST 18H
00B1	93	148	SUB E	0116	E0	223	RET PO
00B2	94	149	SUB H	0117	E1	224	POP HL
00B3	95	150	SUB L	0118	E28405	225	JP P,NN
00B4	96	151	SUB (HL)	011B	E3	226	EX (SP),HL
00B5	97	152	SUB A	011C	E48405	227	CALL P,NN
00B6	98	153	SBC A,B	011F	E5	228	PUSH HL
00B7	99	154	SBC A,C	0120	E620	229	AND N
00B8	9A	155	SBC A,D	0122	E7	230	RST 20H
00B9	9B	156	SBC A,E	0123	E8	231	RET PE
00BA	9C	157	SBC A,H	0124	E9	232	JP (HL)
00BB	9D	158	SBC A,L	0125	E48405	233	JP P,NN
00BC	9E	159	SBC A,(HL)	0128	EB	234	EX DE,HL
00BD	9F	160	SBC A,A	0129	EC8405	235	CALL P,NN
00BE	A0	161	AND B	012C	EE20	236	XOR N
00BF	A1	162	AND C	012E	EF	237	RST 28H
00C0	A2	163	AND D	012F	F0	238	RET P
00C1	A3	164	AND E	0130	F1	239	POP AF
00C2	A4	165	AND H	0131	F28405	240	JP P,NN
00C3	A5	166	AND L	0134	F3	241	DI
00C4	A6	167	AND (HL)	0135	F48405	242	CALL P,NN
00C5	A7	168	AND A	0138	F5	243	PUSH AF
00C6	A8	169	XOR B	0139	F620	244	OR N
00C7	A9	170	XOR C	013B	F7	245	RST 30H
00C8	AA	171	XOR D	013C	F8	246	RET M
00C9	AB	172	XOR E	013D	F9	247	LD SP,HL
00CA	AC	173	XOR H	013E	FA8405	248	JP M,NN
00CB	AD	174	XOR L	0141	FB	249	EI
00CC	AE	175	XOR (HL)	0142	FC8405	250	CALL M,NN
00CD	AF	176	XOR A	0145	FE20	251	CP N
00CE	B0	177	OR B	0147	FF	252	RST 38H
00CF	B1	178	OR C	0148	CB00	253	RLC B
00D0	B2	179	OR D	014A	CB01	254	RLC C
00D1	B3	180	OR E	014C	CB02	255	RLC D
00D2	B4	181	OR H	014E	CB03	256	RLC E
00D3	B5	182	OR L	0150	CB04	257	RLC H
00D4	B6	183	OR (HL)	0152	CB05	258	RLC L
00D5	B7	184	OR A	0154	CB06	259	RLC (HL)
00D6	B8	185	CP B	0156	CB07	260	RLC A
00D7	B9	186	CP C	0158	CB08	261	RRC B
00D8	BA	187	CP D	015A	CB09	262	RRC C
00D9	BB	188	CP E	015C	CB0A	263	RRC D
00DA	BC	189	CP H	015E	CB0B	264	RRC E
00DB	BD	190	CP L	0160	CB0C	265	RRC H
00DC	BE	191	CP (HL)	0162	CB0D	266	RRC L
00DD	BF	192	CP A	0164	CB0E	267	RRC (HL)
00DE	C0	193	RET NZ	0166	CB0F	268	RRC A
00DF	C1	194	POP BC	0168	CB10	269	RL B
00E0	C28405	195	JP NZ,NN	016A	CB11	270	RL C
00E3	C38405	196	JP NN	016C	CB12	271	RL D
00E6	C48405	197	CALL NZ,NN	016E	CB13	272	RL E
00E9	C5	198	PUSH BC	0170	CB14	273	RL H
00EA	C620	199	ADD A,N	0172	CB15	274	RL L
00EC	C7	200	RST 0	0174	CB16	275	RL (HL)
00ED	C8	201	RET Z	0176	CB17	276	RL A
00EE	C9	202	RET J	0178	CB18	277	RR B
00EF	CA8405	203	JP Z,NN	017A	CB19	278	RR C
00F2	CC8405	204	CALL Z,NN	017C	CB1A	279	RR D
00F5	CD8405	205	CALL NN	017E	CB1B	280	RR E
00F8	CE20	206	ADC A,N	0180	CB1C	281	RR H
00FA	CF	207	RST 6	0182	CB1D	282	RR L
00FB	D0	208	RET NC	0184	CB1E	283	RR (HL)
00FC	D1	209	POP DE	0186	CB1F	284	RR A
00FD	D28405	210	JP NC,NN	0188	CB20	285	SLA B
0100	D320	211	OUT N,A	018A	CB21	286	SLA C
0102	D48405	212	CALL NC,NN	018C	CB22	287	SLA D
0105	D5	213	PUSH DE	018E	CB23	288	SLA E
0106	D620	214	SUB N	0190	CB24	289	SLA H
0108	D7	215	RST 10H	0192	CB25	290	SLA L
0109	D8	216	RET C	0194	CB26	291	SLA (HL)
010A	D9	217	EXX	0196	CB27	292	SLA A

0198	CB28	293	SRA B	0230	CB7C	369	BIT ?H
019A	CB29	294	SRA C	0232	CB7D	370	BIT 7,L
019C	CB2A	295	SRA D	0234	CB7E	371	BIT 7,(HL)
019E	CB2B	296	SRA E	0236	CB7F	372	BIT 7,A
01A0	CB2C	297	SRA H	0238	CB80	373	RES 0,B
01A2	CB2D	298	SRA L	023A	CB81	374	RES 0,C
01A4	CB2E	299	SRA (HL)	023C	CB82	375	RES 0,D
01A6	CB2F	300	SRA A	023E	CB83	376	RES 0,E
01A8	CB38	301	SRL B	0240	CB84	377	RES 0,H
01AA	CB39	302	SRL C	0242	CB85	378	RES 0,L
01AC	CB3A	303	SRL D	0244	CB86	379	RES 0,(HL)
01AE	CB3B	304	SRL E	0246	CB87	380	RES 0,A
01B0	CB3C	305	SRL H	0248	CB88	381	RES 1,B
01B2	CB3D	306	SRL L	024A	CB89	382	RES 1,C
01B4	CB3E	307	SRL (HL)	024C	CB8A	383	RES 1,D
01B6	CB3F	308	SRL A	024E	CB8B	384	RES 1,E
01B8	CB40	309	BIT 0,B	0250	CB8C	385	RES 1,H
01BA	CB41	310	BIT 0,C	0252	CB8D	386	RES 1,L
01BC	CB42	311	BIT 0,D	0254	CB8E	387	RES 1,(HL)
01BE	CB43	312	BIT 0,E	0256	CB8F	388	RES 1,A
01C0	CB44	313	BIT 0,H	0258	CB90	389	RES 2,B
01C2	CB45	314	BIT 0,L	025A	CB91	390	RES 2,C
01C4	CB46	315	BIT 0,(HL)	025C	CB92	391	RES 2,D
01C6	CB47	316	BIT 0,A	025E	CB93	392	RES 2,E
01C8	CB48	317	BIT 1,B	0260	CB94	393	RES 2,H
01CA	CB49	318	BIT 1,C	0262	CB95	394	RES 2,L
01CC	CB4A	319	BIT 1,D	0264	CB96	395	RES 2,(HL)
01CE	CB4B	320	BIT 1,E	0266	CB97	396	RES 2,A
01D0	CB4C	321	BIT 1,H	0268	CB98	397	RES 3,B
01D2	CB4D	322	BIT 1,L	026A	CB99	398	RES 3,C
01D4	CB4E	323	BIT 1,(HL)	026C	CB9A	399	RES 3,D
01D6	CB4F	324	BIT 1,A	026E	CB9B	400	RES 3,E
01D8	CB50	325	BIT 2,B	0270	CB9C	401	RES 3,H
01DA	CB51	326	BIT 2,C	0272	CB9D	402	RES 3,L
01DC	CB52	327	BIT 2,D	0274	CB9E	403	RES 3,(HL)
01DE	CB53	328	BIT 2,E	0276	CB9F	404	RES 3,A
01E0	CB54	329	BIT 2,H	0278	CBAA	405	RES 4,B
01E2	CB55	330	BIT 2,L	027A	CBAB	406	RES 4,C
01E4	CB56	331	BIT 2,(HL)	027C	CBAC	407	RES 4,D
01E6	CB57	332	BIT 2,A	027E	CBAD	408	RES 4,E
01E8	CB58	333	BIT 3,B	0280	CBAA	409	RES 4,H
01EA	CB59	334	BIT 3,C	0282	CBAB	410	RES 4,L
01EC	CB5A	335	BIT 3,D	0284	CBAC	411	RES 4,(HL)
01EE	CB5B	336	BIT 3,E	0286	CBAD	412	RES 4,A
01F0	CB5C	337	BIT 3,H	0288	CBAB	413	RES 5,B
01F2	CB5D	338	BIT 3,L	028A	CBAC	414	RES 5,C
01F4	CB5E	339	BIT 3,(HL)	028C	CBAA	415	RES 5,D
01F6	CB5F	340	BIT 3,A	028E	CBAB	416	RES 5,E
01F8	CB60	341	BIT 4,B	0290	CBAC	417	RES 5,H
01FA	CB61	342	BIT 4,C	0292	CBAD	418	RES 5,L
01FC	CB62	343	BIT 4,D	0294	CBAE	419	RES 5,(HL)
01FE	CB63	344	BIT 4,E	0296	CBAF	420	RES 5,A
0200	CB64	345	BIT 4,H	0298	CB80	421	RES 6,B
0202	CB65	346	BIT 4,L	029A	CB81	422	RES 6,C
0204	CB66	347	BIT 4,(HL)	029C	CB82	423	RES 6,D
0206	CB67	348	BIT 4,A	029E	CB83	424	RES 6,E
0208	CB68	349	BIT 5,B	02A0	CB84	425	RES 6,H
020A	CB69	350	BIT 5,C	02A2	CB85	426	RES 6,L
020C	CB6A	351	BIT 5,D	02A4	CB86	427	RES 6,(HL)
020E	CB6B	352	BIT 5,E	02A6	CB87	428	RES 6,A
0210	CB6C	353	BIT 5,H	02A8	CB88	429	RES 7,B
0212	CB6D	354	BIT 5,L	02AA	CB89	430	RES 7,C
0214	CB6E	355	BIT 5,(HL)	02AC	CB8A	431	RES 7,D
0216	CB6F	356	BIT 5,A	02AE	CB8B	432	RES 7,E
0218	CB70	357	BIT 6,B	02B0	CB8C	433	RES 7,H
021A	CB71	358	BIT 6,C	02B2	CB8D	434	RES 7,L
021C	CB72	359	BIT 6,D	02B4	CB8E	435	RES 7,(HL)
021E	CB73	360	BIT 6,E	02B6	CB8F	436	RES 7,A
0220	CB74	361	BIT 6,H	02B8	CB80	437	SET 0,B
0222	CB75	362	BIT 6,L	02BA	CB81	438	SET 0,C
0224	CB76	363	BIT 6,(HL)	02BC	CB82	439	SET 0,D
0226	CB77	364	BIT 6,A	02BE	CB83	440	SET 0,E
0228	CB78	365	BIT 7,B	02C0	CB84	441	SET 0,H
022A	CB79	366	BIT 7,C	02C2	CB85	442	SET 0,L
022C	CB7A	367	BIT 7,D	02C4	CB86	443	SET 0,(HL)
022E	CB7B	368	BIT 7,E	02C6	CB87	444	SET 0,A

02C8	CBC8	445	SET 1,B	036F	DD7105	520	LD (IX+IND),C
02CA	BC9	446	SET 1,C	0372	DD7205	521	LD (IX+IND),D
02CC	CBCA	447	SET 1,D	0375	DD7305	522	LD (IX+IND),E
02CE	CBCB	448	SET 1,E	0378	DD7405	523	LD (IX+IND),H
02D0	CBCC	449	SET 1,H	037B	DD7505	524	LD (IX+IND),L
02D2	CBDD	450	SET 1,L	037E	DD7705	525	LD (IX+IND),A
02D4	CBCE	451	SET 1,(HL)	0381	DD7E05	526	LD A,(IX+IND)
02D6	CBCE	452	SET 1,A	0384	DD8605	527	ADD A,(IX+IND)
02D8	CBDD	453	SET 2,B	0387	DD8E05	528	ADC A,(IX+IND)
02DA	CBDD	454	SET 2,C	038A	DD9605	529	SUB (IX+IND)
02DC	CBDD	455	SET 2,D	038D	DD9E05	530	SBC A,(IX+IND)
02DE	CBDD	456	SET 2,E	0390	DDA605	531	AND (IX+IND)
02E0	CBDD	457	SET 2,H	0393	DDAE05	532	XOR (IX+IND)
02E2	CBDD	458	SET 2,L	0396	DDBE05	533	OR (IX+IND)
02E4	CBDD	459	SET 2,(HL)	0399	DDBE05	534	CP (IX+IND)
02E6	CBDD	460	SET 2,A	039C	DDE1	535	POP IX
02E8	CBDD	461	SET 2,B	039E	DDE3	536	EX (SP),IX
02EA	CBDD	462	SET 2,C	03A0	DDE5	537	PUSH IX
02EC	CBDA	463	SET 2,D	03A2	DDE9	538	JP (IX)
02EE	CBDB	464	SET 2,E	03A4	DDF9	539	LD SP,IX
02F0	CBDD	465	SET 2,H	03A6	DDC80506	540	RLC (IX+IND)
02F2	CBDD	466	SET 2,L	03AA	DDC80506	541	RRC (IX+IND)
02F4	CBDE	467	SET 2,(HL)	03AE	DDC80516	542	RL (IX+IND)
02F6	CBDF	468	SET 2,A	03B2	DDC8051E	543	RR (IX+IND)
02F8	CBDD	469	SET 2,B	03B6	DDC80526	544	SLA (IX+IND)
02FA	CBDD	470	SET 2,C	03BA	DDC8052E	545	SRA (IX+IND)
02FC	CBDD	471	SET 2,D	03BE	DDC8053E	546	SRL (IX+IND)
02FE	CBDD	472	SET 2,E	03C2	DDC80546	547	BIT 0,(IX+IND)
0300	CBDD	473	SET 2,H	03C6	DDC8054E	548	BIT 1,(IX+IND)
0302	CBDD	474	SET 2,L	03CA	DDC80556	549	BIT 2,(IX+IND)
0304	CBDE	475	SET 2,(HL)	03CE	DDC8055E	550	BIT 3,(IX+IND)
0306	CBDE	476	SET 2,A	03D2	DDC80566	551	BIT 4,(IX+IND)
0308	CBDE	477	SET 2,B	03D6	DDC8056E	552	BIT 5,(IX+IND)
030A	CBDE	478	SET 2,C	03DA	DDC80576	553	BIT 6,(IX+IND)
030C	CBDE	479	SET 2,D	03DE	DDC8057E	554	BIT 7,(IX+IND)
030E	CBDE	480	SET 2,E	03E2	DDC80586	555	RES 0,(IX+IND)
0310	CBDE	481	SET 2,H	03E6	DDC8058E	556	RES 1,(IX+IND)
0312	CBDD	482	SET 2,L	03EA	DDC80596	557	RES 2,(IX+IND)
0314	CBDE	483	SET 2,(HL)	03EE	DDC8059E	558	RES 3,(IX+IND)
0316	CBDE	484	SET 2,A	03F2	DDC805A6	559	RES 4,(IX+IND)
0318	CBDD	485	SET 2,B	03F6	DDC805AE	560	RES 5,(IX+IND)
031A	CBDD	486	SET 2,C	03FA	DDC805B6	561	RES 6,(IX+IND)
031C	CBDD	487	SET 2,D	03FE	DDC805BE	562	RES 7,(IX+IND)
031E	CBDD	488	SET 2,E	0402	DDC805C6	563	SET 0,(IX+IND)
0320	CBDD	489	SET 2,H	0406	DDC805CE	564	SET 1,(IX+IND)
0322	CBDD	490	SET 2,L	040A	DDC805D6	565	SET 2,(IX+IND)
0324	CBDD	491	SET 2,(HL)	040E	DDC805DE	566	SET 3,(IX+IND)
0326	CBDD	492	SET 2,A	0412	DDC805E6	567	SET 4,(IX+IND)
0328	CBDD	493	SET 2,B	0416	DDC805EE	568	SET 5,(IX+IND)
032A	CBDD	494	SET 2,C	041A	DDC805F6	569	SET 6,(IX+IND)
032C	CBDD	495	SET 2,D	041E	DDC805FE	570	SET 7,(IX+IND)
032E	CBDD	496	SET 2,E	0422	ED40	571	IN B,(C)
0330	CBDD	497	SET 2,H	0424	ED41	572	OUT (C),B
0332	CBDD	498	SET 2,L	0426	ED42	573	SBC HL,BC
0334	CBDD	499	SET 2,(HL)	0428	ED438405	574	LD (NN),BC
0336	CBDD	500	SET 2,A	042C	ED44	575	NEG
0338	DD09	501	ADD IX,BC	042E	ED45	576	RTN
033A	DD19	502	ADD IX,DE	0430	ED46	577	IM 0
033C	DD218405	503	LD IX,NN	0432	ED47	578	LD 1,A
0340	DD228405	504	LD (NN),IX	0434	ED48	579	IN C,(C)
0344	DD23	505	INC IX	0436	ED49	580	OUT (C),C
0346	DD29	506	ADD IX,IX	0438	ED4A	581	ADC HL,BC
0348	DD2A8405	507	LD IX,(NN)	043A	ED4B8405	582	LD BC,(NN)
034C	DD2B	508	DEC IX	043E	ED4D	583	RET
034E	DD3405	509	INC (IX+IND)	0440	ED50	584	IN D,(C)
0351	DD3505	510	DEC (IX+IND)	0442	ED51	585	OUT (C),D
0354	DD360520	511	LD (IX+IND),N	0444	ED52	586	SBC HL,DE
0358	DD39	512	ADD IX,SP	0446	ED538405	587	LD (NN),DE
035A	DD4605	513	LD B,(IX+IND)	044A	ED56	588	IM 1
035D	DD4E05	514	LD C,(IX+IND)	044C	ED57	589	LD A,1
0360	DD5605	515	LD D,(IX+IND)	044E	ED58	590	IN E,(C)
0363	DD5E05	516	LD E,(IX+IND)	0450	ED59	591	OUT (C),E
0366	DD6605	517	LD H,(IX+IND)	0452	ED5A	592	ADC HL,DE
0369	DD6E05	518	LD L,(IX+IND)	0454	ED5B8405	593	LD DE,(NN)
036C	DD7005	519	LD (IX+IND),B	0458	ED5E	594	IM 2

045A	ED60	595	IN H.(C)	0520	FDCB053E	670	SRL (Y+IND)
045C	ED61	596	OUT (C),H	0524	FDCB054E	671	BIT 0.(Y+IND)
045E	ED62	597	SBC HL,HI	0528	FDCB055E	672	BIT 1.(Y+IND)
0460	ED67	598	RRD	0532	FDCB056E	673	BIT 2.(Y+IND)
0462	ED68	599	IN L.(C)	0530	FDCB055E	674	BIT 3.(Y+IND)
0464	ED69	600	OUT (C),L	0534	FDCB056E	675	BIT 4.(Y+IND)
0466	ED6A	601	ADC HL,HL	0538	FDCB056E	676	BIT 5.(Y+IND)
0468	ED6F	602	RLD	053C	FDCB057E	677	BIT 6.(Y+IND)
046A	ED72	603	SBC HL,SP	0540	FDCB057E	678	BIT 7.(Y+IND)
046C	ED738405	604	LD (NN),SP	0544	FDCB058E	679	RES 0.(Y+IND)
0470	ED7E	605	IN A.(C)	0548	FDCB058E	680	RES 1.(Y+IND)
0472	ED79	606	OUT (C),A	054C	FDCB059E	681	RES 2.(Y+IND)
0474	ED7A	607	ADC HL,SP	0550	FDCB059E	682	RES 3.(Y+IND)
0476	ED7B8405	608	LD SP, (NN)	0554	FDCB05A6	683	RES 4.(Y+IND)
047A	EDA0	609	LDI	0558	FDCB05A6	684	RES 5.(Y+IND)
047C	EDA1	610	CPI	055C	FDCB05B6	685	RES 6.(Y+IND)
047E	EDA2	611	INJ	0560	FDCB05BE	686	RES 7.(Y+IND)
0480	EDA3	612	OUTI	0564	FDCB05C6	687	SET 0.(Y+IND)
0482	EDA8	613	LDD	0568	FDCB05CE	688	SET 1.(Y+IND)
0484	EDA9	614	CPD	056C	FDCB05D6	689	SET 2.(Y+IND)
0486	EDA A	615	IND	0570	FDCB05DE	690	SET 3.(Y+IND)
0488	EDAB	616	OUTD	0574	FDCB05E6	691	SET 4.(Y+IND)
048A	EDB0	617	LDIR	0578	FDCB05EE	692	SET 5.(Y+IND)
048C	EDB1	618	CPIR	057C	FDCB05FE	693	SET 6.(Y+IND)
048E	EDB2	619	INIR	0580	FDCB05FE	694	SET 7.(Y+IND)
0490	EDB3	620	OTIR	0584		695 NK	DEFS 2
0492	EDB8	621	LDDR			696 IND	EQU 5
0494	EDB9	622	CPDR			697 M	EQU 10H
0496	EDBA	623	INDR			698 N	EQU 20H
0498	EDBB	624	OTDR			699 DIS	EQU 30H
049A	FD09	625	ADD IY,BC			700	END
049C	FD19	626	ADD IY,DE				
049E	FD218405	627	LD IY,NN				
04A2	FD228405	628	LD (NN),IY				
04A6	FD23	629	INC IY				
04A8	FD29	630	ADD IY,IY				
04AA	FD2A8405	631	LD IY,(NN)				
04AE	FD2B	632	DEC IY				
04B0	FD3405	633	INC (Y+IND)				
04B3	FD3505	634	DEC (Y+IND)				
04B6	FD360520	635	LD (Y+IND),N				
04BA	FD39	636	ADD IY,SP				
04BC	FD4605	637	LD B,(Y+IND)				
04BF	FD4E05	638	LD C,(Y+IND)				
04C2	FD5605	639	LD D,(Y+IND)				
04C5	FD5E05	640	LD E,(Y+IND)				
04C8	FD6605	641	LD H,(Y+IND)				
04CB	FD6E05	642	LD L,(Y+IND)				
04CE	FD7005	643	LD (Y+IND),B				
04D1	FD7105	644	LD (Y+IND),C				
04D4	FD7205	645	LD(Y+IND),D				
04D7	FD7305	646	LD (Y+IND),E				
04DA	FD7405	647	LD (Y+IND),H				
04DD	FD7505	648	LD (Y+IND),L				
04E0	FD7705	649	LD (Y+IND),A				
04E3	FD7E05	650	LD A,(Y+IND)				
04E6	FD8605	651	ADD A,(Y+IND)				
04E9	FD8E05	652	ADC A,(Y+IND)				
04EC	FD9605	653	SUB (Y+IND)				
04EF	FD9E05	654	SBC A,(Y+IND)				
04F2	FDA605	655	AND (Y+IND)				
04F5	FDAE05	656	XOR (Y+IND)				
04F8	FDB605	657	OR (Y+IND)				
04FB	FDBE05	658	CP (Y+IND)				
04FE	FDE1	659	POP IY				
0500	FDE3	660	EX (SP),IY				
0502	FDE5	661	PUSH IY				
0504	FDE9	662	JP (Y)				
0506	FDF9	663	LD SP,IY				
0508	FDCB0506	664	RLC (Y+IND)				
050C	FDCB050E	665	RRC (Y+IND)				
0510	FDCB0516	666	RL (Y+IND)				
0514	FDCB051E	667	RR (Y+IND)				
0518	FDCB0526	668	SLA (Y+IND)				
051C	FDCB052E	669	SRA (Y+IND)				

ANNEXE 2

Liste des codes opérations par ordre alphabétique

0000	8E	1	ADC	A, (HL)	0056	CB50	74	BIT	2, B
0001	DD8E05	2	ADC	A, (IX+IND)	005A	CB51	75	BIT	2, C
0004	FD8E05	3	ADC	A, (IY+IND)	005C	CB52	76	BIT	2, D
0007	8F	4	ADC	A, A	005E	CB53	77	BIT	2, E
0008	88	5	ADC	A, B	0059	CB54	78	BIT	2, H
0009	89	6	ADC	A, C	0052	CB55	79	BIT	2, L
000A	8A	7	ADC	A, D	0054	CB5E	80	BIT	3, (HL)
000B	8B	8	ADC	A, E	0056	DDXB055E	81	BIT	3, (IX+IND)
000C	8C	9	ADC	A, H	005A	FDCB055E	82	BIT	3, (IY+IND)
000D	8D	10	ADC	A, L	005E	CB5F	83	BIT	3, A
000E	CE20	11	ADC	A, N	00A0	CB58	84	BIT	3, B
0010	ED4A	12	ADC	HL, BC	00A2	CB59	85	BIT	3, C
0012	ED5A	13	ADC	HL, DE	00A4	CB5A	86	BIT	3, D
0014	ED6A	14	ADC	HL, HL	00A6	CB5B	87	BIT	3, E
0016	ED7A	15	ADC	HL, SP	00A8	CB5C	88	BIT	3, H
0018	86	16	ADD	A, (HL)	00AA	CB5D	89	BIT	3, L
0019	DD8605	17	ADD	A, (IX+IND)	00AC	CB66	90	BIT	4, (HL)
001C	FD8605	18	ADD	A, (IY+IND)	00AE	DDCB0566	91	BIT	4, (IX+IND)
001F	87	19	ADD	A, A	00B2	FDCB0566	92	BIT	4, (IY+IND)
0020	80	20	ADD	A, B	00B6	CB67	93	BIT	4, A
0021	81	21	ADD	A, C	00B8	CB60	94	BIT	4, B
0022	82	22	ADD	A, D	00BA	CB61	95	BIT	4, C
0023	83	23	ADD	A, E	00BC	CB62	96	BIT	4, D
0024	84	24	ADD	A, H	00BE	CB63	97	BIT	4, E
0025	85	25	ADD	A, L	00C0	CB64	98	BIT	4, H
0026	C620	26	ADD	A, N	00C2	CB65	99	BIT	4, L
0028	09	27	ADD	HL, BC	00C4	CB6E	100	BIT	5, (HL)
0029	19	28	ADD	HL, DE	00C6	DDCB056E	101	BIT	5, (IX+IND)
002A	29	29	ADD	HL, HL	00CA	FDCB056E	102	BIT	5, (IY+IND)
002B	39	30	ADD	HL, SP	00CE	CB6F	103	BIT	5, A
002C	DD09	31	ADD	IX, BC	00D0	CB68	104	BIT	5, B
002E	DD19	32	ADD	IX, DE	00D2	CB69	105	BIT	5, C
0030	DD29	33	ADD	IX, IX	00D4	CB6A	106	BIT	5, D
0032	DD39	34	ADD	IX, SP	00D6	CB6B	107	BIT	5, E
0034	FD09	35	ADD	IY, BC	00D8	CB6C	108	BIT	5, H
0036	FD19	36	ADD	IY, DE	00DA	CB6D	109	BIT	5, L
0038	FD29	37	ADD	IY, IY	00DC	CB76	110	BIT	6, (HL)
003A	FD39	38	ADD	IY, SP	00DE	DDCB0576	111	BIT	6, (IX+IND)
003C	A6	39	AND	(HL)	00E2	FDCB0576	112	BIT	6, (IY+IND)
003D	DDA605	40	AND	(IX+IND)	00E6	CB77	113	BIT	6, A
0040	FDA605	41	AND	(IY+IND)	00E8	CB70	114	BIT	6, B
0043	A7	42	AND	A	00EA	CB71	115	BIT	6, C
0044	A0	43	AND	B	00EC	CB72	116	BIT	6, D
0045	A1	44	AND	C	00EE	CB73	117	BIT	6, E
0046	A2	45	AND	D	00F0	CB74	118	BIT	6, H
0047	A3	46	AND	E	00F2	CB75	119	BIT	6, L
0048	A4	47	AND	H	00F4	CB7E	120	BIT	7, (HL)
0049	A5	48	AND	L	00F6	DDCB057E	121	BIT	7, (IX+IND)
004A	E620	49	AND	N	00FA	FDCB057E	122	BIT	7, (IY+IND)
004C	CB46	50	BIT	O, (HL)	00FE	CB7F	123	BIT	7, A
004E	DDCB0546	51	BIT	O, (IX+IND)	0100	CB78	124	BIT	7, B
0052	FDBC0546	52	BIT	O, (IY+IND)	0102	CB79	125	BIT	7, C
0056	CB47	53	BIT	O, A	0104	CB7A	126	BIT	7, D
0058	CB40	54	BIT	O, B	0106	CB7B	127	BIT	7, E
005A	CB41	55	BIT	O, C	0108	CB7C	128	BIT	7, H
005C	CB42	56	BIT	O, D	010A	CB7D	129	BIT	7, L
005E	CB43	57	BIT	O, E	010C	DC8405	130	CALL	C, NN
0060	CB44	58	BIT	O, H	010F	FC8405	131	CALL	M, NN
0062	CB45	59	BIT	O, L	0112	D48405	132	CALL	NC, NN
0064	CB4E	60	BIT	1, (HL)	0115	CD8405	133	CALL	NN
0066	DDCB054E	61	BIT	1, (IX+IND)	0118	C48405	134	CALL	NZ, NN
006A	FDCB054E	62	BIT	1, (IY+IND)	011B	F48405	135	CALL	P, NN
006E	CB4F	63	BIT	1, A	011E	EC8405	136	CALL	PE, NN
0070	CB48	64	BIT	1, B	0121	E48405	137	CALL	PO, NN
0072	CB49	65	BIT	1, C	0124	CC8405	138	CALL	Z, NN
0074	CB4A	66	BIT	1, D	0127	3F	139	CCF	
0076	CB4B	67	BIT	1, E	0128	BE	140	CP	(HL)
0078	CB4C	68	BIT	1, H	0129	DDBE05	141	CP	(IX+IND)
007A	CB4D	69	BIT	1, L	012C	FDBE05	142	CP	(IY+IND)
007C	CB56	70	BIT	2, (HL)	012F	BF	143	CP	A
007E	DDCB0556	71	BIT	2, (IX+IND)	0130	B8	144	CP	B
0082	FDCB0556	72	BIT	2, (IY+IND)	0131	B9	145	CP	C
0086	CB57	73	BIT	2, A	0132	BA	146	CP	D

0133	BB	147	CP	E	01AD	F28405	222	JP	P, NN
0134	BC	148	CP	H	01B0	EA6405	223	JP	PE, NN
0135	BD	149	CP	L	01B3	E28405	224	JP	PO, NN
0136	FE20	150	CP	N	01B6	CA6405	225	JP	Z, NN
0138	EDA9	151	CPD		01B9	382E	226	JR	C, DIS
013A	EDB9	152	CPDR		01BB	182E	227	JR	DIS
013C	EDA1	153	CPI		01BD	302E	228	JR	NC, DIS
013E	EDB1	154	CP1R		01BF	202E	229	JR	NZ, DIS
0140	2F	155	CPL		01C1	282E	230	JR	Z, DIS
0141	27	156	DAA		01C3	02	231	LD	(BC), A
0142	35	157	DEC	(HL)	01C4	12	232	LD	(DE), A
0143	DD3505	158	DEC	(IX+IND)	01C5	77	233	LD	(HL), A
0146	FD3505	159	DEC	(IY+IND)	01C6	70	234	LD	(HL), B
0149	3D	160	DEC	A	01C7	71	235	LD	(HL), C
014A	05	161	DEC	B	01C8	72	236	LD	(HL), D
014B	0B	162	DEC	BC	01C9	73	237	LD	(HL), E
014C	0D	163	DEC	C	01CA	74	238	LD	(HL), H
014D	15	164	DEC	D	01CB	75	239	LD	(HL), L
014E	1B	165	DEC	DE	01CC	3620	240	LD	(HL), N
014F	1D	166	DEC	E	01CE	DD7705	241	LD	(IX+IND), A
0150	25	167	DEC	H	01D1	DD7005	242	LD	(IX+IND), B
0151	2B	168	DEC	HL	01D4	DD7105	243	LD	(IX+IND), C
0152	DD2B	169	DEC	IX	01D7	DD7205	244	LD	(IX+IND), D
0154	FD2B	170	DEC	IY	01DA	DD7305	245	LD	(IX+IND), E
0156	2D	171	DEC	L	01DD	DD7405	246	LD	(IX+IND), H
0157	3B	172	DEC	SP	01E0	DD7505	247	LD	(IX+IND), L
0158	F3	173	DI		01E3	DD360520	248	LD	(IX+IND), N
0159	102E	**4	DJNZ	DIS	01E7	FD7705	249	LD	(IY+IND), A
015B	FB	175	EI		01EA	FD7005	250	LD	(IY+IND), B
015C	E3	176	EX	(SP), HL	01ED	FD7105	251	LD	(IY+IND), C
015D	DDE3	177	EX	(SP), IX	01F0	FD7205	252	LD	(IY+IND), D
015F	FDE3	178	EX	(SP), IY	01F3	FD7305	253	LD	(IY+IND), E
0161	08	179	EX	AF, AF'	01F6	FD7405	254	LD	(IY+IND), H
0162	EB	180	EX	DE, HL	01F9	FD7505	255	LD	(IY+IND), L
0163	D9	181	EXX		01FC	FD360520	256	LD	(IY+IND), N
0164	76	182	HALT		0200	328405	257	LD	(NN), A
0165	ED46	183	IM	0	0203	ED436405	258	LD	(NN), BC
0167	ED56	184	IM	1	0207	ED538405	259	LD	(NN), DE
0169	ED5E	185	IM	2	020B	228405	260	LD	(NN), HL
016B	ED78	186	IN	A, (C)	020E	DD228405	261	LD	(NN), IX
016D	DB20	187	IN	A, N	0212	FD228405	262	LD	(NN), IY
016F	ED40	188	IN	B, (C)	0216	ED738405	263	LD	(NN), SP
0171	ED48	189	IN	C, (C)	021A	0A	264	LD	A, (BC)
0173	ED50	190	IN	D, (C)	021B	1A	265	LD	A, (DE)
0175	ED58	191	IN	E, (C)	021C	7E	266	LD	A, (HL)
0177	ED60	192	IN	H, (C)	021D	DD7E05	267	LD	A, (IX+IND)
0179	ED68	193	IN	L, (C)	0220	FD7E05	268	LD	A, (IY+IND)
017B	34	194	INC	(HL)	0223	3A8405	269	LD	A, (NN)
017C	DD3405	195	INC	(IX+IND)	0226	7F	270	LD	A, A
017F	FD3405	196	INC	(IY+IND)	0227	78	271	LD	A, B
0182	3C	197	INC	A	0228	79	272	LD	A, C
0183	04	198	INC	B	0229	7A	273	LD	A, D
0184	03	199	INC	BC	022A	7B	274	LD	A, E
0185	0C	200	INC	C	022B	7C	275	LD	A, H
0186	14	201	INC	D	022C	ED57	276	LD	A, I
0187	13	202	INC	DE	022E	7D	277	LD	A, L
0188	1C	203	INC	E	022F	3E20	278	LD	A, N
0189	24	204	INC	H	0231	46	279	LD	B, (HL)
018A	23	205	INC	HL	0232	DD4605	280	LD	B, (IX+IND)
018B	DD23	206	INC	IX	0235	FD4605	281	LD	B, (IY+IND)
018D	FD23	207	INC	IY	0238	47	282	LD	B, A
018F	2C	208	INC	L	0239	40	283	LD	B, B
0190	33	209	INC	SP	023A	41	284	LD	B, C
0191	EDAA	210	IND		023B	42	285	LD	B, E
0193	EDBA	211	INDR		023C	43	286	LD	B, D
0195	EDA2	212	INI		023D	44	287	LD	B, H, NN
0197	EDB2	213	INIR		023E	45	288	LD	B, I
0199	E9	214	JP	(HL)	023F	0620	289	LD	B, N
019A	DDE9	215	JP	(IX)	0241	ED4B6405	290	LD	BC, (NN)
019C	FDE9	216	JP	(IY)	0245	018405	291	LD	BC, NN
019E	DA8405	217	JP	C, NN	0248	4E	292	LD	C, (HL)
01A1	FA8405	218	JP	M, NN	0249	DD4E05	293	LD	C, (IX+IND)
01A4	D28405	219	JP	NC, NN	024C	FD4E05	294	LD	C, (IY+IND)
01A7	C38405	220	JP	NN	024F	4F	295	LD	C, A
01AA	C28405	221	JP	NZ, NN	0250	48	296	LD	C, B

0251	49	297	LD	C, C	02D5	B7	373	OR	D
0252	4A	296	LD	C, D	02D9	B3	374	OR	E
0253	4B	299	LD	C, E	02DA	B4	375	OR	H
0254	4C	300	LD	C, H	02DB	B5	376	OR	L
0255	4D	301	LD	C, L	02DC	F620	377	OR	N
0256	0E20	302	LD	C, N	02DE	EDB8	378	OTDR	
0258	56	303	LD	D, (HL)	02E0	EDB3	379	OTIR	
0259	DD5605	304	LD	D, (IX+IND)	02E2	ED79	380	OUT	(C),A
025C	FD5605	305	LD	D, (IY+IND)	02E4	ED41	381	OUT	(C),B
025F	57	306	LD	D, A	02E6	ED49	382	OUT	(C),C
0260	50	307	LD	D, B	02E8	ED51	383	OUT	(C),D
0261	51	308	LD	D, C	02EA	ED59	384	OUT	(C),E
0262	52	309	LD	D, D	02EC	ED61	385	OUT	(C),H
0263	53	310	LD	D, E	02EE	ED69	386	OUT	(C),L
0264	54	311	LD	D, H	02F0	D520	387	OUT	N.A
0265	55	312	LD	D, L	02F2	EDA8	388	OUTD	
0266	1620	313	LD	D, N	02F4	EDA2	389	OUTI	
0268	ED5B8405	314	LD	DE, (NN)	02F6	F1	390	POP	AF
026C	118405	315	LD	DE, NN	02F7	C1	391	POP	BC
026F	5E	316	LD	E, (HL)	02F8	D1	392	POP	DE
0270	DD5E05	317	LD	E, (IX+IND)	02F9	E1	393	POP	HL
0273	FD5E05	318	LD	E, (IY+IND)	02FA	DDE1	394	POP	IX
0276	5F	319	LD	E, A	02FC	FDE1	395	POP	IY
0277	58	320	LD	E, B	02FE	F5	396	PUSH	AF
0278	59	321	LD	E, C	02FF	C5	397	PUSH	BC
0279	5A	322	LD	E, D	0300	D5	398	PUSH	DE
027A	5B	323	LD	E, E	0301	E5	399	PUSH	HL
027B	5C	324	LD	E, H	0302	DDE5	400	PUSH	IX
027C	5D	325	LD	E, L	0304	FDE5	401	PUSH	IY
027D	1E20	326	LD	E, N	0306	CB86	402	RES	0,(HL)
027F	66	327	LD	H, (HL)	0308	DCB0586	403	RES	0,(IX+IND)
0280	DD6605	328	LD	H, (IX+IND)	030C	FCB0586	404	RES	0,(IY+IND)
0283	FD6605	329	LD	H, (IY+IND)	0310	CB87	405	RES	0,A
0286	67	330	LD	H, A	0312	CB80	406	RES	0,B
0287	60	331	LD	H, B	0314	CB81	407	RES	0,C
0288	61	332	LD	H, C	0316	CB82	408	RES	0,D
0289	62	333	LD	H, D	0318	CB83	409	RES	0,E
028A	63	334	LD	H, E	031A	CB84	410	RES	0,H
028B	64	335	LD	H, H	031C	CB85	411	RES	0,L
028C	65	336	LD	H, L	031E	CB8E	412	RES	1,(HL)
028D	2620	337	LD	H, N	0320	DDCB058F	413	RES	1,(IX+IND)
028F	2A8405	338	LD	HL, (NN)	0324	FCB058E	414	RES	1,(IY+IND)
0292	218405	339	LD	HL, NN	0328	CB8F	415	RES	1,A
0295	ED47	340	LD	I, A	032A	CB88	416	RES	1,B
0297	DD2A8405	341	LD	IX, (NN)	032C	CB89	417	RES	1,C
029B	DD218405	342	LD	IX, NN	032E	CB8A	418	RES	1,D
029F	FD2A8405	343	LD	IY, (NN)	0330	CB8B	419	RES	1,E
02A3	FD218405	344	LD	IY, NN	0332	CB8C	420	RES	1,H
02A7	6E	345	LD	L, (HL)	0334	CB8D	421	RES	1,L
02A8	DD6E05	346	LD	L, (IX+IND)	0336	CB96	422	RES	2,(HL)
02AB	FD6E05	347	LD	L, (IY+IND)	0338	DCB0596	423	RES	2,(IX+IND)
02AE	6F	348	LD	L, A	033C	FCB0596	424	RES	2,(IY+IND)
02AF	68	349	LD	L, B	0340	CB97	425	RES	2,A
02B0	69	350	LD	L, C	0342	CB90	426	RES	2,B
02B1	6A	351	LD	L, D	0344	CB91	427	RES	2,C
02B2	6B	352	LD	L, E	0346	CB92	428	RES	2,D
02B3	6C	353	LD	L, H	0348	CB93	429	RES	2,E
02B4	6D	354	LD	L, L	034A	CB94	430	RES	2,H
02B5	2E20	355	LD	L, N	034C	CB95	431	RES	2,L
02B7	ED7B8405	356	LD	SP, (NN)	034E	CB9E	432	RES	3,(HL)
02BB	F9	357	LD	SP, HL	0350	DDCB059E	433	RES	3,(IX+IND)
02BC	DDF9	358	LD	SP, IX	0354	FCB059E	434	RES	3,(IY+IND)
02BE	DDF9	359	LD	SP, IY	0358	CB9F	435	RES	3,A
02C0	318405	360	LD	SP, NN	035A	CB98	436	RES	3,B
02C3	EDA5	361	LDD		035C	CB99	437	RES	3,C
02C5	EDB6	362	LDDR		035E	CB9A	438	RES	3,D
02C7	EDA0	363	LDI		0360	CB9B	439	RES	3,E
02C9	EDB0	364	LDIR		0362	CB9C	440	RES	3,H
02CB	ED44	365	NEG		0364	CB9D	441	RES	3,L
02CD	00	366	NOP		0366	CEA6	442	RES	4,(HL)
02CE	B6	367	OR	(HL)	0368	DDCB05A6	443	RES	4,(IX+IND)
02CF	DDR605	368	OR	(IX+IND)	036C	FCB05A6	444	RES	4,(IY+IND)
02D2	FDB605	369	OR	(IY+IND)	0370	CEA7	445	RES	4,A
02D5	B7	370	OR	A	0372	CEA0	446	RES	4,B
02D6	B0	371	OR	B	0374	CEA1	447	RES	4,C
02D7	B1	372	OR	C	0376	CEA2	448	RES	4,D

0378	CBA3	449	RES	4.E	041B	CB1C	524	RR	H
037A	CBA4	450	RES	4.H	041D	CB1D	525	RR	L
037C	CBA5	451	RES	4.L	041F	1F	526	RR	
037E	CBAE	452	RES	5.(HL)	0420	CB0E	527	RR	(HL)
0380	DDCB05AE	453	RES	5.(IX+IND)	0422	DDCB050E	528	RR	(IX+IND)
0384	FDCB05AE	454	RES	5.(IY+IND)	0426	FDCB050E	529	RR	(IY+IND)
0386	CBAF	455	RES	5.A	042A	CB0F	530	RR	A
038A	CBA8	456	RES	5.B	042C	CB08	531	RR	B
038C	CBA9	457	RES	5.C	042E	CB09	532	RR	C
038E	CBA A	458	RES	5.D	0430	CB0A	533	RR	D
0390	CBA B	459	RES	5.E	0432	CB0B	534	RR	E
0392	CBAC	460	RES	5.H	0434	CB0C	535	RR	H
0394	CBAD	461	RES	5.L	0436	CB0D	536	RR	L
0396	CBB6	462	RES	6.(HL)	0438	0F	537	RR	
0398	DDCB05B6	463	RES	6.(IX+IND)	0439	ED67	538	RR	
039C	FDCB05B6	464	RES	6.(IY+IND)	043B	C7	539	RR	
03A0	CBB7	465	RES	6.A	043C	D7	540	RR	
03A2	CBB0	466	RES	6.E	043D	DF	541	RR	
03A4	CBB1	467	RES	6.C	043E	E7	542	RR	
03A6	CBB2	468	RES	6.D	043F	EF	543	RR	
03A8	CBB3	469	RES	6.E	0440	F7	544	RR	
03AA	CBB4	470	RES	6.H	0441	FF	545	RR	
03AC	CBB5	471	RES	6.L	0442	CF	546	RR	
03AE	CBBE	472	RES	7.(HL)	0443	9E	547	RR	
03B0	DDCB05BE	473	RES	7.(IX+IND)	0444	DD9E05	548	RR	
03B4	FDCB05BE	474	RES	7.(IY+IND)	0447	FD9E05	549	RR	
03B8	CBBF	475	RES	7.A	044A	9F	550	RR	
03BA	CBBE	476	RES	7.B	044B	98	551	RR	
03BC	CBB9	477	RES	7.C	044C	99	552	RR	
03BE	CBBA	478	RES	7.D	044D	9A	553	RR	
03C0	CBBB	479	RES	7.E	044E	9B	554	RR	
03C2	CBBC	480	RES	7.H	044F	9C	555	RR	
03C4	CBBD	481	RES	7.L	0450	9D	556	RR	
03C6	C9	482	RET		0451	DE20	557	RR	
03C7	D8	483	RET	C	0453	ED42	558	RR	
03C8	F8	484	RET	M	0455	ED52	559	RR	
03C9	D0	485	RET	NC	0457	ED62	560	RR	
03CA	C0	486	RET	NZ	0459	ED72	561	RR	
03CB	F0	487	RET	P	045B	37	562	RR	
03CC	E8	488	RET	PE	045C	CB06	563	RR	
03CD	E0	489	RET	PO	045E	DDCB05C6	564	RR	
03CE	C8	490	RET	Z	0462	FDCB05C6	565	RR	
03CF	ED4D	491	RETI		0466	CB07	566	RR	
03D1	ED45	492	RETN		0468	CB00	567	RR	
03D3	CB16	493	RL	(HL)	046A	CB01	568	RR	
03D5	DDCB0516	494	RL	(IX+IND)	046C	CB02	569	RR	
03D9	FDCB0516	495	RL	(IY+IND)	046E	CB03	570	RR	
03DD	CB17	496	RL	A	0470	CB04	571	RR	
03DF	CB10	497	RL	B	0472	CB05	572	RR	
03E1	CB11	498	RL	C	0474	CB0E	573	RR	
03E3	CB12	499	RL	D	0476	DDCB05CE	574	RR	
03E5	CB13	500	RL	E	047A	FDCB05CE	575	RR	
03E7	CB14	501	RL	H	047E	CB0F	576	RR	
03E9	CB15	502	RL	L	0480	CB08	577	RR	
03EB	17	503	RLA		0482	CB09	578	RR	
03EC	CB06	504	RLC	(HL)	0484	CB0A	579	RR	
03EE	DDCB0506	505	RLC	(IX+IND)	0486	CB0B	580	RR	
03F7	FDCB0506	506	RLC	(IY+IND)	0488	CB0C	581	RR	
03F6	CB07	507	RLC	A	048A	CB0D	582	RR	
03F8	CB00	508	RLC	B	048C	CB06	583	RR	
03FA	CB01	509	RLC	C	048E	DDCB05D6	584	RR	
03FC	CB02	510	RLC	D	0492	FDCB05D6	585	RR	
03FE	CB03	511	RLC	F	0496	CB07	586	RR	
0400	CB04	512	RLC	H	0498	CB00	587	RR	
0402	CB05	513	RLC	L	049A	CB01	588	RR	
0404	07	514	RLCA		049C	CB02	589	RR	
0405	ED6F	515	RLD		049E	CB03	590	RR	
0407	CB1E	516	RR	(HL)	04A0	CB04	591	RR	
0409	DDCB051E	517	RR	(IX+IND)	04A2	CB05	592	RR	
040D	FDCB051E	518	RR	(IY+IND)	04A4	CB06	593	RR	
0411	CB1F	519	RR	A	04A6	CB0E	594	RR	
0413	CB16	520	RR	B	04A8	DDCB05DE	595	RR	
0415	CB19	521	RR	C	04AC	FDCB05DE	596	RR	
0417	CB1A	522	RR	D	04B0	CBDF	597	RR	
0419	CB1B	523	RR	E	04B2	CB09	598	RR	

0484	CBDA	599	SET	3.D	0566	FD9605	675	SUB	(IX+IND)
0486	CBDB	600	SET	3.E	056B	97	676	SUB	A
0488	CBDC	601	SET	3.H	056C	90	677	SUB	B
048A	CBDD	602	SET	3.L	056D	91	678	SUB	C
048C	CBE6	603	SET	4.(HL)	056E	92	679	SUB	D
048E	DDCB05E6	604	SET	4.(IX+IND)	056F	93	680	SUB	E
04C2	FDCB05E6	605	SET	4.(IX+IND)	0570	94	681	SUB	H
04C6	CBE7	606	SET	4.A	0571	95	682	SUB	L
04C8	CBE0	607	SET	4.B	0572	D620	683	SUB	N
04CA	CBE1	608	SET	4.C	0574	AE	684	XOR	(HL)
04CC	CBE2	609	SET	4.D	0575	DDAE05	685	XOR	(IX+IND)
04CE	CBE3	610	SET	4.E	0578	FDAE05	686	XOR	(IX+IND)
04D0	CBE4	611	SET	4.H	057B	AF	687	XOR	A
04D2	CBE5	612	SET	4.L	057C	A8	688	XOR	B
04D4	CBE6	613	SET	5.(HL)	057D	A9	689	XOR	C
04D6	DDCB05EE	614	SET	5.(IX+IND)	057E	AA	690	XOR	D
04DA	FDCB05EE	615	SET	5.(IX+IND)	057F	AB	691	XOR	E
04DE	CBEF	616	SET	5.A	0580	AC	692	XOR	H
04E0	CBE8	617	SET	5.B	0581	AD	693	XOR	L
04E2	CBE9	618	SET	5.C	0582	EE20	694	XOR	N
04E4	CBEA	619	SET	5.D	0584		695 NN	DEFS	2
04E6	CBEB	620	SET	5.E			696 IND	EQU	5
04E8	CBE C	621	SET	5.H			697 M	EQU	10H
04EA	CBED	622	SET	5.L			698 N	EQU	20H
04EC	CBF6	623	SET	6.(HL)			699 DIS	EQU	30H
04EE	DDCB05F6	624	SET	6.(IX+IND)			700	END	
04F2	FDCB05F6	625	SET	6.(IX+IND)					
04F6	CBF7	626	SET	6.A					
04F8	CBF0	627	SET	6.B					
04FA	CBF1	628	SET	6.C					
04FC	CBF2	629	SET	6.D					
04FE	CBF3	630	SET	6.E					
0500	CBF4	631	SET	6.H					
0502	CBF5	632	SET	6.L					
0504	CBFE	633	SET	7.(HL)					
0506	DDCB05FE	634	SET	7.(IX+IND)					
050A	FDCB05FE	635	SET	7.(IX+IND)					
050E	CBFF	636	SET	7.A					
0510	CBF8	637	SET	7.B					
0512	CBF9	638	SET	7.C					
0514	CBFA	639	SET	7.D					
0516	CBFB	640	SET	7.E					
0518	CBFC	641	SET	7.H					
051A	CBFD	642	SET	7.L					
051C	CB26	643	SLA	(HL)					
051E	DDCB0526	644	SLA	(IX+IND)					
0522	FDCB0526	645	SLA	(IX+IND)					
0526	CB27	646	SLA	A					
0528	CB20	647	SLA	B					
052A	CB21	648	SLA	C					
052C	CB22	649	SLA	D					
052E	CB23	650	SLA	E					
0530	CB24	651	SLA	H					
0532	CB25	652	SLA	L					
0534	CB2E	653	SRA	(HL)					
0536	DDCB052E	654	SRA	(IX+IND)					
053A	FDCB052E	655	SRA	(IX+IND)					
053E	CB2F	656	SRA	(IX+IND)					
0540	CB28	657	SRA	A					
0542	CB29	658	SRA	B					
0544	CB2A	659	SRA	C					
0546	CB2B	660	SRA	D					
0548	CB2C	661	SRA	E					
054A	CB2D	662	SRA	H					
054C	CB3E	663	SRA	L					
054E	DDCB053E	664	SRL	(HL)					
0552	FDCB053E	665	SRL	(IX+IND)					
0556	CB3F	666	SRL	(IX+IND)					
0558	CB38	667	SRL	(IX+IND)					
055A	CB39	668	SRL	A					
055C	CB3A	669	SRL	B					
055E	CB3B	670	SRL	C					
0560	CB3C	671	SRL	D					
0562	CB3D	672	SRL	E					
0564	96	673	SRL	H					
0566	DD9605	674	SRL	L					
			SUB	(HL)					
			SUB	(IX+IND)					

Vous êtes utilisateur d'un micro-ordinateur conçu autour du microprocesseur Z 80 ?

Ce livre vous aidera à faire vos premiers pas à la découverte d'un nouveau langage qui enrichira beaucoup les possibilités de votre machine : L'ASSEMBLEUR.

Vous savez déjà programmer en BASIC ? Alors nous commencerons par ce que vous connaissez et vous montrerons en quoi le langage machine appelé aussi "Assembleur" peut ressembler au BASIC, malgré les apparences.

Nous ne plongerons pas tout de suite dans la fosse aux "LD A (8A4H)", mais aborderons progressivement le jeu d'instruction du Z 80, à l'aide de nombreux exemples !