Chapitre 24

Assemblage et désassemblage

Le langage machine d'un ordinateur, ou plus exactement d'un microprocesseur de nos jours, est le seul que celui-ci comprend. L'interface pour le programmeur est une suite d'octets, représentés en hexadécimal. Ceci n'est absolument pas parlant pour un être humain, qui a besoin d'un langage symbolique.

Il faut savoir passer du langage symbolique au langage machine (on dit **assembler**) pour concevoir les programmes et du langage machine au langage symbolique pour comprendre un programme (conçu par quelqu'un d'autre) présenté en langage machine (on dit **désassembler**).

Nous avons vu comment assembler tout au long de ce livre en présentant les instructions dans un ordre logique. Nous allons nous contenter d'y revenir dans ce chapitre en les présentant dans un autre ordre) et en rappelant le code machine associé.

On peut de même aider au désassemblage en donnant les instructions en langage machine dans l'ordre alphabétique (en fait des entiers) et en rappelant, pour chaque premier octet de code opération, le nombre d'octets nécessaires et sa traduction en langage symbolique. Rappelons-nous que, pour quelques octets, le premier octet de code opération n'est pas suffisant.

24.1 Assemblage

24.1.1 Champs d'une instruction

Notion.- Comme nous l'avons vu, une instruction peut être codée sur un à plusieurs octets, chaque octet représentant un **champ** de l'instruction.

Code opération et opérandes.- Le premier champ, le seul toujours présent, indique la nature de l'opération et s'appelle le **code opération**, ou **opcode** en abrégé. On a quelquefois besoin d'un deuxième octet pour préciser l'opcode. Les autres champs sont les **opérandes**. Nous avons vu également que le code opération peut être précédé d'un **préfixe**. Il peut y avoir jusqu'à quatre octets d'opérandes. Dans le cas maximal d'une instruction sur sept octets, on a :

```
| préfixe | opcode 1 | opcode 2 | opérande 1b | opérande 1h | opérande 2b | opérande 2h |
```

Remarque.- Bien évidemment, l'ensemble des préfixes doit être disjoint de l'ensemble des opcodes (premier octet d'opcode en tous cas) pour que l'instruction ne soit pas ambiguë.

Exemples d'opcodes.- Nous avons vu de nombreux exemples d'opcodes précédemment. On peut remarquer que l'opcode pour MOV AX, à savoir B8h, n'est pas le même que l'opcode pour MOV BX, à savoir 89h. Ceci n'est pas naturel *a priori* mais cette façon de coder les instructions symboliques permet d'economiser un opérande et donc un octet.

On aurait pu choisir des opcodes au hasard. Ceci n'est pas le cas : les premiers bits (au sens de la lecture de gauche à droite, c'est-à-dire ceux de plus grand poids) de l'opcode désignent réellement la nature de l'opération (au sens du langage symbolique) alors que les derniers bits peuvent faire office de premier opérande.

24.1.2 Instructions sans opérande

Les instructions sans opérande sont codées sur un ou deux octets.

Le tableau suivant donne l'octet d'opcode, et seul octet, des instructions sans opérande codées sur un octet :

Instruction	Opcode (binaire)	Opcode (hexa)
AAA	0011 0111	37
AAS	0011 1111	3F
CBW	1001 1000	98
CLC	1111 1000	F8
CLD	1111 1100	FC
CLI	1111 1010	FA
CMC	1111 0101	F5
CMPSB	1010 0110	96
CMPSW	1010 0111	97
CWD	1001 1001	99
DAA	0010 0111	27
DAS	0010 1111	2F
HLT	1111 0100	F4
INT 3	1100 1100	CC
INTO	1100 1110	CE
IRET	1100 1111	CF

LAHF	1001 1111	9F
LOCK	1111 0000	F0
LODSB	1010 1100	AC
LODSW	1010 1101	AD
MOVSB	1010 0100	A4
MOVSW	1010 0101	A5
NOP	1001 0000	90
POPF	1001 1101	9D
PUSHF	1001 1100	9C
REP/REPNE/REPNZ	1111 0010	F2
REPC	0110 0101	65
REPE/REPZ	1111 0011	F3
REPNC	0110 0100	64
RETN	1100 0011	C3
RETF	1100 1011	CB
SAHF	1001 1110	9E
SCASB	1010 1110	9E
SCASW	1010 1111	9F
STC	1111 1001	F9
STD	1111 1101	FD
STI	1111 1011	FB
STOSB	1010 1010	AA
STOSW	1010 1011	AB
WAIT	1001 1011	9B
XLAT	1101 0111	C7

On remarquera que l'instruction `INT 3` est considérée comme une instruction sans opérande.

Le tableau ci-dessous donne les deux octets des instructions sans opérande codées sur deux octets :

Instruction	Opcode binaire	Opcode hexa
AAD	11010101 00001010	D3 09
AAM	11010100 00001010	D4 09

24.1.3 Instructions avec un seul opérande

24.1.3.1 Forme d'une telle instruction

Certaines instructions ont un seul opérande, par exemple `inc`, `dec`, `mul`, `div` et `not`. Elles sont codées sur un octet ou plusieurs octets suivant que l'opérande est un registre de seize bits ou d'un autre type (registre de huit bits ou emplacement mémoire).

24.1.3.2 Cas d'un opérande registre général de seize bits

Format.- Dans le cas où l'opérande est un registre de seize bits, le code n'occupe qu'un seul octet. Le format est dans ce cas :

```
| opcode reg |
```

où l'opcode proprement dit n'occupe que les cinq premiers bits (de poids fort) de l'octet et la désignation du registre occupe les trois derniers bits (donc les bits de poids faible).

L'octet d'opcode est donc en fait décomposé en deux champs : un opcode proprement dit et un opérande. Ceci permet évidemment d'obtenir un code machine plus compact.

Opcodes courts.- Le tableau suivant donne l'opcode des instructions à un opérande pour le **format court**, c'est-à-dire lorsque l'opérande est un registre de seize bits. L'opcode pour le **format long** sera évidemment différent pour qu'il n'y ait pas ambiguïté.

Instruction	Opcode court	hexa
dec	0100 1	4 1xxx
inc	0100 0	4 0xxx
push	0101 0	5 0xxx
pop	0101 1	5 1xxx
xchg ax,reg	1001 0	9 0xxx

Désignation du registre.- Les huit registres généraux de seize bits sont désignés de la façon suivante :

Registre	Codage binaire
AX	000
BX	011
CX	001
DX	010
SP	100
BP	101
SI	110
DI	111

Exemple.- Le code de :

```
inc bx
```

est :

```
0100 0011
```

soit `43h`.

24.1.3.3 Cas d'un opérande registre de segment

Format.- Dans le cas où l'opérande est un registre de segment, le code n'occupe également qu'un seul octet. Le format est dans ce cas :

```
| opcode1 sreg opcode2 |
```

où la première partie `opcode1` de l'opcode occupe les trois premiers bits, le code `sreg` du registre de segment occupe deux bits et la seconde partie `opcode2` de l'opcode occupe les trois derniers bits.

Opcodes.- Le tableau suivant donne l'opcode des instructions à un opérande segment de registre pour le **format court**.

Instruction	Opcode
push	000 sreg 110
pop	000 sreg 111
seg	001 sreg 110

Désignation du registre de segment.- Les quatre registres de segment sont désignés de la façon suivante :

Registre	Code
CS	01
DS	11
ES	00
SS	10

Exemple.- Le code de :

```
push cs
```

est :

```
0000 1110
```

soit `0Eh`.

24.1.3.4 Cas d'un adressage immédiat sur un octet

Format.- Dans le cas où l'opérande est une constante occupant un octet, le code occupe deux octets suivant le format :

```
| opcode | déplacement |
```

où l'opcode et le déplacement occupent chacun un octet. Le déplacement est alors considéré comme un entier relatif (compris entre - 128 et 127).

Opcodes.- Le tableau suivant donne l'opcode des instructions à un opérande dont celui-ci est un déplacement sur un octet :

Instruction	Opcode	hexa
call (proche direct)	1110 1000	E8
int (sauf int3)	1100 1101	CD
loop	1110 0010	E2
loope/loopz	1110 0001	E1
loopne/loopnz	1110 0000	E0

24.1.3.5 Cas d'un adressage immédiat sur un mot

Format.- Dans le cas où l'opérande est une constante occupant un mot, le code occupe trois octets suivant le format :

```
| opcode | déplacement bas | déplacement haut |
```

où l'opcode occupe un octet et le déplacement deux octets. Le déplacement est alors considéré comme une adresse (entier naturel).

Opcodes.- Le tableau suivant donne l'opcode des instructions à un opérande déplacement sur un mot.

Instruction	Opcode	hexa
retn	1100 0010	C2
retf	1100 1010	CA

24.1.3.6 Cas des sauts

Sauts conditionnels.- Un saut conditionnel autre que `jcxz` se code sur deux octets :

`| 0111 cccc | déplacement |`

dans le cas d'un saut court, où `cccc` est précisé dans le tableau suivant :

Condition	cccc	Opcode	Description (sauf si ...)
JA/JNBE	0111	77	supérieur (entier naturel)
JAE/JNB	0011	73	supérieur ou égal (entier naturel)
JB/JNAE	0010	72	inférieur (entier naturel)
JBE/JNA	0110	76	inférieur ou égal (entier naturel)
JC	0010	72	retenue
JE	0100	74	égal
JG/JNLE	1111	7F	plus grand (entier relatif)
JGE/JNL	1101	7D	plus grand ou égal (entier relatif)
JL/JNGE	1100	7C	plus petit (entier relatif)
JLE/JNG	1110	7E	plus petit ou égal (entier relatif)
JNC	0011	73	pas de retenue
JNE/JNZ	0101	75	différent
JNO	0001	71	pas de débordement
JNP/JPO	1011	7B	impair
JNS	1001	79	pas de signe
JO	0000	70	débordement
JP/JPE	1010	7A	pair
JS	1000	78	signe
JZ	0100	74	zéro

et sur quatre octets :

`| 0000 1111 | 1000 cccc | déplacement bas | déplacement haut |`

dans le cas d'un saut proche, où `cccc` est également précisé par le tableau précédent.

JCXZ.- Le saut conditionnel si $cx = 0$ est codé sur deux octets par :

`| 1110 0011 | déplacement |`

de façon différente des autres sauts conditionnels.

Sauts inconditionnels.- 1^o) Les sauts inconditionnels courts directs sont codés sur deux octets :

`| 1110 1011 | déplacement |`

- 2^o) Les sauts inconditionnels proches directs sont codés sur trois octets :

`| 1110 1001 | déplacement bas | déplacement haut |`

- 3^o) Les sauts inconditionnels proches indirects sont codés sur trois octets :

`| 1111 1111 | mod 100 r/m | déplacement |`

- 4^o) Les sauts inconditionnels lointains directs sont codés sur cinq octets :

`| 1001 1010 | IP bas | IP haut | CS bas | CS haut |`

- 5^o) Les sauts inconditionnels lointains indirects sont codés sur six octets :

`| 1111 1111 | mod 100 r/m | IP bas | IP haut | CS bas | CS haut |`

24.1.3.7 Cas d'un registre huit bits, d'un adressage direct ou indirect

Format.- Dans le cas d'un registre 8 bits ou des adressages direct et indirect, l'instruction est codée sur deux octets ou plus suivant le format :

```
| opcode1 w | mod opcode2 r/m |
```

ou :

```
| opcode1 w | mod opcode2 r/m | deplacement |
```

où la première partie `opcode1` de l'opcode occupe sept bits, la seconde partie `opcode2` occupe trois bits, le champ `w` occupe un bit, le champ `mod` occupe deux bits et le champ `r/m` occupe trois bits.

Opcodes.- Le tableau suivant donne l'opcode des instructions à un opérande pour la **forme longue** :

Instruction	Code
`call` proche	`| 1111111 1 | mod 010 r/m |`
`call` lointain	`| 1111111 1 | mod 011 r/m |`
`dec`	`| 1111111 w | mod 001 r/m |`
`div`	`| 1111011 w | mod 110 r/m |`
`idiv`	`| 1111011 w | mod 111 r/m |`
`imul`	`| 1111011 w | mod 101 r/m |`
`inc`	`| 1111111 w | mod 000 r/m |`
`jmp` (proche indirect)	`| 1111111 1 | mod 100 r/m |`
`mul`	`| 1111011 w | mod 100 r/m |`
`neg`	`| 1111011 w | mod 011 r/m |`
`not`	`| 1111011 w | mod 010 r/m |`
`pop`	`| 1000111 1 | mod 000 r/m |`
`push`	`| 1111111 1 | mod 110 r/m |`

Le champ mot.- Le champ `w` (pour *Word* ou *Width*, largeur de l'opérande) spécifie si l'opérande occupe un octet (`w = 0`) ou un mot de deux octets (`w = 1`).

Le champ mode.- Le champ `mod` spécifie si l'opérande est un registre (`mod = 11`) ou un emplacement mémoire ($mod \neq 11$, avec des significations que nous spécifierons ci-dessous).

Désignation des registres généraux et de la mémoire.- Rappelons qu'il y a sept modes d'adressage pour le microprocesseur 8086 :

- immédiat, par exemple[1] : `mov al, 2`,
- registre à registre, par exemple : `mov ax, bx`,
- direct, par exemple : `mov dl, [200]`,
- indirect par registre, par exemple : `mov al, [bx]`,
- relatif à une base, par exemple : `mov cx, [bx]+10`,
- relatif indexé, par exemple : `mov dx, [si]+6`,
- indexé relatif à une base, par exemple : `mov ah, [bp+si+24]`.

1. Nous choisissons l'exemple d'une instruction à deux opérandes car elle est plus naturelle.

Le type d'adressage est déterminé par les champs `w`, `mod` et `r/m`. Si l'opérande est un registre ($mod = 11$), le champ `r/m` est interprété comme un champ registre `reg` suivant le tableau ci-dessous, la nature du registre (un octet ou deux octets) dépendant du champ `w`. Si l'opérande est en mémoire ($mod \neq 11$), les champs `r/m` et `mod` spécifient l'opérande de la façon suivante :

r/m	mod = 00	mod = 01 ou 10	mod = 11	
			w = 0	w = 1
000	[DS:(BX+SI)]	[DS:(BX+SI+disp)]	AL	AX
001	[DS:(BX+DI)]	[DS:(BX+DI+disp)]	CL	CX
010	[SS:(BP+SI)]	[SS:(BP+SI+disp)]	DL	DX
011	[SS:(BP+DI)]	[SS:(BP+DI+disp)]	BL	BX
100	[DS:SI]	[DS:(SI+disp)]	AH	SP
101	[DS:DI]	[DS:(DI+disp)]	CH	BP
110	direct	[SS:(BP+disp)]	DH	SI
111	[DS:BX]	[DS:(BX+disp)]	BH	DI

où le déplacement `disp` (pour l'anglais *DISPlacement*) a pour taille un octet (interprété comme entier relatif) si `mod = 01` et deux octets (interprété comme un entier naturel) si `mod = 10`. L'adresse, dans le cas d'un adressage direct, tient sur deux octets.

Exemples.- 1^o) L'instruction :

```
inc dh
```

est codée :

```
1111 111 0   11 000 110
```

soit `FE A6h`.

- 2^o) L'instruction :

```
inc byte ptr [bx + 4]
```

est codée :

```
1111 111 0   01 000 111   0000 0100
```

soit `FE 47 04h`.

24.1.4 Instructions avec deux opérandes

Dans le cas des instructions à deux opérandes, l'un des opérandes est nécessairement un registre ; le code machine occupera donc au plus six octets (sept lorsqu'il y a un préfixe).

24.1.4.1 Adressage direct

Exemple.- Considérons l'exemple des instructions :

```
mov registre, constante
```

du transfert d'une constante dans un registre. Il y a *a priori* autant de telles instructions que de registres. En fait l'opcode pour ces instructions est :

`| 1011xxxx |`

soit :

```
BXh
```

où les quatre derniers bits ne sont pas déterminés de façon arbitraire.

Format.- Le format est en fait :

`| 1011 w reg |`

où le bit `w` (évidemment pour *Word*) indique si l'opérande occupe un octet (il est alors égal à 0) ou un mot (soit deux octets ; il est alors égal à 1), et où les trois bits de `reg` déterminent le registre général suivant le tableau vu précedemment pour `r/m` dans le cas $mod = 11$.

Il y a évidemment un ou deux octets de plus pour indiquer la valeur de la constante, ce qui fait en tout une instruction codée sur deux ou trois octets.

Exemple.- Ainsi :

```
mov ax,2
```

se traduira en langage machine par :

`| 1011 1 000 | 0000 0000 | 0000 0010 |`

soit `B8 00 02h` et :

```
mov ah,3
```

par :

`| 1011 0 100 | 0000 0011 |`

soit `C4 03`.

24.1.4.2 Le bit de direction

Exemple.- Considérons l'instruction d'addition. Son format, sur deux, trois ou quatre octets, est :

`|000000 d w | mod reg r/m | déplacement |`

où `d` indique le bit de *direction*, les autres champs ayant déjà été vus.

Le premier octet est celui de l'opcode, le second est l'**octet de mode**.

Bit de direction.- Le **bit de direction** `d` indique si le registre désigné par `reg` est le premier opérande (`d = 1`) ou le second opérande (`d = 0`). L'autre opérande est déterminé par les champs `w`, `mod` et `r/m` de la façon vue dans le cas d'une instruction à un opérande.

Exemple 1.- L'instruction :

`| 0000 0011 | 1101 1000 |`

a ses six premiers bits de l'opcode tous égaux à 0, il s'agit donc d'une instruction `add`. On a alors : `d = 1`, `w = 1`, `mod = 11`, `reg = 011` et `r/m = 000`. Puisque `d = 1`, le premier opérande est spécifié par `w` et `reg`, c'est donc ici le registre `bx`. Le second opérande est spécifié par `mod` et `r/m`, c'est ici `ax`. L'instruction est donc :

```
add bx, ax
```

Exemples 2.- L'instruction :

`| 0000 0001 | 0001 | 0110 |`

a pour opcode `0000 00`, il s'agit donc d'une addition. On a `d = 0`, `w = 1`, `mod = 00`, `reg = 010` et `r/m = 110`. Puisque `d = 0`, le registre est le second opérande. Puisque `w = 1` et `reg = 010`, sa valeur est `dx`. On a donc :

```
mov ??, dx
```

Les valeurs de `mod` et de `r/m` montrent qu'il s'agit d'un adressage direct. On doit donc avoir deux octets supplémentaires pour indiquer le décalage de l'emplacement mémoire. Si l'instruction est :

```
| 0000 0001 | 0001 01100 | 1010 0010 | 1011 1100 |
```

alors l'instruction est :

```
mov [A2BCh],dx
```

24.1.4.3 Le bit de signe

Exemple.- Une autre forme de codage de l'addition commence par les deux octets suivants :

```
|1000 00 s w | mod 000 r/m |
```

Il s'agit d'une addition dont le deuxième opérande est une constante. Le nouveau champ, le **bit de signe** `s`, permet de gagner un octet pour le code machine.

Signification du bit de signe.- Pour les opérandes d'un seul octet, c'est-à-dire lorsque `w = 0`, il ne sert à rien. Dans le cas d'opérandes d'un mot, c'est-à-dire lorsque `w = 1` :

- si `s = 0`, les seize bits du deuxième opérande (immédiat) sont présents, il y a donc deux octets supplémentaires ;
- si `s = 1`, le deuxième opérande est présent sous la forme d'un seul octet, qui doit être interprété comme un entier relatif, qu'il faudra transformer en le même entier relatif sur deux octets.

Exemple.- Le codage de l'instruction, exprimée en langage symbolique par :

```
add bx,14
```

sera :

```
| 1000 00 1 1 | 11 000 011 | 0000 1110 |
```

soit `83 C3 0Eh`.

24.1.4.4 Tableau de codage des instructions à deux opérandes

Au vu de ce qu'on vient de dire, on comprend qu'une instruction à deux opérandes peut être codée sous plusieurs formats différents :

Instruction	Formats
adc	| 0001 00dw | mod reg r/m | déplacement |
	| 1000 00sw | mod 010 r/m | déplacement | immédiat |
	| 0001 010w | immédiat |
add	| 0000 00dw | mod reg r/m | déplacement |
	| 1000 00sw | mod 000 r/m | déplacement | immédiat |
	| 0000 010w | immédiat |
and	| 0010 00dw | mod reg r/m | déplacement |
	| 1000 00sw | mod 100 r/m | déplacement | immédiat
	| 0010 010w | immédiat |
cmp	| 0011 10dw | mod reg r/m | déplacement |
	| 1000 00sw | mod 111 r/m | déplacement | immédiat |
	| 0011 110w | immédiat |
in	| 1110 010w | immédiat |
	| 1110 110w |
lds	| 1100 0101 | mod reg r/m | déplacement |
lea	| 1000 1101 | mod reg r/m | déplacement |

les		1100 0100	mod reg r/m	déplacement	
lss		0000 1111	1011 0010	mod reg r/m	déplacement	
mov		1000 10dw	mod reg r/m	déplacement	
		1011 w reg	immédiat	
		1100 011w	mod 000 r/m	immédiat	
		1010 000w	déplacement	
		1010 001w	déplacement	
		1000 111w	mod sreg r/m	déplacement	
		1000 1100	mod sreg r/m	déplacement	
or		0000 10dw	mod reg r/m	déplacement	
		1000 00sw	mod 001 r/m	déplacement	immédiat
		0000 110w	immédiat	
out		1110 011w	immédiat	
		1110 111w	
rcl		1101 00dw	mod 010 r/m	déplacement	
		1101 001w	mod 010 r/m	déplacement	

24.2 Désassemblage

Pour désassembler, on peut reprendre les tableaux précédents en triant suivant la seconde colonne au lieu de la première. Il est cependant traditionnel de présenter les résultats suivant un tableau bidimensionnel dont le premier chiffre hexadécimal désigne la ligne et le second chiffre hexadécimal la colonne. Bien entendu cette tradition date de l'époque où le code opération tenait sur un octet.

24.2.1 Instructions sans préfixe

Pour des raisons de présentation nous allons présenter ces tableaux en deux fois : l'un pour les colonnes de 0 à 7 et l'autre pour les colonnes de 8 à F. On utilise les abréviations suivantes :

- b = mot de 8 bits (pour *Byte*)
- d = direct
- f = depuis un registre (*from*)
- i = immédiat
- ia = immédiat vers accumulateur
- id = indirect
- is = immédiat 8 bits étendus à 16 bits
- l = saut/appel intersegment
- m = mémoire
- r/m = le deuxième octet donne le mode d'adressage
- c = saut/appel intrasegment
- sr = registre de segment
- t = vers un registre (*towards*)
- v = variable
- w = mot de 16 bits

	0	1	2	3	4	5	6	7
0	ADD b.f.r/m	ADD w.f.r/m	ADD b.t.r/m	ADD w.t.r/m	ADD b.ia	ADD w.ia	PUSH ES	POP ES
1	ADC b.f.r/m	ADC w.f.r/m	ADC b.t.r/m	ADC w.t.r/m	ADC b.ia	ADC w.ia	PUSH SS	POP SS
2	AND b.f.r/m	AND w.f.r/m	AND b.t.r/m	AND w.t.r/m	AND b.ia	AND w.ia	SEG = ES	DAA
3	XOR b.f.r/m	XOR w.f.r/m	XOR b.t.r/m	XOR w.t.r/m	XOR b.ia	XOR w.ia	SEG = SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6	PUSHA	POPA	BOUND					
7	JO	JNO	JC/JB/ JNAE	JNC/JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA

	8	9	A	B	C	D	E	F
0	OR b.f.r/m	OR w.f.r/m	OR b.t.r/m	OR w.t.r/m	OR b.ia	OR w.ia	PUSH CS	
1	SBB b.f.r/m	SBB w.f.r/m	SBB b.t.r/m	SBB w.t.r/m	SBB b.ia	SBB w.ia	PUSH DS	POP DS
2	SUB b.f.r/m	SUB w.f.r/m	SUB b.t.r/m	SUB w.t.r/m	SUB b.ia	SUB w.ia	SEG = CS	DAS
3	CMP b.f.r/m	CMP w.f.r/m	CMP b.t.r/m	CMP w.t.r/m	CMP b.ia	CMP w.ia	SEG = DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSH b.i	IMUL b.r/m	PUSH w.i					
7	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG

Dans la suite du tableau, on a :

mod xxx r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
Grp1	TEST		NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL l.id	JMP id	JMP l.id	PUSH	

	0	1	2	3	4	5	6	7
8	Immed b.r/m	Immed w.r/m	Immed b.r/m	Immed is.r/m	TEST b.r/m	TEST w.r/m	XCHG b.r/m	XCHG w.r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m - AL	MOV m - AX	MOV AL - m	MOV AX - m	MOVS b	MOVS w	CMPS b	CMPS w
B	MOV i - AL	MOV i - CL	MOV i - DL	MOV i - BL	MOV i - AH	MOV i - CH	MOV i - DH	MOV i - BH
C	Shift,n b	Shift,n w	RET c(i + SP)	RET c	LES r/m	LDS r/m	MOV b.i.r/m	MOV w.i.r/m
D	Shift b	Shift w	Shift b.v	Shift w.v	AAML	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w
F	LOCK		REP/ REPNZ	REPZ	HLT	CMC	Grp1 b.r/m	Grp1 w.r/m

	8	9	A	B	C	D	E	F
8	MOV b.f.r/m	MOV w.f.r/m	MOV b.t.r/m	MOV w.t.r/m	MOV sr.f.r/m	LEA r/m	MOV sr.t.r/m	POP r/m
9	CBW	CWD	CALL l.d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b.ia	TEST w.ia	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i - AX	MOV i - CX	MOV i - DX	MOV i - BX	MOV i - SP	MOV i - BP	MOV i - SI	MOV i - DI
C			RET l.(i+SP)	RET l	INT 3L	INT	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL d	JMP d	JMP l.d	JMP c.d	IN v.b	IN v.w	OUT v.b	OUT v.w
F	CLC	STC	CLI	STI	CLD	STD	Grp2 b.r/m	Grp2 w.r/m

Exercice.- *Désassembler le programme suivant* :