# Chapitre 20

# Manipulation des bits

Nous avons vu comment accéder aux octets, qui est la plus petite unité adressable. Cependant la plus petite quantité d'information est le bit. On a rarement intérêt à y accéder pour effectuer des calculs (bien que cela puisse être intéressant pour programmer la multiplication, par exemple). La manipulation des bits est, par contre, intéressante en programmation système, plus exactement pour contrôler des périphériques. En effet on a souvent besoin, dans ce cas, de coder de l'information binaire : on pourrait utiliser un octet pour une telle information mais on perdrait alors beaucoup d'espace et, surtout, beaucoup de temps lors des échanges avec les périphériques. On a donc intérêt à coder ces informations par des bits individuels, ce qui permet l'utilisation d'un octet au lieu de huit (tout au moins dans le cas optimum). Il faut donc pouvoir accéder à chacun des bits individuellement pour cela. C'est l'objet des *instructions de manipulation de bits*.

## 20.1 Instructions logiques

### 20.1.1 Mise en place

Introduction.- Les **instructions logiques** correspondent à l'implémentation de certains connecteurs logiques tels que la négation, la conjonction et la disjonction. Un octet peut être considéré comme un octuplet de bits. Lors d'une instruction logique, on exécute la même opération logique en parallèle sur chacun des huit bits, 1 représentant le vrai et 0 le faux.

Langage symbolique.- Il y a quatre instructions logiques implémentées sur le 8086/8088 : les instructions logiques parallèles (ou bit à bit) de négation NOT, de conjonction AND, de disjonction OR et de disjonction exclusive XOR (*eXclusive OR*) :

```
NOT source
AND destination, source
OR  destination, source
XOR destination, source
```

Sémantique.- Si, avant l'instruction, le contenu de l'accumulateur AL est :

1110 0011

et celui du registre BL est :

0010 0001

alors, après l'instruction :

`AND AL, BL`

le contenu de l'accumulateur AL est :

0010 0001.

Langage machine.- 1$^o$) L'instruction NOT est codée sur deux octets par :

`| 1111 011w | mod 010 r/m |`

- 2$^o$) L'instruction AND possède trois codages selon la nature des opérandes :

– a) La conjonction entre mémoire ou registre et mémoire est codée sur deux ou quatre octets par :

`| 0010 00dw | mod reg r/m |        |        |`

– b) La conjonction immédiate dans l'accumulateur est codée sur deux ou trois octets par :

`| 0010 010w | donnée | donnée si` $w = 1$ `|`

– c) La conjonction immédiate avec le contenu d'une case mémoire ou de registre est codée sur trois ou quatre octets par :

`| 1000 000w | mod 100 r/m | donnée | donnée si` $w = 1$ `|`

- 3$^o$) L'instruction OR possède trois codages selon la nature des opérandes :

– a) La disjonction entre mémoire ou registre et mémoire est codée sur deux ou quatre octets par :

`| 0000 10dw | mod reg r/m |        |        |`

– b) La disjonction immédiate dans l'accumulateur est codée sur deux ou trois octets par :

`| 0000 110w | donnée | donnée si` $w = 1$ `|`

– c) La disjonction immédiate avec le contenu d'une case mémoire ou de registre est codée sur trois ou quatre octets par :

`| 1000 000w | mod 001 r/m | donnée | donnée si` $w = 1$ `|`

- 3$^o$) L'instruction XOR possède trois codages selon la nature des opérandes :

– a) La disjonction exclusive entre mémoire ou registre et mémoire est codée sur deux ou quatre octets par :

`| 0011 00dw | mod reg r/m |          |          |`

– b) La disjonction exclusive immédiate dans l'accumulateur est codée sur deux ou trois octets par :

`| 0011 010w | donnée | donnée si` $w = 1$ `|`

– c) La disjonction exclusive immédiate avec le contenu d'une case mémoire ou de registre est codée sur trois ou quatre octets par :

`| 1000 000w | mod 110 r/m | donnée | donnée si` $w = 1$ `|`

Exemple.- Vérifions le résultat de l'exemple vu à propos de la sémantique en écrivant un programme, d'abord en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AL,E3
MOV BL,21
AND AL,BL
MOV [0400],AL
RET
```

puis en langage machine :

```
B8 00 30
8E D8
B0 E3
B3 21
20 D8
A2 00 04 C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B0 E3
-E CS:107 B3 21
-E CS:109 20 D8
-E CS:10B A2 00 04
-E CS:10E C3
-R BX
BX 0000
:
-R CX
CX 0000
:F
-N AND.COM
-W
Ecriture de 0000F octets
```

```
-Q
```

Après exécution du programme `and.com`, on vérifie que l'on a bien 21 = 33h à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 21 00 03 00 0B 00 00 00-00 00 00 00 00 00 00 00 !...............
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

## 20.1.2 Application au masquage

On veut souvent déterminer la valeur d'un ou de plusieurs bits. On pourrait évidemment amener chacun de ces bits à la première position (grâce aux instructions de décalage que nous verrons ci-après) et tester. Mais ceci représente beaucoup d'instructions. On utilise donc plutôt la technique du **masquage** que nous allons décrire maintenant.

### 20.1.2.1 Masquage simple

Introduction.- On parle de **masquage simple** lorsqu'on veut connaître la valeur d'un seul bit.

Principe.- On utilise l'instruction AND et on regarde l'indicateur voulu du registre des indicateurs. Par exemple, pour déterminer si le bit se trouvant à la position 3 du registre BL est égal à 1, on place `0000 1000b`, valeur appelée **masque**, dans l'accumulateur AL et on exécute l'instruction AND AL, BL. Il peut y avoir au plus un 1 dans le résultat : à la position 3. De plus on obtient ce 1 si, et seulement si, la valeur du troisième bit de BL est égale à 1. Ainsi le troisième bit de BL est égal à 1 si, et seulement si, l'indicateur ZF est égal à 0 après cette instruction.

### 20.1.2.2 Masquage et traduction

Introduction.- Le masquage peut servir pour déterminer la valeur d'un bit, comme nous venons de le voir. Il peut également servir dans certains cas à la traduction.

Application.- Dans le codage ASCII, les chiffres '0' à '9' sont codés par les valeurs `30h` à `39h`. Lorsqu'on récupére le code ASCII d'un chiffre, pour obtenir la valeur numérique correspondante, il suffit donc de soustraire `30h`. Ceci peut être obtenu par une soustraction mais on peut aussi utiliser une technique de masquage.

Supposons en effet que le code ASCII se trouve dans le registre AL. Plaçons `0Fh`, c'est-à-dire `0000 1111b` dans le registre BL. Alors AND AL, BL donne la valeur numérique du chiffre correspondant.

### 20.1.2.3 Masquage multiple

On peut vouloir tester plusieurs bits. On peut le faire un par un. On peut aussi les tester tous à la fois, mais uniquement pour savoir s'ils sont tous présents (avec `xor`) ou si au moins l'un d'eux est présent (avec `or`).

#### 20.1.2.4 Initialisation à zéro

On a vu comment initialiser l'accumulateur à zéro grâce à l'instruction :

```
MOV AX, 0
```

qui se traduit par trois octets en langage machine. Pour gagner de la place dans l'espace réservé au code, les programmeurs utilisent souvent l'astuce suivante :

```
XOR AX, AX
```

qui réalise la même chose mais est codé sur deux octets en langage machine.

#### 20.1.2.5 Test à zéro

On a vu comment tester si l'accumulateur est nul grâce à l'instruction :

```
CMP AX, 0
```

qui se traduit par trois octets en langage machine. Pour la même raison que ci-dessus, les programmeurs utilisent souvent :

```
OR AX, AX
```

## 20.2 Les décalages logiques

Introduction.- Puisque les microprocesseurs manipulent des nombres en binaire, ils peuvent effectuer facilement une multiplication ou une division par deux, en décalant tous les bits d'une position à gauche (éventuellement avec un débordement) ou à droite (en perdant le premier bit). Ceci explique l'intérêt des **décalages**.

Les concepteurs du microprocesseur 8080, puis ceux du 8086/8088, ont implémenté deux familles de décalages : l'une pour multiplier et diviser par deux les entiers naturels (appelés **décalages logiques**) et l'autre pour les entiers relatifs (appelés **décalages arithmétiques**). Nous allons nous intéresser au premier type de décalages dans cette section.

Langage symbolique.- $1^o$) L'instruction :

```
SAL source
```

(pour l'anglais *Shift Arithmetic Left*) ou :

```
SHL source
```

(pour l'anglais *SHift Left*) décale tous les bits d'un registre ou d'une case mémoire d'une position vers la gauche et place un zéro comme bit le plus à droite, ce qui équivaut à une multiplication par deux. Lorsqu'il y a débordement, l'indicateur de débordement CF est positionné à un. L'indicateur de dépassement de capacité OF est mis à 1 si le bit de poids fort (celui de signe pour les entiers relatifs) est changé (perte de l'information de siggne), sinon il est mis à 0 :

$$\boxed{\text{CF}} \leftarrow \boxed{\phantom{xxxxxxx}} \leftarrow 0$$

Ce décalage à gauche peut avoir lieu plusieurs fois, le nombre de fois étant le contenu de CL.

2$^o$) L'instruction :

`SHR source`

(pour l'anglais *SHift Right*) décale tous les bits d'un registre ou d'une case mémoire d'une position vers la droite et place un zéro comme bit le plus à gauche, ce qui équivaut à une division par 2 non signée. Le bit de droite est placé comme indicateur de débordement CF. L'indicateur de dépassement de capacité OF est mis à 1 si le bit de poids fort est différent du précédent (pert de l'information de signe), sinon il est mis à 0 :

$$0 \rightarrow \boxed{\phantom{xxxxxxxx}} \rightarrow \boxed{\text{CF}}$$

Ce décalage à droite peut avoir lieu plusieurs fois, le nombre de fois étant le contenu de CL.

3$^o$) L'instruction :

`SAR source`

(pour l'anglais *Shift Arithmetic Right*) décale les 7 ou 15 bits de poids faible d'un registre ou d'une case mémoire d'une position vers la droite et place un zéro comme septième ou quinzième bit, ce qui équivaut à une division par 2 signée. Le bit de droite est placé comme indicateur de débordement CF. L'indicateur de dépassement de capacité OF est mis à 0.

Ce décalage à droite peut avoir lieu plusieurs fois, le nombre de fois étant le contenu de CL.

<u>Langage machine</u>.- 1$^o$) Le décalage à gauche est codé sur deux ou quatre octets par :

```
| 1100 00vw | mod 100 r/m |        |        |
```

avec $v = 0$ pour un décalage et $v = 1$ pour plusieurs décalages dont le nombre est contenu dans CL (dont la valeur n'est pas détruite).

- 2$^o$) Le décalage logique à droite SHR est codé sur deux ou quatre octets par :

```
| 1101 00vw | mod 100 r/m |        |        |
```

avec $v = 0$ pour un décalage et $v = 1$ pour plusieurs décalages dont le nombre est contenu dans CL (dont la valeur n'est pas détruite).

- 3$^o$) Le décalage arithmétique à droite SAR est codé sur deux ou quatre octets par :

```
| 1101 00vw | mod 111 r/m |        |        |
```

avec $v = 0$ pour un décalage et $v = 1$ pour plusieurs décalages dont le nombre est contenu dans CL (dont la valeur n'est pas détruite).

<u>Exemple</u>.- Vérifions que le décalage logique à droite de `8Fh`, soit `1000 1111b`, donne bien `0100 0111b`, soit `47h`. Écrivons un programme, d'abord en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AL,8F
SHR AL,1
MOV [0400],AL
RET
```

puis en langage machine :

```
B8 00 30
```

```
8E D8
B0 8F
D0 E8
A2 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B0 8F
-E CS:107 D0 E8
-E CS:109 A2 00 04
-E CS:10C C3
-R BX
BX 0000
:
-R CX
CX 0000
:D
-N SHR.COM
-W
Ecriture de 0000D octets
-Q
```

Après exécution du programme `shr.com`, on vérifie que l'on a bien 47h à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 47 00 03 00 0B 00 00 00-00 00 00 00 00 00 00 00 G...............
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

## 20.3 Les rotations

Lorsqu'on effectue un décalage à gauche de 4 bits, les 4 bits les plus à gauche sont perdus pour un décalage sur un octet (à part l'un d'eux qui peut être récupéré dans l'indicateur de débordement). Dans certains cas, on ne veut pas perdre ces bits. La famille des **instructions de rotation** permet de réarranger les bits sans les perdre : le bit qui est expulsé d'un côté réapparaît de l'autre côté.

Il existe deux types de rotations : l'une ne concerne que l'octet ou le mot sur lequel on travaille, l'autre fait intervenir l'indicateur de retenue.

### 20.3.1 Les rotations sur un octet ou un mot

Langage symbolique.- Les instructions de rotation du contenu d'un registre ou d'une case mémoire n'affectent que les huit ou seize bits de la source, le bit sortant étant envoyé à la fois à l'autre extrémité et dans l'indicateur de débordement CF :

```
ROL source, n
ROR source, n
```

respectivement pour $n$ rotations à gauche (pour l'anglais *ROtate Left*) et à droite (pour l'anglais *ROtate Right*).

Lorsque le nombre de rotations est 1, l'indicateur de dépassement de capacité OF est significatif : il est mis à 1 si le bit de plus fort poids est changé (c'est-à-dire si le signe est changé), sinon il est mis à 0.

Langage machine.- 1$^o$) La rotation à gauche est codée sur deux à quatre octets :

```
| 1101 00vw | mod 000 r/m |         |         |
```

avec $v = 0$ pour une rotation et $v = 1$ pour plusieurs rotations, $n$ étant alors le contenu du registre CL (dont la valeur ne sera pas détruite).

- 2$^o$) La rotation à droite est codée sur deux à quatre octets :

```
| 1101 00vw | mod 001 r/m |         |         |
```

avec $v = 0$ pour une rotation et $v = 1$ pour plusieurs rotations, $n$ étant alors le contenu du registre CL (dont la valeur ne sera pas détruite).

Exemple.- Vérifions que la rotation à droite de `8Fh`, soit `1000 1111b`, donne bien `1100 0111b`, soit `C7h`. Écrivons un programme, d'abord en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AL,8F
ROR AL,1
MOV [0400],AL
RET
```

puis en langage machine :

```
B8 00 30
8E D8
B0 8F
D0 C8
A2 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B0 8F
-E CS:107 D0 C8
-E CS:109 A2 00 04
-E CS:10C C3
-R BX
BX 0000
:
-R CX
```

```
CX 0000
:D
-N ROR.COM
-W
Ecriture de 0000D octets
-Q
```

Après exécution du programme `ror.com`, on vérifie que l'on a bien C7h à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 C7 00 03 00 0B 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

### 20.3.2 Les rotations sur un octet ou un mot et l'indicateur CF

Langage symbolique.- Dans une rotation sur le contenu d'un registre ou d'une case mémoire et l'indicateur CF, le bit expulsé d'un côté est placé dans l'indicateur CF et le bit de l'indicateur est placé dans la source de l'autre côté. Il s'agit donc d'une rotation sur neuf ou dix-sept bits :

```
RCL source, n
RCR source, n
```

respectivement pour une rotation à gauche (pour l'anglais *Rotate with Carry Left*) et à droite (pour l'anglais *Rotate with Carry Right*).

Lorsque le nombre de rotations est 1, l'indicateur de dépassement de capacité est significatif : il est mis à 1 lorsque le bit de plus fort poids change de valeur (c'est-à-dire si le signe est changé), sinon il est mis à zéro.

Langage machine.- $1^o$) La rotation à gauche est codée sur deux à quatre octets :

```
| 1101 00vw | mod 010 r/m |        |        |
```

avec $v = 0$ pour une rotation et $v = 1$ pour plusieurs rotations, $n$ étant alors le contenu du registre CL (dont la valeur n'est pas détruite).

- $2^o$) La rotation à droite est codé sur deux à quatre octets :

```
| 1101 00vw | mod 011 r/m |        |        |
```

avec $v = 0$ pour une rotation et $v = 1$ pour plusieurs rotations, $n$ étant alors le contenu du registre CL (dont la valeur n'est pas détruite).

Exemple.- Si le contenu de l'accumulateur AL est `0100 0011b`, soit 43h, et celui de l'indicateur de retenue CF est 0, vérifions que la rotation à droite donne bien `0010 0001b`, soit `21h`. Écrivons un sous-programme, d'abord en langage symbolique :

```
MOV AX,3000
MOV DS,AX
SUB AL,AL
```

```
   MOV AL,43
   RCR AL,1
   MOV [0400],AL
   RET
```

puis en langage machine :

```
B8 00 30
8E D8
28 C0
B0 43
D0 D8
A2 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 28 C0
-E CS:107 B0 43
-E CS:109 D0 D8
-E CS:10B A2 00 04
-E CS:10E C3
-R BX
BX 0000
:
-R CX
CX 0000
:F
-N RCR.COM
-W
Ecriture de 0000F octets
-Q
```

Après exécution du programme `rcr.com`, on vérifie que l'on a bien 21h à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400  21 00 03 00 0B 00 00 00-00 00 00 00 00 00 00 00  !...............
3000:0410  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0420  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0430  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0440  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0450  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0470  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
3000:0470  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
-q
```

## 20.4 Changer l'indicateur de retenue

Comme le bit 0 du registre des indicateurs, l'indicateur de retenue CF, est souvent utilisé, trois instructions le concernent.

Langage symbolique.- L'instruction :

`STC`

(pour l'anglais *SeT Carry*) place 1 dans l'indicateur CF. L'instruction :

`CLC`

(pour l'anglais *CLear Carry*) place 0 dans l'indicateur CF. L'instruction :

`CMC`

(pour *CoMplement Carry*) inverse la valeur de l'indicateur CF.

Langage machine.- 1$^o$) Le positionnement de l'indicateur CF est codé sur un octet par `1111 1001`, soit F9h.

- 2$^o$) L'annihilation de l'indicateur CF est codée sur un octet par `1111 1000`, soit F8h.

- 3$^o$) L'inversion de l'indicateur CF est codée sur un octet par `1111 0101`, soit F5h.

## 20.5 Historique

Le premier ordinateur (EDSAC, 1949 ; voir chapitre ? ?) n'utilise qu'une seule instruction logique : le AND sous la forme de l'instruction *collate*. Les instructions de décalage sont un peu compliquées : pour simplifier la mise en place matérielle, le nombre de positions à décaler n'est pas donnée par la valeur du champ d'adresse de l'instruction mais par la position du bit le plus à doite ; ainsi l'instruction `L 8 S` a pour conséquence que le contenu de l'accumulateur est décalé de 5 places à gauche et non 8 comme on pourrait s'y attendre.