Chapitre 4

Calculs sur les entiers naturels

Nous avons vu que les grandes fonctions d'un microprocesseur sont les accès au microprocesseur et les calculs. Nous avons vu les premières dans les trois chapitres précédents. Voyons maintenant comment faire des calculs, en commençant par les calculs sur les entiers naturels.

4.1 Étude générale

On peut décomposer tout "calcul" en des calculs élémentaires (il faut bien partir de quelque chose) et en combinaison de ceux-ci. Les instructions fondamentales d'un microprocesseur sont donc les *instructions de calcul élémentaire* et les *instructions de contrôle*.

Instructions de calcul élémentaire.- L'ordinateur est un descendant des machines à calculer perfectionnées, qui calculaient beaucoup plus vite que l'être humain : machines mécaniques puis electrotechniques puis électroniques. Il résulte de cette descendance que les opérations courantes, tout au moins les opérations les plus élémentaires, sont implémentées sur le microprocesseur. C'est le cas en général de l'addition et de la soustraction (éventuellement de la multiplication) sur les entiers relatifs. La division euclidienne est aussi souvent implémentée. Les opérations sur les nombres réels sont rarement implémentées sur le microprocesseur lui-même. Elles peuvent l'être, sous une forme élémentaire, sur ce qu'on appelle le **coprocesseur arithmétique**. Par contre les opérations logiques (conjonction, disjonction, négation globales et bits à bits et les rotations des bits) sont implémentées, à la fois parce que c'est facile et parce qu'on les utilise très souvent en programmation système, alors qu'elles ne l'étaient pas

sur les machines à calculer.

Instructions de contrôle.- Nous avons vu, lors de l'initiation à la programmation, qu'en langage évolué on dispose de beaucoup moins d'opérations que sur une calculette scientifique mais que ce qui fait toute la force d'un tel langage sont ses instructions de contrôle. On doit donc retrouver de telles instructions au niveau du microprocesseur, même s'il ne s'agit pas du même jeu d'instructions.

4.2 Opérations élémentaires sur les entiers naturels

Les premières opérations à lesquelles on s'attend sur un ordinateur concernent, bien entendu, les opérations sur les entiers naturels, à savoir essentiellement l'addition, la multiplication et la soustraction (cette dernière opération étant une opération partielle).

En fait on ne peut pas manipuler l'ensemble des entiers naturels mais seulement une partie d'entre eux, celle-ci dépendant de la taille des registres.

L'opération fondamentale est, en théorie, l'*incrémentation*, c'est-à-dire le passage au suivant, car elle permet, en utilisant les *structures de contrôle* que nous verrons après, d'obtenir toutes les opérations qui nous intéressent. Cependant l'addition est câblée sur tous les microprocesseurs, ce qui n'est pas toujours le cas de la multiplication.

4.2.1 Représentation des entiers naturels

Amplitude des entiers naturels manipulés.- Puisque la taille des mots sur un `i8086` est de 16 bits (deux octets), on peut manipuler les entiers naturels allant de `0h` à `FFFFh`, c'est-à-dire de 0 à 65 535 ($= 2^{16}$ - 1).

Entiers plus grands.- On peut aussi coder un entier naturel sur plusieurs mots. Nous verrons qu'alors il faut utiliser plus d'instructions pour pouvoir effectuer des opérations sur ces "grands entiers".

4.2.2 Incrémentation et décrémentation

4.2.2.1 Incrémentation

Synatxe.- La syntaxe du passage au suivant est :

```
INC dest
```

où `dest` est un registre ou un emplacement mémoire.

Sémantique.- Après cette instruction, le contenu de `dest` est augmenté de 1. L'**arithmétique** est **modulaire**, c'est-à-dire que l'on passe de `FFFFh` à 0.

Exemple.- Incrémentons le registre `ax`, initialisé à `10h` :

```
C:>debug
-a
249C:0100 mov ax,10
249C:0103 inc ax
249C:0104 int3
```

```
249C:0105
-g
AX=0011 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C IP=0100 NV UP EI NZ NA PE NC
249C:0104 CC INT 3
-q
```

On vérifie bien que `AX` a été incrémenté puisqu'il a la valeur `11` (en hexadécimal, rappelons-le, mais cela n'a pas d'importance pour notre exemple).

Exercice 1.- Vérifier que le suivant de `FFFFh` est 0.

4.2.2.2 Décrémentation

Synatxe.- La syntaxe du passage au prédécesseur est :

```
DEC dest
```

où `dest` est un registre ou un emplacement mémoire.

Sémantique.- Après cette instruction, le contenu de `dest` est diminué de 1. L'arithmétique est également modulaire, c'est-à-dire que l'on passe de 0 à `FFFFh`.

Exercice 2.- 1^o) Écrire un programme qui initialise `AX` à `10h` puis qui décrémente `AX`.

2^o) Vérifier que le prédécesseur de 0 est `FFFFh`.

4.2.3 Addition

Sur un `i8086`, l'addition peut s'effectuer sur des octets (en utilisant des registres à un octet ; réminiscence du `i8085`) ou sur des mots (de deux octets). Tout est prévu également pour effectuer des additions sur plusieurs mots, mais pas en une seule fois.

4.2.3.1 Addition sur des octets ou des mots

Syntaxe.- La syntaxe est :

```
ADD dest, source
```

où `dest` est un registre ou un emplacement mémoire et où `source` est un registre, un emplacement mémoire ou une constante. Cependant `dest` et `source` ne peuvent pas être à la fois des emplacements mémoire.

Sémantique.- Après l'exécution de cette instruction le contenu de `dest` est la somme des contenus de `dest` et de `source` (si elle est inférieure à la capacité), le contenu de `source`, lui, étant inchangé.

Exemple.- Initialisons les registres `ax`, `bx` et `dl` respectivement à `10h`, `2h` et `3h`, puis additionnons `ax` et `bx` et ajoutons `5h` à `dl` :

```
C:>debug
-a
249C:0100 mov ax,10
249C:0103 mov bx,2
249C:0106 mov dl,3
249C:0108 add ax,bx
249C:010A add dl,5
249C:010D int3
```

```
249C:010E
-g
AX=0012 BX=0002 CX=0000 DX=0008 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010D NV UP EI NZ NA PO NC
249C:010D CC INT 3
-q
```

On vérifie bien les résultats (en hexadécimal, mais cela n'a pas d'importance pour nos exemples).

4.2.3.2 Cas du dépassement de capacité

Introduction.- Pour la sémantique de l'addition, nous avons dit que le contenu de `dest` est la somme des contenus de `dest` et de `source` si elle est inférieure à la capacité d'un registre. Bien entendu, dans certains cas, la somme de deux nombres inférieurs à 2^{16} est un nombre supérieur à 2^{16}. Remarquons cependant qu'il ne peut pas être supérieur à 2^{17} et qu'il y a une **retenue** (*carry* en anglais) égale à 1 (et à 0 sinon).

La sémantique complète de l'addition est que `source` est la somme des contenus de `dest` et de `source` modulo 2^8 ou 2^{16} (suivant la capacité des registres utilisés) et l'indicateur `CF` du registre des indicateurs est égal à la somme divisée par 2^8 ou 2^{16}, autrement il est égal à la retenue.

Exemple 1.- Initialisons les registres `ax` et `bx` respectivement à `FFFEh` et `2h` puis additionnons-les :

```
C:>debug
-a
1569:0100 mov ax,FFFE
1569:0103 mov bx,2
1569:0106 add ax,bx
1569:0108 int3
1569:0109
-g

AX=0000  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1569  ES=1569  SS=1569  CS=1569  IP=0108   NV UP EI PL ZR AC PE CY
1569:0108 CC            INT     3
-q
```

On remarque que l'indicateur de retenue est bien positionné (on a `CY` et non plus `NC` comme précédemment).

Exemple 2.- Initialisons les registres `al` et `bl` respectivement à `FEh` et `5h` puis additionnons-les :

```
C:>debug
-a
15BD:0100 mov al,FE
15BD:0102 mov bl,5
15BD:0104 add al,bl
15BD:0106 int3
15BD:0107
-g

AX=0003  BX=0005  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15BD  ES=15BD  SS=15BD  CS=15BD  IP=0106   NV UP EI PL NZ AC PE CY
15BD:0106 CC            INT     3
-q
```

Là encore, on remarque que l'indicateur de retenue est bien positionné.

4.2.3.3 Addition sur plusieurs mots

Syntaxe.- Si les entiers à additionner sont trop grands, on peut utiliser plusieurs mots pour les représenter. Par exemple si on veut additionner `2FFFEh` et `10003h`, il suffit d'additionner, d'une part `FFFEh` et `3h` et, d'autre part, `2h` et `1h`. Mais il y a une retenue dont il faut tenir compte.

On dispose de d'une autre instruction d'addition :

```
ADC dest, source
```

(pour *ADdition with Carry*) qui place dans `dest` la somme du contenu de `dest`, du contenu de `source` et de `CF`.

Exemple.- Pour le cas pris en exemple en introduction, on peut utiliser le programme suivant :

```
C:>debug
-a
249C:0100 mov ax,FFFE
249C:0103 add ax,3
249C:0106 mov bx,2
249C:0109 adc bx,1
249C:010C int 3
249C:010D
-g
AX=0001 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010C NV UP EI PL NZ NA PO NC
249C:010C CC INT 3
-q
```

En consultant `ax` et `bx`, on trouve bien le résultat `4001h`.

4.2.4 Soustraction

Comme l'addition, la soustraction peut se faire sur des octets (en utilisant des registres à un octet), sur des mots (en utilisant des registres de deux octets) ou sur plusieurs mots.

4.2.4.1 Soustraction sur des octets ou des mots

Syntaxe.- Dans ce cas on écrit :

```
SUB dest, source
```

où `dest` est un registre ou un emplacement mémoire et où `source` est un registre, un emplacement mémoire ou une constante. La seule restriction est que `dest` et `source` ne peuvent pas être à la fois des emplacements mémoire.

Sémantique.- Après l'exécution de cette instruction, le contenu de `dest` est la différence des contenus de `dest` et de `source` (si elle est positive), le contenu de `source` étant inchangé :

```
dest = dest - source.
```

Exemple.- Initialisons les registres `ax`, `bx` et `dl` respectivement à `10h`, `2h` et `5h`, puis effectuons la différence de `ax` et de `bx`, à placer dans `ax`, puis de `dl` et de `3h` à placer dans `dl` :

```
C:>debug
```

```
-a
249C:0100 mov ax,10
249C:0103 mov bx,2
249C:0106 mov dl,5
249C:0108 sub ax,bx
249C:010A sub dl,3
249C:010D int 3
249C:010E
-g
AX=000E BX=0002 CX=0000 DX=0002 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010D NV UP EI PL NZ NA PO NC
249C:010D CC INT 3
-q
```

On vérifie bien les résultats (en hexadécimal : $10_{16} - 2_{16} = 14_{16}$).

4.2.4.2 Cas d'une différence négative

Introduction.- Pour la sémantique de la soustraction, nous avons dit que le contenu de `dest` est la différence des contenus de `dest` et de `source` si elle est positive. Si le contenu de `source` est strictement supérieur à celui de `dest`, il y a un emprunt (*borrow* en anglais). Dans ce cas, la sémantique de la soustraction est : le contenu de `source` est égal à 2^8 ou 2^{16} (suivant la capacité des registres utilisés) plus le contenu de `source` moins le contenu de `dest` et l'indicateur de retenue `CF` du registre des indicateurs est égal à 1.

Exemple 1.- Initialisons les registres `ax` et `bx` respectivement à `8h` et `10h` puis soustrayons-les :

```
C:>debug
-a
15BD:0100 mov ax,8
15BD:0103 mov bx,10
15BD:0106 sub ax,bx
15BD:0108 int3
15BD:0109
-g

AX=FFF8  BX=0010  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15BD  ES=15BD  SS=15BD  CS=15BD  IP=0108   NV UP EI NG NZ NA PO CY
15BD:0108 CC            INT     3
-q
```

On remarque que l'indicateur de retenue est bien positionné. Le contenu de `ax` est bien $2^{17} + 8h - 10h$.

Exemple 2.- Initialisons les registres `al` et `bl` respectivement à `8h` et `10h` puis soustrayons-les :

```
C:>debug
-a
15BD:0100 mov al,8
15BD:0102 mov bl,10
15BD:0104 sub al,bl
15BD:0106 int3
15BD:0107
-g

AX=00F8  BX=0010  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15BD  ES=15BD  SS=15BD  CS=15BD  IP=0106   NV UP EI NG NZ NA PO CY
15BD:0106 CC            INT     3
-q
```

Là encore, on remarque que l'indicateur de retenue est positionné.

4.2.4.3 Soustraction sur plusieurs mots

Introduction.- On peut utiliser plusieurs mots pour représenter des entiers assez grands. Par exemple si on veut soustraire `1F002h` à `21FFEh`, il suffit, d'une part de soustraire `F002h` à `1FFEh` et, d'autre part, de soustraire `1h` à `2h`.

Cependant il y a un emprunt dans le cas de la première soustraction, dont il faut tenir compte. On se sert, pour cela, d'une nouvelle instruction : la soustraction avec emprunt.

Syntaxe.- Pour effectuer une **soustraction avec emprunt**, on utilise l'instruction :

```
SBB dest, source
```

(pour *SuBtract with Borrow*) qui place dans `dest` la différence $CY \times 2^{n+1} + dest - source$, avec $n = 8$ ou $n = 16$ suivant le cas.

Exemple.- Pour le cas précédent, on peut utiliser le programme suivant :

```
C:>debug
-a
249C:0100 mov ax,1FFE
249C:0103 sub ax,F002
249C:0106 mov bx,2
249C:0109 sbb bx,1
249C:010C int 3
249C:010D
-g
AX=2FFC BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010C NV UP EI PL ZR NA PE NC
249C:010C CC INT 3
-q
```

dont on vérifie les résultats.

4.2.5 Multiplication

Dans beaucoup de microprocesseurs la multiplication et la division ne sont pas câblées. C'était d'ailleurs le cas du `i8080/85` : le câblage de ces opérations est une nouveauté du `i8086` chez Intel.

Il faut distinguer le cas de la multiplication de nombres d'un octet et d'un mot, non pas pas par compatibilité avec le `i8085` mais parce que le résultat est un mot ou un double mot.

4.2.5.1 Multiplication de deux octets

Syntaxe.- Pour multiplier deux octets, l'un des opérandes doit être placé dans le registre `AL` et l'autre opérande doit dans un un registre de un octet ou un emplacement mémoire suivant la syntaxe :

```
MUL source
```

Sémantique.- Le produit de ces deux octets est placé dans le registre `AX`.

Exemple.- Pour multiplier `10h` par `11h` on utilise le programme suivant :

```
C:>debug
-a
```

```
249C:0100 mov al,10
249C:0102 mov bl,11
249C:0104 mul bl
249C:0106 int3
249C:0107
-g
AX=0110 BX=0011 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0106 OV UP EI PL NZ NA PE CY
249C:0106 CC INT 3
-q
```

dont on vérifie le résultat.

4.2.5.2 Multiplication de deux mots

Syntaxe.- Pour multiplier deux mots, l'un des opérandes doit être placé dans le registre `AX` et l'autre opérande placé dans dans un registre de deux octets ou dans un emplacement mémoire en utilisant la syntaxe :

```
MUL source
```

Sémantique.- Le produit de ces deux entiers est alors placé dans les registres `DX:AX` (puisqu'il peut occuper quatre octets), `AX` étant le mot de poids inférieur. L'indicateur de retenu `CF` indique si on doit tenir compte de `dx`.

Exemple.- Pour multiplier `1045h` par `111h`, on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov ax,1045
249C:0103 mov bx,111
249C:0106 mul bx
249C:0108 int3
249C:0109
-g
AX=5995 BX=0111 CX=0000 DX=0011 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0108 OV UP EI PL NZ NA PE CY
249C:0106 CC INT 3
-q
```

dont on vérifie le résultat (y compris le positionnement de l'indicateur de retenue).

Cas de la multiplication d'un mot par un octet.- On fait comme pour la multiplication de deux mots, l'octet étant placé dans `AL` et en mettant `AH` à zéro.

Exemple.- Pour multiplier `45h` par `1110h` on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov al,45
249C:0102 mov ah,0
249C:0104 mov bx,1110
249C:0107 mul bx
249C:0109 int3
249C:010A
-g
AX=9950 BX=1110 CX=0000 DX=0004 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0109 OV UP EI PL NZ NA PE CY
249C:0109 CC INT 3
-q
```

dont on vérifie le résultat.

4.2.6 Division euclidienne

4.2.6.1 Division de deux octets

Syntaxe.- Pour effectuer la divison euclidienne de a par b, entiers d'un octet, le dividende a doit être placé dans le registre `AL`, avec le registre `AH` mis à zéro, et le diviseur b placé dans un registre de un octet ou dans un emplacement mémoire, en utilisant la syntaxe :

```
DIV diviseur
```

Sémantique.- Le quotient est placé dans le registre `AL` et le reste dans le registre `AH`.

Exemple.- Pour effectuer la division euclidienne de 22 (ou `16h`) par 3 on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov al,16
249C:0102 mov bl,3
249C:0104 div bl
249C:0106 int 3
249C:0107
-g
AX=0107 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0106 NV UP EI PL NZ NA PO NC
249C:0106 CC INT 3
-q
```

dont on vérifie le résultat $q = 7$ et $r = 1$.

4.2.6.2 Division de deux mots

Syntaxe.- Pour effectuer la division euclidienne de l'entier a par l'entier b, tous deux d'une capacité maximale d'un mot, le dividende a doit être placé dans le registre `AX`, le registre `DX` devant être mis à zéro, et le diviseur b être placé dans un registre de deux octets ou dans un emplacement mémoire. On utilise la syntaxe :

```
DIV diviseur
```

Sémantique.- Le quotient q est alors placé dans le registre `AX` et le reste r dans le registre `DX`.

Exemple.- Pour effectuer la division euclidienne de `1050h` par `100h` on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov ax,1050
249C:0103 mov dx,0
249C:0106 mov bx,100
249C:0109 div bx
249C:010B int 3
249C:010C
-g
AX=0010 BX=0100 CX=0000 DX=0050 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010B NV UP EI PL NZ NA PO NC
```

```
249C:010B CC INT 3
-q
```

dont on vérifie le résultat $q = 10h$ et $r = 50h$.

Cas de la division euclidienne d'un mot par un octet.- Le dividende a doit être placé dans le registre `AX` et le diviseur b dans un registre de un octet ou un emplacement mémoire, toujours en utilisant la syntaxe :

```
DIV diviseur
```

le quotient q est placé dans le registre `AH` et le reste dans le registre `AL`.

Exemple.- Pour effectuer la division euclidienne de `1050h` par `51h` on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov ax,1050
249C:0103 mov bl,51
249C:0105 div bl
249C:0107 int 3
249C:0108
-g
AX=2D33 BX=0051 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0107 NV UP EI PL NZ NA PE NC
249C:0107 CC INT 3
-q
```

4.2.6.3 Division d'un mot double par un mot

Syntaxe.- Pour effectuer la division euclidienne de a par b, l'entier a ayant une capacité maximale de deux mots et l'entier b d'un seul mot, le dividende a doit être placé dans les registres `AX` et `DX`, les deux octets de poids fort étant dans `DX`, et le diviseur b dans un registre de deux octets ou dans un emplacement mémoire, toujours en utilisant la syntaxe :

```
DIV diviseur
```

Sémantique.- Le quotient q est placé dans le registre `AX` et le reste r dans le registre `DX`.

Exemple.- Pour effectuer la division euclidienne de `10502h` par `100h` on utilise le programme suivant :

```
C:>debug
-a
249C:0100 mov dx,1
249C:0103 mov ax,0502
249C:0106 mov bx,100
249C:0109 div bx
249C:010B int 3
249C:010C
-g
AX=0105 BX=0100 CX=0000 DX=0002 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=010B NV UP EI PL NZ NA PE NC
249C:010B CC INT 3
-q
```

4.3 Un exemple d'utilisation de la mémoire

Introduction.- Donnons un exemple de problème de programmation qui nous force réellement à avoir accès à la mémoire vive, c'est-à-dire pour lequel l'utilisation des registres n'est pas suffisante.

Le problème.- Considérons le polynôme suivant :

$$P(n) = n^3 + 5n^2 + 2n + 3$$

à coefficients entiers naturels. Nous voulons évaluer ce polynôme en un certain nombre de points (entiers naturels). Plus généralement les coefficients seront choisis.

Méthode.- Nous plaçons les coefficients $a_0 = 1$, $a_1 = 5$, $a_2 = 2$ et $a_3 = 3$ dans des emplacements mémoires ainsi que la valeur choisie de n. Nous évaluons le polynôme en utilisant la méthode de Hörner :

$$P(n) = (((0 + a_0).n + a_1).n + a_2).n + a_3.$$

Ceci donne lieu au programme suivant pour $n = 1$:

```
C:>debug
-a
249C:0100 mov ax, 1
249C:0103 mov [1000],ax
249C:0106 mov ax, 5
249C:0109 mov [1004],ax
249C:010C mov ax, 2
249C:010F mov [1008], ax
249C:0112 mov ax, 3
249C:0115 mov [1012],ax
249C:0118 mov ax, 1
249C:011B mov [1016],ax
249C:011E mov dx,0
249C:0121 mov ax, 0
249C:0124 add ax,[1000]
249C:0128 mov bx, [1016]
249C:012C mul bx
249C:012E add ax, [1004]
249C:0132 mul bx
249C:0134 add ax, [1008]
249C:0138 mul bx
249C:013A add ax, [1012]
249C:013E  int 3
249C:013F
-g
AX=000B BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=013E NV UP EI PL NZ NA PO NC
249C:013E CC INT 3
-q
```

dont on vérifie bien le résultat, à savoir 11 ou `Bh`, dans le registre `AX`.