CHAPITRE

4

Représentation des informations

I- Introduction :

Les informations traitées par l'ordinateur sont de différents types (nombres, textes, instructions, images, séquences d'images animées, sons) mais elles sont toujours représentées à la base, sous forme binaire. Une information élémentaire correspond donc à un chiffre binaire (0 ou 1) ; appelé **bit**. Le codage d'une information consiste à établir une correspondance entre sa représentation externe et sa représentation interne qui est une suite de bits.

L'arithmétique binaire peut être réalisée à partir de la logique symbolique à deux états (0 et 1). Les ordinateurs sont généralement organisés pour travailler sur une succession de groupe de bits appelés mots (words qui peuvent être de 8,16, 32 bits); ils peuvent représenter des nombres ou des lettres de l'alphabet, en fonction d'une loi de codage, qui fait correspondre une combinaison possible à un nombre ou une lettre.

On rappelle que la plus petite unité d'information binaire appelée bit ou digit, peut prendre la valeur 0 ou 1. L'association de 8 bits donne naissance à un octet. Les multiples de l'octet sont :

Le kilo Octets : 1024 Octets

Le Méga Octets : 1024 Kilo Octets

Le Giga Octets : 1024 Méga Octets

Le Tétra Octets : 1024 Giga Octets

II- Représentation des caractères :

Les données non numériques correspondent aux caractères alphanumériques : A, B, …, Z, a, b, …z, 0, 1, …, 9 et aux caractères spéciaux : ?, !, $,…etc. Le codage est réalisé par une table de correspondance propre à chaque code utilisé. Parmi les codes les plus connus, on peut citer :

BCD (Binary Coded Décimal), codage des caractères sur 6 bits.

ASCII (American Standard Code for Information Interchange), codage des caractères sur 7 bits .

EBCDIC (Extended Binary Coded Decimal Internal Code) codage des caractères sur 8 bits.

UNICODE : codage sur 16 bits.

Le codage est réalisé par une table de correspondance, propre à chaque code utilisé.

Caractère	BCD	ASCII	EBCDIC
0	**000000**	**0110000**	**11110000**
1	**000001**	**0110001**	**11110001**
2	**000010**	**0110010**	**11110010**
…	**…**	**….**	**….**
9	**001001**	**0111001**	**11111001**
A	**010001**	**1000001**	**11000001**
B	**010010**	**1000010**	**11000010**
C	**010011**	**1000011**	**11000011**

Figure 7 : Table de correspondance entre les différents codes

III- Représentation des numériques :

Les valeurs numériques manipulées dans l'ordinateur, peuvent être soit des entiers naturels, soit des entiers relatifs, soit des nombres fractionnaires. Ces nombres vont être représenté en binaire pour pouvoir les traiter correctement.

1. Représentation des entiers naturels :

La représentation d'un entier naturel est tout simplement sa représentation binaire, ce qui revient à faire sa conversion dans la base binaire.

Exemple :

$(4)_{10} = (100)_2$, $(13)_{10} = (1101)_2$

2. Représentation des entiers relatifs :

Les nombres relatifs se représentent simplement dans la base décimale en ajoutant le signe plus pour désigner un nombre positif et le signe moins pour désigner un nombre négatif devant la valeur absolue du nombre. Dans la base binaire, il fallait trouver des techniques d représentations adéquates.

Il y a trois techniques permettant de représenter ce type d'entier :

- **Bit de signe :**

Cette technique permet de représenter le nombre sous forme d'un mot de n bits en précisant un bit particulier comme bit d'indication de signe appelé *bit de signe*.

Par convention, un 0 représente le signe plus (+) et un 1 représente le signe (-).

Pour représenter un nombre relatif N en binaire, on procède comme suit :

- Représenter la valeur absolue de N en binaire sur (n-1) bits.
- Consacrer le $n^{ième}$ bit pour le signe.

Si N est positif alors le bit de signe sera positionné à 0.

Si N est négatif alors le bit de signe sera positionné à 1.

Exemple :

Représenter (-5) sur 8 bits :

$(5)_{10} = (\underline{\mathbf{0}}0000101)_2 \longrightarrow (-5)_{10} = (\underline{\mathbf{1}}0000101)_2$

Inconvénients :

Les inconvénients de cette méthode se manifestent au niveau de l'implémentation des opérations arithmétiques qui s'avèrent difficile, en plus de la double représentation binaire du zéro à savoir (**+0** : 0000) et (**-0** :1000).

- **Complément à 1:**

Cette technique consiste à représenter un nombre négatif par le complément à 1 de sa valeur absolue.

On appelle complément à 1 de N ($\alpha_0\ \alpha_1 \ldots \alpha_p$) le nombre $N = C_1(N)$ tel que pour $0 \leq i \leq p$,vérifie :

Si $\alpha_i = 1$ alors $\alpha'_i = 0$

Si $\alpha_i = 0$ alors $\alpha'_i = 1$

Exemple :

N=01101110 $\longrightarrow$ $\mathbf{C_1\ (N) =}$ (10010001)

Le codage en complément à 1 revient à inverser les bits à 1 en bits à 0 et vis vers ça.

Lorsqu'il s'agit d'un nombre binaire pure (positif), son interprétation en décimal est directe et son bit du poids fort est à 0 alors que pour un nombre binaire négatif, gardant aussi son bit du

poids fort à 1, ne peut être interpréter en décimal qu'on cherchant sa valeur absolue par le complément à 1.

Exemple :

$N= (\mathbf{0}111)_2 \longrightarrow N= (+7)_{10}$: *passage par interprétation directe*

$N= (\mathbf{1}010)_2 < 0 \longrightarrow$: *passage par le complément à 1 :* $C_1(N) = (\mathbf{0}101)_2 \longrightarrow$: *passage par interprétation directe* $C_1(N) =(+5)_{10} \longrightarrow N=(-5)_{10}$

Inconvénients :

Les inconvénients de cette méthode se manifestent au niveau de l'implémentation des algorithmes compliqués pour effectuer et rectifier des opérations arithmétiques, en plus du constat qui souligne que la somme d'un entier avec son complément est différente du zéro.

- **Complément à 2:**

Cette technique consiste à représenter un nombre négatif par le complément à 2 de sa valeur absolue. On appelle complément à 2 d'un nombre binaire N, le nombre binaire

$N' = C_1(N) +1$.

En résumé, pour trouver la représentation d'un nombre négatif ; il suffit de chercher la représentation en complément à deux de sa valeur absolue.

Exemples :

Représenter en complément à 2 sur 5 bits signés les nombres suivants :

-13 et -3

$(+13)_{10} = (01101)_2$

$C_1 (13) = (10010)_2 \longrightarrow C_2(13) = C_1(13) + 1 = 10010 + 1 = 10011$

Donc le complément à 2 de 13 est **10011**.

$(+3)_{10} = (00011)_2$

$C_1 (3) = (11100)_2 \longrightarrow C_2(3) = C_1(3) + 1 = 11100 + 1 = 11101$

Donc le complément à 2 de 3 est **11101**.

Remarques :

- La représentation en complément à 2 des nombres binaires signés conserve toujours le bit du poids fort du nombre comme étant un bit de signe.
- Il est important de fixer la taille du nombre signé.
- Un nombre binaire négatif ne peut pas être interprété directement en décimale : On doit d'abord chercher sa valeur absolue en lui appliquant le complément à 2 ; ensuite on convertie cette valeur en décimale à laquelle on attribue le signe moins (-).

3. Représentation des nombres fractionnaires:

Les nombres fractionnaires se représentent selon deux techniques :

- **Virgule fixe :**

La technique de virgule fixe représente un nombre fractionnaire sur un nombre n de bits dont on fixera la position de virgule entre deux bits consécutifs. Cette méthode fixe dés le début le nombre de bits de la partie fractionnaire. Elle présente comme inconvénients les phénomènes de troncatures et d'arrondissement qui mène à des problèmes de précision.

Exemple :

La représentation de $(7,30)_{10}$ avec la virgule fixe : sur 12 bits dont 4 bits sont réservés pour la partie fractionnaire.

$(7,30)_{10} \approx (111,01001100)_2$

On garde de cette valeur approximative 4 chiffres après la virgule, on obtient comme représentation binaire :

0	0	0	0	0	1	1	1	**0**	**1**	**0**	**0**

Partie entière **Partie Fractionnaire**

- **Virgule flottante :**

Pour résoudre le problème de l'approximation qui résulte de la technique de la virgule fixe, et pour représenter un maximum de bits significatif, on adopte la technique de représentation en virgule flottante se basant sur le transfert de la virgule vers le premier chiffre significatif (le premier 1 de la valeur entière).Ensuite, on prend le maximum des bits significatifs, ce qui revient à « flotter » la virgule d'une position à une autre, d'où vient son nom.

Une représentation normalisée connue pour représenter un nombre fractionnaire est la suivante :

$(-1)^{s}$ 1, M 2^{E-127}

S : est le signe du nombre (0 : positif, 1 négatif)

1 : est le premier 1 de la partie entière

M : est la mantisse: constitue la partie fractionnaire après la transformation

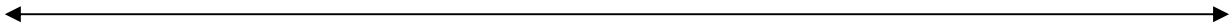
E : est l'exposant du à cette transformation

2 : est la base

Représentation sur la machine :

La représentation selon la technique virgule flottante s'effectue sur n bits, répertorié en trois compartiments.

n bits



S	E	M

Cette représentation est faite soit en *simple précision* avec 32 bits dont 1 bit de signe, 8 bits pour l'exposant et le reste pour la mantisse ; soit en *double précision* avec 64 bits dont 1 pour le signe, 16 pour l'exposant et le reste pour la mantisse.

Exemple :

La représentation de $(7,30)_{10}$ avec la virgule flottante en simple précision :

$(7,30)_{10} \approx (111,01001100110011001100110011)_2$

L'écriture normalisée est :

$(-1)^0$ 1,11010011001100110011001100110011 * 2^2 =

$(-1)^0$ 1,11010011001100110011001100110011 * $2^{129-127}$

Donc S=0

E=129 = (10000001)

M= 11010011001100110011001100110011

On aboutit finalement à cette représentation :

0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1