Chapitre 3 : Codage des symboles et des nombres non-entiers

1. Les entiers négatifs

Les entiers négatifs peuvent être codés selon 3 méthodes : signe et valeur absolue, complément à 1 (complément logique) et complément à 2 (ou arithmétique).

1.1. Signe et valeur absolue

Les entiers sont codés de la façon suivante : ± valeur absolue. On sacrifie un bit pour représenter le signe. On représentera le signe + par un 0 et le signe – par un 1. On peut ainsi avec un mot de k bits, coder les entiers positifs ou négatifs N, tel que N est dans l'intervalle :

$$-(2^{k-1}-1) \leq N \leq +(2^{k-1}-1)$$

L'inconvénient de cette méthode est que le 0 a deux représentation distinctes : 000..0 et 10..0, soit +0 et -0. Aussi, les opérations arithmétiques sont compliquées à cause du bit de signe qui doit être traité à part.

1.2. Les compléments à 1 et à 2

On calcule le complément logique (complément à 1) en remplaçant pour les valeurs négatives, chaque bit à 0 par 1 et vice-versa dans la valeur absolue. Le complément à 2 (arithmétique) est obtenu par addition du (complément à 1) +1.

En complément à 1 aussi et pour k bits, on a l'intervalle suivant :

$$-(2^{k-1}-1) \leq N \leq +(2^{k-1}-1),$$

Mais en complément à 2 l'intervalle est :

$$-2^{k-1} \leq N \leq +(2^{k-1}-1)$$

(**c à d :** on a une valeur de plus car elle évite la double représentation du zéro)

Exemple : la représentation de (-6) sur 4 bits est :

- En signe et valeur absolue : **1110**
- En complément à 1 : **1001**
- En complément à 2 : **1001 + 1** = **1010**

Question : quel est l'intervalle des nombres signés qu'on peut représenter sur 4 bits pour les 3 méthodes ?

- On remarque que le bit le plus à gauche (bit de signe), dans les trois méthodes, est toujours à 1 si le nombre est négatif et est à 0 si le nombre est positif.
- En complément à 1 et à 2, les opérations arithmétiques sont avantageuses car la soustraction d'un nombre se réduit à l'addition de son complément. On n'utilisera que des circuits réalisant l'addition et il n'y a pas de traitement particulier pour le bit de signe.
- Dans une addition en complément à 1, une retenue générée par le bit de signe doit être ajoutée au résultat obtenu. Par contre, en complément à 2, on ignore tout simplement cette retenue.

Exemple : la soustraction de nombres sur 4 bits :

Décimal	**Signe + Val.abs**	**Complément à 1**	**Complément à 2**
+ 7	**0 1 1 1**	**0 1 1 1**	**0 1 1 1**
- 6	**+ 1 1 1 0**	**+ 1 0 0 1**	**+ 1 0 1 0**
= + 1	**= ? 1 0 1**	**= 1 0 0 0 0**	**= 1 0 0 0 1**

Remarque : En complément à 2, un dépassement de capacité ne se produit que si les retenues générées juste avant le bit de signe et par le bit de signe lui-même sont différentes.

Exemple : addition en complément à 2 sur 3 bits :

Décimal	**Complément à 2**
(- 4)	**1 0 0**
(- 1)	**1 1 1**
= - 5	**= ~~1~~ 0 1 1 ≠ - 5 , donc le résultat est faux**

2. Les nombres fractionnaires

2.1. Représentation en virgule flottante

Nous savons qu'il est nécessaire de stocker des données dans les machines. Ainsi le nombre 9,750 se trouvera mémorisé sous la forme suivante: 1001,11. Toutefois cette expression binaire ne suffit pas à définir totalement notre donnée car il n'y a aucune indication sur la valeur du poids binaire affecté aux différents bits, d'où la notion de virgule.

En utilisant cette notion de virgule, notre nombre peut s'écrire de la manière suivante :

- $N = 1001{,}11 \times 2^0$
- $N = 100{,}111 \times 2^1$
- $N = 10{,}0111 \times 2^2$
- $N = 1{,}00111 \times 2^3$
- $N = 0{,}100111 \times 2^4$

Cette dernière expression présente l'avantage de représenter la grandeur par un nombre inférieur à 1 multiplié par une puissance de 2. L'exposant 4 (100 en binaire) est bien entendu représentatif de la position de la virgule. Donc pour définir totalement notre information (9,750) il faudra dans ce système de représentation deux termes : le terme 100111 appelé **Mantisse** et le terme 100 appelé **Exposant**.

Donc, la représentation en virgule flottante consiste à représenter les nombres N sous la forme suivante :

$N = M \times B^E$ avec :

- B : base (dans notre cas, on étudie B=2)
- M : Mantisse
- E : exposant

L'exposant est un entier, la mantisse un nombre purement fractionnaire (n'ayant pas de chiffres significatifs à gauche de la virgule). Celle-ci est normalisée, c'est-à-dire qu'elle comporte le maximum de chiffres significatifs: le premier bit à droite de la virgule et à 1 (ex : 0.101110). A l'exception de la valeur 0 (qui est en général représentée par le mot 00…0), on a donc toujours:

$$\mathbf{(0.1)_2 \leq |M| < (1)_2 \text{ soit } 0.5 \leq |M| < 1_{10}}$$

L'exposant et la mantisse doivent pouvoir représenter des nombres positifs ou négatifs, donc pourraient être codés sous la forme signe+val.abs, complément à 1 ou complément à 2. Souvent, la mantisse est de la forme **signe+val.abs** et l'exposant est sans signe, mais **biaisé** (ou décalé).

Exemple :

SM	E	M

Où :

- **SM** : est le signe de la mantisse,
- **E** : est l'exposant biaisé
- **M** : la mantisse.

Avec 4 bits, par exemple, on peut représenter $2^4 = 16$ valeurs de E, qui vont de 0 à 15. On peut faire correspondre les 8 premières valeurs (de 0 à 7) à un exposant < 0 et les 8 suivants (de 8 à 15) à un exposant ≥ 0. Un exposant nul est ainsi représenté par la valeur 8, un exposant égal à +1 par la valeur 9, un exposant égal à -1 par la valeur 7. On dit que le biais est égal à 8. C'est la valeur qu'il faut soustraire à l'exposant biaisé (de 0 à 15) pour obtenir l'exposant effectif (de -8 à +7).

Exemple : Représentation d'entiers signés sur 3 bits (exposant codé sur 3 bits →8 valeurs possibles entre 0 et 7)

Décimal	Signe et Val.abs	Complément à 1	Complément à 2	Représentation biaisé
+ 3	0 1 1	0 1 1	0 1 1	1 1 1
+ 2	0 1 0	0 1 0	0 1 0	1 1 0
+ 1	0 0 1	0 0 1	0 0 1	1 0 1
0	0 0 0 // 1 0 0	0 0 0 // 1 1 1	0 0 0	1 0 0
- 1	1 0 1	1 1 0	1 1 1	0 1 1
- 2	1 1 0	1 0 1	1 1 0	0 1 0
- 3	1 1 1	1 0 0	1 0 1	0 0 1
- 4	------	------	1 0 0	0 0 0

On peut remarquer que la représentation biaisée est identique au complément à 2, à l'exception du bit de signe, qui est inversé (représentation biaisée : bit de signe à 1 → valeur ≥ 0, bit à 0 → valeur < 0).

L'exposant détermine l'intervalle des nombres représentables et la taille de la mantisse détermine la précision de ces nombres.

2.1.1. Les opérations arithmétiques en virgule flottante

Pour la multiplication, il suffit d'additionner les exposants, de multiplier les mantisses et de re-normaliser le résultat si nécessaire.

Exemple : $(0.2 \times 10^{-3}) \times (0.3 \times 10^{7}) = ?$

- Addition des exposants : $-3 + 7 = 4$
- Multiplication des mantisses : $0.2 \times 0.3 = 0.06$
- Résultat avant normalisation : 0.06×10^{4}
- Résultat normalisé : 0.6×10^{3}

Pour **la division**, il suffit de soustraire les exposants et diviser les mantisses et de re-normaliser le résultat si nécessaire.

Pour **l'addition**, il faut que les exposants aient la même valeur ; on est donc obligé de dénormaliser la plus petite valeur pour amener son exposant à la même valeur que celui du plus grand nombre. Après avoir additionné les mantisses, une normalisation peut être nécessaire.

Exemple : $(0.300 \times 10^4) + (0.998 \times 10^6) = ?$

- Dénormalisation : $0.300 \times 10^4 \rightarrow 0.003 \times 10^6$
- Addition des mantisses : 0.003 + 0.998 = 1.001
- Normalisation du résultat : $1.001 \times 10^6 \rightarrow 0.1001 \times 10^7$

La soustraction s'effectue de la même façon que l'addition, sauf que l'on doit effectuer la soustraction et non plus l'addition des mantisses.

2.2. Représentation en virgule fixe

La représentation de nombre en virgule flottante n'est pas la seule imaginable. Il existe la représentation de nombres en virgule fixe. La différence de base avec la représentation en virgule flottante est, comme son nom l'indique, le nombre de chiffres après la virgule (le plus à droite) est toujours le même, donc la précision des nombres fractionnaires représentés par virgule fixe est toujours la même.

Exemple :

Soit : $(25,75)_{10} = (11001,110)_2$

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

Dans cette configuration par exemple, la position de la virgule est **fixe** (entre le 3ème et le 4ème bit), mais comme la virgule n'est pas réellement visualisée ou représentée, à la base la case la plus à droite représente le poids 2^0: ce qui est évidemment faux dans notre cas. Cette représentation suppose la multiplication implicite de ce nombre par $\mathbf{2^{-3}}$ pour obtenir la valeur exacte. Le terme **-3** est représentatif du positionnement fixe de la virgule. Il devra impérativement être mémorisé dans la machine.

3. Les différents codages

3.1 Le code BCD :

Abréviation de Binary Coded Decimal en anglais et DCB (Décimal Codé Binaire). Ce code cherche à concilier les avantages du système décimal et du code binaire. Il est surtout utilisé pour l'affichage de données décimales (calculatrices par exemple). A chaque chiffre du système décimal, on fait correspondre un mot binaire (de quatre bits en général).

Pour coder un nombre décimal en BCD, on va coder séparément chaque chiffre du nombre de base dix en Binaire.

Exemple : (BCD sur 4 bits) : 1985 = 0001 1001 1000 0101(BCD)

Remarque :

- Le nombre codé en BCD ne correspond pas au nombre décimal converti en binaire naturel. Le codage décimal BCD est simple, mais il n'est pas possible de faire des opérations mathématiques directement dessus.
- . Il existe plusieurs types de codes BCD, mais le plus connu est celui présenté dans cette section.

3.2 Le code EBCDIC : (Extended Binary Coded Decimal Interchange) :

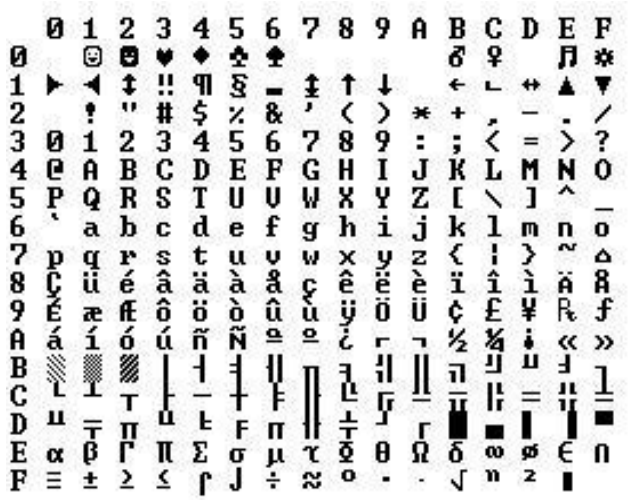
Ce code est utilisé principalement par IBM. Il est représenté sur 8 bits et est utilisé dans le codage de caractère, c'est-à-dire, pour chaque caractère est associé son code EBCDIC.

Exemple : code du caractère **A** (en majuscule) en **EBCDIC = 11000001** code du caractère **0** en **EBCDIC = 11110000**

3.3 Le code ASCII : *American Standard Code for Information Interchange*

Ce code propose des extensions différentes selon le " code de page ". Le code de page 850 est un jeu de caractères " multilingue " alors que le code de page 864 définit le jeu de caractères arabes, le code de page 437 définit le jeu de caractères français...etc

La table des codes ASCII



Remarque : La table des codes ASCII ci-dessus, affiche les caractères imprimables et non les codes de contrôle. En effet les caractères dont les codes sont 10, 13 et 27 en décimal, représentent respectivement Line Feed (Aller à la ligne), Carriage Return (Retour Chariot) et Scape (Escape).