REPRESENTATION DES NOMBRES REELS ET OPERATIONS

1. REPRESENTATION DES NOMBRES REELS

1.1 Les nombres signés

La solution la plus simple consiste à réserver un digit binaire (bit) pour ce signe. Les autres bits représentant une valeur absolue du nombre. La convention retenue impose de mettre le premier bit (qu'on appelle aussi bit de poids fort) à **0** pour repérer un nombre positif et à **1** pour un nombre négatif. On parle alors de donnée signée quand on utilise cette convention.
1 $1100_{(2)} = -12_{(10)}$
0 $1100_{(2)} = 12_{(10)}$
Toutefois, une telle représentation des nombres signés entraînerait un traitement spécial du signe et des circuits électroniques différents selon que l'on veuille réaliser des divisions ou des soustractions.
Cet inconvénient est résolu par une autre forme de représentation des nombres négatifs dite représentation en complément ou encore représentation sous forme complémentée.

1.2 Complément d'un nombre

La représentation des nombres sous la forme en complément se fait selon deux modes qui ne s'appliquent en principe qu'aux nombres négatifs. Le premier mode est le complément restreint (complément à 1) et le second est le complément vrai (ou complément à 2).

1.2.1 Complément à 1 (Complément restreint)

Le complément restreint d'un nombre binaire s'obtient par la simple inversion des valeurs des bits.
Exemple : 1100 devient 0011

1.2.2 Complément à 2 (Complément vrai)

Le complément à 2 (complément vrai) d'un nombre s'obtient en ajoutant 1 au complément restreint.
Exemple : le complément restreint de 0010111 est 1101000

```
    1101000
+   0000001
-------------

    1101001 (est le complément vrai de 0010111)
```

1.3 Les nombres réels

1.3.1 Notion de mot

Les systèmes informatiques manipulent les informations binaires et travaillent en général sur une longueur fixe de bits que l'on appelle mot. Suivant la machine, la taille du mot sera différente, les tailles classiques étant 16, 32, 64 bits.
La représentation des nombres à l'intérieur de la machine se fait selon deux méthodes : La virgule fixe utilisée avant les processeurs de type Pentium, et la virgule flottante (Norme IEEE* 754).
*Institute of Electrical and Electronic Engineers

1.3.2 Virgule fixe

Un nombre en virgule fixe est une valeur munie ou non d'un signe enregistré comme un entier binaire. Un tel nombre est dit en virgule fixe car le soin de placer la virgule, revient au programmeur. On décale la puissance « 0 » vers la gauche et on utilise ensuite des puissances négatives. La virgule n'apparaît pas dans le stockage du nombre, mais sera placée par le programmeur qui utilise ce nombre, décomposant ainsi la valeur lue en une partie entière et une partie fractionnaire.

Exemple sur un mot de 16 bits :

$000000110011.1011_{(2)} = 1*2^5 + 1*2^4 + 1*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-3} + 1*2^{-4}$

$= 32 + 16 + 2 + 1 + 0,5 + 0,125 + 0,0625$

$000000110011.1011_{(2)} = 51,6875_{(10)}$

Cette notation implique d'avoir beaucoup de chiffres après la virgule pour obtenir une bonne approximation.

Que devient ce nombre si on déplace la virgule de 2 rangs vers la gauche ?

Même question si on la déplace de 1 rang vers la droite.

1.3.4 Virgule flottante (sur 32 bits)

Cette notation consiste à écrire les nombres sous la forme :

$$N = (-1)^{S} * \mathbf{1,M} * 2^{(E-127)} \text{ ou } N = (-1)^{S} * \mathbf{(1+M)} * 2^{(E-127)}$$

Où **S** représente le signe, **M** la mantisse* et **E** l'exposant.

Dans un mot de 32 bits, on attribue un certain nombre de bits à chaque terme.

31	30 23	22 0
Signe	**Exposant**	**Mantisse**
1 bit	8 bits = 1 octet	23 bits

** le terme de mantisse correspond à la différence entre un nombre et sa partie entière.*

L'exposant sera établi de telle manière que la mantisse soit de la forme « 1,..» ou encore « **1**, **M** », c'est-à-dire de telle manière que la virgule soit située juste après le premier bit égal à "1". On parle alors de **mantisse normalisée**. Cela nous permet d'omettre ce premier bit qui vaut toujours **1** (on parlera de bit caché) et donc de gagner un bit supplémentaire pour la précision.
La mantisse s'exprimera alors comme un nombre réel à virgule fixe où la virgule serait fixée en tête de la séquence. (On doit donc en tenir compte dans le calcul et ajouter **1** à la valeur de **M**)

Exemple1 : **01000100000001101100000000000**$_{(2)}$ = +525,5$_{(10)}$

- Le bit de poids fort « **0** » donne le signe : **+**
- Les bits 23 à 30 donnent la valeur de l'exposant (la puissance de 2), soit : **10001000**$_{(2)}$ = **136**$_{(10)}$. (Remarque :$2^{(136-127)} = 2^9 = 512$)
- Les bits 0 à 22 donnent la valeur de la mantisse, soit :
  00000110110000000000$_{(2)}$ $= 1*2^{-6} + 1*2^{-7} + 1*2^{-9} + 1*2^{-10}$
  $= \mathbf{0,026367188}_{(10)}$
- $N = (-1)^{S} * \mathbf{(1+M)} * 2^{(E-127)} = (-1)^{0} * \mathbf{1,026367188} * 2^{(136-127)} = +525,5_{(10)}$

Exemple 2 : -3333$_{(10)}$ = **11000101010100000101000000000000**$_{(2)}$

- Le signe prend donc la valeur **S=1**, soit le signe **-**
- Rechercher l'expression de 3333 en binaire : 3333$_{(10)}$ = 1101 0000 0101$_{(2)}$
- En notation normalisée, on obtient donc :**1,10100000101**
- La mantisse prend donc la valeur **10100000101**, soit :
  $1*2^{-1} + 1*2^{-3} + 1*2^{-9} + 1*2^{-11} = \mathbf{0,6274414063}$
- 3333 est compris entre 2^{11} (2048) et 2^{12} (4096)
- Donc, **E**-127 = **11**, l'exposant vaut donc (**11**+127) = **138**, soit **1000 1010**
- Vérification :
  $N = (-1)^{S} * \mathbf{(1+M)} * 2^{(E-127)} = (-1)^{1} * \mathbf{(1+0,6274414063)} * 2^{(138-127)}$
  N = -3333$_{(10)}$

Remarque : La norme IEEE-754 a prévu certaines configurations particulières afin de coder quelques cas remarquables et autres monstruosités mathématiques

http://www.arcanapercipio.com

Signe	Exposant	Mantisse	Signification
0 / 1	tous les bits à 0	tous les bits à 0	**Zéro positif (0^+) / Zéro négatif (0^-)** A noter que de par cette double configuration du zéro, la norme prévoit explicitement que la comparaison $0^+ = 0^-$ soit toujours vraie.
0 / 1	tous les bits à 1	tous les bits à 0	**$+\infty$ / $-\infty$** **Attention !** Ce terme est un peu trompeur car il pourrait indiquer que ces deux configurations ne codent que les résultats d'opérations improbables comme, par exemple, une division par zéro. En réalité, tout résultat d'opération donnant un nombre supérieur (en valeur absolue) au plus grand nombre représentable par le format sera également considéré comme infini, bien que mathématiquement parlant, il ne le soit pas.
0 / 1	tous les bits à 1	non nulle	**Not A Number (NaN)** Terme assez explicite afin d'indiquer le résultat d'une opération aussi amusante que $0/0$, $\sqrt{(-1)}$ ou encore $0x\infty$.

1.4 Applications

- *Déterminer le nombre décimal représenté par le mot* ***0 10001001*** *11110111000000000000000*
- *Comment s'écrit le nombre 1 sur 32 bits ?*
- *Déterminer le nombre décimal représenté par le mot* ***1 10011111*** *00001100001110001000110*
- *Comment s'écrit le nombre 20 sur 32 bits ?*
- *Quel est le plus grand nombre que l'on peut coder en 32 bits?*
- *Quel est le plus petit nombre que l'on peut coder en 32 bits?*

3. OPERATIONS

3.1 Addition

3.1.1 Addition de deux nombres positifs

L'addition binaire est analogue à l'addition décimale. Il faut commencer par le bit de poids faible en utilisant les relations suivantes :

0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 0 avec un report de **1**

Exemple : **1 1**

1er nombre $(+27)_{10}$		**0** 0 0 1 1 0 1 1
2ème nombre $(+12)_{10}$	+	**0** 0 0 0 1 1 0 0
Résultat $(+39)_{10}$		**0** 0 1 0 0 1 1 1

Application: ajouter $45_{(10)}$ à $73_{(10)}$

3.1.2 Addition de deux nombres de signes contraires (soustraction)

Lorsqu'un nombre est négatif, on utilise sa représentation complémentée. Ici, on cherchera le complément vrai (complément à 2).

Deux cas sont à envisager :

- **La valeur absolue du nombre positif est supérieure à celle du nombre négatif.**

Effectuons l'addition des nombres suivants :
$(+17)_{10}$ = **0** 0010001 et $(-12)_{10}$ = % **1** 0001100
$(-12)_{10}$ étant négatif, il faut le remplacer par son complément à 2.
Pour cela, écrivons d'abord le complément à 1 de $(-12)_{10}$:
$(-12)_{10}$ = **1**1110011
Puis écrivons le complément à 2 de $(-12)_{10}$ en lui ajoutant 1 :
$(-12)_{10}$ = **1** 1110100
Effectuons l'addition :

1er nombre	**0** 0 0 1 0 0 0 1
2ème nombre	+ **1** 1 1 1 0 1 0 0
Résultat	1 **0** 0 0 0 0 1 0 1

Remarque : Nous avons additionné les bits de signe et la retenue ; cela peut entraîner, comme dans le cas présent, un débordement qui est toujours à rejeter. La somme étant positive, le résultat est en notation exacte :

0 0000101 = $(+5)_{10}$

Application: soustraire $45_{(10)}$ à $73_{(10)}$

- **La valeur absolue du nombre positif est inférieure à celle du nombre négatif.**

Effectuons l'addition des nombres

$(-17)_{10}$ = **1** 0010001 et $(+12)_{10}$ = **0** 0001100

Le complément à 2 de $(-17)_{10}$ est : **1** 1101111

L'écriture de l'addition est alors :

1er nombre	**1** 1 1 0 1 1 1 1
2ème nombre	+ **0** 0 0 0 1 1 0 0
Résultat	**1** 1 1 1 1 0 1 1

Cette somme est négative, le résultat est le complément à 2 du total cherché qui s'écrit en notation exacte : **1** 0 0 0 0 1 0 1 = $(-5)_{10.}$

Application: soustraire $85_{(10)}$ à $53_{(10)}$

3.1.3 Addition de deux nombres négatifs

Effectuons l'addition de $(-17)_{10}$ et $(-12)_{10}$.

Les compléments à 2 sont : $(-17)_{10}$ = **1** 1101111 et $(-12)_{10}$ = **1** 1110100

L'écriture de l'addition est alors :

1er nombre	**1** 1 1 0 1 1 1 1
2ème nombre	+ **1** 1 1 1 0 1 0 0
Résultat	1 **1** 1 1 0 0 0 1 1

Cette somme est négative, et on rejette le débordement. Le résultat est le complément à 2 du résultat cherché qui s'écrit en notation exacte :

1 0011101 = $(-29)_{10.}$

3.3 Multiplication

Le principe est le même qu'en base 10. On utilise les relations suivantes :

0 x 0 = 0 ; 0 x 1 = 0 ; 1 x 0 = 0 ; 1 x 1 = 1

Exemple :

				1	0	1	1	0	1	
			x		1	0	0	1	1	
				1	0	1	1	0	1	
			1	0	1	1	0	1	.	
1	0	1	1	0	1	.	.	.	.	
1	1	0	1	0	1	0	1	1	1	

		4	5
x		1	9
	4	0	5
	4	5	.
	8	5	5

Application: multiplier $33_{(10)}$ à $13_{(10)}$

3.4 Division

Le principe est le même qu'en base 10 (si on se souvient comment on fait une division Euclidienne !) Dividende = Diviseur x Quotient + Reste

	1	1	0	1	1	0	1	0	1	0
-	1	0	1	0			1	0	1	
	0	0	1	1	1					
	-	0	0	0	0					
			1	1	1	0				
		-	1	0	1	0				
			0	1	0	0				

	5	4	1 0
-	5	0	5
	0	4	

$110110_{(2)} = 1010_{(2)}$ x $101_{(2)}$ + $100_{(2)}$

54 = 10 x 5 + 4

Application: faire la division de $893_{(10)}$ par$12_{(10)}$

5. APPLICATIONS

5.1 **Application 1 :** Effectuer les opérations suivantes (vérifier en décimal)

```
  001          111          11010000
+ 110        +  10        + 01011100
-----        -----        --------
```

```
  001          1101
+ 110        + 1110
+ 111        + 1110
-----        -----
```

5.2 **Application 2 :** Effectuer les opérations suivantes (vérifier en décimal)

```
  101          1010
x  10        x  10
-----        -----
```

```
  10111        101101
x   111      x  1011
-------      -------
```

5.3 **Application 3**

a) Donnez le complément à 2 des nombres binaires suivants :

0101 1011 1100011 00100110

b) A partir des résultats précédents, effectuer les soustractions :

```
    1010
-   0101
```

```
    1111
-   1011
```

```
   10100111
-  00100110
```

```
   1110001
-  1100011
```