# Représentation et codage des entiers relatifs et des nombres réels

## Table des matières



## 1) Les additions en binaire

**Rappel :**

$$\begin{array}{rr} & 27 \\ + & 19 \\ \hline = & 46 \end{array}$$

Donc cet exemple, étant donné que 7+9=16 et que 16 > 9 le plus grand des chiffres associés à la base 10 (0,1,2,3,4,5,6,7,8,9), nous positionnons 6 en dessous du 9 et nous reportons une unité au-dessus du 2 et ensuite nous additionnons 1+2+1=4

**En binaire :** il faut reproduire exactement le même protocole.

**Exemple :**

$$\begin{array}{rr} & 010101 \\ + & 111111 \\ \hline = & 1010100 \end{array}$$

**Explication** .....

.....

**Exercice :**

$$\begin{array}{rr} & 1011 \\ + & 0110 \\ \hline = & \ldots\ldots\ldots\ldots \end{array} \qquad \begin{array}{rr} & 0111 \\ + & 1110 \\ \hline = & \ldots\ldots\ldots\ldots \end{array} \qquad \begin{array}{rr} & 11010101 \\ + & 00000001 \\ \hline = & \ldots\ldots\ldots\ldots \end{array} \qquad \begin{array}{rr} & 11010100 \\ + & 11111111 \\ \hline = & \ldots\ldots\ldots\ldots \end{array}$$

# 2) Représentation des entiers relatifs

## 2.1) Contexte

Pour coder un entier relatif, il est nécessaire de coder les informations tout en respectant les propriétés mathématiques des entiers signés. Pour cela, il faut :

- s'assurer que -(-a) = a
- s'assurer que a + (-a) = 0

## 2.2) En binaire : codage en complément à 2

L'opposé d'un nombre est obtenu en 2 étapes :

- Première étape : inverser tous les bits du nombre
- Deuxième étape : ajouter 1 au résultat

⇒ On obtient l'opposé du nombre

**Exemple :** prenons le cas de +43, nous savons que $(+43)_{10} = (00101011)_2$ (cf chapitre 1).

↦ On inverse et on obtient : $(11010100)_2$

↦ On ajoute 1 :

$$\begin{array}{rr} & 11010100 \\ + & 00000001 \\ \hline = & 11010101 \end{array}$$

↦ Donc $(-43)_{10} = (11010101)_2$

**A faire :** vérifiez qu'en binaire (-43)+(43)=0 et que -(-43)=43.

## 2.3) Exercices

1. Convertir $(7)_{10}$ en binaire et en déduire $(-7)_{10}$ en binaire également.
2. Convertir $(35)_{10}$ en binaire et en déduire $(-35)_{10}$ en binaire également.

# 3) Représentation et codage des nombres réels

## 3.1) De la base 2 vers la base 10

**Rappel :** $(108,893)_{10} = 1 \times 10^2 + 0 \times 10^1 + 8 \times 10^0 + 8 \times 10^{-1} + 9 \times 10^{-2} + 3 \times 10^{-3}$

De même :

$(10,1011)_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$

$(10,1011)_2 = 1 \times 2 + 0 \times 1 + 1 \times 0.5 + 0 \times 0.25 \times 1 \times 0.125 + 1 \times 0.0625$

$(10,1011)_2 = (2.6875)_{10}$

**Prolongement :** traduire $(4B,8)_{16}$ de la base 16 vers la base 10

**Exercice :** traduire $(11,101)_2$ de la base 2 vers la base 10

## 3.2) De la base 10 vers la base 2

**Théorie** :

Pour la partie entière, on utilise la méthode des divisions successives vue précédemment. Pour la partie fractionnaire, on utilise la méthode des multiplications successives.

**Exemple : voir tableau 1**

| | | | |
|---|---|---|---|
| | 2,625 | → | la partie entière est :$(2)_{10} = (10)_2$ |
| On en extrait | ↓ | | |
| | 0,625 | | |
| × | 2 | | |
| | 1,25 | → | On sort : 1 |
| On en extrait | ↓ | | |
| | 0,25 | | |
| × | 2 | | |
| | 0,5 | → | On sort : 0 |
| On en extrait | ↓ | | |
| | 0,5 | | |
| × | 2 | | |
| | 1 | → | On sort : 1 |

En lisant du haut vers le bas, on obtient :
Donc le résultat est : $(2,625)_{10} = (10,101)_2$

Tableau 1 – Conversion de 2,625 en binaire

**Exercice :** traduire en binaire $(8,3125)_{10}$, $(0,546875)_{10}$,$(2,921875)_{10}$.
**Problème :** traduire en binaire $(0,4)_{10}$.

## 3.3) Codage en virgule flottante

**Représentation normalisée :** tout nombre binaire $(N)$ peut se représenter comme ceci :

$$N = 1, mantisse \times 2^{exposant}$$

Exemple : $(1011,101)_2 = 1,0011101 \times 2^3$ et $(0,0101)_2 = 1,01 \times 2^{-2}$

**Codage** : chaque nombre peut se coder de la manière suivante :

$$Signe+Partie\ exposant+Mantisse\ avec \begin{cases} Signe : 1\ bit ; \\ Partie\ exposant : décalage + exposant\ permettant\ une\ représentation\ normalisée\ du\ nombre ; \\ Mantisse : partie\ fractionnaire\ du\ nombre. \end{cases}$$

Explications pour la partie signe : On utilise la convention 0 pour les nombres positifs et le 1 pour les nombres négatifs)

Explications pour la partie exposant :

Pour stocker l'exposant de la représentation normalisée du nombre, il faut stocker la valeur absolue de l'exposant ainsi que son signe. Pour cela, on applique un décalage à l'exposant trouvé et on stocke cette valeur décalée. La valeur du décalage dépend du nombre $n$ de bits utilisés pour cette partie.

$$décalage = 2^{n-1} - 1$$

Donc il faudra stocker dans la partie exposant le décalage et l'exposant
Exemple : pour une partie exposant stockée sur 5 bits :

$1011,101101 = 1,011101101 \times 2^3$ Le décalage vaut $2^{5-1} - 1 = 15$, l'exposant vaut $+3$ donc la valeur stockée sur ces 5 bits vaut 15+3=18.

Explications pour la partie mantisse :

À une exception près, tous les nombres ont une représentation normalisée sous la forme : $1, mantisse \times 2^{exposant}$

Par conséquent, il n'est pas nécessaire de stocker le 1 situé à gauche de la virgule. On économise ainsi un précieux bit pouvant être utilisé à meilleur escient.

Remarque : comme indiqué ci-dessus, il existe une exception : 0 (zéro). Pour cette valeur, il sera nécessaire d'utiliser une combinaison binaire spéciale.

**Exemples :**

Exemple 1 : coder 2007 sur 12 bits repartis de la manière suivante :

- 1 bit pour le signe ;
- 5 bits pour la partie exposant ;
- 6 bits pour la partie mantisse.

- première étape : convertir le nombre en binaire :
  $(2007)_{10} = (11111010111)_2$
- deuxième étape : obtenir la représentation normalisée :
  $(11111010111)_2 = (1,1111010111 \times 2^{10})_2$
- troisième étape : calculer la valeur stockée dans la partie exposant. Tout d'abord, il faut le décalage. Pour cela, il faut regarder sur combien de bits est codée la partie exposant. Ici, elle est codée sur 5 bits. Donc, le décalage vaut $2^{5-1} - 1 = 15$. Ensuite, on ajoute au décalage l'exposant et on obtient la partie stockée dans la partie exposant : décalage +exposant=15+10=25.Ensuite, on traduit en binaire. $(25)_{10} = (11001)_2$.
- quatrième étape : on reporte la matisse dans la partie correspondante.
- Ce qui nous donne :


Exemple 2 : coder $(-0,28125)_{10}$ sur 12 bits repartis de la même manière que dans l'exemple 1.

- première étape :
  $(-0,28125)_{10} = (-0,01001)_2$
- deuxième étape : obtenir la représentation normalisée :
  $(-0,01001)_2)_2 = (-1,001 \times 2^{-2})2$
- troisième étape : calculer la valeur stockée dans la partie exposant. Tout d'abord, il faut le décalage. Pour cela, il faut regarder sur combien de bits est codée la partie exposant. Ici, elle est codée sur 5 bits. Donc, le décalage vaut $2^{5-1} - 1 = 15$. Ensuite, on ajoute au décalage l'exposant et on obtient la partie stockée dans la partie exposant : décalage +exposant=15-2=13.Ensuite, on traduit en binaire. $(13)_{10} = (01101)_2$.
- quatrième étape : on reporte la matisse dans la partie correspondante.
- Ce qui nous donne :


**Exercices :**

1. Donner la représentation en virgule flottante de $(0,9375)_{10}$ sur 11 bits repartis de la manière suivante :
   - 1 bit pour le signe ;
   - 4 bits pour la partie exposant ;
   - 6 bits pour la partie mantisse.
2. Donner la représentation en virgule flottante de $(-3,625)_{10}$ sur 10 bits repartis de la manière suivante :
   - 1 bit pour le signe ;
   - 3 bits pour la partie exposant ;
   - 6 bits pour la partie mantisse.

## 3.4) Norme IEEE 754

Cette norme définit deux types de nombres en virgule flottante : simple précision (32 bits) et double précision (64 bits).

**Simple précision**

taille totale : signe(1 bit)+partie exposant(8 bits)+mantisse(23) = 32 bits

exposant sur 8 bits =› décalage = $28^{8-1} - 1 = 127$

Dans de nombreux langages de programmation (C, C++, Java...) le type de donnée associé est nommé float

**Double précision**

taille totale : signe(1 bit)+partie exposant(11 bits)+mantisse52) = 64 bits

exposant sur 11 bits =› décalage = $2^{11-1} - 1 = 1023$

Dans de nombreux langages de programmation (C, C++, Java...) le type de donnée associé est nommé double

**Valeurs particulières**

La norme IEEE 754 réserve les exposants 000...000 (uniquement des 0) et 111...111 (uniquement des 1) pour coder des valeurs particulières

| Exposant | Mantisse | Valeur représentée |
|---|---|---|
| 000...000 | 000...000 | 0 (zéro) |
| 000...000 | 000.001 à 111...111 | nombre dénormalisé $valeur = \pm 0, mantisse \times 2^{-126 \ ou \ -1022}$ |
| 111...111 | 000...000 | $\pm\infty$ |
| 111...111 | 000.001 à 111...111 | NaN |

Tableau 2 – Codages

**Exercices**

1. Soit le nombre flottant au format simple précision : 00111101110011001100110011001100. Trouvez la représentation en base 10 de ce nombre.

2. Déterminez la représentation au format simple précision d'un tiers (1/3) en binaire et en hexadécimal.