Codage de l'information
Représentation des nombres en Informatique
Cours 2 - ITU 2022

Rakotoarimalala Tsinjo

Introduction

Le binaire est utilisé en informatique parce qu'il est facilement *simulable* physiquement. La présence d'un seuil de tension aux bornes des transistors, en négligeant la valeur exacte de cette tension, représentera 0 ou 1. Par exemple le chiffre 0 sera utilisé pour signifier une absence de tension à $0.5V$ près, et le chiffre 1 pour signifier sa présence à plus de $0.5V$.

Cependant les notations octales et hexadécimales sont souvent utilisées par les programmeurs dans un soucis de lisibilité et pour diminuer les risques d'erreur de transcription.

Dans ce deuxième cours on travaillera sur les représentations des nombres en informatique. Essentiellement on verra comment les nombres entiers non signées (ou naturels), les nombres entiers relatifs et ainsi que les nombres à virgules sont représentés en binaire.

1 Nombres entiers naturels

Dans la représentation des nombres entiers naturels, appelés entiers non signés, il n'y a rien de nouveaux. n bits peuvent représenter tous les entiers de 0 (les bits tous à 0) à $2^n - 1$ (les bits tous à 1).

2 Nombres entiers relatifs

2.0.1 Valeur absolue

Le bit de poids fort (celui de gauche dans la notation positionnelle habituelle) devient le signe du nombre appelé ***bit de signe***: 0 pour les positifs et 1 pour les négatifs.

Par exemple dans un système à 4 bits par exemple: 0001 vaut 1 et 1001 vaut -1.

Donc dans un système à n bits on peut représenter les nombres allant de $-2^{n-1} + 1$ à $2^{n-1} - 1$.

Mais cette représente présente deux inconvénients:

- On a deux représentations de 0 ($+0$ et -0) (par exemple à 4 bits, on 0000 et 1000). Cette double représentations complique les tests (comparaisons, etc).
- L'addition usuelle ne marche pas.

  Par exemple, si dans un système à 4 bits on veut faire $-4 + 3$, c'est-à-dire $1100 + 0011$, ça nous donne 1111 qui vaut -7 alors que le résultat attendu est -1 (1001).

2.0.2 Complément à deux

Pour remédier aux inconvénients de la représentation en valeur absolue, le *complément à deux* fut introduit. Les nombres positifs sont représentés comme attendu, en revanche les nombres négatifs sont obtenus de la manière suivante :

- On inverse les bits de l'écriture binaire de sa valeur absolue (opération binaire **NON**), on fait ce qu'on appelle *le complément à un* ;
- On ajoute 1 au résultat (les dépassements sont ignorés).

Les deux inconvénients précédents disparaissent alors. En effet, l'opération précédente effectuée sur 00000000 permet d'obtenir 00000000 ($-0 = +0 = 0$) et l'addition usuelle des nombres binaires fonctionne.

Revenons par exemple à l'opération ci-dessus $-4+3$. pour coder -4:

- on prend le nombre positif 4 : 0100 ;
- on inverse les bits : 1011 ;
- on ajoute 1 : 1100.

Donc $-4+3$ se fait donc $1100+0011 = 1111$. Or 1111 n'est rien d'autre que la représentation à complément à deux de -1.

Dans la représentation à complément à deux, avec n bits on peut répresenter tous les nombres de -2^{n-1} ($100\ldots0$) à $2^{n-1}-1$ ($011\ldots1$).

3 Nombres avec virgule

3.1 Conversion de binaire en décimal

On peut ainsi facilement convertir un nombre réel de la base 2 vers la base 10. Par exemple : $110,1012 = 1\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3} = 4+2+0,5+0,125 = 6,62510$.

3.2 Conversion de décimal en binaire

Le passage de base 10 en base 2 est plus subtil. Par exemple : convertissons $1234,347$ en base 2.

- La partie entière se transforme comme d'habitude : $1234_{10} = 10011010010_2$
- On transforme la partie décimale selon le schéma suivant :

$$\begin{array}{lll}
0,347\times2 = 0,694 & \text{donc} & 0,347 = 0,0... \\
0,694\times2 = 1,388 & \text{donc} & 0,347 = 0,01... \\
0,388\times2 = 0,766 & \text{donc} & 0,347 = 0,010... \\
0,766\times2 = 1,552 & \text{donc} & 0,347 = 0,0101... \\
0,766\times2 = 1,552 & \text{donc} & 0.347 = 0,01011... \\
0,766\times2 = 1,552 & \text{donc} & 0.347 = 0,01011... \\
0,552\times2 = 1,104 & \text{donc} & 0,347 = 0,010110... \\
0,104\times2 = 0,208 & \text{donc} & 0,347 = 0,010110... \\
0,208\times2 = 0,416 & \text{donc} & 0,347 = 0,0101100... \\
0,416\times2 = 0,832 & \text{donc} & 0,347 = 0,01011000... \\
0,832\times2 = 1,664 & \text{donc} & 0,347 = 0,010110001...
\end{array}$$

On continue ainsi jusqu'à la précision désirée...

Ce schéma repose sur le fait que la multiplication par deux n'est rien d'autres que le décalage à gauche.

Attention ! Un nombre à développement décimal fini en base 10 ne l'est pas forcément en base 2.

3.3 La norme IEEE 754

Dans la norme IEEE 754, un nombre flottant est toujours représenté par un triplet

$$(s, e, m)$$

où

- la première composante s détermine le signe du nombre représenté, ce signe valant 0 pour un nombre positif, et 1 pour un nombre négatif ;
- la deuxième e désigne l'***exposant*** ;
- et la troisième m désigne la ***mantisse***.

Signe	Exposant biaisé	Mantisse
1 bit	e bits	m bits

Biais de l'exposant

L'exposant peut être positif ou négatif. Cependant, la représentation habituelle des nombres signés (complément à 2) rendrait la comparaison entre les nombres flottants un peu plus difficile. Pour régler ce problème, l'exposant est « biaisé », afin de le stocker sous forme d'un nombre non signé.

Ce biais est de $2^{e-1} - 1$ (e représente le nombre de bits de l'exposant) ; il s'agit donc d'une valeur constante une fois que le nombre de bits e est fixé.

L'interprétation d'un nombre (autre qu'infini) est donc : $valeur = signe \times mantisse \times 2^{(exposant-biais)}$ avec

- signe $= \pm 1$
- biais $= 2^{e-1} - 1$

Exemple pour la simple précision : 32bits

La norme IEEE 754 pour la simple préciion se propose de coder le nombre en 32 bits et définit trois composantes :

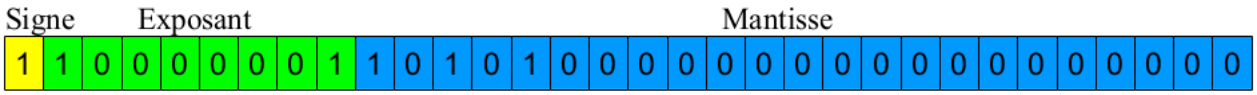
- le signe est représenté par un seul bit, le bit de poids fort
- l'exposant est codé sur les 8 bits consécutifs au signe
- la mantisse (les bits situés après la virgule) sur les 23 bits restants

Le biais est donc $2^{8-1} - 1 = 127$.

Traduisons en binaire, en utilisant la norme IEEE 754, le nombre $-6,625$

- Codons d'abord la valeur absolue en binaire : $6,625_{10} = 110,1010_2$
- Nous mettons ce nombre sous la forme : $1,$ ***partie fractionnaire***

  $110,1010 = 1,101010 \times 2^2$ (2^2 décale la virgule de 2 chiffres vers la droite)
- La partie fractionnaire étendue sur 23 bits est donc 10101000000000000000000.
- **Exposant** $= 127 + 2 = 129_{10} = 10000001_2$ (codé sur 8 bits)



Donc la représentation en utilisant la norme IEEE 754 de $6,625_{10}$ en binaire est 11000000 11010100 00000000 000 ou $C0$ $D4$ 00 00 en hexadécimal.

Type de précision dans la norme IEEE 754

La version d'origine de la norme IEEE 754, datant de 1985, définissait quatre formats pour représenter des nombres à virgule flottante en base 2 dont les deux les plus utilisées sont:

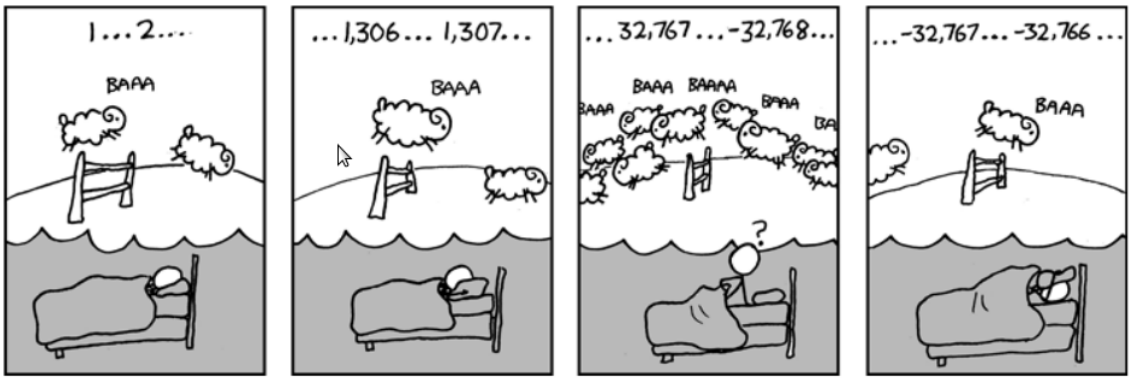
- simple précision (32 bits : 1 bit de signe, 8 bits d'exposant (-126 à 127), 24 bits de mantisse, dont un bit 1 implicite) ;
- double précision (64 bits : 1 bit de signe, 11 bits d'exposant (-1022 à 1023), 53 bits de mantisse, dont un bit 1 implicite) ;

Remarques

Un bit 1 implicite veut dire que si la mantisse vaut x dans la présentation, cette dernière sera interprété comme $1,x$. (voir exemple ci-dessus).

4 Exercices

1. Expliquer cet étrange rêve



2. Codez les entiers relatifs suivants sur 8 bits (16 si nécessaire) : 456, -1, -56, -5642
3. Codez le nombre réel suivant en utilisant la norme IEEE 754 sur 32 bits : $29,75$ et $-12,15$
4. Écrire l'algorithme de conversion en binaire d'un nombre réel écrit en base 10 avec une précision ϵ (nombre de chiffre après la virgule).