CHAPITRE 3 : CODAGE DES NOMBRES REELS

On peut, par analogie avec les nombres décimaux, écrire un nombre à virgule en notation binaire en utilisant les puissances négatives de 2 :

Par exemple : $(11{,}0101)_2=1\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}+0\times 2^{-3}+1\times 2^{-4}$
$(11{,}0101)_2=2+1+0+0{,}25+0+0{,}0625=3{,}3125$

Il devient vite compliqué d'utiliser cette méthode pour les nombres très petits ou très grands.

En notation décimale, on utilise la notation scientifique : pour écrire un nombre x, on précise son signe s, puis un nombre décimal m (appelé **mantisse**) et un entier relatif n (appelé **exposant**) :

$x=s\,m\times 10^n$.

Exemple : $173{,}95=+1{,}7395.10^2$*. On peut aussi écrire* $173{,}95=+17{,}395.10^1$.

On appelle cette écrire **virgule flottante**. La virgule « flotte » de droite à gauche, on peut la placer où on le souhaite.

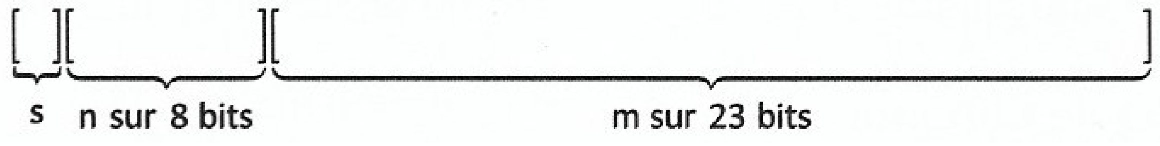
La norme IEEE-754 définit l'adaptation de cette méthode à la base 2.

Tout nombre réel peut être représenté sous la forme $x=(-1)^s m.2^E=(-1)^s m.2^{n-(N-1)}$, où s correspond au signe de x ($s=0$ pour un nombre positif, $s=1$ pour un nombre négatif), m (la mantisse) est un réel de l'intervalle ¿, $n$est un entier relatif tel que $E=n-(N-1)$, avec N le niveau de précision de la machine (32 ou 64 bits sur les machines modernes).

Cette norme définit les points suivants :

Encodage	Signe s	n	Mantisse m	Valeur x
32 bits	1 bit	8 bits	23 bits	$(-1)^s m 2^{(n-127)}$
64 bits	1 bit	11 bits	52 bits	$(-1)^s m 2^{(n-1023)}$

Soit la représentation suivante sur 32 bits :



Attention !!!! Le premier bit de la mantisse d'un nombre normalisé est forcément 1 (la mantisse est du type 1,…), donc il n'est pas représenté. N'apparaissent que les « nombres après la virgule ».

Remarques : par convention, on considère qu'un nombre vaut zéro si tous les bits de son exposant et de sa mantisse sont nuls.

Les valeurs extrêmes (0 et $2^N - 1$) sont réservées pour zéro et les infinis (en Python NaN : Not a Number.

Exemple : On souhaite savoir à quoi correspond le nombre ci-dessous codé en 32 bits :

$$\underbrace{1}_{s}\ \underbrace{0100\ 0110}_{n \text{ sur } 8 \text{ bits}}\ \underbrace{1001\ 0000\ 0000\ 0000\ 0000\ 000}_{m \text{ sur } 23 \text{ bits}}$$

Le premier 1 représente le signe, ici négatif.

0100 0110 représente l'exposant : $n = 2^6 + 2^2 + 2^1 = 70$*, donc* $E = 70 - 127 = -57$*.*

1001 0000 0000 0000 0000 0000 représenta la mantisse : $m = 1 + 2^{-1} + 2^{-4} = 1{,}5625$ *(attention de ne pas oublier le 1 implicite).*

Le nombre codé en base 10 est donc $x = -1{,}5625.2^{-57} \approx 1{,}0842.10^{-17}$ *(on obtient une valeur approchée décimale).*

Déterminer de même la valeur du nombre codé par

0 0111 1100 0100 0000 0000 0000 0000 000

(réponse : 0,15625, valeur exacte)

Attention aux arrondis

On ne peut représenter de manière exacte en virgule flottante que les nombres du type $\frac{a}{2^n}$, avec a et n entiers (vous pouvez faire le rapprochement avec les décimaux, que l'on peut écrire sous la forme $\frac{a}{10^n}$).

La représentation des réels est dite approximative.

La principale (et non des moindres) conséquence est que la notion d'égalité stricte en virgule flottante n'a bien souvent aucun sens. Il ne faut donc pas utiliser le test == pour comparer deux flottants. On préfèrera regarder si leur différence est suffisamment proche de zéro.

Exemple :

Dans la console Python, effectuer le test suivant : $0{,}1+0{,}2-0{,}3=\text{¿}0$. *Que vous renvoie le shell ?*

Proposer une méthode plus fiable pour tester « l'égalité ».