Représentation des nombres et conséquences

Renaud Costadoat
Lycée Dorian

Les nombres binaires

La numération binaire n'est constituée que de 0 et de 1. La succession des nombres binaires et la suivante:

0	1	2	3	4	5	6	7	...
0	1	10	11	100	101	110	111	...

Sous forme polynomiale, un nombre binaire quelconque est exprimé par: $N = \sum_{0}^{n} \alpha_i . 2^i$, avec $\alpha_i = 0$ ou 1.

ex:

- 10110 donne $1.2^4 + 0.2^3 + 1.2^2 + 1.2^1 + 0.2^0 = 22$ en décimal
- 101100 donne ..... en décimal

Opérations sur les nombres binaires

Signe par bit de signe

0	0	0	0	1	1	0	0	12
1	0	0	0	1	1	0	0	-12

Cette représentation du signe ne permet pas d'effectuer de soustraction.

Signe par complément à 2

0	0	0	0	1	1	0	0	12
1	1	1	1	0	0	1	1	C1
							1	+1
1	1	1	1	0	1	0	0	-12

Addition

	1	0	0	1	0	1	37
+	0	1	0	1	1	1	23
	.	.	.	.	.	.	.

Soustraction

	0	0	0	1	0	0	0	1	17
+	1	1	1	1	0	1	0	0	-12
	.	.	.	.	.	.	.	.	.

La représentation des nombres en mémoire

- Les **systèmes informatiques** travaillent sur des longueurs fixes de bits appelés MOT.
- Un MOT est la plus grande série de bits qu'un ordinateur puisse traiter en une seule **opération**. *Exemple:* la famille de micro-processeur actuelle utilise des mots de 32 bits ou plus récemment de 64 bits.
- L'entité de base de l'information sur un système informatique est l'**octet** (8 bits) (attention *octet* se dit *byte* en anglais, c'est un faux ami). Néanmoins, les nombres peuvent être codés de différentes façons. Il est important de bien comprendre comment se présentent les nombres dans divers format, afin de:
  - **Minimiser la place** occupée sur le support de stockage,
  - Mais aussi la place occupée en mémoire centrale et donc la **rapidité des traitements** que l'on fera sur ces données.

Problème de l'overflow

- En informatique, le bug de l'an XXXX est un problème similaire au bug de l'an 2000 qui pourrait perturber le fonctionnement d'ordinateurs 32 bits.
- Le problème concerne des logiciels qui utilisent la représentation POSIX du temps, dans lequel le temps est représenté comme un nombre de secondes écoulées depuis le 1er janvier 1970 à minuit (0 heure). Sur les ordinateurs 32 bits, la plupart des systèmes d'exploitation concernés représentent ce nombre comme un nombre entier **signé** de 32 bits, ce qui limite le nombre de secondes.
- Lorsque ce nombre sera atteint, dans la seconde suivante, la représentation du temps « bouclera »et sera négative par complément à deux.

1. Quand cet évènement aura-t-il lieu ? .....
2. Quelle sera la date affichée la seconde après cette limite ?
   .....

Représentation de la partie fractionnaires des réels

Il est possible de représenter la partie fractionnaire des réels de la même manière que la partie entière. En effet, au lieu de prendre des puissances positives de 2, il suffit de prendre les négatives. Ainsi, on peut donc écrire un nombre réel comme suit.

Exemple

$$10,0110_2 = 1.2^1 + 0.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3} + 0.2^{-4}$$
$$10,0110_2 = 2 + 0 + 0 + 0,25 + 0,125 + 0 = 2,375_{10}$$

Une méthode permet alors de coder cette partie fractionnaire en suivant la procédure suivante:

1. Multiplier la partie fractionnaire par 2.
2. La partie entière obtenue représente le poids binaire (limité aux seules valeurs 0 ou 1).
3. La partie fractionnaire restante est à nouveau multipliée par 2.
4. Procéder ainsi de suite jusqu'à ce qu'il n'y ait plus de partie fractionnaire ou que le nombre de bits obtenus corresponde à la taille du mot mémoire dans lequel on stocke cette partie.

Représentation de la partie fractionnaires des réels

Proposer le codage de 0.1.

0,1	x	2	=	0,2	=	0	+	0,2
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...								

Cet exemple montre qu'il n'est pas possible de coder tous les nombres réels avec cette méthode.

Représentation de la virgule flottante

C'est pour cette raison qu'une autre méthode a été inventée. Elle consiste à utiliser la représentation en **virgule flottante** (float en anglais). L'exemple suivant montre cette méthode appliquée à la base **10**:

Exemple

$$-418,22_{(10)} = \underbrace{-}_{1}\ \underbrace{0,41822}_{2} \cdot 10^{\overbrace{3}^{3}}$$

On appelle alors :

1. le signe (positif ou négatif),
2. la mantisse (nombre de chiffres significatifs),
3. l'exposant : puissance à laquelle la base est élevée.

Stockage des flottants

Il sera ainsi nécessaire d'utiliser:

- **1 bit:** pour le signe,
- **n bit:** pour l'exposant,
- **m bit:** pour la mantisse.

Ces valeurs sont réparties en fonction de la précision:

	Signe	Exposant	Mantisse
Simple précision (32bits)	1	8	23
Double précision (64bits)	1	11	52
Précision étendue (80bits)	1	15	64

Conversion réel en binaire (hexadécimal)

1. Convertir les parties entière et fractionnaire du nombre sans tenir compte du signe.
2. Décaler la virgule vers la gauche pour le mettre sous la forme normalisée (IEEE 754).
3. Codage du nombre réel avec les conventions suivantes :
   - *signe* = 1 : Nombre négatif (*Signe* = 0 : Nombre positif),
   - ajouter à l'exposant simple la valeur 127 en simple précision et 1023 en double précision (c'est à dire $2^{n-1}-1$ où n est le nombre de bits de l'exposant),
   - la mantisse est complétée à droite avec des zéros.

Convertir le nombre $238,125$ en virgule flottante au format simple précision.

1. $238,125_{10} = 11101110,001_2$,
2. $238,125_{10} = 1,1101110001_2 * 2^7$: décalage (par multiplication) de 7 bits vers la gauche,
3. Exposant: $7+127 = 134_{10} = 10000110_2$ sur $n = 8$ bits,
4. Mantisse: 11011100010000000000

S	Exposant								Mantisse																			
0	1	0	0	0	0	1	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4				3				6				E				2				0				0				0	

Conversion binaire (hexadécimal) en réel

La résolution dans ce cas de figure consiste à utiliser la procédure inverse à la précédente.

4				5				B				3				E				0				2				0			
0	1	0	0	0	1	0	1	1	0	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0
S	Exposant											Mantisse																			

1. Mantisse: .....
2. Exposant: ....., donc l'exposant simple est .....
3. .....

Problèmes de précision

Il est alors impossible de représenter exactement la plupart des nombres décimaux.

Par exemple, la suite va consister à représenter le nombre $0,2$ en virgule flottante.

Commencer par convertir en binaire la partie fractionnaire du nombre:

0,2	x	2	=	0,4	=	0	+	0,4
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...
...	x	2	=	...	=	.	+	...

Le résultat est donc:

$0,2_{10}$ = ...

$0,2_{10}$ = ..., donc l'exposant décalé est $-3+127 = 124_{10} = 01111100_2$

S	Exposant							Mantisse																							
0	0	1	1	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
3				E				4				C				C				C				C				C			

Problèmes de précision

S	Exposant								Mantisse																			
0	0	1	1	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	**1**
3				E					4			C				C				C				C				C			

Le dernier bit est passé à 1 afin de s'approcher de la valeur la plus proche de $0,2$.

Les résultats approchés sont donc:

Précision	Valeur	Erreur
Simple 32 bits	$2.00000002980232238769531 25E-1$	$1,5.10^{-6}\%$
Double 64 bits	$2.00000000000000011102230246252E-1$	$0,5.10^{-14}\%$