Représentation des nombres

Représentation des entiers

Représentation des réels

Opérations arithmétiques

Représentation des réels

- En virgule fixe
- En virgule flottante

Dans la suite, sauf indication contraire toutes les écritures de nombres sont en base dix

Représentation des nombres réels

En base dix, un nombre réel qui s'écrit

$\pm d_m d_{m-1} \ldots\ldots d_1 d_0, d_{-1} d_{-2} \ldots d_{-n}$

Représente la valeur

$$\pm \sum_{-n}^{+m} d_i 10^i$$

En conséquence, en base dix on ne peut représenter exactement que des nombres fractionnaires de la forme $n/10^k$ où n est un nombre entier

Par exemple impossible de donner une écriture finie exacte de 1/3 (0.333333….)

Mais impossible de donner une écriture finie exacte de un tiers en base 2 :

1/3 2x1/3 = 2/3 < 1 2x4/3 = 8/3 = 1 + 1/3 2x1/3 = 2/3 ….

0, 0 1 0 ….

0,01(01)(01)(01)(01) …

ni de un dixième :

1/10 2/10 = 1/5 2/5 4/5 8/5 = 1 + 3/5 6/5 = 1 + 1/5 2/5 …..

0, 0 0 0 1 1 0 ……

0,0(0011)(0011)(0011)(0011) …

Ce qui risque de générer des problèmes d'arrondis.

Virgule fixe

- 1 bit de signe
- m bits pour la partie entière
- n bits pour la partie fractionnaire

- En complément à deux

Résolution et dynamique

- La plus petite valeur que l'on peut coder : -2^m
- La plus grande $2^m - 2^{-n}$

- *Dynamique* : différence entre ces deux valeurs
- *Résolution* : écart minimum entre deux réels représentés : 2^{-n}

- Pour obtenir la valeur représentée par une telle représentation :
  1. on considère que cette représentation est celle d'un entier en complément à 2 sur 1+m+n bits
  2. on calcule la valeur entière obtenue et on divise par 2^n

Représentation en virgule flottante

- Un nombre réel est représenté par:
  - un signe s : sur 1 bit (0 positif, 1 négatif)
  - une mantisse (en base b) m
  - Un exposant e

  La valeur du réel est : $(-1)^s\ m\ b^e$
- Exemple : π

  $(-1)^0 \times 0{,}031 \times 10^2$

  $(-1)^0 \times 3{,}142 \times 10^0$

  $(-1)^0 \times 31{,}42 \times 10^{-1}$

  $(-1)^0 \times 0{,}003 \times 10^3$

  Plusieurs représentations approchées

  Pas la même précision

Représentation normalisée (et notation scientifique)

- Une représentation est ***normalisée*** si elle est sous la forme

$$(-1)^s \; M \; b^e$$

Avec ***M*** **qui est dans l'intervalle [1,b[**

- On ne stocke que s, M et e .
  En Base 2 ça impose : *M = 1,M'*
  et **c'est *M'* la pseudo mantisse que l'on stocke**

- tout réel (non nul) a une unique notation scientifique
- Mais tout réel (non nul) n'a pas nécessairement une représentation normalisée exacte sur N_m bits pour représenter la mantisse et N_e bits pour représenter l'exposant e
- Étant donnés N_m bits pour représenter la mantisse et N_e bits pour représenter l'exposant e, la représentation normalisée donne meilleure précision possible pour ce réel
- 0 n'a pas de représentation normalisée !

Le système IEEE 754

- Un standard pour la représentation des nombres à virgule flottante en binaire

1+8+23=32 simple précision utilisée par exemple pour le type ***float*** java

1+11+52=64 double précision utilisée par exemple pour le type ***double*** java

Codage de l'exposant

- Il faut qu'il puisse être négatif !
- Plutôt que le choix du complément à deux, translation par une constante d'excentrement (ou biais)
- On passe de $[0, 2^{N_e} - 1]$ à $[-2^{N_e-1}+1, 2^{N_e-1}]$ en translatant de $2^{N_e-1}-1$
- Translation de 127 en simple précision, de 1023 en double précision

Exemple avec 8 bits

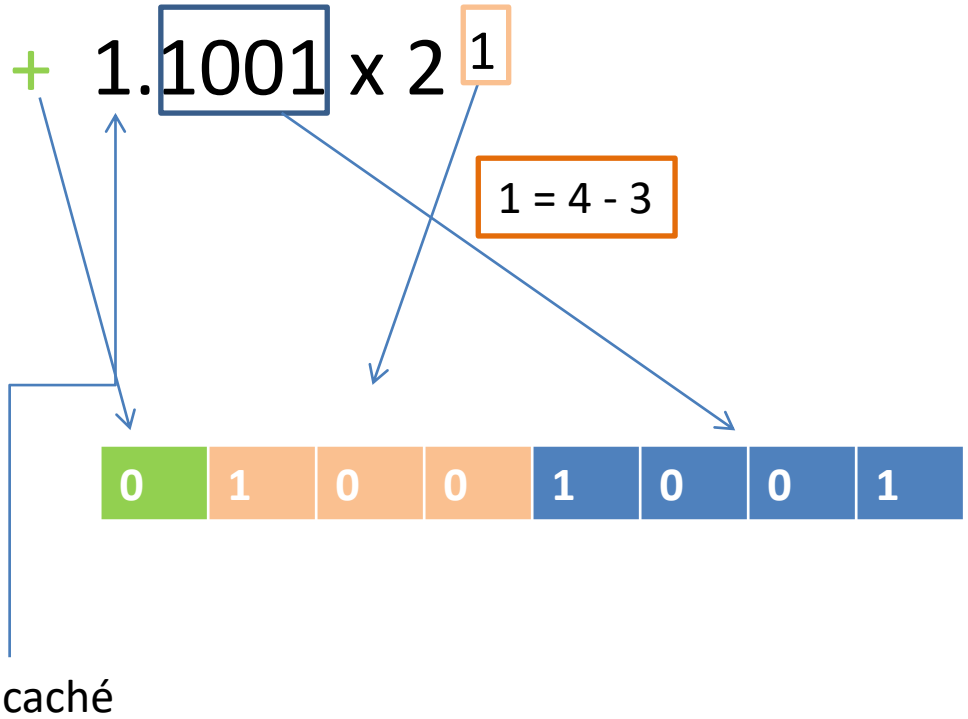
1 bit pour le signe, 3bits pour l'exposant, 4 bits pour la pseudo mantisse

L'exposant pourra varier entre -3 [000] et +4 [111] grâce à un excentrement de 3

Le bit de signe est à 0 : nombre positif
L'exposant vaut 3 : 6-excentrement
La pseudo mantisse est 1011
La mantisse est donc 1,1011
Le réel représenté écrit en base 2 est donc : **11O1,1**
Soit **13,5**

Représenter le réel 3,125

- 11,001 [écrire en base 2]
- $1{,}1001 \cdot 2^1$ [normaliser]
- + 1.1001×2^1



Et le zéro

- Pas moyen d'utiliser une écriture normalisée pour 0
- Du coup on réserve le plus petit et le plus grand des exposants pour des écritures non normalisées (dont 0)

Codage de l'exposant, revisité pour inclure des nombres non normalisés

- On passe de $[0, 2^{N_e} - 1]$ à $[-2^{N_e-1}+1, 2^{N_e-1}]$ en translatant de $2^{N_e-1}-1$,mais on réserve
  - Le plus grand codé 1..1 pour représenter les valeurs spéciales
    - L'infini [pseudo mantisse à 00000000…00] le signe dira si positif ou négatif
    - NaN [NotANumber pseudo mantisse pas à 0]
  - Le plus petit codé 0..0 pour représenter :
    - Zéro [pseudo mantisse à 00000000…00] : deux codages en fait, positif ou négatif
    - Nombres dénormalisés avec mantisse non nulle
  - Les nombres normalisés ont donc un intervalle d'exposant possible un peu plus réduit

Nombres dénormalisés

Normalement, la valeur de l'exposant d'un nombre non normalisé devrait être :

0 – excentrement

Toutefois, pour assurer une meilleure transition

entre les nombres normalisés et les non normalisés, l'exposant est calculé dans ces cas comme 1 - excentrement

Exemple sur 6 bits avec $N_e = 3$ et $N_m = 2$

- L'excentrement vaut $3 = 2^{3-1} - 1$
- Il y a 64 mots de longueur 1+3+2 = 6 dont
  - 8 valeurs spéciales (exposant 111)
  - 48 =2 x 6 x 4 nombres normalisés, dont 24 sont strictement positifs et 24 strictement négatifs (6 exposant 001 à 110) mantisse possible (00,01,10,11)
  - 8 nombres non normalisés (exposant 000) qui représentent 7 réels (dont zéro qui a deux écritures) tous entre 0 et le plus petit nombre positif normalisé

Exemple sur 6 bits avec N_e= 3 et N_m= 2

- Les valeurs spéciales (exposant 111)

Nombres non normalisés	Valeurs spéciales
0 111 00	+ infini
0 111 01	NaN
0 111 10	NaN
0 111 11	NaN
1 111 00	- infini
1 111 01	NaN
1 111 10	NaN
1 111 11	NaN

Nombre normalisés positifs

Nombre normalisé			Base 10				Base 10
0 001 00	$+1{,}00.\ 2^{-2}$	+0,01	0,25	0 100 00	$+1{,}00.2^{1}$	10	2
0 001 01	$+1{,}01.\ 2^{-2}$	+0,0101	0,3125	0 100 01	$+1{,}01.2^{1}$	10,1	2,5
0 001 10	$+1{,}10.\ 2^{-2}$	+0,011	0,375	0 100 10	$+1{,}10.2^{1}$	11	3
0 001 11	$+1{,}11.\ 2^{-2}$	+0,0111	0,4375	0 100 11	$+1{,}11.2^{1}$	11,1	3,**5**
0 010 00	$+1{,}00.2^{-1}$	+0,1	0,5	0 101 00	$+1{,}00.2^{2}$	100	4
0 010 01	$+1{,}01.2^{-1}$	+0,101	0,625	0 101 01	$+1{,}01.2^{2}$	101	5
0 010 10	$+1{,}10.2^{-1}$	+0,11	0,75	0 101 10	$+1{,}10.2^{2}$	110	6
0 010 11	$+1{,}11.2^{-1}$	+0,111	0,875	0 101 11	$+1{,}11.2^{2}$	111	7
0 011 00	$+1{,}00\ .2^{0}$	+1	1	0 110 00	$+1{,}00\ .2^{3}$	1000	8
0 011 01	$+1{,}01\ .2^{0}$	+1,01	1,25	0 110 01	$+1{,}01\ .2^{3}$	1010	10
0 011 10	$+1{,}10\ .2^{0}$	+1,1	1,5	0 110 10	$+1{,}10\ .2^{3}$	1100	12
0 011 11	$+1{,}11\ .2^{0}$	+1,11	1,75	0 110 11	$+1{,}11\ .2^{3}$	1110	14

Nombres non normalisés

- Les valeurs spéciales (exposant 000)

Nombres non normalisés			En base dix
0 000 00	0,00	0	0
0 000 01	$+0,01\ 2^{-2}$	0,0001	0,0625
0 000 10	$+0,10\ 2^{-2}$	0,001	0,125
0 000 11	$+0,11\ 2^{-2}$	0,0011	0,1875
1 000 00	-0,00	0	0
1 000 01	$-0,01\ 2^{-2}$	-0,0001	-0,0625
1 000 10	$-0,10\ 2^{-2}$	-0,001	-0,125
1 000 11	$-0,11\ 2^{-2}$	-0,0011	-0,1875

Arithmétique en virgule flottante

- On va avoir des problèmes d'arrondis
  - Soit parce qu'on veut utiliser un nombre qui a écriture finie en base dix mais pas en base deux, par exemple un dixième
  - Soit parce qu'il y a bien une écriture finie mais en base deux, mais qu'il faudrait plus de bits pour pouvoir stocker le nombre exact :
  - 8+5 = on peut espérer 12 ou 14 mais pas 13 !!

Addition de deux flottants positifs

1. Décaler à droite la mantisse (sans oublier le bit caché) du nombre possédant le plus petit exposant jusqu'à arriver à l'exposant de l'autre nombre
2. Additionner les mantisses
3. Normaliser le résultat
4. Arrondir

Exemple d'addition de flottant positifs

0 100 10 + 0 011 10 (3 + 1,5)
$1{,}1 \times 2^1 + 1{,}1 \times 2^0$

1. $1{,}10 \times 2 + 0{,}11 \times 2$
2. $10{,}01 \times 2$
3. $1{,}001 \times 2^2$
4. $1{,}00 \times 2^2$ (c'est-à-dire quatre !) parce qu'on a seulement deux chiffres après la virgule

le résultat est donc 0 101 00

Il n'y aurait pas eu d'erreur d'arrondi dans ce cas si on avait eu $N_m >= 3$

Et 5+8 alors ?

- $1{,}01 \times 2^2 + 1{,}00 \times 2^3$
- $0{,}101 \times 2^3 + 1{,}00 \times 2^3$ alignement des exposants
- $1{,}101 \times 2^3$ addition des mantisses
- pas de normalisation à faire
- $1{,}10 \times 2^3$ arrondi (car mantisse sur 2 bits)
- Résultat : 0 110 10 soit 12

Et 8+8 alors ?

- $1{,}00 \times 2^3 + 1{,}00 \times 2^3$
- $1{,}00 \times 2^3 + 1{,}00 \times 2^3$ alignement des exposants
- $10{,}00 \times 2^3$ addition des mantisses
- $1{,}0 \times 2^4$ on normalise
- pas d'arrondi
- Résultat : 0 111 00 une valeur spéciale, normal on a dépassé la plus grande valeur qu'on pouvait représenter (overflow)

Arrondi

- L’arrondi se fait à la valeur la plus proche possible
- En cas d’équidistance on va vers la valeur paire

Addition de flottants de signe différents (ou soustraction)

- Même principe que l'addition, mais si les signes sont différents l'addition se fait en complément à deux

Multiplication de flottants

- On détermine le signe du résultat
- On calcule la somme des exposants (sans oublier de corriger l'excentrement)
- On multiplie les mantisses (virgule fixe !)
- On gère les arrondis
- Et les éventuels dépassement de capacité (overflow ou underflow)

Multiplier par B, c'est seulement ajouter un à l'exposant !

Avantages de la norme

- les nombres flottants sont portables d'un système à l'autre
- les opérations sont uniquement définies
- les programmes ont un comportement unique
- les programmes numériques sont analysables