Représentation des nombres

Représentation des entiers

Représentation des réels

Opérations arithmétiques

Représentation des entiers

- La notion mathématique d'entier n'est pas équivalente à la notion de type entier en informatique
- En maths, tout entier à un successeur qui est aussi un entier, mais les entiers codés en machine ont une valeur maximale

L’addition n’est pas associative

a, b, c sont trois entiers naturels, en maths

$$(a+b)-c = a+(b-c)$$

a,b,c sont de type integer, il est possible que

$$(a+b)-c \neq a+(b-c)$$

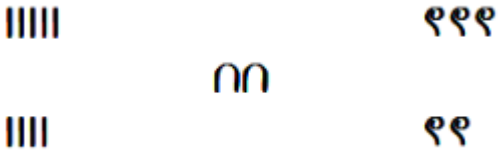
Si a+b est trop grand !

Même problème avec la distributivité

$a(b-c) \neq ab-ac$ si ab est trop grand

Systèmes additifs

529



En egyptien ancien (troisième millenaire avant JC)

DXXIX

En chiffres romains

Nécessite un symbole pour chaque puissance de 10

Ecriture positionnelle

- Nécessite un symbole pour 0 (marqueur de position). Inventé indépendamment par les babyloniens (-1000) , les mayas (premier millénaire) et les indiens (troisième siècle)

- Ecriture décimale positionnelle arrive en Europe au Xième siècle et mets plusieurs siècles à s'imposer

Représentation des entiers en base dix

En base dix, l'écriture

$\pm d_m d_{m-1} \ldots\ldots d_0$ où

$\forall i : d_i \in \{\text{'0'}, \text{'1'}, \text{'2'}, \text{'3, '4'}, \text{'5'}, \text{'6'}, \text{'7'}, \text{'8'}, \text{'9 '}\}$

représente l'entier dont la valeur est

$$\pm \sum_0^{+m} val(d_i) 10^i$$

Ajouter un '0' à la fin, c'est multiplier par 10!

๐	๑	๒	๓	๔	๕	๖	๗	๘	๙
0	1	2	3	4	5	6	7	8	9

Représentation des entiers en base B

En base B, le mot

$\pm d_m d_{m-1} \ldots\ldots d_0$ avec les d_i dans un ensemble de B symboles associés aux valeurs 0,1,2,.....,B-1

représente l'entier dont la valeur est

$$\pm \sum_0^{+m} val(di) Bi$$

Ajouter le symbole qui vaut 0 en fin c'est multiplier par B

Exemples

- Ecriture binaire (en base 2) : Deux symboles 0 et 1 auxquels on attribue les valeurs 0 et 1
- Octal (base 8) : Huit symboles 0,1,2,3,4,5,6,7
- Hexadécimal (base 16): 16 symboles 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F auxquels on attribue les valeurs de zéro à quinze.

Conversion de la base 2 vers la base dix

- On fait le calcul de la valeur et comme la base dix est notre base usuelle, on a fini

- Exemple 11001
- $1x2^4+1x2^3+0x2^2+0x2^1+1x2^0$
- $(16)_{dix}+(8)_{dix}+(1)_{dix}=(25)_{dix}$

Conversion de la base dix vers la base deux

- On procède par division par 2 successives

- Exemple $(28)_{dix}=(11100)_{deux}$
-

	/2	reste
28	14	0
	7	0
	3	1
	1	1
	0	1

Entiers représentables

- Si on dispose de n bits pour coder un entier naturel, on peut représenter 2^n entiers.
- Par exemple :

  les entiers compris entre 0 et 2^n -1.

Dans la suite, par défaut toutes les écritures d'entiers sont en base dix

  - Pour n=16 0 à 65 535
  - Pour n=32 0 à 4 294 967 295
  - Pour n=64 0 à environ $1.8\ 10^{19}$

Sauf que… il y a aussi les négatifs

- 4 solutions:
  - Signe Grandeur
  - Complément à 1
  - Complément à 2
  - Excentrement

Signe Grandeur sur 8 bits

- 1 bit pour le signe, les autres pour la valeur :
  - 24 est codé 00011000
  - -24 est codé 10011000

- Pas pratique, parce que ça additionne pas bien !
- Deux codages possibles pour zéro

Complément à 1 sur 8 bits

- Pour les entiers négatifs on prend le complément à 1 bit à bit
  - 24 est codé 00011000
  - -24 est codé 11100111

- Pas pratique, parce que ça additionne toujours pas bien !
- 24 +(-1) = 00011000 + 11111110=(0)00010110
- Deux codages possibles pour zéro

Complément à 2 sur 8 bits

- Pour les entiers négatifs on prend le complément à 1 bit à bit auquel on ajoute 1
- 24 est codé 00011000
- -24 est codé 11101000

- Pratique, parce que ça additionne bien !
- 24+(-1) = 00011000+11111111=(0)00010111
- Un seul codage pour zéro

Remarque

- On dit complément à 2, mais en fait c'est plutôt un complément à 2^n
- Ce qu'on fait c'est que pour représenter en complément à deux sur n bits (dont un bit de signe) l'entier négatif -m compris entre en -2^{n-1} et -1, on code 2^n-m (compris entre 2^{n-1} et 2^n - 1) sur n bits (le premier bit sera forcement à un

Types entiers en java, codés en complément à deux

	Codés sur	Min	Max
short	8 bits	$-2^{15}=-32768$	$2^{15}-1=32767$
int	16 bits	$-2^{31}=-2\ 147\ 483\ 648$	$2^{31}-1=2\ 147\ 483\ 647$
long	32 bits	-2^{63} = -9 223 372 036 854 775 808	$2^{63}-1=9\ 223\ 372\ 036\ 854\ 775\ 807$

Excentrement

On passe de l'intervalle [0, 2k+1] à l'intervalle [-k-1,k] par une translation de k+1

On verra cela dans quelques slides…

Addition sur les entiers signés

Addition de deux entiers naturels (codés en complément à deux sur 8 bits)

	0 0001010		10	
+	0 0001111	+	15	
=	0 0011001		=	25

	0 1111010		positif
+	0 1101111	+	positif
=	1 1100101	=	négatif

Le résultat n’est égal que sur un bit de plus !

Addition de deux entiers négatifs (codés en complément à deux sur 8 bits)

Même problème, le résultat est faux si la valeur absolue de la somme est trop grande

Dans les deux cas le résultat serait correct sur n+1 bit

Dans les deux cas, le résultat est faux si et seulement si le bit de signe est modifié

Addition de deux entiers de signes contraires

	1 1111111		-1
+	0 0010100	+	20
= (1)	0 0010011	=	19

Résultat toujours correct, (mais faux sur n+1 bits)

Soustraction

- Soustraire n, c’est ajouter -n

Multiplication

- On détermine le signe en fonction du signe des opérandes
- On calcule la multiplication des valeurs absolues, en utilisant le fait que la multiplication par B est un décalage à gauche

Représentation des réels

- En virgule fixe
- En virgule flottante

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 $X/10^k$

Par exemple impossible de donner une écriture finie exacte de 1/3 (0.333333[3]*)

En base deux, un nombre réel qui s'écrit

$\pm b_m b_{m-1} \ldots\ldots b_1 b_0 , b_{-1} b_{-2} \ldots b_{-n}$

représente la valeur

$$\pm \sum_{-n}^{+m} b_i 2^i$$

En conséquence, en base deux on ne peut représenter exactement que des nombres fractionnaires de la forme $X/2^k$

Par exemple impossible de donner une écriture finie exacte de un tiers :

0,01(01)(01)(01)(01) …

ou de un dixième :

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 pour les négatifs

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}

Représentation en virgule flottante

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

π

$-1^0.0{,}031.10^2$

$-1^0.3{,}142.10^0$

$-1^0.31{,}42.10^{-1}$

$-1^0.0{,}003.10^3$

Plusieurs représentations approchées
Pas la même précision

Représentation normalisée

- Une représentation est normalisée si elle est sous la forme

$$(-1)^s \text{x } M \text{x} b^e$$

Avec M qui ne doit pas commencer par 0 .

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

- Une seule représentation
- 0 n'a pas de représentation normalisée !
- Précision la meilleure possible pour ce réel

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

Pour comprendre 8 bits suffiront!

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

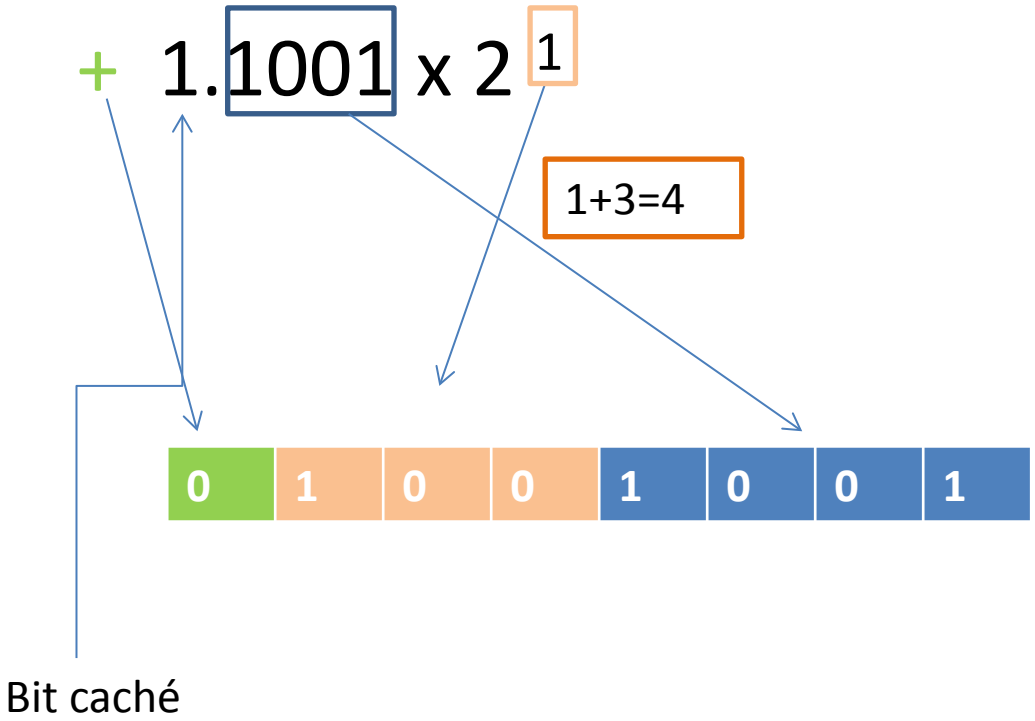
L'exposant pourra varier entre -3 [000] et +4 [1111] grâce à un excentrement de 3

Le bit de signe est à 0, c'est un nombre positif
L'exposant vaut 3 (6-3)
La pseudo mantisse est 1011
La mantisse est donc 1,1011 e t le réel représenté est

1101,1 donc treize et demi

Représenter le réel 3,125 (base dix)

- 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 exposant pour des écritures non normalisées

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

Nombre 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

Mini exemple, N_e=3 et N_m=2

- L'excentrement vaut 3
- Il y a 64 mots de longueur 1+3+2=6 dont
  - 8 valeurs spéciales (exposant 111)
  - 48 =2.6.4 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 qui représentent 7 réels (dont zéro qui a deux écritures) (exposant 000) tous entre 0 et le plus petit nombre positif normalisé

Mini exemple, 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 111)

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 utilisé un nombre qui a écriture finie en base dix mais pas en base deux, par exemple un dixième (et ça peut faire très mal!)
  - 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 (trois plus trois demis)
$+1,10.2^{1+}+1,10.2^{0}$

1. 11,10.2+0,11.2
2. 100,01
3. $1,0001.2^{2}$
4. $1,00.\ 2^{2}$ (c'est-à-dire quatre !)

le résultat est donc 0 101 00

Il n'y aurait pas eu d'erreur d'arrondi dans ce cas si on avait eu $N_m >=4$

Et cinq et huit alors ?

- $+1,01.2^2$ $+1,00.2^3$
- $0,101.\ 2^3+1,00.2^3$ alignement des exposants
- $1,101\ .\ 2^3$ addition des mantisses
- pas de normalisation à faire
- $1,10\ .\ 2^3$ arrondi
- Résultat 0 110 10 (douze donc)

Et huit et huit alors ?

- $+1{,}00.2^3$ $+1{,}00.2^3$
- $0{,}10.\ 2^3 + 1{,}00.2^3$ alignement des exposants
- $10{,}00\ .\ 2^3$ addition des mantisses
- $1{,}0.\ 2^4$ 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