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

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 B

En base B (B entier > 1), 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

$$\pm \sum_0^m val(di)Bi$$

Remarque : ajouter le symbole qui vaut 0 en fin d'écriture c'est multiplier par B, la valeur de l'entier écrit

Représentation des entiers en base dix

En base dix, l'écriture

$\pm d_m d_{m-1} \ldots\ldots d_0$ où

$\forall i : d_i \in \{$'0', '1', '2', '3, '4', '5', '6', '7', '8', '9 '$\}$
représente l'entier dont la valeur est

$$\pm \sum_0^m val(d_i) dix^i$$

Remarque : ajouter un '0' à la fin, c'est multiplier par dix la valeur de l'entier écrit

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

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 « habituel » i.e. en base dix, 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 divisions 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 $2^n -1$.

  – 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 n’additionne pas bien !
  – deux codages possibles pour zéro

Complément à 1 (exemple 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 n’additionne toujours pas bien !

24 +(-1) = 00011000 + 11111110=(0)00010110

  – deux codages possibles pour zéro

Complément à 2 (exemple 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 (ici 2^8)
- 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 2 sur 8 bits)

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

	0 1111010		positif
+	0 1101111	+	positif
=	1 1101001	=	négatif

Overflow

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

Addition de deux entiers négatifs (codés en complément à 2 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 bits

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