Représentation des nombres Numération Binaire

Ordinateur et arithmétique

Représentation des nombres positifs

Représentation des nombres signés

Virgule fixe, virgule flottante

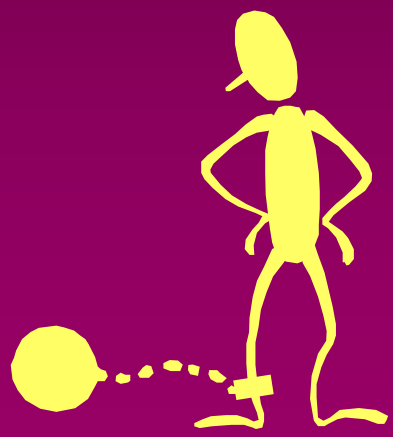
Nombre en précision finie

Ordinateurs : nombres en précision finie et fixe

Exemple : ensemble des entiers positifs à trois chiffres décimaux

A = {000,001,002, ..., 999} Card(A)=1000

Ne peut représenter :
- > 999
- < 0
- fractionnaires
- irrationnels
- complexes



Arithmétique sur Z : fermeture vis à vis de +,-,* (pas de /)

Soient $i, j \in Z \quad \Rightarrow i + j \in Z$

$i - j \in Z$

$i * j \in Z$

$i / j \notin Z$ en général

Fermeture pas applicable en précision finie !

Sur A : 600 + 600 = 1200 $\notin$ A (overflow)

003 - 005 = -2 $\notin$ A (underflow)

050 * 050 = 2500 $\notin$ A (overflow)

007 / 002 = 3,5 $\notin$ A (arrondi)

Conséquences sur ordinateurs (1)

Possibilités de résultats faux en conditions normales de fct. (pas de panne)

Dans A : a = 700, b = 400, c = 300

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

Conséquences sur ordinateurs (2)

Possibilités de résultats faux en conditions normales de fct. (pas de panne)

Dans A : a = 700, b = 400, c = 300

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

700 + 100 = 800 Overflow -300 = OverFlow

Conséquences sur ordinateurs (3)

Dans A : a = 5, b = 210, c = 195

a * (b - c) = a * b - a * c distributivité

5 * 15 = 75

Overflow - Overflow = OverFlow

Il faut connaître les méthodes de représentations pour prévoir les problèmes éventuels

Représentation des nombres

Evolution : romaine, (invention du zéro), inde, arabe

Principe de numération : Juxtaposition de symboles
appelés chiffres (caillou en arabe)

Système décimal : dix symboles {0,1,2, ...,9}
Nombre de symbole = Base de numération

Ecriture d'un nombre : position du chiffre détermine son poids

NUMERATION DE POSITION

$$1578 = 1.10^3 + 5.10^2 + 7.10^1 + 8.10^0$$

(en europe : 970 Gesbert d'Aurillac devenu en 999 Sylvestre II, relayé en 1202 par Fibonnacci)

Généralisation

Base b associée à b symboles $\{S_0, S_1, S_2, ..., S_{b-1}\}$

N s'écrit ($a_n\, a_{n-1}\, a_{n-2} \ldots a_0\, ,\, a_{-1} \ldots a_{-m}$) (dépend de la base)

avec a_i dans $\{S_0, S_1, S_2, ..., S_{b-1}\}$

Valeur de N = $a_n.b^n + a_{n-1}.b^{n-1} + \ldots + a_0.b^0 + a_{-1}.b^{-1} + \ldots + a_{-m}.b^{-m}$

$$= \sum_{-m}^{n} a_i.b^i$$

Forme polynomiale

La valeur est indépendante de la base

Définitions

$$N = (a_n\, a_{n-1}\, a_{n-2} \ldots a_0 , a_{-1} \ldots a_{-m})_b$$

a_i chiffre de rang i (ou digit)

b^i poids associé à a_i

a_n chiffre le plus significatif (ou de poids fort) MSD

a_{-m} chiffre le moins significatif LSD

$(a_n\, a_{n-1}\, a_{n-2} \ldots a_0)$ partie entière

$(a_{-1} \ldots a_{-m})$ partie fractionnaire (<1)

Systèmes utilisés

Base 2 (ou binaire) {0,1} digit = élément binaire ou eb
binary digit ou bit

la plus utilisée : MSB, LSB 00110101 = octet
1101 = quartet

Base 10 (ou décimal)

celle de l'école primaire ou de tous les jours

Base 8 (ou octal) {0,1,2,3,4,5,6,7}

Base 16 (ou héxadécimal) {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

raccourci d'écriture de la base 2

Méthodes de conversion

Problème : exprimer le même nombre dans des bases différentes

Sous problèmes : de b^m vers b
de b vers b^m

de b vers 10
de 10 vers b

de i vers j

Conversion : de 2^m vers 2 / 2 vers 2^m

2^m vers 2 : expansion d'un digit en m bits

2 vers 2^m : regroupement de bits par paquets de m

$$N = a_7.2^7 + a_6.2^6 + a_5.2^5 + a_4.2^4 + a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0$$

$$= (a_7.2^3 + a_6.2^2 + a_5.2^1 + a_4.2^0).2^4 + (a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0)$$

$$= (a_7.2^3 + a_6.2^2 + a_5.2^1 + a_4.2^0).16^1 + (a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0).16^0$$

$$N = b_1.16^1 + b_0.16^0$$

$$0 \leq a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0 \leq 15$$

Ecriture de $(622,663)_8$ en base 2 et base 16 ?

$(622{,}663)_8$?

6 2 2 , 6 6 3	base 8
110 010 010 , 110 110 011	base 2
1 1001 0010 , 1101 1001 1	base 2
1 9 2 , D 9 8	base 16

Conversion B vers 10

Application de la forme polynomiale

si B = 2 $\quad (10001101)_2 = 2^7 + 2^3 + 2^2 + 1 = (141)_{10}$

Conversion 10 vers B (1)

Première méthode : soustraction $(363)_{10}$ en base 2 ?

Recherche de la puissance 2 juste supérieure = $2^9 = 512$

$363 - 2^8 = 107$	1	MSB
2^7 trop grand	0	
$107 - 2^6 = 43$	1	
$43 - 2^5 = 11$	1	
2^4 trop grand	0	
$11 - 2^3 = 3$	1	
2^2 trop grand	0	
$3 - 2^1 = 1$	1	
1	1	LSB

$$(363)_{10} = (101101011)_2$$

Conversion 10 vers B (2)

Autre exemple : $(363)_{10}$ en base 16 ?

$363 < 16^3 = 4096$

$363 = 1.16^2 + 107$	1
$107 = 6.16^1 + 11$	6
$11 = B.16^0$	B

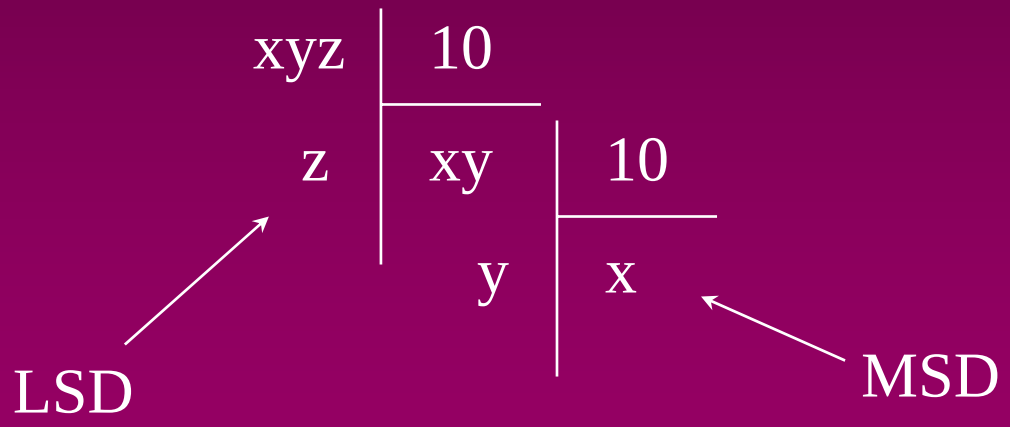
$$(363)_{10} = (16B)_{16}$$

Conversion 10 vers B (3)

inconvénient de la 1ère méthode : il faut connaître les puissances

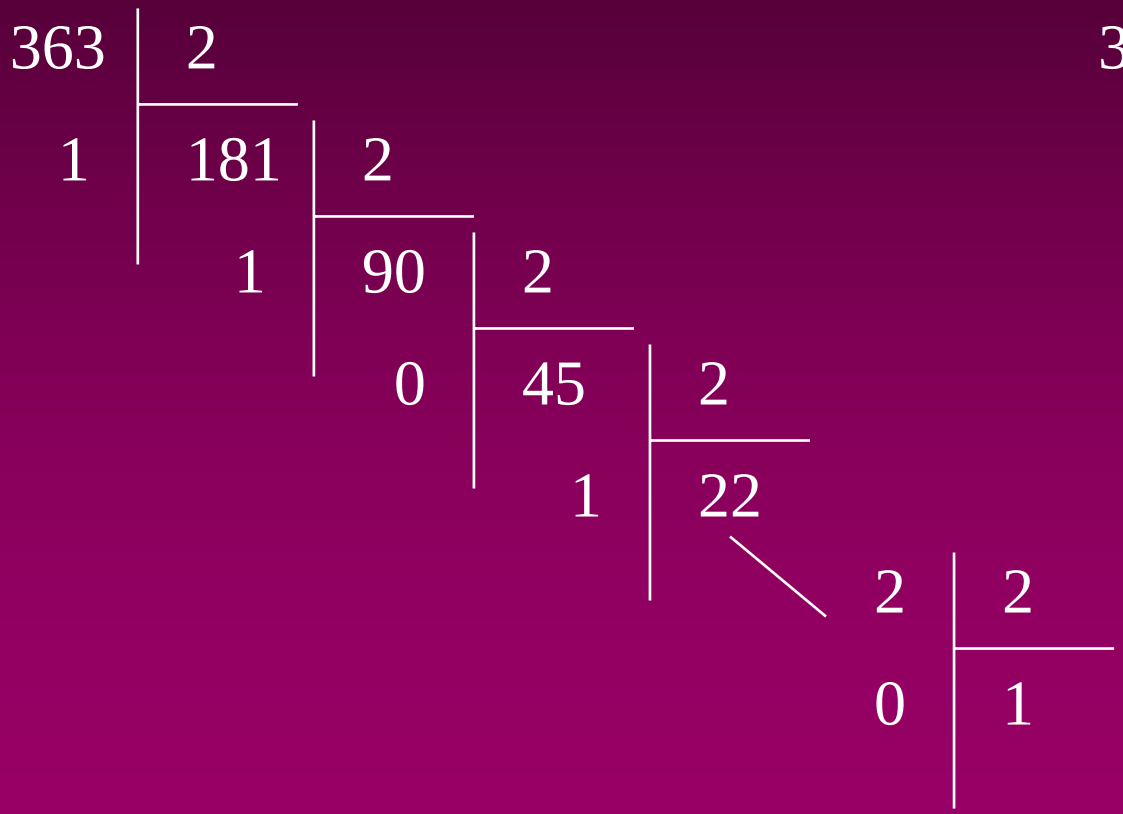
Deuxième méthode : méthode de la division/multiplication

Principe : En base 10 $xyz = xy * 10 + z$

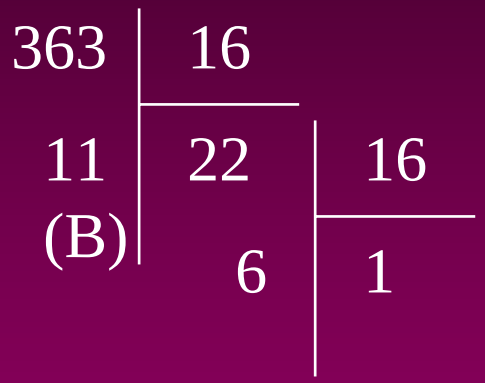


Conversion 10 vers B (4)

$(363)_{10}$ en base 2 ?



$(363)_{10}$ en base 16 ?



Conversion 10 vers B (5)

Pour la partie fractionnaire : algorithme de multiplication

Principe : En base 10 $0,xyz * 10 = x,yz = x + 0,yz$

	0,xyz	* 10	= x,yz	x
partie fractionnaire de x,yz	0,yz	* 10	= y,z	y
partie fractionnaire de y,z	0,z	* 10	= z	z

Conversion 10 vers B (6)

$(0,45)_{10}$ en base 2 ?

0,45 * 2 = 0,90		0
0,90 * 2 = 1,8		1
0,8 * 2 = 1,6		1
0,6 * 2 = 1,2		1
0,2 * 2 = 0,4		0
0,4 * 2 = 0,8		0
0,8 * 2 = 1,6		1
0,6 * 2 = 1,2	..	...

$(0,45)_{10} = (0,0111001...)_2$

Une longueur finie en base 10 peut être infinie en base B
On conserve la précision relative 10^{-3} est approximée par 2^{-10}

Conversion base I vers base J

si $I = B^m$ et $J = B^n$

B^m vers B puis B vers B^m

sinon

I vers 10 puis 10 vers J

Représentations binaires

Définitions :	format	nb de bit de utilisés
	convention	protocole de codage
	dynamique	différence entre le max et le min
	résolution	différence entre deux consécutifs

Exemple : format 8 bits
convention entiers positifs
dynamique 2^8
résolution 1 (constante sur la dynamique)

$(255)_{10} = (1111\ 1111)_2$ $(7)_{10} = (0000\ 0111)_2$

Nombres signés : signe + module

Sur n bits on garde 1 bit pour indiquer le signe

S	Msb xxxxxx Lsb
Signe	Module (positif)
1 bit	n-1 bits

Convention :

S=0 pour positif

S=1 pour négatif

Exemple sur 8 bits : -23 = $(1\ 0010111)_{2,S+M}$

Dynamique : $-(2^{n-1}-1)$ à $(2^{n-1}-1)$

Inconvénient : Deux représentations du zéro

Sur 4 bits +0 = 0000, -0 = 1000

Multiplications faciles

$N_1 * N_2$

$Abs(N_1) * Abs(N_2)$

$S = S_1$ xor S_2

Additions moins simples

Nombres signés : complément restreint (complément à B-1, ou complément à 1)

Def $CR(X) = \overline{X}$

Complément chiffre à chiffre
$(xyz)_b$ donne $(x'y'z')_b$
tel que $x + x' = b^n - 1$

On a $X + CR(X) = b^n - 1$

En binaire 00110 donne 11001
et 00110 + 11001 = 11111

Dans le format considéré $2^n = 0$ Partie interprétée

sur 4 bits : $2^4 = 10000 = 0$

d'où : $CR(X) + 1 = -X$

Nombres signés :complément vrai (complément à B, complément à 2, 2*)

Définition : N^* complément vrai de N sur n chiffres en base B

$N^* = B^n - N = -N$ (rappel : $B^n = 0$ sur n chiffres)

Calcul de l'opposé (sur n bits) :

$N^* = (-N) = 2^n - N = [2^n - 1] - N + 1$

$= [N + CR(N)] - N + 1 = CR(N) + 1$

$$N^* = CR(N) + 1 = CV(N)$$

Complément à 2 (2)

Sur 4 bits :	7	0111	-7	1001
	6	0110	-6	1010
	...			
	0	0000	-0	0000

Remarques : le bit de poids fort = signe (0:positif, 1:négatif)
0 n'a qu'un représentation
1000 jamais rencontré
-(1000) = 0111 + 1 = 1000 (d'où = 0)
et 1000 + 0001 = 1001 = -1 (d'où = -8)

Dynamique sur n bits : $-(2^{n-1}-1)$ à $(2^{n-1}-1)$

Nombres signés : binaire décalé sur m bits (ou excédent 2^{m-1})

On stocke les nombres de m bits comme

$$N_{stocké} = N_{xs} = N_m + 2^{m-1}$$

(translation de la demi-dynamique)

Exemple sur 8 bits : $N_{xs} = N_8 + 128$

Valeur à coder :	-128	...	0 ...	127
	↓		↓	↓
Valeur stockée :	0		128	255

Opération à la restitution $N_{lu} = N_{xs} - 128$

Avantage : on garde la relation d'ordre
on peut effectuer les comparaisons facilement

Remarque : identique au 2* au signe près

Nombres signés : comparaison

N_{10}	N_2	$-N_{S+M}$	$-N_{2,CR}$	$-N_{2*}$	$-N_{2,XS8}$
	mêmes positifs				positifs différents
0	0000	1000	1111	0000	1000
1	0001	1001	1110	1111	0111
2	0010	1010	1101	1110	0110
3	0011	1011	1100	1101	0101
4	0100	1100	1011	1100	0100
5	0101	1101	1010	1011	0011
6	0110	1110	1001	1010	0010
7	0111	1111	1000	1001	0001
8	1000	....	....	(1000)	0000

Remarque :
relation d'ordre
signe du zéro
symétrie
gestion retenues

2* : propriétés

Propriétés : pas de gestion de retenue intermédiare
détection simple d'overflow

sur 4 bits : (-7 à +7)

3	0011
+ 6 (9 = 0F)	0110 = 1001
- 5	1011
= 4	= 1 0100

↑ hors format

X + (- Y) = N avec X > N > -Y

4 + 5 = 9 (of) 0100 + 0101 = 1001

-4 - 5 = -9 (of) 1100 + 1011 = 0111

Note : modification de signe
Indicateur d'overflow :
(dans les microprocesseurs)

$$Fd = S_a.S_b.\overline{S_r} + \overline{S_a}.\overline{S_b}.S_r$$

Nombres non entiers

Dans un calculateur : nombre sous format déterminé (entier, virgule fixe, virgule flottante ...)

TOUT EST QUESTION DE CONVENTION

Quoi associer à 1101100011100110 ?

Caractère ASCII, pixel d'une image, nombre entier 2* nombre fractionnaire ... ?

Dans l'ordinateur (le système numérique) il n'y a pas de virgule

Virgule fixe

Par convention on place la virgule quelque part et on interprète

2^{n-1} 2^{0} avant de placer la virgule

MSB xxxxxx , xxxx LSB

2^{n-1-k} 2^{0},2^{-1} 2^{-k} avec la virgule au rang k

Dynamique : 2^{n-1-k}

Résolution : 2^{-k} # 0

Virgule fixe : analyse

Bon format pour l'addition :

```
   12,32
+  23,45
=  35,77  La virgule reste fixe
```

Mauvais format pour la multiplication :

```
    13,4
 *  10,1
= 135,34  dépassement à gauche (overflow)
          dépassement à droite (arrondi)
```

Virgule fixe : solution

Problèmes réglés si les nombres sont inférieurs à 1 :

$$\begin{array}{l} 0{,}87 \\ *\ 0{,}74 \\ \hline = 0{,}64\,38 \end{array}$$

- On place la virgule toujours à gauche
- On utilise un autre groupe de bit pour connaître la position de la virgule

⟹ Format virgule flottante

Format virgule flottante

$N = M.b^E$

M = mantisse en 2* de forme 0,xxx
b = base de l'exponentiation (2 ou 16)
E = exposant en binaire décalé

On stocke la chaine de bit **ME** dans le calculateur

Exemple : codage de PI sur 5 chiffres de mantisse
et 2 chiffres d'exposant (en décimal)

→ $0{,}3141.10^1 = 0{,}0003.10^4$!!! PI*10000=3

On dit qu'un flottant est normalisé quand le premier chiffre significatif est juste derrière la virgule (précision maximum)

Virgule flottante : calcul/stockage

Multiplication : $M_1.b^{E_1} * M_2.b^{E_2} = M_1.M_2.b^{(E_1+E_2)}$

Addition : $M_1.b^{E_1} + M_2.b^{E_2} = M_1.b^{(E_1-E_2)}.b^{E_2} + M_2.b^{E_2}$

$= (M_1.b^{(E_1-E_2)} + M_2).b^{E_2}$

dénormalisation du plus petit nombre (vers la droite)

Si $E_2 > E_1$

puis renormalisation

il faut comparer facilement

⟹ Exposant codé en binaire décalé

S_M	EXPOSANT	MANTISSE

Virgule flottante : IEEE 754/854

Norme internationale : IEEE 754 flottant sur 32 bits

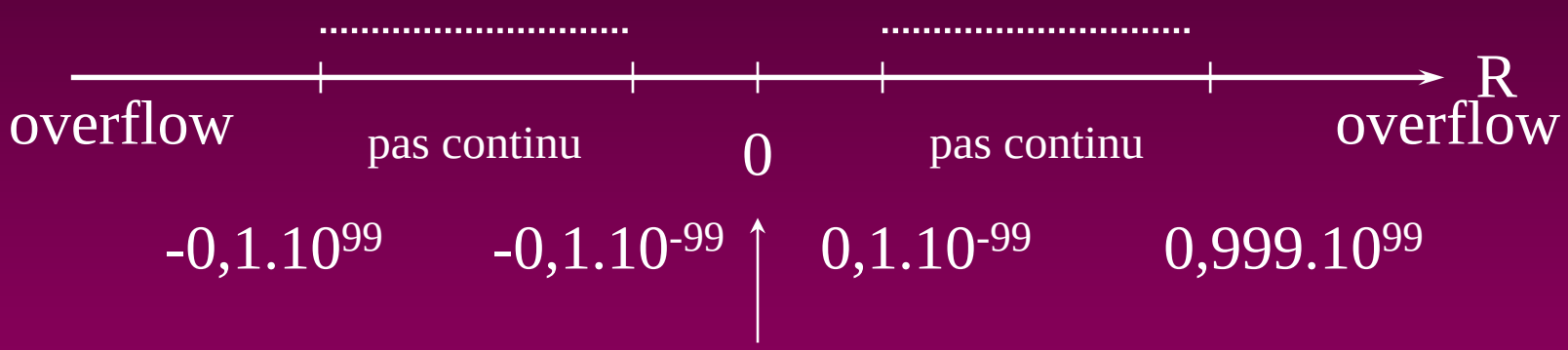
b_{31} .... b_0
signe mantisse, exposant, mantisse
1 bit 8 bit 23 bits

Le bit de signe est 1 pour négatif et 0 pour positif
La mantisse vaut toujours 1,xxxx et on ne stocke que xxxx
L'exposant est en excédent 127
La valeur 0 correspond à des 0 partout (en fait $1,0.2^{-127}$)

exemple : 1 10000011 11000000000000000000000 = $-1,75.2^4 = -28$
0 01111111 00000000000000000000000 = $1,0.2^0 = 1$

Virgule flottante : performances

Exemple : (10) Mantisse 3 chiffres $0{,}999 \geq |M| \geq 0{,}1$
Exposant 2 chiffres -99 à 99



convention spéciale (non normalisé)

ON NE MANIPULE JAMAIS L'ENSEMBLE DES REELS