

Informatique Générale

Représentation des nombres

Jacques Farré

Jacques.Farre@unice.fr

<http://deptinfo.unice.fr/~jff/InfoGene>

Représentation des nombres

Nous sommes habitués à représenter les nombres sous forme **décimale**, avec une notation de **position**, et en utilisant le **zéro**

Par exemple, nous écrivons **107** pour le nombre **cent-sept**, soit **un** multiplié par **dix** au carré plus **sept**, ou $1 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$

Mais cela n'a pas toujours été le cas

En chiffres romain : **CVII**, soit **cent** plus **cinq** plus **deux**

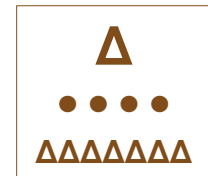
En chiffres sumériens (30 siècles avant JC), mélange de système décimal et sexagésimal (en base 60) :

soixante (Δ) plus **quatre** fois **dix** (\bullet) plus **sept** fois **un** (Δ)

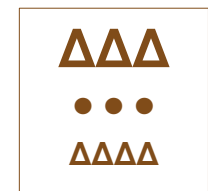
(ces chiffres pouvaient être représentés par des cailloux -calculi en latin- ou des jetons d'argile plus ou moins gros)



$$60 + 40 + 7 = 107$$



$$180 + 30 + 4 = 214$$



Mais dans tous les cas, $107 \times 2 = 214$: **CCXIV** ou

Ne pas confondre nombre et chiffre

Représentation binaire

Pour des raisons technologiques (passage de courant ou non, sens de polarisation, trou ou bosse ...), les ordinateurs actuels ne manipulent que 2 « chiffres »

Les nombres sont donc représentés en base 2, les chiffres étant des **bits** (BInary digiT), que l'on note par convention 0 et 1

Un nombre s'écrivant $B_n B_{n-1} \dots B_1 B_0$ en binaire aura pour valeur :

$$B_n \times 2^n + B_{n-1} \times 2^{n-1} + \dots + B_1 \times 2^1 + B_0 \times 2^0$$

Par exemple, 1011 en binaire = 11 en décimal car
 $1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_{10}$

Addition binaire

Puisqu'un chiffre binaire vaut 0 ou 1, toute valeur ≥ 2 impliquera une retenue

Par exemple ($11_{10} + 14_{10}$) :

$$\begin{array}{r} 1 1^1 0^1 1 \\ + 1 1 0 \\ \hline = 1 1 0 1 \end{array}$$

de droite à gauche :

- **1+0 = 1, je pose 1**
- **1+1 = 2, je pose 0 et je retiens 1**
- **¹+0+1 = 2, je pose 0 et je retiens 1**
- **¹+1+1 = 3, je pose 1 et je retiens 1**
- **¹, je pose 1**

C'est bien 25_{10}

Questions :

Donner les puissances de 2 de 2^0 à 2^{15}

Que vaut le nombre binaire 11111111 ? Et 11111111 + 1

Quelle est l'écriture binaire du nombre décimal 1026 ?

Comment évaluer approximativement le nombre binaire $10\dots 0$ ($10 \times n$ zéros) ?

Représentation binaire (réponse aux questions précédentes)

Les puissances de 2 (puisque $2^n = 2 \times 2^{n-1}$, il suffit de multiplier par 2)

$$\begin{array}{ccccc} 2^0=1 & 2^1=2 & 2^2=4 & 2^3=8 & 2^4=16 \\ 2^5=32 & 2^6=64 & 2^7=128 & 2^8=256 & 2^9=512 \\ 2^{10}=1024 & 2^{11}=2048 & 2^{12}=4096 & 2^{13}=8192 & 2^{14}=16384 \dots \end{array}$$

Le nombre binaire 11111111_2 : $128+64+32+16+8+4+2+1=255_{10}$
noter que $11111111_2 + 1 = 100000000_2 = 256_{10}$

De manière générale : $2^n + 2^{n-1} + \dots + 2^1 + 2^0 = 2^{n+1} - 1$

La représentation binaire du nombre 1026_{10} : $1024_{10} + 2_{10}$, soit $2^{10} + 2^1$,
donc 10000000010_2

$10\dots0_2$ (avec $10 \times n$ zéros) = 2^{10n}

$2^{10} = 1024$ et $10^3 = 1000$, donc $2^{10} \approx 10^3$, et donc $2^{10n} \approx 10^{3n}$

$2^{20} \approx 10^6$ (1 million), $2^{30} \approx 10^9$ (1 milliard), $2^{32} = 2^2 \times 2^{30} \approx 4 \times 10^9$ (4 milliards)

Représentation binaire

(autre façon de répondre aux questions précédentes)

Le nombre binaire 10110010 ($128+32+16+2=178_{10}$)

10110010 : $0 \times 2 + 1 = 1$
10110010 : $1 \times 2 + 0 = 2$
10110010 : $2 \times 2 + 1 = 5$
10110010 : $5 \times 2 + 1 = 11$
10110010 : $11 \times 2 + 0 = 22$
10110010 : $22 \times 2 + 0 = 44$
10110010 : $44 \times 2 + 1 = 89$
10110010 : $89 \times 2 + 0 = 178$

Repose sur le schéma dit *de Horner* (en fait de *Newton*) :

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = 0 + a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + x a_n) \dots))$$

La représentation binaire du nombre 1026_{10} (10000000010_2)

$1026 = 2 \times 513 + \mathbf{0}$, je mets un 0 à droite : **0**
 $513 = 2 \times 256 + \mathbf{1}$, je mets un 1 : **10**
 $256 = 2 \times 128 + \mathbf{0}$, je mets un 0 : **010**
 $128=2 \times 64, 64=2 \times 32, 32=2 \times 16, 16=2 \times 8, 8=2 \times 4, 4=2 \times 2$: **000000010**
 $2 = 2 \times 1 + \mathbf{0}, 1 = 2 \times \mathbf{0} + \mathbf{1}$: **10000000010**

Octets et mots

Puisque l'information est représentée par une suite continue de 0 et de 1, où commence et où fini un nombre dans la mémoire ou sur un disque ?

Regroupement des bits par paquets de 8 : l'*octet*

Un octet peut coder **256** valeurs, de 0 à 255 ($2^8 - 1$)

Il sert à coder toutes sortes d'informations : nombres, caractères, couleurs, sons ...

Mais se limiter à 256 valeurs pour les nombres, c'est un peu court !

Les octets sont eux-même regroupés, par exemple par paquets de 2, 4 ou 8 pour les nombres : les *mots*

Sur 4 octets (donc 32 bits), on peut coder les entiers de 0 à $2^{32}-1$ (soit un peu plus de 4 milliards : $2^{32} \approx 4 \times 10^9$ comme vu précédemment)

Notation hexadécimale

La représentation binaire est longue à lire pour les humains : 8 bits pour un octet, et jusqu'à 64 bits pour un nombre

On utilise aussi une notation plus concise, **en base 16**

Comme $16 = 2^4$, 2 chiffres hexadécimaux suffisent pour représenter la valeur d'un octet

Pour les valeurs de 0 à 9, on utilise les chiffres décimaux, et pour les valeurs de 10 à 15, on utilise les 6 premières lettres de l'alphabet latin : A, B, C, D, E, F

Le nombre hexadécimal $\mathbf{H_n H_{n-1} \dots H_1 H_0}$ aura pour valeur :

$$\mathbf{H_n} \times 16^n + \mathbf{H_{n-1}} \times 16^{n-1} + \dots + \mathbf{H_1} \times 16^1 + \mathbf{H_0} \times 16^0$$

$$15AACF7_{16} = 1 \times 16^6 + 5 \times 16^5 + 10 \times 16^4 + 10 \times 16^3 + 12 \times 16^2 + 15 \times 16^1 + 7 \times 16^0 = 22\,719\,735_{10}$$

Notations hexadécimale et binaire

$16 = 2^4$, on a donc les correspondances suivantes :

décimal	binaire	hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
	...	
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

$$15AACF7_{16} = 0001\ 0101\ 1010\ 1010\ 1100\ 1111\ 0111_2$$

Questions

Quelles sont les représentations hexadécimale et binaire du nombre 27_{10}

Quelles sont les représentations binaire et décimale du nombre $FB3_{16}$

Notations hexadécimale et binaire (réponse aux questions précédentes)

Représentations hexadécimale et binaire de 27_{10}

$27 = 16 + 11$, donc

1B en hexadécimal

0001 1011 en binaire (soit $16 + (8 + 2 + 1)$)

1 hexa \rightarrow 0001 binaire, B hexa \rightarrow 1011 binaire

Représentations binaire et décimale de $FB3_{16}$

$$15 \times 16^2 + 11 \times 16^1 + 3 \times 16^0 = 3840 + 176 + 3 = 4019_{10}$$

ou encore (*schéma de Horner*) : $(15 \times 16 + 11) \times 16 + 3$

FB3 en binaire : F B 3
 1111 1011 0011

Codage des caractères, des couleurs ...

Les caractères sont codés sur 1, 2 ou 4 octets

7 bits pour l'ASCII (donc 128 caractères), par exemple

41_{16} (A), $7A_{16}$ (z), 20_{16} (l'espace) 30_{16} (le **chiffre** 0)

8 bits pour l'ISO : ajoute aux 128 caractères ASCII des variantes nationales, par exemple (ISO 8859-15 pour le français) :

$C0_{16}$ (À), $E6_{16}$ (æ)

Aussi, codages sur 16 ou 32 bits pour alphabets non latins (nombre variable de bits pour UTF-8)

Les couleurs sont (en général) codées sur 4 octets

Le plus souvent, 3 octets codent l'intensité de chacune des 3 couleurs (**rouge**, **vert**, **bleu**) composant la teinte, et 1 octet est inutilisé, ou sinon il code la transparence

Par exemple, la couleur  sera codée **FB****B****0****97**, le noir 000000 et le blanc FFFFFFFF

Représentation des nombres réels en *virgule flottante*

On peut représenter un nombre avec une *mantisse* et un *exposant* : vous connaissez la *notation scientifique* :

$$1,234 \times 10^3 (=1234)$$

Imaginez que la mantisse et l'exposant doivent être représentés sur un **nombre fixe de chiffres** (respectivement 5 et 2 par exemple), et qu'il faille ôter 50 de l'exposant (le *biais*, pour avoir des exposants négatifs) ; pour 1234, on aurait alors

mantisse	exposant	
12340	53	soit $(1+2/10+3/10^2+4/10^3) \times 10^{53-50}$

0,0005 (5×10^{-4}) serait représenté par

50000	46
-------	----

Dans toute base B, décaler de n chiffres à droite revient à diviser par Bⁿ

Approximation et nombres réels en virgule flottante

On ne peut pas représenter tous les nombres réels en virgule flottante avec une taille limitée pour la mantisse et l'exposant

Avec une mantisse de 5 chiffres et un exposant de 2

le plus grand nombre représentable est

99999	99
-------	----

soit $9,9999 \times 10^{99-50} \approx 1 \times 10^{50}$

le plus petit (strictement positif) est :

00001	00
-------	----

soit $(1/10^4) \times 10^{-50} = 10^{-54}$

Il est clair qu'on ne peut pas représenter **exactement** 1,23456, puisque la mantisse ne peut avoir que 5 chiffres : il devra être arrondi, par exemple à 1,2346

L'utilisation des nombres flottants peut produire des erreurs de calcul dus à ces arrondis

Représentation des nombres réels binaires en virgule flottante

Les machines utilisent une représentation flottante pour les nombres réels, mais en binaire

La représentation binaire en virgule flottante est analogue à celle en décimale, mais en base 2 évidemment

Avec par exemple une mantisse sur 5 bits, un exposant sur 3 bits et un biais D valant 3

mantisse **exposant**

$$\boxed{10110} \boxed{110} = (1 + 1/4 + 1/8) \times 2^{6-3} = 11,$$

ou (autre façon de calculer) : $(11 \times 2^{-3}) \times 2^3$
(puisque $1011_2 = 11_{10}$, $1,0110_2 = 11_{10} \times 2^{-3}$)

Représentation des nombres réels binaires en virgule flottante

La mantisse $b_0 b_1 \dots b_m$ représente le nombre **rationnel**

$$m = b_0 + b_1/2 + b_2/2^2 + \dots + b_m/2^m,$$

(ou, si m' est le nombre binaire $b_0 b_1 b_2 \dots b_{m'}$, $m = m' \times 2^{m-1}$)

L'exposant $x_n x_{n-1} \dots x_1 x_0$, représente l'**entier** $x = x_n 2^n + \dots + x_1 2 + x_0$

La valeur représentée par le couple (m, x) est **$m \cdot 2^{x-D}$**

En forme *normalisée*, on a toujours $b_0 = 1$

La **norme IEEE 754** définit 2 formats (un peu plus compliqués que le modèle ci-dessus, car les mantisses sont signées)

- Simple précision : mantisse sur 23 bits, exposant sur 8, $D=127$
- Double précision : mantisse sur 52 bits, exposant sur 11, $D=1023$
- Calculs faits en précision *étendue* : mantisse sur au moins 64 bits, exposant sur au moins 15

Représentation des nombres réels

attention aux pièges

La représentation entière ne permet de compter que jusqu'à l'ordre de grandeur 4×10^9 (sur 32 bits), alors que la représentation flottante permet de représenter les nombres entre les ordres de grandeur 10^{-37} et 10^{37} (sur 32 bits) ou 10^{-308} et 10^{308} (sur 64 bits) : on serait tenté de n'utiliser qu'elle !

Mais cette faculté d'exprimer de très petits ou de très grands nombres se paye par une approximation puisque seuls peuvent être représentés **exactement** les nombres de la forme $(b_0 + b_1/2 + b_2/2^2 + \dots + b_m/2^m) \times 2^{x-D}$

On a donc des **erreurs d'arrondi** : par exemple sur 32 bits, $1000 \times (1/3000 + 1/3000 + 1/3000) \neq 1$

$m \cdot 2^x = m/2 \times 2^{x+1}$, et par exemple

(avec biais valant 0)

11000...0 | 0...010

$(1+1/2) \times 4 = 6$

01100...0 | 0...011

$(1/2+1/4) \times 8 = 6$

Un même nombre a donc plusieurs représentations flottantes : sur n bits, on représente moins de nombres réels flottants *différents* que de nombres entiers !

Avez vous bien compris ?

Supposons qu'on utilise une mantisse de 2 bits et un exposant de 2 bits avec $D=0$

Questions :

Quelles sont les nombres qui peuvent être représentés ?

Combien de ces nombres sont différents ?

Quel est le résultat de l'addition flottante $4 + 1$?

Combien d'entiers peuvent être codés sur 4 bits ?

Quel est le résultat de l'addition entière $4 + 1$?

Les réponses

Les 4 configurations de la mantisse sont

00 (0), 01 (1/2), 10 (1), 11 (3/2)

et l'exposant représente 1 (2^0), 2 (2^1), 4 (2^2), 8 (2^3)

Les nombres représentables sont donc, avec un biais D de 0 :

0 1/2 1 3/2 (exposant = 0)

0 1 2 3 (exposant = 1)

0 2 4 6 (exposant = 2)

0 4 8 12 (exposant = 3)

soit 10 nombres différents seulement : 0, 1/2, 1, 3/2, 2, 3, 4, 6, 8, 12

Notez que, puisque $6 = 110_2$, sa mantisse (sur 2 bits) est 11 (on garde bien sûr les bits de plus fort poids) et son exposant est 2 (car la mantisse représente la valeur $1 + 1/2$, ou plus simplement, pour passer de 110_2 à $1,1_2$, on a dû décaler la virgule de 2 à gauche)

On voit que $5 = 4 + 1$ n'est pas représentable, même si 1 et 4 le sont : comme $5 = 101_2$, on perd le 1 de droite avec une mantisse sur 2 bits, et il sera approximé par 4 ou 6 ;

Sur 4 bits, on code tous les entiers de 0 à 15, donc on aura bien $4 + 1 = 5$

Avez vous vraiment bien compris ?

On prend une mantisse sur 5 bits et un exposant x sur 3 bits avec un biais de 4 : le nombre représenté vaut

$$(b_0 + b_1/2 + b_2/2^2 + b_3/2^3 + b_4/2^4) \times 2^{x-4}$$

10100	011
-------	-----

 $101_2 = 5$, donc $5 \times 2^{-2} \times 2^{3-4} = 5/8$, ou si vous préférez vous compliquer la vie, $(1+1/4) \times 2^{3-4} = 1/2 + 1/8 = 4/8 + 1/8$

00110	110
-------	-----

 $3 \times 2^{-3} \times 2^{6-7} = 3/2^4$, ou $(1/4 + 1/8) \times 2^{-1} = 1/2^3 + 1/2^4$

Pour représenter 12 :

11000	111
-------	-----

 puisque la mantisse représente 12×2^{-3} , il faut un exposant de 3 (+ 4 de biais, donc 7)

Pour représenter 0,3125 :

10100	010
-------	-----


 $0,3125 = 5/16$, et puisque la mantisse vaut 5×2^{-2} , l'exposant doit être -2 (+4 de biais, donc 2)

Les nombres négatifs

Avant l'utilisation des bouliers (à partir du 13^e siècle), les chinois comptaient à l'aide de baguettes (dès -700) dans un système décimal de position en virgule flottante (l'écriture des nombres était inspirée de l'arrangement de ces baguettes)

Les baguettes étaient orientées verticalement pour les puissances de la forme 10^{2p} et horizontalement pour celles correspondant à 10^{2p+1} ,

par exemple 256  (ou 25600, ou 2.56, mais pas 2560 ni 25.6)

Le 0 était implicite : un espace entre 2 «chiffres» de même orientation :  représente 206 (ou plus exactement 206×10^{2p}), mais pas 26

On trouve des traces des nombres négatifs dans des sources du 3^e siècle

La couleur des baguettes ou de l'encre (noire ou rouge) indiquait si le nombre était négatif ou positif

Mais c'est d'Inde (6^e siècle) puis de Perse (9^e siècle) que vient à l'occident (vers le 15^e siècle) la notion de quantité négative (notion encore rejetée par certains mathématiciens à la fin du 18^e siècle !)

Les nombres négatifs pour les ordinateurs

Vous écrivez -17 sans vous posez de question

Vous **pourriez** écrire en binaire -10001 (puisque $17_{10} = 10001_2$)

Mais les machines ne connaissent que 0 et 1, pas le signe -

Donc par convention, **le premier bit représente le signe S (0 si positif ou nul, 1 si négatif)**

On **pourrait** imaginer qu'un nombre (par exemple sur 8 bits) aurait la représentation suivante

1 bit	7 bits	
1	0010001	exemple avec -17

Pour additionner 2 nombres de même signe, pas de problème ; mais s'ils sont de signe différent ?

En décimal, pour $-17 + 35$, ce n'est pas si simple (on calcule $35-17$)

Ce ne serait pas mieux en binaire (et il faudrait donc un circuit électronique compliqué pour faire un additionneur)

Il faut donc trouver une autre représentation

Les nombres négatifs pour les ordinateurs (complémentation à 1)

Si on inverse les bits, $17 = 0\ 0010001$
 $-17 = 1\ 1101110$

Et donc 0 a deux représentations : $+0$, soit $00\dots0$
 -0 , soit $11\dots1$

ce qui ne facilite pas la comparaison d'un nombre à 0

Et on aimerait **pouvoir additionner le bit de signe** comme les autres (pour simplifier les circuits)

Et effectivement, $17+(-17) = -0$, et $17-17 = +0$

Mais malheureusement, $5-22$ donne -18 , et $-17+22$ donne 4 ; il manque 1, ce n'est pas encore tout à fait ça

On peut aussi remarquer que $-0+1$ donne $+0$

Il faut encore trouver une autre représentation

Les nombres négatifs pour les ordinateurs (complémentation à 2^n)

Donc si $17 = 00010001$,
essayons avec $-17 = 11101110 + 1$, soit 11101111

C'est bien le résultat de $5 - 22$ (c'est à dire $5 + (-22)$), et on peut vérifier que $-17 + 22 = 00000101$, c'est à dire 5
Bingo !

Cette représentation, appelée *complémentation à 2^n* (ou simplement *complémentation à 2*), est celle des ordinateurs actuels

Principe : inverser les bits et ajouter 1

Par exemple sur 8 bits :

$$10_{10} = 00001010_2, -10_{10} = 11110110_2$$

$$10011101_2 = -99_{10}, 99_{10} = 01100011_2$$

Intervalle des nombres signés sur n bits : $[-2^{n-1} : 2^{n-1}-1]$

Les nombres négatifs pour les ordinateurs

La complémentation à 2 permet des calculs **modulo 2^n** :

- on a vu que sur 8 bits -10 s'écrit 11110110
- mais c'est aussi la représentation binaire (non signée) de 246 : $246 + 10 = 256 \equiv 0 \pmod{256}$
- **en règle générale, si k est un nombre positif, la représentation sur n bits de $-k$ est la même que celle de $2^n - k$**

On en déduit par exemple que la représentation (sur 8 bits)

- de -1 est la même que celle de 255 (11111111)
- et celle de -128 la même que 128

Les nombres négatifs pour les ordinateurs

11...1 (-1) ; c'est aussi le plus grand nombre non signé : $2^n - 1$

00...0 (0)

00...1 (1)

en nombres non signés,
 $2^{n-1} + 1 = 0 !$

< 0

≥ 0

en nombres signés,
 $2^{n-1} - 1 + 1 = -2^{n-1} !$

(-maxint) **10...1**

(minint = -2^{n-1} , le plus petit négatif) **10...0**

01...1 (le plus grand positif signé, maxint = $2^{n-1} - 1$)