Informatique

10

Représentation des nombres

Résumé

CC BY NC SA

Représentation des nombres

Binaires entiers

Définition

Un nombre codé en binaire sur n bits est écrit sous la forme d'une chaîne de 0 et 1.
On appelle mot m l'ensemble des bits représentant le nombre en binaire :
$m = 000110110$
On trouve parfois l'écriture $m = 0b000110110$ pour spécifier que c'est un binaire

Propriétés

Un entier codé sur n bits peut prendre 2^n valeurs différentes (2 possibilités par bit), 0 inclus
Le plus grand entier représenté sur n bits vaut donc $2^n - 1$
Exemple sur 8 bits : $255_{(10)} = 11111111_{(2)} = 2^8 - 1$

Les premiers entiers

Base 10	Base 2	Base 10	Base 2	Base 10	Base 2
0	0	6	110	11	1100
1	1	7	111	13	1101
2	10	8	1000	14	1110
3	11	9	1001	15	1111
4	100	10	1010	16	10000
5	101	11	1011	17	10001

Transcodage

Base 10 → Binaire

Diviser par 2 l'entier et diviser le quotient obtenu jusqu'à ce qu'il soit nul.
Remonter les restes de chaque division pour créer le mot binaire.

	1000	2									
n_0	0	500	2								
	n_1	0	250	2							
		n_2	0	125	2						
			n_3	1	62	2					
				n_4	0	31	2				
					n_5	1	15	2			
						n_6	1	7	2		
							n_7	1	3	2	
								n_8	1	1	2
									n_9	1	**0**

```
>>> 0b1111101000
1000
```

$1000_{(10)} = 1111101000_{(2)}$

```
>>> bin(1000)
'0b1111101000'
```

Binaire → Base 10

Il suffit de sommer les puissances de 2 associées à chaque bit

$$1111101000_{(2)}$$
$$= 1.2^9 + 1.2^8 + 1.2^7 + 1.2^6 + 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 0.2^0$$
$$= 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 = 512 + 256 + 128 + 64 + 32 + 8$$
$$1111101000_{(2)} = 1000_{(10)}$$

Entiers multi-précision

Lorsque les entiers dépassent la plus grande valeur stockable de $2^n - 1$, ils sont codés en multi-précision, ce qui revient à les représenter par une liste de leurs termes ($N = 111; L = [1,1,1]$), puis d'utiliser des algorithmes spécifiques réalisant les opérations arithmétiques de base (addition, soustraction, multiplication et division). La seule limite est alors la taille mémoire utilisée pour les stocker.

CC BY NC SA

Binaires réels

Principe

En base 10, lorsque nous manipulons des nombres à virgule :

- Les chiffres avant la virgule sont des puissances de 10
- Les chiffres après la virgule sont des puissances de 1/10

C'est la même chose en base 2, en puissances de 2 ou 1/2

Transcodage

Base 10 → Binaire

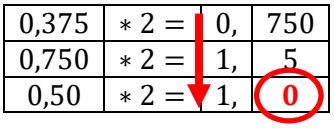
Décomposer le réel en sa partie entière et sa partie décimale :

$$5{,}375 = 5 + 0{,}375$$

Transcoder la partie entière comme vu précédemment : $5_{(10)} = 101_{(2)}$

Transcoder la partie décimale de la sorte :

- La multiplier par 2. Séparer l'entier avant la virgule (0 ou 1) de la nouvelle partir décimale
- Recommencer jusqu'à obtenir une partie décimale nulle (si existe)
- Récupérer la partie décimale binaire

Principe	Présentation améliorée
$0{,}375 * 2 = 0{,}750$ $\rightarrow \begin{cases} Bit\ 0 \\ Nouvelle\ partie\ décimale\ 0{,}75 \end{cases}$ $0{,}750 * 2 = 1{,}5$ $\rightarrow \begin{cases} Bit\ 1 \\ Nouvelle\ partie\ décimale\ 0{,}5 \end{cases}$ $0{,}50 * 2 = 1$ $\rightarrow \begin{cases} Bit\ 1 \\ Nouvelle\ partie\ décimale\ 0 \end{cases}$	
$(0{,}375)_{10} = (0{,}011)_2$	

Finalement : $(5{,}375)_{10} = (101{,}011)_2$

Remarque : on peut réaliser ce transcodage en notation scientifique binaire : $(0{,}011)_2 = (11)_2 . 2^{-3} = \frac{3}{8} = 0{,}375$

Binaire → Base 10

Il suffit de sommer les puissances de 2 associées à chaque bit

$$101{,}011_{(2)}$$
$$= 1.2^2 + 0.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3}$$
$$= 4 + 1 + \frac{1}{4} + \frac{1}{8} = 5 + 0{,}25 + 0{,}125 = 5 + 0{,}375 = 5{,}375$$

CC BY NC SA

Entiers relatifs

Principe

Associer à 2^n entiers naturels N codés de manière classique sur n bits 2^n entiers relatifs Z selon une formule $N = f(Z)\ ou\ N = f^{-1}(Z)$.

Complément à 2

Le principe consiste à utiliser un entier naturel N (pour $n = 3$ bits, $N \in [0,7]$), et de lui associer un entier relatif Z tel que :

$$N = \begin{cases} Z\ si\ Z \geq 0 \\ 2^n - |Z| = 2^n + Z\ si\ Z < 0 \end{cases} \Leftrightarrow Z = \begin{cases} N\ si\ N \leq 2^{n-1} - 1 \\ N - 2^n\ si\ N > 2^{n-1} - 1 \end{cases}$$

$$N \in [0, 2^n] \quad ; \quad Z \in [-2^{n-1}, 2^{n-1} - 1]$$

$(Z)_{10}$	-4	-3	-2	-1	0	1	2	3
$(N)_{10}$	4	5	6	7	0	1	2	3
$(N)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$

Remarque : On voit que le premier bit est un bit de signe

Avantage : La somme binaire de deux entiers relatif Z fonctionne 😊

$$(-3)_{10} + (1)_{10} \leftrightarrow (101)_2 + (001)_2 = (110)_2 \leftrightarrow (-2)_{10}$$

Exemple de transcodage sur 8 bits :

$(N)_2$	$(N)_{10}$	$(Z)_{10}$
$(11010111)_2$	$(215)_{10}$	$(-41)_{10}$

Codage par excès

Le principe consiste à utiliser un entier naturel N (pour $n = 3$ bits, $N \in [0,7]$), et de lui associer un entier relatif Z « décalé ». Deux décalages semblent logiques :

- Décaler de 3 : 0 devient -3 et 7 devient 4
- Décaler de 4 : 0 devient -4 et 7 devient 3

Le choix est fait de procéder au premier décalage dont l'expression en fonction de n s'écrit :

$$B = 2^{n-1} - 1$$

$$Z = N - B \Leftrightarrow N = Z + B$$

$$N \in [0, 2^n] \quad ; \quad Z \in [-2^{n-1} + 1, 2^{n-1}]$$

$(Z)_{10}$	-3	-2	-1	0	1	2	3	4
$(N)_{10}$	0	1	2	3	4	5	6	7
$(N)_2$	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$

Exemple de transcodage sur 8 bits ($B = 127$) que nous utiliserons plus tard :

N	Z
0	-127
255	128

$(N)_2$	$(N)_{10}$	$(Z)_{10}$
$(10101100)_2$	$(86)_{10}$	$(-41)_{10}$

CC BY NC SA

Virgule flottante – Norme IEEE 754

Contexte

Il est impossible de représenter une infinité de nombres dans l'ordinateur. Une représentation officielle a été choisie, permettant de représenter les réels en machine à l'aide de « flottants », ensemble fini de nombres décimaux

Exemple

$-289456{,}5$ en base 10			
$-2{,}894565.10^5$			
$-$	2	894565	5
Signe	Caractéristique	Mantisse	Puissance
	Partie significative		

$-289456{,}5$ en virgule flottante			
$-(1{,}00011010101011000 01)_2.2^{18}$			
$-$	1	$00011010101011000 01$	18
Signe	Caractéristique	Mantisse	Puissance
	Partie significative		

La caractéristique est toujours égale à 1 sauf le cas exceptionnel de 0

Le bit de signe vaut $\begin{cases} s = 0 \Leftrightarrow x > 0 \\ s = 1 \Leftrightarrow x < 0 \end{cases}$

La puissance (entier relatif Z), est codée sur n bits (8 en simple précision) en un entier naturel N : $N = Z + (2^{n-1} - 1)$. Dans ce cas, $18 + 127 = (145)_{10} = (10010001)_2$

1	10010001	$00011010101011000 01$
Signe	Puissance	Mantisse

En simple précision, la mantisse est codée sur 23 bits, on complète donc de quelques 0 :

Représentation	1	10010001	00011010101011000010000
	Signe	Puissance	Mantisse
32 bits	1 bit	8 bits	23 bits

Le nombre $-289456{,}5$ est codé en virgule flottante simple précision (32 bits):

$$11001000100011010101011000010000$$

Norme

	Nombre de bits	Bit de signe	Bits exposant	Bits implicite	Bits mantisse	Décalage
Simple précision	32	1	8	1	23	127
Double précision	64	1	11	1	52	1023
Quadruple précision	128	1	15	1	112	16383

Remarques

Certains nombres sont réservés, il existe une représentation normalisée et une représentation dénormalisée (cf cours), etc.. Nous n'abordons ici que la structure des nombres pour comprendre qu'ils puissent avoir des conséquences sur nos calculs.

CC BY NC SA

<table>
<tr><td rowspan="2">Transcodage
En 32 bits</td><td>Virgule
flottante
32 bits
→
Réel</td><td>
$11000001001001110000000000000000$
<table>
<tr><td colspan="3">Identification des 3 parties</td></tr>
<tr><td>1</td><td>10000010</td><td>01001110000000000000000</td></tr>
<tr><td>Signe</td><td>Puissance</td><td>Mantisse</td></tr>
</table>
Le signe est négatif ; $(10000010)_2 = (130)_{10}$
$\mathrm{N} = 130 \quad ; \quad \mathrm{Z} = \mathrm{N} - (2^{n-1} - 1) = 130 - 127 = 3$
Sans oublier d'ajouter le bit implicite, la partie significative vaut : $(\mathbf{1}, 0100111)_2 = 1{,}3046875$;
$x = -1{,}3046875.2^3 = -10{,}4375$
</td></tr>
<tr><td>Réel
→
Virgule
flottante
32 bits</td><td>
$x = (-10{,}125)_{10} = (-1010{,}001)_2 = (-1{,}010001)_2.2^3$
$\mathrm{Z} = 127 \quad ; \quad \mathrm{N} = 3 + (2^{n-1} - 1) = 3 + 127 = 130$
$(130)_{10} = (10000010)_2$
<table>
<tr><td colspan="3">$(-10{,}125)_{10}$</td></tr>
<tr><td>1</td><td>10000010</td><td>01000100000000000000000</td></tr>
<tr><td>Signe</td><td>Puissance</td><td>Mantisse</td></tr>
<tr><td colspan="3">$11000001001000100000000000000000$</td></tr>
</table>
</td></tr>
<tr><td rowspan="3">Limites
(cf cours)</td><td>Justesse</td><td>
C'est la correspondance entre nombre représenté et réel associé).
On peut représenter une quantité finie de nombres :
<table>
<tr><td>32 bits</td><td>4 294 967 296</td></tr>
<tr><td>64 bits</td><td>18 446 744 073 710 000 000</td></tr>
</table>
La justesse est assurée si le nombre de bits de la mantisse suffit à la représenter complètement.
</td></tr>
<tr><td>Min
Max</td><td>
<table>
<tr><td>Valeurs normalisées</td><td>Valeur min normalisée</td><td>Valeur min dénormalisée</td><td>Valeur max</td></tr>
<tr><td>Simple précision</td><td>$1{,}2.10^{-38}$</td><td>$1{,}4.10^{-45}$</td><td>$3{,}4.10^{38}$</td></tr>
<tr><td>Double précision</td><td>$2{,}2.10^{-308}$</td><td>$4{,}9.10^{-324}$</td><td>$1{,}8.10^{308}$</td></tr>
</table>
</td></tr>
<tr><td>Ecart entre nombres successifs et applications en virgule flottante</td><td>
ODG de l'écart ΔV entre $A = a.10^n$ avec $a \epsilon [0\,;10[$ et le suivant :
<table>
<tr><td>Simple précision</td><td>Double précision</td><td rowspan="2">$7 \approx 23 \log 2$
$16 \approx 52 \log 2$</td></tr>
<tr><td>10^{n-7}</td><td>10^{n-16}</td></tr>
</table>
Une erreur d'arrondi peut conduire à $A = B + \Delta V$ alors que A et B devraient être égaux → Etudier $|A - B| < \varepsilon$, avec ε de l'ordre de grandeur de ΔV du tableau précédent… Attention, c'est un ODG !
On prendra $\varepsilon = 10^{n-6}$ (32) ou $\varepsilon = 10^{n-15}$ (64) par exemple
ε trop faible ⇒ Le test dit $A \neq B$ alors que $A = B$
ε trop grand ⇒ Le test dit $A = B$ alors que $A \neq B$
ODG du nombre de chiffres significatifs (caractéristique + mantisse) en base 10, auxquels on peut avoir confiance :
<table>
<tr><td>Simple précision</td><td>Double précision</td></tr>
<tr><td>7</td><td>16</td></tr>
</table>
</td></tr>
<tr><td>Conséquences</td><td colspan="2">Les erreurs d'arrondis ou l'overflow (nombre non représentable, induisant le codage d'une mauvaise valeur) ont déjà eu de lourdes conséquences (explosion de la fusée Ariane 5, Missile Patriot passant à travers un système anti-missile…)
Attention au test == – Attention aux sommes répétées
1 et 1.0 sont fondamentalement différents – Attention à l'évaluation d'une dérivée</td></tr>
<tr><td>Numpy</td><td colspan="2">

```
a = np.float32(0.1) - b = np.uint8(10)
A = np.array([1,2],dtype='int8')
```

</td></tr>
</table>

CC BY NC SA