Représentation des nombres

Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-11-22

Plan

1 Base de numération

1.1 Rappel de CP : la base dix

Chiffre : symbole utilisé pour représenter certains entiers.

Les chiffres «usuels» : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Rôle particulier du nombre dix :

- Plus petit nombre non représentable par un chiffre. chiffre pour le représenter.
- Pour compter des objets en grand nombre, on les regroupe par paquets de dix.

Exemple : Pour compter ||||||||||||||||||||||, on obtient

||||||||| ||||||||| ||||

Deux paquets (deux dizaines), reste quatre unités.

Quand il y a trop de dizaines, on regroupe les dizaines par paquets de dix (centaines), les centaines par paquets de dix (milliers), etc.

1.2 Numération de position

- Décomposition d'un entier en dizaines, centaines, milliers, etc.
- Essentiel : strictement moins de dix éléments de chaque type de paquet.
- Regroupement des paquets par type, comptage par type, représentation par un chiffre pour chaque type.
- Écriture : tous les chiffres à la suite.
- À gauche, les *chiffres de poids fort* (gros paquets).
- À droite, les *chiffres de poids faible*.

Ainsi 2735 représente deux milliers plus sept centaines plus trois dizaines plus cinq unités.

De manière générale : $\underline{a_n a_{n-1} \dots a_1 a_0}_{10} = \sum_{k=0}^{n} a_k B^k$, où B vaut dix.

1.3 Pourquoi dix ?

Pourquoi regrouper par dix pas plutôt par deux ? ou trois ? ou six ? ou huit ?

- Raison anthropomorphique (dix doigts) et poids de l'histoire.
- À peu près aucune raison mathématique.

1.4 Autres possibilités

- Deux : Amérique du Sud et Océanie.
- Cinq : Afrique, Romains et Maya (partiellement).
- Six : Papouasie Nouvelle-Guinée.
- Huit : certains dialectes amérindiens (Pame, Mexique ; Yuki, Californie), proposition de Charles XII de Suède.
- Douze : Népal, Europe.
- Vingt : Bhoutan, Aztèques, Maya, Gaulois (?), Basques (?).
- Soixante : Babyloniens, Indiens et Arabes (trigo)

(source : Wikipédia, article *Numération*)

1.5 Base huit (octale)

– Besoin de huit symboles pour représenter les huit chiffres (représentant les nombres de zéro inclus à huit exclu). On prendra 0, $\ldots$, 7.

– Pour compter, on regroupe les unités par huitaine puis par huitaines de huitaines, $\ldots$

– Même principe pour noter les nombres : $\underline{2735}_8$ représente cinq unités plus trois huitaines plus sept huitaines de huitaines, plus deux huitaines de huitaines de huitaines (total 1501).

– De manière générale

$$\underline{a_n a_{n-1} \dots a_1 a_0}_8 = \sum_{k=0}^{n} a_k B^k \text{ où } B \text{ vaut huit}$$

Comptons en octal :

$\underline{0}_8 = 0$
$\underline{1}_8 = 1$
$\underline{2}_8 = 2$
$\underline{3}_8 = 3$
$\underline{4}_8 = 4$
$\underline{5}_8 = 5$
$\underline{6}_8 = 6$
$\underline{7}_8 = 7$

$\underline{10}_8 = 8$
$\underline{11}_8 = 9$
$\underline{12}_8 = 10$
$\underline{13}_8 = 11$
$\underline{14}_8 = 12$
$\underline{15}_8 = 13$
$\underline{16}_8 = 14$
$\underline{17}_8 = 15$

$\underline{20}_8 = 16$
$\underline{21}_8 = 17$
$\underline{22}_8 = 18$
$\underline{23}_8 = 19$
$\underline{24}_8 = 20$
$\underline{25}_8 = 21$
$\underline{26}_8 = 22$
$\underline{27}_8 = 23$

$\underline{30}_8 = 24$
$\underline{40}_8 = 32$
$\underline{50}_8 = 40$
$\underline{60}_8 = 48$
$\underline{70}_8 = 56$

$\underline{100}_8 = 64$
$\underline{200}_8 = 128$
$\underline{1\,000}_8 = 512$
$\underline{10\,000}_8 = 4096$
$\underline{100\,000}_8 = 32768$

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Table d'addition en octal

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Table de multiplication en octal

Comment faire une addition de nombres à plusieurs chiffres ?

Exactement comme en base 10, mais en utilisant la table d'addition de la base 8.

Pour la multiplication : aussi.

1.6 Base seize (hexadécimale)

On manque de chiffres pour représenter les nombres de zéro inclus à seize exclu (il en manque six).

On rajoute de nouveaux chiffres :

a dix

b onze

c douze

d treize

e quatorze

f quinze

$\underline{0}_{16} = 0$
$\underline{1}_{16} = 1$
$\vdots$
$\underline{9}_{16} = 9$
$\underline{a}_{16} = 10$
$\underline{b}_{16} = 11$
$\underline{c}_{16} = 12$
$\underline{d}_{16} = 13$
$\underline{e}_{16} = 14$
$\underline{f}_{16} = 15$

$\underline{10}_{16} = 16$
$\underline{11}_{16} = 17$
$\vdots$
$\underline{19}_{16} = 25$
$\underline{1a}_{16} = 26$
$\underline{1b}_{16} = 27$
$\underline{1c}_{16} = 28$
$\underline{1d}_{16} = 29$
$\underline{1e}_{16} = 30$
$\underline{1f}_{16} = 31$

$\underline{20}_{16} = 32$
$\underline{21}_{16} = 33$
$\vdots$
$\underline{29}_{16} = 41$
$\underline{2a}_{16} = 42$
$\underline{2b}_{16} = 43$
$\underline{2c}_{16} = 44$
$\underline{2d}_{16} = 45$
$\underline{2e}_{16} = 46$
$\underline{2f}_{16} = 47$

$\underline{30}_{16} = 48$
$\underline{40}_{16} = 64$
$\vdots$
$\underline{90}_{16} = 144$
$\underline{a0}_{16} = 160$
$\underline{b0}_{16} = 176$
$\underline{c0}_{16} = 192$
$\underline{d0}_{16} = 208$
$\underline{e0}_{16} = 224$
$\underline{f0}_{16} = 240$

$\underline{100}_{16} = 256$
$\underline{200}_{16} = 512$
$\underline{1\,000}_{16} = 4096$
$\underline{10\,000}_{16} = 65536$

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table d'addition en hexadécimal

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Table de multiplication en hexadécimal

1.7 Base deux

$\underline{0}_2 = 0$

$\underline{1}_2 = 1$

$\underline{10}_2 = 2$

$\underline{11}_2 = 3$

$\underline{100}_2 = 4$

$\underline{101}_2 = 5$

$\underline{110}_2 = 6$

$\underline{111}_2 = 7$

$\underline{1000}_2 = 8$

$\underline{1\,0000}_2 = 16$

$\underline{10\,0000}_2 = 32$

$\underline{100\,0000}_2 = 64$

$\underline{1000\,0000}_2 = 128$

$\underline{1\,0000\,0000}_2 = 256$

+	0	1
0	0	1
1	1	10

Table d'addition en binaire

*	0	1
0	0	0
1	0	1

Table de multiplication en binaire

Intérêts du binaire

1. Facilement représentable par un dispositif mécanique/électrique/-électronique/optique/électromagnétique...
2. Tables d'opérations très simples, facilement calculable par un dispositif mécanique/électrique/électronique.

Système utilisé pour représenter les nombres en interne dans un ordinateur.

Écriture d'un naturel p en binaire

1. Algorithme par divisions successives : voir livre.
2. Calculer 2^k pour $k = 0, \ldots$ jusqu'à avoir $2^k > p$. Alors p s'écrit sur k bits et le bit de poids fort est 1. Le reste des bits est donné par la représentation en binaire de $p - 2^{k-1}$.

Calcul d'entier représenté en binaire

Calcul de l'entier p représenté par une suite de bits $\underline{a_n a_{n-1} \ldots a_1 a_0}_2$: calculer $\sum_{k=0}^{n} a_k 2^k$ (voir livre).

Remarques

1. S'ils sont bien mis en œuvre, ces algorithmes de conversion demandent un temps de calcul en $\Theta(n^2)$ pour un nombre de n chiffres (binaires ou décimaux), soit en $\Theta\left((\log p)^2\right)$.
2. Il existe des algorithmes plus efficaces. Meilleure complexité connue : complexité d'une multiplication de nombres de n chiffres, soit $O(n \log n \log\log n)$ [Knuth].
3. Peu importe la base dans laquelle vous faites vos calculs, ces algorithmes permettent de convertir entre la base 2 et votre base habituelle.

2 Représentation des entiers sur ordinateur

2.1 Cadre

Sur un ordinateur récent :

1. mots-machine de 64 bits (8 octets).
2. opérations d'addition/multiplication d'entiers internes au processeur : sur 64 bits.

De manière générale, on s'intéressera au fonctionnement sur des ordinateurs travaillant sur des mots de n bits ($n \geq 2$), mais pour les exemples, on prendra systématiquement $n = 16$.

2.2 Somme de naturels

Dans un processeur n bits :

1. Un registre du processeur a n bits et peut représenter tout entier de $[\![0, 2^n[\![$.
2. Lorsqu'on effectue l'addition de deux registres r_1 et r_2 pour stocker le résultat dans r_3, le registre fait n bits : s'il y a une retenue, elle est perdue. [1]

   Exemple : après addition de $\underline{1111\,0000\,1111\,0000}_2$ et $\underline{0011\,0011\,0011\,0011}_2$ sur 16 bits, registre résultat : $\underline{0010\,0100\,0010\,0011}_2$.

1. En fait une trace en est généralement gardée dans un autre registre du processeur.

Troncature d'un entier p à ses n bits de poids faibles : valeur du reste de la division de p par 2^n (noté $p \,\%\, 2^n$).

Définitions :

1. Soit $p \in \mathbb{N}$. p *représentable comme entier non signé sur* n *bits* si $p \in [\![0, 2^n[\![$.
2. *Représentation de* p *comme entier non signé sur* n *bits* : suite des n chiffres de son écriture en binaire.
3. Abus de notation : on identifie $[\![0, 2^n[\![$ et les représentations sur n bits.
4. Soit $(p, q) \in [\![0, 2^n[\![^2$. *somme (non signée) de* p *et* q *sur* n *bits* : $(p + q) \,\%\, 2^n$, notée $p +_n q$ (notation non canonique).

Remarques pour $(p, q) \in [\![0, 2^n[\![^2$:

1. $p +_m q \equiv p + q \ [2^n]$.
2. $p +_n q = p + q$ si $p + q < 2^n$.
3. $p +_n q = p + q - 2^n$ si $p + q \geq 2^n$.

2.3 Entiers relatifs

Avec signe et valeur absolue

Première possibilité : $n-1$ bits pour la valeur absolue et 1 bit pour le signe, par ex. :

- on représente -4 par $\underline{1000\,0000\,0000\,0100}_2$;
- on représente 4 par $\underline{0000\,0000\,0000\,0100}_2$.

Inconvénients :

1. Deux représentations pour zéro ($\underline{1000\,0000\,0000\,0000}_2$ et $\underline{0000\,0000\,0000\,0000}_2$).
2. Plus compliqué à additionner que les entiers naturels.

Représentation quasiment jamais utilisée pour les nombres *entiers* d'un processeur.

Complément à deux

Idée géniale :

1. L'addition d'entiers naturels sur le processeur n'est pas correcte mais l'est modulo 2^n ;
2. pour tout $p \in \mathbb{Z}$, $p \,\%\, 2^n \in [\![0, 2^n[\![$;
3. donc $p \,\%\, 2^n$ représentable sur n bits ;
4. l'addition de ces entiers relatifs sera correcte modulo 2^n.

Exemples :

1. -5 est codé par $-5 \,\%\, 2^{16} = 65531 = \underline{1111\ 1111\ 1111\ 1011}_2$.
2. 3 est codé par $3 \,\%\, 2^{16} = 3 = \underline{0000\ 0000\ 0000\ 0011}_2$.
3. La somme obtenue par le processeur est

$$\underline{1111\ 1111\ 1111\ 1110}_2 = 65534 = 2^{16} - 2 = -2 \,\%\, 65536$$

4. -4 est codé par $-4 \,\%\, 2^{16} = 65532 = \underline{1111\ 1111\ 1111\ 1100}_2$.
5. 6 est codé par $6 \,\%\, 2^{16} = 6 = \underline{0000\ 0000\ 0000\ 0110}_2$.
6. La somme obtenue par le processeur est

$$\underline{0000\ 0000\ 0000\ 0010}_2 = 2 = 2 \,\%\, 65536$$

Définitions :

1. Soit $p \in [\![0, 2^n[\![$. *Complément à deux de p sur n bits* : $(-p)\,\%\,2^n$, noté $c_n(p)$ (non canonique).
2. Soit $p \in \mathbb{Z}$. p est *représentable comme entier signé sur n bits* si $p \in [\![-2^{n-1}, 2^{n-1}[\![$.
3. Soit $p \in [\![-2^{n-1}, 2^{n-1}[\![$. *Représentation en complément à deux de p* : $p\,\%\,2^n$, notée $r_n(p)$ (non canonique).

Remarques :

1. c_n involution [2] de $[\![0, 2^n[\![$.
2. Soit $p \in [\![0, 2^n[\![$. $c_n(p) = 2^n - p$ si $p \neq 0$, 0 si $p = 0$.
3. r_n bijection de $[\![-2^{n-1}, 2^{n-1}[\![$ sur $[\![0, 2^n[\![$.
4. $\forall p \in [\![-2^{n-1} + 1, 2^{n-1}[\![\quad r_n(-p) = c_n(r_n(p))$.
5. Pour tout $(p, q) \in [\![-2^{n-1}, 2^{n-1}[\![^2$ on a $r_n^{-1}\left(r_n(p) +_n r_n(q)\right) \equiv p + q \ [2^n]$.
6. On a l'égalité si $p + q \in [\![-2^{n-1}, 2^{n-1}[\![$ (en particulier si p et q de signes différents).

2. Bijection d'un ensemble sur lui-même qui est sa propre bijection réciproque.

7. Le bit de poids fort de $r_n(p)$ vaut 1 si et seulement si $p < 0$.

Intérêt du complément à deux : Pour calculer une addition, on utilise exactement les mêmes circuits électroniques que pour des entiers non signés !

Calcul de la représentation en complément à deux

Définition *Complément à un sur n bits de* $\underline{a_{n-1}\dots a_0}_2$: $\underline{b_{n-1}\dots b_0}_2$ où $b_k = 1 - a_k$ pour $k \in [\![0, n[\![$. On note $c'_n(p)$ cette valeur.

Proposition Pour tout $p \in [\![0, 2^n[\![$,

$$p + c'_n(p) = \sum_{k \in [\![0,n[\![} 2^k = 2^n - 1$$

Conséquence Pour tout p entier non signé sur n bits,

$$c_n(p) = c'_n(p) +_n 1$$

Conséquence (bis) Pour tout $p \in [\![-2^{n-1} + 1, 2^{n-1}[\![$

$$r_n(-p) = c'_n(r_n(p)) +_n 1$$

Exemple (sur 16 bits) :

1. Représentation de 5 en tant qu'entier non signé ?
2. Complément à un de 5 ?
3. Complément à deux de 5 ?
4. Représentation de -5 sur 16 bits ?
5. Calcul de l'opposé de l'entier signé $\underline{1111\,1111\,1111\,1000}$?
6. Que vaut l'entier signé $\underline{1111\,1111\,1111\,1000}$?

Soustraction

Soit $(p, q) \in [\![2^{n-1} - 1, 2^{n-1}[\![$.

Définition *Différence sur n bits de p et q :* $(p-q) \,\%\, 2^n$, notée $p -_n q$ (non canonique).

Proposition $p -_n q = p +_n c_n(q)$.

1. Complément à deux facile à calculer.
2. Addition/Soustraction : utilisation du même circuit !

2.4 Dans les langages de programmation

Dans de nombreux langages (C, Java, ...) :

$$\text{Entiers du langage} = \text{Entiers sur } n \text{ bits}$$

Dans ces langages, sur une machine 64 bits, `4 * 2**62` vaut `0`.

Dans quelques langages :

1. Les entiers ne sont pas les entiers machines. Plusieurs représentation possibles. Possibilités classiques : tableau dont les éléments sont des octets/mots machines/chiffres dans une base B (avec B puissance de 2 ou 10).
2. Des fonctions internes au langage prennent soin d'effectuer les opérations correctement (en utilisant le fait que le processeur sait calculer sur n bits)

Python est dans ce cas.

3 Octal et hexadécimal en informatique

1. Représenter des données par une succession de chiffres binaires est bien adapté à l'utilisation de l'électronique pour construire des ordinateurs.
2. C'est également bien adapté au calcul sur les entiers.

Exemple de représentation par du binaire :

- Une adresse IPv6 est une suite de 128 bits. Exemple d'adresse :

  00100000000000010000011001111100
  00000010111010000000000000100010
  00000000000000000000000000000000
  11000001000000000000011010001011

- Une adresse IPv4 est une suite de 32 bits. Exemple :

  10000001101011110000111100001011

Ce n'est pas très pratique à manipuler...

Comment trouver une représentation plus simple à manipuler ?

Deux possibilités :

1. Considérer ces suites de bits comme des entiers binaires et convertir en décimal (difficile à faire de tête)
2. Grouper ces suites de bits, par exemple par octet, remplacer chaque octet par sa représentation décimale.

Adresses IPv4 : 2e solution. Sur l'adresse précédente : $129.175.15.11$

2e solution : pas pratique si la suite de bits représente un entier (difficile ensuite de calculer sur la représentation).

Autre solution :

1. Grouper les chiffres binaires par 4 (en partant de la droite)
2. Remplacer chaque groupement par le chiffre hexadécimal correspondant.

Avantages :

1. Facile à faire à la main et même de tête.
2. A du sens mathématiquement : le nombre hexadécimal obtenu a la même valeur que le nombre binaire de départ.

Solution prise pour les adresses IPv6.

Exemples :

- traduire en hexadécimal le nombre $\underline{\texttt{11 0101 0010 1010 1110}}_2$
- traduire en binaire le nombre $\underline{\texttt{90f5e56712}}_{16}$

Pour l'octal : même principe mais en groupant par 3.

1. Binaire, octal et hexadécimal sont très utilisés en informatique.
2. En Python, notations autorisées :

```
>>> 0b101010
42
>>> 0o52
42
>>> 0x2a
42
```

3. Écriture d'un entier `n` en décimal, binaire, octal ou hexadécimal : `str(n)`, `bin(n)`,`oct(n)`, `hex(n)`.

Danger, ancienne notation encore autorisée en Python 2 mais à ne plus utiliser :

```
>>> 052
42
```

En python 3 :

```
>>> 052
  File "<stdin>", line 1
    052
      ^
SyntaxError: invalid token
```