Représentation des données en machine 1/2

Mardi 14 Novembre 2017

Michael FRANÇOIS

francois@esiea.fr

https://francois.esiea.fr/

Objectif de ce cours

- Rappels sur quelques systèmes de numération historiques.
- Comprendre les concepts de base binaire, octale et hexadécimale.
- Comprendre comment sont représentés les entiers naturels, relatifs et les réels en machine.
- Comprendre les limites de ces représentations et les conséquences possibles.

Système de numération

La représentation des quantités (la numération) est l'une des plus vieilles pratiques humaines car, elle permet entre autres :

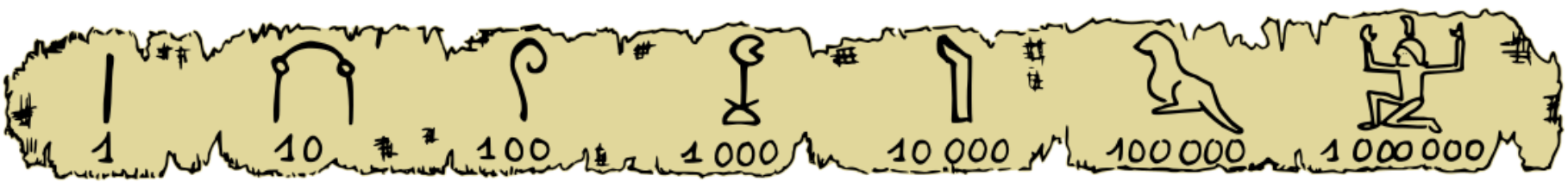
- de recenser les ressources disponibles (nombre de chasseurs, de proies, de graines, ...) au sein d'un groupe d'individus ;
- l'échange entre groupes ;
- de manipuler une connaissance mathématique qui peut être très complexe (astronomie ou architecture par exemple).

Quelques systèmes de numération historiques

Numération égyptienne (-3000)

- Les égyptiens utilisaient un système de numération additionnel : chacun des hiéroglyphes (chiffres) suivants correspondait à une quantité donnée et les nombres étaient donc représentés par la juxtaposition de ces hiéroglyphes.

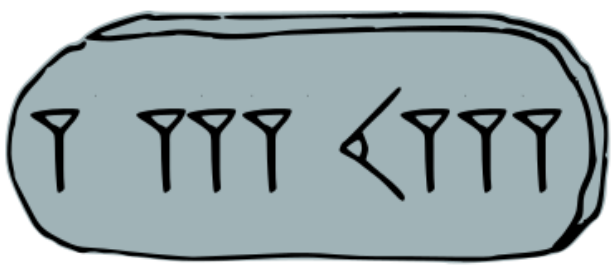
- Les chiffres sont représentés par des hiéroglyphes :



- Le système est additionnel :

Numération babylonienne (-1800)

- Les babyloniens utilisaient uniquement deux symboles :
  - les clous  pour l'unité ;
  - les chevrons  pour les dizaines.
- Le système est additionnel et positionnel et surtout en base 60 :



vaut $1 \times 3600 + 3 \times 60 + 13 = 3793$

Pratiquement chaque civilisation importante avait défini son propre système de numération. Tous ces systèmes sont entièrement définis par :

- leur base b qui est le nombre de symboles élémentaires différents disponibles ;
- la représentation de tout nombre sous une forme polynomiale, c'est à dire une loi permettant de passer d'un ensemble de digits au nombre N lui même. Par exemple en base 10 on a :

  $$(N)_{10} = (a_k.10^k + a_{k-1}.10^{k-1} + ... + a_1.10^1 + a_0.10^0)_{10}$$

  Par exemple le nombre **105 230** peut être représenté sous la forme :

  $$\mathbf{1}.10^5 + \mathbf{0}.10^4 + \mathbf{5}.10^3 + \mathbf{2}.10^2 + \mathbf{3}.10^1 + \mathbf{0}.10^0$$

En informatique, les bases utilisées sont :

- la base binaire (ou base 2), composée des symboles élémentaires : 0 et 1 ;
- la base octale (ou base 8), avec les symboles élémentaires : 0, 1, 2, 3, 4, 5, 6, 7 ;
- la base hexadécimale (ou base 16), avec les symboles élémentaires : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

• En prenant un formalisme mathématique, écrire un nombre N en base 10 équivaut à l'écrire sous la forme $(a_n a_{n-1} \dots a_1 a_0)_{10}$ où les coefficients a_i correspondent au polynôme :

$$N = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \cdots + a_1 \times 10^1 + a_0 \times 10^0$$

• La valeur 1234 s'écrit par exemple $(1234)_{10}$ en base 10, car nous pouvons écrire :

$$\mathbf{1234} = \mathbf{1} \times 10^3 + \mathbf{2} \times 10^2 + \mathbf{3} \times 10^1 + \mathbf{4} \times 10^0$$

• Il est possible de généraliser cette démarche pour n'importe quelle base B, via la formule suivante :

$$(N)_B = (a_n a_{n-1} \dots a_1 a_0)_B = a_n \times B^n + a_{n-1} \times B^{n-1} + \cdots + a_1 \times B^1 + a_0 \times B^0$$

Exemples :

- Base binaire (*i.e.* base 2) :

$$13 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

donc en base 2 la valeur 13 s'écrit $(1101)_2$

- Base octale (*i.e.* base 8)

$$83 = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$$

donc en base 8 la valeur 83 s'écrit $(123)_8$

Calculs élémentaires dans les bases

Tous les calculs élémentaires se font de la même manière qu'en base 10 c'est-à-dire en utilisant la notion de retenue.

Exemples : en base 10 :

$$\begin{array}{rr} & \scriptstyle 1 \\ & (\ 1\ 7\ )_{10} \\ +\ & (\ \ \ 8\ )_{10} \\ \hline & (\ 2\ 5\ )_{10} \end{array} \qquad \begin{array}{rr} & (\ 1\ 7\ )_{10} \\ & \scriptstyle 1 \\ -\ & (\ \ \ 8\ )_{10} \\ \hline & (\ \ \ 9\ )_{10} \end{array} \qquad \begin{array}{rr} & (\ \ \ 1\ 7\ )_{10} \\ \times\ & (\ \ \ \ \ 8\ )_{10} \\ \hline & (\ \ \ 5\ 6\ )_{10} \\ & (\ \ \ 8\ \bullet\ )_{10} \\ \hline & (\ 1\ 3\ 6\ )_{10} \end{array}$$

En base 2 :

$$\begin{array}{rcccc} & & \color{red}{1} & & \\ & (\ & 1 & 1\ )_2 \\ + & (\ & 1 & 0\ )_2 \\ \hline & (\ 1 & 0 & 1\ )_2 \end{array} \qquad \begin{array}{rcccc} & (\ 1 & 1 & 0\ )_2 \\ & & \color{red}{1} & \\ - & (\ 1 & 0 & 1\ )_2 \\ \hline & (\ 0 & 0 & 1\ )_2 \end{array} \qquad \begin{array}{rcccc} & (\ & 1 & 1\ )_2 \\ \times & (\ & 1 & 0\ )_2 \\ \hline & (\ & 0 & 0\ )_2 \\ & (\ 1 & 1 & \bullet\ )_2 \\ \hline & (\ 1 & 1 & 0\ )_2 \end{array}$$

Conversion entre la base 2 et la base 8

• Binaire $\Longrightarrow$ octal :

L'astuce consiste à grouper les bits par paquets de 3 (pour obtenir un nombre entre 0 et 7, donc en octal) en partant du bit le plus à droite (bit de poids faible).

Exemple : 111 001 $\Rightarrow$ 57 et 1 111 001 $\Rightarrow$ 121

$$(\ \underbrace{1\,1\,1}\ \ \underbrace{0\,0\,1}\ )_2 \qquad\qquad (\ \underbrace{\ 1}\ \ \underbrace{1\,1\,1}\ \ \underbrace{0\,0\,1}\ )_2$$

$$(\ \ 7\ \ \ \ 1\ \ )_8 \qquad\qquad (\ \ 1\ \ \ 7\ \ \ 1\ \ )_8$$

• Octal $\Longrightarrow$ binaire :

Il suffit de prendre chaque chiffre et le coder sur 3 bits et concaténer les bits résultants.

$$(\ \ 7\ \ \ \ 1\ \ )_8 \qquad\qquad (\ \ 1\ \ \ 7\ \ \ 1\ \ )_8$$

$$(\ \overbrace{1\,1\,1}\ \ \overbrace{0\,0\,1}\ )_2 \qquad\qquad (\ \overbrace{\ 1}\ \ \overbrace{1\,1\,1}\ \ \overbrace{0\,0\,1}\ )_2$$

Conversion entre la base 2 et la base hexadécimale

- Binaire $\Longrightarrow$ hexadécimale :

L'astuce consiste à grouper les bits par paquets de 4 (pour obtenir un nombre entre 0 et 15, donc en hexadécimal) en partant du bit le plus à droite (bit de poids faible).

Exemple : 0101 1100 $\Rightarrow$ 92 et 10 1111 $\Rightarrow$ 47

$$(\ \underbrace{0\,1\,0\,1}\ \ \underbrace{1\,1\,0\,0}\ )_2 \qquad (\ \underbrace{\quad 1\,0}\ \ \underbrace{1\,1\,1\,1}\ )_2$$

$$(\ 5\ \ C\ )_{16} \qquad (\ 2\ \ F\ )_{16}$$

- Hexadécimale $\Longrightarrow$ binaire :

Il suffit de coder directement chaque symbole sur 4 bits et de concaténer les bits résultants.

$$(\ B\ \ A\ \ 5\ \ E\ \ 5\ )_{16}$$

$$(\ \overbrace{1\,0\,1\,1}\ \ \overbrace{1\,0\,1\,0}\ \ \overbrace{0\,1\,0\,1}\ \ \overbrace{1\,1\,1\,0}\ \ \overbrace{0\,1\,0\,1}\ )_2$$

Conversion entre la base 8 et la base hexadécimale

- Octale $\Longrightarrow$ hexadécimale :

L'astuce consiste à passer de la base octale vers la base binaire, puis du binaire à la base hexadécimale.

- Hexadécimale $\Longrightarrow$ octale :

L'astuce consiste à passer de la base hexadécimale vers la base binaire, puis du binaire à la base octale.

Conversion de la base b (b = 2, 8 ou 16) vers la base 10

Le problème est une application directe de la forme polynomiale :

$$(N)_b = (N)_{10} = \sum_{i=0}^{k} (a_i)_{10} (b)_{10}^{i}$$

Quelques exemples :

- binaire $\Rightarrow$ décimal :

$$(1000001100)_2 = (1.2^9 + 1.2^3 + 1.2^2)_{10} = (512 + 8 + 4)_{10} = (524)_{10}$$

- octal $\Rightarrow$ décimal :

$$(1014)_8 = (1.8^3 + 1.8 + 4)_{10} = (512 + 8 + 4)_{10} = (524)_{10}$$

- hexadécimal $\Rightarrow$ décimal :

$$(20C)_{16} = (2.16^2 + 12)_{10} = (512 + 8 + 4)_{10} = (524)_{10}$$

Conversion de la base 10 vers la base b (b = 2, 8 ou 16)

Nous allons voir deux méthodes pour passer de la base 10 à une base b quelconque.

Méthode de Hörner

- Cette méthode consiste à diviser successivement le nombre de départ puis les quotients obtenus, par la base voulue jusqu'à obtenir 0.

- Les restes successifs, **lus à partir du dernier** en remontant, donnent alors le nombre d'arrivé.

- Exemple : décimal $\Rightarrow$ binaire
$(34)_{10} = (100010)_2$ car

34	2	
0	17	2
1	8	2
0	4	2
0	2	2
0	1	2
1	0	

Méthode des puissances

• Cette méthode à connaître (ou à retrouver) toutes les puissances de b et à soustraire au nombre de départ, puis aux restes, la plus grande puissance possible.

• Le nombre d'arrivé est une écriture polynomiale en fonction des différentes grandes puissances soustraites.

Algorithme des puissances : algorithme particulièrement adapté pour des nombres relativement petits.

$(N)_b \Leftarrow 0$
`residue` $\Leftarrow (N)_{10}$
tant que *(`residue` est $\neq$ 0)* **faire**
  $(N)_b \Leftarrow (N)_b +$ `maxPower` /* `maxPower` est la plus grande puissance de b immédiatement $\leq$ `residue` */
  `residue` $\Leftarrow$ `residue - maxPower`
fin

Algorithme des puissances : $(387)_{10}$ s'écrit en base 2 :

- $(387)_{10} = (0)_2$
- *residue* $= (387)_{10}$ non nul
  - *maxPower* $= (256)_{10}$
  - $(387)_{10} = (256)_{10} = (100000000)_2$
- *residue* $= (387)_{10} - (256)_{10} = (131)_{10}$ non nul
  - *maxPower* $= (128)_{10}$
  - $(387)_{10} = (100000000)_2 + (128)_{10} = (110000000)_2$
- *residue* $= (131)_{10} - (128)_{10} = (3)_{10}$ non nul
  - *maxPower* $= (2)_{10}$
  - $(387)_{10} = (110000000)_2 + (2)_{10} = (110000010)_2$
- *residue* $= (3)_{10} - (2)_{10} = (1)_{10}$ non nul
  - *maxPower* $= (1)_{10}$
  - $(387)_{10} = (110000010)_2 + (1)_{10} = (110000011)_2$

Représenter un entier naturel

- Un entier naturel est un entier positif ou nul (*i.e.* appartenant à $\mathbb{N}=\{0, 1, 2, 3, 4, 5, \ldots \infty\}$)

- En informatique, que vaut ∞ ?
  - Problème : avec un nombre fini de bits peut-on représenter un nombre aussi grand qu'on veut ?
  - Pragmatiquement, ∞ c'est le plus grand nombre représentable sur un nombre fini de bits, et dépend donc du type choisi pour représenter la variable

- Choisir le type adapté à la variable revient à chercher le meilleur compromis entre la plage des valeurs possibles et la taille mémoire à réserver.

ATTENTION : grande taille $\Rightarrow$ coût en puissance et coût en mémoire !!!

Type	Nb octets	Descripteur
`unsigned char`	1	`"%hhu"`
`unsigned short` `unsigned short int`	2	`"%hu"`
`unsigned int`	4	`"%u"`
`unsigned long` `unsigned long int`	4 4	`"%lu"`
`unsigned long long` `unsigned long long int`	8	`"%llu"`

Remarque : rien n'interdit que deux types différents possèdent la même taille. Dans une implémentation donnée, la taille des entiers courts est inférieure ou égale à celle des entiers et que celle des entiers est inférieure ou égale à celle des entiers longs.

- Les algorithmes de conversion de la base 10 en base 2 s'appliquent (méthode de Hörner/méthode des puissances)

- Exemple d'une variable codée en `unsigned char`

Valeur bit à bit		$\mathbb{N}$
0000	0000	0
0000	0001	1
...		...
0111	1111	127
...		...
1111	1111	255

Et que vaut $255 + 1$?

Exercice : Remplir les cases manquantes.

Système décimal	0	1	2	3	4	5	6	7	8	9
Système binaire	0	1	10							
Système hexadécimal										

Système décimal	10	11	12	13	14	15	16	17	18	19
Système binaire										
Système hexadécimal		B								

Solution :

Système décimal	0	1	2	3	4	5	6	7	8	9
Système binaire	0	1	10	11	100	101	110	111	1000	1001
Système hexadécimal	0	1	2	3	4	5	6	7	8	9

Système décimal	10	11	12	13	14	15	16	17	18	19
Système binaire	1010	1011	1100	1101	1110	1111	10000	10001	10010	10011
Système hexadécimal	A	B	C	D	E	F	10	11	12	13

Bibliographie

• L. BEAUDOIN, Introduction à l'algorithmique et au langage C (Représentation des données en machine 1/2), cours 1A 2016-2017 ESIEA-Paris.

• C. DELANNOY, Langage C, éditions EYROLLES, 4ème tirage 2005.