Bases de numération

M1103 - Architecture des équipements informatiques

Sebastien.Kramm@univ-rouen.fr

IUT R&T Rouen, site d'Elbeuf

2018-2019

Base de numération

- Un signal va être représenté à un instant t par une valeur numérique (un nombre) N s'exprime dans une base de numération.
  Par exemple, la base 10 est celle utilisée par l'humain.
- Mais le nombre en lui même est **indépendant** de sa base de numération.
- D'un point de vue mathématique, on peut convertir un nombre dans une **autre** base de numération, puis faire l'opération inverse : le nombre est inchangé.
- Les ordinateurs (comme les humains) représentent le nombre de façon avec un nombre de symboles **fini** :
  π : $\underbrace{3,1415}_{\text{5 symboles}}$, ou $\underbrace{3,1415926}_{\text{8 symboles}}$, etc.
- Conséquence :
  - En math : $a + b - a = b$
  - Sur une machine : ... pas toujours.
    Par exemple : $a = 1^{100}$, $b = 1$

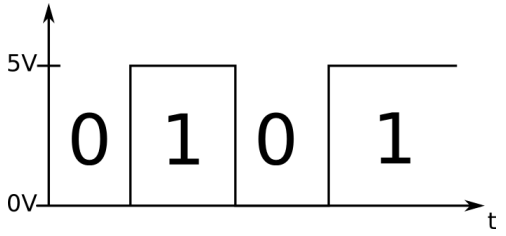
Base de numération, symboles et alphabets

- Le mot "2387" désigne un nombre exprimé en base 10.
- Avec **un** symbole d'une base b, on pourra coder b valeurs.
- Avec n symboles d'une base b, on pourra coder b^n valeurs.
  Exemple : avec 3 symboles de l'alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, je peux coder $10^3 = 1000$ valeurs différentes (0 à 999)
- Un entier naturel a une écriture unique (si on interdit les '0' à gauche) :
  $2387 = 2 \cdot 10^3 + 3 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$
  $= 2 \cdot b^3 + 3 \cdot b^2 + 8 \cdot b^1 + 7 \cdot b^0$ (avec $b = 10$)
- On peut généraliser cette notation à n'importe quelle base b :
  (avec $b > 0$)

$$n = \sum_{i=0}^{p} a_i b^i = a_p b^p + ... + a_2 b^2 + a_1 b^1 + a_0 b^0$$

Codage binaire

- Les ordinateurs travaillent en interne en **binaire** : l'élément de base est le chiffre binaire, le bit (BInary digiT).
  $\Rightarrow$ alphabet à deux symboles : a= {'0' ; '1' }
- Au niveau interne (électronique), ceci correspond à des niveaux de tensions :
  - '0' : 0V
  - '1' : niveau haut (en général, la tension d'alimentation des circuits intégrés, 5V ou moins)



Regroupement de bits

- Pour représenter plus d'information (=des valeurs supérieures à 2), on **associe** plusieurs bits en parallèle.
  $\Rightarrow$ Avec n bits, on peut coder 2^n valeurs différentes.
  - 2 bits $\rightarrow 2^2 = 4$ valeurs différentes (0 à 3)
  - 3 bits $\rightarrow 2^3 = 8$ valeurs différentes (0 à 7)
  - 8 bits $\rightarrow 2^8 = 256$ valeurs différentes (0 à 255)
  - 16 bits $\rightarrow 2^{16} = 65.536$ valeurs différentes
  - 32 bits $\rightarrow 2^{32} = 4.294.967.296$ valeurs différentes

Nombre de bits	8	16	32
Terme anglais	byte	word	double word
Terme français	octet	mot	double mot

Arithmétique binaire

- Les ordinateurs sont dotés d'unités de calcul binaire.
- L'algèbre binaire fonctionne de façon similaire à l'algèbre en base 10.
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 1 = 10$ (=2 en base 10)
  - $1 + 1 + 1 = 11$ (=3 en base 10)
- On fait de l'arithmétique comme en base 10, avec des retenues :

$$\begin{array}{cccccc} & {\scriptstyle 0} & {\scriptstyle 1} & & \\ & 1 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 1 \\ \hline & 1 & 1 & 0 & 1 \end{array}$$

Blague d'informaticien

Il n'y a que 10 sortes de personnes dans le monde :
celles qui comprennent le binaire et celles qui ne le comprennent pas.

Multiplication/ division par la base

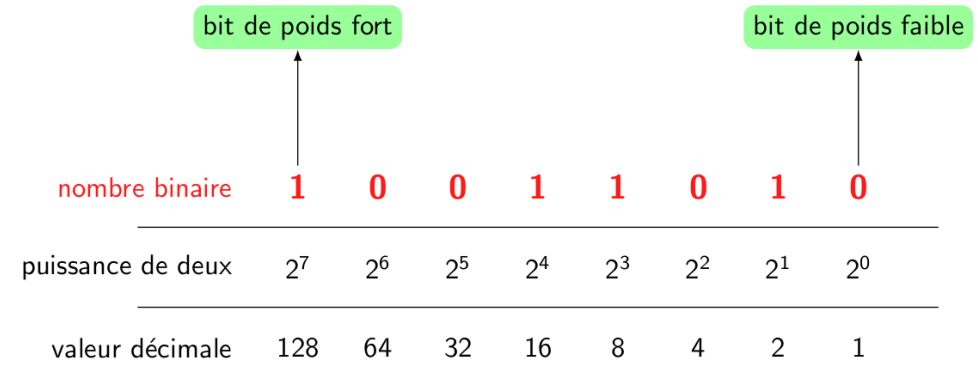
- En base 10 : pour multiplier/diviser par 10, on ajoute/enlève un 0 (= on décale la position de la virgule)
  1230 x 10 = 12300
  1230 / 10 = 123
- Principe identique quel que soit la base de numération !
- En binaire, multiplier et diviser par 2 revient à **décaler les bits**.
  - Division par 2 : 0100.0010 /2 = 0010.0001 $\leftrightarrow$ 66/2 = 33
  - Multiplication par 2 : $0001.1000 \times 2 = 0011.0000 \leftrightarrow 24 \times 2 = 48$

Codage binaire

- On parle de "codage en binaire naturel" :

Valeur (base 10)	Codage en binaire
0	000
1	001
2	010
3	011
4	100
...	...
7	111

- Dans un système informatique, les bits sont regroupés par 8
- Un groupe de 8 bits s'appelle un **octet**.
  $2^8 = 256$ valeurs possibles, numérotées de 0 à 255.



source : J. Landré, IUT Troyes

- MSB (*Most Significant Bit*) : "bit le plus signifiant", on dit "bit de poids fort"
- LSB (*Least Significant Bit*) : "bit le moins signifiant", on dit "bit de poids faible".

Hexadécimal : base 16

- Problème du binaire : difficile à lire par l'humain...
- Exemple : le nombre 1010101111001101 est-il plus grand ou plus petit que le nombre 1010101111101101 ?
- Pour "montrer" une valeur binaire à un humain, on a donc adopté une représentation en base 16 : l'**hexadécimal**.
- Les 16 symboles sont les 10 du système décimal, plus les lettres A,B,C,D,E,F

Symbole	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Valeur associée	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Question : Pourquoi la base 16, et pas la base 9 ou 14 ?
- Réponse :
  - **un** symbole hexa correspond à **quatre** bits. ($2^4 = 16$)
  - **deux** symboles hexa correspondent à **huit** bits = 1 octet.

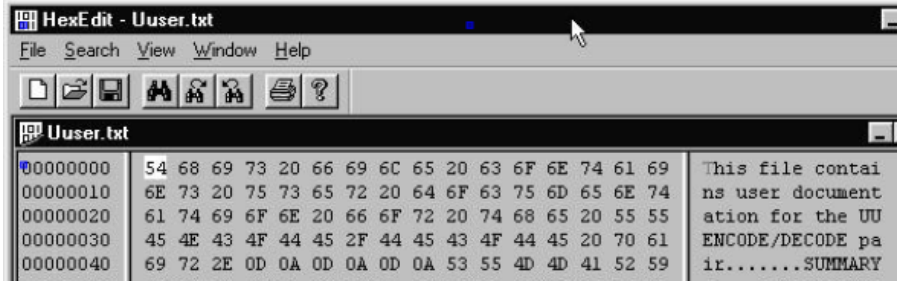
Ex : `A0EF45ED` ⇒ valeur sur 8 symboles x 4 bits = 32 bits

Intérêt de l'hexadécimal

- Le codage hexadécimal permet une représentation plus compacte du binaire.
- La conversion entre binaire et hexadécimal est très simple :
  - binaire ⇒ hexa : on regroupe les bits par groupe de 4, en partant de la droite.
  - hexa ⇒ binaire : on utilise la table précédente.
- Exemple :
  - 1010101111001101 = 1010.1011.1100.1101 = `ABCD`
  - 1010101111101101 = 1010.1011.1110.1101 = `ABED`
- Notation : pour signifier la base hexadécimale, plusieurs notations peuvent être rencontrées.
  `0xABCD` ; `$ABCD` ; `ABCDh` ; ...

Quand rencontre-t-on de l'hexadécimal ?

- Quand on s'interesse (de près) au contenu d'un fichier sur un ordinateur.
- Un fichier est une suite d'octets, dont le sens dépend de ce qui y est stocké (texte, image, son, vidéo, ...)
- Un **éditeur hexadécimal** montre le contenu du fichier sous forme brute (binaire), mais représenté en hexa.



Conversion d'une base 2 ou 16 en base 10

- Le passage d'une base b en base 10 se fait toujours de la même façon, par un développement de la série des puissances.
- Exemple en base 2 :

$$\begin{aligned}(10010001)_2 &= 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2^7 + 2^4 + 2^0 \\ &= 128 + 16 + 1 \\ &= 145\end{aligned}$$

- Exemple en base 16 :

$$\begin{aligned}(C3E)_{16} &= 12 \cdot 16^2 + 3 \cdot 16^1 + 14 \cdot 16^0 \\ &= 12 \cdot 256 + 3 \cdot 16 + 14 \cdot 1 \\ &= 3072 + 48 + 14 \\ &= 3134\end{aligned}$$

Conversion en base 2 ou 16

- Pour convertir un nombre n exprimé en base 10 dans une base b, il faut faire une succession de divisions entières par b, jusqu'à ce que le résultat de la division soit inférieur à b.
- Exemple : soit la valeur 135, à convertir en base 2 :

135	2
$a_0 = 1$	67
	2
$a_1 = 1$	33
	2
$a_2 = 1$	16
	2
$a_3 = 0$	8
	2
$a_4 = 0$	4
	2
$a_5 = 0$	2
	2
$a_6 = 0$	$1 < 2$

- On relève le résultat de la **dernière** division, et l'on ajoute les **restes** des divisions précédentes : $(135)_{10} = (10000111)_2$
- Remarque : pour le binaire, le premier chiffre sera toujours un 1.

Comment coder des décimales ?

- Rappel : $0,1 = 10^{-1}$
- Principe général : identique à celui utilisé pour les entiers, en étendant aux puissances négatives :

$$\begin{aligned}n &= \sum_{i=-q}^{p} a_i b^i \\ &= a_p b^p + \ldots + a_2 b^2 + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots + a_{-q} b^{-q}\end{aligned}$$

avec :

- p : plus grande puissance positive du nombre,
- q : plus grande puissance négative du nombre.

- Exemple en base 10 :
$(3,1415)_{10} = 3 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2} + 1 \cdot 10^{-3} + 5 \cdot 10^{-4}$

Puissance négatives en base 2

- On étend ce principe en base 2 aux puissances de 2 négatives :

10,101

$1 \times 2^{-3} = 1 \times 0,125 = 0,125$
$0 \times 2^{-2} = 0 \times 0,25 = 0$
$1 \times 2^{-1} = 1 \times 0,5 = 0,5$
$0 \times 2^{0} = 0 \times 1 = 0$
$1 \times 2^{1} = 1 \times 2 = 2$

2,625

Réels : conversion de base 10 en base 2

- On traite **séparemment** partie entière et partie fractionnaire
- Pour la partie fractionnaire :
  - On effectue une suite de **multiplications** par 2, jusqu'à obtenir un 1.
  - A chaque étape, on garde la partie **entière**, et on continue avec la partie **fractionnaire** du résultat.
  - Puis on regroupe les bits dans l'ordre d'apparition.
- Exemple : partie fractionnaire : 0,125

Etape			Résultat	Bit	poids
1	0,125	x 2 =	0,25	0	2^{-1}
2	0,25	x 2 =	0,5	0	2^{-2}
3	0,5	x 2 =	1	1	2^{-3}

$\Rightarrow 0,125|_{10} = (0,001)_2$

Remarque

En général, on arrive jamais à 1. On s'arrête à un nombre de bits donné.

Caractère irrationel de la conversion

Théorème

Si un nombre n a un nombre de décimales **fini** dans une base $b1$, il peut avoir un nombre de décimales **infini** lorsqu'il est exprimé dans une autre base $b2$.

- Exemple : soit le nombre $N = (0,2)_{10}$ à convertir en base 2 :

Etape			Résultat	Bit	poids
1	0,2	x 2 =	0,4	0	2^{-1}
2	0,4	x 2 =	0,8	0	2^{-2}
3	0,8	x 2 =	1,6	1	2^{-3}
4	0,6	x 2 =	1,2	1	2^{-4}
5	0,2	x 2 =	0,4	0	2^{-5}
6	0,4	x 2 =	etc.		

La valeur 0,2 est codée en binaire par 0,0011 0011 0011 00...
$\Rightarrow$ **infinité** de décimales.

Corollaire

- Conséquence : une représentation avec un nombre de symboles **fini** dans une base $b1$ n'est qu'une **approximation** du même nombre exprimé dans une autre base $b2$.
- Exemple : soit le nombre $N = (0,2)_{10}$, qu'on représente en binaire sur un nombre de bits n :

n	N (base 2)	N (base 10)	
4	0,0011	$\frac{1}{8} + \frac{1}{16}$	= 0,1875
7	0,0011001	$... + \frac{1}{128}$	= 0,1953125
8	0,00110011	$... + \frac{1}{256}$	= 0,19921875
12	0,001100110011	.	=0,199951172

Représentation en virgule flottante

- Tout nombre N peut être représenté en **virgule flottante**, dans une base de numération quelconque b sous la forme : $N = M \cdot b^E$
  - M : **Mantisse** signée
  - b : base de numération
  - E : **Exposant** (entier relatif)
- Exemples :

N		M	b	E
$3,1415 \cdot 10^0$	$\Rightarrow$	3,1415	10	0
$1011,11 \cdot 2^{-4}$	$\Rightarrow$	1011,11	2	-4
$2345,67 \cdot 8^9$	$\Rightarrow$	2345,67	8	9

- Problème : plusieurs représentations possibles pour un même nombre.
  $3,1415 \cdot 10^0 = 31,415 \cdot 10^{-1} = 0,31415 \cdot 10^1$
- Pour comparer des nombres, on ne peut pas ainsi comparer les mantisses et les exposants.

Normalisation de la représentation

- Afin d'avoir une représentation **unique** d'un nombre, on définit le concept de **normalisation** :

Normalisation
Décalage de la virgule jusqu'à avoir **un seul** chiffre à gauche de la virgule, et ajustement de l'exposant en fonction du nombre de décalages.

- Décalage vers la gauche $\leftrightarrow$ incrémentation de l'exposant.
- Décalage vers la droite $\leftrightarrow$ décrémentation de l'exposant.

- Par exemple (avec $b = 10$) :
  $250.000 = 2,5 \cdot 10^5 \rightarrow M = 2,5,\ E = 5$
  $0,000.004.59 = 4,59 \cdot 10^{-6} \rightarrow M = 4,59,\ E = -6$