# Cours 1 - Système de numération et arithmétique binaire

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 1.0

# Introduction

Pour manipuler, afficher ou transmettre des nombres en utilisant des circuits électroniques, il est nécessaire de représenter chaque symbole par un état différent du circuit.

- Il faut un circuit à dix états pour représenter les symboles de la base dix, de 0 à 9

Problème :

- Il est difficile et excessivement cher de fabriquer des circuits à dix états

Implication :

- Il faut chercher à utiliser un système de numération avec peu de symboles

# Pourquoi on compte jusqu'à 10 ?

- Deux mains, dix doigts

Les symboles de la base 10

- Symboles de la base 10 $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}_{10 \text{ symboles}}$

Exemple :

| | | | | | |
|---|---|---|---|---|---|
| 0 | 10 | . | 90 | 100 | 1000 |
| 1 | 11 | . | 91 | 101 | 1001 |
| 2 | 12 | . | 92 | 102 | 1002 |
| 3 | 13 | . | 93 | ... | ... |
| 4 | 14 | | 94 | 199 | |
| 5 | 15 | | 95 | 200 | |
| 6 | 16 | | 96 | ... | |
| 7 | 17 | | 97 | 999 | |
| 8 | 18 | | 98 | | |
| 9 | 19 | | 99 | | |

# La base décimale

Exemple :

$$(6810)_{10} = (6_3 8_2 1_1 3_0)_{10} = 6 \times 10^3 + 8 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

- La formule permet de convertir un nombre de n'importe quelle base en base dix

# Base Binaire

En informatique le nombre binaire s'écrit : b101010 (le nombre commence par b)

- Symboles de la base binaire = $\underbrace{0, 1}_{2 \text{ symboles}}$

Exemple :

0
1
10
11
100
101
110
111
1000

1001
1010
1011
1100
1101
1110
...

# Octet

- Un octet est formé de 8 bits
- Le bit le plus à droite est appelé bit du poids faible
- Le bit le plus à gauche est appelé bit du poids fort
- L'octet est représenté avec le signe Ø

poids fort $1_7 0_6 1_5 0_4 0_3 1_2 1_1 1_0$ poids faible

# Les unités de capacité

- Le bit (noté petit *b*)
- L'octet = $2^3$ bits = 8 bits. (noté 1 ∅)
- Le Kilo-octet = $2^{10}$ octets = 1024 ∅ (noté 1 *K*∅)
- Le Mega-octet = $2^{20}$ octets = $1024^2$ ∅ (noté 1 *M*∅)
- Le Giga-octet = $2^{30}$ octets = $1024^3$ ∅ (noté 1 *G*∅)
- Le Tera-octet = $2^{40}$ octets = $1024^4$ ∅ (noté 1 *T*∅)

# Conversion de la base binaire en base 10

Exemple :

$$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(1011)_2 = (11)_{10}$$

# Conversion de la base binaire en base 10

Exercice :

Exprimer en base décimale les nombres binaires suivants :

- $(1100)_2 = (?)_{10}$
- $(10101010)_2 = (?)_{10}$
- $(1010101)_2 = (?)_{10}$
- $(1111111)_2 = (?)_{10}$
- $(110011001100)_2 = (?)_{10}$

# Base Octale

En informatique le nombre octal s'écrit : 04532 (le nombre commence par O (zero))

Les symboles de la base Octale ou Base 8

- Symboles de la base Octale (8) $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7}_{8 \text{ symboles}}$

Exemple :

| | |
|---|---|
| 0 | 11 |
| 1 | 12 |
| 2 | 13 |
| 3 | 14 |
| 4 | 15 |
| 5 | 16 |
| 6 | 17 |
| 7 | 20 |
| 10 | ... |

# Conversion de la base Octale en base 10

Exemple :

$$(04752)_8 = 4 \times 8^3 + 7 \times 8^2 + 5 \times 8^1 + 2 \times 8^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(04752)_8 = (?)_{10}$$

# Conversion de la base Octale en base 10

Exercice :

Exprimer en base décimale les nombres octaux suivants :

- $(02457)_8 = (?)_{10}$
- $(0421561)_8 = (?)_{10}$
- $(0101010)_8 = (?)_{10}$
- $(0441243)_8 = (?)_{10}$
- $(010011001100)_8 = (?)_{10}$

# Base Hexadécimale

En informatique le nombre Hexadécimal s'écrit : 0x8A5F3 (le nombre commence par 0x (zero x))

Les symboles de la base Hexadécimale ou Base 16

- Symboles de la base Hexadécimale (16) $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}_{16 \text{ symboles}}$

Exemple :

0
...
9
A
B
C
D

E
F
10
11
...

# Conversion de la base Hexadécimale en base 10

Exemple :

$$(0x1D5B)_{16} = 1 \times 16^3 + 13 \times 16^2 + 5 \times 16^1 + 11 \times 16^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(0x1D5B)_{16} = (?)_{10}$$

# Conversion de la base Hexadécimale en base 10

Exercice :

Exprimer en base décimale les nombres Hexadécimaux suivants :

- $(A4BC)_{16} = (?)_{10}$
- $0x12345 = (?)_{10}$
- $0x1BD4FC3 = (?)_{10}$
- $0xCCCCCC = (?)_{10}$
- $0xDFA34CDFFF = (?)_{10}$

# Conversion entre bases

Nous allons utiliser une méthode simple et rapide pour résoudre les problèmes de conversion.
Supposons que nous voulons convertir en base $b$ un nombre $N$ donnée en base $a$.
La méthode consiste à utiliser une succession de divisions en base $a$ où $N_i(i = 1, 2, \cdots, n)$ représente le quotient de la division, $b$ le diviseur et $R_j(j = 0, 2, \cdots, n)$ le reste de la division.

# Conversion entre bases

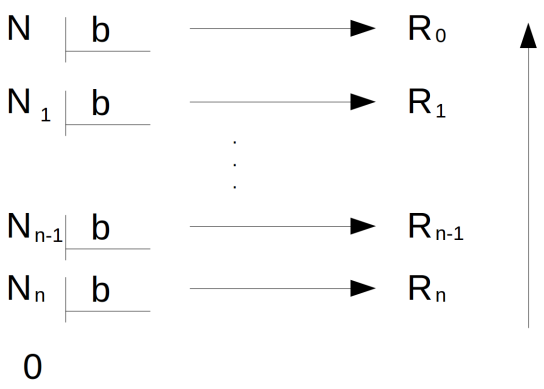



Figure – Conversion

# Conversion de la base Décimale à la base Binaire

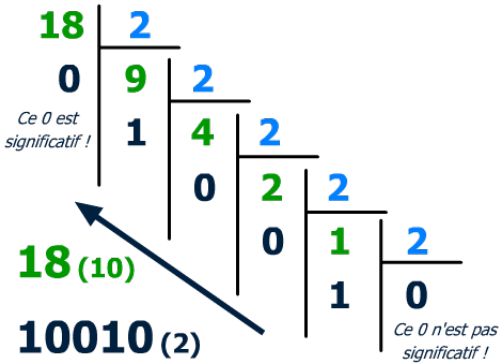



Figure – Conversion décimal - binaire

# Astuce : Conversion entre bases

1er cas : la base origine n'est pas une puissance de la base cible, par exemple convertie de la base hexadécimal vers la base décimale :

- Effectuer des divisions successives

2er cas : la base origine est une puissance de la base origine, par exemple convertie de la base hexadécimal vers la base binaire. Ici on remarque la que la base hexadécimale (16) peut s'écrire sous la forme $2^{\underline{4}} = 16$, dans ce cas :

- Il suffi de convertir chaque nombre de la base origine dans la base cible individuellement sur 4 bits

$$\textit{Exemple} : 0xA5 => (1010\,0101)_2 = (A\,5)_{16}$$

# Les unités de capacité

- Le bit (noté petit *b*)
- L'octet = $2^3$ bits = 8 bits. (noté 1 ∅)
- Le Kilo-octet = $2^{10}$ octets = 1024 ∅ (noté 1 *K*∅)
- Le Mega-octet = $2^{20}$ octets = $1024^2$ ∅ (noté 1 *M*∅)
- Le Giga-octet = $2^{30}$ octets = $1024^3$ ∅ (noté 1 *G*∅)
- Le Tera-octet = $2^{40}$ octets = $1024^4$ ∅ (noté 1 *T*∅)

# Les unités de capacité

Les préfixes métriques dans système international d'unités (SI) k, M, G signifiaient :

- $k = 10^3$
- $M = 10^6$
- $G = 10^9$

Or en informatique $1k = 1024\varnothing \ldots$
A partir de 1998, le SI a clarifié la différence :

- $k = 1000$, $ki = 1024$
- $M = 1000000$, $Mi = 1048576$
- $G = 1000000000$, $Gi = 1073741824$

LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

# Addition binaire

$$\begin{array}{cc} & 0 \\ + & 0 \\ \hline 0 & 0 \end{array}$$

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

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

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

# Addition binaire

Exemple : Additionnez les deux nombres binaire :

$11101101 + 10111001 = ?$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 1 | 1 | 1 | | | 1 | |
| | | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | (237) |
| + | | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | (185) |
| | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | (422) |

# Exercice

Exercice : Additionnez les deux nombres binaires suivants :

- $10101010 + 10101010$
- $11111111 + 10000000$
- $11111111 + 00111111$
- $01111111 + 01111111$

# Soustraction binaire

La soustraction en binaire est faite de la même manière qu'en base décimale. En additionnant un nombre positif à un nombre négatif. Le nombre dont la valeur absolue est plus petite est soustrait du nombre dont la valeur absolue est plus grande et le signe de plus grand nombre est affecté au résultat.

# Soustraction binaire

Exemple : Soustraire les deux nombres binaires :

$$11001010 - 10101001 = ?$$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | $_1$0 | 0 | 1 | 0 | 1 | $_1$0 | (202) |
| - | 1 | $_1$0 | 1 | 0 | 1 | 0 | $_1$0 | 1 | (169) |
| | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | (33) |

**Attention** : Cette méthode est à titre explicatif, ce n'est pas la méthode utilisée par la machine.

# Avantages et problèmes des nombres négatifs

Pour implémenter les nombres négatifs, il faut représenter le signe (-), or dans un ordinateur, nous pouvons uniquement écrire des 0 ou des 1. Donc la solution consiste à désigner le bit le plus significatif comme étant le bit du signe : 1 pour (-) et 0 pour (+).

# Avantages et problèmes des nombres négatifs

**Avantage :**

- Si on peut représenter les nombres négatifs alors on peut transformer les soustractions en addition, par exemple $10 - 30$ peut ainsi s'écrire $10 + (-30)$
- Un ordinateur ne distingue pas entre une addition et une soustraction. Ainsi, le circuit utilisé pour l'opération d'addition peut être utilisé pour l'opération de soustraction grâce au complément à 2.

# Avantages et problèmes des nombres négatifs

**Problème :**

- Si l'ordinateur propose de mettre un bit pour distinguer les nombres positifs et négatifs alors le nombre Zéro 0 aura deux représentations $-0$ et $+0$
- Les nombres doivent avoir le même nombre de bits
- Il y a un risque de débordement de capacité

# Solution : complément à 2

Complément à 2 d'un nombre positif est lui même : Donc rien à faire

---

Complément à 2 d'un nombre négatif est :

**Complément à 2**

$$Complment\ \ 2\,(N) = Complment\ \ 1(N) + 1$$

**Complément à 1**

$$Complment\ \ 1(N) = inverse(N)$$

Complément : Inverser les bits : 1 => 0 et 0 => 1

# Complément à 2

Exemple :

Trouvez le complément à 2 de : $N = 10101001$

C1 :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\sim$ | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

C2 :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| + | | | | | | | | 1 |
| | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

# Complément à 2

Exemple de soustraction : $10 - 12$ sur 5 bits

-Donc on peut écrire : $10 + (-12)$

-Pour effectuer ce calcul il suffi d'additionner 10 avec le C2 de 12

10 en binaire sur 5 bits = 01010

$-12$ en binaire sur 5 bits "$C2(12) = C1(12) + 1$" :

12 s'écrit 01100

$C1(12)$ s'écrit 10011

$C2(12)$ s'écrit 10100

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

# Complément à 2

Donc :

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

Ici on remarque le résultat est négatif car le dernier bit est égale à 1

Pour trouver le résultat sans le signe, il faut appliquer au résultat un complément à 2

Donc :

$C2(11110) = 00001 + 1 = 00010 = (2)_{10}$

Résultat en décimal : $10 - 12 = -2$

# Complément à 2

- Le calcul en complément à 2 se fait sur $n$ bits, si une retenue est générée à la fin de l'addition (par exemple le résultat de l'addition est sur $n + 1$ bits) alors la retenue est tout simplement ignorée

# Débordement

Les opérations signées utilisent le dernier bit pour représenter le signe du nombre, donc pour $n$ bits, il est possible de représenter les valeurs suivantes :

- Nombres signés sur $n$ bits : $[-2^{n-1}, 2^{n-1} - 1]$

Les opérations signées peut engendre des débordement de capacité (noté V pour *overflow*), il est possible de détecter le débordement en appliquant la formule suivante :

$$Pour\ l'operation\ :\ s = a + b$$

$$V = \bar{a}_{n-1} * \bar{b}_{n-1} * s_{n-1} + a_{n-1} * b_{n-1} * \bar{s}_{n-1}$$

V : est une expression booléen, si V=1 alors i y a débordement

# Exercice complément à 2

Exercice : Effectuez les opérations suivantes en binaire (sur 8 bits) :

- $125 - 117$
- $0xA5 - 0xA4$
- $b10100101 + b0001111$
- $054 - 015$

# Nombres à virgule fixe

Un nombre décimal est composé d'une partie entière et d'une partie fractionnaire après la virgule

$$(N)_B = a_{n-1}a_{n-2}...a_0, b_1b_2...b_m$$

- Exemple :

$$128,75 = 1x10^2 + 2x10^1 + 8x10^0 + 7x10^{-1} + 5x10^{-2}$$
$$(101,01)_2 = 1x2^2 + 1x2^0 + 1x2^{-2}$$
$$(AE,1F)_{16} = 10x16^1 + 14x16^0 + 1x16^{-1} + 15x16^{-2}$$

# Conversion des nombres à virgule en base B

Exemple : Convertir 28, 8625

- $28 = (11100)_2$ , $0.8625 = (?)_2$

| | | |
|---|---|---|
| 0,8625 x 2 | 1,725 | 1 + 0,725 |
| 0,725 x 2 | 1,45 | 1 + 0,45 |
| 0,45 x 2 | 0,9 | 0 + 0,9 |
| 0,9 x 2 | 1,8 | 1 + 0,8 |
| 0,8 x 2 | 1,6 | 1 + 0,6 |
| 0,6 x 2 | 1,2 | 1 + 0,2 |
| 0,2 x 2 | 0,4 | 0 + 0,4 |
| 0,4 x 2 | 0,8 | 0 + 0,8 |

28, 8625 peut être représenté par $(11100, 1101\underline{1100}\ldots)_2$

# Notation exponentielle

## Notation exponentielle

$$N = \pm MB^{\pm e}$$

Exemple :

- $510^{8}$
- $1.984510^{-4}$
- $-8510^{-14}$
- $9,238410^{4}$

# La norme IEEE 754

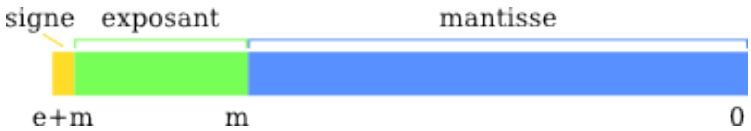
- Un format standardisé
- Format simple précision codé en 32 bits
  - Bit du signe (1 bit)
  - Exposant (8 bits)
  - Mantisse (23 bits)
- Format double précision en 64 bits
  - Bit du signe (1 bit)
  - Exposant (11 bits)
  - Mantisse (52 bits)

# Notation exponentielle IEEE 754 (Format général)

Un nombre flottant est formé de trois éléments :

| Signe | Exposant décalé | Mantisse |
|---|---|---|
| 1 bit | $e$ bits | $m$ bits |

- Le signe (Le bit de poids fort, si ce bit est à 1, le nombre est négatif, et s'il est à 0, le nombre est positif.)
- L'exposant noté $e$ (les $e$ bits suivants représentent l'exposant décalé)
- La mantisse notée $m$ (les $m$ bits de poids faible, représentent la mantisse)



# Décalage de l'exposant

- L'exposant peut être positif ou négatif.
- La notation en complément à 2 rendrai la comparaison entre les nombres difficile.
- Pour régler ce problème, l'exposant est décalé, afin de le stocker sous forme d'un nombre non signé.
- Le décalage est de $2^{e1}1$ ($e$ représente le nombre de bits de l'exposant)

$$valeur = signemantisse2^{(exposantdcalage)}$$

# La mantisse

- La mantisse doit s'écrire sous sous la forme normalisée c-a-d :

$$1, xxx\ldots$$

- Uniquement les nombres après la virgules sont écrits la partie *mantisse* du nombre, le 1 est implicite. Exemple :

$$0,0010110x2^5$$

Sous la forme normale, doit s'écrir :

$$1,0110x2^2$$

Seule 0110... est reporté dans la partie Mantisse, le 1 est implécite

# Exemple

On a vu que $+28,8625$ s'écrit en binaire $(11100, 11011100\ldots)_2$

- Écrire sous la forme normale :

$$11100, 11011100x2^0$$
$$1, 11001101100x2^4$$

- Bit du signe (+) : 0
- Exposant :

$$2^4 => \text{exposant} = 4$$
$$\text{donc en excédent } 127 => 127 + 4 = 131$$
$$\text{en binaire} = 10000011$$

Résultat : 0 10000011 11001101110011001100110

# Remarques à propos IEEE 754

- Étant donnée que la forme normalisée impose d'écrire 1, ... donc il n'est pas possible de représenter les nombres entres 0 et $2^{-127}$. Pour remédier à cela il existe la forme démoralisée pour représenter les nombres entre 0 et $2^{-127}$
- Étant donnée que l'exposant représente une puissance de 2 et la mantisse est constituée de 23 bits, quelque soit l'exposant. Par exemple entre $2^0$ et $2^1$ (entre 1 et 2) il y a $2^{23}$ possibilité et $2^1$ et $2^2$ (entre 2 et 4) il y a toujours $2^{23} \simeq 8 \times 10^6$. Ce qui a pour conséquence de perdre en précision si l'exposant augmente.

# Remarques à propos IEEE 754 (2)

Par exemple on remarque la perte de l'associative de l'addition pour :

$2^{24} + 1 + 1 \neq 1 + 1 + 2^{24}$ ou $2^{24} + 2$ puisque à ce stade ($2^{24}$) il n'est plus possible de représenter les nombres impaire.

- L'erreur relative $\epsilon$ avec 23 bits (simple précision)

$$\epsilon = 2^{-23} = (\frac{1}{2})^{23} = \frac{1}{2^{20} \times 2^3} \simeq \frac{10^-6}{8}$$

6 chiffres significatifs après la virgule

# IEEE 754 les exceptions

| Type | Exposant décalé | Mantisse |
|---|---|---|
| Zéros | 0 | 0 |
| Nombres dénormalisés | 0 | différente de 0 |
| Nombres normalisés | $12^e - 2$ | quelconque |
| Infinis | $2^e 1$ | 0 |
| NaNs | $2^e 1$ | différente de 0 |

# Format simple précision (32 bits)

- 32 bits : 1 bit de signe, 8 bits pour l'exposant et 23 pour la mantisse.
- L'exposant est donc décalé de $2^{8-1} - 1 = 127$
- L'exposant d'un nombre normalisé va donc de -126 à +127. L'exposant -127 (qui est décalé vers la valeur 0) est réservé pour zéro et les nombres dénormalisés, tandis que l'exposant 128 (décalé vers 255) est réservé pour coder les infinis et les NaN.

# Format double précision (64 bits)

- 64 bits : 1 bit de signe, 11 bits pour l'exposant et 52 pour la mantisse.
- L'exposant est donc décalé de $2^{11-1} - 1 = 1023$
- Pour les nombres normalisés, le décalage de l'exposant est +1023. Pour les nombres dénormalisés, l'exposant est 1022 (l'exposant minimum pour un nombre normalisé). Ce n'est pas 1023 car les nombres normalisés ont un 1 avant la virgule, et les nombres dénormalisés n'en ont pas. Comme précédemment, zéro et l'infini sont signés.

# Codage des caractères

Les différents type codages :

- Code BCD (Binary Coded Decimal)
- Code 7 segments
- Code ASCII (American Standard Code for Information Interchange)
- UTF

# Code ASCII

| | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0: | | ( | 2 | < | F | P | Z | d | n | x |
| 1: | | ) | 3 | = | G | Q | [ | e | o | y |
| 2: | | * | 4 | > | H | R | \ | f | p | z |
| 3: | ! | + | 5 | ? | I | S | ] | g | q | { |
| 4: | " | , | 6 | @ | J | T | ^ | h | r | \| |
| 5: | # | - | 7 | A | K | U | _ | i | s | } |
| 6: | $ | . | 8 | B | L | V | ` | j | t | ~ |
| 7: | % | / | 9 | C | M | W | a | k | u | DEL |
| 8: | & | 0 | : | D | N | X | b | l | v | |
| 9: | ' | 1 | ; | E | O | Y | c | m | w | |