Architecture des Ordinateurs

Prof. Yérali Gandica

CY-Tech Cergy Paris Université
2022

The Course

- Lecture 1: Représentation de données, Codage, Algèbre de Boole.
- Lecture 2: Circuits Logiques (combinatoires et séquentiels) $+$ Mémoires.
- Lecture3: Processeur.
- Lecture 4: Assembleur.

Objectif du cours

Comprendre l'architecture matérielle des ordinateurs, les différents composants, le cheminement des données, l'adaptation aux différentes applications.

L'ordinateur

Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

Pour fonctionner correctement il a besoın de:

- **HARDWARE, Matériel** Entrées, sorties, processeur et mémoires.
- **SOFTWARE, Logiciel** Instructions qui indiquent au ≪ Hardware ≫ ce qu'il doit faire.
- **USER, Utilisateur** Personne qui se sert de l'ordinateur pour faire son travail ou pour s'amuser.

L'ordinateur

Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

Pour fonctionner correctement il a besoın de:

- **HARDWARE, Matériel** Entrées, sorties, processeur et mémoires.
- **SOFTWARE, Logiciel** Instructions qui indiquent au ≪ Hardware ≫ ce qu'il doit faire.
- **USER, Utilisateur** Personne qui se sert de l'ordinateur pour faire son travail ou pour s'amuser.

L'ordinateur

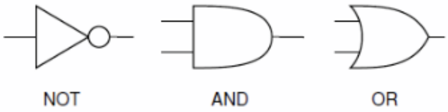
Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

Pour fonctionner correctement il a besoın de:

- **HARDWARE, Matériel** Entrées, sorties, processeur et mémoires.
- **SOFTWARE, Logiciel** Instructions qui indiquent au $\ll$ Hardware $\gg$ ce qu'il doit faire.
- **USER, Utilisateur** Personne qui se sert de l'ordinateur pour faire son travail ou pour s'amuser.

Composants de base

NOT – AND – OR



A	B	NOT B	A AND B	A OR B
0	0	1	0	0
0	1	0	0	1
1	0	1	0	1
1	1	0	1	1

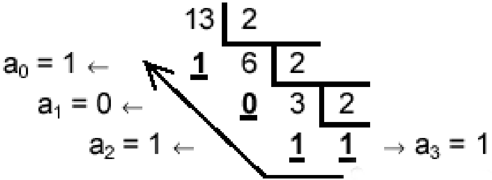
Numeration en base b

Étant donné un entier positif b, chaque nombre entier x peut être représenté de manière unique par un nombre $a_n a_{n1} a_0$, tel que $a_n \neq 0$ et pour tout $i \in [0, n], a_i \in [0, b-1]$ et

$$x = a_n b^n + ... + a_0 b^0. \tag{1}$$

Conversion de la base 10 à la base b

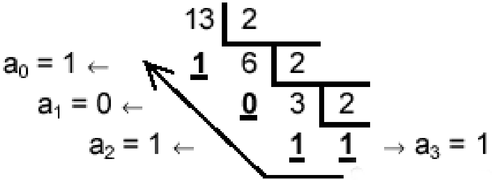
Pour convertir la partie entière du nombre de base 10, on le divise successivement, par b, jusqu'à obtenir un quotient plus petit que b. La partie entière du nombre recherché sera composée du dernier quotient et des restes obtenus, pris dans l'ordre inverse.



- Donc, $13_{10} = 1101_2$
- Pour vérifier si les calculs sont faits correctement, on peut effectuer la conversion inverse:
- $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = ?$
- Maintenant vous: 145 en base 8.

Conversion de la base 10 à la base b

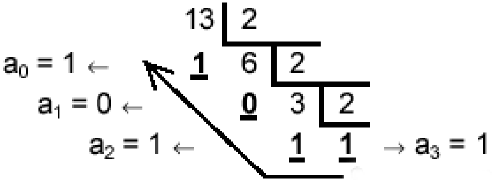
Pour convertir la partie entière du nombre de base 10, on le divise successivement, par b, jusqu'à obtenir un quotient plus petit que b. La partie entière du nombre recherché sera composée du dernier quotient et des restes obtenus, pris dans l'ordre inverse.



- Donc, $13_{10} = 1101_2$
- Pour vérifier si les calculs sont faits correctement, on peut effectuer la conversion inverse:
- $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = ?$
- Maintenant vous: 145 en base 8.

Conversion de la base 10 à la base b

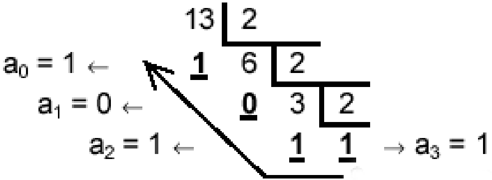
Pour convertir la partie entière du nombre de base 10, on le divise successivement, par b, jusqu'à obtenir un quotient plus petit que b. La partie entière du nombre recherché sera composée du dernier quotient et des restes obtenus, pris dans l'ordre inverse.



- Donc, $13_{10} = 1101_2$
- Pour vérifier si les calculs sont faits correctement, on peut effectuer la conversion inverse:
- $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = ?$
- Maintenant vous: 145 en base 8.

Conversion de la base 10 à la base b

Pour convertir la partie entière du nombre de base 10, on le divise successivement, par b, jusqu'à obtenir un quotient plus petit que b. La partie entière du nombre recherché sera composée du dernier quotient et des restes obtenus, pris dans l'ordre inverse.



- Donc, $13_{10} = 1101_2$
- Pour vérifier si les calculs sont faits correctement, on peut effectuer la conversion inverse:
- $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = ?$
- Maintenant vous: 145 en base 8.

L'arithmétique binaire

L'arithmétique binaire ressemble à l'arithmétique décimale. Voici la table d'addition des nombres binaires: Somme

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Example:

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

CODAGE – Représentation des nombres en machine

Codes BCD, Signe & Valeur Absolue (SVA) et Complément à 2 (Cà2)

BCD : Binary Code Decimal

n bits → 2^n combinaisons
4 bits → 16 valeurs

N	BCD	SVA	Cà2
+ 9	1001		
+ 8	1000		
+ 7	0111	0111	0111
+ 6	0110	0110	0110
+ 5	0101	0101	0101
+ 4	0100	0100	0100
+ 3	0011	0011	0011
+ 2	0010	0010	0010
+ 1	0001	0001	0001
+ 0	0000	0000	0000
- 0		1000	
-1		1001	1111
-2		1010	1110
-3		1011	1101
-4		1100	1100
-5		1101	1011
-6		1110	1010
-7		1111	1001
-8			1000

Représentation complément à deux

Les nombres positifs sont donc représentés en binaire classique mais sont seulement codables les entiers allant de 0 à $2^{n-1} - 1$. Etant donné un entier positif x, on obtient x de la facon suivante : on remplace les 1 par des 0 et les 0 par des 1, puis on ajoute 1 au résultat. Si une retenue est crée sur le dernier bit, elle est effacée.
On passe de la même façon d'un nombre négatif à positif: on inverse tous les bits et on ajoute 1 au résultat.

Exemple :

Conversion en Complément à 2 sur 8 bits de (données en décimal)

-7

$$
\begin{array}{rl}
 & \mathbf{0000\ 0111} \ \ (+7) \\
 & \mathbf{1111\ 1000} \ \ (\text{Complément de } 7) \\
+ & \mathbf{1} \\
\hline
 & \mathbf{1111\ 1001} \ \ (-7)
\end{array}
$$

Exemple :

Calcul en Complément à 2 sur 8 bits de (données en décimal)

122 + (-7)

```
 1 1111  000    (retenues)
   0111  1010   (122)
 + 1111  1001   (-7)
 ---------------
 1 0111  0011   (115) => 0111 0011 représente bien 115(10)
```

La retenue est perdue car le calcul se fait sur 8 bits (➔ capacité de la machine)

$$\mathbf{0111\,0011} = \mathbf{0}\cdot 2^7 + \mathbf{1}\cdot 2^6 + \mathbf{1}\cdot 2^5 + \mathbf{1}\cdot 2^4 + \mathbf{0}\cdot 2^3 + \mathbf{0}\cdot 2^2 + \mathbf{1}\cdot 2^1 + \mathbf{1}\cdot 2^0 = \mathbf{115}$$

Débordement de capacité

Le problème de coder sur un nombre fixé de bits est que l'on peut déborder lors de calculs.
Par exemple, si on effectue l'opération $(01000000 + 01000000)_{2c8}$ on obtient $(10000000)_{2c8}$ c'est à dire un nombre négatif alors qu'on a additionné deux nombres positifs ! Le résultat est donc faux, on dit qu'il y a débordement (overflow).
Pour le codage en complément à deux, on peut facilement détecter un débordement : il engendre forcément un erreur de signe. Il suffit donc d'observer les règles suivantes:

- Si on additionne deux nombres de signes contraires, il ne peut pas y avoir de débordement.
- Si on additionne deux nombres positifs, il y a débordement si et seulement si le résultat est négatif, i.e., si le bit de gauche est à 1.
- Si on additionne deux nombres negatifs, il y a débordement si et seulement si le résultat est positif, i.e., si le bit de gauche est à 0.

Nombres à virgule fixe

Conversion décimal binaire : Partie entière

Exemple : convertir 125

On divise le nombre 125 par 2 autant de fois qu'il est possible.

Les restes successifs étant les poids binaires (dans l'ordre des puissances croissantes)

125	:	2	=	62	reste	1
62	:	2	=	31	reste	0
31	:	2	=	15	reste	1
15	:	2	=	7	reste	1
7	:	2	=	3	reste	1
3	:	2	=	1	reste	1
1	:	2	=	0	reste	1

$\mathbf{125_{(10)} = 0111\ 1101_{(2)}}$ sur 8 bits

Nombres à virgule fixe

Conversion Partie fractionnaire

- Décimal vers Binaire

Exemple : conversion de $125.375_{(10)}$

On multiplie la partie fractionnaire par 2, la partie entière obtenue est le poids binaire, la nouvelle partie fractionnaire étant de nouveau multipliée par 2, etc.

0.375	x	2	=	**0**.75
0.75	x	2	=	**1**.50
0.50	x	2	=	**1**.00

$\mathbf{0.375_{(10)}=0*2^{-1}+1*2^{-2}+1*2^{-3}=0.\overrightarrow{011}_{(2)}}$

Et comme $\mathbf{125_{(10)}=0111\ 1101_{(2)}}$ on a :

$\mathbf{125.375_{(10)}=0111\ 1101.0110\ 0000_{(2)}}$ sur 8 bits

Nombres à virgule fixe

Conversion Partie fractionnaire

- Binaire vers Décimal

Exemple : conversion de $0.011_{(2)}$

La partie fractionnaire représente les coefficients des puissances négatives de 2 :

$2^{-1}=1/2=0.5$
$2^{-2}=1/4=0.25$
$2^{-3}=1/8=0.125$
…
$2^{-p}=1/2^{p}$

$$\mathbf{0.011_{(2)} = (0 \times 2^{-1})+(1 \times 2^{-2})+(1 \times 2^{-3}) = 0.25+0.125}$$
$$\mathbf{= 0.375_{(10)}}$$

Représentation en virgule flottante

Elle correspond en fait à la notation dite “scientifique” des grands nombres comme 3×10^{27} ou encore 8×10^{-18}.

Pour des raisons évidentes d'espace mémoire, il n'est possible de représenter qu'un nombre borné de rééls, on parle alors plutôt de *flottants*

Depuis les années 70, il existe un standard pour la représentation des flottants. Aujourd'hui la plupart des ordinateurs utilisent ce standard. C'est la **représentation IEEE 754**.

Un nombre flottant est codé par 3 nombres représentés de la façon suivante :

Le coefficient f est appelé la *mantise*, e est appelé l'*exposant* et s représente le signe : positif si $s = 0$ et négatif si $s = 1$.

Le standard inclu deux représentations : simple précision et double précision.

Nombre de bits	taille de s	taille de f	taille de e	E_{min}	E_{max}
32 (simple précision)	1	23	8	-126	+127
64 (double précision)	1	52	11	-1022	+1023

où E_{min} et E_{max} représentent respectivement le plus petit et le plus grand exposant codable dans la représentation.

Codage

Considérons le codage sur 32 bits. On commence par écrire la valeur absolue du réel r à coder en binaire à virgule fixe. On décale ensuite la virgule en multipliant par des puissances de 2, jusqu'à avoir un et un seul chiffre avant la virgule.

Prenons par exemple $r = -123,5$. On le code par $-1111011,1$ puis on décale la virgule et on obtient $-(1,1110111) \times 2^6$. On en déduit

- le bit de signe $s = 1$
- la mantisse $M = 1110111$ qu'on complète pour obtenir un mot f sur 23 bits : $f = 111\,0111\,0000\,0000\,0000\,0000$
- l'exposant $E = 6$ que l'on code en excès à 127 : le nombre e codé sur 8 bits est donc $e = E + 127 = 133 = (1000\,0101)_2$

Le codage de r est donc

1	1000 0101	111 0111 0000 0000 0000 0000

Codage

Nombres à virgule flottante

Norme IEEE754

Décomposition	**Signe**	**Exposant (entier)**	**Mantisse**
Simple précision (32 bits)	**1**	**8**	**23**
Double précision (64 bits)	1	11	52

Nombre normalisé :

Nombre = $(-1)^{\text{signe}}$ x 1,mantisse x $2^{(\text{exposant} - \text{biais})}$

biais = 127 (simple précision) et 1023 (double précision)

Exemple 1 (32 bits)

$18 = 2^k \times (1,\ldots) = 16 \times 1{,}125 = 2^4 \times (1 + 0{,}125) = 2^{131-127} \times (1 + 0{,}125)$ avec $k \in \mathbf{Z}$

s (1) = **0**

e (8) = **131 = 1000 0011**

m (23) = **0.125 = 001 0000 0000 0000 0000 0000**

sem = **0100 0001 1001 0000 0000 0000 0000 0000 = 41 90 00 00$_{(H)}$**

Système Hexadécimal

Système hexadécimal - Système décimal - Système binaire

0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B.	11	1011
C.	12	1100
D.	13	1101
E.	14	1110
F.	15	1111

Nombres à virgule flottante

Norme IEEE754

Décomposition	**Signe**	**Exposant (entier)**	**Mantisse**
Simple précision (32 bits)	**1**	**8**	**23**
Double précision (64 bits)	1	11	52

Nombre normalisé :

Nombre = $(-1)^{\text{signe}} \times 1{,}\text{mantisse} \times 2^{(\text{exposant} - \text{biais})}$

biais = 127 (simple précision) et 1023 (double précision)

Exemple 2 (32 bits)

$1/10 = 2^k \times (1,\ldots) = 0.0625 \times 1{,}6 = 2^{-4} \times (1 + 0{,}6) = 2^{123-127} \times (1 + 0{,}6)$ avec $k \in \mathbf{Z}$

s (1) = **0**

e (8) = **123 = 0111 1011**

m (23) = **0.6 = 100 1100 1100 1100 1100 1100** (partie fractionnaire infinie)

sem = **0011 1101 1100 1100 1100 1100 1100 1100 = 3D CC CC CC$_{(H)}$**

Codage

Nombres à virgule flottante

Addition en IEEE754

- Restituer le « 1 » entier des mantisses
- Ramener les deux nombres au même exposant
- Effectuer l'addition/soustraction des mantisses comme pour les entiers
- Renormaliser le résultat (arrondi, bit de poids fort, exposant,… éventuellement)

Exemple

1/10 + 1/10 =

1,**100 1100 1100 1100 1100 1100** * $2^{123-127}$	*opérandes identiques même exp*
+ 1,**100 1100 1100 1100 1100 1100** * $2^{123-127}$	
--	
11,001 1001 1001 1001 1001 1000 * $2^{123-127}$	*à normaliser*
1,**100 1100 1100 1100 1100 1100 * $2^{124-127}$**	*dernier bit (0) perdu*

Résultat 1/10 + 1/10

s (1) = **0**

e (8) = ***124*** **= 0111 1100**

m (23) **= 100 1100 1100 1100 1100 1100** (partie fractionnaire infinie)

sem = **0011 1110 0100 1100 1100 1100 1100 1100 = 3E 4C CC CC$_{(H)}$**

Codage

Nombres à virgule flottante

Multiplication en IEEE754

- Calculer le signe puis la somme des exposants
- Restituer le « 1 » entier des mantisses
- Effectuer la multiplication des valeurs absolues comme pour les entiers
- Eventuellement, arrondir, ajuster l'exposant et renormaliser

Exemple **-18 x 10 =**

Conversions

X = -18 = 2^k x (1,…) = -16 x 1,125 = - 2^4 x (1 + 0,125) = $-2^{131-127}$ x (1 + 0,125)
s (1) = **1**
e (8) = *4 % 127* = ***131*** **= 1000 0011**
m (23) **=** ***0,125*** **= 001 0000 0000 0000 0000 0000** = m_X
sem = **1100 0001 1001 0000 0000 0000 0000 0000 = C1 90 00 00**$_{(H)}$

Y = 10 = 2^k x (1,…) = 8 x 1,25 = 2^3 x (1 + 0,25) = $2^{130-127}$ x (1 + 0,25)
s (1) = **0**
e (8) = *3 % 127* = ***130*** **= 1000 0010**
m (23) **=** ***0,25*** **=** 2^{-2} **= 010 0000 0000 0000 0000 0000** = m_Y
sem = **0100 0001 0010 0000 0000 0000 0000 0000 = 41 20 00 00**$_{(H)}$ 17

Codage

Nombres à virgule flottante

Multiplication en IEEE754

Z = X x Y = (-18) x 10 =

signe : 1 (XOR entre les signes des 2 opérandes)

exposant : $1000\ 0011 + 1000\ 0010 - (127)_2$

$= 1000\ 0011 + 1000\ 0010 - 0111\ 1111$

$= 1000\ 0011 + 1000\ 0010 + (-127)_{C2}$

$= 1000\ 0011 + 1000\ 0010 + 1000\ 0001 =$ **1000 0110**

$= 134_{10} = 4 + 127$ (OK : 7 = 4 + 3)

mantisse : multiplication des mantisses m_X et m_Y :

remarque : $(1 + m_X)(1 + m_y) = 1 + m_X m_Y + m_X + m_Y$

$$\mathbf{m_Z = m_X m_Y + m_X + m_Y}$$

	$1,m_X$	=		1,001 0…0
x	$1,m_Y$	=	x	1,01 0…0
				1001
				0000
				1001
	$1,m_Z$	=		1,01101 0…0

m_Z = 01101 0…0

m_Z (23) = 011 0100 0000 0000 0000 0000

18

Nombres à virgule flottante

Multiplication en IEEE754

m_Z = 01101 0…0 → m_Z (23) = 011 0100 0000 0000 0000 0000

Interprétation du résultat : Z

s (1) = **1**
e (8) = **1000 0110** **=>** e = 134 **=> exposant = 134 – 127 = 7 % 127**
m (23) **= 011 0100 0000 0000 0000 0000 => mantisse = $2^{-2} + 2^{-3} + 2^{-5}$ = 0,40 625**

sem = **1100 0011 0011 0100 0000 0000 0000 0000 = C3 34 00 00$_{(H)}$**

Z = - 1,40 625 x 2^7 = - 1,40 625 x 128 = -180

Algèbre de Boole

Le fonctionnement des circuits est décrit en utilisant l'algèbre binaire (algèbre de Boole à deux valeur). Nous en donnons les bases dans cette section.

L'algèbre de Boole est une structure algébrique

- donnée par un ensemble contenant au moins deux valeurs $\{0, 1\}$, et les trois opérations suivantes
  - la conjonction (ou produit) : opération binaire qui peut être notée ".", "et" ou bien "and".
  - la disjonction (ou somme) : opération binaire qui peut être notée "+", "ou" ou bien "or".
  - la négation (ou complément) : opération unaire qui peut être notée "non" ou "not" ou bien par une barre sur l'opérande.

Algèbre de Boole

Nous allons nous intéresser à l'algèbre de Boole binaire, c'est à dire que l'ensemble des éléments de l'algèbre est $\{0, 1\}$ (ou bien $\{Vrai, Faux\}$).

Le définition suivante des opérateurs satisfait l'ensemble des axiomes. C'est celle que nous utiliserons.

— le complément

a	$\bar{a}$
0	1
1	0

Algèbre de Boole

— la conjonction

a	b	$a.b$
0	0	0
0	1	0
1	0	0
1	1	1

— la disjonction

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1

Algèbre de Boole:
Portes Logiques de base

	Symbole	Opération	Table de vérité

ET (AND)

$A \cdot B$

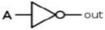
Entrées		Sortie
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OU (OR)

$A + B$

Entrées		Sortie
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

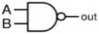
NON (NOT)



$\overline{A}$

Entrée	Sortie
A	NOT A
0	1
1	0

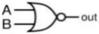
NON-ET (NAND)



$\overline{A \cdot B}$

Entrées		Sortie
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

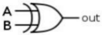
NON-OU (NOR)



$\overline{A + B}$

Entrées		Sortie
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

OU exclusif (XOR)



$A \oplus B$

$= A \cdot \overline{B} + \overline{A} \cdot B$

Entrées		Sortie
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Algèbre de Boole: Relations et propriétés

propriétés de la somme

$0 + 0 = 0$	$a + 1 = 1$
$0 + 1 = 1$	$a + 0 = a$
$1 + 0 = 1$	$a + a = a$
$1 + 1 = 1$	$a + \bar{a} = 1$

propriétés du produit

$0 . 0 = 0$	$a . 1 = a$
$0 . 1 = 0$	$a . 0 = 0$
$1 . 0 = 0$	$a . a = a$
$1 . 1 = 1$	$a . \bar{a} = 0$

négation

$\bar{0} = 1$

$\bar{1} = 0$

$\bar{\bar{a}} = a$

commutativité

$a . b = b . a$

$a + b = b + a$

associativité

$a . (b . c) = (a . b) . c$

$a + (b + c) = (a + b) + c$

distributivité

$a . (b + c) = a . b + a . c$

$(a + b) . (c + d) = ac + ad + bc + bd$

propriétés combinées

$a . (a + b) = a$	$(a + b) . (a + \bar{b}) = a$
$a + a . b = a$	$(a + b) . (a + c) = a + b . c$
$a + \bar{a} . b = a + b$	$(a + b) . (\bar{a} + c) = a . c + \bar{a} . b$

théorème de Morgan

$\overline{a + b + c} = \bar{a} . \bar{b} . \bar{c}$

$\overline{a . b . c} = \bar{a} + \bar{b} + \bar{c}$