

# CODAGE ET REPRÉSENTATION DE L'INFORMATION

version 2015-2016

## Plan du cours

### Chapitre 0 : INTRODUCTION

- 0.1 Informatique et codage binaire
- 0.2 Discrétisation
- 0.3 Calcul à base de bits

### Chapitre 1 : CODAGE DES ENTIERS NON SIGNES

- 1.1 Systèmes de numérations
- 1.2 Arithmétique

### Chapitre 2 : CODAGE DES ENTIERS SIGNES

- 2.1 Représentation en SVA (signe et valeurs absolues)
- 2.2 Représentation en CPL1 (Complément à 1)
- 2.3 Représentation en CPL2 (Complément à 2)

### Chapitre 3 : REPRÉSENTATION DES NOMBRES RÉELS

- 2.1 Représentation en virgules fixe
- 2.2 Représentation en virgules flottantes
- 2.3 Introduction à l'arithmétique flottantes

### Chapitre 4 : REPRÉSENTATION DES CARACTÈRES ALPHANUMÉRIQUES

- 4.1 Code ASCII
- 4.2 Unicode

### Chapitre 5 : INTRODUCTION À L'ALGÈBRE DE BOOLE

- 5.1 Notions de base
- 5.2 Formes canoniques

### Bibliographies

- Introduction à l'informatique : F. Denis, V. Poupet (<http://pageperso.lif.univ-mrs.fr/~francois.denis/initInfo/>)
- Structure et Fonctionnement des Ordinateurs : Michel Castan (<http://intranet-gei.insa-toulouse.fr:8181/Enseignements/SFO/Polykopie/index.html>)
- internet (wikipedia, wikibooks, ... etc.)
- mon blog : [www.algocri.blogspot.com](http://www.algocri.blogspot.com)

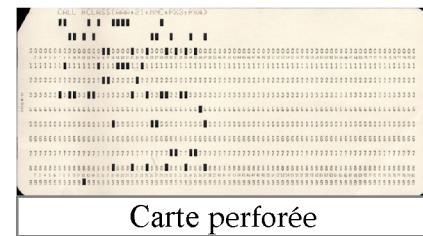
## Chapitre 0

# INTRODUCTION

## 1. INFORMATIQUE ET SYSTÈME BINAIRE

Les ordinateurs sont capables de stocker et de manipuler des nombres, du texte voire du son, des images et de la vidéo. Ceci est possible grâce au codage de l'information. En informatique, quelles que soient le type de ces données ou informations, leurs codes sont toujours transcrits en **binnaire**. Le système binaire utilise deux états élémentaires notés 0 et 1 ou faux et vrai pour coder l'information. La raison est que la bivalence est techniquement plus facile à mettre en œuvre dans un ordinateur pour le stockage, le traitement et la transmissions de données. En effet, pour représenter ces deux états élémentaires on peut utiliser :

- la présence ou l'absence de trou dans une position donnée dans une carte perforée ([média de stockage aujourd'hui obsolète](#));
- Les irrégularités (cavités) dans la surface réfléchissante d'un CD.
- La direction du champs magnétique sur un point de la surface de disques durs.
- Les états chargé et déchargé des condensateurs de RAM.
- Des tensions de courant électrique différentes pour la transmission de données utilisant des fils électriques (bus, câble réseau, ... etc.).
- Des intensités de luminosité différentes pour la transmission avec fibre optique.
- ... etc.



Carte perforée

## 2. DISCRÉTISATION

La **numérisation** est le procédé permettant la construction d'une représentation discrète d'un objet réel.

Exemples :

1. Numérisation d'images.
2. Numérisation d'une capture de mouvement.

La numérisation comporte en général trois phases :

1. L'**échantillonnage** où une des dimensions de l'objet (par exemple, le temps) est explorée à intervalles réguliers ;
2. La **quantification**, où la valeur du signal représentant l'objet dans les zones explorées est arrondie à une valeur prise dans un ensemble fini ;
3. L'**encodage**, qui fait correspondre à chacune de ces valeurs un code binaire.

### 3. CALCULS A BASE DE BITS

En binaire l'information la plus élémentaire pouvant avoir comme valeur 0 ou 1 est appelée «bit». On regroupe les bits par mots de différentes longueurs : mots de 8, 16, 32 ou 64 bits. Les mots de 8 bits sont appelés **octets** ou « *bytes* » en anglais.

En informatique on utilise un système d'unités basée sur l'octet. Les unités suivantes sont le kilooctet ( $1\text{ko} = 2^{10}$  octets = 1024 octets), le mégaoctet ( $1\text{Mo} = 2^{10}$  Ko), le gigaoctet ( $1\text{Go} = 2^{10}$  Mo), le téraoctet ( $1\text{To} = 2^{10}$  Go) et le pétaoctet ( $1\text{Po} = 2^{10}$  To). ([il existe une autre convention utilisant les puissances de 10](#)).

Combien d'objets différents peut-on coder avec n bits ? Les codes possibles sont toutes les suites de n éléments pris parmi 0 ou 1. Il y en a  $2^n$ . On peut donc coder  $2^n$  objets avec n bits : 2 avec 1 bit,  $2^8 = 256$  avec 1 octet.

Inversement, si on dispose de m objets distincts ; combien de bits sont nécessaires pour pouvoir les coder ? Réponse : c'est le plus petit nombre entier n tel que  $2^n \geq m$ . Le nombre y tel que  $2^y = m$  est  $\log_2(m)=y$  donc si y est entier on le prend sinon c'est le premier entier immédiatement supérieur.

**Exemple :** Combien faut-il de bits pour coder 16000 objets ? Puisque  $\log_2(16000) = 13,96$ . Il faut donc 14 bits. On remarquera que  $2^{10} = 1024$  est proche de 1000. Ceci permet des calculs approximatifs :  $16000 \approx 2^4 \cdot 2^{10} = 2^{14}$ .

**Remarque :** Les puissances de 2 qui suivent sont à apprendre :

$2^3$	<b>8</b>	$2^6$	<b>64</b>	$2^9$	<b>512</b>
$2^4$	<b>16</b>	$2^7$	<b>128</b>	$2^{10}$	<b>1024</b>
$2^5$	<b>32</b>	$2^8$	<b>256</b>		

## Chapitre1

# REPRÉSENTATION DES ENTIERS "POSITIFS" (NON SIGNÉS)

### 1.1. Systèmes de numérations

#### 1.1.1 Numérations positionnelle

Dans une numération **positionnelle**, la valeur d'un nombre dépend des chiffres qui le composent et de leurs positions (rangs).

**Exemple :** Considérons le nombre 1975.

- La position de 1 est celle des milliers elle donne  $1 \times 1000$ .
- Celle de 9 représente les centaines, et donne  $9 \times 100$ .
- Celle de 7 représente les dizaines, et donne  $7 \times 10$ .
- Celle de 5 donne  $5 \times 1$ .

C'est un **système de numération positionnel**. En effet, la valeur représentée par le chiffre un de gauche est différente de celle représentée par le deuxième un.

2. L'heure 13:25:30 est écrite dans un système de numération positionnel composé de heures:minutes:secondes.

#### 1.1.2 Système de numération avec base

Un système de numération positionnel peut être associé à une base qui définit:

- le nombre de chiffres ;
- le poids associé à chaque position.

**Exemple :** Dans la base 10 ou base décimale nous disposons de 10 chiffres (de 0 à 9). Après les avoir épuisé par comptage, nous utilisons deux chiffres pour écrire le nombre qui suit.

#### 1.1.3 Poids des chiffres

Dans un système de numération avec base, le **poids d'un chiffre** dépend de sa position (rang), en commençant par 0 depuis la droite, et de la base et est défini par:

$$\text{Poids} = \text{base}^{\text{rang}}$$

**Exemple :**

rang	3	2	1	0
chiffre	1	9	7	5
poids	$10^3$	$10^2$	$10^1$	$10^0$
valeur	1000	900	70	5

**Rem :** Le chiffre le plus à gauche (ici 1) possède le poids **le plus fort** et celui qui est le plus à droite (ici 5) le poids **le plus faible**.

#### 1.1.4 Numération binaire

la base est 2 donc nous n'utilisons que deux chiffres 0 et 1.

**Exemple :** que vaut le nombre binaire 10110 ?

En base 2, le **poids = 2 position**

Positions	4	3	2	1	0
Chiffres binaires	1	0	1	1	0
Valeurs générées	$1 \times 2^4$	$0 \times 2^3$	$1 \times 2^2$	$1 \times 2^1$	$0 \times 2^0$
	<b>16</b>	<b>0</b>	<b>4</b>	<b>2</b>	<b>0</b>

Ce qui donne 22.

**Remarques :**

1. En base 2, le chiffre 2 n'existe pas ; tout comme pour la base 10 il n'existe pas de chiffre pour représenter seul la valeur 10.
2. On peut généraliser les systèmes de numération pour toutes les bases  $\geq 2$  mais les plus utilisées sont :

Système	Base	Symbolique
Décimal	10	0,1,2,3,4,5,6,7,8,9
Binaire	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Hexadécimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

3. Soit N un nombre. La notation suivante permet de distinguer son écriture selon la base utilisée :

$$N = (a_n a_{n-1} \dots a_0)_b$$

ou  $a_0, \dots, a_n$  sont des symboles du système et b est sa base (écrite en décimal).

**Exemple :**

$(1545)_{10}$  en décimal ;  $(9A180)_{16}$  en hexadécimal ;  $(1001001)_2$  en binaire ;  $(74510)_8$  en octal.

**Exercice :** L'écriture  $(18209)_8$  puis  $(1G23E)_H$  sont elles correctes ? Justifier.

## 1.2 Passages entre bases

### 1.2.1 Conversion au système décimal

Si  $N = (a_n a_{n-1} \dots a_0)_b$  alors  $(N)_{10} = a_n \times b^n + a_{n-1} \times b^{n-1} + \dots + a_0 \times b^0$ . On l'appellera la technique du polynôme.

**Exemples :**

$$\begin{aligned}(110101)_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 16 + 0 + 4 + 0 + 1 \\ &= (53)_{10}\end{aligned}$$

$$(175)_8 = 1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 = 64 + 56 + 5 = (125)_{10}$$

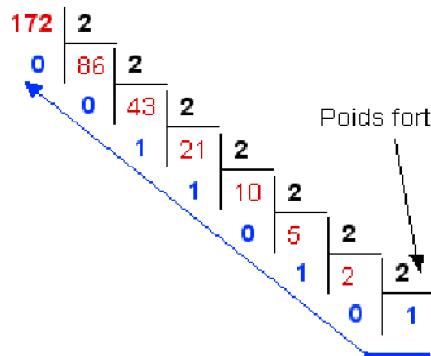
$$(1A2F)_{16} = 1 \times 16^3 + 10 \times 16^2 + 2 \times 16^1 + 15 \times 16^0 = (6703)_{10}$$

### 1.2.2 Conversion de la base 10 à une base b

On effectue des divisions entières successives sur la base puis on prend la suite des restes. Le dernier reste est associé au poids le plus fort.

**Exemples :**

1.  $(172)_{10} = (?)_2$



2.  $(47)_{10} = (?)_8$

$$\begin{array}{r} 47 \\ 7 \quad 5 \quad 8 \\ \quad 5 \quad 0 \end{array}$$

3.  $(2623)_{10} = (?)_H$

$$\begin{array}{r} 2623 \\ F \quad 163 \quad 16 \\ \quad 3 \quad A \end{array}$$

### 1.2.3 Passage entre octal et binaire

Pour passer de l'octal au binaire, il suffit de convertir chaque chiffre octal en son équivalent en binaire sur trois positions comme suit :

Chiffre octal	Nombre binaire équivalent	Chiffre octal	Nombre binaire équivalent
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

**Exemple :**  $(57)_8 = (?)_2$

puisque  $(5)_8 = (101)_2$  et  $(7)_8 = (111)_2$  alors  $(57)_8 = (101111)_2$

Pour passer du binaire à l'octal, on regroupe les bits du nombre triplets en partant de la droite. Si le dernier groupe contient moins de trois bits, on ajoute des zéros. Ensuite on associe à chaque triplet son équivalent en octal.

**Exemples:**

- $(101111101)_2 = (?)_8$

$$101111101 = 101 \ 111 \ 101 = (575)_8$$

- $(1111101)_2 = (?)_8$

$$1111101 => 1 \ 111 \ 101 = 001 \ 111 \ 101 = (175)_8$$

### 1.2.4 Passage entre hexadécimal et binaire

Pour passer de l'hexadécimal au binaire, il suffit de convertir chaque chiffre hexadécimal en son équivalent en binaire sur 4 positions comme suit :

Chiffre hexadécimal	Nombre binaire équivalent	Chiffre hexadécimal	Nombre binaire équivalent
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

**Exemple :**  $(5A7)_H = (?)_2$

puisque  $(5)_H = (0101)_2$  et  $(A)_H = (1010)_2$  et  $(7)_H = (0111)_2$  alors  $(5A7)_H = (010110100111)_2$

Pour passer du binaire à l'hexadécimal, on regroupe les bits du nombre quadruplets en partant de la

droite. Si le dernier groupe contient moins de 4 bits, on ajoute des zéros. Ensuite on associe à chaque quadruplet son équivalent en hexadécimal.

**Exemples :**

**Exemple 1:**  $(101111101)_2 = (?)_{16}$   
 $101111101 = 1 \ 0111 \ 1101 = (17D)_{16}$

**Exemple 2:**  $(1111101)_2 = (?)_{16}$   
 $111 \ 1101 = 111 \ 1101 = 0111 \ 1101 = (7D)_{16}$

**Rems :**

1. Pour passer d'une base  $b_1$  à une base  $b_2$  où  $b_2$  est une puissance de  $b_1$ , on utilise la méthode de compactage décompactage.
2. Pour passer entre deux bases quelconques, on peut utiliser la base 10 comme base intermédiaire.

**Exemple :**  $(A24)_H = (?)_8$

- $(A24)_H = (2596)_{10}$  en appliquant la technique du polynôme.
- $(2596)_{10} = (5044)_8$  en appliquant la technique des divisions successives.

Donc  $(A24)_H = (5044)_8$

### 1.3 Arithmétique binaire

Comme pour le système décimal, les 4 opérations arithmétiques élémentaires peuvent être définies pour le système binaire.

**1.3.1 addition :** Elle repose sur les règles suivantes :

$$0+0 = 0 \quad 0+1 = 1 \quad 1+0 = 1 \quad 1+1 = 0 \text{ avec une retenue } r=1.$$

**Exemple :**  $1011 + 10011 = ?$

$$\begin{array}{r} \text{retenue} & & 1 & 1 \\ & 1 & 0 & 1 & 1 \\ + & 1 & 0 & 0 & 1 & 1 \\ \hline = & 1 & 1 & 1 & 1 & 0 \end{array}$$

**Remarque :** lorsqu'une retenue est générée par le bit le plus à gauche on dit qu'il y a débordement ou dépassement de capacité.

**1.3.2 soustraction :** Elle repose sur les règles suivantes :

$$0-0 = 0 \quad 1-0 = 1 \quad 1-1 = 0 \quad 0-1 = 1 \text{ avec une retenue } r=1.$$

**Exemple :**  $10111 - 1011 = ?$

$$\begin{array}{r} 1 & 0 & 1 & 1 & 1 \\ \text{retenue} & 1 \\ - & & 1 & 0 & 1 & 1 \end{array}$$

$$= \quad 0 \quad 1 \quad 1 \quad 0 \quad 0$$

**1.3.3 multiplication :** Elle repose sur les règles suivantes :

$$0 \times 0 = 0 \quad 1 \times 0 = 0 \quad 1 \times 1 = 0 \quad 0 \times 1 = 1.$$

**Exemple :**  $10111 \times 1011 = ?$

$$\begin{array}{r}
 & 1 & 0 & 1 & 1 & 1 \\
 \times & & 1 & 0 & 1 & 1 \\
 \hline
 & 1 & 0 & 1 & 1 & 1 \\
 & 1 & 0 & 1 & 1 & . \\
 & 1 & 0 & 1 & 1 & . . . \\
 \hline
 = & 1 & 1 & 1 & 1 & 1 & 0 & 1
 \end{array}$$

**1.3.4 division :** on a les 2 règles  $0 / 1 = 0$  et  $1 / 1 = 1$  ; les autres cas sont indéterminés.

**Exemple :**  $10111 / 11 = ?$

$$\begin{array}{r|rr}
 1 & 0 & 1 & 1 & 0 \\
 \hline
 1 & 1 \\
 \hline
 1 & 0 & 1 \\
 & 1 & 1 \\
 \hline
 1 & 0 & 0 \\
 & 1 & 1 \\
 \hline
 0 & 1
 \end{array}$$

Donc  $10111 / 11 = 111$  et le reste est 1.

#### 1.4 Arithmétique en base b

Comme pour le système décimal et binaire, les 4 opérations arithmétiques élémentaires peuvent être définies pour des systèmes à base  $b > 1$ .

**Exemples :**

- division octal : On peut utiliser la table de multiplication octale.

$\times$	1	2	3	4	5	6	7	10
1	1	2	3	4	5	6	7	10
2	2	4	6	10	12	14	16	20
3	3	6	11	14	17	22	25	30
4	4	10	14	20	24	30	34	40
5	5	12	17	24	31	36	43	50
6	6	14	22	30	36	44	52	60
7	7	16	25	34	43	52	61	70
10	10	20	30	40	50	60	70	100

$$(4472)_8 / (13)_8 = (?)_8$$

4	4	7	2		1	3	
3	9				3	9	A
0	B	7					
	A	B					
0	C	2					
	B	E					
0		4					

Donc  $(4472)_8 / (13)_8$  donne  $(10)_8$

2. division hexadécimal : On peut utiliser la table de multiplication hexadécimale

x	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	20
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	30
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	50
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	60
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	70
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	80
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	90
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A0
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B0
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C0
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D0
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E0
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F0
10	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100

$$(4472)_8 / (13)_8 = (?)_8$$

4	4	7	2		1	3	
3	9				3	9	A
0	B	7					
	A	B					
0	C	2					
	B	E					
0		4					

Donc  $(4472)_H / (13)_H = (39A)_H$  et le reste = 4.

## Chapitre 2

# REPRÉSENTATION DES ENTIERS SIGNES

### 2.1 Représentation en SVA (signe et valeur absolue)

En SVA le bit le plus à gauche représente le signe (0 pour positif) et le reste la valeur absolue du nombre en binaire.

Exemples :

1. En SVA sur 4 bits, le nombre 0110 représente +6.
2. En SVA sur 8 bits, le nombre 11110110 représente -118.

Remarques :

1. Avec  $n$  bits on peut représenter des nombres dans l'intervalle *symétrique*  $[-(2^{n-1}-1), +(2^{n-1}-1)]$ .
2. Les nombres dans  $[-7,7]$  sont codés :

$(\text{nombre})_{10}$	SVA	$(\text{nombre})_{10}$	SVA
+7	0111	-0	1000
+6	0110	-1	1001
+5	0101	-2	1010
+4	0100	-3	1011
+3	0011	-4	1100
+2	0010	-5	1101
+1	0001	-6	1110
+0	0000	-7	1111

3. En SVA, zéro possède deux codes 100...0 et 000...0.
4. Arithmétique SVA : Pour faire une addition le bit de signe doit être traité à part : i.e. comme en décimal signé. Si les signes sont différents, on soustrait le plus petit en valeur absolue du plus grand puis on ajoute le bit de signe du plus grand. Si les signes sont les mêmes on additionne les valeurs absolues puis on ajoute le bit de signe. Le résultat est juste s'il n'y a pas de retenue dans le bit le plus fort associé à la valeur absolue.

Exemple : Sommes en SVA sur 4 bits :

a.  $(0001)_{\text{sva}} + (1011)_{\text{sva}} = (?)_{\text{sva}}$

Les signes sont différents et  $(011) > (001)$  donc on fait  $(011)-(001)=(010)$  puis on ajoute le signe

du plus grand en valeur absolue. On obtient  $(1010)_{sva}$ . La somme binaire donnerait 1100 ce qui est faux  $(1+(-3)) = -2$  et non pas  $-4$ .

$$b. (0101)_{sva} + (0111)_{sva} = (?)_{sva}$$

Les signes sont identiques donc on fait  $(101) + (111) = (100)$  avec une retenue donc il y a débordement. L'opération ne peut être réalisée en SVA sur 4 bits.

## 2.2 Représentation en CPL1 (Complément à 1)

Les nombres positifs sont représentés en SVA. Un nombre négatif  $x$  est déduit de son opposé  $-x$  en inversant tous ses bits.

**Exemple :** sur 4 bits +3 est codé 0011. Donc -3 est codé par 1100.

### Remarques

1. En CPL1  $n$  bits permettent de coder des nombres dans  $[-(2^{n-1}-1), +(2^{n-1}-1)]$ .
2. Les nombres dans  $[-7, 7]$  sont codés ainsi :

(nombre) <sub>10</sub>	SVA	CPL1	(nombre) <sub>10</sub>	SVA	CPL1
+7	0111	0111	-0	1000	1111
+6	0110	0110	-1	1001	1110
+5	0101	0101	-2	1010	1101
+4	0100	0100	-3	1011	1100
+3	0011	0011	-4	1100	1011
+2	0010	0010	-5	1101	1010
+1	0001	0001	-6	1110	1001
+0	0000	0000	-7	1111	1000

3. comme en SVA, zéro possède 2 représentations.

4. L'addition en CPL1 obéit aux règles suivantes :

- Si aucune retenue n'est générée par le bit de signe le résultat est correct.
- Si une retenue est générée par le bit de signe, elle est additionnée au résultat.
- Si les deux nombres ont un même signe qui est différent de celui du résultat il y a dépassement de capacité ; sinon le résultat est correct.

### Exemple

$$\begin{array}{r}
 1 & 0 & 0 & 1 & 1 & (-12)_{10} \\
 + & 0 & 1 & 1 & 0 & 1 & (+13)_{10} \\
 \hline
 1 & 0 & 0 & 0 & 0 & 0 & \\
 \end{array}
 \quad
 \begin{array}{r}
 2. \\
 (-12)_{10} & 1 & 0 & 0 & 1 \\
 & 12 \\
 & 0 & 0 & 0 & 1 & (+1)_{10}
 \end{array}$$

End-around carry

$$\begin{array}{r}
 (-12)_{10} \quad 1\ 0011 \\
 \hline
 0\ 0110 \\
 \text{retenue } 1 \\
 \hline
 (-24)_{10} \quad 0\ 0111
 \end{array}$$

Le résultat +7 est faux, dépassement.

### 2.3 Représentation en CPL2 (Complément à 2)

Les nombres positifs sont représentés en SVA. Un nombre négatif  $x$  est déduit de son opposé  $-x$  en inversant tous ses bits puis on ajoute un.

**Exemple :** sur 4 bits +3 est codé 0011. Donc -3 est codé par  $1100+1=1101$ .

#### Remarques

1. En CPL2  $n$  bits permettent de coder des nombres dans  $[-2^{n-1}, +(2^{n-1}-1)]$  (non symétrique).
2. Les nombres dans  $[-8, 7]$  sont codés ainsi :

Décimal	Binary	Décimal	Binary
+7	0111	-1	1111
+6	0110	-2	1110
+5	0101	-3	1101
+4	0100	-4	1100
+3	0011	-5	1011
+2	0010	-6	1010
+1	0001	-7	1001
+0	0000	-8	1000

3. Zéro possède une seule valeur.
4. La conversion positif-négatif et inversement est simple. En effet, il suffit d'inverser les bits et d'ajouter un. Cette conversion est cohérente.

#### Exemple

- + 2 = 0010 inverser les bits donne 1101 et l'ajout de 1 donne 1110 qui est -2
- 2 = 1110 inverser les bits donne 0001 et l'ajout de 1 donne 0010 qui est +2

5. L'opposé du plus petit nombre est lui même (vérifier -8 sur 4 bits).

6. L'addition en CPL2 obéit aux règles suivantes :

- Si une retenue est générée par le bit de signe, elle est ignorée.
- Si les deux nombres ont un même signe qui est différent de celui du résultat il y a dépassement de capacité.

#### Exemples

1.

$$\begin{array}{r}
 (+2)_{10} \quad 0010 \\
 (-4)_{10} \quad \underline{1100}
 \end{array}$$

$$(-2)_{10} \quad 1110$$

2.

$$(+2)_{10} \quad 0010$$

$$\begin{array}{r} (-2)_{10} \quad 1110 \\ (+2)_{10} \quad 0010 \\ \hline (0)_{10} \quad 0000 \end{array}$$

3.

$$(-7)_{10} \quad 1001$$

$$(-7)_{10} \quad 1001$$

Les deux nombres sont négatifs et le résultat

$$(-14)_{10} \quad \textcolor{red}{0010}$$

positif donc débordement. Résultat +2 faux.

## Chapitre3

# REPRÉSENTATION DES NOMBRES RÉELS

## 3.1 REPRÉSENTATION EN VIRGULES FIXE

### 3.1.1 Nombres binaires fractionnaires

Le nombre  $0,135$  désigne le nombre obtenu par :  $1 \times 10^{-1} + 3 \times 10^{-2} + 5 \times 10^{-3}$ .  $0$  est sa partie entière et  $135$  sa partie décimale (abus de langage) ou fractionnaire.

De même, on peut représenter des nombres à virgule en binaire. Par exemple, le nombre  $(1,11)_2$  désigne le nombre  $2^0 + 2^{-1} + 2^{-2} = 1 + 0,5 + 0,25 = 1,75$ .

### 3.1.2 Passage du décimal au binaire

On utilise une méthode à deux étapes :

- **étape 1** : convertir la partie entière par divisions successives.
- **étape 2** : convertir la partie fractionnaire en prenant la partie entière des résultats des multiplications successives par la base 2. On arrête lorsque la partie fractionnaire est nulle ou que la précision voulue est atteinte.

**Exemple :**  $(4.25)_{10} = (?)_2$

**1. étape 1 :**  $(4)_{10} = (100)_2$

**2. étape 2 :**  $0.25 * 2 = 0.5$  on prend 0 ;  $0.5 * 2 = 1.00$  on prend 1 et on arrête. Donc  $(0.25)_{10} = (0.01)_2$

Par conséquent,  $(4.25)_{10} = (100.01)_2$ .

**Remarques :**

1. Cette méthode de conversion peut être généralisée à n'importe quelle base  $b >= 2$ .
2. La conversion peut ne pas être précise. Par exemple,  $(0.8)_{10} = (0.11001100...)_2$ . Si on prend donc 4 bits pour la partie fractionnaire  $(0.1100)_2$  donne  $(0.75)_{10}$  ce qui est une erreur relativement considérable.
3. De manière générale, un nombre signé fractionnaire peut être représenté en SVA à virgule fixe avec un bit de signe+une partie entière+une partie fractionnaire. Par exemple, avec six bit dont 3 pour la partie entière +7.75 qui est codé 011111 est le plus grand réel représentable tandis que le plus petit est -7.75. (la position de la virgule est déduite de la taille des parties.).

## 3.2 REPRÉSENTATION EN VIRGULE FLOTTANTE

### 3.2.1 Principe général

La notation en virgule flottante est tirée de la notation scientifique.

**Exemple :**  $0.85$  est écrit  $8.5 * 10^1$  ou  $85 * 10^{-2}$ .

Utiliser cette notation revient à coder trois informations :

Signe s	Exposant e	Mantisse M
---------	------------	------------

Ce qui donne pour la première écriture de 0.85 : 0 pour le signe -1 pour l'exposant et 85 pour la mantisse ou [0,-1,85].

### Rem :

1. La position du point décimal et la base ne sont pas représentées.
2. Pour représenter l'exposant on utilise CPL2 ou on applique un **décalage** ou **biais** qui rend l'exposant non négatif.
  - le biais doit être fixe et suffisant pour rendre toute valeur possible  $\geq 0$  ;
  - En binaire, nous verrons que la valeur du biais est en général égale à  $2^{\text{nombre de bits associés}} - 1$  ;
  - CPL2 facilite l'addition mais ne facilite pas les comparaisons.
  - Par défaut, on supposera une représentation par biais.

**Exemple :** avec un décalage de 16 le nombre  $0.85 \times 10^6$  devient [0,-6+16,85] ou [0,10,85].

Réciproquement, le nombre [1,13,76] représenté avec décalage vaut  $-0.76 \times 10^{13-16} = -0.00076$

3. Un nombre réel est **normalisé** si

- le chiffre à droite du point décimal est différent de zéro et
  - sa partie entière est nulle.
- (exception faite de zéro qui est supposé normalisé).

**Exemple :**  $5.86 \times 10^2$  et  $0.0062 \times 10^{-5}$  ne sont pas normalisés tandis que  $0.8105 \times 10^2$  l'est.

4. Pour pouvoir interpréter de manière unique un nombre codé en [signe, exposant, mantisse] ou [s,e,M], on doit connaître :

- la base,
- le biais et
- on supposera que la mantisse est normalisée.

**Exemple :** Si la base est 10 et le biais 16, [1,13,76] est interprété par  $-0.76 \times 10^{13-16} = -0.00076$ .

### 3.2.2 Virgule flottantes en binaire

En VF binaire on utilise une exponentiation en base 2. La normalisation de la Mantisse M consiste à rendre  $1 \leq M < 2$ . En outre, parce que le bit le plus significatif est toujours 1 pour un nombre normalisé, ce bit n'est pas stocké est dit bit caché ou implicite (hidden bit).

**Exemples :** On suppose que l'exposant est codé sur 7 bits, et la mantisse sur 16 bits.

1. On veut représenter le nombre 125.5 :

Étape 1 : convertir le nombre en binaire :  $(125.5)_{10} = (1111101.1)_2$

Étape 2 : déterminer la mantisse normalisée :  $(1111101.1)_2 = 1.1111011 \times 2^6$ . Le bit le plus significatif sera caché.

Étape 3 : ajouter le biais= $2^{\text{nbrBitExpo}-1}$  à l'exposant et déterminer sa représentation en binaire : biais= $2^6=64$  donc Exposant= $6+64=70=(1000110)_2$ . La représentation de 125.5 est donc :

0	100 0110	1111 0110 0000 0000
signe	Exposant	Mantisso

Ou 0 1000110 1111 0110 0000 0000.

2. Réciproquement, quel est en décimal le nombre  $A=[1,1000110,1110101010111100]$  ?

Le signe est -.  $e=(1000110)_2-(64)_{10}=70-64=6$ .

$M=(1.1110101010111)_2$  après ajout du bit caché donc

$$A=-M \times 2^6 = 1111010.10101111 = -(2^1+2^3+2^4+2^5+2^6+2^{-1}+2^{-3}+2^{-5}+2^{-6}+2^{-7}+2^{-8}) = -122.68359375.$$

### 3.2.2 Notation IEEE 754

Le standard utilise la base 2 et inclut deux représentations : simple et double précision.

mode	Taille totale	Exposant	biais	$E_{\min}$	$E_{\max}$	mantisso
Simple précision	32	8	127	-126	+127	23
Double précision	64	11	1023	-1022	+1023	52

**Remarque :**

Forme dénormalisée : Si  $e=E_{\min}$ , le nombre est dit dénormalisé et le bit caché est dans ce cas nul. Cette forme permet de représenter les nombres très petits.

## 3.3 Arithmétique de base en virgules flottantes

### 3.3.1 Multiplication

$$(-1)^{s1} \cdot M_1 \times 2^{e1} \times (-1)^{s2} \cdot M_2 \times 2^{e2} = (-1)^{(s1+s2)} \cdot (M_1 \times M_2) \times 2^{e1+e2-\text{biais}}$$

L'algorithme de la multiplication flottante est donc :

1. Multiplication entière des mantisses.
2. Calcul du nouvel exposant égal à la somme des exposants moins le biais.
3. Construction de la représentation  $[s,e,M]$  préliminaire.
4. Normaliser le résultat : déplacer le point décimal à droite d'une position décrémenter l'exposant par un. Dans le cas contraire il est incrémenté.
5. Supprimer le bit caché et arrondir au nombre de bits de la mantisse.

**Rem :** Pour simplifier on n'utilisera que l'arrondi par troncature.

**Exemple :**

Le biais est 127 et la mantisse à 4 bits

$$0 \ 1000100 \ 0100 \times 1 \ 00111100 \ 1100 = ?$$

1. Multiplication des mantisses:

$$\begin{array}{r} 1.0100 \\ \times 1.1100 \\ \hline 101 \\ 101. \\ 101.. \\ \hline 100011 \end{array}$$

ce qui donne 10.0011

2. On additionne les exposants:

$$\begin{array}{r} \text{biased:} \\ 10000100 \\ + 00111100 \\ \hline 11000000 \end{array}$$

On enlève le biais en plus

$$\begin{array}{r} 11000000 \\ - 01111111 \\ \hline 1000001 \end{array}$$

3. On construit la représentation :

1 01000001 10.00110000

4. On normalise le résultat:

1 01000001 10.00110000 devient  
1 01000010 1.000110000

5. On arrondit après suppression du bit caché :

1 01000010 0001 qui est la représentation finale.

### 3.3.2 Division

$$(-1)^{s1} \cdot M_1 \times 2^{e1} \times (-1)^{s2} / M_2 \times 2^{e2} = (-1)^{(s1+s2)} \cdot (M_1/M_2) \times 2^{e1-e2+\text{biais}}$$

L'algorithme de la division flottante est donc :

1. Division des mantisses.
2. Calcul du nouvel exposant égal à la soustraction des exposants plus le biais.
3. Construction de la représentation [s,e,M] préliminaire.
4. Normaliser le résultat.
5. Supprimer le bit caché et arrondir au nombre de bits de la mantisse.

**Exemple :** Le biais est 127, la mantisse a 5 bits et l'exposant en a 8.

Soit à réaliser  $2.375/8.25$  en VF binaire

$$(2.375)_{10} = [0-1000\ 0000-00110]_{VF2}$$

$$\text{et } (8.25)_{10} = [0-1000\ 0010-00001]_{VF2}$$

**IEEE representation of "2.375":**

0 10000000 001100000000000000000000  
[0—128—0.187500]

**IEEE representation of "8.25":**

0 10000010 000010000000000000000000  
[0—130—0.031250]

1. On divise les mantisses :

1.0011/1.00001 donne 1.001001...

2. On calcule le nouvel exposant :

$10000000 - 10000010 = -(10)_2$  on ajoute le biais 127 on aura  $127 - 2 = 125 = (0111\ 1101)_2$

Le signe du résultat est positif.

On aura donc après suppression du bit caché et troncature :

0 0111 1101 0010

### 3.3.3 Addition et soustraction

L'algorithme est :

1. Aligner les nombres vers l'exposant le plus grand.
2. le signe du résultat et celui du nombre ayant la mantisse la plus grande;
3. Additionner si les signes sont identiques ; autrement soustraire.
4. Normaliser le résultat et arrondir.

#### Exemple :

Le biais est 127, la mantisse et l'exposant en ont 8.

Soit à additionner 0.25 et 100 en VF binaire

$$\begin{array}{r} 0.25 = \quad 0\ 0111\ 1101\ 000\ 0000\ 0000\ 0000\ 0000 \\ 100 = \quad 0\ 1000\ 0101\ 100\ 1000\ 0000\ 0000\ 0000 \end{array}$$

1. Aligner les points décimaux

On choisit 0.25 pour déplacer le point fractionnaire de  
 $10000101 - 01111101 = 00001000$  (8) places.

01111101 1.00000000 (original value)  
01111110 0.10000000 (shifted 1 place)  
01111111 0.01000000 (shifted 2 places)  
10000000 0.00100000 (shifted 3 places)  
10000001 0.00010000 (shifted 4 places)  
10000010 0.00001000 (shifted 5 places)  
10000011 0.00000100 (shifted 6 places)  
10000100 0.00000010 (shifted 7 places)  
10000101 0.00000001 (shifted 8 places)

2. On additionne

$$1.10010000 \quad (100)$$

+ 0.0000001 (.25)

-----

1.1001001

3. On normalise le résultat et on supprime le bit caché :

0 10000101 10010001

## Chapitre4

# REPRÉSENTATION DES CARACTÈRES ALPHANUMÉRIQUES

Afin de pouvoir transmettre ou stocker tous les caractères alphanumériques des codes ont été établis. Il existe de nombreux codes tels que le code ASCII , l'EBCDIC et L'UNICODE.

## 4.1 CODE ASCII

### 4.1.1 Description

Le code ASCII représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Utilisé initialement par les **téléscripteurs** il est adopté par les systèmes informatiques pour coder les caractères envoyés depuis le clavier ou vers l'imprimante.

La table des codes ASCII est composée de 2 parties :

a) La première de 0 à 31 regroupe les caractères non imprimables utilisés pour le contrôle. Par exemple le caractère 12 ordonne à l'imprimante de passer au début d'une nouvelle page ; 7 cause un beep sonore dans le téléscripteur récepteur.



**Table ASCII des caractères de contrôle non  
imprimables**

Téléscripteur Siemens T100 (1958)

Décimal	Caractère	Décimal	Caractère
0	nul	16	échappement transmission
1	début d'en-tête	17	commande de dispositif 1
2	début de texte	18	commande de dispositif 2
3	fin de texte	19	commande de dispositif 3
4	fin de transmission	20	commande de dispositif 4
5	demande	21	accusé de réception négatif
6	accusé de réception	22	synchronisation
7	sonnerie	23	fin de bloc de transmission
8	espace arrière	24	annulation
9	tabulation horizontale	25	fin de support
10	changement de ligne/nouvelle ligne	26	substitution
11	tabulation verticale	27	échappement
12	saut de page/nouvelle page	28	séparateur de fichiers
13	retour de chariot	29	séparateur de groupes

14	hors code	30	séparateur d'enregistrements
15	en code	31	séparateur d'unités

b) La deuxième regroupe les caractères imprimables ASCII et leur affecte les nombres de 32 à 126. Le nombre 127 représente la commande SUPPRESSION.

**Table des caractères imprimables ASCII**

Décimal	Caractère	Décimal	Caractère	Décimal	Caractère	Décimal	Caractère
32	espace	56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	"	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	\$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	w	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	Suppr

**Exemple :** En écrivant Bonjour en ASCII nous obtenons : 066 111 110 106 111 117 114 en décimal et 42 6F 6E 6A 6F 75 72 en hexadécimal et 01000010 01101111 01101110 01101010 01101111 01110101 01110010 en binaire.

**Remarque :** Les codes 65 à 90 représentent les majuscules et ceux de 97 à 122 les minuscules. La

différence étant 32, il suffit de modifier le 6<sup>ème</sup> bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale. Exemple : 01000010 vaut 66 et représente B. On modifie le sixième bit on obtient 01100010 qui représente b.

#### 4.1.2 Table ASCII étendue

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués. Il a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères. Mais si les 128 premiers caractères sont identiques a ceux de la table ASCII de base et le reste fut différent d'un pays voire d'une région a une autre. Parmi les extensions on trouve :

- ISO-8859-1 : ou Latin-1 langues de l'Europe de l'Ouest (caractères accentués).
- windows-1252 : une variation Microsoft sur le ISO-Latin-1;
- ISO-8859-2 : langues de l'Europe Centrale et de l'Est ;
- ISO-8859-3 : langues turcs ;
- ISO-8859-4 : langues baltes (Lituanie, Biélorussie, ...);
- ISO-8859-5 : alphabet cyrillique (russe, bulgare, ...);
- ISO-8859-6 : arabe ;
- ISO-8859-7 : grec moderne ;
- ISO-8859-8 : hébreu ;
- ..., etc.

**Exemple :** é est codé par 233. Le même code représente σ en latin/arabic et ι iota en grec.

### 4.2 UNICODE

#### 4.2.1 Description

Unicode est un système d'encodage pouvant contenir tous les caractères utilisés dans le monde. Il attribut à chaque caractère un identifiant nommé **code point**. Unicode utilise la notation hexadécimale préfixée par « U+ » pour représenter un code point : Par exemple le caractère A est codé U+0041.

#### 4.2.2 Formes d'encodage

Pour mettre en œuvre UNICODE des protocoles d'encodage ont été proposés. Parmi les plus utilisés:

- **UTF-16** : Utilise généralement deux octets pour représenter les caractères.
- **UTF-8** : Utilise un nombre variable de bits selon le numéro du caractère a encoder:

- + de 0 à 127 (caractères ASCII) : un octet de la forme `0xxx xxxx`;
- + de 128 à 2047 : deux octets de la forme `110x xxxx 10xx xxxx`;
- + de 2048 à 65 535 : trois octets de la forme `1110 xxxx 10xx xxxx 10xx xxxx`;
- + de 65 536 à 1 114 111 : quatre octets de la forme `1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx`.

<b>Bits of code point</b>	<b>First code point</b>	<b>Last code point</b>	<b>Bytes in sequence</b>	<b>Byte 1</b>	<b>Byte 2</b>	<b>Byte 3</b>	<b>Byte 4</b>
7	U+0000	U+007F	1	0xxxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Exemple :

On considère la chaîne "hé !". En ISO-Latin-1 décimal les codes sont : 104-233-32-33. Par conséquent, on utilisera la chaîne 0110 1000-1110 1001-0010 0000-0010 0001 pour la coder en ISO-Latin-1. En UTF-8 les caractères non accentués garde leur encodage. Le code point de é est U+00E9 ou  $(233)_{10}$  donc on utilise le motif 110x xxxx 10xx xxxx. Sur 11 bits 233 s'écrit 000 1110 1001 qui distribués sur le motif donnent : 1100 0011 1010 1001. Enfin la chaîne "hé !" sera donc encodée en UTF-8 par : 0110 1000-1100 0011 1010 1001-0010 0000-0010 0001.

Pour l'opération inverse, on essaye de décoder la chaîne 0110 1000-1100 0011 1010 1001-0010 0000-0010 0001 par UTF-8. Le 1ier octet commence par zéro donc ASCII, le deuxième par 110 donc double octets et le 2ième doit commencer par 10 ce qui est le cas. On peut alors tirer du motif double octets les bits du code point c-a-d xxx0 0011 xx10 1001 → 000 1110 1001 → U+00E9 qui est le code point du é. Le reste c'est du code ASCII.

**Remarque :** On dit que UTF-8 est un encodage auto-synchronisant car il est en mesure de détecter la non conformité d'un texte à son protocole. En effet, si un décodeur utf8 essaye d'interpréter la chaîne 0110 1000-1110 1001-0010 0000-0010 0001, le 2<sup>ième</sup> octet annonce l'arrivée de 2 autres octets préfixés par 10 pour former un motif de 3 octets ce qui n'est pas le cas. Par contre, si un décodeur Latin-1 essaye d'interpréter de l'utf8, il n'y a pas de détection d'anomalie. Par exemple, la chaîne 0110 1000-1100 0011 1010 1001-0010 0000-0010 0001, donne les codes décimaux 104-195-169-32-33, i.e. le texte "hÃ© !".

## Chapitre 5

# INTRODUCTION A L'ALGÈBRE DE BOOLE

## 5.1 Notions de base

### 5.1.1 Description

L'algèbre de Boole traite des opérations et des règles applicables à l'ensemble réduit à deux valeurs  $\{0,1\}$  notées aussi faux et vrai respectivement. Une variable booléenne (on dit aussi logique) peut prendre une valeur 0 ou 1 uniquement. L'algèbre de Boole définit trois opérateurs de base:

- Négation ou inversion : Il est noté NON ou  $\overline{\dots}$ . C'est un opérateur unaire qui appliqué à une variable booléenne permet d'en inverser la valeur. Exemple : si  $a$  est une variable booléenne alors sa négation notée NON  $a$  ( NOT  $a$ ,  $\bar{a}$  ou  $\neg a$ ) donne son complément. Donc si  $a=1$  alors  $(\text{NON } a)=0$  et inversement.
- Conjonction : Il est noté ET ou ' $\cdot$ '. C'est un opérateur binaire qui appliqué sur des opérandes booléens retourne 1 ssi les deux opérandes sont à 1. Exemple : si  $a=1$  et  $b=0$  alors  $a \text{ ET } b$  vaut 0.
- Disjonction : Il est noté OU ou '+'. C'est un opérateur binaire qui appliqué sur des opérandes booléens retourne 0 ssi les deux opérandes sont à 0. Exemple : si  $a=1$  et  $b=0$  alors  $a \text{ OU } b$  vaut 1.

Rem : On supposera que la priorité des opérateurs est  $\text{NON} > \text{ET} > \text{OU}$ .

### 5.1.2 Propriétés de base

1. Involution :  $\overline{\overline{a}} = a$
2. Idempotence :  $a+a=a$        $a.a=a$
3. Complémentarité :  $a.\overline{a}=0$        $a+\overline{a}=1$
4. Éléments neutres :  $a.1=1.a=a$        $a+0=0+a=a$
5. Absorption :  $a+1=1$        $a.0=0$  (0 est un absorbant pour la conjonction et 1 pour la disjonction)
6. Associativité :  $(a.b).c=a.(b.c)$        $(a+b)+c=a+(b+c)$
7. Distributivité :  $a.(b+c)=(a.b)+(a.c)$   
 $a+(b.c)=(a+b).(a+c)$
8. Règles de De Morgan :  $\overline{(a+b)}=\overline{a}.\overline{b}$        $\overline{a.b}=\overline{a}+\overline{b}$

**Remarque :** Une écriture comme  $a+(b.c)$  ou  $a+1$  est dite expression booléenne ou logique. Elle

combine entre constantes, variables et opérateurs booléens et si elle est correct son évaluation donne une valeur booléenne.

### 5.1.3 Table de vérité

Elle donne la valeur de l'expression booléenne pour toutes les combinaisons possibles des valeurs de ses variables. Les tables de vérité des opérateurs de base sont:

$a$	$\bar{a}$	$a$	$b$	$a + b$	$a$	$b$	$a \cdot b$
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	0
		1	0	1	1	0	0
		1	1	1	1	1	1

### 5.1.4 Fonctions logiques

Une fonction logique ou booléenne est une fonction qui prend en entrée une ou plusieurs variables booléenne et retourne une valeur booléenne qui dépend des valeurs des variables d'entrée. On peut définir une fonction logique de deux façons :

- Par son expression logique (ou booléenne). Exemple : fonction à 3 variables :  $f(a,b,c) = a.b + a.\bar{c} + c.\bar{b}$ .
- Par sa table de vérité. Exemple :

$a$	$b$	$c$	$a.b$	$\bar{c}$	$a.\bar{c}$	$a.b + a.\bar{c}$	$\bar{b}$	$c.\bar{b}$	$f$
0	0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	0	1	1	1
0	1	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	0	1
1	0	1	0	0	0	0	1	1	1
1	1	0	1	1	1	1	0	0	1
1	1	1	1	0	0	1	0	0	1

**Remarque :** Deux fonctions logiques sont identiques si :

- on peut montrer qu'elles sont équivalentes via les propriétés et règles de l'algèbre de Boole.

**Exemple :** Soient  $f_1(a,b,c) = a.b + a.\bar{b} + c$  et  $f_2(a,c) = a + c$ .

Nous avons :  $a.b+a.\bar{b}+c=(a.(b+\bar{b}))+c=(a.1)+c=a+c$  donc  $f_1$  et  $f_2$  sont identiques.

- Leurs tables de vérité sont identiques.

**Exercice :** Vérifier que les tables de vérité de  $f_1$  et  $f_2$  sont identiques.

## 5.2 FORMES CANONIQUES

### 5.2.1 Définitions

Soit  $f$  une fonction booléenne de  $n$  variables.

**Définition 1 :** On appelle **minterme** toute conjonction des  $n$  variables pouvant être complémentées.

**Exemple :** avec 3 variables  $a,b,c$  :  $a.b.c$  et  $a.\bar{b}.c$  sont des mintermes.  $a+bc$  et  $ab$  et  $a+b+c$  ne le sont pas.

**Définition 2 :** On appelle **maxterme** toute disjonction des  $n$  variables pouvant être complémentées.

**Exemple :** avec 3 variables  $a,b,c$  :  $a+b+c$  et  $a+\bar{b}+\bar{c}$  sont des maxtermes.  $a+bc$  et  $a+b$  et  $\overline{a+b+c}$  ne le sont pas.

**Définition 3,4 :** On appelle **première forme canonique** l'écriture d'une fonction logique comme disjonction de mintermes et on appelle **seconde forme canonique** son écriture comme conjonction de maxtermes. **Exemple :**  $f_1(a,b,c)=\bar{a}\bar{b}c+\bar{a}.\bar{b}c+abc$  est écrite en première forme canonique.  $f_2(a,b,c)=(a+\bar{b}+c).(\bar{a}+\bar{b}+c).(a+b+c)$  est écrite en seconde forme canonique.

### 5.2.2 Passage algébrique aux formes canoniques

Pour déterminer la première forme canonique d'une fonction logique on peut utiliser les règles et propriétés de l'algèbre de Boole et en particulier les règles  $a+\bar{a}=1$  et  $a.\bar{a}=0$ .

**Exemple :** Soit  $f(a,b,c)=ab+\bar{b}c+a\bar{c}$

$$\begin{aligned} f(a,b,c) &= ab(c+\bar{c}) + \bar{b}c(a+\bar{a}) + a\bar{c}(b+\bar{b}) \\ &= abc+abc+\bar{b}ca+\bar{b}ca+acb+acb \\ &= abc+abc+\bar{a}bc+\bar{a}bc+a\bar{c}+\bar{a}bc \end{aligned}$$

Pour déterminer la seconde forme canonique d'une fonction logique  $f$ , on peut écrire  $\bar{f}$  sous première forme puis utiliser la règle de De Morgan pour avoir car  $\bar{\bar{f}}=f$ .

**Exemple :** Soit  $f(a,b,c)=ab+\bar{b}c+a\bar{c}$

$$\begin{aligned} \bar{f}(a,b,c) &= (\bar{a}+\bar{b}).(\bar{b}+\bar{c}).(\bar{a}+c) \\ &= (\bar{a}b+\bar{a}c+0+\bar{b}\bar{c}).(\bar{a}+c) \\ &= \bar{a}b+\bar{a}bc+\bar{a}c+0+\bar{b}\bar{c} \\ &= \bar{a}b(c+\bar{c})+\bar{a}bc+\bar{a}(\bar{b}+\bar{b})\bar{c}+\bar{a}\bar{b}\bar{c} \\ &= \bar{a}bc+\bar{a}bc+\bar{a}\bar{b}\bar{c}+\bar{a}\bar{b}\bar{c}+\bar{a}\bar{b}\bar{c} \\ &= \bar{a}bc+\bar{a}\bar{b}\bar{c}+\bar{a}\bar{b}\bar{c} \end{aligned}$$

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

### 5.2.3 Passage aux formes canoniques par table de vérité

Pour avoir la première forme canonique on construit la disjonction des mintermes associés à la valeur de sortie 1.

1. Pour chaque minterme si la valeur de la variable est 1 on utilise la variable ; si elle vaut 0 on utilise son complément. Pour la seconde forme, on fait la même chose pour déterminer  $\overline{f(a,b,c)}$  puis De Morgan.

**Exemple :** Dans l'exemple précédent, pour avoir la première forme :

- ◆  $f(a,b,c) = 1$  quand :
  - ◆  $a = 0, b = 0$  et  $c = 1$  d'où le minterme  $\bar{a}\bar{b}c$
  - ◆  $a = 1, b = 0$  et  $c = 0$  d'où le minterme  $a\bar{b}\bar{c}$
  - ◆  $a = 1, b = 0$  et  $c = 1$  d'où le minterme  $a\bar{b}c$
  - ◆  $a = 1, b = 1$  et  $c = 0$  d'où le minterme  $a\bar{b}\bar{c}$
  - ◆  $a = 1, b = 1$  et  $c = 1$  d'où le minterme  $a\bar{b}c$
- ◆ On fait le OU de ces mintermes
  - ◆  $f(a, b, c) = abc + ab\bar{c} + a\bar{b}c + \bar{a}\bar{b}c + a\bar{b}\bar{c}$

Pour avoir la seconde forme :

- ◆  $f(a,b,c) = 0$  quand :
  - ◆  $a = 0, b = 0$  et  $c = 0$  d'où le minterme  $\bar{a}\bar{b}\bar{c}$
  - ◆  $a = 0, b = 1$  et  $c = 0$  d'où le minterme  $\bar{a}b\bar{c}$
  - ◆  $a = 0, b = 1$  et  $c = 1$  d'où le minterme  $\bar{a}bc$
- ◆ On fait le OU de ces mintermes
  - ◆  $\overline{f(a, b, c)} = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}bc$
- ◆ Au final :
  - ◆  $\overline{f(a, b, c)} = (a + b + c)(a + \bar{b} + c)(a + \bar{b} + \bar{c})$

## Appendice

Les pages suivantes contiennent dans l'ordre :

1. Table ASCII de base.
2. Extension Latin-1.
3. Extension ISO-8859-6.
4. Une partie de la table d'UNICODE.

# Character sets: ISO-8859-6 (Arabic)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	Ø 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	:	;	<	=	>	?
40	Ø 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[ 005B	\ 005C	] 005D	^ 005E	— 005F
60	~ 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	{ 007D	~ 007E	<u>DEL</u> 007F
80	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
90	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
A0	<u>NBSP</u> 00A0	.....	.....	** 00A4	.....	.....	.....	.....	.....	.....	.....	‘ 060C	— 00AD	.....	.....	.....
B0	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	!	.....	.....	.....	.....	?
C0	ـ 0621	ـ 0622	ـ 0623	ـ 0624	ـ 0625	ـ 0626	ـ 0627	ـ 0628	ـ 0629	ـ 062A	ـ 062B	ـ 062C	ـ 062D	ـ 062E	ـ 062F	ـ 061F
D0	ـ 0630	ـ 0631	ـ 0632	ـ 0633	ـ 0634	ـ 0635	ـ 0636	ـ 0637	ـ 0638	ـ 0639	ـ 063A	ـ 063B	ـ 063C	ـ 063D	ـ 063E	ـ 063F
E0	ـ 0640	ـ 0641	ـ 0642	ـ 0643	ـ 0644	ـ 0645	ـ 0646	ـ 0647	ـ 0648	ـ 0649	ـ 064A	ـ 064B	ـ 064C	ـ 064D	ـ 064E	ـ 064F
F0	ـ 0650	ـ 0651	ـ 0652	ـ 0653	ـ 0654	ـ 0655	ـ 0656	ـ 0657	ـ 0658	ـ 0659	ـ 065A	ـ 065B	ـ 065C	ـ 065D	ـ 065E	ـ 065F