Université des Sciences et de la Technologie d'Oran
Faculté des Mathématiques et Informatique
Département d'informatique

Polycopié

Intitulé

Codification et Représentation de l'Information

Auteur

Dounia YEDJOUR

Année Universitaire : 2017 - 2018

Sommaire

Liste des tableaux

Chapitre 0

Introduction

L'homme a besoin d'un système de codage pour identifier, quantifier, qualifier les objets, les lieux, les évènements…Ces codes lui permettent de mémoriser, traiter et communiquer les informations. Ils peuvent être : le langage, l'écriture, numérations, le graphisme, etc..
Chaque code respecte des règles bien définies. Par exemple, à l'écriture correspond:

- Une liste des symboles prédéfinis : L'alphabet
- Des règles d'utilisations des symboles :
  Plusieurs symboles (lettres) → un mot
  Plusieurs mots → une phrase
- Des règles de syntaxe pour ordonner les mots dans une phrase.

Pour un ordinateur, le codage de l'information permet d'établir une correspondance qui permet sans ambiguïté de passer d'une représentation d'une information dite externe (une image, un son, un texte..) à une autre représentation de la même information (dite interne: sous forme binaire, n'utilisant que des 0 et des 1) suivant un ensemble de règles précises.

La raison pour laquelle les ordinateurs manipulent des données binaires est liée au fonctionnement de leurs composants physiques. Les transistors et les condensateurs, qui sont les éléments de base d'un ordinateur, possèdent deux états stables : activé/désactivé ou chargé/déchargé. Ainsi, un transistor dans l'état activé va stocker l'information 1 (ou 0 s'il est dans l'état désactivé).

L'objectif de ce document est de prendre connaissance des notions de base du codage de l'information, ainsi que des connaissances sur la théorie formelle basée sur l'Algèbre de Boole pour la synthèse des circuits.

Ce travail est destiné aux étudiants de première année licence.

Chapitre 1

Codification et représentation des nombres

1 Objectifs

- Savoir définir la base d'un système de numération.
- Savoir définir le rang et le poids d'un chiffre.
- Savoir représenter un nombre sous forme polynomiale.
- Savoir déterminer la valeur décimale d'un nombre de base b quelconque et vice versa.
- Savoir convertir un nombre binaire en un nombre octal ou en hexadécimal et vice versa.
- Savoir effectuer les opérations arithmétiques directement dans le système binaire, octal et hexadécimal.
- Comment représenter les nombres négatifs dans la machine.
- Comment représenter les nombres réels dans la machine.

2 Introduction

Les systèmes numériques complexes tels que les calculateurs doivent traiter toute sorte d'informations. A cette fin, ces informations doivent être codées à l'aide des chaines binaires. Ce chapitre décrit les notions fondamentales du codage de l'information utilisé par les systèmes numériques ainsi que les opérations arithmétiques réalisées sur ces codes.

3 Les systèmes de numération

Le système de numération décrit la façon avec laquelle les nombres sont représentés ; et il est définit par :

- Un alphabet: ensemble de symboles ou chiffres,
- Des règles d'écritures des nombres: Juxtaposition de symboles.

Il existe plusieurs systèmes de numérations dont les plus connus sont [1]:

- Le système décimal (b=10) qui est utilisé et pratiqué dans notre vie quotidienne. Ce système utilise dix chiffres: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.
- Le système binaire (b=2) qui est utilisé par les ordinateurs. Ce système utilise deux chiffres {0, 1}. Par convention on identifie 0 comme une absence de tension, et 1 comme une présence de tension. L'ordinateur comprend donc uniquement des nombres en base 2. Exemple : 101101 , 01100110, 11111111, 10000001. Ces 0 et 1 sont appelés bit qui est une abréviation de **bi**nary digi**t**.

- Le système octal (b=8=2^3) qui permet de coder trois bits par un seul symbole. Ce système utilise huit chiffres: {0, 1, 2, 3, 4, 5, 6, 7}.
- Le système hexadécimal (b=16) est utilisé pour réduire encore plus l'écriture des nombres binaire. La base hexadécimale est aussi une puissance de 2 (16 = 2^4). qui permet de coder quatre bits par un seul symbole. Ce système utilise size chiffres: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

4 Les entiers positifs

4.1 Forme polynomiale

On peut décomposer tout nombre N en fonction de puissances entières de la base de son système de numération. On notera en indice la base du système de numération dans lequel le nombre N envisagé est écrit (10 dans l'exemple ci-dessous) [2].
Considérons par exemple, le nombre décimal 1234, On aura :

$$(1234)_{(10)} = 1x10^3 + 2x10^2 + 3x10^1 + 4x10^0$$

Le chiffre de droite (4 dans l'exemple) s'appelle le chiffre de poids faible. Celui de gauche (1 dans l'exemple) s'appelle le chiffre de poids fort. L'exposant de la base, associé à un chiffre d'un nombre quelconque, s'appelle le **rang**. Par exemple 4 est de rang 0 tandis que 1 est de rang 3 dans l'exemple ci dessus. On peut généraliser cette notion et écrire sous forme polynomiale tout nombre N de base b quelconque.

On aura :

$$N = \sum_{i=0}^{i=n} a_i \times b^i \qquad (1)$$

Où a_i est un chiffre qui appartient à la base b tel que $0 \leq a_i < b$. i est le rang du chiffre a_i et n est l'exposant de b du chiffre de poids fort.

4.2 Changement de base (transcodage)

Le transcodage (ou conversion de base) est l'opération qui permet de passer de la représentation d'un nombre exprimé dans une base (b) à la représentation du même nombre mais exprimé dans une autre base (b').

4.2.1 Conversion d'un nombre de base quelconque en un nombre de base décimale

Elle s'obtient par la forme polynomiale vue au paragraphe précédent.

Exemple : donner la valeur décimale du nombre binaire N=1010
Solution :

$$(1010)_{(2)} = \sum_{i=0}^{i=3} a_i \times 2^i = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 8 + 0 + 2 + 0 = 10_{(10)}$$

$a_3\ a_2\ a_1\ a_0$

4.2.2 Conversion d'un nombre de base décimale en un nombre de base quelconque

Soit N un nombre écrit en base décimale. Si on veut le convertir dans une autre base b, il faut donc que l'égalité donné par l'équation 1 soit vérifiée. Le problème revient donc à déterminer les valeurs de a_i. Pour cela deux méthodes sont possibles :

1ère méthode : la méthode des puissances qui consiste à chercher les différentes puissances entières de la base b. l'algorithme commence par chercher la plus grande puissance entière de b contenue dans N, retrancher ensuite cette quantité du nombre N, recommencer ce processus en considérant le reste obtenu.

Exemple : convertir le nombre $N=75_{(10)}$ en nombre Octal.

Solution

On aura successivement :

75 - 64 ⟶ 1×8^2

11 - 8 ⟶ 1×8^1

3 ⟶ 3×8^0

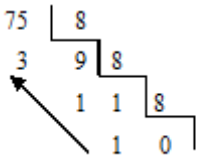
Donc $N=75_{(10)}= 1 \times 8^2 + 1 \times 8^1 + 3 \times 8^0$

On a donc : $N=75_{(10)} =113_{(8)}$

i	8^i
0	1
1	8
2	64

2ème méthode (L'algorithme d'Euclide): cette méthode est simple et plus rapide que la précédente. Elle consiste à faire des divisions successives par la base b jusqu'à ce que le quotient soit égal à zéro. On écrit ensuite tous les restes à partir de la fin et de gauche à droite, en les convertissant en lettres s'il y a lieu (dans le cas hexadécimale par exemple).

Pour le nombre précédant, on obtient :



On a donc : $N=75_{(10)} =113_{(8)}$

5 Les nombres fractionnaires

On écrira un nombre fractionnaire Nf inferieur à 1 sous la forme :

$$Nf = \sum_{i=1}^{i=n} a_i \times b^{-i}$$

$0 \leq a_i < b$, i est le rang du chiffre a_i et n est l'exposant du chiffre de poids faible.

Problème 1: convertir un nombre fractionnaire de base b en décimal.
Exemple : convertir $Nf=0,1011_{(2)}$ en décimal.

Solution : nous obtenons :
$Nf=1x2^{-1} + 0x2^{-2} + 1x2^{-3}+ 1x2^{-4} =0.5+0.125+0.0625=0.6875_{(10)}$

Problème 2 : convertir un nombre décimal fractionnaire en un nombre de base b.
Pour convertir un nombre décimal fractionnaire en un nombre de base b, il faut multiplier sa partie fractionnaire par la base. La partie entière du résultat est encadré, et la partie fractionnaire résultante est multipliée par la base. On recommence ce processus jusqu'à ce qu'un de ces critères soit vérifié :

1. Le résultat de la multiplication est un nombre entier
2. Après une certaine précision (nombre de chiffres après la virgule)

Exemple : convertir $Nf=0,85_{(10)}$ en binaire, prendre 04 chiffres après la virgule.

Solution : nous obtenons :
0.85 x 2 = 1,70
0.70 x 2 = 1.4
0.4 x 2 = 0.8
0.8 x 2= 1.6

On écrit de gauche à droite les nombres encadrés pris de haut en bas.
On aura donc $Nf=0,85_{(10)} = 0,1101_{(2)}$

6 Les conversions directes (Conversions par paquets)

6.1 Conversion binaire-octal et vice versa

Correspondance	
Octal	Binaire
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Tableau 1 : Correspondance entre le système octal et le système binaire

Le système octal (b=8) permet de coder trois bits par un seul chiffre. Ce système utilise les huit chiffres suivants: {0, 1, 2, 3, 4, 5, 6, 7}.Pour convertir un nombre fractionnaire binaire en octal, il faut juste grouper les bits par blocs de trois à partir de la virgule, en allant vers la gauche pour la partie entière et vers la droite pour la partie fractionnaire. Convertir ensuite ces blocs en octal en se basant sur le tableau 1 suivant :

Exemple 1 :
Convertir le nombre binaire N=11010101,11(2) en Octal.

Solution :

N= 011 010 101 , 110 $_{(2)}$
3 2 5 , 6 $_{(8)}$

Exemple 2 :
Convertir le nombre octal N=657,12$_{(8)}$ en binaire.

Solution :
Ecrire par blocs de trois bits, la valeur binaire des chiffres du nombre octal. On obtient :

6 5 7 , 1 2 $_{(8)}$
110 101 111 , 001 010$_{(2)}$

6.2 Conversion binaire-hexadécimal et vice versa

Le système hexadécimal (b=16) permet de coder quatre bits par un seul symbole. Ce système utilise les dix chiffres de la base décimale: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, plus, les six symboles suivants : {A, B, C, D, E, F}. Pour convertir un nombre fractionnaire binaire en Hexadécimal, il faut juste grouper les bits par blocs de quatre à partir de la virgule, en allant vers la gauche pour la partie entière et vers la droite pour la partie fractionnaire. Convertir ensuite ces blocs en Hexadécimal en se basant sur le tableau 2 :

Correspondance			
Hexa	Binaire	Hexa	Binaire
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

Tableau 2: Correspondance entre le système Hexadécimal et le système binaire

Exemple1 :
Convertir le nombre binaire N=11010101101,001(2) en Octal.

Solution :

N= 0110 1010 1101 , 0010 $_{(2)}$
6 A D , 6 $_{(16)}$

Exemple 2 :
Convertir le nombre Hexadécimal N=AB0,C1$_{(16)}$ en binaire.

Solution
Ecrire par blocs de quatre bits, la valeur binaire des chiffres du nombre hexadécimal. On obtient :

A B 0 , C 1 $_{(16)}$
1010 1011 0000 , 1100 0001 $_{(2)}$

7 Les opérations arithmétiques en binaire :

7.1 L'addition

L'algorithme de l'addition des nombres binaires est le même que celui de l'addition des nombres décimaux [3].

Dressons la table d'addition
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 et retenue de 1

Exemple :

```
    1  1
 10001101
+    1001
---------
=10010110
```

Remarque : 1 + 1 + 1 = 1 et retenue de 1.

7.2 La soustraction

Dressons la table de soustraction

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1 avec retenue

Comme dans le cas du système décimal, la soustraction en binaire est également basée sur la notion de retenue

Exemple

```
      1 1 0 1 1 0
-       1 1 0 0
  ---------------
=       0 1 0 1 0
```

7.3 La multiplication

Toujours comme dans le cas du système décimal, chaque nombre de la ligne du bas sera distribué à la ligne du haut. Ensuite, chaque ligne sera sommée (ne pas oublier les retenues) et le résultat sera alors obtenu.

Exemple

```
                    1 0 1 0
x                 1 0 1 0 1 0
-------------------------------
                    0 0 0 0
+                 1 0 1 0 0
+               0 0 0 0 0 0
+             1 0 1 0 0 0 0
+           0 0 0 0 0 0 0 0
+         1 0 1 0 0 0 0 0 0
-------------------------------
=         1 1 0 1 0 0 1 0 0
```

7.4 La division

Pour la division en binaire, c'est le même algorithme classique de la division décimale enseigné dans les classes élémentaires. Elle s'effectue à l'aide de soustractions et de décalages, Les chiffres du quotient, plus simples, sont 0 ou 1 ; 1 lorsque le diviseur est plus petit que le dividende et 0 sinon.

Exemple

```
1 0 1 1   | 100
-1 0 0    | 10,11
 0 0 1 1
 - 0 0 0
     1 1 0
   -1 0 0
       1 0 0
     - 1 0 0
           0
```

8 Les opérations arithmétiques en Octal :

8.1 L'addition

Comme pour le système binaire, on applique les mêmes règles pour les nombres octaux. Toutefois, dans ce cas, on aura la retenue **«1»** à gauche à chaque fois que la somme dépasse la valeur **7** car $\mathbf{7_{(8)}+1_{(8)}=10_{(8)}}$**.** (Voir le tableau 3)

Pour additionner deux chiffres octaux, on cherche ces chiffres sur le tableau 3**:** un en début de colonne, l'autre en début de ligne ; la somme se trouve dans la case correspondant au croisement entre colonne et ligne [3]. Par exemple, si l'on doit additionner les chiffres octaux **7** et 3**,** on cherche la somme dans la case située au croisement de la colonne **7** et de la ligne **3.** Dans cette case, on trouve **12.** Ainsi, dans le système octal, on aura**: 7 + 5 = 14.**

Exemple d'addition en base octale:

$$\begin{array}{rrrr} & {}^{1}\mathbf{1} & {}^{1}\mathbf{2} & \mathbf{3_{(8)}} \\ + & \mathbf{4} & \mathbf{5} & \mathbf{7_{(8)}} \\ \hline & \mathbf{6} & \mathbf{0} & \mathbf{2_{(8)}} \end{array}$$

±	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Tableau 3 : Tableau d'addition et de soustraction dans le système octal

8.2 La soustraction

Comme en décimal mais en se limitant à 7 (octal). Il faut juste apprendre de nouvelles tables. Pour faire la soustraction de deux chiffres octaux, on cherche ces chiffres sur le tableau 3**:** le diminuende dans la case correspondant au croisement entre colonne et ligne, le diminuteur en début de colonne (ou en début de ligne), le résultat de soustraction se trouve en début de ligne (ou en début de colonne).

Exemple de soustraction octale:

	$^{1}1$	$^{1}2$	$_{1}3_{(8)}$
-	$_{1}4$	$_{1}5$	$7_{(8)}$
	4	**4**	$4_{(8)}$

9 Les opérations arithmétiques en Hexadécimal :

9.1 L'addition

Comme en décimal, l'addition s'effectue chiffre par chiffre. Toutefois, dans ce cas, on aura la retenue **«1»** à gauche à chaque fois que la somme dépasse la valeur **F** car $F_{16)}+1_{16)}=10_{(16)}$**.** (Voir le tableau 4)

Exemple d'addition Hexadécimale:

	1	$^{1}2$	$A_{(16)}$
+	**E**	**5**	$7_{(16)}$
	F	**8**	$1_{(16)}$

9.2 La soustraction

En se basant le tableau 4, la soustraction de deux chiffres hexadécimaux, se fait en cherchant ces chiffres sur le tableau**:** le diminuende dans la case correspondant au croisement entre colonne et ligne, le diminuteur en début de colonne (ou en début de ligne), le résultat de soustraction se trouve en début de ligne (ou en début de colonne).

Exemple de soustraction Hexadécimale**:**

	F	$^{1}2$	$A_{(16)}$
-	$_{1}E$	**5**	$7_{(16)}$
	0	**D**	$3_{(16)}$

±	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	20
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	20	21
D	D	E	F	10	11	12	13	14	15	16	17	18	19	20	21	22
E	E	F	10	11	12	13	14	15	16	17	18	19	20	21	22	23
F	F	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

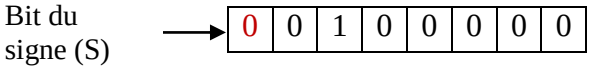
Tableau 4 : Tableau d'addition et de soustraction dans le système Hexadécimal

10 Les Entiers Négatifs

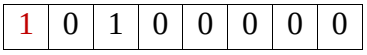
Les signe + et – ne sont pas reconnus par un ordinateur lequel ne connait que deux états : 0 et 1. On les représente donc par un bit qui occupera la case de gauche du nombre considéré. Ce bit s'appelle le bit de signe. Donc, par convention, on représente le signe + par 0 et signe – par 1. Les nombre négatifs sont représentés en machine par une des trois méthodes :**Signe et Valeur absolue**, **en complément à 1** ou **en complément à 2.**

10.1 Représentation des nombres négatifs en SVA (signe et valeur absolue)

C'est la représentation la plus simple d'un nombre négatif, il suffit de coder sa valeur absolue en binaire puis rajouter le bit du signe. Ainsi le nombre +32 est représenté sur 8 bits par :



Le nombre -32 s'écrira en SVA par :



Question :

Peut-on représenter le nombre -8 sur 04 bits.

Réponse :

Il est impossible de représenter le chiffre -8 sur 4 bits car sa valeur absolue $|-8_{(10)}|$ qui est égale à $1000_{(2)}$ prends déjà 04 bits et donc on aura besoin au minimum de 5 bits pour pouvoir représenter son bit de signe.

S	VA			Chiffre En décimal
0	0	0	0	+0
0	0	0	1	+1
0	0	1	0	+2
0	0	1	1	+3
0	1	0	0	+4
0	1	0	1	+5
0	1	1	0	+6
0	1	1	1	+7

S	VA			chiffre En décimal
1	0	0	0	-0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Tableau 5 : Représentation des nombres par la méthode Signe et Valeur Absolue

Question :
Quels sont les nombres qu'on peut représenter sur 04 bits ?

Réponse : d'après le tableau 5, on peut représenter sur 4 bits, l'intervalle de nombres entiers : De $[-(2^3 - 1) , (2^3 - 1)]$ soit de [-7 , +7].

Plus généralement, si on travaille sur n bits, l'intervalle des valeurs qu'on peut représenter en SVA est : $[-(2^{n-1} - 1), +(2^{n-1} - 1)]$.

Cette méthode présente deux inconvénients :

- Le zéro possède deux (2) représentations distinctes 0000 et 1000 soit +0 et -0;
- Les tables d'additions et de multiplication sont compliquées, à cause du bit de signe qui doit être traité à part.

10.2 Représentation des nombres négatifs en CP1 (Complément à 1)

En binaire, on forme le complément à 1 (CP1) d'un nombre en soustrayant de 1chaque bit de ce nombre. Donc pour obtenir le complément a 1 d'un nombre binaire, il suffit de complémenter (ou d'inverser) chaque bit. Le 1 devient 0 et le 0 devient 1.

Exemple :

$N= 1\,0\,0\,0\,1\,1\,1\,0_{(2)} \rightarrow 0\,1\,1\,1\,0\,0\,0\,1_{(2)}$

$N= 0\,0\,1\,1\,0_{(2)} \rightarrow 1\,1\,0\,0\,1_{(2)}$

Remarque :

- la somme d'un nombre binaire et de son complément à 1 est un nombre binaire composé uniquement de 1.

- Le bit de poids fort est utilisé pour représenter le signe du nombre :

- Si ce bit = 1 alors il s'agit d'un nombre négatif
- Si ce bit = 0 alors le nombre est positif.

Question :
Quelle est la valeur décimale du nombre binaire suivant : 10110110 ?

Réponse :
Le bit de poids fort indique qu'il s'agit d'un nombre négatif. Donc la Valeur décimale = - CP1(10110110) = - $(01001001)_2$= - $(73)_{10}$

Question :
Quels sont les nombres qu'on peut représenter sur 04 bits ?

Valeurs En CP1				Valeurs En binaire	chiffre En décimal
0	0	0	0	0000	+0
0	0	0	1	0001	+1
0	0	1	0	0010	+2
0	0	1	1	0011	+3
0	1	0	0	0100	+4
0	1	0	1	0101	+5
0	1	1	0	0110	+6
0	1	1	1	0111	+7

Valeurs En CP1				Valeurs En binaire	Chiffre En décimal
1	0	0	0	-0111	-7
1	0	0	1	-0110	-6
1	0	1	0	-0101	-5
1	0	1	1	-0100	-4
1	1	0	0	-0011	-3
1	1	0	1	-0010	-2
1	1	1	0	-0001	-1
1	1	1	1	-0000	-0

Tableau 6 : Représentation des nombres par la méthode CP1

D'après le tableau 6, on peut déduire que sur 4 bits :

- Le plus grand nombre positif représentable est donc 0111 ce qui représente $2^3 - 1$ soit +7
- Le plus petit négatif est -0111. Ce qui donne $-(2^3 - 1)$ soit -7

Réponse :
Donc, on constate que sur 04 bits, on peut représenter les nombres qui sont dans l'intervalle [$-7_{(10)}$,$+7_{(10)}$], soit [$-(2^3 - 1)$, $+(2^3 - 1)$]

Plus généralement, si on travaille sur n bits, l'intervalle des valeurs qu'on peut représenter en CP1 est : [$-(2^{n-1} - 1)$, $+(2^{n-1} - 1)$].

Limite de la méthode :
Même cas pour la méthode SVA, le zéro possède deux (2) représentations distinctes. Par exemple sur 8 bits $+0_{(10)}$=**$00000000_{(2)}$**, $-0_{(10)}$= cp1(00000000)= **$11111111_{(2)}$**

10.3 Représentation des nombres négatifs en CP2 (Complément à 2)

Il existe trois méthodes pour calculer le complément à 2 (CP2) d'un nombre binaire.

10.3.1 Première méthode :

La première consiste à le soustraire à la puissance de 2 immédiatement supérieure. Par exemple : trouver le complément à 2 du nombre N= $1\ 0\ 0\ 0\ 1\ 1\ 1\ 0_{(2)}$,

```
1 0 0 0 0 0 0 0 0
- 1 0 0 0 1 1 1 0
-----------------
  0 1 1 1 0 0 1 0
```

Donc CP2(N) = 0 1 1 1 0 0 1 0

10.3.2 Deuxième méthode :

Elle consiste à trouver d'abord le complément à 1 et à ajouter 1 au résultat. Par exemple :

N= $1\ 0\ 0\ 0\ 1\ 1\ 1\ 0_{(2)}$, CP1(N) = 0 1 1 1 0 0 0 1

```
0 1 1 1 0 0 0 1
+             1
---------------
0 1 1 1 0 0 1 0
```

Donc CP2(N) = 0 1 1 1 0 0 1 0

10.3.3 Troisième méthode

Consiste à conserver tous les bits à partir de la droite jusqu'au premier 1 compris et de changer les autres bits de 1 à 0 ou de 0 à 1. Pour l'exemple précédent, on obtient :

CP2(1 0 0 0 1 1 1 0)= 0 1 1 1 0 0 **1 0** → **bits conservés**

Question :

Quelle est la valeur décimale du nombre binaire suivant : 10110110 ?

Réponse :

Le bit de poids fort indique qu'il s'agit d'un nombre négatif. Donc la Valeur décimale = - CP1(10110110) = - (01001010)2= - (74)10

Question :

Quels sont les nombres qu'on peut représenter sur 04 bits ?

Valeurs En CP2				Valeurs En binaire	chiffre En décimal
0	0	0	0	0000	+0
0	0	0	1	0001	+1
0	0	1	0	0010	+2
0	0	1	1	0011	+3
0	1	0	0	0100	+4
0	1	0	1	0101	+5
0	1	1	0	0110	+6
0	1	1	1	0111	+7

Valeurs En CP2				Valeurs En binaire	Chiffre En décimal
1	0	0	0	-1000	-8
1	0	0	1	-0111	-7
1	0	1	0	-0110	-6
1	0	1	1	-0101	-5
1	1	0	0	-0100	-4
1	1	0	1	-0011	-3
1	1	1	0	-0010	-2
1	1	1	1	-0001	-1

Tableau 7 : Représentation des nombres par la méthode CP2

Sachant que le bit du poids fort est utilisé pour représenter le signe du nombre, on peut déduire que sur 4 bits :

- Le plus grand nombre positif représentable est donc 0111 ce qui représente $2^3 - 1$ soit +7
- Le plus petit négatif est codé par 1000, ce qui donne la valeur binaire -1000, soit $-8(-2^3)$ en décimal.

Réponse :
Donc, d'après le tableau 7, on constate que sur 04 bits, on peut représenter les nombres qui sont dans l'intervalle $[-8_{(10)},+7_{(10)}]$, soit $[-2^3 , +(2^3 - 1)]$

Plus généralement, si on travaille sur n bits, l'intervalle des valeurs qu'on peut représenter en CP2 est : $[-(2^{n-1}), +(2^{n-1} - 1)]$.

Avantage :
Un seul codage pour le nombre 0. Par exemple sur 8 bits :
$+0_{(10)} = 00000000_{(2)}$, $-0_{(10)}=cp2(00000000)=00000000_{(2)}$

10.4 Soustraction en base 2 en utilisant les compléments

Avant de faire une soustraction en complément à 1 ou 2 il faut s'assurer que le diminuteur et le diminuende ont le même nombre de bits. Cette remarque est très importante sinon on ne retrouve pas le résultat de la soustraction.

10.4.1 Soustraction par complémentation à 1.

- La soustraction par complément à 1 revient à calculer le complément à 1 du diminuteur ensuite l'ajouter au diminuende. La dernière retenue est ajoutée au résultat.

- Lorsque le dernier bit du résultat (poids fort)=1, cela veux dire que le résultat est négatif. On calcule donc le complément à 1 de ce dernier afin d'obtenir le résultat final.

Exemple 1:
Effectuons sur 5 bits, l'opération (+8) + (- 9).

Solution:
Les nombres doivent être sur 5 bits y compris le bit de signe

$(+8) = 01000_{(2)}$
$(+9) = 01001_{(2)}$

Le complément à 1 de 01001 est 10110 = - $9_{(10)}$

```
  0 1 0 0 0   <-- diminuende
+ 1 0 1 1 0   <-- diminuteur
= 1 1 1 1 0
```

Le bit du signe =1 → résultat négatif → le résultat = - Cp1(11110).
Dans ce cas, calculons le complément à 1 du résultat

Cp1(11110)=$00001_{(2)}$. Cela veux dire que le résultat = $-1_{(2)}$

Exemple 2: toujours sur 8 bits effectuons l'opération : (-8) + (-9)

Solution :
Dans ce cas chaque nombre est représenté par son complément à 1 :
(+8) = 01000, (-8)= cp1(01000)=10111
(+9) = 01001, (-9)= cp1(01001)=10110

```
  1 1 0 1 1 1
    1 0 1 1 0
  ---------
    0 1 1 0 1
+               1
  ---------
    0 1 1 1 0
```

Remarque 1:
Nous remarquons que le dernier bit du résultat = zero. Ce qui veux dire que le résultat est positif. Hors, l'addition de deux nombres négatifs ne peut donner qu'un nombre négatif !!!. Ce cas s'appelle « dépassement » , en anglais, « overflow ». Il résulte du fait que sur 5 bits on ne peux représenter que les nombres qui sont dans l'intervalle [-15 , 15] (voir la section 8.2) alors que -8-9 donne -17 qui est hors intervalle.

Remarque 2
Dans l'addition arithmétique signée, par exemple (A+B), on dit qu'il y a débordement si et seulement si les deux opérandes A et B sont de même signe et le résultat S est de signe différent.

10.4.2 Soustraction par complément à 2

- Complémenter le diminuteur ensuite l'ajouter au diminuende. La dernière retenue est ignorée.
- Tout comme la méthode du complément à 1, lorsque le dernier bit du résultat (poids fort)=1, cela veux dire que le résultat est négatif. On calcule donc le complément à 2 de ce dernier afin d'obtenir le résultat final.

Exemple 3:
Effectuons la même opération en complément à 2. (+8) - (+9).

Solution
(+8) = $01000_{(2)}$, (+9) = $01001_{(2)}$
Le complément à 2 de 01001 est 10111 = -9

```
   0 1 0 0 0
+  1 0 1 1 1
-------------
=  1 1 1 1 1
```

Le bit du signe =1 → résultat négatif → le résultat = - Cp2(11111).

Dans ce cas, calculons le complément à 1 du résultat :
Cp2(11111)=00001(2). Cela veux dire que le résultat = -1(2)

11 Les Nombres Réels

11.1 Représentation en virgule fixe

Soit un nombre N tel que N= - $1010{,}1001_{(2)}$. Le codage du nombre en virgule fixe consiste à définir la position de la virgule selon un format donné, c'est-à-dire la taille de la partie entière ainsi que la taille de partie fractionnaire. Les bits à gauche de la virgule représentent la partie entière signée du nombre tandis que la partie droite représente la parie fractionnaire. Dans l'exemple ci-dessous (tableau 8), la partie entière signée est sur 8 bits et la partie fractionnaire sur 8 bits.

1	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0
signe	Partie entière								Partie fractionnaire							

Tableau 8 : Représentation d'un nombre réel par la méthode de la virgule fixe

Exemple :

1. Représenter le nombre réel (-6.125) en format virgule fixe (1 bit de signe, 8 bit pour la partie entière et 7 bits pour la partie fractionnaire.
2. Quelle est le plus petit nombre positif représentable dans ce format.
3. Quelle est le plus grand nombre positif représentable dans le même format.

Solution :

1. Il faut d'abord convertir le nombre en binaire pour pouvoir le représenter en machine. On a : $-6.125_{(10)} = -110{,}001_{(2)}$.On représente donc le nombre selon le format indiqué. On obtient donc : 1|00000110|0010000
2. le plus petit nombre positif est représenté comme suit : 0|00000000|0000001 ce qui donne la valeur $Nmin=2^{-7}$
3. le plus grand nombre positif est représenté comme suit : 0|11111111|1111111

Pour donner l'équivalent en décimale, calculons la partie entière max (PEmax) et la partie fractionnaire max (PFmax).

$PEmax=2^0+2^1+\ldots+2^7= 2^8-1$
$PFmax =2^{-1}+2^{-2}+\ldots+2^{-7}=1- 2^{-7}$
$Nmax=PE\ max=+PF\ max=2^8-1 +1- 2^{-7}= 2^8- 2^{-7}$

11.2 Représentation en virgule flottante

Il existe plusieurs formats de représentation en virgule flottante proposés par l'IEEE (Institute of Electrical and Electronics Engineers) et qui ont été adoptés par les fabricants de microprocesseurs, parmi ces formats, on cite :

- Le format simple précision, utilisant 32 bits.
- Le format double précision, utilisant 64 bits.
- Les formats étendus à simple et à double précision.

Le lecteur intéressé pourra se reporter aux nombreuses références sur le sujet [1-3]. En ce qui suit, nous allons décrire les différentes méthodes de représentation des nombres réels en virgule flottante.

11.2.1 Par la méthode de l'exposant réel.

Il existe deux méthodes pour représenter les nombres réels en virgule flottante. La première consiste à représenter un nombre N par un bit de signe (0 si N est positif, 1 sinon), une mantisse M (qui doit être normalisée) et un exposant Exp (entier positif ou négatif). Le nombre N est donc écrit sous la forme suivante :

$$N= \pm|M| \times 2^{Exp} \quad \text{avec} \quad 0.1_{(2)} \leq |M| < 1_{(2)}$$

Selon le type de machine, un certain nombre de bit est réservé pour la mantisse ainsi que pour l'exposant.

Exemple1 : Représenter N= - $1010,1001_{(2)}$ sur 16 bits en format virgule flottante (12 bits pour la mantisse, 4 bits pour l'exposant et 1 bit pour le signe de la mantisse).

Solution :
On commence par normaliser la mantisse :
N= - $1010,1001_{(2)}$ =-$0.10101001_{(2)}$ x 2^{+4}. (+4 représente le nombre de déplacement de la virgule vers la gauche).

Dans ce cas : M= - $0.10101001_{(2)}$ et Exp= + $4_{(10)}$ =$0100_{(2)}$

1	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0
Signe	Exp				Mantisse											

Tableau 9 : Représentation d'un nombre réel par la méthode de la virgule flottante –exposant positif-

Exemple2 : représenter N= - $0,001001_{(2)}$ sur 16 bits en format virgule flottante (12 bits pour la mantisse, 4 bits pour l'exposant et 1 bit pour le signe de la mantisse).

Solution :
On commence par normaliser la mantisse :
N= - $0,001001_{(2)}$ =- $0.1001_{(2)}$ x 2^{-2}. (-2 représente le nombre de déplacement de la virgule vers la droite).
Dans ce cas : M=- $0.1001_{(2)}$ et Exp=-$2_{(10)}$

En virgule flottante, les exposants négatifs sont représentés par la méthode du complément à 2 (voir la section 8.3).

|Exp|= +$2_{(10)}$ =$0100_{(2)}$
Exp=-2=CP2(0100) =$1100_{(2)}$

1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
Signe (1 bit)	Exp (4 bits)				Mantisse (12 bits)											

Tableau 10 : Représentation d'un nombre réel par la méthode de la virgule flottante –exposant négatif-

Remarque : le bit le plus à gauche de l'exposant représente le bit du signe.

11.2.2 Par la méthode de l'exposant Biaisé.

La deuxième méthode de la représentation des nombres réels en format virgule flottante consiste toujours à représenter le nombre N sous la forme : N= ±|M| x 2^{Exp_biaise} tel que $0.1_{(2)} \leq |M| < 1_{(2)}$. Mais cette fois-ci, il ne s'agit pas de représenter l'exposant réel mais un autre exposant dit baisé. Ce dernier est calculé en fonction de l'exposant réel comme suit :

$$\textbf{\textit{Exp\_biaisé = Exp + biais tel que biais} = 2^n/2 = 2^{(n-1)}.}$$

n est le nombre de bit de l'exposant.
La valeur du biais ajoutée, rend la valeur de l'exposant biaisé toujours positif.

Exemple : Prenons le même exemple précédent.
$N = -0{,}001001_{((2)} = -0.1001_{(2)} \times 2^{-2}$
$M = -0.1001_{(2)}$ et $Exp = -2_{(10)}$

Calculons maintenant la valeur de l'exposant biaisé :

$$Exp\_biaisé = -2 + (2^4/2) = 2 + 8 = 10_{(10)} = 1010_{(2)}$$

Ce nombre est représenté comme suit dans le format suivant : 12 bits pour la mantisse, 4 bits pour l'exposant biaisé et 1 bit pour le signe de la mantisse.

1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0
Signe (1 bit)	Exp_biaisé (4 bits)				Mantisse (12 bits)											

Tableau 11 : Représentation d'un nombre réel par la méthode de la virgule flottante –exposant biaisé-

Remarque : *Exp_biaisé* est toujours positif donc à l'inverse du cas précédent, le bit le plus à gauche de l'exposant biaisé ne représente pas le bit du signe mais il fait parti de la valeur.

Démonstration

On a déjà démontré dans la section 8.3 que dans méthode du complément à 2, l'intervalle des valeurs qu'on peut représenter sur n bits est:

$$-2^{(n-1)} \leq N \leq 2^{(n-1)} - 1$$

Si on rajoute la valeur $2^{(n-1)}$ à tous les termes de cette inégalité, on obtient :

$$-2^{(n-1)} + 2^{(n-1)} \leq N + 2^{(n-1)} \leq 2^{(n-1)} - 1 + 2^{(n-1)} = 0 \leq N + 2^{(n-1)} \leq 2^{n-1}$$

• On pose $N' = N + 2^{(n-1)}$, on aura donc : $0 \leq N' \leq 2^{n-1}$. Dans ce cas on obtient que des valeurs positives. La valeur 2^{n-1} s'appelle le biais ou le décalage.

Remarque :

Si la mantisse est sur k bits et si elle est représenté sur la machine sur k' bits tel que k> k' , alors la mantisse sera tronquée : on va prendre uniquement k' bits.

11.3 Arithmétique

Soit deux nombres réels N1 et N2 tel que : $N1 = M_1 * 2^{exp1}$ et $N2 = M_2 * 2^{exp2}$. On veut calculer N1+N2 ? Deux cas se présentent :

- Si exp1 = exp2 alors $N3 = (M_1 + M_2)\, b^{exp1}$

- Si exp1 ≠ exp2 alors élever au plus grand exposant et faire l'addition des mantisses et par la suite normaliser la mantisse du résultat.

Exemple : soient les deux nombres suivants :

$$N1=0,001101(2) \quad , \quad N2=1,101(2)$$

Représenter N1+N2 en format virgule flottante selon le format suivant : (12 bits pour la mantisse, 4 bits pour l'exposant et 1 bit pour le signe de la mantisse).

Solution

On normalise d'abord les mantisses pour cela :

$N1=0,1101 *2^{-2}$, $N2=0,1101 *2^{1}$

$N1+N2= 0,1101 *2^{-2} +0,1101 *2^{1}$

$= 0,0001101 *2^{1} + 0,1101 *2^{1}$

$= 0, 1110101 * 2^{1}$

Dans ce cas, N1+N2 est représenté comme suit :

0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0
Signe(1 bit)	Exposant (4 bits)				Mantisse (12 bits)											

Tableau 12 : Addition de deux nombres réels en virgule flottante

La soustraction est faite de la même manière.

12 Exercices :

1. Convertir $94_{(10)}$ en binaire puis compléter le tableau suivant

Chiffres du nombre en base 2								
Rang du chiffre								
Poids de chiffre								
Valeurs								

2. Déterminer x et y tel que : $(3x2, y3)_6 = (134,25)_{10}$
3. Déterminer la base B sachant que $(25)_B=(10111)_2$
4. Effectuer les conversions suivantes :
   - $18_{(10)} = N_{(2)} = N_{(8)} = N_{(16)}$
   - $1010,011_{(2)} = N_{(10)} = N_{(8)} = N_{(16)}$
   - $70C,A_{(16)} = N_{(4)} = N_{(8)} = N_{(2)} = N_{(10)}$
5. Effectuer, en binaire, les additions suivantes :
   - $7A0B_{(16)} + 56,4_{(8)}$
   - $143,12_{(5)} + 31,43_{(6)}$
6. Effectuer en octal : $326_{(8)} + 735_{(8)}$ et en hexadécimal : $3AD2_{(16)} + B0FF_{(16)}$.
7. Effectuer, en binaire, les soustractions suivantes :
   - $35_{(10)} - 25_{(10)}$
   - $5,25_{(10)} - 10,75_{(10)}$
   - $A0137_{(16)} - 33_{(4)}$

8. Effectuer en octal : 524(8) – 263(8) et en hexadécimal : 5E2(16) – 3DA(16).
9. Effectuer, en binaire les multiplications suivantes :
   - $23_{(10)} \times 29_{(10)} = N_{(2)}$
   - $2436_{(8)} \times 37_{(8)} = N_{(2)}$
   - $20{,}7_{(10)} \times 7{,}5_{(10)} = N_{(2)}$
10. Effectuer, en binaire les divisions suivantes :
    - $35_{(10)} : 7_{(10)} = N_{(2)}$
    - $237{,}5625_{(10)} : 71_{(8)} = N_{(2)}$
    - $10000111_{(2)} : 101_{(2)} = N_{(2)}$

11. Représenter sur 8 bits les nombres négatifs suivants par les 3 méthodes : Signe et valeur absolue, complément à 1 et complément à 2.

    -45 ; -128 ; -12, -36

12. Représenter en décimal les nombres binaires qui sont sous le format complément à 2

    10110110 ; 11101000

13. Soient N1, N2 et N3 trois entiers signés représentés sur 08bits. Sachant que N1= 01101110(2), N2=01011001(2) et N3=01110011(2)
    - Effectuer les opérations suivantes en complément à 1 puis en complément à 2.

      N1-N2 ; N2-N3, -N1-N3.
    - Déterminer s'il y a un dépassement (overflow)

14. Représenter le nombre réel -6.125 suivant en format virgule fixe (1 bit de signe, 8 bit pour la partie entière et 7 bits pour la partie fractionnaire.
    - Quelle est le plus petit nombre positif représentable dans ce format.
    - Quelle est le plus grand nombre positif représentable dans le même format.

15. On suppose que les nombres réels sont représentés sur 32 bits en format virgule flottante (23 bits pour la mantisse, 8 bits pour l'exposant et 1 bit pour le signe de la mantisse). Quelle est la représentation binaire du nombre réel 0.015(8) représenté avec un exposant signé:

16. Donner la valeur décimale du nombre suivant codé en format virgule flottante (23 bits pour la mantisse, 8 bits pour l'exposant et 1 bit pour le signe de la mantisse). Puis trouver le plus petit et le plus grand nombre représentables dans ce format.

    46C86000(16)

17. Supposant qu'on utilise le même format de l'exercice 17. Donner la valeur décimale des représentations binaires suivantes

    1 11111010 11110110000000000000000

    1 00000100 11000000000000000000000

    - Calculer l'addition ensuite la soustraction de ces deux nombres puis représenter le résultat.

18. Soit le format à virgule flottante: mantisse=23 bits, exposant=8 bits, calculer l'exposant biaisé du nombre n=7(10) x 2(-3). Donner la représentation binaire du nombre réel suivant représenté dans le même format avec un exposant biaisé égale à -123.75.

13 Conclusion

Dans ce chapitre, nous avons présenté les systèmes de numération les plus utilisés dans les systèmes numériques notamment le système binaire, octal et hexadécimal. Aussi, nous avons étudié les opérations de codage, décodage, conversion de codes ainsi que la réalisation des opérations arithmétiques. Nous avons étudié aussi le codage ainsi que l'arithmétique dans les nombres réels. Dans le chapitre suivant nous allons étudier d'autres codes permettant la représentation des caractères alphanumériques.

Chapitre2

Codification et représentation a-Numériques

1 Objectifs

- Savoir coder et décoder un mot selon le code ASCII.
- Savoir coder un nombre décimal en BCD et vice versa.
- Savoir coder un nombre décimal ou BCD en code « BCD+3 » et vice versa.
- Savoir faire les additions en code BCD.
- Savoir coder un nombre binaire en code Gray et vice versa.
- Savoir comment la machine détecte et corrige une erreur de transmission.

2 Introduction

Nous avons vu dans le chapitre précédent comment sont codés les nombres par notre ordinateur. Nous savons donc comment est codé un nombre positif, négatif ainsi que les réels. Mais comment est représenté un texte dans un ordinateur ?une image ? un son ?, etc.. Il existe pour cela de multiples codes, nous verrons dans ce chapitre quelques types de codes existants.

3 Le code ASCII (American Standard Code for Information Interchange).

Un ordinateur ne serait pas d'une grande utilité s'il n'était pas capable de traiter l'information non numérique. On veut dire par là qu'un ordinateur doit reconnaître des codes qui correspondent à des nombres, des lettres et des caractères spéciaux. Les codes de ce genre sont dits alphanumériques. Le code ASCII a été créé pour représenter les caractères alphanumériques usuels de l'anglais. Il transforme tout symbole d'un alphabet contenant 87 caractères (les 10 chiffres décimaux, 26 lettres majuscules, 26 lettres minuscules et environ 25 caractères spéciaux par exemple +,-,*,/,$, etc.…) en un mot binaire de 7 bits (car $2^6 < 87 < 2^7$). Ce code est une norme presque universelle dans les transmissions entre machines qui s'effectuent souvent sur un format de 8 bits [4]. Le huitième bit est dit de parité sert à détecter les erreurs de transmission. Le tableau 13 illustre quelques exemples de mots du code ASCII.

Exemples de codes :

Le code binaire est donné par $b_6\, b_5\, b_4\, b_3\, b_2\, b_0$ (b_0 est le bit de poids faible)

- Le code de U (majuscule) est représenté par 1010101 en code Ascii, par 85 en code décimale et par 55 en code hexadécimal.
- u (minuscule) est $1110101_{(2)}$, soit $117_{(10)}$ soit $75_{(16)}$.
- Le code ascii 1000001 $=41_{(16)}=65_{(10)}$.
- $(9)_{10}$ = (011**1001**) en code ASCII.

Mais le problème est que cette table a été inventée par les Américains : les accents sont donc inexistants. Néanmoins, avec le développement des technologies de l'information, des représentations plus évoluées ont vu le jour, dont le code Unicode, comportant 16 bits, pour un total de 65536 mots possibles et permettant de représenter de nombreux caractères de différents alphabets utilisés un peu partout dans le monde.

Binaire						b6	0	0	0	0	1	1	1	1
						b5	0	0	1	1	0	0	1	1
						b4	0	1	0	1	0	1	0	1
				Hexadécimal			0	1	2	3	4	5	6	7
b3	b2	b1	b0		Décimal		0	16	32	48	64	80	96	112
0	0	0	0	0	+0		NUL	TC7 (DEL)	SP	0	@	P	`	p
0	0	0	1	1	+1		TC1 (SOH)	DC1	!	1	A	Q	a	q
0	0	1	0	2	+2		TC2 (STX)	DC2	"	2	B	R	b	r
0	0	1	1	3	+3		TC3 (ETX)	DC3	#	3	C	S	c	s
0	1	0	0	4	+4		TC4 (EOT)	DC4	$	4	D	T	d	t
0	1	0	1	5	+5		TC5 (ENQ)	TC8 (NAK)	%	5	E	U	e	u
0	1	1	0	6	+6		TC6 (ACK)	TC9 (SYN)	&	6	F	V	f	v
0	1	1	1	7	+7		BEL	TC10 (ETB)	'	7	G	W	g	w
1	0	0	0	8	+8		FE0 (BS)	CAN	(	8	H	X	h	x
1	0	0	1	9	+9		FE1 (HT)	EM	)	9	I	Y	i	y
1	0	1	0	A	+10		FE2 (LF)	SUB	*	:	J	Z	j	z
1	0	1	1	B	+11		FE3 (VT)	ESC	+	;	K	[	k	{
1	1	0	0	C	+12		FE4 (FF)	IS4 (FS)	,	<	L	\	l	|
1	1	0	1	D	+13		FE5 (CR)	IS3 (GS)	-	=	M	]	m	}
1	1	1	0	E	+14		SO	IS2 (RS)	.	>	N	^	n	~
1	1	1	1	F	+15		SI	IS1 (US)	/	?	O	_	o	DEL

Tableau 13 : Table des codes ASCII

4 Unicode

Le code UNICODE a été développé par le Consortium Unicode. Contrairement au code ASCII qui permet d'utiliser seulement les codes 0 à 127, UNICODE utilise des codes de valeurs bien plus grandes [4]. Il possède 16 bits de largeur et permet de coder 65536 caractères différents. Il permet de représenter les caractères spécifiques aux différentes langues : latins (accentués ou non), grecs, arabes, cyrillics, arméniens, hébreux, thaï, hiragana, katakana... L'alphabet Chinois Kanji comporte à lui seul 6879 caractères. L'Unicode est

accepté par les systèmes d'exploitation, applications et langages actuels tels que Windows, JAVA, etc. Il permet donc :

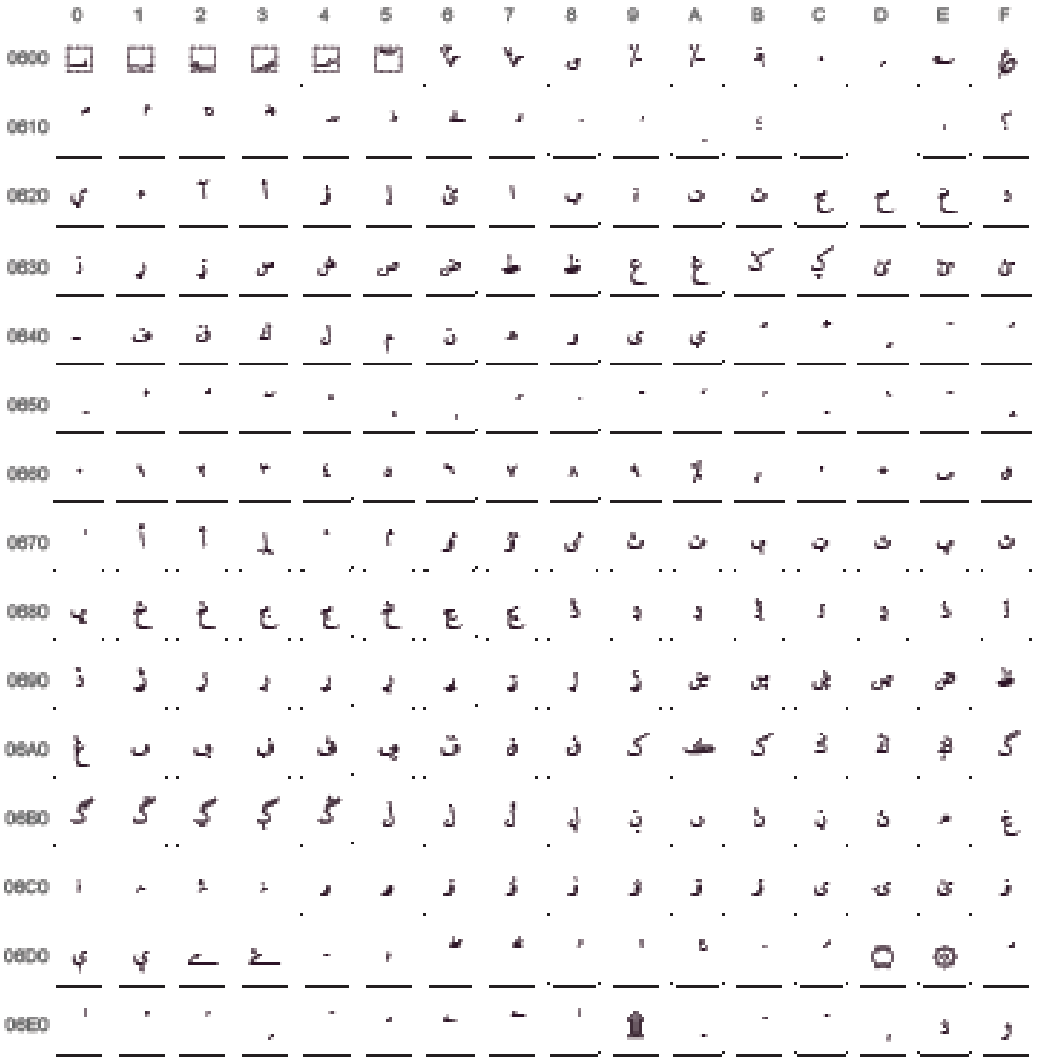
- D'unifier l'encodage de caractères, quelque soit le système d'exploitation,
- D'utiliser simultanément plusieurs alphabets et de multiples symboles dans un même document,

L'Unicode définie donc une correspondance entre symboles et nombres. Chaque caractère ou symbole est représenté par un code qui est noté U+xxxx où xxxx est en hexadécimal, et comporte 4 à 6 chiffres :

- 4 chiffres pour le plan multilingue de base (entre U+0000 et U+FFFF) ;
- 5 chiffres pour les 15 plans suivants (entre U+10000 et U+FFFFF) ;
- 6 chiffres pour le dernier plan (entre U+100000 et U+10FFFF).

Ainsi, le caractère nommé (*ƒ*) a un index de U+0192. Il appartient au premier plan.
256 caractères arabes sont codés par l'unicode grâce au bloc « Arabe », entre U+0600 à U+06FF. par exemple le caractère 'أ'est codé par U+0623.

Voici une toute petite partie des tables UNICODE destiné pour la langue arabe [4],[5]:



5 UTF-8.

L'UTF-8 rassemble le meilleur de deux mondes: l'efficacité de l'ASCII et l'étendue de l'Unicode. Généralement en UNICODE, un caractère prend 2 octets. Autrement dit, le moindre texte prend deux fois plus de place qu'en ASCII. Ce qui permet de coder 65 535 caractères. De plus, si on prend un texte en français, la grande majorité des caractères utilisent seulement le code ASCII. Seuls quelques rares caractères nécessitent L'UNICODE. D'où l'intérêt du codage UTF-8 [6].

Un texte en UTF-8 est simple: il est partout en ASCII, et dès qu'on a besoin d'un caractère appartenant à l'Unicode, on utilise un caractère spécial signalant "attention, le caractère suivant est en Unicode". D'ailleurs l'UTF-8 a été adopté comme norme pour l'encodage des fichiers XML. La plupart des navigateurs récents supportent également l'UTF-8 et le détectent automatiquement dans les pages HTML.

6 Le Code Barre

Ce principe de codage, apparu dans les années 80, est largement utilisé sur les produits de grande consommation, car il facilite la gestion des produits. Le marquage comporte un certain nombre de barres verticales ainsi que 13 chiffres :

- Le 1er chiffre désigne le pays d'origine : 3 = France, 4 = Allemagne, 0 = U.S.A, etc. …
- Les cinq suivants sont ceux du code « fabricant »,
- Les six autres sont ceux du code de l'article,
- Le dernier étant une clé de contrôle
- Les barres représentent le codage de ces chiffres sur 7 bits, à chaque chiffre est attribué un ensemble de 7 espaces blancs ou noirs (voir figure ci_dessous) [7],[8].

7 Le code BCD

Le code BCD, qui est l'abréviation de Binary Coded Decimal en anglais, est un code binaire permettant de convertir directement un nombre décimal en un nombre binaire [1], [2]. Il est utilisé par les machines à calculer. Pour coder un nombre décimal en BCD, on va coder séparément chaque chiffre du nombre de base dix en Binaire selon le tableau 14.

Décimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tableau 14 : Codage des chiffres décimaux par le code BCD

On appelle ce code aussi code 8421 car les 4 bits (du plus fort au plus faible) pondèrent les nombres 8, 4, 2 et 1 respectivement. Ainsi, 1001 correspond à 9 car 9 = **1** x 8 + **0**x 4 + **0** x 2 + **1** x 1. En BCD, chaque chiffre d'un nombre peut prendre une valeur entre 0 et 9 c'est-à-dire entre {0000 et 1001}. Un nombre de plusieurs décimaux est formé en groupant les quartets.

Exemple :
Coder le nombre décimal $804_{(10)}$ en code BCD

Solution
$8\ 0\ 4_{(10)} = 1000\ \ 0000\ \ 0100_{(BCD)}$.

Les opérations arithmétiques effectuées dans ce code sont plus compliquées qu'en binaire naturel.
Exemple : effectuer l'addition suivante en BCD :

$$153_{(10)} + 351_{(10)}$$

Solution
$1\ 5\ 3_{(10)} = 0001\ \ 0101\ \ 0011_{(BCD)}$
$3\ 5\ 1_{(10)} = 0011\ \ 0101\ \ 0001_{(BCD)}$

$$\begin{array}{r} 0001\ \ 0101\ \ 0011 \\ +\ \ 0011\ \ 0101\ \ 0001 \\ \hline 0100\ \ \boxed{1010}\ \ 0100 \end{array}$$

Nous rencontrons ici un mot codé qui ne correspond pas à une valeur BCD connue. Pour résoudre ce problème, on peut ajouter la valeur $(6)_{10} = (0110)_2$ à ce mot codé, ce qui donnera

$$\begin{array}{r} 1010 \\ 0110 \\ \hline 10000 \end{array}$$

Ajouter 1 (report) au nombre suivant, le résultat sera donc :

0101 0000 0100 $_{(BCD)}$.=5 0 4 $_{(10)}$

Cette difficulté provient du fait quatre bits donnent 2^4=16 bits dont 10 seulement servent à coder les chiffres décimaux.

Remarque *:*

- Le nombre codé en BCD ne correspond pas au nombre décimal converti en binaire naturel.
- Les combinaisons supérieures à 9 sont interdites. Par exemple la combinaison 1010 n'appartient pas au code BCD.
- Le codage décimal BCD est simple, mais il n'est pas possible de faire des opérations mathématiques directement dessus.
- Ce code est surtout utilisé pour l'affichage de données décimales. (dans les calculatrices par exemple).

8 Le code « Plus Trois » ou « BCD+3 »

L'inconvénient cité ci-dessus à propos des opérations arithmétiques en BCD a été contourné dans les premiers ordinateurs à l'aide du code « plus Trois »[1]. Comme son nom l'indique, ce code consiste à calculer pour chaque chiffre décimal (de 0 à 9) son code BCD, puis lui ajouter la valeur 3. Voir le tableau suivant :

Remarque :

L'addition de deux nombres en BCD+3, nous mène vers deux cas possibles :

- L'addition génère un report : dans ce cas ajouter la valeur 3 au résultat.
- L'addition ne génère pas un report : dans ce cas, retrancher 3 du résultat.

Retrancher la valeur 3 (-3) revient à ajouter la valeur 1101. Car (-3) = -0011= 1101 en complément à 2.

Exemple 1 : avec report

```
  8                      1011
+ 6                      1001
 --    En BCD +3 →   ---------
 14                  0001 0100
                     0011 0011
                     ---------
                     0100 0111 (BCD+3)
```

Exemple 2 : sans report

```
  3                        0110
+ 4                        0111
 --     En BCD +3 →   ---------
  7                        1101
                         + 1101
                        ---------
                        (1) 1010 (BCD+3)
Ignorer le débordement
```

Décimal	BCD
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

Tableau 15 : Codage des chiffres décimaux par le code BCD+3

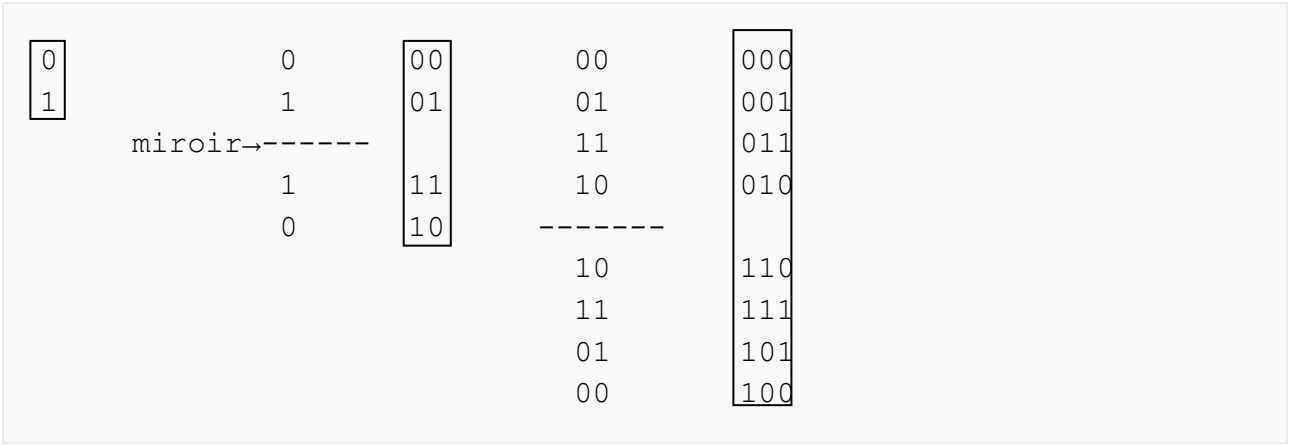
9 Le code Gray

Dans les conversions d'une grandeur analogique en une grandeur numérique, nous avons besoin d'un code dans lequel les grandeurs successives ne différent que d'un caractère. Cela évite les erreurs de transmission. Dans le système binaire naturelle, on peut remarquer que lors du passage de la valeur sept (7) à la valeur huit (8), c'est-à-dire le passage de 0111 à 1000, les quartes bits changent en même temps. S'il y a un problème de synchronisation, on peut détecter des valeurs erronées [1-3]. Le code de Gray est inventé afin d'éviter ce genre de problèmes. Il encode les entiers de telle façon que le passage d'un nombre au suivant ne change qu'un seul bit à la fois. On le retrouve également dans certains systèmes de télécommunication pour la correction d'erreur. Le tableau 15 donne l'équivalent en code Gray des entiers de 0 à15 ainsi que leur équivalent en binaire.

Décimal	Binaire	Gray	Décimal	Binaire	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Tableau 16 : Code Gray

Le nom du code gray ou binaire réfléchi veut dire que les bits du code Gray peuvent être générés par réflexion (comme une réflexion dans un miroir) comme l'illustre le tableau suivant, puis on rajoute un 0 puis un 1 au début (à gauche) de chacun des codes. On a ainsi doublé le nombre de codes formés.



9.1 Conversion du code binaire en code gray

Soit B un nombre écrit en binaire pur sur m bits

$$B_{(2)} = Bm \ldots\ldots B4B3B2B1$$

Bm est le bit du poids fort

G est l'équivalent en code Gray du nombre B écrit lui aussi sur m bits

$$G_{(Gray)} = Gm \ldots\ldots G4G3G2G1$$

Le passage du binaire pur au code Gray se fait en effectuant une opération OU Exclusif. En désignant par Bn un bit quelconque en code binaire pur et par Gn le bit recherché en code Gray, nous avons alors :

Bm=Gm

Pour tout n <m, nous avons :

$$G_n = B_n \oplus B_{n+1}$$

L'opération consiste donc à calculer le OU exclusif entre le binaire de départ et ce même binaire décalé d'un rang à droite.

Exemple :

Trouver l'équivalent en code Gray du nombre binaire B= $0111_{(2)}$

Solution :

```
   0111
⊕
   0011
  ------
   0100
```

B= $0111_{(2)}$ =$0100_{(gray)}$

9.2 Conversion du code gray en code binaire

Pour la conversion du code Gray en code binaire appliquons la relation suivante :

Bm=Gm

Pour tout n <m, nous avons :

$B_n = G_n \oplus B_{n+1}$

Donc pour obtenir le code binaire d'un code de Gray donné, on passe de gauche (poid fort) à droite (poids faible). Le bit de poids fort est le même en binaire que dans le code de Gray. Le bit binaire suivant reste le même si le bit de Gray suivant vaut zéro et change si le bit de Gray suivant vaut 1. On répète cela pour tous les bits, jusqu'au bit de poids le plus faible.

Exemple : Trouver l'équivalent en code binaire du nombre binaire en code gray $G=(11011)_{Gray}$

Solution :

$G=(11011)_{Gray} = (10010)_{(2).}$

10 Code détecteurs d'erreurs

Une erreur de transmission est traduite par le changement d'un bit dans le mot à transmettre d'où l'apparition d'un autre mot. Par exemple le code $(0011)_{(2)} =3_{(10)}$ transmis par erreur $(0111)_{(7)}$ (inversion du troisième bit). Plusieurs codes ont été inventés afin de détecter les erreurs de transmissions. La plus utilisée est celle de bit de parité paire ou impaire [9],[10].

Exemple de code de parité paire (P) en code Gray

Décimal	Gray	P	Décimal	Gray	P
0	0000	0	8	1100	0
1	0001	1	9	1101	1
2	0011	0	10	1111	0
3	0010	1	11	1110	1
4	0110	0	12	1010	0
5	0111	1	13	1011	1
6	0101	0	14	1001	0
7	0100	1	15	1000	1

Tableau 17 : Code de parité paire en code Gray

P est le bit de parité paire

Si le nombre de 1 du code est pair alors P=0 sinon P=1.

Une simple erreur dans le code à transmettre ou dans le bit de parité sera immédiatement détectée. Le récepteur pourra demander une retransmission jusqu'à la réception du bon mot. Par contre une double erreur dans le code ne sera pas détectée.

11 Code détecteurs et correcteurs d'erreurs

Ce code est utilisé dans les bandes magnétiques. Généralement les codes sont envoyés ainsi que leurs parités paires (parité longitudinale (P_L) et parité verticale (P_V) comme l'illustre le tableau suivant. Si un code parmi les codes envoyés comporte une erreur, la machine localisera une erreur et la corrigera immédiatement.[9-11]

	0	**1**	**2**	**3**	**P_L**
	0	0	0	0	**0**
	0	0	0	0	**0**
	0	0	1	1	**0**
	0	1	1	0	**0**
P_V	**0**	**1**	**0**	**1**	

Tableau 18 : Code détecteurs et correcteurs d'erreurs

Le tableau affiche les mots à transmettre en code Gray (0, 1, 2 et 3). P_L est la parité calculée sur les lignes et P_V est la parité calculée sur les colonnes. Si par exemple il y a eu une erreur de transmission du nombre 3 (0010 en Gray), par exemple le $3^{ème}$ bit est erronée, la machine localisera une erreur dans la $3^{ème}$ ligne et dans la $4^{ème}$ colonne et la corrigera immédiatement.

12 Exercices

1. Donner le nom du code qui permet le codage les caractères et expliquer son principe.
2. Convertissez les nombres binaires suivants en code gray :
   a) (11010001)2
   b) (1000010)2
   c) (11011101)2
   d) (11000110)2

3. Convertissez chaque code gray en binaire:
   a) $10100000_{(gray)}$
   b) $10010101_{(gray)}$
   c) $11001001_{(gray)}$
   d) $10100111_{(gray)}$
4. Convertissez les nombres décimaux suivants en BCD:
   a) $(125)_{10}$
   b) $(1015)_{10}$
   c) $(187)_{10}$
   d) $(175)_{10}$
5. Convertissez chaque nombre BCD en nombre décimal
   a) 10100000
   b) 10010101
   c) 11001001
   d) 10100111
6. Additionner les nombres BCD suivants :
   a) $1100000_{(bcd)} + 1010111_{(bcd)}$
   b) $10010101_{(bcd)} + 1000100_{(bcd)}$
   c) $1011001_{(bcd)} + 1110101_{(bcd)}$
   d) $10000110_{(bcd)} + 1010101_{(bcd)}$

7. Convertissez les caractères suivants en code ASCII (voir le tableau ASCII):
   a) U
   b) *
   c) 65
   d) 107
   e) a
8. Donner l'équivalent de chaque caractère ASCII (voir le tableau ASCII):
   a) 0100000
   b) 0010101
   c) 1001001
   d) 0100111
9. Déterminer s'il y a une erreur dans les codes de parité suivants (parité paire) :
   a) 11101010
   b) 11101010
   c) 11101010
   d) 11101010
10. Associer les bits de parité impaire correspondant à chaque caractère suivant :
   a) 1101010
   b) 1110110
   c) 1101010
   d) 1111010

13 Conclusion

Il existe un ensemble de codes conçus pour pouvoir satisfaire aux besoins de l'informatique. Dans ce chapitre, nous avons dressé une liste non exhaustive des différents codes binaires utilisés pour représenter les caractères alphanumériques. Dans le chapitre suivant nous nous intéressons aux bases et aux propriétés fondamentales de l'algèbre de Boole indispensables à la compréhension du fonctionnement des systèmes numériques.

Chapitre 3

Algèbre de Boole

1 Objectifs

- Connaitre les lois fondamentales de l'algèbre de Boole ainsi que les deux théorèmes de De Morgan,
- Savoir définir les trois opérations de base : Négation, ET (intersection), OU (union).
- Connaitre la table de vérité de chacune de ces opérations et leur porte logique.
- Savoir dresser la table de vérité d'une fonction logique et savoir l'implanter.
- Savoir donner une définition sous forme algébrique ou d'une table de vérité des opérations NON-OU, OU exclusif et NON-ET.
- savoir simplifier algébriquement une fonction logique

2 Introduction

L'algèbre de Boole est une algèbre binaire mise en œuvre par le mathématicien anglais **George BOOLE** (1815- 1864) pour étudier la logique. D'importantes applications du domaine des ordinateurs et des appareils numériques reposent sur elle [12],[13].

3 Terminologie

soit ρ un ensemble de variables à deux états, de valeurs de vérité 1 (vrai), 0 (faux), muni d'un nombre limité d'opérateurs : NON(¯), ET(X) , OU (+). ρ est une algèbre de Boole si les postulats suivants sont vérifiés : Soit a, b et c trois variables booléennes

Loi de Commutativité

a + b = b + a

a × b = b× a

Associativité

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

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

Distributivité

a+ (b × c) = (a + b) × (a + c)

a× (b + c) = (a × b) + (a × c)

Identité (élément neutre)

a+ 0 = a

a× 1 = a

Complémentarité

$a + \bar{a} = 1$

$a\ x\ \bar{a} = 0$

$\bar{a}$ est le complément de a

Idempotence

a + a = a

a × a = a

Absorption

a + 1 = 1

a ×0 = 0

Lois de Morgan

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

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

Loi d'involution

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

4 Les opérations de base

4.1 NON (Négation)

• NON : est un opérateur unaire (une seule variable) qui à pour rôle d'inverser la valeur d'une variable. NON(X) est appelé aussi le complément de x. il est noté $\bar{X}$. La table de vérité de NON (x) est donnée par la table de vérité suivante[1].

X	$\bar{X}$
0	1
1	0

4.2 ET (AND)

Le ET est un opérateur binaire (deux variables) , à pour rôle de réaliser le Produit logique entre deux variables booléennes. il est aussi noté par : X.Y. La table de vérité de X.Y est donnée par le tableau 16.

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 19 : Table de vérité de la fonction X.Y

4.3 OU (OR)

Le OU est un opérateur binaire (deux variables) , à pour rôle de réaliser la somme logique entre deux variables logiques. Il est aussi noté par : X + Y. La fonction X + Y est représentée par la table de vérité suivante (voir tableau 17).

X	Y	X+ Y
0	0	0
0	1	1
1	0	1
1	1	1

Tableau 20 : Table de vérité de la fonction X+Y

4.4 OPÉRATEURS NAND ET NOR

Ces deux fonctions sont utilisées pour générer toutes les fonctions booléennes possibles. Ils sont aussi appelés « Eléments de connexion universels », ils sont définis comme suit :

$$\text{NAND (X, Y)} = \overline{X.Y} \quad , \quad \text{NOR} \quad \text{(X,Y)} = \overline{X + Y}$$

[1] On expliquera plus tard comment obtenir une table de vérité à partir d'une fonction.

Les deux fonctions $\overline{X.Y}$ et $\overline{X+Y}$ sont représentées par la table de vérité suivante (voir tableau 18).

X	Y	X+Y	X.Y	$\overline{X+Y}$	$\overline{X.Y}$
0	0	0	0	1	1
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	1	0	0

Tableau 21 : Table de vérité des fonctions NAND (X, Y) et NOR (X,Y)

4.5 Opérateur (OU exclusif) :

Il est définit par la fonction algébrique f :

$$f = x \oplus y = x\overline{y} + \overline{x}\,y$$

Cette expression nous permet de dresser la table de vérité du OU exclusif plus simple (voir tableau 19).

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Tableau 22 : Table de vérité de la fonction OU exclusif.

Sa fonction inverse est donnée par :

$\overline{x \oplus y}$=$xy + \overline{x}\,\overline{y}$ cette fonction vaut 1 si et seulement si les deux entrées sont égales.

5 Dualité de l'algèbre de Boole :

Chaque axiome et chaque postulat possède un équivalent dual, où les éléments 0 sont remplacés par des 1, les 1 par des 0, les (·) par des (+) et vice et versa. Aussi, tout théorème de l'algèbre de Boole a son équivalent dual. Le théorème dual est formulé à partir du théorème de base en remplaçant les éléments 0 par des 1 (respectivement, les 1 par des 0) et les (·) par des (+) (respectivement, les (+) par des (·)) [14]. Voici quelques exemples :

$x.y + x.\overline{y} = x$, sa fonction dual est : $(x + y).(x + \overline{y}) = x$
$x + x.y = x$, sa fonction dual est $x.(x + y) = x$
$x + \bar{x}.y = x + y$, sa fonction dual est : $x.(\bar{x} + y) = x.y$

6 FONCTIONS BOOLÉENNES

Dans le cas général, on appelle fonction booléenne ou fonction logique toute combinaison de variables booléennes reliées par les opérateurs NON et OU. La valeur d'une fonction logique est égale à 1 ou 0 également.

Exemples :

$F2=\overline{(A+\overline{B})+\overline{A}.B}$

$F3=\overline{\overline{A+B}} + \overline{(\overline{\overline{A}+B)+\overline{A}.B)}}$

$F4=xy+\overline{x}y\overline{z}+yz$

6.1 Les tables de vérité

Si une fonction logique possède N variables logiques, ça implique qu'il peut y avoir 2^n combinaisons de ses variables, donc cette même fonction peut avoir 2^n valeurs. Les 2^n combinaisons sont représentées dans une table qui s'appelle table de vérité.

Exemple : vérifions par exemple la loi d'associativité a.(b.c)=(a.b).c, déjà vu dans la section 3 de ce chapitre.

Solution: nous avons trois variables a, b et c, donc 2^3=8 combinaisons possibles. D'où la table de vérité illustrée par le tableau 20:

a	**b**	**C**	(b.c)	a.(b.c)	(a.b)	(a.b).c
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

Entrées en binaire naturel

Identité des colonnes a.(b.c) et (a.b).c

Tableau 23 : Table de vérité à 03 variables

6.2 Extraction de la fonction logique à partir de la T.V (FORMES CANONIQUES)

Une fonction peut être exprimée sous sa forme canonique à partir de sa propre table de vérité.

On appel forme canonique d'une fonction logique, la forme qui permette de localiser chaque ligne d'une table de vérité comportant un 1 ou 0. Il existe plusieurs formes canoniques : les plus utilisées sont la première et la deuxième forme.

6.2.1 Première forme canonique (forme disjonctive)

C'est la forme exprimée en somme de produits (SOP) (ou somme des **mintermes**). On dit aussi que c'est une disjonction de conjonctions. Cette forme est la forme la plus utilisée.
Le principe est de localiser toutes lignes dont la variable de sortie vaut 1 (supposons qu'il y a p lignes). Pour chaque ligne, faire correspondre un produit de tous les variables d'entrée sous la forme normale si la variable d'entrée est à 1, sous la forme complément si la variable d'entrée est à 0. Faire ensuite la somme logique de ces p produits.

Exemple:
Trouver la 1ère forme canonique S.O.P de la fonction à partir de la table de vérité suivante (tableau 21) :

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tableau 24 : Extraction des formes canoniques à partir de la table de vérité

Dans ce cas la forme canonique de F est : $(\bar{a}\,.\,\bar{b}.\,\bar{c} + \bar{a}.\,b.\,c + a.\,\bar{b}.\bar{c} + a.\,b.\,c)$.

6.2.2 Deuxième forme canonique (conjonctive):

C'est la forme exprimée en produit de sommes (POS) (ou produit des **maxtermes).**
Le principe est de localiser toutes lignes dont la variable de sortie vaut 0 (supposons qu'il y a g lignes). Pour chaque ligne, faire correspondre une somme de tous les variables d'entrée sous la forme normale si la variable d'entrée est à 0, sous la forme complément si la variable d'entrée est à 1. Faire ensuite le produit logique de ces g produits.

Exemple :
Prenons la table de vérité ci-dessus, la 2ème forme canonique POS de la fonction F est :

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

Remarque *:*
On peut toujours ramener n'importe qu'elle fonction logique à l'une des formes canoniques.

Cela revient à rajouter les variables manquants dans les termes qui ne contiennent pas toutes les variables (les termes non canoniques). Cela est possible en utilisant les postulats de l'algèbre de Boole :

Exemple :
Mettre la fonction f= xy+$\overline{x}$y$\overline{z}$+yz sous la première forme canonique.

Solution :
Nous remarquons qu'il manque la variable z dans le premier terme et la variable x dans le troisième terme.
Appliquons la loi Complémentarité *:*
$z+\overline{z}=1\ et\ x+\overline{x}=1$
Alors f= xy(z+ $\overline{z}$) + $\overline{x}$y$\overline{z}$+ (x+$\overline{x}$)yz
Faisons la distribution
F= xyz+ xy$\overline{z}$+$\overline{x}$y$\overline{z}$+ xyz + $\overline{x}$yz
D'après l'idempotence :
xyz + xyz = xyz

f devient : xyz + xy$\overline{z}$+$\overline{x}$y$\overline{z}$+$\overline{x}$yz. Cette expression représente la première forme canonique de f.

6.3 Simplification algébrique des fonctions booléennes :

Simplifier une fonction revient à l'exprimer à l'aide d'un nombre minimum de termes. Pour cela il faut d'abord la développer, effectuer des mises en facteur et ensuite simplifier en utilisant les lois fondamentales et les relations démontrées. Néanmoins, les méthodes algébriques de simplification présentent un inconvénient majeur puisque elles ne sont pas systématiques, et leur efficacité dépend donc largement du savoir-faire de la personne qui les applique. Il existe d'autres méthodes de simplification comme la simplification par la table de karnaugh et la simplification par la méthode de Quine Mc-Cluskey. Le lecteur intéressé pourra se reporter aux nombreuses références sur le sujet [12-15].

Exemple :
Simplifiez par la méthode algébrique les fonctions suivantes :
F1=(x+y)$(\overline{y}+z)(\overline{x}+z)$

Solution
Appliquons la distribution du premier terme sur le deuxième

F1= (x $\overline{y}+xz+$ y$\overline{y}+yz)(\overline{x}+z)$
$y\overline{y}=0$ (d'après le théorème de la complémentarité)

F1= (x $\overline{y}+xz+$ $yz)(\overline{x}+z)$

Appliquons les lois de distributivité, commutativité et de l'associativité des opérateurs logiques, nous obtenons :

F1= x $\overline{x}\,\overline{y}$ + x $\overline{y}$z + x $\overline{x}z + xzz + \overline{x}yz + yzz$

Sachant zz=z (d'après l'idempotence)
F1= x $\overline{y}$z + xz + $\overline{x}yz + yz$

D'après la distribution : F1= xz($\overline{y} + 1$) + $(\overline{x} + 1)yz$
Sachant que $\overline{x} + 1 = 1$ (loi d'élément absorbant) alors F1=xz+ yz

7 Circuits logique :

7.1 Définition

Un circuit logique est un ensemble de portes logiques reliées entre elles pour répondre à une expression algébrique [16].

7.2 Portes logiques

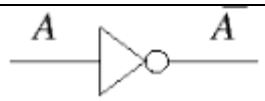
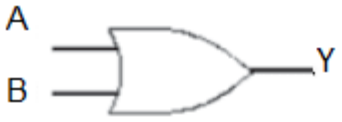
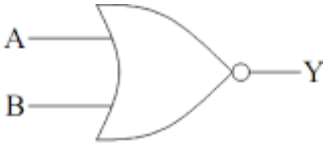
Opérateurs	Fonctions élémentaires	Portes
NOT (Inverseur)	$\overline{x}$	
AND	A.B (A∧B)	
OR	A+B (A∨B)	
NAND	$\overline{x.y}$ $\overline{(A \wedge B)}$	
NOR	$\overline{x+y}$ $\overline{(A \vee B)}$	
XOR	A⊕B	

Tableau 25 : Les portes logiques.

Une porte logique est un circuit électronique élémentaire qui permet de réaliser la fonction d'un opérateur logique de base. Le tableau 22 résume les portes logiques existantes :

7.3 Logigramme d'une fonction

C'est la traduction de la fonction logique en un schéma électronique. Connaissant les portes logiques, on peut construire immédiatement le logigramme de la fonction. Les variables d'entrées correspondent aux fils (lignes) électriques.

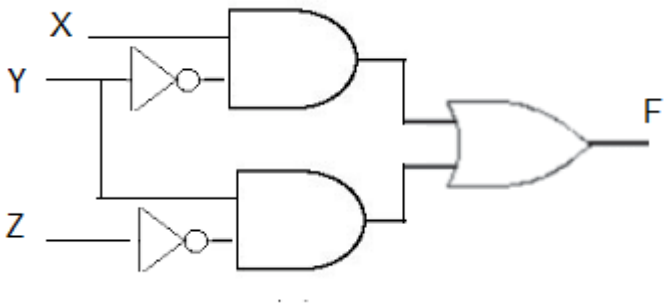
Exemple : Soit la fonction algébrique **F(X,Y,Z)** donnée par :

$$\mathbf{F(X,Y,Z) = X\overline{Y} + Y\overline{Z}.}$$

Donner le logigramme de F.

Solution

Il existe 03 variables d'entrées donc 3 lignes dans le logigramme, la variable Y est inversé dans le premier terme et relié avec x avec une porte AND ainsi que Z est inversé dans le deuxième terme et relié avec Y avec une porte AND. Les deux termes sont reliés par une porte OU . Le schéma en dessous représente le logigramme de la fonction F.



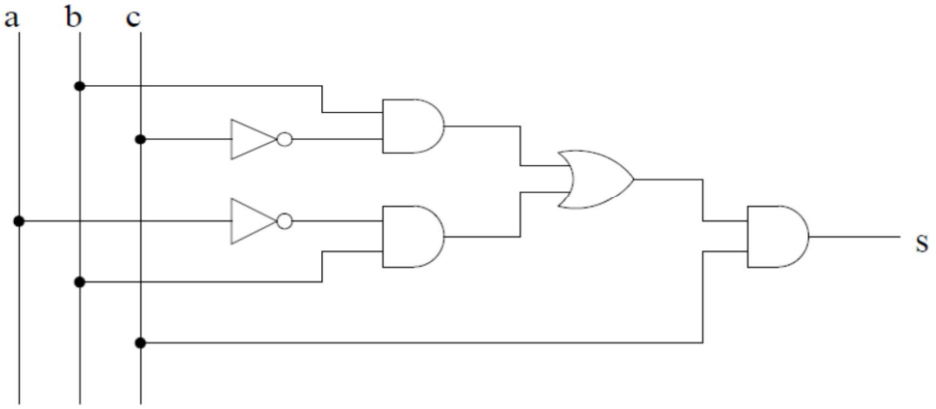
7.4 Étapes de conception et de réalisation d'un circuit logique

Pour faire l'étude et la réalisation d'un circuit logique, il faut suivre les étapes suivantes :

- Il faut définir les variables d'entrée.
- Il faut définir les variables de sortie.
- Etablir la table de vérité.
- Ecrire les équations algébriques des sorties (à partir de la table de vérité).
- Effectuer des simplifications.
- Faire le schéma avec un minimum de portes logiques.

Exemple

Soit le circuit logique de la fonction de sortie S :



1. Dresser la table de vérité correspondante à S.

2. Donner la première forme canonique de S (SOP).

3. Simplifier la forme canonique de s

4. Tracer le logigramme de S simplifiée

Solution

1. On doit d'abord extraire la forme algébrique de S à partir du circuit logique.

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

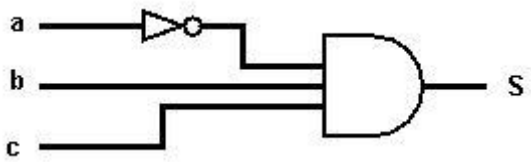
La table de vérité de S est donnée par le tableau suivant. Nous avons 03 variables d'entrées ce qui donne 8 lignes dans la table de vérité.

A	b	C	$\bar{a}$	$\bar{c}$	$b\bar{c}$	$\bar{a}\,b$	$b\bar{c}+\bar{a}\,b$	S
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	1	1	1	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	1	0	1	0
1	1	1	0	0	0	0	0	0

2. Nous remarquons qu'il y a une seule ligne dont S=1, ce qui fait que la forme SOP (S) contient seulement un terme. donc, SOP(S)= $\bar{a}\,b.\,c$

3. La fonction simplifiée = SOP(S) = $\bar{a}\,b.\,c$

4. Logigramme de S simplifiée est donné par le schéma suivant



8 Exercices

1. Simplifiez algébriquement les fonctions booléennes suivantes avec un nombre minimum d'opérateurs :

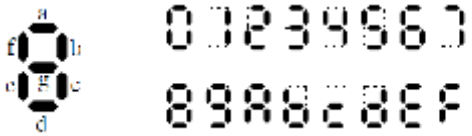
$$F(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.B.C.D + A.\overline{B}.C.\overline{D}$$

$$G(A,B,C,D) = A.\overline{B}.\overline{C}.\overline{D} + A.B.\overline{C}.D + \overline{A}.B.C.D + \overline{A}.\overline{B}.C.\overline{D}$$

$$H(A,B,C,D) = F(A,B,C,D) + G(A,B,C,D)$$

$$K(A,B,C,D) = F(A,B,C,D).G(A,B,C,D)$$

2. On cherche à réaliser un circuit afficheur hexadécimal pour une calculette. L'entrée est un nombre n en binaire sur 4 bits : b0, b1, b2, b3. Les 7 sorties sont appelées a, b, c, d, e, f, g. Une sortie est à 1 si le segment correspondant est noir.



   - Ecrire les tables de vérité des 7 sorties..
   - En déduire le circuit correspondant.

3. Soient les deux fonctions algébriques F1 et F2

$$F1 = (X + Y)(XY + \overline{Z}) Z$$

$$F2 = XY + ZT + X (\overline{Y}\ T + X) + X\overline{T}$$

   - Établir les tables de vérité des deux fonctions.
   - Ecrire F1 et F2 sous les deux formes canoniques SOP et POS.
   - Simplifier les deux fonctions par la méthode algébrique.
   - Soit la fonction algébrique **F(X,Y,Z)** donnée par :

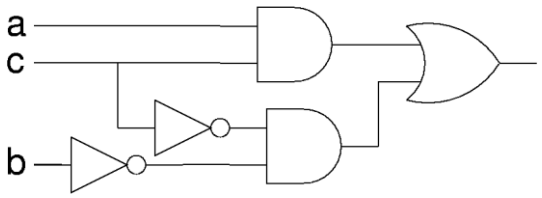
4. Soit la fonction F tel que : $F(X,Y,Z) = [(X+Y)(XY + \overline{Z})] Z + \overline{X}\ Z$
   - Établir la table de vérité de **F(X,Y,Z)**.
   - Ecrire **F(X,Y,Z)** sous la forme **SOP** puis la forme **POS**.
   - Simplifier **F(X,Y,Z)** en utilisant la méthode algébrique.
   - Tracer le logigramme de la fonction **F(X,Y,Z)** simplifiée, en utilisant uniquement les portes **NAND.**

5. Considérons la fonction définie par la table de vérité suivante :

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

   a) Donnez la première forme canonique de **F (S.O.P)**
   b) Faire le schéma logique de la première forme canonique de **F(S.O.P)**
   c) Simplifiez la première forme canonique de **F(S.O.P)** par la méthode algébrique
   d) Faire le schéma logique de cette première forme canonique de **F(S.O.P)** simplifiée
   e) Donnez la deuxième forme canonique de **F(P.O.S)**

6. Soit le circuit logique suivant :
   - Donner la fonction logique de sortie.



- Simplifier la fonction en utilisant les théorèmes de Boole.
- Donner le circuit logique de la fonction simplifiée.

9 Conclusion

Dans ce chapitre, nous avons présenté les notions de base de l'algèbre de Boole. Nous avons vu comment déduire une expression algébrique à partir d'une table de vérité et comment la simplifier en réduisant ainsi le nombre de variables. L'expression donne un logigramme qui est la base de l'implantation de la fonction.

Références :

1. Alain Cazes et al., Architecture des machines et des systèmes informatique. Cours et exercices corrigés. Edition : Dunod 2005.
2. Robert. Strandh et al.,Architectures des l'ordinateurs. Edition : DUNOD 2005.
3. J.M. Muller, Arithmétique des ordinateurs, opérateurs et fonctions élémentaires, Etudes et recherches en informatique, Masson, 1989.
4. Unicode - Official Site, http://www.unicode.org.
5. UT, table de caractères unicode, https://unicode-table.com/fr/blocks/arabic/.
6. Wikipédia, «code ASCII», https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange.
7. Didier Muller, Chapitre 3: Codage de l'information, https://www.apprendre-en-ligne.net/info/codage/codage.pdf,2018.
8. Wikipédia, « Code-barres EAN », <https://fr.wikipedia.org/wiki/Code-barres_EAN>
9. Wikipédia, « Code QR », https://fr.wikipedia.org/wiki/Code_QR.
10. JG. Dumas et al., Théorie des codes : Compression, cryptage, correction, Dunod, 2006.
11. Martin B., Codage, cryptologie et applications, Presses Polytechniques et Universitaires Romandes (PPUR), 2004.
12. Duvallet Claude, « Les codes correcteurs et les codes détecteurs d'erreurs », http://litis.univ-lehavre.fr/~duvallet/enseignements/Cours/LPRODA2I/UF9/LPRODA2I-TD2-UF9.pdf
13. Jean Jacques et al. Architectures des l'ordinateurs. Edition : EYROLLES 2005.
14. Phillipe Darch. Logique booléenne et implémentation Technologique. Edition VUIBERT : 2004.
15. Emanuel Lazard. Architectures des l'ordinateurs. Edition : PEARSON EDUCATION 2006.
16. Tanenbaum Andrew. Architectures des l'ordinateurs. Edition : PEARSON EDUCATION 2005.