République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de RELIZANE
Faculté des Sciences et de la Technologie
Département : Informatique

Polycopie

Matière : Structure Machine 1

Niveau : L1

Auteure : Dr. Naima Beladam

Année universitaire : 2022-2023

Sommaire

Liste de figures

Liste de Tableaux

Introduction Générale

Introduction générale

Ce polycopié de cours à destination des étudiants de L1 a été préparé à l'université de RELIZANE. Il comporte la matière « Structure Machine 1 » établie selon le canevas officiel du ministère de l'éducation supérieure et de la recherche scientifique Algérien.

L'objectif principal de la matière est d'offrir aux étudiants une partie du savoir nécessaire pour comprendre le fonctionnement et la structure des ordinateurs. À l'issue de son suivi, l'étudiant sera capable, dans un premier temps: (1) d'expliquer comment sont représentés les nombres par le biais de systèmes de numération (notamment le système binaire), (2) d'effectuer des conversions entre systèmes de numération, (3) d'expliquer comment sont représentés les nombres dans le système binaire, (4) de réalisés les calculs dans le système binaire et enfin (5) d'expliquer comment on code les caractères. Dans un second temps, l'étudiant sera capable d'expliquer l'algèbre de Boole et de l'appliquer pour représenter et simplifier des fonctions logiques et concevoir des circuits logiques de base.

Le polycopié est organisé en quatre chapitres

Chapitre 1 : Généralités,

Chapitre 2 : Les systèmes de numération,

Chapitre 3 : La représentation de l'information,

Chapitre 4 : L'algèbre de Boole binaire.

Pour s'assurer que l'étudiant assimile bien les connaissances acquises dans le cours, des exercices corrigés en TD viendront consolider ces connaissances.

Chapitre 1 : Généralités

Introduction

L'ordinateur traite de l'information digitale, Tous traitements d'informations est codées sous une forme binaire (0 ou 1). L'ensemble des organes physiques qui servent à ce traitement sont appelés matériel (en anglais Hardware), Le logiciel est un ensemble de programmes, langages et systèmes d'exploitation (en anglais Software) chargés d'assurer l'exploitation des taches de cette machine.

Dans ce chapitre, nous présentons la structure générale de l'ordinateur avec des notions sur le codage de l'information traitée par cette machine.

1. Structure d'un ordinateur

Il est structuré autour de trois composants : un processeur qui manipule l'information et donne un résultat, une mémoire qui mémorise les données à manipuler et des périphériques d'entrées/sorties qui permettent à l'ordinateur de communiquer avec l'extérieur :

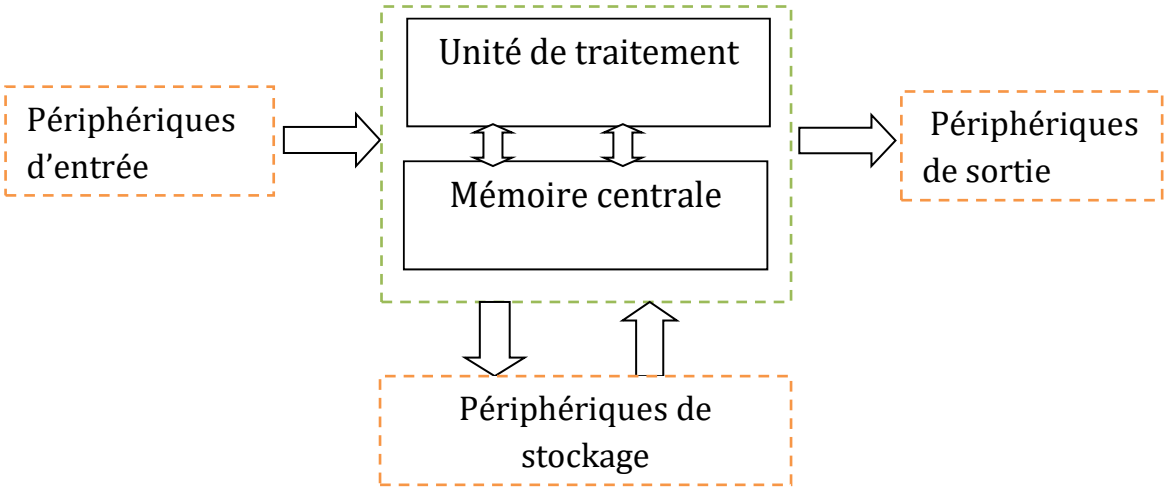


Figure 1.1: Structure d'un ordinateur.

1.1. L'unité de traitement ou le processeur

Le processeur est **l'organe principal** de l'ordinateur, il permet de traiter et de transmettre l'information. Il est constitué de :

1.1.1. L'unité de calcul : elle permet d'effectuer les opérations arithmétiques (addition, multiplication, division, soustraction) et les opérations logiques (la comparaison, et, ou).

1.1.2. L'unité de commande : elle permet de contrôler, gérer et organiser les travaux réalisés par le **CPU**.

1.2. Mémoire centrale

La mémoire centrale est un organe qui permet d'enregistrer, de stocker et de restituer les informations. On distingue deux types :

1.2.1. RAM (Random Access Mémory) : c'est une **mémoire vive**, accessible en **lecture** et en **écriture,** sert à stocker **temporairement** les informations, elle est dite **volatile** parce qu'elle perd son contenu dès qu'elle est hors tension, elle se présente sous forme de petites barrettes. La mémoire vive formée de millions de composants électroniques pouvant retenir ou relâcher une charge électrique.

1.2.2. ROM (Read Only Mémory) : c'est une mémoire morte, permanente, accessible seulement en lecture, Elle contient les programmes de constructeur (**BIOS**) nécessaires au démarrage de l'ordinateur.

1.3. Périphériques

1.3.1. Les périphériques d'entrée

Les périphériques d'entrée sont des organes qui permettent d'envoyer les informations à l'unité centrale, parmi ces périphériques on peut citer :

- Souris
- Scanner
- Clavier…

1.3.2. Les périphériques de sortie

Les périphériques de sortie sont des organes qui permettent de restituer (de faire sortir) les informations sortant de l'unité centrale, parmi ces périphériques on peut citer :

- Imprimante
- Écran
- Haut parleur …

1.3.3. Les périphériques de stockage

Les périphériques de stockage sont des organes qui permettent de stocker et de conserver les informations, parmi ces périphériques on peut citer :

- Disque dur
- Clé USB

- DVD, CD**, …**

2. Codage de l'information

Les informations traitées par les ordinateurs sont de différentes natures : nombres, texte, images, sons, vidéo, programmes, etc. Dans un ordinateur, elles sont toujours représentées sous forme binaire (BIT : Binary digIT) une suite de 0 et de 1. Il est possible de représenter physiquement cette information binaire par un signal électrique ou magnétique, qui correspond à deux états symbolisés: 1 (courant) et 0 (pas de courant).

Un codage est donc nécessaire pour convertir les données symboliques de leur forme externe (habituelle) à la forme binaire (0 et 1) exploitable par l'ordinateur.

2.1. Définition

Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information et sa représentation interne dans la machine (une suite de bits), suivant un ensemble de règle précise

Exemple 1.1

- 35 est la représentation externe du nombre trente cinq;

- La représentation interne de 35 sera une suite de 0 et 1 (100011).

2.2. Les étapes de codage d'information

En informatique, Le codage de l'information s'effectue principalement en trois étapes :

- L'information sera exprimée par une suite de nombres (Numérisation)
- Chaque nombre est codé sous forme binaire (suite de 0 et 1)
- Chaque élément binaire est représenté par un état physique ex : Charge électrique (RAM Chargé (bit 1) ou non chargé (bit 0)).

3. Quantité de l'information traitée

L'unité de base de mesure de la quantité d'information en informatique est donc le bit « binary digit », tel que 1 bit peut prendre la valeur 0 ou 1.

L'octet(en anglais byte ou B majuscule dans les notations) symbolisé par Ø est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.

Une unité d'information composée de 16 bits est généralement appelée mot(en anglais word).

Une unité d'information de 32 bits de longueur est appelée mot double(en anglais double word).

Aussi:

- 1 Kilo-bits (kb) = 2^{10} bits;
- 1 Méga-bits (Mb) = 2^{10} kb=2^{20} bits;
- 1 Giga-bits (Gb) = 2^{10}Mb=2^{30} bits;
- 1 Téra-bits (Tb) = 2^{40} bits;
- 1 Kilo-octet (ko) = 2^{10} octets;
- 1 Méga-octet (Mo) = 2^{20} octets;
- 1 Giga-octet (Go) = 2^{30} octets;
- 1 Téra-octet (To) = 2^{40} octets.

Chapitre 2 : Systèmes de numération

Introduction

Les ordinateurs calculent en binaire. Il s'agit de la base 2 dans laquelle seuls les symboles 0 et 1 sont utilisés pour écrire les nombres. Ceux-ci vont être stockés dans des cases ayant une taille fixée. L'ordinateur est donc limité dans ses capacités à exprimer n'importe quel nombre. Il existe cependant d'autres formes de numération qui fonctionnent en utilisant un nombre de symboles distincts, par exemple le système octal (oct: huit), le système hexadécimal (hexa: seize). En fait, on peut utiliser n'importe quel nombre de symboles différents (pas nécessairement des chiffres) dans un système de numération; ce nombre de symboles distincts est appelé la base du système de numération.

1. Définition

Un Système de numération décrit la façon avec laquelle les nombres sont représentés, il est défini par :

- Une base b : est le nombre qui sert à définir un système de numération ;
- Un alphabet (A): ensemble de symboles ou chiffres ;
- Des règles de représentation des nombres : Juxtaposition de symboles.

2. Représentation d'un nombre dans une base

Un nombre: $(XXX)_b$ indique la représentation d'un nombre XXX dans la base b.

- **Cas 1:** b ≤10, on utilise simplement les chiffres de 0 à b-1.

Exemple 2.1: base 8 (système octal) : n'importe quel nombre sera la combinaison de chiffres appartenant à l'ensemble {0,…, 7}.

- **Cas 2:** b >10, on utilise simplement les chiffres de 0 à 9 ensuite des lettres dans l'ordre alphabétique.

Exemple 2.2: base 16 (système hexadécimal) : n'importe quel nombre sera la combinaison de symboles appartenant à l'ensemble {0,…, 9, A, B, C, D, E, F} tel que : (A=10, B=11,. . . F=15).

Remarque 2.1

La base du système de numération est égale au cardinal (Le nombre de ces chiffres) de l'ensemble des symboles utilisés dans cette base.

Exemple 2.3

en octal, base du système octal = 8; ensemble des symboles utilisés : A ={0,1,2,3,4,5,6,7}, Card (A)=8=base du système octal.

3. Système de numération de position

Lorsqu'on utilise le système décimal: on énumère tous les symboles possibles, 0, 1, 2, jusqu'à 9. Une fois la liste de chiffres épuisée, on ajoute une position à gauche pour former le nombre suivant: 10. La notion de dizaine, centaines, milliers exprime en fait l'exposant que prend la base 10 donnant le poids de la deuxième, troisième position du chiffre dans la représentation du nombre. La position des chiffres à une grande importance. Les chiffres les moins significatifs se situent à droite du nombre, et leur importance augmente au fur et à mesure du déplacement vers la gauche. Donc, chaque chiffre à une valeur qui dépend de sa position. On élève la BASE utilisée à la puissance de la position du chiffre, on multiplie par le chiffre de la position et on additionne l'ensemble. Cette règle est applicable dans toutes les bases.

Cette façon d'écrire les nombres est appelée **système de numération de position** (Figure 2.1)**.** Elle est valable pour tous les systèmes de numération que nous verrons dans ce cours (décimal, binaire, octal et hexadécimal).

Exemple 2.4: Par exemple, le nombre 1985 s'écrit comme :

$1985 = 1 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 5 \times 10^0$

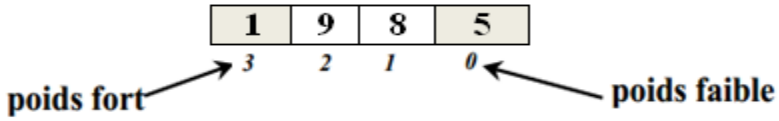


Figure 2.1 : Exemple de système de numération de position.

4. Présentation des systèmes: décimal, binaire, octal et hexadécimal

4.1. Système décimal

Le système décimal est le système de numération dans lequel nous avons le plus l'habitude d'écrire. Chaque chiffre peut avoir 10 valeurs différentes : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, de ce fait, Le nombre 10 est la base de cette numération. C'est un système positionnel. Chaque position possède un poids: la position la plus à droite exprime les unités, la position suivante : les dizaines, ensuite les centaines.

Exemple 2.5

Si on décompose le nombre 2023, nous aurons :

$2023 = 2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

4.2. Système binaire

Le système binaire est le système de fonctionnement des ordinateurs. Chaque chiffre peut avoir 2 valeurs différentes : 0, 1, de ce fait, le système binaire a pour base 2.

4.3. Système Octal

Le système octal est un système de numération qui utilise huit symboles : 0, 1, 2, 3, 4, 5, 6, 7, de ce fait, Le nombre 8 est la base de cette numération. L'intérêt de ce système est que la base 8 est une puissance de 2 ($8 = 2^3$). Chaque symbole de la base 8 est exprimé sur 3 bits.

Exemple 2.6

$(453,01)_8=(100101011,000001)_2$

4.4. Système hexadécimal

Le système hexadécimal utilise les 16 symboles suivant : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $A=10_{(10)}$, $B=11_{(10)}$, $C=12_{(10)}$, $D=13_{(10)}$, $E=14_{(10)}$, $F=15_{(10)}$. De ce fait, le système a pour base 16. Ce système permet de coder 4 bits par un seul symbole.

Exemple 2.7

$(453,01)_{16}=(010001010011,00000001)_2$

Exemple 2.8 : Représentions des nombres de 0 à 16 en décimal et leurs équivalents en binaire et octal et hexadécimal (Tableau 2.1) :

Système décimal	Système octal	Système hexadécimal	Système binaire
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111
16	20	10	10000

Tableau 2.1 : Correspondance entre divers systèmes de bases.

5. Conversion et changement de base (Transcodage)

L'ordinateur ne sait calculer qu'en base 2. Malheureusement, l'écriture binaire n'est ni pratique (à cause de la taille des écritures), ni intuitive (le cerveau humain ne calcule facilement qu'en base 10). On doit donc souvent effectuer des changements de base entre la base 2 et les bases 8, 10 ou 16.

Le transcodage (ou conversion de base) est l'opération qui permet de passer de la représentation d'un nombre exprimé dans une base à la représentation du même nombre, mais exprimé dans une autre base. Les transcodages les plus utilisées sont les suivantes:

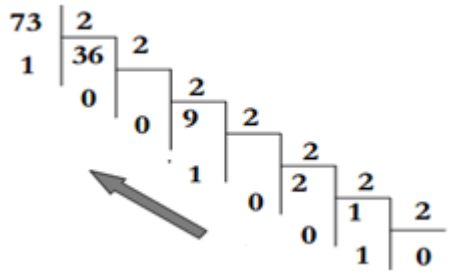
- Base 10 vers base b.
- Base b vers base 10.
- Base 2 vers base 2^n (8 ou 16).
- Base 2^n (8 ou 16) vers base 2.

5.1. Changement de la base 10 vers une base b

Pour effectuer une conversion d'un entier décimal dans une autre base on applique la méthode des divisions successives : on effectue des divisions successives du nombre sur b, et prendre le reste des divisions dans l'ordre inverse.

Exemple 2.9: Décimale vers binaire

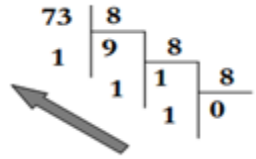
$(73)_{10=}(?)_2$



Donc : $(73)_{10=}(1001001)_2$

Exemple 2.10: Décimale vers octale

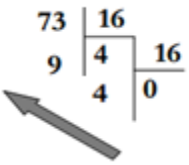
$(73)_{10=}(?)_8$



Donc : $(73)_{10=}(111)_8$

Exemple 2.11: Décimale vers hexadécimale

$(73)_{10=}(?)_{16}$



Donc : $(73)_{10=}(49)_{16}$

5.2. Conversion d'un nombre de base b quelconque en nombre décimal

Tout nombre entier naturel peut se coder comme la somme pondérée des puissances de sa base b, quel que soit cette base. Donc, quelque soit la base numérique employée, elle suit la relation suivante :

$$\sum_{i=0}^{n-1}(a_i b^i) = (a_{n-1}b^{n-1} + \cdots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0)$$

Ou a_i : chiffre de la base de rang i ;

b^i : puissance de la base b d'exposant de rang i.

Exemple 2.12 :

$(1011)_2 = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) = (11)_{10}$

$(16257)_8 = (1 \times 8^4 + 6 \times 8^3 + 2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0) = (7343)_{10}$

$(F53)_{16} = (15 \times 16^2 + 5 \times 16^1 + 3 \times 16^0) = (3923)_{10}$

5.3. Conversion d'un nombre de la base binaire vers les bases : octal, hexadécimal

L'utilisation des bases (8 et 16) permet de réduire le nombre de symboles à écrire tout en conservant la possibilité de conversion instantanée en binaire. Pour convertir un nombre de la base 2 vers la base 8 et 16, il suffit de regrouper les bits par groupes de n (3 pour la base octale et 4 pour la base hexadécimale), et de remplacer chacun de ces groupes par le symbole correspondant dans la base d'arrivée. Pour la partie entière, le regroupement part du bit de poids le plus faible, et pour la partie fractionnaire, du bit de poids le plus fort (de la virgule). Lorsqu'un groupe est incomplet, on le complète avec des 0.

Exemple 2.13:

1) binaire vers octale

$N = (1010011101)_2 = (?)_8$

$N = (\underbrace{001}\ \underbrace{010}\ \underbrace{011}\ \underbrace{101})_2$

$= (1\ \ 2\ \ 3\ \ 5)_8$

2) binaire vers Hexadécimale

$N = (1010011101)_2 = (???)_{16}$

$N = (\underbrace{0010}\ \underbrace{1001}\ \underbrace{1101})_2$

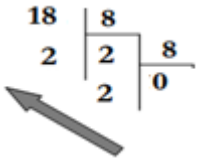
$= (2\ \ 9\ \ D)_{16}$

5.4. Conversion d'un nombre d'une base b_1 vers d'une base b_2

Pour convertir un nombre d une base b_1 vers une autre base b_2 , on doit convertir le nombre en base b_1 vers la base décimale puis convertir ce nombre en base 10 vers la base b2.

Exemple 2.14 :

$(10010)_2=(\ ?)_8$



$(10010)_2 = (1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0) = (18)_{10}$. **Et :**

Donc: $(10010)_2 = (22)_8$

6. Opérations arithmétiques

Les règles des opérations arithmétiques en décimal sont aussi valables pour les opérations arithmétiques dans n'importe quelle base.

6.1. Addition binaire

Pour additionner deux nombres binaires, on procède exactement comme en décimale, mais en prenant en compte les règles de calcul suivantes :

- 0+0 = 0 retenue 0
- 0+1 = 1 + 0 = 1 retenue 0
- 1 + 1 = 0 retenue 1
- 1 + 1 + 1 = 1 retenue 1

Exemple 2.15

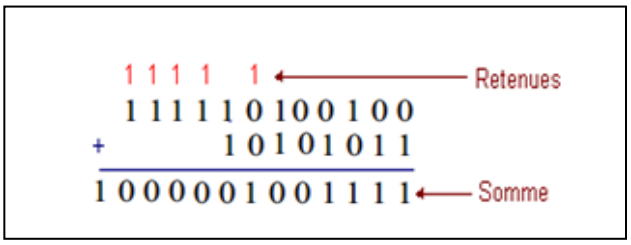


Figure 2.2 : Exemple d'addition binaire.

6.2. Soustraction binaire

Dans la soustraction binaire, on procède comme en décimal. Quand la quantité à soustraire est supérieure à la quantité dont on soustrait, on emprunte 1 au voisin de gauche. En binaire, ce 1 ajoute 1 a la quantité dont on soustrait. Voici les règles de calcul à utiliser :

- 0 – 0 = 0 emprunte 0
- 0 –1 = 1 emprunte 1
- 0 – 1– 1= 0 emprunte 1
- 1 – 1– 1 = 1 emprunte 1
- 1 – 0 = 1 emprunte 0
- 1 – 1 = 0 emprunte 0

Exemple 2.16

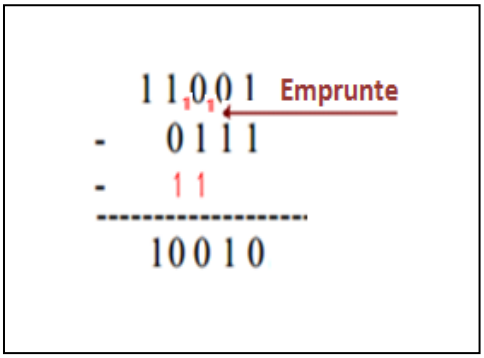


Figure 2.3 : Exemple de soustraction binaire.

6.3. Multiplication binaire

La multiplication binaire se réalise comme une multiplication décimale. Voici les règles de calcul à utiliser :

- 0 x 0 = 0
- 0 x 1 = 0
- 1 x 0 = 0
- 1 x 1 = 1

Exemple 2.17

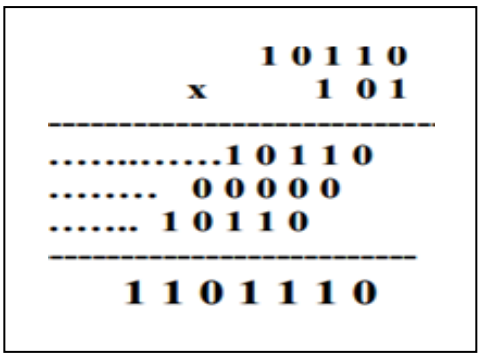


Figure 2.4 : Exemple de multiplication binaire.

6.4. Division binaire

La division binaire s'effectue à l'aide de soustractions et de décalages, comme la division décimale, sauf que les chiffres du quotient ne peuvent être que 1 ou 0. Le bit du quotient est 1 si on peut soustraire le diviseur, sinon il est 0.

Exemple 2.18

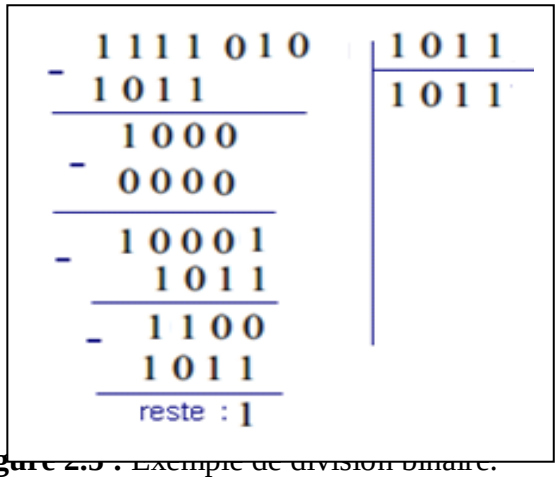


Figure 2.5 : Exemple de division binaire.

Chapitre 3 : Représentation de l’information

Introduction

Les informations traitées au niveau de l'ordinateur se présentent sous forme des instructions et des données. Avant qu'elles subissent des traitements, elles vont subir une opération de codage qui permet de déduire leur représentation interne. Ensuite, les opérations demandées sont de la représentation interne vers la représentation externe) est nécessaire pour les restituer vers l'extérieur.
Dans ce chapitre, nous apprendrons à représenter les différents types de donnés (numérique et non numérique).

1. Le codage binaire

Le codage binaire d'une information revient à établir une correspondance entre la représentation externe de l'information et sa représentation interne sous forme de suite de bits.

1.1. Le code binaire pur

Il est aussi appelé code binaire naturel. C'est le code binaire sans aucune codification, c'est-à-dire qui découle directement du principe général de la numération. C'est le code naturel utilisé dans les systèmes numériques (ordinateur, etc.).

1.2. Le code binaire réfléchi (code de GRAY)

On dit qu'un code est continu au sens large si dans la table de vérité qui le définit, les états successifs sont adjacents, c'est-à-dire que quand on passe de l'un à l'autre, il y a un seul chiffre qui change. Un code est continu au sens strict si en plus le premier et le dernier état sont adjacents.

Un code réfléchi est un code naturel dont on a renversé le sens de variation par endroits afin de le rendre continu. On renverse une période sur deux en commençant par la deuxième.

Le code de Gray est le code binaire réfléchi (ou un seul bit change quand on passe d'une valeur à la valeur suivante), c'est un cas très important des codes continu. Il est très fréquemment utilisé notamment sur les tables de Karnaugh pour simplifier les fonctions logiques.

1.3. Le code DCB (Décimale codé binaire) ou BCD (*binary coded decimal*)

Les informations traitées par l'ordinateur ne sont pas toujours converties selon le système de numération binaire. En effet, ces informations peuvent être manipulées sous forme décimale codée binaire, c'est-à-dire, que des codes (en binaires) sont associés à des nombres décimaux sans pour autant faire la conversion traditionnelle décimal → binaire.

Le principe consiste à faire des éclatements sur 4 bits et de remplacer chaque chiffre décimal par sa valeur binaire correspondante (Tableau 3.1), les combinaisons supérieures à 9 sont interdites. Ce type de codage est très utilisé pour les systèmes d'affichage sur afficheurs 7 segments.

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

Tableau 3.1 : Code BCD.

Exemple 3.1

$(129)_{10} = (\underbrace{0001}\ \underbrace{0010}\ \underbrace{1001})_{BCD}$

1.4. Le code excède de trois (BCD+3)

Ce code ressemble beaucoup au code BCD. Son principe est basé sur le fait d'associer à chaque chiffre décimal son équivalent binaire additionné de 3 (Tableau 3.2).

Décimal	BCD+3	Binaire
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Tableau 3.2 : Code BCD+3.

Exemple 3.2

$(129)_{10}=\underbrace{(0100\ 0101\ 1100)}_{BCD+3}$

2. Représentation des caractères

Les caractères, appelés symboles alphanumériques, incluent les lettres majuscules et minuscules, les symboles de ponctuation (&~ , . ; # " - etc...), et les chiffres. Les caractères sont des données non numériques (addition n'a pas de sens) ; Par contre, il est souvent utile de comparer deux caractères, par exemple pour les trier dans l'ordre alphabétique.

Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère. Les différents codes utilisés sont:

- ASCII (7 bits)
- ASCII étendu (8 bits)
- Unicode (16 bits)
- Etc.

2.1. Code ASCII (7 bits)

Autrement dit : American Standard Code for Information Interchange. C'est un codage sur 7 bits ce qui donne 128 codes différents (0 à 127) (voir le Tableau 3.3) :

- 0 à 31 : caractères de contrôle (retour à la ligne, Bip sonore, etc.…)

- 48 à 57 : chiffres dans l'ordre (0,1,…,9)
- 65 à 90 : les alphabets majuscules (A,…,Z)
- 97 à 122 : les alphabets minuscule (a,…z)

Dans ce code, les caractères accentués ne sont pas représentées et c'est la raison pour laquelle il est limité uniquement à quelques langues (anglais notamment).

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char	Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	*NUL*	040	32	20	*SPACE*	100	64	40	@	140	96	60	`
001	1	01	*SOH*	041	33	21	!	101	65	41	A	141	97	61	a
002	2	02	*STX*	042	34	22	"	102	66	42	B	142	98	62	b
003	3	03	*ETX*	043	35	23	#	103	67	43	C	143	99	63	c
004	4	04	*EOT*	044	36	24	$	104	68	44	D	144	100	64	d
005	5	05	*ENQ*	045	37	25	%	105	69	45	E	145	101	65	e
006	6	06	*ACK*	046	38	26	&	106	70	46	F	146	102	66	f
007	7	07	*BEL*	047	39	27	'	107	71	47	G	147	103	67	g
010	8	08	*BS*	050	40	28	(	110	72	48	H	150	104	68	h
011	9	09	*HT*	051	41	29	)	111	73	49	I	151	105	69	i
012	10	0A	*LF*	052	42	2A	*	112	74	4A	J	152	106	6A	j
013	11	0B	*VT*	053	43	2B	+	113	75	4B	K	153	107	6B	k
014	12	0C	*FF*	054	44	2C	,	114	76	4C	L	154	108	6C	l
015	13	0D	*CR*	055	45	2D	-	115	77	4D	M	155	109	6D	m
016	14	0E	*SO*	056	46	2E	.	116	78	4E	N	156	110	6E	n
017	15	0F	*SI*	057	47	2F	/	117	79	4F	O	157	111	6F	o
020	16	10	*DLE*	060	48	30	0	120	80	50	P	160	112	70	p
021	17	11	*DC1*	061	49	31	1	121	81	51	Q	161	113	71	q
022	18	12	*DC2*	062	50	32	2	122	82	52	R	162	114	72	r
023	19	13	*DC3*	063	51	33	3	123	83	53	S	163	115	73	s
024	20	14	*DC4*	064	52	34	4	124	84	54	T	164	116	74	t
025	21	15	*NAK*	065	53	35	5	125	85	55	U	165	117	75	u
026	22	16	*SYN*	066	54	36	6	126	86	56	V	166	118	76	v
027	23	17	*ETB*	067	55	37	7	127	87	57	W	167	119	77	w
030	24	18	*CAN*	070	56	38	8	130	88	58	X	170	120	78	x
031	25	19	*EM*	071	57	39	9	131	89	59	Y	171	121	79	y
032	26	1A	*SUB*	072	58	3A	:	132	90	5A	Z	172	122	7A	z
033	27	1B	*ESC*	073	59	3B	;	133	91	5B	[	173	123	7B	{
034	28	1C	*FS*	074	60	3C	<	134	92	5C	\	174	124	7C	|
035	29	1D	*GS*	075	61	3D	=	135	93	5D	]	175	125	7D	}
036	30	1E	*RS*	076	62	3E	>	136	94	5E	^	176	126	7E	~
037	31	1F	*US*	077	63	3F	?	137	95	5F	_	177	127	7F	*DEL*

Tableau 3.3 : Table ASCII.

2.2. Table ASCII Etendu (8 bits)

Afin de représenter plus de langues et notamment des langues occidentales comme la langue Français, le code ASCII a été étendu de 7 à 8 bits (0 à 255).

2.3. Code Unicode (16 bits)

Unicode utilise des codes de valeurs bien plus grandes que celle allant de 0 à 127. C'est un codage sur 16 bits. Il permet de représenter tous les caractères spécifiques aux différentes langues. Par exemple plusieurs langues partagent les mêmes lettres (français, anglais, …) notamment les lettres de l'alphabet français non accentués.

2.4. Code UTF8

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". Par exemple, pour le texte "Bienvenue à Relizane!", seul le « à » ne fait pas partie du code ASCII. Donc seul un caractère sera sur 16 bits, les autres seront représentés sur 8 bits.

3. Représentation des nombres

La représentation (codification) des nombres est nécessaire afin de les stocker et de les manipuler par un ordinateur. Les nombres (données numériques) sont de différents types :

- Nombres entiers positifs ;
- Nombres entiers négatifs ;
- Nombres fractionnaires.

3.1. Représentation des entiers positifs

Dans la représentation des entiers positifs, on utilise le codage binaire pur. Dans ce codage on associe à chaque entier positif la valeur qui lui correspond selon le système de numération binaire. Ainsi, en ayant *n* bits on pourra coder les valeurs comprises entre [0 et $2^n - 1$].

Exemple 3.3: sur un octet, $(10)_{10} = (0\ 0\ 0\ 0\ 1\ 0\ 1\ 0)_2$

3.2. Représentation des entiers relatifs

Il existe 3 méthodes pour représenter les nombres négatifs :

- Code binaire signé (par signe et valeur absolue) ;
- Complément à 1(complément restreint) ;
- Complément à 2 (complément à vrai).

3.2.1. Représentation avec signe et valeur absolue

Dans cette représentation, on utilise le bit de poids fort du nombre pour représenter un nombre négatif. Sur n bits, le bit de poids fort (aussi appelé « bit de signe ») indique le signe du nombre (1 pour un nombre négatif) et les n – 1 bits restants donnent la valeur absolue binaire du nombre. Ainsi, en ayant *n* bits on pourra coder les valeurs comprises entre $-(2^{n-1}-1)$ et $(2^{n-1}-1)$

Exemple 3.4 : Sur 8 bits, codage des nombres $(-24)_{10}$ et $(-128)_{10}$ en (SVA)

- $(-24)_{10} = (1\ 0\ 0\ 1\ 1\ 0\ 0\ 0)_{SVA}$
- $(-128)_{10}$ hors limite nécessite ⟶ 9 bits au minimum car $(128)_{10} = (10000000)_2$

- Limitation :

Ce codage présent deux valeurs pour deux représentations du zéro : $(+0)_{10}$ et $(-0)_{10}$, qui conduisent à des difficultés au niveau des opérations arithmétiques. Par exemple Sur 4 bits :

- $(+0)_{10} = (0000)_{sva}$,
- $(-0)_{10} = (1000)_{sva}$

3.2.2. Représentation en Complément à 1(ou complément restraint)

Les nombres positifs sont codés de la même façon qu'en binaire pure. Un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue. Le bit le plus significatif est utilisé pour représenter le signe du nombre :

- si le bit le plus fort = 1 alors nombre négatif
- si le bit le plus fort = 0 alors nombre positif

Exemple 3.5 : $(-24)_{10}$ en complément à 1 sur 8 bits

- |-24|en binaire pur ⟶ $(0\ 0\ 0\ 1\ 1\ 0\ 0\ 0)_2$ puis
- on inverse les bits ⟶ $(1\ 1\ 1\ 0\ 0\ 1\ 1\ 1)_{cà1}$

Remarque 3.1

- Dans cette représentation, le bit du poids fort nous indique le signe (0 : positif, 1: négatif).
- Le complément à un du complément à un d'un nombre est égale au nombre lui-même.

CA1(CA1(N))= N

- Limitation :

Ce codage présent deux valeurs pour deux représentations du zéro : $(+\ 0)_{10}$ et $(\ -\ 0)_{10}$, qui conduisent à des difficultés au niveau des opérations arithmétiques. Par exemple Sur 8 bits :

- $(+0)_{10} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)_{cà1}$
- $(-0)_{10} = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)_{cà1}$

3.2.3. Représentation en Complément à 2 (ou complément Vrai)

La représentation en complément à deux (complément à vrai) est la représentation la plus utilisée pour la représentation des nombres négatifs dans la machine. Dans cette représentation, les nombres positifs sont codés de la même manière qu'en binaire pure. Un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1.

Ainsi, en ayant *n* bits on pourra coder les valeurs comprises entre $-(2^{n-1})$ à $(2^{n-1}-1)$

Exemple 3.6 : $(\ -24)_{10}$ en complément à 2 sur 8 bits

- $(24)_{10} = (0\ 0\ 0\ 1\ 1\ 0\ 0\ 0)_2$
- $(\ -24)_{10} = (1\ 1\ 1\ 0\ 0\ 1\ 1\ 1)_{cà1} = (1\ 1\ 1\ 0\ 1\ 0\ 0\ 0)_{cà2}$

- Avantage :

Dans le complément à deux ; il existe un seul codage pour le zéro. Par exemple sur 8 bits :

- $(+0\ )_{10}$ est codé par $(00000000)_{cà2}$
- $(-0\ )_{10} = (11111111)_{cà1} = (1\ 1\ 1\ 0\ 1\ 0\ 0\ 0)_{cà2}$

Remarque 3.2

- Dans cette représentation, le bit du poids fort nous indique le signe (0 : positif, 1 : négatif).
- Le complément à deux du complément à deux d'un nombre est égale au nombre lui même.

  CA2(CA2(N))= N

3.2.4. Opérations arithmétiques en CA2

A. Principe :

• Dans l'addition, une retenue générée par le bit de signe sera ignorée.

• Une condition nécessaire pour accepter le résultat ainsi obtenu, est qu'une retenue doit être **aussi** générée par le bit juste avant signe.

• Si une retenue est générée exclusivement par les bits de signe **ou** par le bit juste avant signe, l'addition est impossible a réalisée. Une erreur de dépassement de capacités est déclarée.

Exemple 3.7

Effectuer les opérations suivantes sur 5 Bits, en utilisant la représentation en CA2

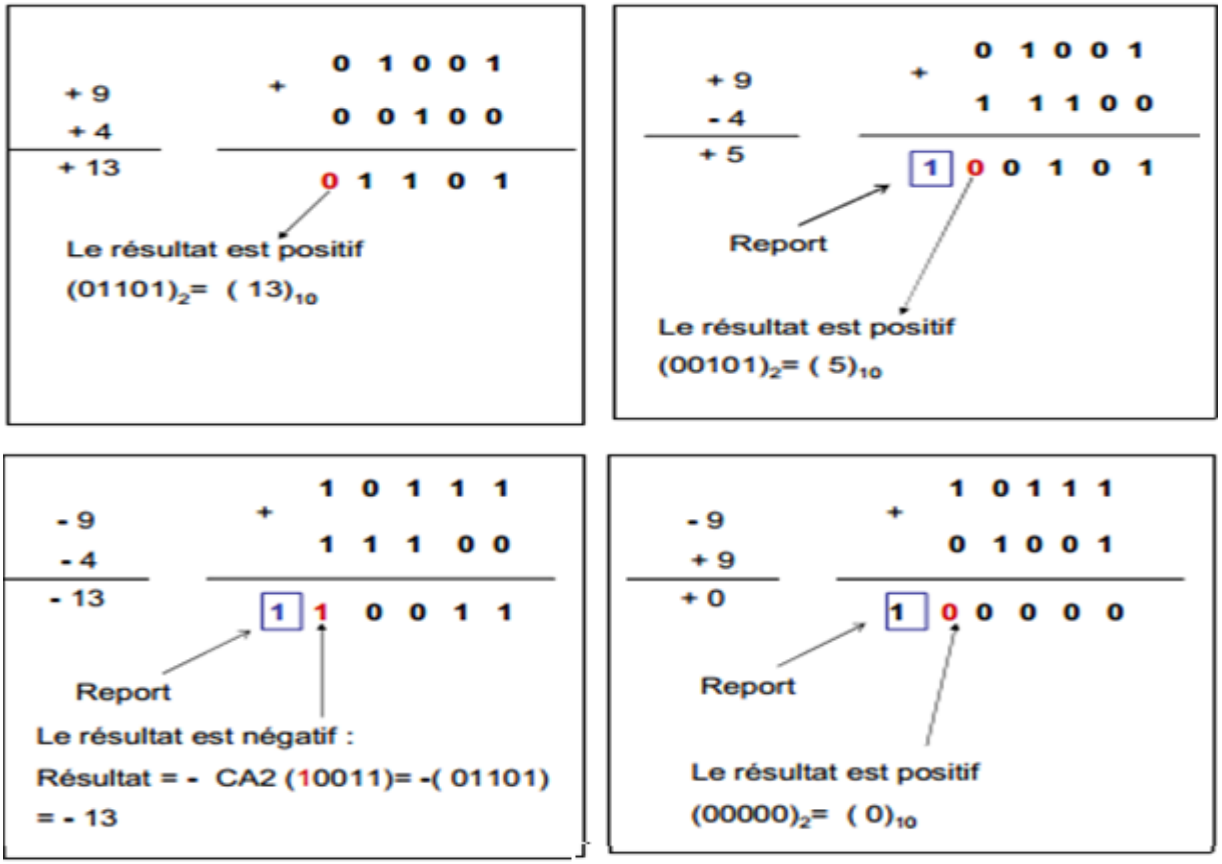


Figure 3.1 : Opérations d'addition en CA2.

3.2.5. La retenue et le débordement

- On dit qu'il y a une retenue si une opération arithmétique génère un report.
- On dit qu'il y a un débordement (Over flow) ou dépassement de capacité, si le résultat de l'opération sur n bits est faux. Autrement dit le résultat dépasse l'intervalle des valeurs sur les n bits utilisés.

Exemple 3.8 : Effectuer les opérations suivantes sur 5 Bits, en utilisant la représentation en CA2

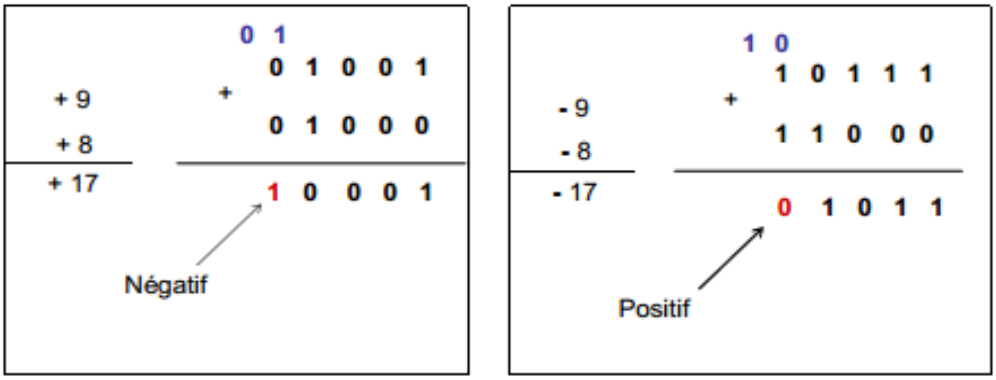


Figure 3.2 : Exemple d'un débordement.

Remarque 3.3 : Nous avons un débordement si :

- la somme de deux nombres positifs donne un nombre négatif ;
- Ou la somme de deux nombres négatifs donne un Nombre positif ;
- Il n'y a jamais un débordement si les deux nombres sont de signes différents.

3.3. Représentation des nombres fractionnaires

On distingue deux formats de représentations des nombres réels:

- **Format virgule fixe** (utilisé par les premières machines)
- **Format virgule flottante** (utilisé actuellement sur machine)

3.3.1. Représentation en virgules fixe

Cette représentation possède une partie 'entière' et une partie 'décimale' séparés par une virgule. La position de la virgule est fixe d'où le nom. Etant donné une base b; un nombre x est représenté par :

$$x = (a_{n-1}a_{n-2} \dots a_1a_0, a_{-1}a_{-2} \dots a_{-P})_b$$

- a_{n-1} est le chiffre de poids fort
- a_{-p} est le chiffre de poids faible
- n est le nombre de chiffre avant la virgule
- p est le nombre de chiffre après la virgule
- la valeur de x en base 10 est : $x = \sum_{-p}^{n-1} a_i b^i$

Exemple 3.9

$(101,01)=1.2^2+0.2^1+1.2^0+0.2^{-1}+1.2-2 = (5,25)_{10}$

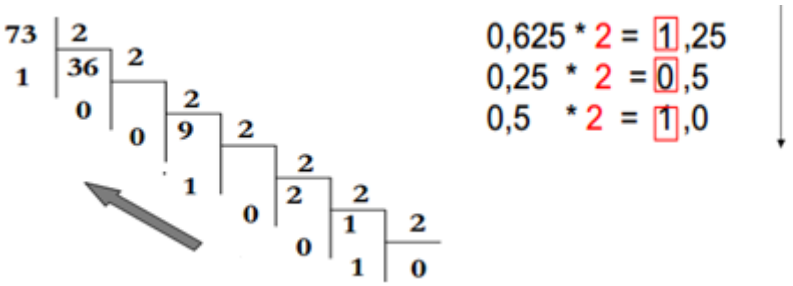
Donc le codage d'un nombre réel en virgule fixe est comme suivant :

- Partie entière est codée sur p bits (est transformée en effectuant des divisions successives par b)
- Partie décimale est codée sur q bits en multipliant par b successivement jusqu'à ce que la partie décimale soit nulle ou le nombre de bits q est atteint.

Exemple 3.10 : $(73,625)_{10}=(?)_2$

P.E = 73

P.F= 0,625



Donc : $(73,625)_{10}= (1001001,101)_2$

3.3.2. Représentation en virgules flottante (IEEE 754)

Chaque nombre réel peut s'écrire de la façon suivante :$N= \pm M .b^E$ (**Figure 3.3**)

- M : une mantisse (en virgule fixe)
- E ; un exposant (un entier relative)
- b : une base (2,8,10,16,etc.)

S : signe	E : Exposant	M : Mantisse (non signé)

Figure 3.3: Format de la virgule flottante.

Exemple 3.11

- $15,6 = 1,56 * 10^{+1}$
- $(110,101)_2= - (1,10101)_2 * 2^{+2}$
- $(0,00101)_2= (1,01)_2 * 2^{-3}$

Donc, le codage en base 2, format virgule flottante, revient à coder le signe, la mantisse et l'exposant.

On appelle la notation scientifique chaque nombre écrit sous la forme $1,M\ 2^E$, où 1,M s'appelle la mantisse du nombre et E l'exposant. Comme la mantisse commence toujours par une partie entière égale à 1, on ne l'écrit pas et on n'exprime que la partie fractionnaire, M, appelée « Mantisse Normalisée». De nos jours, pratiquement tous les constructeurs ont des processeurs dédiés qui effectuent des calculs en virgule flottante et qui emploient la représentation au standard IEEE 754, cette normalisation adoptée définit deux principaux types de nombres en virgule flottante

- La simple précision sur 32 bits et la double précision sur 64 bits. Les nombres en simple précision possèdent une mantisse sur 23 bits (correspondant aux puissances négatives de 2), un exposant sur 8 bits et un bit de signe (voir **Figure 3.4**)

Signe	**Exposant**	**Mantisse Normalisée**	$\mathbf{E_{min}}$	$\mathbf{E_{max}}$
1 bit	8 bits	23 bits	-26	+127

Figure 3.4: Représentation d'un nombre au format IEEE 754 en simple précision.

- Les nombres en double précision ont une pseudo-mantisse sur 52 bits, un exposant sur 11 bits et un bit de signe (voir **Figure 3.5)**

Signe	**Exposant**	**Mantisse Normalisée**	$\mathbf{E_{min}}$	$\mathbf{E_{max}}$
1 bit	11 bits	52 bits	-1022	+1023

Figure 3.5: Représentation d'un nombre au format IEEE 754 en double précision.

Exemple 3.12 : Nous voulons représenter -6,53125 au format IEEE 754 simple précision.

On a : $(6,53125)_{10} = (110,10001)_2 = 1,1010001 * 2^2$

Donc : S= 1 car -6,53125 est négatif.

- Eb =E + 127 = 2 + 127 = 12910 = $(10000001)_2$ en base 2 sur 8 bits.
- M .N= 10100010000000000000000. sur 23 bits.

On obtient la représentation suivante :

1	**10000001**	**10100010000000000000000**
1 bit	8 bits	23 bits

Figure 3.6: Exemple de représentation d'un nombre au format IEEE 754 en simple précision.

Chapitre 4 : Algèbre de Boole

Introduction

Les machines numériques sont constituées d'un ensemble de circuits électroniques. Chaque circuit fournit une fonction logique bien déterminée (addition, comparaison, etc). Pour concevoir et réaliser ces circuits on doit avoir un modèle mathématique de la fonction réalisée par le circuit. Ce modèle doit prendre en considération le système binaire. En 1847 Georges Boole, mathématicien anglais, a créé une algèbre applicable au raisonnement logique qui traite des fonctions à variables binaires (ne pouvant prendre que deux états). Ce chapitre décrit les fondements d'algèbre de Boole

1. Définitions et conventions

1.1. Niveau logique

Lorsqu'on fait l'étude d'un système logique il faut bien préciser le niveau du travail.

Niveau	Logique Positive	Logique Négative
H(Hight) haut	1	0
L (Low) bas	0	1

Tableau 4.1 : les niveaux logiques.

1.2. Variable logique (booléenne)

Une variable logique (booléenne) est une variable qui peut prendre soit la valeur 0 ou 1 . Généralement elle est exprimée par un seul caractère alphabétique en majuscule (A ,B,S ,…)

1.3. Fonction logique

C'est une fonction qui relie N variables logiques avec un ensemble d'opérateurs logiques. On peut décrire une fonction donnée en explicitant ses 2^n valeurs, par sa table de vérité. Il s'agit d'un tableau à 2^n lignes, qui liste pour chaque combinaison possible ; la valeur prise par la fonction. L'ordre des lignes n'a pas d'importance mais pour faciliter la lecture et les comparaisons, on écrit souvent les 2^n combinaisons dans l'ordre numérique.

2. Opérateurs logiques de base

Un certain nombre de fonctions booléennes forment les fonctions logiques de base qui permettent d'en construire des plus complexes. Ces fonctions de base peuvent être définies par leur table de vérité ou leur expression algébrique.

2.1. Fonction logique NON (NOT : négation)

Le rôle de cette fonction est d'inverser la valeur d'une variable.

- **Expression logique :** $\mathbf{F(A) = Non\ A = \bar{A}}$
- **Table de vérité**

Entrée	Sortie
A	F(A)
0	1
1	0

Tableau 4.2: Table de vérité de la fonction Non

- **Symbole graphique**

Figure 4.1: Symbole graphique la fonction Non**.**

2.2. Fonction logique ET (AND)

Le rôle de cette fonction est de réaliser le Produit logique entre deux variables booléennes. Elle fait la conjonction entre deux variables.

- **Expression logique :** F(A,B) = A * B ou A • B ou AB
- **Symbole graphique**

Figure 4.2: Symbole graphique la fonction ET(AND).

Table de vérité

Entrée		Sortie
A	B	F(A)
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 4.3: Table de vérité de la fonction ET (AND).

2.3. Fonction logique OU (OR)

Le rôle de cette fonction est de réaliser la somme logique entre deux variables logiques. Elle fait la disjonction entre deux variables.

- **Expression logique :** F(A,B) = A + B
- **Table de vérité**

Entrée		**Sortie**
A	**B**	**F(A)**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Tableau 4.4: Table de vérité de la fonction OU (OR).

- **Symbole graphique**

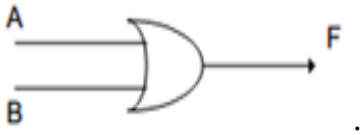


Figure 4.3: Symbole graphique la fonction OU(OR).

2.4. Précédence des opérateurs (priorité des opérateurs)

Pour évaluer une expression logique (fonction logique) : on commence par évaluer les sous expressions entre les parenthèses, puis le complément (NON), ensuite le produit logique (ET), enfin la somme logique (OU).

2.5. Lois fondamentales de l'Algèbre de Boole

Les fonctions et les circuits logiques étant définis à partir d'expressions algébriques, il est important de pouvoir simplifier celles-ci afin de réduire la complexité des circuits en utilisant un certain nombre de propriétés algébriques (Tableau 4.5)

Fermeture	Si A et B sont des variables booléennes, alors A+B , AB sont aussi des variables booléennes.
Commutativité	A+B = B+A A. B= B.A
Associativité	A + (B + C) = (A+B) + C A . (B .C) = (A. B) . C
Distributivité	ET/OU A . (B + C) = (A.B) + (A.C) OU/ET A + (B . C) = (A+B) . (A+C)
Idempotence	A+A=A A . A=A
Complémentarité	A+$\overline{A}$=1 A. $\overline{A}$= 0
Identique remarquable	1+A=1 1.A=A 0+A=A 0.A=0
Distributivité interne	A . (B . C) = (A.B) .(A.C) A +(B + C) = (A+B) + (A+C)
Théorème de DE-MORGANE	$\overline{A + B} = \bar{A}.\bar{B}$ $\overline{A.B} = \bar{A} + \bar{B}$

Tableau 4.5 : Propriétés algébriques des opérateurs.

3. Autres opérateurs logiques

3.1. Fonction logique OU exclusif (XOR)

Il porte sur deux variables d'entrée. Elle prend la valeur 1 si l'une ou l'autre de ses entrées est à 1, mais pas les deux. Elle vaut 0 si les deux entrées sont égales (à 0 ou à 1). Le tableau 4.6 suivant résume l'action de cet opérateur. Son expression algébrique est :

$$F(A,B) = A \oplus B = \bar{A}B + A\bar{B}$$

- **Table de vérité**

Entrée		**Sortie**
A	**B**	**F(A)**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Tableau 4.6: Table de vérité de la fonction OU exclusif(XOR).

- **Symbole graphique**

Figure 4.4: Symbole graphique la fonction OU exclusif(XOR).

3.2. Fonction logique NON ET (NAND)

L'opérateur NAND (NON ET) porte sur deux variables d'entrée, elle prend donc la valeur 1 lorsqu'au moins une de ses deux entrées est à 0. L'opérateur NAND est l'inverse de l'opérateur AND. Le tableau 4.7 suivant résume l'action de cet opérateur. Son expression algébrique est :

$$F(A,B) = \overline{A.B} = A \uparrow B$$

- **Table de vérité**

Entrée		**Sortie**
A	**B**	**F(A)**
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Tableau 4.7: Table de vérité de la fonction NON ET(NAND).

- **Symbole graphique**

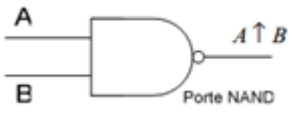


Figure 4.5: Symbole graphique la fonction OU exclusif(XOR).

3.3. Fonction logique NON OU (NOR)

L'opérateur NOR est l'inverse de l'opérateur OR. Elle prend donc la valeur 1 uniquement quand ses deux entrées sont à 0. Le tableau 4.8 suivant résume l'action de cet opérateur. Son expression algébrique est :

$$F(A,B) = \overline{A+B} = A \downarrow B$$

- **Table de vérité**

Entrée		Sortie
A	**B**	**F(A)**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

Tableau 4.8: Table de vérité de la fonction NON OU(NOR).

- **Symbole graphique**

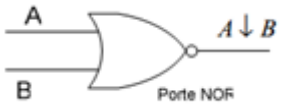


Figure 4.6: Symbole graphique la fonction NON OU(NOR).

3.4. NAND et NOR sont des opérateurs universels

En utilisant les NAND et les NOR on peut exprimer n'importe qu'elle expression (fonction) logique. Pour cela, Il suffit d'exprimer les opérateurs de base (NON , ET , OU) avec des NAND et des NOR.

3.4.1. Réalisation des opérateurs de base avec des NOR

$\overline{A} = \overline{A + A} = A \downarrow A$

$A + B = \overline{\overline{A + B}} = \overline{A \downarrow B} = (A \downarrow B) \downarrow (A \downarrow B)$

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

3.4.2. Réalisation des opérateurs de base avec des NAND

$\overline{A} = \overline{A.A} = A \uparrow A$

$A.B = \overline{\overline{A.B}} = \overline{A \uparrow B} = (A \uparrow B) \uparrow (A \uparrow B)$

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

3.4.3 Propriétés des opérateurs NAND et NOR

$A \downarrow 0 = \overline{A}$ $\qquad$ $A \uparrow 0 = 1$

$A \downarrow 1 = 0$ $\qquad$ $A \uparrow 1 = \overline{A}$

$A \downarrow B = B \downarrow A$ $\qquad$ $A \uparrow B = B \uparrow A$

$(A \downarrow B) \downarrow C \neq A \downarrow (B \downarrow C)$ $\qquad$ $(A \uparrow B) \uparrow C \neq A \uparrow (B \uparrow C)$

4. Schéma d'un circuit logique (Logigramme)

C'est la traduction de la fonction logique en un schéma électronique. Le principe consiste à remplacer chaque opérateur logique par la porte logique qui lui correspond.

Exemple 4.1

Soit la fonction: $\mathrm{F(A, B, C)} = \mathrm{A.B} + \overline{\mathrm{B}}\mathrm{C}$

La **Figure 4.7** présente le logigramme de la fonction F

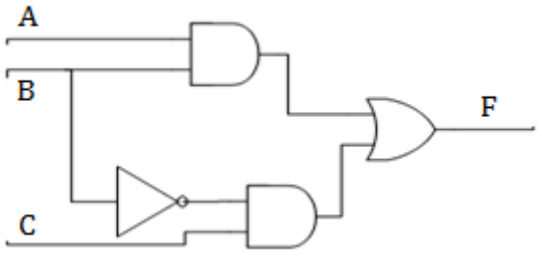


Figure 4.7: Logigramme de la fonction F.

5. Formes canoniques d'une fonction logique

Une fonction est exprimée dans une forme canonique si elle ne composé que des termes canoniques. Un terme algébrique est dit canonique s'il contient une occurrence de chacune de variables impliquées dans la fonction (les occurrences de la variable x est x et $\bar{x}$).

Exemple 4.2

La fonction $f(x, y) = \bar{x}y + x\bar{y}$ est une forme canonique. Les deux termes canoniques sont: $\bar{x}y$ et $x\bar{y}$.

Exemple 4.3

La fonction $f(x, y) = x + y$ est une forme canonique car le terme algébrique $(x + y)$ contient les deux variables impliquées dans la fonction.

Ils existent plusieurs formes canoniques : les plus utilisées sont la première et la deuxième forme.

5.1. Première forme canonique (Forme disjonctive)

Elle correspond à une somme de produits logiques, chacun des produits est appelé minterme, chacun des mintermes contient alors toutes les variables d'entrée sous une forme directe ou complémentée.

Exemple 4.4

$f(x, y, z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

5.2. Deuxième forme canonique (Forme conjonctive)

Elle correspond à une produit de sommes logiques, chacun des sommes est appelé maxterme, chacun des maxtermes contient alors toutes les variables d'entrée sous une forme directe ou complémentée.

Exemple 4.5

$f(x, y, z) = (\bar{x} + y + z)(x + \bar{y} + z)(x + y + \bar{z})(x + y + z)$

Remarque 4.1

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 règles de l'algèbre de Boole :

- Multiplier un terme avec une expression qui vaut 1
- Additionner à un terme avec une expression qui vaut 0
- Par la suite faire la distribution

Exemple 4.6

$f(x, y) = x + y$

$f(x, y) = x(y + \bar{y}) + y(x + \bar{x})$

$f(x, y) = (xy + x\bar{y}) + (yx + y\bar{x})$

$f(x, y) = xy + x\bar{y} + y\bar{x}$

Remarque 4.2

Il existe une autre représentation des formes canoniques d'une fonction, cette représentation est appelée forme numérique.

• R : pour indiquer la forme disjonctive
• P : pour indiquer la forme conjonctive.

Exemple 4.7

Si on prend une fonction avec 3 variables x, y et z

$R(2,4,6) = \sum(2,4,6) = R(010,100,110) = \bar{x}y\bar{z} + x\bar{y}\bar{z} + xy\bar{z}$

$P(0,1,3,4,5,7) = \prod(0,1,3,4,5,7) = P(000,001,011,101,111)$

$= (x + y +)(x + y + \bar{z})(x + \bar{y} + \bar{z})(\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + \bar{z})$

6. Simplification des fonctions logiques

La simplification est une partie importante de réalisation des circuits logiques. On doit simplifier le plus possible la fonction logique avant d'essayer d'implanter le circuit logique afin de réduire sa complexité, et son cout. Pour simplifier une expression d'une fonction logique trois méthodes sont utilisables :

– La Méthode algébrique(en utilisant des propriétés et des théorèmes d'algèbre de boole)
– Les Méthodes graphiques : (ex : table de karnaugh)
– Les méthodes programmables

6.1. Méthode algébrique

Le principe consiste à appliquer les règles de l'algèbre de Boole afin d'éliminer des variables ou des termes. Mais il n'y a pas une démarche bien spécifique. Nous allons traiter quelques exemples qui permettront de passer en revue la plupart des astuces utilisées pour mener à bien les simplifications.

Exemple 4.8 : ajouter un terme qui déjà existe
Soit la fonction F de trois variables x,y,z

$f(x,y,z) = xyz + \bar{x}yz + x\bar{y}z + xy\bar{z}$

$= xyz + \bar{x}yz + xyz + x\bar{y}z + xyz + xy\bar{z} = yz + xz + xy$

Exemple 4.9 : Suppression d'un terme superflu (un terme en plus),

Soit la fonction F de trois variables x,y,z

$$f(x,y,z) = xy + \bar{x}z + xz$$
$$= xy + \bar{x}z + xz(y + \bar{y})$$
$$= xy + \bar{x}z + xyz + x\bar{y}z$$
$$= xy(1 + z) + \bar{x}z(1 + x)$$
$$= xy + \bar{x}z$$

$$f(x,y,z) = (x + y)(\bar{y} + z)(x + z)$$
$$= (x + y)(\bar{y} + z)(x + z + y\bar{y})$$
$$= (x + y)(\bar{y} + z)(x + z + y)(x + z + \bar{y})$$
$$= (x + y)(x + z + y)(\bar{y} + z)(x + z + \bar{y})$$
$$= (x + y)(\bar{y} + z)$$

Exemple 4.10: simplification de la forme canonique ayant le nombre de termes minimum.

Soit la fonction F de trois variables x,y,z

$$f(x,y,z) = R(2,3,4,5,6,7)$$
$$\overline{f(x,y,z)} = R(O,1)$$
$$= \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z$$
$$= \bar{x}\bar{y}(\bar{z} + z)$$
$$= \bar{x}\bar{y}$$
$$= \overline{x + y}$$

Donc $f(x,y,z) = \overline{\overline{f(x,y,z)}}$

$$= \overline{\overline{x + y}}$$
$$= x + y$$

6.2. Méthode graphique (table de Karnaugh)

Le diagramme ou tableau de Karnaugh est un outil graphique sous forme matricielle qui permet de simplifier de façon graphique une fonction logique. les diagrammes de Karnaugh ne sont en pratique utilisables « à la main » que pour un nombre de variables inférieur ou égal à 6 , Figure 4.8. Cette méthode permet consiste à mettre en évidence par une méthode graphique tous les termes qui sont adjacents (qui ne différent que par l'état d'une seule variable).

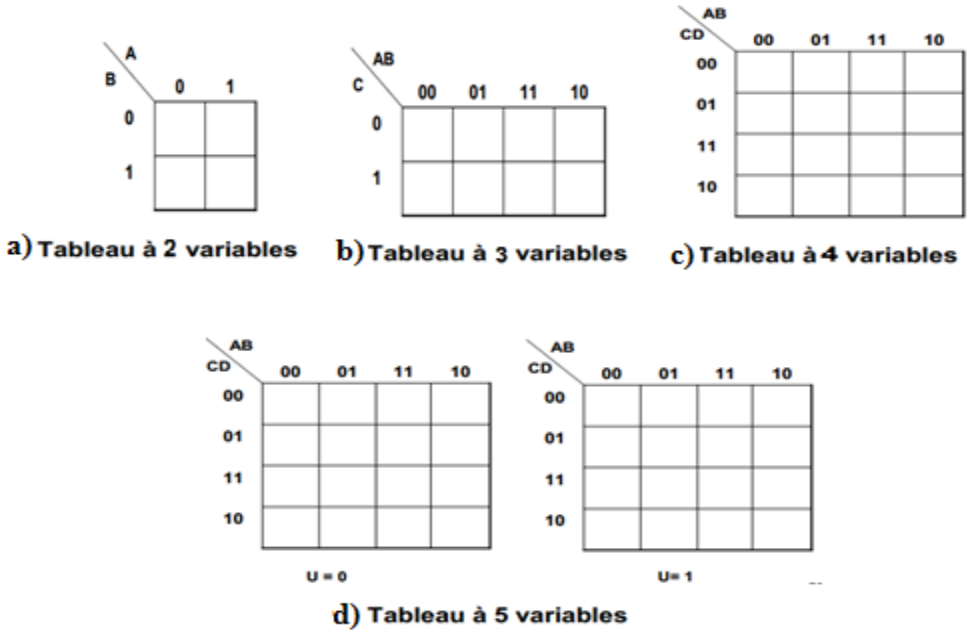


Figure 4.8: Tableaux de Karnaugh.

Un tableau de Karnaugh comportent 2^n cases (N est le nombre de variables), Les lignes et les colonnes d'une table de Karnaugh sont écrites dans l'ordre binaire réfléchi, ce qui met en évidence la propriété d'adjacence

Exemple 4.11:

Ces termes sont adjacents :

- $xy + \bar{x}y = y$
- $xyz + x\bar{y}z = xz$
- $xyzw + xy\bar{z}w = xyw$

Ces termes ne sont pas adjacents :

- $xy + \bar{x}\bar{y}$
- $xyz + \bar{x}\bar{y}z$
- $xyzx + \bar{x}\bar{y}zw$

Dans un tableau de karnaugh, chaque case possède un certain nombre de cases adjacentes. En disant que deux cases sont adjacentes quand leurs adresses le sont. Ainsi les cases de l'extrémité droite sont adjacentes à celles de l'extrémité gauche et les case de l'extrémité supérieure sont adjacentes à celles de l'extrémité inférieure comme le montre la figure 4.9 suivante.

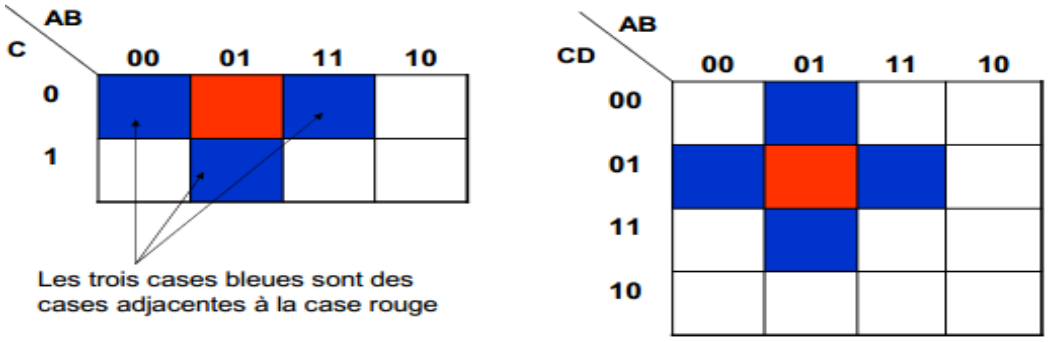


Figure 4.9: Les cases adjacentes dans le tableau de Karnaugh.

En résume, pour simplifier une fonction logique par la table de karnaugh il faut suivre les étapes suivantes :

1. Remplir le tableau à partir de la table de vérité ou à partir de la forme canonique.

2. Faire des regroupements : des regroupements de 16,8,4,2,1 cases (Les même termes peuvent participer à plusieurs regroupements) .

3. Dans un regroupement :

- Qui contient un seule terme on peut pas éliminer de variables.
- Qui contient deux termes on peut éliminer une variable (celle qui change d'état).
- Qui contient 4 termes on peut éliminer 2 variables.
- Qui contient 8 termes on peut éliminer 3 variables.
- Qui contient 16 termes on peut éliminer 4 variables.

5. L'expression logique finale est la réunion (la somme) des groupements après simplification et élimination des variables qui changent d'état.

Exemple 4.12:

Soit une fonction F (A,B,C) :

F(A,B,C)=1 si au moins deux variable sont à 1,

F(A,B,C)=0 dans tous les autres cas.

- **Table de vérité :** le Tableau 4.9 suivant résume les différentes valeurs de la fonction F

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tableau 4.9: Table de vérité.

- **Simplification par table de karnaught (Tableau 4.10)**

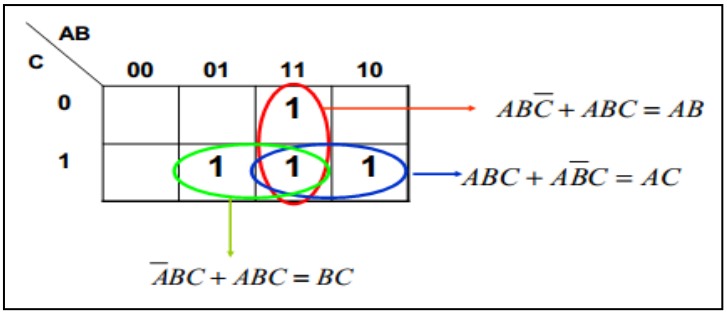


Tableau 4.10: Table de karnaught.

- L'expression F après la simplification

  $f(A,B,C) = AB + AC + BC$

Références

Références

J.R. Gregg et O.Zeros: « Understanding Boolean Algebra», Digital Circuits, and the Logic of Sets 1st Edition , Wiley & sons Inc. publishing, 1998, ISBN: 978-0-7803-3426-7.

B. H. Arnold : « Logic and Boolean Algebra», Dover publication, Inc., Mineola, New York, 2011, ISBN-13: 978-0-486-48385-6.

A.Cazes et J.Delacroix : « Architecture Des Machines Et Des Systèmes Informatiques : Cours et exercices corrigés », 3° édition, Dunod 2008.

M. AUMIAUS : « Logique binaire: fonctions logiques et arithmétique », 2eme édition DUNODE.

R. Strandh, I.Durand : « Architecture de l'ordinateur », Collection: Sciences Sup, Dunod 2005.

L. BOUZIDI: « Cours Structure Machine 1 », Univesité de Béjaya , 2018-2019.

N. KOBSI : « Cours Structure Machine 1 », Université Badji Mokhtar-Annaba, 2020-2021.

M.F KHALFI : « Cours Structure Machine », Université djilali liabès de SIDI BEL-ABBES, 2015-2016