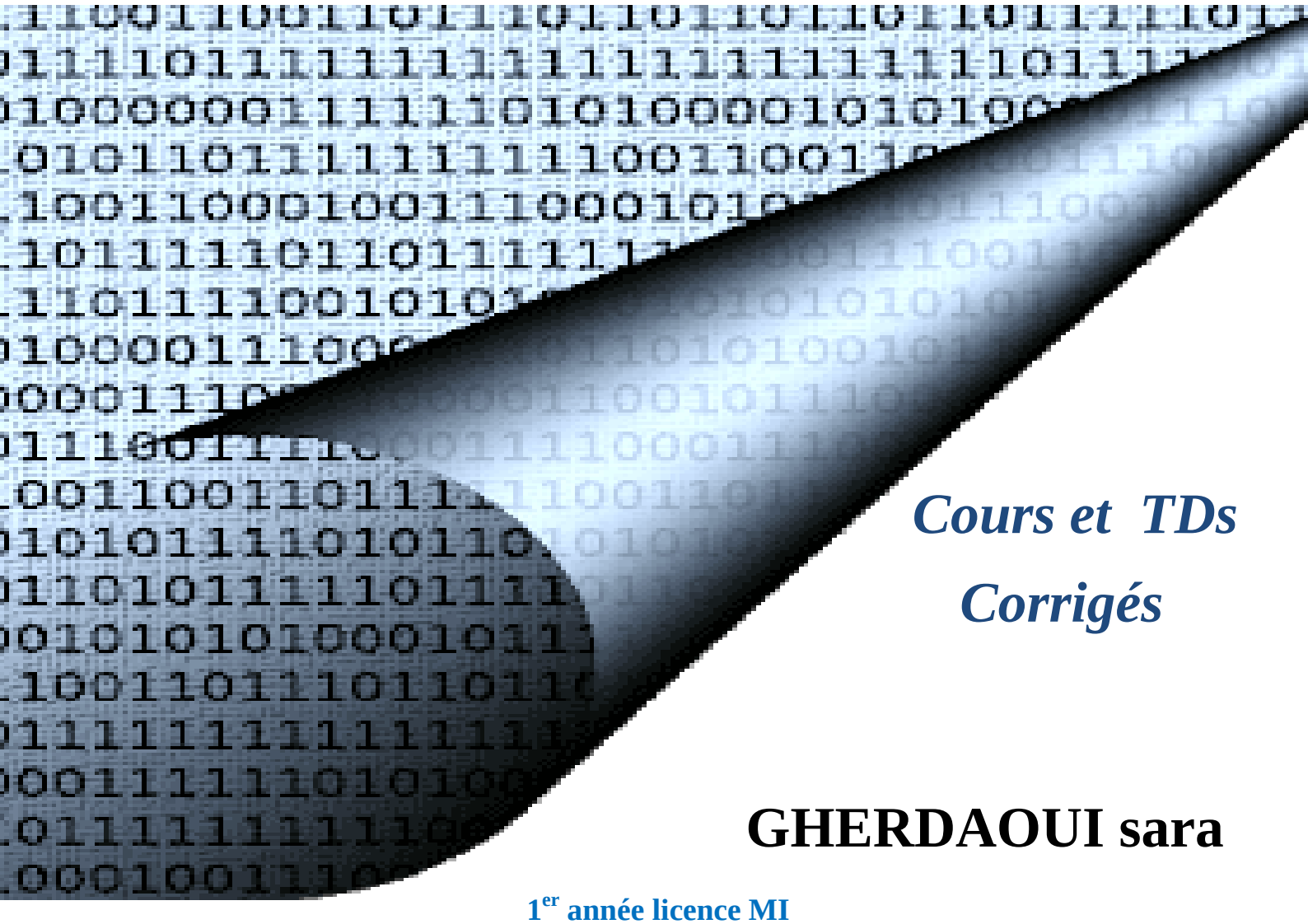
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur Et de la Recherche Scientifique
Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF
Faculté des Mathématiques et Informatique
Département d'Informatique



Cours et TDs
Corrigés

GHERDAOUI sara

1er année licence MI

Structure Machine 1

2020/2021

Avant-propos

Ce polycopié est destiné essentiellement aux étudiants LMD (1^{iere} année licence) socle commun Mathématique et Informatique et ceux des instituts et écoles de formation. Il constitue un support de cours et des travaux dirigés avec solution.

Dans ce cours, nous aborderons quelques concepts de base qui mènent à la conception des systèmes numériques. Nous traiterons, dans un premier temps, les systèmes de numération et le codage de tout type d'information. Puis, nous développerons l'Algèbre de Boole. Ces deux premiers chapitres, servirons pour aborder la conception de circuits logiques qui feront l'objet du module de structure machine 2 qui sera abordé au second semestre.

Généralités

1- Introduction

Sur une machine (Ordinateur, Tablette, Smartphone), toute l'information se trouve sous forme **numérique**, que ce soit dans la mémoire de masse (stockage), dans la mémoire vive, dans le microprocesseur et au niveau de tous les périphériques.

2- Structure d'un ordinateur

L'ordinateur est un appareil électronique programmable qui traite automatiquement les informations. Il est constitué de l'unité centrale et des périphériques.

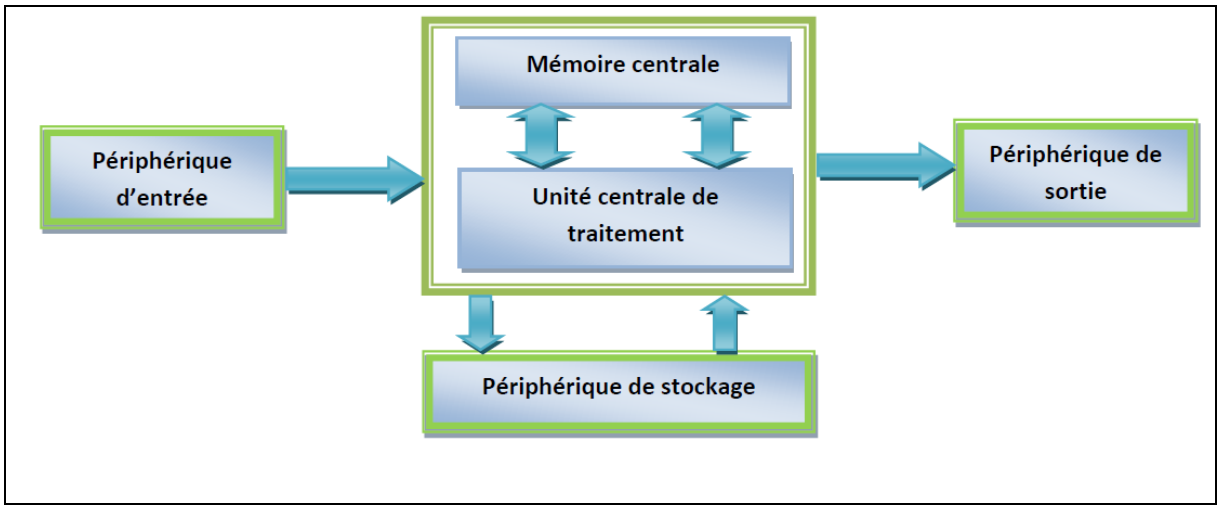


Figure 1. Structure d'un ordinateur.

2-1 Unité centrale

L'unité centrale est constituée de deux principaux blocs:

2.1.1- L'unité Centrale de traitement ou le processeur

Le processeur est le **cerveau** de l'ordinateur, il permet de traiter et de transmettre l'information[1]. Il est constitué de :

- ***L'unité de calcul*** : elle permet d'effectuer les opérations arithmétiques (addition, multiplication, division, soustraction) et les opérations logique (la comparaison, et, ou)
- ***L'unité de commande*** : elle permet de contrôler, gérer et organiser les travaux réalisés par le **CPU**.

2.1.2- Mémoire centrale

La mémoire centrale est un organe qui permet d'enregistrer, de stocker et de restituer les informations[1]. On distingue deux types :

- ***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 que 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.

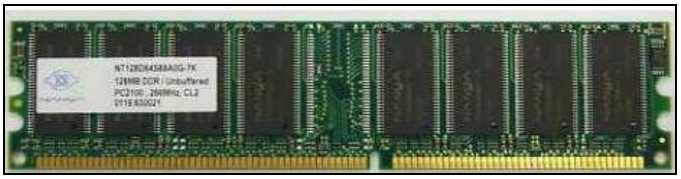


Figure 2. RAM.

- ***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.

Figure 3. ROM.

2.2- Périphériques :

On distingue trois types de Périphériques

2.2.1- 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…

2.2.2- 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
- Ecran
- Haut parleur …

2.2.3- 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**, …**

3- Nature de l'information

Une information est une entité abstraite, liée à la notion de connaissance. Nous nous intéressons naturellement aux informations d'un point de vue technique en informatique.

Les informations rencontrées sur les machines sont de natures différentes : principalement du texte, des images, du son, des vidéos, … et des programmes. Chaque type d'information fait l'objet de standards de codage, selon sa nature ou sa destination (stockage, utilisation, communication, …)[E2].

D'autre part certaines informations devront être **numérisées** avant d'être codées. Cela concerne toutes les informations portées par des grandeurs physiques :

- Les images (fixes ou vidéo) : intensité lumineuse, longueurs d'onde
- Les sons : pression (ou bien tension électrique après acquisition par un microphone)
- Et plus généralement toutes les grandeurs physiques …

4- Généralités sur le codage de l'information :

En informatique, Le codage de l'information s'effectue principalement en trois étapes[E2] :

- L'information sera exprimée par une suite de nombres (Numérisation)
- Chaque nombre est codé sous forme binaire (suite de 0 et 1)
- Chaque élément ou information binaire est représenté par un état physique (Arrêt / **Marche**, Eteint / **Allumé**, Non Chargé / **Chargé**, Fermé / **Ouvert**, Faux /**Vrai …**) représenté par **(0**/ **1)**

L'information élémentaire, dans un système binaire est le « Bit » (Binary Digit) ; un bit ne peut prendre que deux états distincts (0 ou 1).

Généralement, l'ordinateur manipule l'information sous forme d'un groupement de bits. Un groupement de huit bits est nommé « Octet »; le tableau suivant indique les unités de mesure d'une quantité d'informations binaires :

1 Kilo-octet = 1024 octets

1Méga-octet =1024 Ko

1Giga-octet = 1024Mo

1Téra- octet= 1024Go

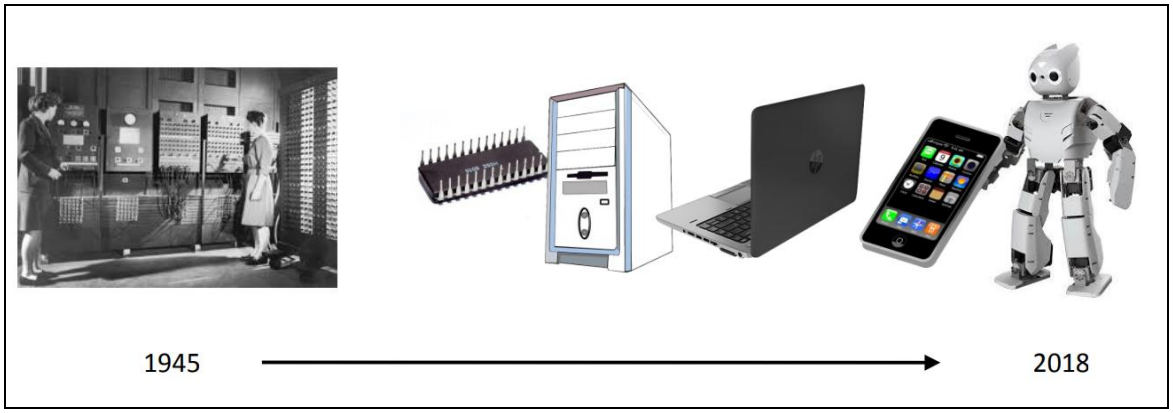


Figure 4. Evolution de la technologie des composants électroniques

Systèmes De Numération

1-Introduction

Tout système de numération possède une base **b** représentant le nombre de chiffres du système, **(b=0,1, …,b-1).**

Aujourd'hui, le système utilisé par les humains est le système de numération **décimal** *(base=10)* et utilise un maximum de **dix symbole** qui sont : **0,1,2,3,4,5,6,7,8,9.**

Un ordinateur marche avec du courant électrique et pour communiquer avec les composants électronique, on doit lui envoyer des signaux électriques.

Donc le système de numération utilisé par l'ordinateur (machine) est le système **binaire** *(base=2)* Et utilise un maximum de **deux symbole** qui sont **: 0,1**. **(1** représente **allumé** et **0** représente **éteint)**

Il existe d'autres systèmes de numération tels que

Le système de numération **octal** (*base = 8*) on utilise un maximum de **huit symboles qui** sont : **0,1,2,3,4,5,6,7**.
Le système de numération **Hexadécimal** (*base =16*) on utilise un maximum de **seize symboles** qui sont : **0,1,2,3,4,5,6,7,8,9,A ,B,C,D,E,F**.

2- Rang d'un chiffre de numération

Le système Décimale le système de numération le plus pratique actuellement[9].

- **L**e nombre **10** est la base de cette numération
- **C**'est un système positionnel. Chaque position possède un poids.

Exemple:
Le nombre N=**152** s'écrit sous la forme polynomiale comme suite : **N= 152** = **1** x 10^2 + **5** x 10^1 + **2** x 10^0 **N= 4134** = **4** x 10^3 + **1** x 10^2 + **3** x 10^1 + **4** x 10^0

Tout nombre **N** peut être décomposé en fonction de puissances entières de la base. Donc on peut écrire N sous la forme polynomiale:

$$N = a_n . b^n + a_{n-1} . b^{n-1} + a_{n-2} . b^{n-2} + \dots + a_0 . b^0$$

$N = \sum_0^n a_n * b^n$ où $a_n \in$ 0,1…….n

Tel que :

n : rang du chiffre a_n

a_n: le chiffre de poids le plus fort (Most Significant Bit : MSB)

a_0 : le chiffre de poids le plus faible (Least Significant Bit : LSB).

b : La base

La valeur en décimal d'un nombre **n** de base **b** quelconque s'obtient en effectuant les opérations de l'expression de sa forme polynomiale.

Exemple:

Soit à déterminer la valeur décimale des nombres suivants N1=$(30212)_4$ et N2=$(1011)_2$

- N1=$(30212)_4 = 3x4^4 + 0x4^3 + 2x4^2 + 1x4^1 + 2x4^0$
  $= (806)_{10}$
- N2=$(1011)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x\,2^0$
  $= (11)_{10}$

3- Représentation des nombres Entiers non signé (Conversion entre système)

Conversion de base est l'opération qui permet de passer de la représentation d'un nombre **N** exprime dans une base **b1** à la représentation du même nombre mais exprime dans une autre base **b2**.

On cite les conversions suivantes:

- **Codage** : il consiste à passer du Décimal vers Binaire, Octale, Hexadécimale, …
- **Décodage** : il consiste à passer du Binaire, Octale, Hexadécimale vers Décimal
- **Transcodage** : il consiste à passer d'une base **b ≠ 10** vers une base **b ≠ 10**

3.1- Codage

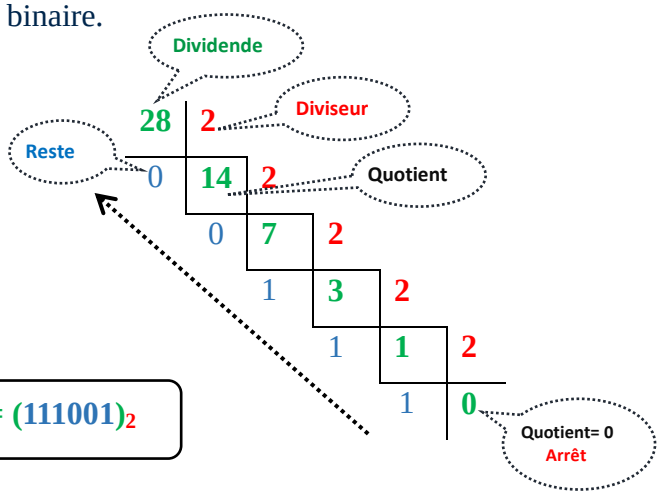
Le **codage** permet la conversion de la base du **Décimal** vers une autre **base b** quelconque**.** La règle à suivre est comme suite :

1. La conversion s'effectue par des divisions entières successives du nombre par la **base b**
2. Puis on divise le quotient par la **base b** et ainsi de suite
3. La condition d'arrêt correspond à un quotient nul
4. La suite des restes correspond aux symboles de la base visée.
5. On obtient en premier le chiffre de **poids faible** et en dernier le chiffre de **poids fort** (le nombre dans la **base b** est obtenu en lisant les restes du dernier vers le premier « *de bas en haut* »)

Exemple 1:

- Convertir le nombre **N= $(28)_{10}$** en binaire.

28 : 2= 14 reste 0
14 : 2= 7 reste 0
7: 2= 3 reste 1
3: 2= 1 reste 1
1: 2= 0 reste 1

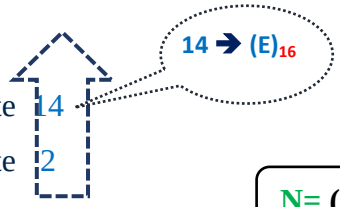


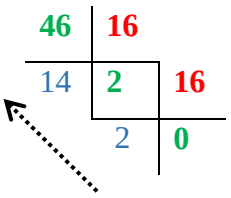
N= $(111001)_2$

Exemple 2:

- Convertir le nombre **N=$(740)_{10}$** en hexadécimal.

46:16= 2 reste 14
3:16= 0 reste 2





N= $(2E)_{16}$

3.2- Décodage :

Le **Décodage** permet d'effectuer la conversion d'un nombre **N** de **base b** quelconque vers la **base Décimale.**

Cette conversion s'obtient en effectuant les opérations de l'expression de la forme polynomiale de **N**.

Exemples 1 :

Déterminer la valeur décimale du nombre $N= (213)_6$

$(213)_6 = 2x6^2 +1x6^1+3x6^0 = 81$

$N = (81)_{10}$

Exemples 2 :

Déterminer la valeur décimale du nombre $N= (1011)_2$

$(1011)_2 = 1x2^3 +1x2^2 +0x2^1+1x2^0 = 11$

$N = (11)_{10}$

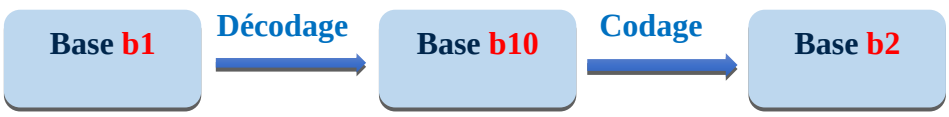
3.3- Transcodage :

Le **Transcodage** permet d'effectuer la conversion d'un nombre **N** de **base b1** quelconque tel que **b1 ≠ 10** vers une autre **base b2** tel que **b2 ≠ 10.**

On a deux types de transcodage :

3.3.1-Transcodage Indirect *(cas général)*:

Cette méthode est valable quel que soit **b1**et **b2.** Un Transcodage Indirect passe d'abord par un décodage suivi d'un codage



3.3.2. Transcodage Direct (*cas particulier*):

Cette méthode s'applique lorsque la base **b1** est une **puissance de 2 (b1=2^n)** et la base **b2** correspond à **2.**

Chaque chiffre de nombre **N** sera exprimé sous la forme d'un nombre binaire ayant **n** bits.

Exemple 1:

N1 = $(354)_8$ ➔ $8=2^3$ ➔ éclatement sur 3 bits

$(3)_{10} = (011)_2$
$(5)_{10} = (101)_2$
$(4)_{10} = (100)_2$

N1 = $(354)_8$ = $(011\ 101\ 100)_2$

Remarque:

Pour passer d'un nombre **N** dans la base **b1=2** à une base **b2 ≠ 10** mais qui est une puissance de **2 (b1=2^n)** on regroupe les n bits par n pour obtenir **N** codé dans la base **b1.**

Exemple 2:

N2 = $(011101100110)_2$

$16=2^4$ ➔ regroupement de 4 bits

N2 = $(011101100110)_2$ = (0111 0110 0110)$_2$ = $(766)_{16}$

7 6 6

$4=2^2$ ➔ regroupement de 2 bits

N2 = $(011101100110)_2$ = (01 11 01 10 01 10)$_2$ = $(131212)_4$

1 3 1 2 1 2

$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	2	Base / Décimale
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
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

Tableau 1 : Tableau récapitulatif de transcodage des bases 4, 8, 16.

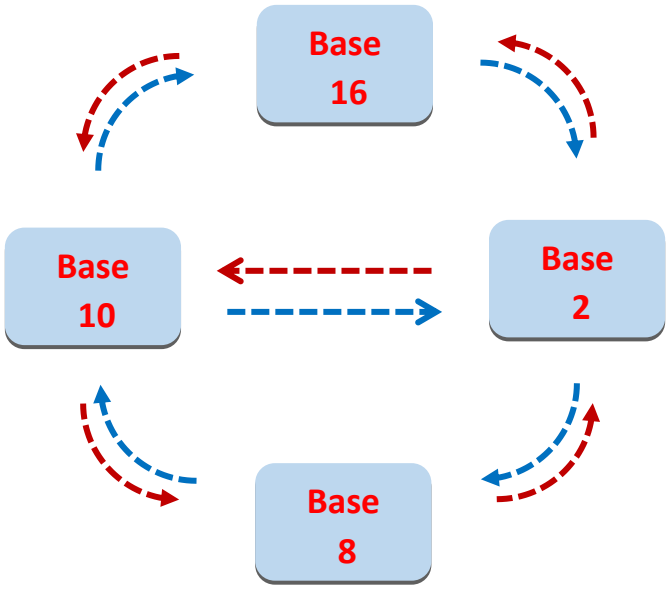


Figure 5. Schéma récapitulatif des conversions possibles entre les systèmes de numération de base

4-Représentation des nombres Fractionnaires non signés

De même pour les nombres entiers, il existe trois types de conversion des nombres fractionnaires :

- **Codage**
- **Décodage**
- **Transcodage**

4.1- Codage :

Pour convertir un nombre fractionnaire **N** en base **b** quelconque, il faut traiter séparément les Parties Entière (**PE**) et Partie Fractionnaire (**PF**).

- **Partie Entière (PE):** Il faut procéder par des divisions successives par la base **b** comme pour le codage des nombres entiers.

- **Partie Fractionnaire(PF):** Il faut procéder par des multiplications successives par la base **b**. Les parties entières successives constituent le résultat écrit dans l'ordre des puissances croissantes de la base.

On arrête les multiplications quand la précision voulue est obtenue.

Remarque:

Donc le nombre dans la **base b** est obtenu en lisant les *Parties Entières* du premier vers le dernier (*de haut en bas*).

Exemples 1 :

Convertir le nombre $N = (4,57)_{10}$ en binaire.

- **Partie Entière:** $(4)_{10}$

4	2	
0	2	2
0	1	2
1	0	

➔ $(4)_{10} = (100)_2$

- **Partie Fractionnaire:** $(0,57)_{10}$

0,57 x 2= 1, 14
0,14 x 2= 0, 28
0,28 x 2= 0, 56

Partie Entière du résultat est égale à 1 On prend la partie fractionnaire 0.14 et on la multiplie par la base (ici b=2) et ainsi de suite.

➔ $N = (4,57)_{10} = (110,100)_2$

Exemples 2 :

Soit à convertir en **octal** (b=8) le nombre décimal $N = (0,732)_{10}$

- **Partie Entière:** $(0)_{10} = (0)_8$
- **Partie Fractionnaire:** $(0,732)_{10}$

0,732 x 8 = 5, 856
0,856 x 8 = 6, 848
0,848 x 8 = 6, 784
0,784 x 8 = 6, 272
0,272 x 8 = 2, 176

➔ $N = (0,732)_{10} = (0,56662)_8$

4.2- Décodage :

Pour réaliser le **Décodage** d'un nombre fractionnaire **N** de **base b** quelconque vers la **base 10,** on effectue les opérations de l'expression de la forme polynomiale de **N**.

Exemple 1:

Soit à déterminer la valeur décimale du nombre N

- $N = (11,101)_2 = (1x2^1+1x2^0, 1x2^{-1} + 0x2^{-2} + 1x2^{-3})_{10}$

$$N = (3,625)_{10}$$

- $N = (20,41)_8 = (2x8^1 + 0x8^0 + 4x8^{-1} + 1x8^{-2})_{10}$

$$N = (16,515625)_{10}$$

4.3-Transcodage :

Le Transcodage consiste à passer d'une base b1 quelconque tel que $b1 \neq 10$ vers une autre base b2 tel que $b2 \neq 10$.

Soit à réaliser une Conversion d'un nombre binaire (b=2) vers une base b tel que $b = 2^n$

Règle :

1°/ A partir de la virgule, grouper les bits par groupes de **n** en allant vers la gauche pour la partie entière et vers la droite pour la partie fractionnaire.

2°/ Convertir ensuite chaque bloc séparément en base b selon le code binaire naturel**.**

Partie Entière , Partie Fractionnaire

Exemple 1 :

- Binaire en octal

Soit à convertir en octal le nombre binaire $N= (111011101,000100)_2$

$8=2^3$ ➔ regroupement de 3 bits

$N = (111011101,000100)_2 = (\underbrace{111}_{7}\ \underbrace{011}_{3}\ \underbrace{101}_{5}\ ,\ \underbrace{000}_{0}\ \underbrace{100}_{4})_2 = (735,04)_8$

Exemple 2:

- Octal en binaire

Soit à convertir en binaire le nombre en octal **N= $(57,146)_8$**

$8=2^3$ ➔ éclatement sur 3 bits

$(5)_{10} = (101)_2$
$(7)_{10} = (111)_2$
$(1)_{10} = (001)_2$
$(4)_{10} = (100)_2$
$(6)_{10} = (110)_2$

N = $(57,146)_8$ = (101 111 , 001 100 110) $_2$

Exemple 3 :

- **B**inaire en Hexadécimal

Soit à convertir en Hexadécimal le nombre binaire **N= $(1110011101,01110001)_2$**

$16=2^4$ ➔ regroupement de 4 bits

N = $(1110011101,01110001)_2$ = (0011 1001 1101 , 0111 0001)$_2$ = $(39D,71)_{16}$

3 9 D , 7 1

5- Operations Arithmétiques en Binaire

Les opérations arithmétiques (addition, soustraction, multiplication et division) en systeme binaire, sont similaires à celle du système décimal. La seule différence est que le système de nombres décimaux comprend le chiffre de 0 à 9, alors que le système de nombres binaires ne comprend que deux chiffres (0 et 1)[3],

5.1- Addition en Binaire :

L'addition en Binaire est l'opération qui sert à calculer une somme entre deux nombres binaires **a** et **b**.

Règles de l'addition :

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

Exemple :

Effectuer l'addition suivante en binaire $(110111)_2 + (100110)_2$

Retenue : 1 1

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

$(110111)_2 + (100110)_2 = (1011101)_2$

5.1- Soustraction en Binaire :

La soustraction en binaire est l'opération qui permet de calculer la différence entre deux nombres binaires **a** et **b**.

Règles de Soustraction :

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

Exemple :

Effectuer la soustraction suivante en binaire $(1100111)_2 - (101100)_2$

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

$(1100111)_2 - (101100)_2 = (0111011)_2$

5.3- Multiplication en Binaire:

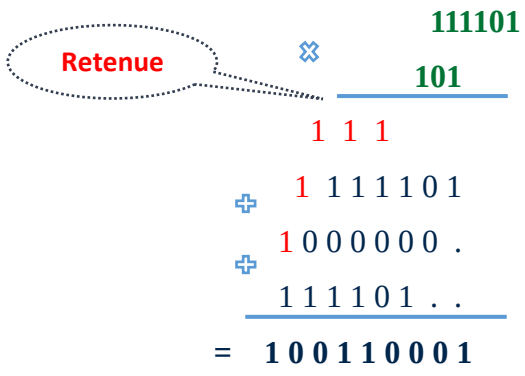
La multiplication binaire est une opération qui, à partir de deux nombres binaires **a** et **b**, donne un autre nombre binaire appelé produit. La multiplication permet d'éviter une addition répétée.

Règles de Multiplication :

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

Exemple :

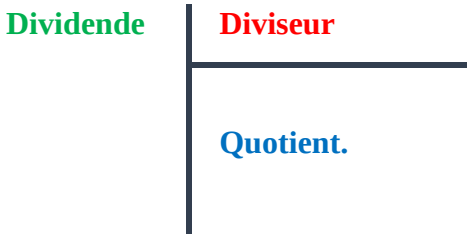
Effectuer la multiplication suivante en binaire $(111101)_2$ X $(101)_2$



$(111101)_2$ X $(101)_2$ = $(100110001)_2$

5.4- Division en Binaire :

La **division** en binaire est une opération qui à deux nombres binaires ***a*** et ***b*** *(***Dividende** et **Diviseur***)*, associe un troisième nombre en binaire, appelé **Quotient.**

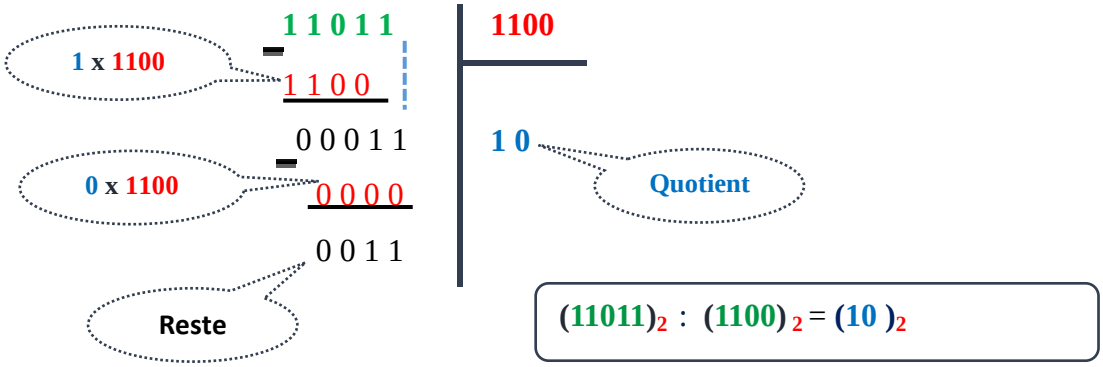


Remarque:

Si le **dividende** est supérieur ou égale au **diviseur** alors le quotient est égal à **1** sinon il est égal à **0**

Exemple :

Effectuer la division suivante en binaire $(11011)_2 : (1100)_2$



FICHE TD1

(Systèmes de numérotation)

Exercice 1 :

Convertissez le nombre décimal (210) dans les bases 2, 4, 8,16

Exercice 2 :

Décodez les entiers suivants :

$(101011)_2$, $(301)_4$, $(101)_6$, $(G7)_{16}$, $(65)_8$

Exercice 3 :

Effectuez les conversions suivantes en utilisant le transcodage direct :

- $(1010111010)_2 = (\quad)_8 = (\quad)_{16}$
- $(\text{F04})_{16} = (\quad)_2 = (\quad)_4$
- $(\text{23})_4 = (\quad)_2 = (\quad)_{16} = (\quad)_8$

Exercice 4:

a) Donner la valeur décimal de : $(1001{,}011)_2$, $(10{,}4)_8$

b) Donner la valeur binaire de : $(7D1{,}E1)_{16}$, $(16{,}4)_8$

Exercice 5 :

1) Calculer en binaire :

   a) $(11001)_2 + (1111)_2$
   b) $(16)_{16} + (20)_8$
   c) $(30)_{10} - (15)_{10}$
   d) $(101101)_2 - (1011)_2$
   e) $(25)_8 * (31)_4$
   f) $(111010)_2/(5)_{10}$

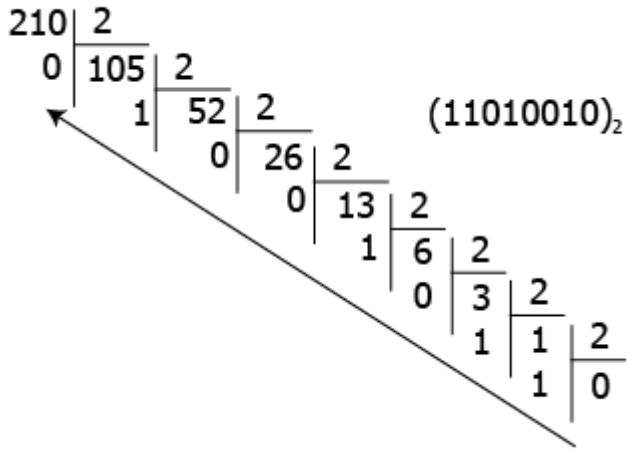
2) Effectuer l'addition suivante sur 1 octet

$(145)_{10} + (111)_{10}$

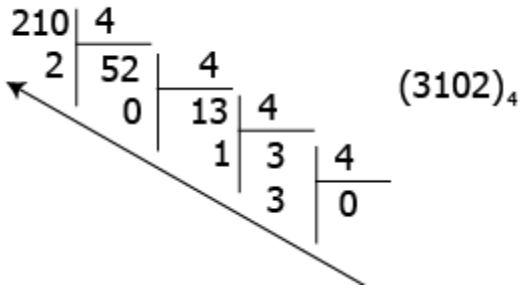
SOLUTION

Exercice 1:

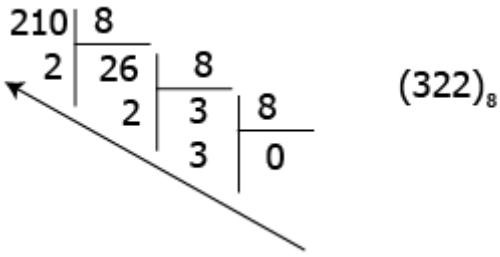
Base 2 :



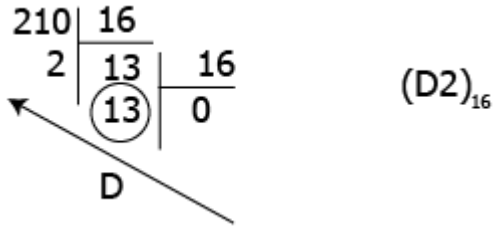
Base 4 :



Base 8 :



Base 16 :



Exercice 2 :

$(1\ 0\ 1\ 0\ 1\ 1)_2 = 1 . 2^0 + 1 . 2^1 + 0 . 2^2 + 1 . 2^3 + 0 . 2^4 + 1 . 2^5$
$= 1 . 1 + 1 . 2 + 0 . 4 + 1 . 8 + 0 . 16 + 1 . 32$
$= 1 + 2 + 0 + 8 + 0 + 32$
$= (43)_{10}$

$(301)_4 = 1 . 4^0 + 0 . 4^1 + 3 . 4^2$
$= 1 . 1 + 0 . 4 + 3 . 16$
$= 1 + 0 + 48$
$= (49)_{10}$

$(101)_6 = 1 . 6^0 + 0 . 6^1 + 1 . 6^2$
$= 1 + 0 + 36$
$= (37)_{10}$

$(G7)_{16} = 7 . 16^0 + G . 16^1$
$= 7 . 1 + 12 . 16$
$= 7 + 192$
$= (199)_{10}$

$(65)_8 = 5 . 8^0 + 6 . 8^1$
$= 5 . 1 + 6 . 8$
$= 5 + 48$
$= (53)_{10}$

Exercice 3:

$(1010111010)_2 = (1272)_8 = (2BA)_{16}$

$(F04)_{16} = (111100000100)_2 = (330010)_4$

$(23)_4 = (1011)_2 = (B)_{16} = (13)_8$

Exercice 4:

a) $(1001,011)_2 = (2^3+2^0+2^{-2}+2^{-3})$
$=8+1+1/2^2+1/2^3$
$=9+0.25+0.125$
$=(9.375)_{10}$

$(10,4)_8 = 1.8^1 + 4.8^{-1}$
$= 8+0.5$
$= (8.5)_{10}$

b) $(7D1,E1)_{16} = (0111\ 1101\ 0001,1110\ 0001)_2$

$(16,4)_8 = (001110,100)_8$

Exercice 5:

1)

a)
$$\begin{array}{r} 11001 \\ +\ 1111 \\ \hline \end{array}$$
$(101000)_2$

b) $(16)_{16} = (00010110)_2$
$(20)_8 = (010000)_2$

$$\begin{array}{r} 00010110 \\ +\ \ 010000 \\ \hline \end{array}$$
$= (00100110)_2$

c) $(30)_{10} = (11110)_2$
$(15)_{10} = (1111)_2$

$$\begin{array}{r} 11110 \\ -\ 1111 \\ \hline \end{array}$$
$= (01111)_2$

d)
$$\begin{array}{r} 101101 \\ -\ \ 1011 \\ \hline \end{array}$$
$= (100010)_2$

e) $(25)_8 = (010101)_2$
$(31)_4 = (1101)_2$

$$\begin{array}{r} 010101 \\ *\ \ \ 1101 \\ \hline 10101 \\ +\ \ 00000\ . \\ +\ \ 10101\ . . \\ +\ 10101\ . . . \\ \hline \end{array}$$
$= (100010001)_2$

f) $(5)_{10} = (101)_2$

```
111010  | 101
101     |-----
------  | 1011
0100
- 000
-----
 1001
 - 101
 -----
  1000
  - 101
  -----
  0011
```

2) $(145)_{10} + (111)_{10}$

$(145)_{10} = (10010001)_2$
$(111)_{10} = (01101111)_2$

```
  10010001
+ 01101111
----------
1 00000000
```

8 bits $\Longrightarrow$ dépassement de capacité

(145 + 111) = 256

256 = $(100000000)_2$

9 bits

Représentation de nombres binaires signés

1- Introduction:

Dans la représentation des nombres binaires signés (positifs ou négatifs) sont représentés uniquement en 0 et 1.

Un nombre positif ne change pas, il garde la même représentation, par contre le signe négatif précédant les valeurs entières pose un problème de représentation sur machine.

Le bit le plus significatif indique le signe du nombre **0 pour positif** et **1 pour négatif**.

2- Approches de représentation des Nombres entiers signés :

Il y a trois approches sont utilisées pour représenter les nombres entiers signés sur un ordinateur: Signe et Valeur Absolue **(SVA)**, Complément à un **(CP1)** et Complément à deux **(CP2)**[3].

2.1- Approche signe et valeur absolue (SVA) :

Pour faire la distinction d'un nombre entier positif et un nombre entier négatifs:

- La valeur absolue des entiers positifs est précédée d'un **0.**
- La valeur absolue des entiers positifs est précédée d'un **1.**

Exemple :

Pour la représentation signe et valeur absolue **SVA** sur **4bits** *de* +7 et – 7

$7 = (111)_2$ ➡

+7 s'écrit $(0111)_{SVA}$

– 7 s'écrit $(1111)_{SVA}$

2.2- Approche complément à un (CP1) :

Pour trouver le **complément à 1** d'un nombre il suffit d'inverser les bits de ce nombre

- Si le bit est **0**, il faut mettre à sa place un **1**
- Si le bit est **1**, il faut mettre à sa place un **0**

Exemple:

le complément à 1 **(CP1)** sur **4bits** de +7 et – 7 :

7 = $(111)_2$

7 sur **4bits** = 0111

– 7 s'écrit $(1000)_{CP1}$ (Signe - on inverse)

+ 7 s'écrit $(0111)_{CP1}$ (Signe + on n'inverse pas)

2.3- Approche complément à deux (CP2) :

Pour trouver le **complément à 2** d'un nombre il suffit de chercher son complément à 1 **(CP1)** auquel on ajoute **1**.

CP2(X) = CP1(X) + 1

Exemple:

le complément à 2 sur **4bits** de+7 et – 7 :

7 = $(111)_2$

7 sur **4bits** = 0111

+ 7 s'écrit $(0111)_{CP2}$ (Signe + on ne change rien)

– 7 s'écrit $(1001)_{CP2}$ (Signe - CP2(-7) = CP1(-7) + 1)

CP1(-7)+1= 1000 +1 = **1001**

Remarque:

La représentation en **CP2** est la plus utilisée pour la représentation des nombres négatifs.

L'intervalle de représentation **des nombres binaires signés** avec un nombre de **n bits** est

Signe et Valeur Absolue (**SVA**)	**$[-(2^{n-1}-1), +(2^{n-1}-1)]$**
Complément à 1 (CP1)	**$[-(2^{n-1}-1), +(2^{n-1}-1)]$**
Complément à 2 (CP2)	**$[-(2^{n-1}), +(2^{n-1}-1)]$**

Exemple

Sur 4 bits :

	SVA		CP1		CP2	
+N	+7	0111	+7	0111	+7	0111
	+6	0110	+6	0110	+6	0110
	+5	0101	+5	0101	+5	0101
	+4	0100	+4	0100	+4	0100
	+3	0011	+3	0011	+3	0011
	+2	0010	+2	0010	+2	0010
	+1	0001	+1	0001	+1	0001
	+0	0000	+0	0000	+0	0000
-N	-0	1000	-0	1111	-1	1111
	-1	1001	-1	1110	-2	1110
	-2	1010	-2	1101	-3	1101
	-3	1011	-3	1100	-4	1100
	-4	1100	-4	1011	-5	1011
	-5	1101	-5	1010	-6	1010
	-6	1110	-6	1001	-7	1001
	-7	1111	-7	1000	-8	1000

Remarque:

Pour les trois méthodes il faut préciser le nombre de bits de représentation.

3- Opérations en CP1 et CP2 :

3.1- Opérations arithmétique en CP1 :

- **1er Cas:**

A>0 et B>0 ⟹ **A+B**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 1

soit A= +6 et **B= -4**

A	6	0110
+ B	4	0100
	= 10	= 1010

- **2éme Cas:**

| A | > | B | ⟹ **A + CP1(B)**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 1

soit A= +6 et **B= -4**

A 0 1 1 0

+ CP1(B) 1 0 1 1

= **1 0 0 0 1**

= **0 0 1 0** **Résultat**

- **3éme Cas:**

| A | < | B | ⟹ **A + CP1(B)** = - CP1[A + CP1(B)]

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 1

soit A=+5 et **B = -7**

A 0 1 0 1

+ CP1(B) 1 0 0 0

1er Résultat = **1 1 0 1**

On doit calculer le **CP1(1er Résultat)** donc CP1(1101) = 0010 d'où **Résultat = - 2**

- **4éme Cas:**

A<0 et B<0 ⟹ **CP1(A) + CP1(B)= - CP1[CP1(A)+ CP1(B)]**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 1

Soit A= -5 et **B= - 7**

CP1(A)	1 0 1 0	
+ CP1(B)	1 0 0 0	
=	**1 0 0 1 0**	
	1	
1er Résultat	**= 0 0 1 1**	**Résultat = -CP1[1er Résultat]= -12**

3.2- Opérations arithmétique en CP2 :

- **1er Cas:**

A > 0 et B > 0 ⟹ **A+B**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 2

Soit **A= +6** et **B = +4**

A	6	0110
+ B	4	0100
	= 10	**= 1010**

- **2éme Cas:**

| A | > | B | ⟹ **A + CP2(B)**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 2

soit A= +6 et **B= -4**

A	0 1 1 0
+ CP2(B)	1 1 0 0
=	**1 0 0 1 0**

Retenue à ignorer

Donc le **Résultat est 0 0 1 0 = +2**

- **3éme Cas:**

  | A | < | B | ⟹ **A + CP2(B) = - CP2[A + CP2(B)]**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 2

soit A=+5 et **B = -7**

A	0 1 0 1
+ CP2(B)	1 0 0 1
1er Résultat	**= 1 1 1 0**

On doit calculer le **CP2(1er Résultat)** donc CP2(1110) = 0010 d'où **Résultat = - 2**

- **4éme Cas:**

  A< 0 et B <0 ⟹ **CP2(A) + CP2(B) = - CP2[CP2(A)+ CP2(B)]**

Exemple:

Effectuer l'opération sur **4bits** en utilisant le complément à 2

Soit A= -5 et B= - 7

CP2(A)	1 0 1 1
+ CP2(B)	1 0 0 1
=	**1** 0 1 0 0

Retenue à ignorer

le **Résultat est 0 1 0 0**

FICHE TD 2

(Nombres Signés)

Exercice 1 :

Convertir en SVP, Cp1 et en Cp2 sur 8 bits les nombres suivants :
(-7), (-107), (+54), (-135).

Exercice 2 :

Indiquer la valeur décimal des codes binaires de 8 bits 00110110, 11011011 dans le cas où ils sont codes en SVA en Cp1 et en Cp2.

Exercice 3

Effectuer les opérations suivantes sur 8 bits en Cp1 et en Cp2 :

* $(10010110)_{cp1} + (11111011)_{cp1}$

* $(100)_{10} - (36)_{10}$

Solution

Exercice 1 :

1) $(-7)_{10} = -(111)_2 \Longrightarrow$ sur 8 bits

0	0	0	0	0	1	1	1

* SVA :

1	0000111

\- 7

* Cp1 :

1	1	1	1	1	0	0	0

* Cp2 :

1	1	1	1	1	0	0	1

2) $-(107) = -(1101011)_2 \Longrightarrow$ sur 8 bits

0	1	1	0	1	0	1	1

* SVA :

1	1101011

\- 107

* Cp1 :

1	0	0	1	0	1	0	0

* Cp2

1	0	0	1	0	1	0	1

3) $+(54) = +(110110)_2 \Longrightarrow$ sur 8 bits

0	0	1	1	0	1	1	0

* SVA :

0	0110110

\+ 107

* Cp1 :

0	0	1	1	0	1	1	0

* Cp2

0	0	1	1	0	1	1	0

Positif reste le même

4) $-(135) = -(10000111)_2 \Longrightarrow$ sur 8 bits

1	0	0	0	0	1	1	1

↳ Le 135 s'écrit sur 8 bits donc ne peut pas être code en SVA
Cp1 et Cp2 sur 8 bits (dépassement de capacité).

- Intervalle de représentation en SVA et en Cp1 sur 8 bits est (-127 ,127) (-135 n'est pas inclus).
- Intervalle de représentation en Cp2 sur 8 bits est (-128,-127) (-135 n'est pas inclus).

Exercice 2 :

1) SVA

a)

0	0110110

* signe = ''0'' = ''+''
* $(0110110)_2 = 54$

$\Longrightarrow (00110110)_{sva} = (+54)_{10}$

b) | 1 | 1011011 |

* signe = ‘’1’’ = ’’- ’’

* $(1011011)_2 = 91$

$\Rightarrow (11011011)_{sva} = (-91)_{10}$

2) Cp1

a) 00110110 MSB =0 $\Rightarrow$ nombre positif

$\Longrightarrow +(00110110)_2 = (+54)_{10}$

b) 11011011 MSB =1 $\Rightarrow$ nombre négatif

$(11011011)_{cp1} = -(00100100)$

$= (-36)_{10}$

3) Cp2

a) 00110110 MSB =0 $\Rightarrow$ nombre positif

$\Longrightarrow +(00110110)_2 = (+54)_{10}$

b) 11011011 MSB =1 $\Rightarrow$ nombre négatif

$(11011011)_{cp2} = -$ [cp1(11011011) + 1]

$= -$ [(00100100)+1] = $-(00100101) = (-37)_{10}$

Exercice 3 :

1) En Cp1

a) 10010110 $(10010110)_{cp1} = -105$

+ 11111011 $(11111011)_{cp1} = -4$

(1)10010001 $-(109)$

\+ 1

$= (10010010)_{cp1}$ MSB =1 ⇨ nombre négatif

$= -(01101101)_2 = -(109)_{10}$

b) 100 – 36 = 100 + (-36)

$(100)_{10} = (01100100)_2 = (01100100)_{cp1}$

$(-36)_{10} = -(00100100)_2 = (11011011)_{cp1}$

01100100

+ 11011011

(1)00111111

1

=(01000000) MSB =0 ⇨ nombre positif

= +(64)

2) En Cp2

a) $(10010110)_{cp1}$ ⇨ $(\underbrace{10010110 + 1}_{cp1})_{cp2}$

$= (10010111)_{cp2}$

$(11111011)_{cp1}$ ⇨ $(11111011 + 1)_{cp2}$

$= (11111100)_{cp2}$

10010111

\+ 11111100

=1(10010011)$_{cp2}$ MSB = 1 ⇨ ombre negatif

retenue

ignorée

$(10010011)_{cp2} = -[(01101100)_{cp1} + 1]$

$= -(01101101)_2$

$= -(109)_{10}$

b) 100 – 36 = 100 + (-36)

$(100)_{10} = (01100100)_{cp2}$

$(-36) = (11011100)_{cp2}$

01100100

\+ 11011100

1(01000000)$_{cp2}$ MSB = 0 ⇨ nombre positif

retenue

ignorée

$(01000000)_{cp2} = +(64)$

Représentation des nombres Réels

1- Introduction :

Un nombre réel est constitué de deux parties séparées par une virgule.

- Partie Entière **(PE)**
- Partie Fractionnaire **(PF)**

Nombre Réel =Partie Entière, Partie Fractionnaire

Pour indiquer à la machine la position de la virgule, Il existe deux méthodes :

- Représentation en **virgule fixe** ⇨ *la virgule a une position fixe.*
- Représentation en **virgule flottante** ⇨ *la virgule change de position.*

2- Représentation en virgule fixe :

Dans la représentation en virgule fixe, la **Partie Entière (PE)** est représentée sur **n bits** et la **Partie Fractionnaire (PF)** est représentée sur **m bits.**

❖ Pour un nombre entier pur, le nombre maximal qu'on peut présenter sur **n bits** est $N1_{max}$ tel que pour :

$$PE\ max \Rightarrow N1_{max} = 2^n - 1$$

❖ Pour un nombre fractionnaire, le nombre maximal qu'on peut présenter sur **m bits** est $N2_{max}$ tel que pour :

$$PF\ max \Rightarrow N2_{max} = 1 - 2^{-m}$$

❖ Le nombre maximal qu'on peut présenter en virgule fixe avec **n bits** pour la **PE** et **m bits** pour la **PF** est N_{max} tel que:

$$N_{max} = N1_{max} + N2_{max} = (2^n - 1) + (1 - 2^{-m}) = 2^n - 2^{-m}$$

❖ Le nombre minimal qu'on peut présenter en virgule fixe avec **n bits** pour la PE et **m bits** pour la PF est N_{min} tel que:

$$N_{min} = 2^{-m}$$

N_{min} est appelé le **PAS** *de la représentation*

Exemple

Représenter le nombre N= $(+14,25)_{10}$ en virgule fixe où la **Partie Entière(PE)** est représentée sur **8 bits,** la **Partie Fractionnaire (PF)** st représentée sur **7 bits** et **1 bit** pour le **signe.**

Signe = + = 0

PE = $(14)_{10}$ =$(1110)_2$

PF = $(0,25)_{10}$=$(01)_2$

0.25 x 2 = 0.5
0.5 x 2 = 1.0

N=

0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0

On complète avec des 0 de droite vers la gauche.

On complète avec des 0 de gauche vers la droite.

3- Représentation en virgule flottante :

La représentation en virgule flottante consiste à représenter le nombre N sous la forme suivante:

$$N = \pm 0,M \times B^{E}$$

Avec:

M: Mantisse normalisée

B: Base

E: Exposant

Remarque:

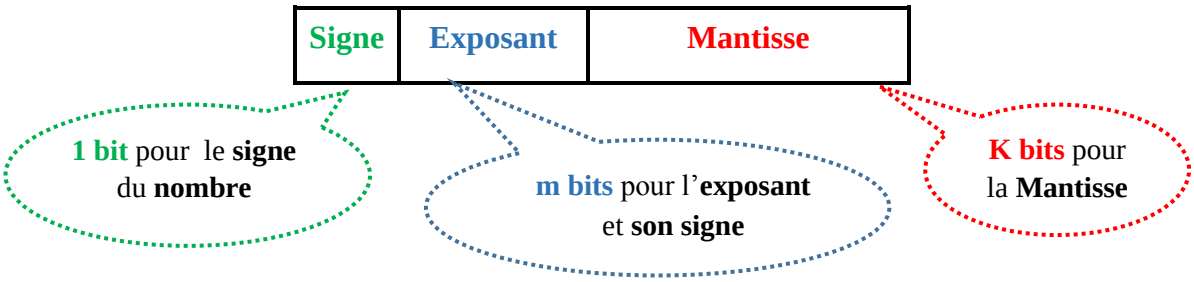
On dit que la mantisse est normalisée si le premier chiffre après la virgule est différent de **0** et le premier chiffre avant la virgule est égale à **0**

Exemple:

Soit un nombre réel N $(+ 26,41)_{10}$

$(+ 26,41)_{10} = + 0,2641 \times 10^{2}$ ⇒ **M= 2641**, **B= 10**, **E= +2**

Le format de représentation des nombres flottants est comme suite:



Le nombre maximal qu'on peut présenter en virgule flottante avec **m bits** pour l'exposant est N_{max} tel que:

$$N_{max} = \text{Mantisse max} * 2^{Emax}$$

Avec : **Emax** = $2^{m} - 1$

Emin = **-Emax**

Exemple:

Représenter le nombre $(100,01)_2$ en format virgule flottante selon le format suivant: **1bit** pour le **signe**, **4bits** pour **l'exposant avec son signe** et **7bits** pour **la mantisse.**

On a : $(100,01)_2 = + 0, 1\,0\,0\,0\,1\,0\,0 \times 2^{+3}$

0	0 0 1 1	1 0 0 0 1 0 0

3- Représentation en virgule flottante selon la Norme IEEE -754 :

La Représentation en virgule flottante selon la Norme IEEE -75 est un cas particulier de la représentation en virgule flottante des nombres réels[4].

Pour normaliser une représentation selon la norme IEEE-754 on décale la virgule jusqu'à avoir **1, Mantisse** donc

$$N = \pm 1, M \times B^E$$

Dans la norme IEEE-754, on a 2 types de représentations

- Représentation **simple précision** ⟹ sur **32 bits**
- Représentation **double précision** ⟹ sur **64 bits**

3.1- Norme IEEE-754 simple précision :

Dans la Représentation simple précision on a :

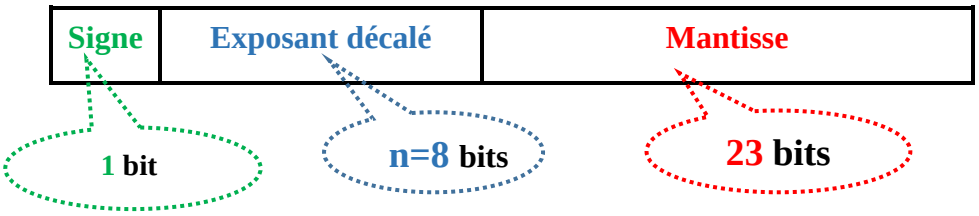
$$N = \pm 1, M \times B^E$$

Avec:

- **B**= base (B=2)
- **E**= exposant réel
- **M**= mantisse
- **1**= bit caché (*il n'est pas stocké mais il est présent*)

Représentation simple précision est sur **32 bits** organisés comme suite;

- **1** bit pour le signe
- **8** bits pour l'exposant décalé
- **23** bits pour la mantisse



Pour calculer l'exposant décalé on a :

Exposant Décalé = Exposant Réel + Décalage

$$\textbf{Décalage} = 2^{n-1} - 1$$

Avec:

n = *nombre de bit de l'exposant décalé* (**n= 8**)

Décalage = $2^{8-1} - 1 = 2^{7} - 1 = 128 - 1 = 127$

Décalage = 127

Remarque:

Le décalage dans la norme IEEE-754 simple précision, est toujours égal à **127.**

Exemple

Donner la représentation en virgule flottante simple précision du nombre **N = + $(32,75)_{10}$**

N = + $(32,75)_{10}$ = + $(100000,11)_2$

On décale la virgule vers la gauche tel que: **N= + $(1,0000011 \times 2^5)_2$**

- Signe = « + » ⟹ **Bit du Signe = 0**
- **Mantisse = 0000011**
- Exposant réel = **+5**

  Exposant Décalé = Exposant Réel + Décalage = 5+ 127 = **$(132)_{10}$**

  ➔ **Exposant Décalé = $(132)_{10}$ = $(10000100)_2$**

La représentation de **N** devient:

0	**10000100**	**00000110000000000000000**

3.2- Norme IEEE-754 double précision :

Dans la Représentation double précision on a :

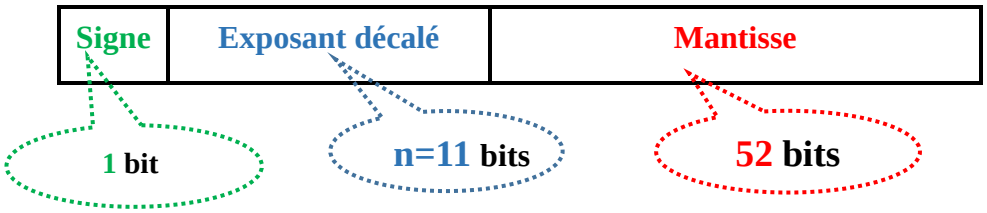
$$N = \pm 1, M \times B^{E}$$

Avec:

- **B**= base (B=2)
- **E**= exposant réel
- **M**= mantisse
- **1**= bit caché (*il n'est pas stocké mais il est présent*)

Représentation simple précision est sur **64 bits** organisés comme suite;

- **1** bit pour le signe
- **11** bits pour l'exposant décalé
- **52** bits pour la mantisse



Pour calculer l'exposant décalé on a :

Exposant Décalé = Exposant Réel + Décalage

$$\text{Décalage} = 2^{n-1} - 1$$

Avec:

n = *nombre de bit de l'exposant décalé* (**n= 11**)

Décalage = $2^{11-1} - 1 = 2^{10} - 1 = 1024 - 1 = 1023$

Décalage = 1023

Remarque:

Le décalage dans la norme IEEE-754 double précision, est toujours égal à **1023.**

Exemple

Donner la représentation en virgule flottante double précision du nombre **N = + (32,75)** $_{10}$

N = + (32,75)$_{10}$ = + (100000,11)$_2$

On décale la virgule vers la gauche tel que: **N= + (1,0000011 x 2^5)$_2$**

- Signe = « + » ⟹ **Bit du Signe = 0**
- **Mantisse = 0000011**
- Exposant réel = **+5**

  Exposant Décalé = Exposant Réel + Décalage = 5+ 1023 = **(1028)$_{10}$**

  ➔ **Exposant Décalé = (1028)$_{10}$ = (10000000100)** $_2$

La représentation de N devient :

0	10000000100	000001100000……..00

52 bits

FICHE TD 3

(Nombres réels)

Exercice 1 :

On dispose d'une machine où les nombres réels sont représentés en virgule fixe sur 16 bits comme suit :

- 1 bit de signe
- 9 bits partie entier (PE)
- 6 bits partie fractionnaire (PF)

Représenter le nombre $(- 29.774)_2$ sur cette machine

Exercice 2 :

Représenter le nombre 127.192 en virgule flottante norme IEEE 754 en simple précision puis en double précision.

Exercice 3 :

Exprimer N en décimal sachant que N est codé en IEEE 754.

$$N= (3021020311000000)_4$$

Solution

Exercice 1 :

$(-29.774)_{10}$

* signe= - $\Longrightarrow$1
* PE = $(29)_{10} = (11101)_2$
* PF =0.774

0.774*2=1.548

0.548*2=1.096

0.092*2=0.192

0.192*2=0.384

0.384*2=0.768

0.768*2=1.536

$(- 29.774)_{10} = - (11101.110001)_2$

1	0	0	0	0	1	1	1	0	1	1	1	0	0	0	1

Exercice 2 :

$(+127.192) = + (1111111.0011)_2$

$= + (1.1111110011 \times 2^{+6})$

* **simple précision**

BS = 0

MN = 1.1111110011

E = +6 $\Longrightarrow$ ED = +6+127 = 133

$ED=(10000101)_2$

0	10000101	111111001100……0

1BS 8bits ED 23bits MN

* **Double precision**

BS=0
MN=1.1111110011
E=+6 $\Longrightarrow$ ED=+6+1023=1029
$=(10000000101)_2$

0	10000000101	1111110011000……0

1BS 11bits ED 52bits MN

Exercice 3 :

$N = (3021020311000000)_4$

$= (11001001001000110101000000000000)_2$

1	10010010	01000110101000000000000

BS=1 $\Longrightarrow$ signe = " - "

MN= 1.01000110101

$ED= (10010010)_2 =(146)_{10}$

ED=E+D $\Longrightarrow$ E=ED-D=146-127=19

$N= - (1.01000110101 \times 2^{19})$

$= - (10100011010100000000)_2$

$= - (2^8+2^{10}+2^{12}+2^{13}+2^{17}+2^{20})$

$= - (1193216)_{10}$

Codification et Représentation des caractères (Alpha –Numérique)

1- Introduction:

Jusqu'ici on a vu le codage des nombres en vue de leur utilisation dans les opérations arithmétiques.

A présent, il s'agit de représenter les chaines de caractères, dans ce qui suit, on va étudier les codes suivants :

- Code Binary Coded Decimal (**BCD**)
- Code Binaire Réfléchi (**Gray**)
- Code BCD Excédent-3 (**EXC-3**)
- Code **ASCII**
- **Unicode.**
- **UTF-8**
- **Code EBCDIC**

2- Code Binary Coded Decimal (BCD)

Le code **BCD** permet de représenter les différents symboles, donc pour passer du système décimal au BCD, il faut remplacer chaque chiffre par sa valeur en binaire sur **4 bits**.

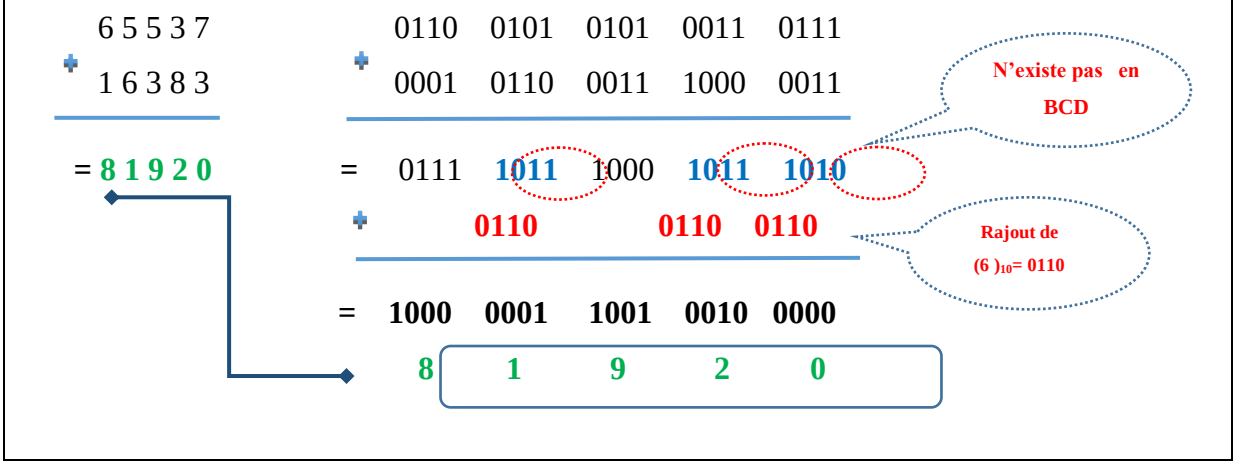
Les combinaisons supérieures à **9** sont interdites en **BCD**

Décimal	Binaire pur			
0	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**
2	**0**	**0**	**1**	**0**
3	**0**	**0**	**1**	**1**
4	**0**	**1**	**0**	**0**
5	**0**	**1**	**0**	**1**
6	**0**	**1**	**1**	**0**
7	**0**	**1**	**1**	**1**
8	**1**	**0**	**0**	**0**
9	**1**	**0**	**0**	**1**

2.1- Opérations en BCD :

Pour additionner deux nombres codés en BCD, Il faut procéder à l'addition binaire naturelle. Tout quartet dont la valeur ne correspond pas à une valeur du code BCD **(***nombres supérieur à* $(9)_{10} = (1001)_2$**)**, Il faut lui rajouter un $(6)_{10}$ soit $(0110)_2$.

Exemple:



3- Code Excess-3 :

Le code Excess-3 (aussi appelé code de Stibitz, XS3 ou XS-3 ou même BCD+3) est un code décimal binaire sur 4 bits (comme le BCD) créé par **George Robert Stibitz,** dans le but d'optimiser certains calculs en base 10 sur des anciens processeurs.

La représentation d'un nombre en code Excess-3 est dite biaisée car elle présente un décalage de **3** avec les valeurs attendues et utilisée par le code BCD (**code BCD Naturel + 3**).

Décimal	Excèdent- 3			
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**

Remarque:

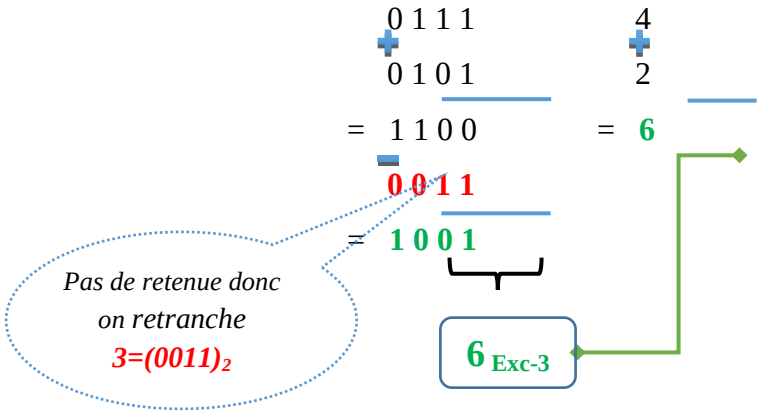
Il est plus facile de détecter une retenue que de tester si le résultat partiel est supérieur ou inférieur à **12**

3.1- Opérations en BCD+3

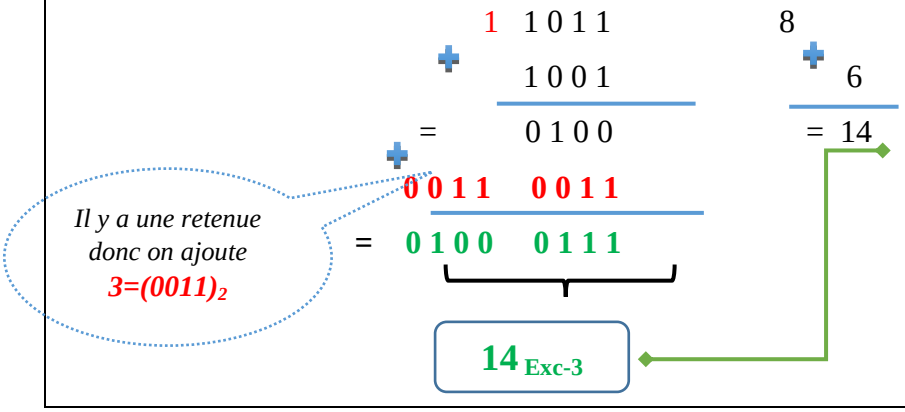
- Ajouter **3** en binaire naturel **$(0011)_2$** au résultat partiel s'il y a une retenue au résultat
- Retrancher **3** en binaire naturel **$(0011)_2$** au résultat partiel s'il n'y a pas de retenue au résultat partiel
- Ajouter **3** en binaire naturel **$(0011)_2$** à la retenue de débordement si elle existe

Exemples:

- Calculer (**4+6**) en **xs-3**



- Calculer (**8+6**) en **xs-3**



4- Code GRAY ou Binaire réfléchi :

Le code **GRAY** ou **Binaire réfléchi** est utilisé essentiellement dans la conversion d'une grandeur analogique en une grandeur numérique ; car dans ces conversions on a besoin d'un code dans lequel les grandeurs successives ne diffèrent que d'un seul caractère.

A chaque élément du système décimal correspond un mot de 4 bits du code binaire pur.

- Les opérations arithmétiques sont plus complexes
- 10 combinaisons sur 16 (4 bits) sont utilisées

4.1- Passage du binaire au code GRAY

Pour passer du binaire au code Gray, il faut ajouter au nombre N la valeur de N sans le bit de poids faible. Autrement dit si **N= a_3 a_2 a_1 a_0**

1. On garde **a_3** le bit du poids fort
2. On élimine **a_0** le bit du poids faible
3. On fait l'addition suivante : **(a_3 a_2 a_1 a_0 + a_3 a_2 a_1)**
4. Le résultat trouvé est en **gray**

Exemple :

Soit **N= (1 0 1 0)$_2$**

a_3 a_2 a_1 a_0

bit du poids fort de N **=0**

bit du poids faible de N=**1**

Donc on élimine le poids faible et on fait l'addition suivante :

N : **(a_3 a_2 a_1 a_0)**

N (sans le poids faible) : **(a_3 a_2 a_1)**

= **Résultat en gray**

1 0 1 0

1 0 1

= **1 1 1 1** en GRAY

N= (1010)$_2$ = (1111) gray

Remarque:

On ne tient pas compte de la retenue lorsqu'on a 1+1 =0 et retenue =1 à ignorer. Donc 1+1=0

Il existe une autre méthode pour Passer du Binaire au code Gray. Il suffit de changer le bit qui précède directement le bit 1.

Exemple

$(10)_{10}$ = $(1\ 0\ 1\ 0)_2$ = $(1\ 1\ 1\ 1)_{gray}$

4.2- Passage du Code GRAY au binaire

Pour passer du binaire au code Gray, il faut procéder comme suite :

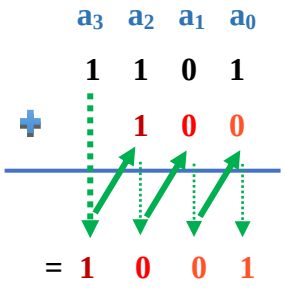
1. On garde a_3 le bit du poids fort
2. On ajoute le bit du poids fort a_3 à a_2
3. Le résultat trouvé dans le **2°/** est ajouté à a_1
4. Le résultat trouvé dans le **3°/** est ajouté à a_0

Remarque:

L'addition se fait de gauche à droite et (1+1)=0, la retenue est ignorer.

Exemple**:**

Soit **N= (1 0 1 0)$_2$**



5- Code ASCII

Est un code Américain (Americain Standard Code for Interchange Information) Appelé aussi code télégraphique international n°5.

La première version de ASCII était sur 7bits pour représenter les caractères de l'alphabet (minuscules et majuscules, les chiffres, ponctuation,…), et peut coder 2^7=128 caractères.

Par la suite il y a eu une version étendue sur 8 bits ***ASCII étendu*** il peut coder 2^8 =256 caractères, même les caractères accentués **é, è, ù, à,…**

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 6. Table ASCII.

Exemple

Le code (**65**) = $(01000001)_2$ correspond au caractère **A** (Voir la table ASCII)

Le code (**38**) = $(00100110)_2$ correspond au caractère **&** (Voir la table ASCII)

6- Code UNICODE ((UNIversal CODE)

L'idée de ce code est au lieu d'utiliser seulement les codes 0 à 7F sur 8 bits , il utilise des codes bien plus grands qui peuvent aller jusqu'à 32 bits.

L'Unicode permet de représenter tous les caractères spécifiques aux différentes langues. (Caractères latins accentués ou non, grecs, cyrillics, arméniens, hébreux, thaï, hiragana, katakana...)

Unicode définie aussi une correspondance entre symboles et code binaire et les nombres sont présentés en notation hexadécimal[E2].

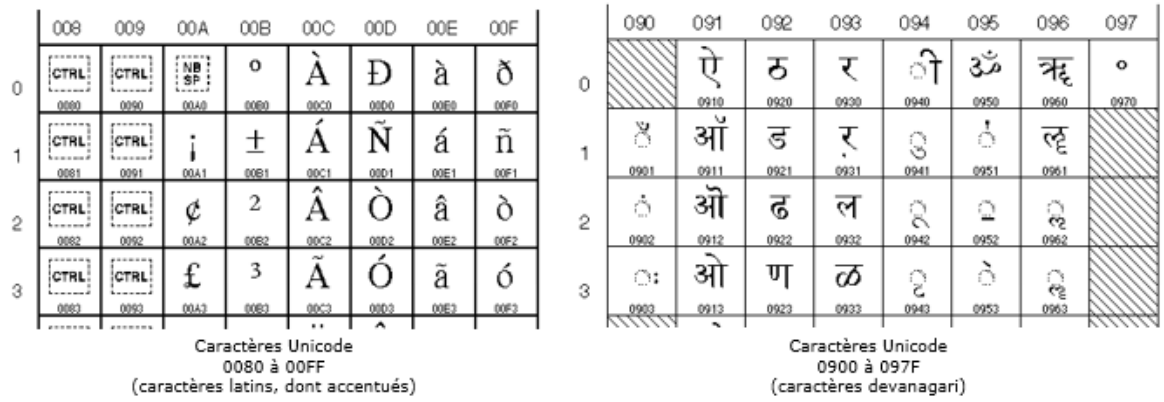


Figure 7. Extraits de tables UNICODE.

Exemple
Le code **bin/hexa = 00D2** correspond au caractère **Ő** (Voir la table UNICODE)
Le code **bin/hexa = C3A0** correspond au caractère **à** (Voir la table UNICODE)

7- Code UTF-8

L'UTF-8 rassemble le meilleur de deux codes précédents: l'efficacité de l'ASCII et l'étendue de l'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.

8- Code EBCDIC

C’est un code à 8bits dérivé du code BCD (**E**xtended **BCD I**nterchange **C**ode), donc sa logique de numérotation est celle du code BCD.

Inventé par IBM en 1964 pour ses ordinateurs, il a été très longtemps utilisé par les autres constructeurs d’ordinateurs; mais de nos jours on ut il n’est guère utilisé et le code ASCII a pris sa place.

FICHE TD 4

(Codification et Représentation des caractères (Alpha –Numérique))

Exercice 1 :

Convertir en BCD puis en excédent de 3 les nombres suivants :
$(109)_{10}$ $(11010)_2$ $(17)_8$ $(68,02)_{10}$

Exercice 2 :

Indiquer la valeur décimale des nombres suivants ou bien compléter les conversions suivantes :

$(1010\ 1000)_{xs-3}$ = ($)_{10}$

$(0001\ 1001\ 0011)_{BCD}$ = ($)_{10}$

$(0100)_{BCD+3}$ =($)_{xs-3}$

$(10)_{10}$ = ($)_2$ = ($)_{BCD}$

$(0111\ 0000\ 0110)_{BCD}$ = ($)_{10}$

Exercice 3 :

Calculer en BCD puis en XS-3 (vérifier les résultats en décimal) :

a) $(67)_{10} + (34)_{10}$

b) $(0100)_2 + (F)_{16}$

Exercice 4 :

Effectuer les conversions suivantes :

$(100110101)_2$ = ($)_{gray}$

$(16)_{10}$ = ($)_{gray}$

$(110010)_{gray}$ = ($)_2$

$(100111)_{gray}$ = ($)_{10}$

Exercice 5 :
Traduire le message suivant codé en ASCII:

0100 0010 0101 0010 0100 0001 0101 0110 0100 1111 0010 0001

.

Solution

Exercice 1 :

1) en BCD :

* $(109)_{10} = (0001\ 0000\ 1001)_{BCD}$

* $(11010)_2 = 2^1 + 2^3 + 2^4 = 16+8+2 = (26)_{10} = (0010\ 0110)_{BCD}$

* $(17)_8 = 7*8^0 + 1*8^1 = (15)_{10} = (0001\ 0101)_{BCD}$

* $(68,02)_{10} = (0110\ 1000,0000\ 0010)_{BCD}$

2) en excèdent de 3 :

* $(109)_{10} = (0100\ 0011\ 1100)_{xs-3}$

* $(11010)_2 = (26)_{10} = (0101\ 1001)_{xs-3}$

* $(17)_8 = (15)_{10} = (0100\ 1000)_{xs-3}$

* $(68,02)_{10} = (1001\ 1011,0011\ 0101)_{xs-3}$

Exercice 2 :

$(1010\ 1000)_{xs-3} = (75)_{10}$

$(0001\ 1001\ 0011)_{BCD} = (0100\ 1100\ 0110\ )_{xs-3}$

$(0100)_{BCD+3} = (1\ )_{10}$

$(10)_{10} = (1010\ )_2 = (0001\ 0000)_{BCD}$

$(0111\ 0000\ 0110)_{BCD} = (706)_{10}$

Exercice 3 :

En BCD :

a)

```
   67          0110  0111
 + 34    +      0011  0100
 ----        ------------
  101          1001  1011
                    0110
               ------------
                1010  0001
                0110
               ------------
```

$(\underbrace{0001}_{1}\ \underbrace{0000}_{0}\ \underbrace{0001}_{1})_{BCD}$

b)

$(1100)_2 = (12)_{10} = (0001\ 0010)_{BCD}$

$(F)_{16} = (15)_{10} = (0001\ 0101)_{BCD}$

```
  12          0001 0010
+ 15        + 0001 0101
----        -----------
  27          0010 0111
```

$\underbrace{0010}_{2}\ \underbrace{0111}_{7}$

En XS-3

a)

```
   67          (1001 | 1010)xs-3
+ 34           (0110 | 0111)xs-3
---        ------------------------
  101        0001 | 0000 | 0001
         +     11 + 0011 + 0011
        ------------------------
```

$(\underbrace{0100}\ |\ \underbrace{0011}\ |\ \underbrace{0100})_{xs\text{-}3}$

$(1\ \ 0\ \ 1)_{10}$

b)

```
  12          0100 | 0101
+ 15      +   0100 | 1000
----        ------------
  27          1000 | 1101
          -   0011 - 0011
              ------------
```

$(\underbrace{0101}\ |\ \underbrace{1010})_{xs\text{-}3}$

$(2\ \ 7)_{10}$

Exercice 4 :

Effectuer les conversions suivantes :

$(100110101)_2 = (\ 110101111)_{gray}$

$(16)_{10} = (10000)_2 = (11000)_{gray}$

$(110010)_{gray} = (100011\ )_2$

$(100111)_{gray} = (111010)_2 = 2^1 + 2^3 + 2^4 + 2^5 = 2 + 8 + 16 + 32 = (58\ )_{10}$

Exercice 4 :

(0100 0010 0101 0010 0100 0001 0101 0110 0100 1111 0010 0001)$_2$

42 52 41 56 4F 21)$_{16}$

B R A V O !

Chapitre 6 : Algèbre de Boole

1-Introduction

En 1847, George Boole un mathématicien anglais, a défini une algèbre qui s'applique à des fonctions logiques de variables logiques. Il avait pour but de traduire des idées et des concepts en équations, de leur appliquer certaines lois et de traduire le résultat en termes logiques[7].

Pour cela, George Boole crée une *algèbre binaire* n'acceptant que deux valeurs numériques : 0 et 1. Ces travaux ont été utilisés pour faire l'étude des systèmes qui possèdent deux états qui s'excluent mutuellement. Les travaux théoriques de *Boole*, trouveront des applications primordiales dans des domaines aussi divers que les systèmes informatiques.

2- Notion de base de l'algèbre de Boole

2.1 - Variable binaire

La variable logique est une grandeur qui peut prendre l'une de deux valeurs : vrai ou faux, 1 ou 0, qui sont repérées habituellement **0** ou **1**. La variable binaire est aussi appelée *variable booléenne*.

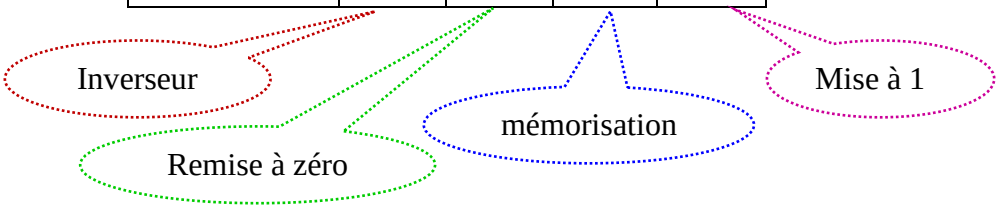
2.2 - Fonction logique

Une fonction logique est le résultat de la combinaison d'une ou de plusieurs variables logiques reliées entre elles par des opérations et règles mathématiques booléennes bien définies. Une fonction logique possède donc une ou des **variables logiques d'entrée** et une **variable logique de sortie**. La valeur résultante de cette fonction **ne peut être que 0 ou 1**.

2.2.1- Fonction à une variable logique

Du fait qu'une variable logique ne peut prendre que 2 valeurs (0 ou 1), le nombre de fonctions s'en trouve limité [6]. Ce qui nous donne le tableau de synthèse suivant :

Valeur de A	$F_1(A)$	$F_2(A)$	$F_3(A)$	$F_4(A)$
0	1	0	0	1
1	0	0	1	1



2.2.2- Fonction à 'n'variables logiques

Pour n variables on obtient 2^n combinaisons, c.à.d. si on a en entrée 2 variables on peut obtenir 4 combinaisons, (4 variables ⇒ 16 combinaisons)

2.3- Table de vérité

Une table de vérité nous permet de connaitre la réaction du circuit logique avec les diverses combinaisons d'entrées. Elle définit les relations entrée(s)/sortie(s) en faisant la liste de toutes les possibilités.

Une table de vérité contient $\mathbf{2^N}$ **lignes**, **N**= nombre d'entrées (variables).

3- Operateurs logiques de base

Il existe 3 operateurs logiques élémentaires en algèbre de Boole

- **OU** logique (**"addition"**),
- **ET** logique (**"multiplication"**)
- **NON** logique (**" négation"**)

3.1- OU logique

L'operateur **OU (**en anglais **OR)** est une addition logique symbolisée par **"+"**

$$f = a + b$$

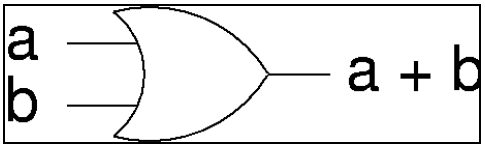


Figure 8. Représentation schématique OR.

a	b	a+b
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Tableau 2 : Table de vérité OR

3.2- ET logique :

l'operateur **ET (**en anglais **AND)** est une multiplication logique symbolisée par " **.** "

$$f = a \,.\, b$$

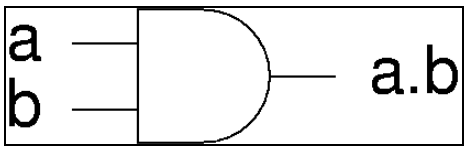


Figure 9. Représentation schématique AND.

a	b	a.b
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Tableau 3 : Table de vérité AND.

3.3- NON logique

L'operateur **NON (**en anglais **NOT)** est la fonction complément (inverseur) symbolisée par **"–"**

$$f = \bar{a}$$

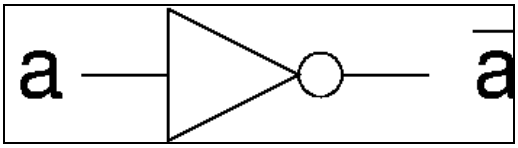


Figure 10. Représentation schématique NOT.

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

Tableau 4 : Table de vérité NOT.

Remarque:

Pour les opérateurs (**ET , OU**) nous avons juste donné la définition de base avec deux variables logiques, ils peuvent être réaliser avec plusieurs variables logiques ***(a+b+c+d..)***

4- Autres Operateurs logiques :

Il existe d'autres operateurs logiques en algèbre de Boole

4.1- NON-OU(NOR):

La fonction **NON-OU (**en anglais **NOR)** est une fonction logique exprimer par f **:**

$$f = \overline{a + b} = \overline{a}.\overline{b}$$

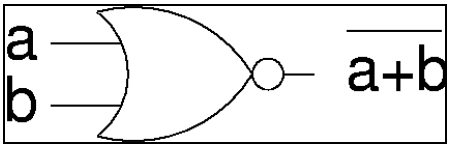


Figure 11. Représentation schématique NOR.

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0

Tableau 5: Table de vérité NOR.

4.2- NON-ET (NAND):

La fonction **NON-ET (**en anglais **NAND)** est une fonction logique exprimer par f **:**

$$f = \overline{a.b} = \overline{a} + \overline{b}$$

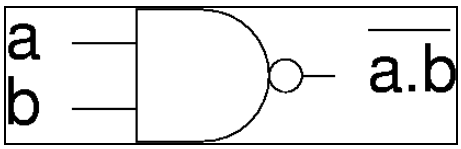


Figure 12. Représentation schématique NAND.

a	b	$\overline{a.b}$
0	0	1
0	1	1
1	0	1
1	1	0

Tableau 6: Table de vérité NAND.

4.3- OU-Exclusif (XOR):

La fonction **OU exclusif (**en anglais**)** est une fonction logique symbolisée exprimer par f **:**

$$f = a \oplus b = \bar{a}b + a\bar{b}$$

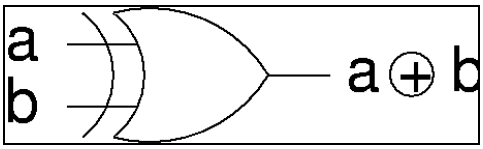


Figure 13. Représentation schématique XOR.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Tableau 7: Table de vérité XOR.

5- Propriétés de l'algèbre de Boole

Soient a, b et c, trois variables logiques.

5.1- Associativité :

Comme avec les opérations habituelles, certaines parenthèses sont inutiles :

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

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

5.2- Commutativité :

L'ordre est sans importance :

$a + b = b + a$

$a.b = b.a$

5.3- Distributivité

Comme avec les opérations mathématiques habituelles, il est possible de distribuer :

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

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

5.4- Elément neutre

$a + 0 = a$

$a.1 = a$

5.5- Elément absorbant

$a + 1 = 1$

$a.0 = 0$

5.6- involution

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

5.7- Complémentation

$a.\bar{a} = 0$

$a + \bar{a} = 1$

5.8- Idempotence

$a + a + a + \cdots + a = a$

$a.a.a. \cdots .a = a$

6- Théorème de MORGANE

La somme logique complimentée de deux variables est égale au produit des compléments des deux variables

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

Le produit logique complimenté de deux variables est égale au somme logique des compléments des deux variables.

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

Exemple

Déterminer le complément de la fonction suivante:

- $\boldsymbol{F_1(A,B,C,D) = (\bar{C}.D + A.B).(\bar{A}.\bar{C} + B.D)}$

$F_1(A,B,C,D) = (\bar{C}.D + A.B).(\bar{A}.\bar{C} + B.D)$

$= \overline{F_1(A,B,C,D)} = \overline{(\bar{C}.D + A.B)(\bar{A}.\bar{C} + B.D)}$

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

$= (C + \bar{D}).(\bar{A} + \bar{B}) + (A + C).(\bar{B} + \bar{D})$

Remarque:

- Pour chacune de ces combinaisons, la fonction peut prendre une valeur 0 ou 1.
- L'ensemble de ces $\mathbf{2^n}$ combinaisons et la valeur de la fonction correspondante représente la table de vérité

7- Forme canonique :

On a deux expressions logiques à partir table de vérité

7.1- Première Forme canonique

La Première forme canonique est une forme disjonctive dite aussi **SOP (Sum Of Products)** somme des mintermes, en prenant compte que les **« 1 » de la fonction *f***

Un minterme est le produit logique des variables de la même ligne de la table de vérité.

7.2- Deuxième Forme canonique

La Deuxième Forme canonique est une forme conjonctive dite aussi **POS (Product Of Sums)** produit des maxtermes, en prenant compte que les **« 0 » de la fonction *f***

Un maxterme est la somme logique des variables sous forme inversée de la même ligne de la table de vérité.

Remarque:

- La première et la deuxième forme canonique sont équivalentes.
- La première forme canonique est la forme la plus utilisée.

.

Exemple

a	b	c		F(a,b,c)
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		0
1	0	1		1
1	1	0		1
1	1	1		1

1er forme

SOP : $f(a,b,c) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}c + ab\bar{c} + abc$

2ème forme

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

8- Simplification d'une fonction logique par la méthode algébrique :

La simplification d'une équation logique se fait très souvent par « calcul » algébrique en cherchant à mettre en facteur les variables et en utilisant les propriétés de l'algèbre de Boole (§ 5).

Exemple

Simplifier ***f***

$$f(A,C) = (A+C)(A+\bar{C})$$
$$= AA + AC + A\bar{C} + C\bar{C}$$
$$= A + A(C+\bar{C})$$
$$= A + A$$
$$= A$$

$AA = A$; $C\bar{C} = 0$; $(C+\bar{C}) = 1$; $(C+\bar{C}) = 1$; $A + A = A$

9- Table de KARNAUGH

Au même titre que les propriétés et relations booléennes, la table de KARNAUGH est un outil de simplification des expressions logiques [8].

Une table de KARNAUGH ne diffère de la table de vérité que par sa présentation. La table de KARNAUGH est une représentation de la table de vérité sous une forme graphique, et comporte autant de cases que la table de vérité tel que :

Nombre de cases de table KARNAUGH = Nombre de lignes de table de vérité

Nombre de cases de table KARNAUGH = 2^n *(n : nombre)*

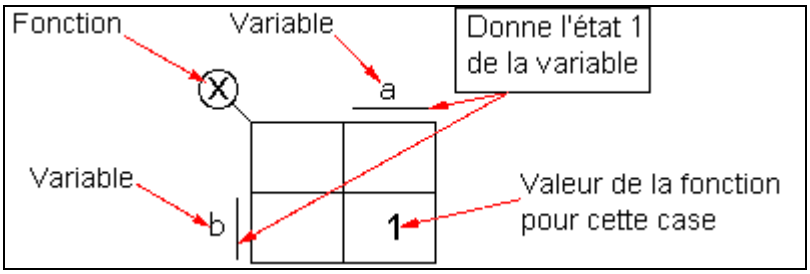


Figure 14. Table de KARNAUGH.

On réalise une table pour une sortie et chaque case correspond à une seule combinaison des variables d'entrées.

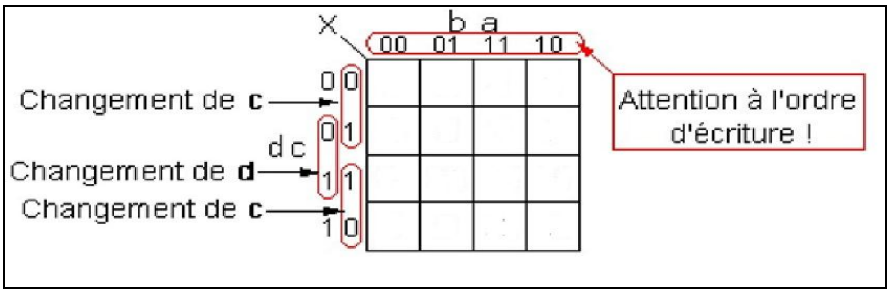


Figure 15. Variable d'entrées de la Table de KARNAUGH.

10- Simplification d'une fonction logique par methode KARNAUGH

Pour simplifier une fonction logique par la table KARNAUGH, on suit les étapes suivantes :

1- A partir de la table de KARNAUGH on simplifie en groupant les « 1 » adjacents.

2- Les « 1 » adjacents sont mis en évidence par l'ordre utiliser pour former la table de vérité

3- La taille d'un groupe est un multiple de 2^k = (1, 2, 4, 8, 16, etc..)

4- Les groupes sont soit rectangulaires soit carrés

5- Les « 1 » des bords (extrémités) d'une table KARNAUGH sont adjacents (la table se referme sur elle-même).

Remarque:

- Il faut former les plus gros groupes possibles.
- un « 1 »peut faire partie de plusieurs groupes.

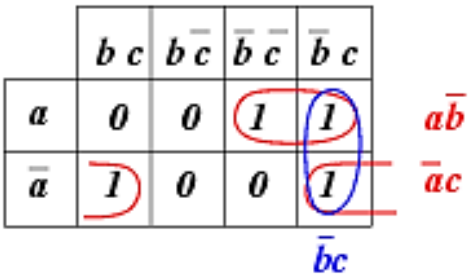
Exemple 1

Soit une fonction a trois variables ***f(a,b,c)***, les groupements considérés de plusieurs cases marquées '1' sont :

- **8 cases (toutes)** : la fonction est égale à la **constante '1'**
- **4 cases (consécutives ou en carré)** : le terme correspondant aux 4 cases est formé d'une **seule variable** ou de son complément
- **2 cases (accolées)** : le terme est composé de **deux variables** (a et b par exemple).
- **1 case** : les termes sont composés de **trois variables** (ou de leurs compléments)
- **0 case** : la fonction est **nulle.**

Exemple 2

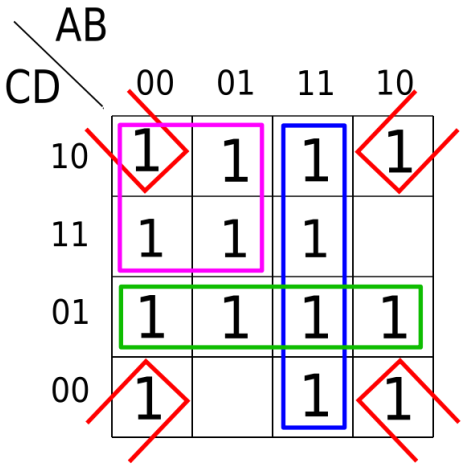
a	b	c		f(a,b,c)
0	0	0		**0**
0	0	1		**0**
0	1	0		**1**
0	1	1		**1**
1	0	0		**1**
1	0	1		**0**
1	1	0		**1**
1	1	1		**0**



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

Exemple 3

A	B	C	D		f(A,B,C)
0	0	0	0		**1**
0	0	0	1		**1**
0	0	1	0		**1**
0	0	1	1		**1**
0	1	0	0		**0**
0	1	0			**1**
0	1	1	0		**1**
0	1	1	1		**1**
1	0	0	0		**1**
1	0	0	1		**1**
1	0	1	0		**1**
1	0	1	1		**0**
1	1	0	0		**1**
1	1	0	1		**1**
1	1	1	0		**1**
1	1	1	1		**1**



f(A,B,C) = $AB + \bar{C}D + \bar{A}C + \bar{B}\bar{D}$

11- Schéma d'un circuit logique (Logigramme):

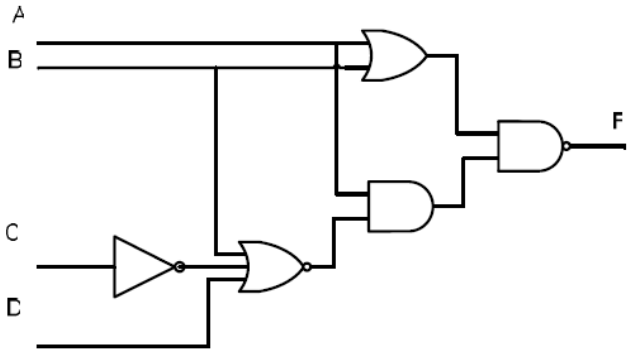
Schéma d'un circuit logique ou logigramme est la représentation graphique (schéma électronique) d'une fonction logique[8].

Le principe consiste à remplacer chaque opérateur logique d'une fonction par son symbole logique

Exemple

- Donner le logigramme de F

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



FICHE TD 5

(Algèbre de Boole)

Exercice 1 :

Simplifier au maximum les expressions logiques suivantes:

1- $B.\bar{C} + \bar{A}.\overline{(B.\bar{C})}.(A.D + B)$

2- $(A + B + C).(\bar{A} + B + C) + A.B + B.C$

Exercice 2 :

Déterminer le complément des fonctions suivantes :

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

Exercice 3 :

a) Écrire sous la première forme canonique la fonction définie par les propositions suivantes :

f (A, B, C) = 1, si et seulement si exactement une des variables A, B, C prend la valeur 1.

b) Écrire sous la seconde forme canonique la fonction définie par les propositions suivantes :

g (A, B, C) = 0, si et seulement si les variables A, B, C prennent la valeur 1.

Exercice 4 :

Considérer la fonction définie par la table de vérité suivante :

A	B	C	F (A, B, C)
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

1- Générer une expression logique correspondante :

a) Sous forme de sommes de produit.

b) Sous forme de produit de sommes.

2- Simplifier les deux expressions en utilisant les règles de l'algèbre de Boole.

3- Construire la table de KARNAUGH et déterminer une expression logique associée.

Exercice n° 5:

Considérer les fonctions logiques suivantes.

1-F_1 (A, B, C) = A. $\overline{B}$. C + A. B. $\overline{C}$ + A. B. C

2-F_2 (A, B, C) =$\overline{A}$. $\overline{B}$. $\overline{C}$+ A. $\overline{B}$ + A. B. C

Pour chacune d'elles :

1 - Construire la table de KARNAUGH ;

2- Utiliser le diagramme pour simplifier les expressions.

Solution

Exercice 1 :

1- $B.\bar{C} + \bar{A}.\left(\overline{B.\bar{C}}\right).(A.D + B)$

$= B\bar{C} + \bar{A}.(\bar{B} + C).(A.D + B)$

$= B\bar{C} + (\bar{A}.\bar{B} + \bar{A}.C)(A.D + B)$

$= B\bar{C} + \bar{A}.\bar{B}.A.D + \bar{A}.C.A.D + \bar{A}.\bar{B}.B + \bar{A}.C.B$

$= B\bar{C} + \bar{A}.C.B$

$= B(\bar{C} + \bar{A}C)$

2-$(A + B + C).(\bar{A} + B + C) + A.B + B.C$

$= A\bar{A} + AB + AC + \bar{A}B + BB + BC + \bar{A}C + BC + CC + AB + BC$

$= AB + AC + \bar{A}B + B + BC + \bar{A}C + C$

$= AB + C(A + 1) + \bar{A}B + B(1 + C) + \bar{A}C$

$= AB + C + \bar{A}B + B + \bar{A}C$

$= AB + \bar{A}B + C + B + \bar{A}C$

$= B(A + \bar{A}) + C + B + \bar{A}C$

$= B + C + \bar{A}C$

$= B + C(1 + \bar{A})$

$= B + C$

Exercice 2 :

Le complément des fonctions suivantes

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

$= (\bar{A} + B)(C + D)(A + \bar{C} + D)(\bar{D} + C) + \left((\bar{A} + \bar{B})(A + B)\right)(\bar{D} + \bar{B}) + \left((\bar{A} + C)(A + \bar{C})\right)$

La question est de trouver le complément non pas de simplifier.

Exercice 3 :

$f(A,B,C) = 1$ *si et seulement si une des variables* A,B,C *prend la valeur* 1

$f(A,B,C) = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C}$

$f(A,B,C) = 0$ *si et seulement si les variables* A,B,C *prennent la valeur* 1

$f(A,B,C) = \bar{A} + \bar{B} + \bar{C}$

Exercice 4 :

A	B	C	$F(A,B,C)$
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/ Expression logique correspondante

- Sous forme de sommes de produits

$\bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C + A\,B\,\bar{C}$

- Sous forme de produits de sommes

$\overline{\bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,C + A\,B\,C} = (A + B + C)\,(A + \bar{B} + \bar{C})\,(\bar{A} + \bar{B} + \bar{C})$

b/ Simplification :

A+C	A	C	$\bar{A}$	$\bar{A}C$	$\bar{A}C + A$
0	0	0	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

$\bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C + A\,B\,\bar{C}$

$\bar{A}\,\bar{B}\,C + B\,\bar{C}\,(A + \bar{A}) + A\,\bar{B}\,(C + \bar{C})$

$\bar{A}\,\bar{B}\,C + B\,\bar{C} + A\,\bar{B}$

$\bar{B}\,(\bar{A}\,C + A) + B\,\bar{C}$

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

$(A + B + C)\,(A + \bar{B} + \bar{C})\,(\bar{A} + \bar{B} + \bar{C})$

$= (A + A\,B + A\,C + A\,\bar{B} + \sout{B\,\bar{B}} + \bar{B}\,C + A\,\bar{C} + B\,\bar{C} + \sout{C\,\bar{C}})\,(\bar{A} + \bar{B} + \bar{C})$

$= \sout{A\,\bar{A}} + A\,\bar{B} + A\,\bar{C} + \sout{A\,B\,\bar{A}} + \sout{A\,B\,\bar{B}} + A\,B\,\bar{C} + \sout{A\,C\,\bar{A}} + A\,C\,\bar{B} + \sout{A\,C\,\bar{C}} + \sout{A\,\bar{B}\,\bar{A}} + A\,\bar{B} + A\,\bar{B}\,\bar{C} + \bar{B}\,C\,\bar{A} + \bar{B}\,C + \sout{\bar{B}\,C\,\bar{C}} + \sout{A\,\bar{C}\,\bar{A}} + A\,\bar{C}\,\bar{B} + A\,\bar{C} + B\,\bar{C}\,\bar{A} + \sout{B\,\bar{C}\,\bar{B}} + B\,\bar{C}$

$= A\,\bar{B} + A\,\bar{C} + A\,B\,\bar{C} + A\,C\,\bar{B} + A\,\bar{B} + A\,\bar{B}\,\bar{C} + \bar{A}\,\bar{B}\,C + \bar{B}\,C + A\,\bar{B}\,\bar{C} + A\,\bar{C} + \bar{A}\,B\,\bar{C} + B\,\bar{C}$

$= A\,\bar{B} + A\,\bar{C} + A\,B\,\bar{C} + A\,C\,\bar{B} + A\,\bar{B} + A\,\bar{B}\,\bar{C} + \bar{A}\,\bar{B}\,C + \bar{B}\,C + A\,\bar{B}\,\bar{C} + \bar{A}\,B\,\bar{C} + B\,\bar{C}$

$= A\,\bar{B}\,\underbrace{(1 + C + \bar{C})}_{1} + A\,\bar{C}\,(1 + B) + A\,\bar{B}\,\bar{C} + \bar{A}\,\bar{B}\,C + \bar{B}\,C + \bar{A}\,B\,\bar{C} + B\,\bar{C}$

$= A\,\bar{B} + A\,\bar{C} + A\,\bar{B}\,C + \bar{A}\,\bar{B}\,C + \bar{B}\,C + \bar{A}\,B\,\bar{C} + B\,\bar{C}$

$= A\,\bar{B} + A\,\bar{C} + \bar{B}\,C\,(A + \bar{A}) + \bar{B}\,C + B\,\bar{C}\,(1 + \bar{A})$
$= A\,\bar{B} + A\,\bar{C} + \bar{B}\,C + \bar{B}\,C + B\,\bar{C}$
$= A\,\bar{B} + A\,\bar{C} + \bar{B}\,C + B\,\bar{C}$

c/ $\bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C + A\,B\,\bar{C}$

B C / A	$B\,C$	$\bar{B}\,C$	$\bar{B}\,\bar{C}$	$B\,\bar{C}$
A	0	1	1	1
$\bar{A}$	0	1	0	1

$= \bar{B}\,C + A\,\bar{B} + B\,\bar{C}$

Exercice 5 :

- $F_1(A,B,C,D) = A\,\bar{B}\,C + A\,B\,\bar{C} + A\,B\,C$

B C / A	$B\,C$	$\bar{B}\,C$	$\bar{B}\,\bar{C}$	$B\,\bar{C}$
A	1	1	0	1
$\bar{A}$	0	0	0	0

$F_1(A,B,C) = A\,B + A\,C$

- $F_2(A,B,C) = \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B} + A\,B\,C$

B C / A	$B\,C$	$\bar{B}\,C$	$\bar{B}\,\bar{C}$	$B\,\bar{C}$
A	1	1	1	0
$\bar{A}$	0	0	1	0

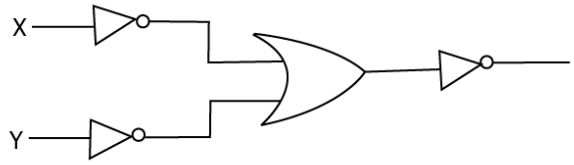
$F_2(A,B,C) = A\,C + \bar{B}\,\bar{C}$

FICHE TD 6

(Les Circuits Combinatoires)

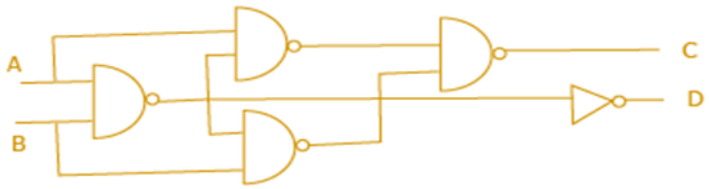
Exercice 1 :

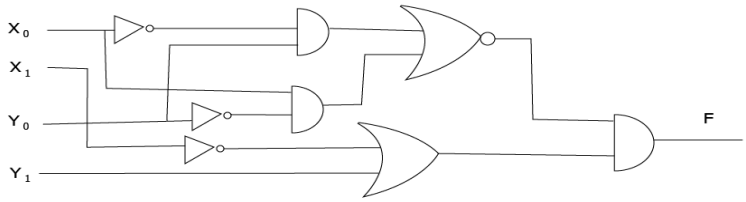
Quelle est la fonction logique réalisée par le circuit de la figure suivante :

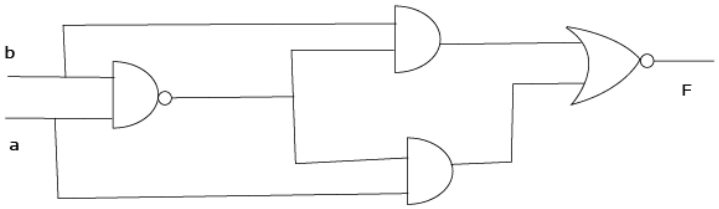


Exercice 2 :

Analyser les circuits suivants et précisez leur fonction







Exercice 3 :

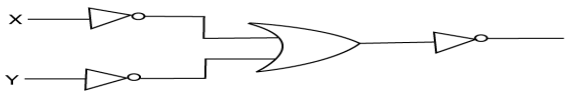
Soi**t X,Y,Z** de variables d'entrées, F est définie comme suit :

$$F(A, B, C) = \begin{cases} 1 & Si\ la\ majorité\ des\ variables\ valent\ 1 \\ 0 & sinon \end{cases}$$

1- Réaliser la table de vérité.
2- Simplifier la fonction F en utilisant la table de Karnaugh
3- Réaliser le circuit électronique de F

SOLUTION

Exercice 1 :



$$\overline{\overline{X} + \overline{Y}} \equiv \overline{\overline{X}}.\overline{\overline{Y}} \equiv X.Y$$

Exercice 2:

a- $\overline{\overline{A.B}\,.\,A}\,.\overline{\overline{A.B}\,.\,B} \equiv \overline{A\,B + \bar{A}\,.\,A\,B + \bar{B}}$

$\equiv (\bar{A} + \bar{B})\,A + (\bar{A} + \bar{B})\,B$

$\equiv \bar{B}\,A + \bar{A}\,B$

$\equiv B \oplus A$

B	A	$\bar{B}A$	$\bar{A}B$	+
1	1	0	0	0
1	0	1	1	1
0	1	1	1	1
0	0	0	0	0

b- $\overline{\overline{X_0}\,Y_0 + X_0\,\overline{Y_0}}\;.\;\overline{X_1} \oplus Y_1$

$\overline{\overline{X_0}\,Y_0}\;.\;\overline{X_0\,\overline{Y_0}}\;.\;\overline{X_1}\,\bar{Y}_1 + X_1\,Y_1$

$(X_0 + \bar{Y_0})\,(\overline{X_0} + Y_0)\,.\;\overline{X_1}\,\bar{Y}_1 + X_1\,Y_1$

$\cancel{\overline{X_0}\,X_0} + \bar{Y_0}\,\bar{X}_0 + X_0\,Y_0 + \cancel{\bar{Y}_0\,Y_0} + \overline{X_1}\,\bar{Y}_1 + X_1\,Y_1$

$X_0\,Y_0 + \overline{X_0}\,\bar{Y}_0 + X_1\,Y_1 + \overline{X_1}\,\bar{Y}_1$

$\overline{X_0} \oplus Y_0 + \overline{X_1} \oplus Y_1$

c- $\overline{\overline{a\,b}\,.\,b + \overline{a\,b}\,.\,a}$

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

$= \left(a\,b + \bar{b}\right).(a\,b + \bar{a}) = a\,b + a\,b\,\bar{b} + a\,\bar{a}\,b + \bar{a}\,\bar{b}$

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

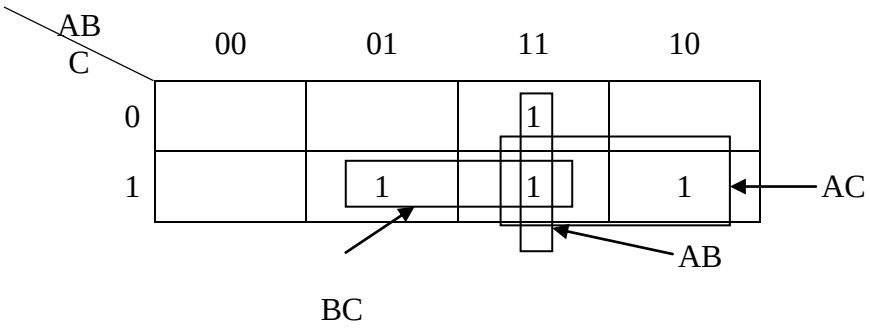
$= \bar{a} \oplus \bar{b}$

Exercice 3 :

F(A, B, C) = $\begin{cases} 1 & Si\ la\ majorité\ des\ variables\ valent\ 1 \\ 0 & sinon \end{cases}$

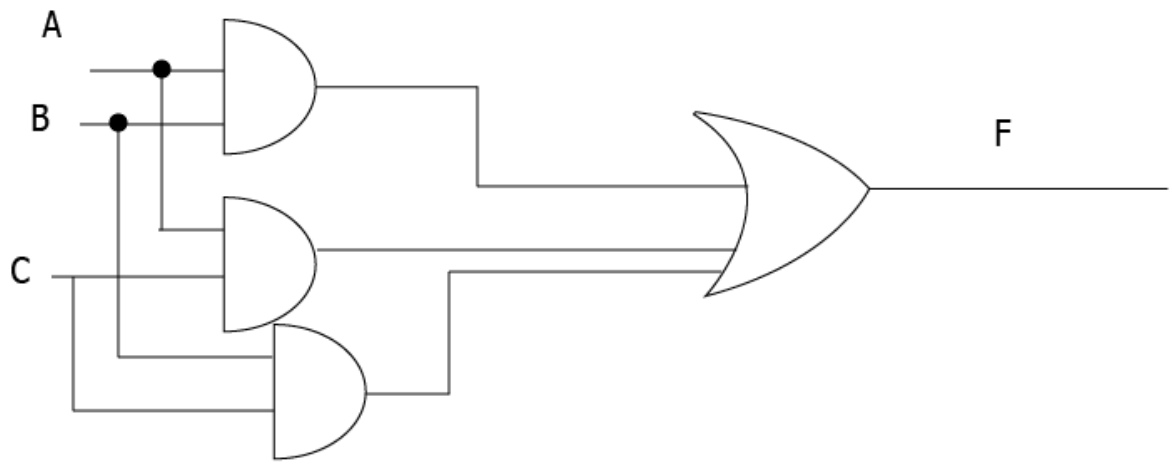
Table de vérité :

A	B	C	F
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



$$F(A,B,C) = A\,B\; +\; A\,C + B\,C$$

Circuit électronique de la fonction



Abréviation

Bit : Binary Digit

B ou **b:** Base

MSB : Most Significant Bit

LSB : Least Significant Bit.

(**SVA**) : Signe et Valeur Absolue

(CP1) : Complément à 1

(CP2) : Complément à 2

(BCD) : Binary Coded Decimal

ASCII : Americain Standard Code Information Interchange

Char : Caractère

Hex : Hexadécimal

F ou ***f*** : Fonction

Références

[1] Alain Cazes,Joëlle Delacroix, Architecture Des Machines Et Des Systèmes Informatiques : Cours et exercices corrigés, 3° édition, Dunod 2008.

[2] C. Alexandre, Circuits numériques, Polycopié de cours électronique, Conservatoire national des arts et métiers, France, 2004.

[3] Cédric Frayssinet et Mathieu Hibou numérique et science informatique Spécialité NSI - Année 2020 - 2021 Version 2 -15 mars 2021

[4] Jean Privat Chapitre 10 Arithmétique réelle Université du Québec à Montréal INF2170 Organisation des ordinateurs et assembleur Automne 2013

[5] John R. Gregg, Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets 1st Edition , Wiley & sons Inc. publishing, 1998, ISBN: 978-0-7803-3426-7.

[6] Bradford Henry Arnold , Logic and Boolean Algebra, Dover publication, Inc., Mineola, New York, 2011, ISBN-13: *978-0-486-48385-6*

[7] Daniel Etiemble « Algèbre De Boole Et Fonctions Booléennes » S4-CLM

[8] Mange Daniel, Analyse et synthèse des systèmes logiques, PPUR, 1995

[9] Jean-Christophe Dubacq, Introduction à l'informatique Cours complet IUT de Villetaneuse, S1 2016.

[E1] https://info.blaisepascal.fr/isn-architecture-dun-ordinateur (PUBLIÉ 23 AOÛT 2016 · Mis à Jour 20 Mars 2019)

[E2] www. Unicode.org (Mis à jour 2020)

[E3]www.didel.com/C/CalculsBCD.pdf

Sommaire