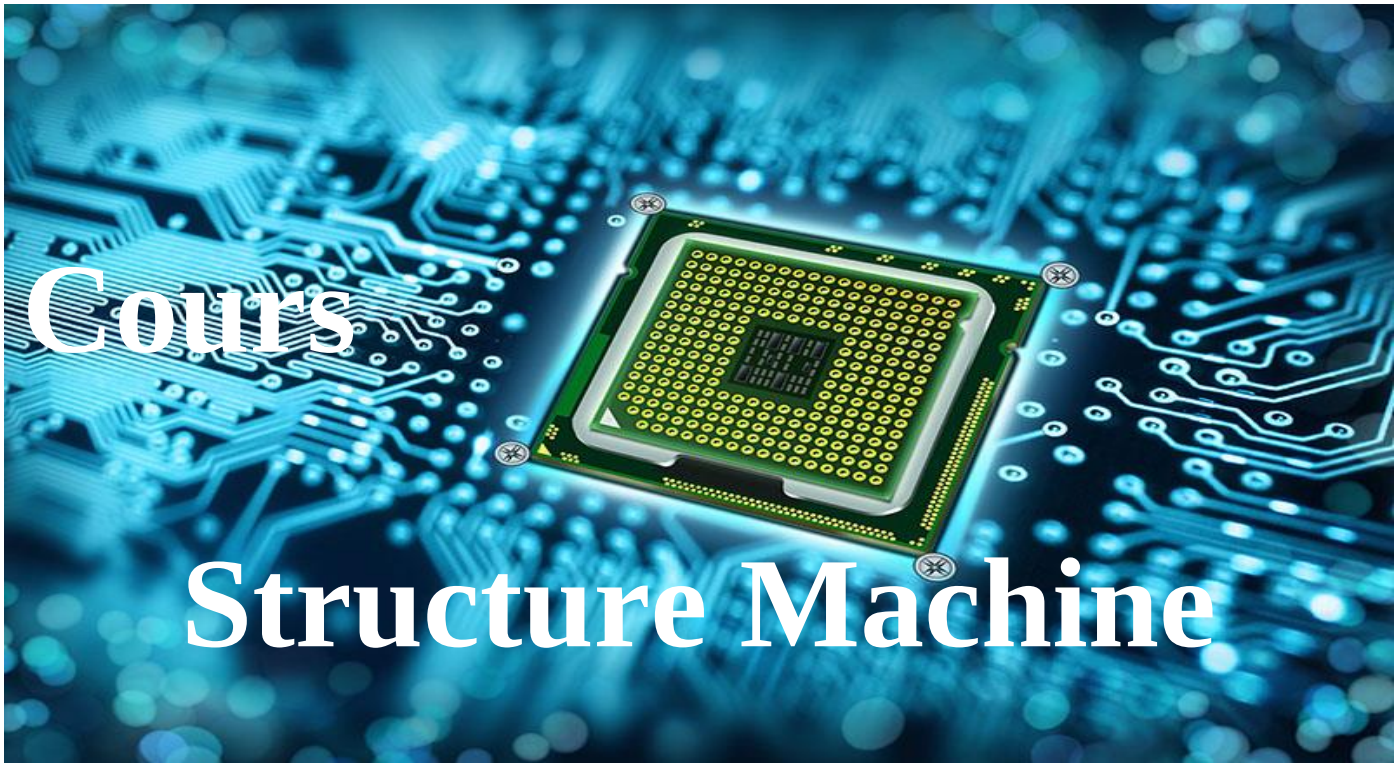
Ministère de l'Enseignement Supérieur

Et de laRecherche Scientifique

CENTRE UNIVERSITAIRE CHERIF BOUCHOUCHA AFLOU

Faculté de Mathématiques et de l'Informatique

Département de l'Informatique



GUIDE DE L'ÉTUDIANT

Présenté Par : Dr. AMMAR. Sabrina

Cet ouvrage offre une présentation pédagogique des cours et des exercices, synthétisant les concepts fondamentaux du module de Structure Machine

Pour plus d'informations ou des précisions supplémentaires,
Veuillez contacter l'auteur :
Sabrina AMMAR *Maître de conférence au centre universitaire* ***Cherif Bouchoucha-Aflou***
s.ammar@cu-aflou.edu.dz

Remercîment

Je souhaite exprimer ma sincère reconnaissance à l'ensemble des personnes qui ont contribué à la réussite de mon habilitation à diriger des recherches.

En premier lieu, je remercie chaleureusement mon directeur, pour sa guidance exceptionnelle, ses conseils pertinents et son soutien constant tout au long de ce parcours. Votre rigueur scientifique, votre expérience et vos encouragements ont été des éléments essentiels qui m'ont permis de mener à bien ce travail.

Je tiens également à remercier les membres du jury pour avoir accepté d'évaluer ce travail. Leurs remarques constructives et leurs commentaires éclairés ont enrichi la réflexion et permis de l'améliorer considérablement.

Enfin, je voudrais exprimer ma gratitude à mes collègues, amis et proches, dont le soutien moral et l'encouragement m'ont été précieux tout au long de cette aventure académique.

À toutes et à tous, je vous adresse mes plus sincères remerciements.

Avant-propos

Chers étudiants,

Ce polycopié est conçu pour vous accompagner tout au long du module **"Structure Machine"**, en vous fournissant un support clair et structuré pour la compréhension des concepts fondamentaux et avancés qui seront abordés. Ce module est essentiel dans votre cursus universitaire en informatique,

Nous avons structuré ce document de manière à faciliter l'apprentissage progressif des notions théoriques, tout en les liants à des applications pratiques. Vous y trouverez des explications détaillées, des exemples concrets et des exercices d'application qui vous aideront à approfondir vos connaissances et à développer les compétences nécessaires dans ce domaine. N'hésitez pas à utiliser ce support comme une référence tout au long de vos études et même au-delà.

Je vous encourage à aborder ce module avec curiosité et détermination, car la compréhension des structures des machines est au cœur des innovations technologiques actuelles. Vos efforts dans l'assimilation de ces savoirs contribueront à vous former en tant que futurs professionnels compétents et créatifs.

Je reste à votre disposition pour toute question ou éclaircissement que vous jugerez nécessaire. En vous souhaitant un excellent parcours dans ce module.

Avec mes meilleures salutations

SYLLABUS

Unit d'enseignement : **UEF122 : Structure machine 1**

Matière : **Structure Machine 1**

Domaine/ Filière : **1er année Licence Informatique académique.**

Semestre : **01**

Crédit : **05**

Volume Horaire Hebdomadaire Total : **42h**

Cours : **1h 30**

Travaux dirigés : **1h30**

Enseignant responsable : **Dr. AMMAR Sabrina**

E-mail : **s.ammar@cu-aflou.edu.dz**

Table des matières

CHAPITRE-I-INTRODUCTION GENERALE

CHAPITRE -II-Les Systèmes de Numération.

CHAPITRE-III-La représentation de l'information

CHAPITRE-IV-L'algèbre de Boole binaire.

Liste des figures

Liste des tableaux

Liste des abréviations

ASCII : Americain Standard Code Information Interchange .

BCD : Binary Coded Decimal.

Bit: Binary Digit .

CP1 : Complément à 1.

CP2 : Complément à 2.

CPU: Microprocesseur .

EBCDIC : Extended Binary Coded Decimal Interchange Code.

f : Fonction.

IBM : International Business Machines Corporation.

IEEE -754 : Institute of Electrical and Electronics Engineers.

LSB: Least Significant Bit.

MSB: Most Significant Bit.

NAND : NOT AND.

PE : Partie Entière**.**

PF : Partie Fractionnaire.

RAM: Random Access Memory.

ROM: Read-Only Memory.

SSD : Solide State Drives.

SVA : Signe et Valeur Absolue.

UAL : Unité arithmétique et logique.

USB: Universal Serial Bus.

UTF-8 : Universal Character Set Transformation Format - 8 bits.

CHAPITRE-I-INTRODUCTION GENERALE

I.1 Introduction :

La structure machine en informatique, souvent désignée sous le terme d'architecture des ordinateurs, est un domaine fondamental qui explore la conception, le fonctionnement et l'interaction des composants matériels d'un ordinateur et la manière dont ces éléments interagissent pour exécuter des instructions.

L'histoire des ordinateurs est étroitement liée aux découvertes théoriques en mathématiques, logique et en électronique. Elle est marquée par la détermination de l'homme d'automatiser les calculs pour les rendre plus précis et fiables. Aujourd'hui, l'informatique s'étend à presque toutes les sphères de l'activité humaine. Les machines à calculer, qui nous intéressent ici, sont des dispositifs physiques dont le fonctionnement repose sur l'utilisation du courant électrique. En effet, la forme la plus simple d'information que l'on peut représenter sur ces dispositifs est "le courant passe" ou "le courant ne passe pas". En d'autres termes, un ordinateur traite des informations numériques, où l'information est soutenue par un système binaire à deux états stables, 0 et 1, correspondant aux unités élémentaires d'information appelées bits. Tout traitement de données est codé sous une forme binaire (0 ou 1).

Les composants physiques qui participent à ce traitement sont appelés matériel (hardware en anglais). Le logiciel (software en anglais) regroupe l'ensemble des programmes, des langages et des systèmes d'exploitation qui permettent d'exploiter les capacités de la machine. Un ordinateur communique avec son environnement grâce à des dispositifs d'entrée et de sortie (comme le clavier, l'écran, l'imprimante ou le modem). Les résultats intermédiaires sont stockés dans des mémoires auxiliaires telles que les bandes magnétiques, les disques magnétiques, et les disques optiques. Le cœur d'un ordinateur est l'unité centrale, où sont exécutées les instructions. Les ordinateurs sont utilisés dans divers domaines (traitement de texte, gestion, calcul scientifique), ce qui nécessite l'utilisation de langages de programmation adaptés à chaque application. Le principal avantage de l'ordinateur réside dans sa rapidité de calcul et son accès rapide aux informations.

I.2 Naissance du nombre et du calcule :

L'histoire des premières machines à calculer est une aventure fascinante qui remonte à plusieurs siècles. Elle reflète la volonté humaine de simplifier et d'automatiser les opérations arithmétiques, facilitant ainsi les calculs complexes et réduisant les erreurs humaines. Voici un aperçu historique des premières machines à calculer.

I.2.1 L'Abacus (3000 av. J.-C.) :

L'une des premières méthodes connues pour effectuer des calculs est l'abacus, apparu dans l'Antiquité. Ce dispositif manuel, composé de perles ou de cailloux se déplaçant le long de tiges ou dans des rainures,

était utilisé dans plusieurs civilisations anciennes, notamment en Mésopotamie, en Égypte, en Chine et en Grèce. L'Abacus permet de réaliser des opérations arithmétiques de base comme l'addition, la soustraction, la multiplication et la division.

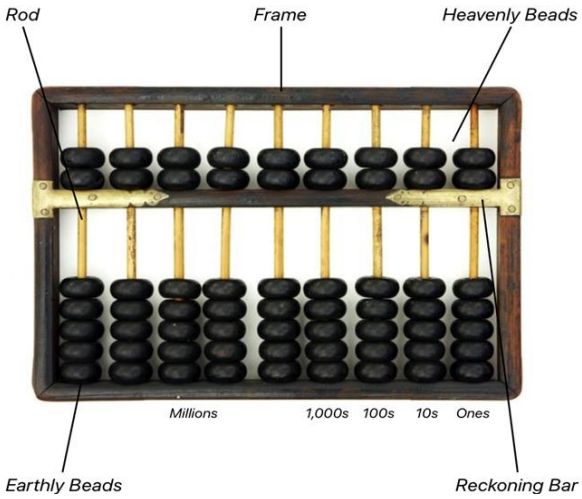


Figure I.1: *L'Abacus (Boulier Chinois).*

I.2.2 La Pascaline (1642) :

Conçue par **Blaise Pascal**, la Pascaline est considérée comme la première calculatrice mécanique. Elle était capable d'effectuer des additions et des soustractions en utilisant une série de roues dentées. Bien que rudimentaire, la Pascaline représente une avancée significative, car elle permettait d'automatiser des calculs simples sans erreur humaine.

Figure I.2: *La machine à calculer de Blaise PASCAL.*

I.2.3 La Machine de Leibniz (1673) :

Inspiré par la Pascaline, le mathématicien allemand **Gottfried Wilhelm Leibniz** améliora le concept en créant une machine capable de réaliser non seulement des additions et des soustractions, mais aussi des multiplications et des divisions.

La machine de **Leibniz**, ou **"Stepped Reckoner"**, utilisait une roue cylindrique appelée **"tambour de Leibniz"**, qui reste un principe fondamental dans la conception de calculatrices mécaniques ultérieures.

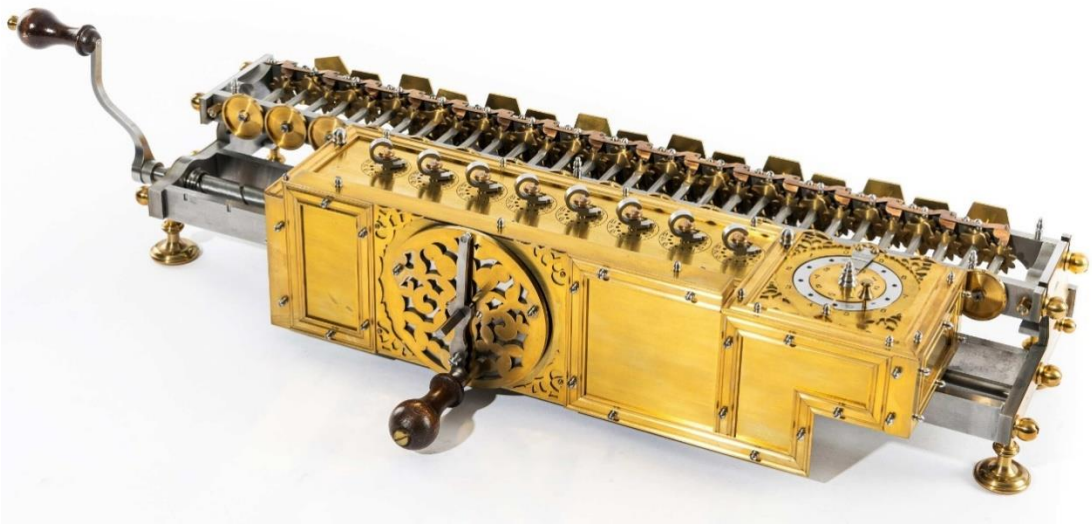


Figure I.3: *La Machine de Leibniz.*

I.2.4 La Machine de Babbage (19e siècle) :

Charles Babbage, souvent appelé "le père de l'ordinateur", imagina deux machines révolutionnaires : la machine à différences et la machine analytique.

La Machine à Différences (1822) : Destinée à automatiser les calculs des tables de logarithmes, cette machine devait réduire les erreurs courantes dans les calculs manuels.

La Machine Analytique (1837) : Babbage conceptualisa cette machine comme un ordinateur mécanique programmable, avec une unité centrale de traitement, une mémoire, et des instructions programmables. Bien que jamais construite de son vivant, la machine analytique est considérée comme le précurseur des ordinateurs modernes.

L'utilisateur décrit la séquence des opérations que doit effectuer la machine sur cette bande. Elle est introduite dans la machine à chaque nouvelle exécution. En effet, si la machine de Babbage est capable de mémoriser des résultats intermédiaires, elle ne dispose d'aucun moyen pour mémoriser les programmes dont le support est toujours externe,

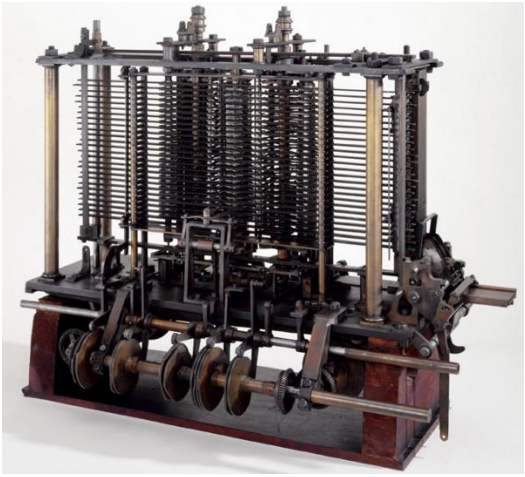


Figure I.4: *La Machine de Babbage.*

Dans cette machine apparaissent les notions de mémoire (le magasin) et de processeur (le moulin).

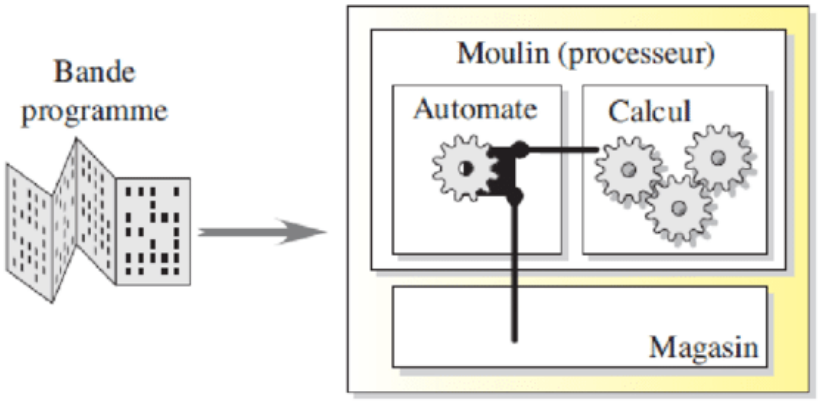


Figure I.5: *La machine analytique de Babbage.*

I.2.5 Le Comptomètre (1887) :

Inventé par **Dorr Felt**, cette machine pouvait effectuer des calculs simplement en appuyant sur les touches, sans besoin de tourner de manivelle. Il était largement utilisé pour les calculs commerciaux.

Figure I.6:*Comptomètre.*

I.2.6 Les Machines Électromécaniques (Début du 20e siècle) :

Avant même l'utilisation de l'électricité, on voit déjà se profiler les principes de base de l'informatique moderne. Avec l'avènement de l'électricité, les machines à calculer commencèrent à intégrer des composants électromécaniques, ce qui améliorait leur vitesse et leur fiabilité. Toutes ces machines traitent des données et fournissent un résultat. Ils sont été formalisés à l'ère des calculateurs électriques par **John Von Neumann (1903-1957).**

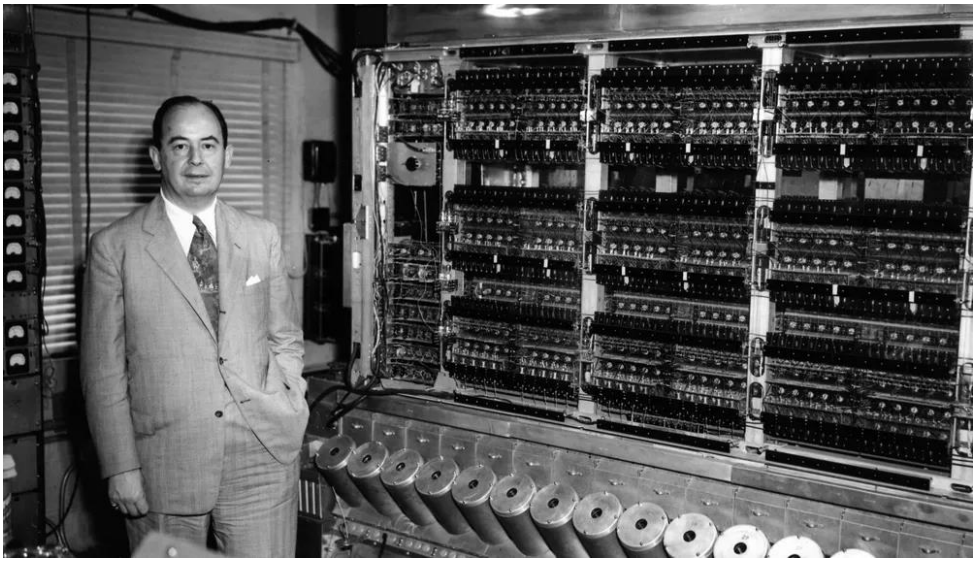


Figure I.7: *John Von Neumann.*

Les ordinateurs actuels fonctionnent tous selon le principe proposé en 1945 par le mathématicien **John Von Neumann** dans lequel les données et les instructions sont stockées dans la même mémoire. **[1]**

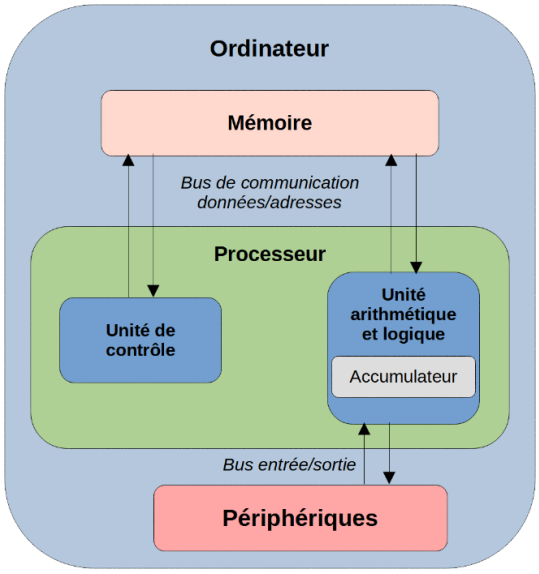


Figure I.8: *La machine de John Von Neumann.*

Ce modèle comporte quatre types de composants :

- Une **unité arithmétique et logique (UAL)**
- Une **unité de contrôle**
- La **mémoire** de l'ordinateur
- Les **périphériques d'entrée-sortie**.

Ces quatre composants forment deux grands circuits :

- Le **processeur** (ou microprocesseur ou CPU) dans lequel on trouve l'unité de contrôle et l'UAL
- La **mémoire**.

Ces deux circuits sont reliés entre eux par des liaisons physiques (câbles ou circuits intégrés) qui constituent un ou plusieurs **bus de communication**, notamment un **bus de données** et un **bus d'adresses**. Enfin, pour communiquer avec d'autres parties de l'ordinateur (les **périphériques**), le CPU dispose de **bus d'entrée/sortie**.

I.2.7 Le passage à l'électronique (1938-1953) :

Le passage à l'électronique, s'est fait grâce à l'invention du tube à vide. Il permet de produire un courant direct d'électrons dans un tube sous vide capable de générer deux états : **ON/OFF**, **Figure 2.9**. A l'aide d'interrupteurs **fermés** pour **"vrai"** et **ouverts** pour **"faux"** il était possible d'effectuer des opérations logiques en associant le nombre **" 1 "** pour **"vrai"** et **"0"** pour **"faux"**. Ce codage de l'information est nommé base **Binaire**. C'est avec ce codage que fonctionnent les ordinateurs modernes. Il est possible de représenter physiquement cette information binaire par un signal électrique qui lorsqu'elle atteint une certaine valeur, correspond à la valeur 1.

Figure I.9: *Tube à vide.*

I.2.8 L'ère du Transistor (1953-1963) :

Inventé en 1947 par les Américains **John Bardeen**, **William Shockley** et **Walter Brattain**, le **transistor** est le composant fondamental de tout ordinateur moderne.

Dans le cadre de son utilisation en informatique, c'est essentiellement un interrupteur capable de laisser passer le courant en fonction d'un signal électrique.

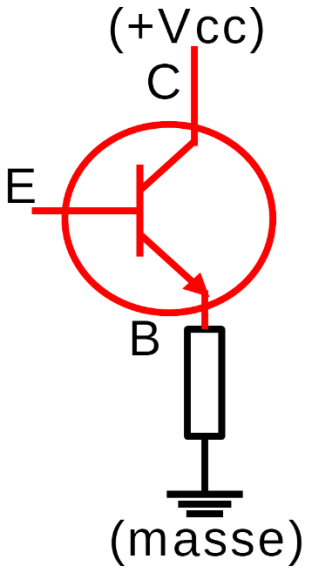


***Figure I.10**: Schéma d'un transistor.*

La tension de référence est reliée au collecteur, et on peut lire le signal aux bornes de la résistance située entre la base et la masse. Si une tension suffisante est appliquée à l'émetteur, le transistor laisse passer le courant et la tension de référence se retrouve au point **B**. Dans le cas contraire, le point **B** reste au potentiel de la masse, soit **0V**.

Ainsi un transistor peut opérer des **0** et des **1** qui correspondront au potentiel haut ou bas.

I.3 Éléments d'architecture d'un ordinateur :

Pour commencer, interrogeons-nous sur la signification-même du terme **« informatique »**

I.3.1 Définition :

INFORMATIQUE : (**INFOR**mation auto**MATIQUE**), définit la science de traitement automatique de l'information (c.à.d. automatiser l'information que nous manipulons). Cette informatisation permettra de réaliser un gain considérable en temps et en effort.

I.3.2 Présentation de l'ordinateur :

L'architecture d'un ordinateur se réfère à la manière dont ses composants internes sont organisés et interagissent pour exécuter des instructions et traiter des données. Elle englobe le matériel (hardware), les logiciels (software), et les interfaces de communication.

Chaque élément joue un rôle crucial dans le fonctionnement global du système, assurant que l'ordinateur peut effectuer une variété de tâches, allant du calcul simple à l'exécution de programmes complexes.

Tous les ordinateurs modernes **(Desktop, Laptop, Smartphones, Consoles, ...)** sont constitués des mêmes éléments :

- Une **carte mère** sur laquelle sont assemblés ou branchés tous les autres composants. Son rôle est d'assurer leur communication.

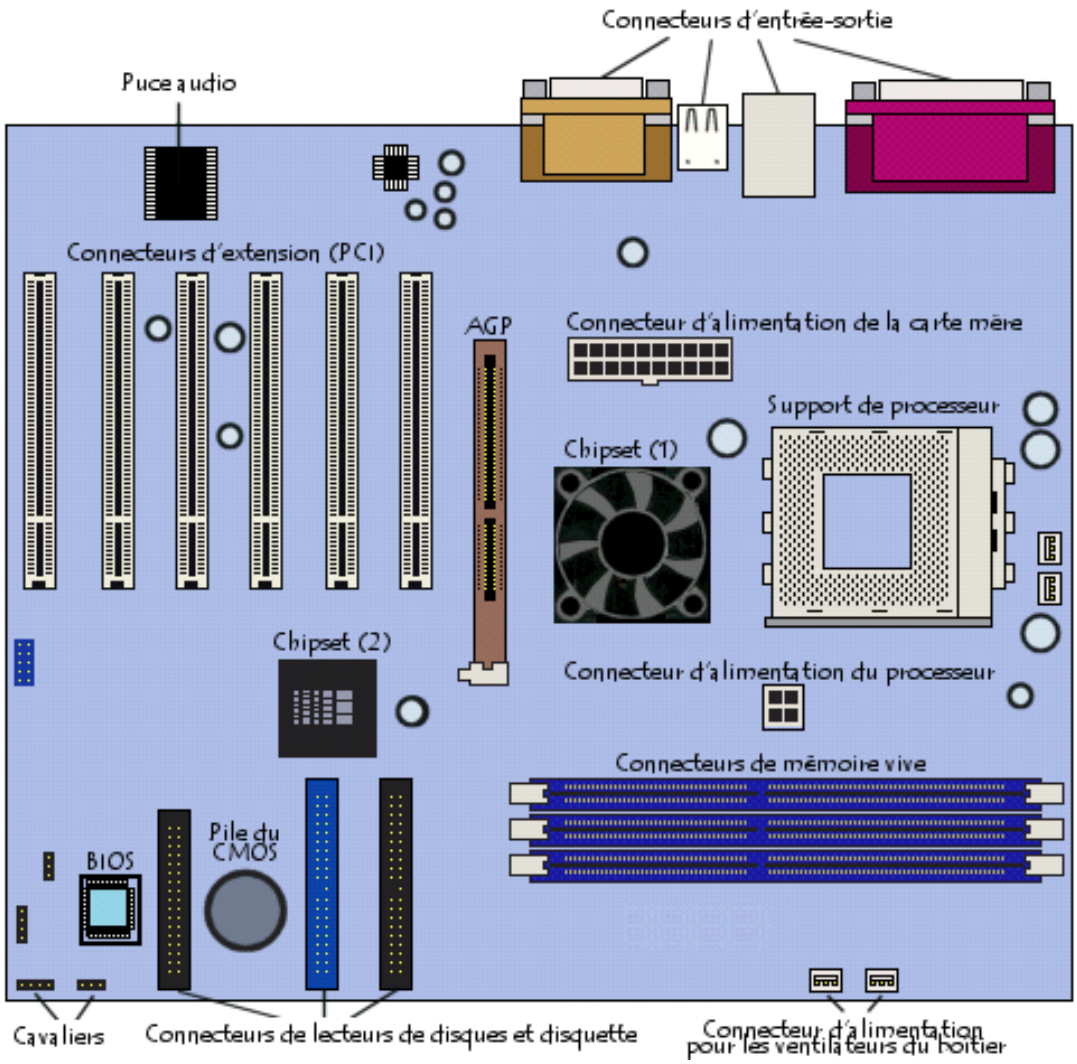


***Figure I.11**: Architecture de la carte mère.*

- Un **microprocesseur** (ou **CPU**, pour Central Processing Unit) qui a la charge de réaliser les opérations. Ce microprocesseur peut être épaulé par d'autres coprocesseurs qui s'occuperont de calculs spécialisés (typiquement dans un ordinateur moderne, une carte graphique).

Figure I.12: *Microprocesseur (Intel /AMD).*

- Des **mémoires** qui stockeront les données. Il en existe de plusieurs types :

  ✓ Mémoire vive (**RAM** - Random Access Memory) : Stocke temporairement les données et les instructions en cours d'utilisation par le processeur. La **RAM** est volatile, ce qui signifie que son contenu est perdu lorsque l'ordinateur est éteint.

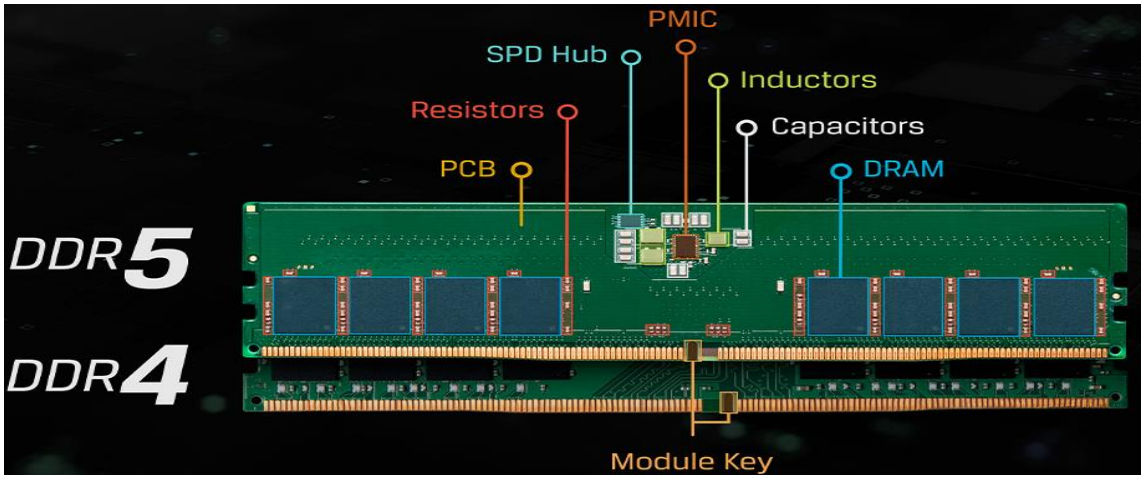


Figure I.13: *Random Access Memory.*

- ✓ Mémoire morte (**ROM** - Read-Only Memory) : Contient des instructions pré-enregistrées et non modifiables, telles que celles nécessaires au démarrage de l'ordinateur (**BIOS**).

Figure I.14*: Read-Only Memory.*

- ✓ Cache : Mémoire très rapide située à l'intérieur ou à proximité du **CPU**, utilisée pour stocker des données fréquemment utilisées, réduisant ainsi le temps d'accès à ces données.
- ✓ Mémoire secondaire : Comprend des dispositifs de stockage non volatiles comme les disques durs (**HDD**), les disques **SSD** (Solide State Drives), les **CD/DVD**, les clés **USB**, etc. Utilisée pour stocker des données et des programmes de manière permanente.

Figure I.15*: Mémoire secondaire.*

CHAPITRE-II-Systèmes de Numération.

II.1 Introduction :

Les systèmes de numérotation sont essentiels pour comprendre comment les nombres sont représentés, manipulés et utilisés dans divers contextes, notamment en mathématiques, en informatique et en électronique. Chaque système de numérotation est défini par une base, qui correspond au nombre de symboles utilisés pour représenter les valeurs. Par exemple, le système décimal, que nous utilisons couramment, est basé sur dix symboles (0 à 9), tandis que les systèmes binaire, octal, et hexadécimal, utilisés principalement en informatique, reposent respectivement sur deux, huit, et seize symboles.

Tableau II.1: *Bases de système de numérotation.*

Système	Base	Symboles utilisés
Binaire	2	01
Octal	8	01234567
Décimal	10	0 1 2 3 4 5 6 7 8 9
Hexadécimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

L'importance des systèmes de numérotation réside dans leur capacité à simplifier les calculs, à faciliter la communication des informations numériques, et à permettre une manipulation efficace des données. En informatique, par exemple, le système binaire est essentiel, car il correspond directement aux deux états possibles d'un circuit électronique (marche/arrêt). Comprendre les différents systèmes de numérotation et savoir convertir les nombres entre ces systèmes est crucial pour les scientifiques, les ingénieurs, et les informaticiens.

II.2 Objectifs :

- Traiter en détails les différents systèmes de numération : systèmes décimaux,binaire, octal et hexadécimal.
- Convertir un nombre d'un système de numération en un autre.
- Traiter les opérations arithmétiques sur les nombres.
- Coder une information dans un format numérique.

II.3 systèmes de numération :

Pour qu'une information numérique soit traitée par un circuit, elle doit être mise sous forme adaptée à celui-ci. Pour cela Il faut choisir un système de numération de base **B** (**B** un nombre entier naturel **≥ 2**).

De nombreux systèmes de numération sont utilisés en technologie numérique. Les plus utilisés sont les systèmes : **Décimal (base 10)**, **Binaire (base 2)**, **Octal (base 8)** et **Hexadécimal (base 16)**.Le tableau ci-dessous représente un récapitulatif sur ces systèmes :

Tableau II.2: *la correspondance entre divers systèmes de bases différentes.*

Décimal	Binaire	Octal	Hexadécimal
0	0	0	**0**
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

II.3.1 La représentation polynomiale d'un nombre :

Tout nombre **N** peut se décomposer en fonction des puissances entières de la base de son système de numération. Cette décomposition s'appelle la forme **polynomiale** du nombre **N** et qui est donnée par :

$$N = a_n.b^n + a_{n-1}.b^{n-1} + a_{n-2}.b^{n-2} + \ldots + a0.\ b\ 0$$

- **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 :** Base du système de numération, elle représente le nombre des différents chiffres qu'utilise ce système de numération **[4]**

II.3.2 Rang d'un chiffre de numération :

Le système Décimale le système de numération le plus pratique actuellement

- Le 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

II.3.3 Le Système Décimal (Base 10) :

- **Symbole : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
- **Fonctionnement :** Dans le système décimal, chaque chiffre d'un nombre a une valeur en fonction de sa position (ou place) dans le nombre. Par exemple, le nombre **543**peut être décomposé en $5\times10^2+4\times10^1+3\times10^0$.
- **Applications :** Le système décimal est le système de numérotation le plus couramment utilisé dans la vie quotidienne pour compter, mesurer, et effectuer des calculs.

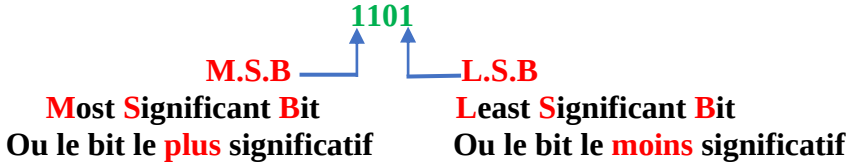
II.3.4 Le Système Binaire (Base 2) :

- **Symbole** : **0, 1**
- **Fonctionnement :** Le système binaire utilise uniquement deux symboles, **0** et **1**. Chaque chiffre (appelé bit) représente une puissance de **2**. Par exemple, le nombre binaire **1101** peut être décomposé en $1\times2^3+1\times2^2+0\times2^1+1\times2^0$, ce qui équivaut à **13** en décimal.
- **Applications :** Le système binaire est fondamental en informatique et en électronique numérique, où il représente les deux états possibles d'un circuit (marche/arrêt).

Remarque :

Le dernier bit de droite s'appelle le bit de poids le plus faible **L.S.B** **(Least Significant Bit).**

Celui de gauche s'appelle le bit de poids le plus fort **M.S.B (Most Significant Bit).** **[8]**



II.3.5 Le Système Octal (Base 8) :

- **Symbole : 0, 1, 2, 3, 4, 5, 6, 7**
- **Fonctionnement :** Le système octal utilise huit symboles, de **0** à **7**. Chaque chiffre représente une puissance de **8**. Par exemple, le nombre octal **157**peut être décomposé en $1\times8^2+5\times8^1+7\times8^0$, ce qui équivaut à **111** en décimal.
- **Applications :** Le système octal est utilisé en informatique pour simplifier la représentation des nombres binaires. Il est plus compact que le binaire tout en étant plus facile à convertir en binaire

II.3.6 Le Système Hexadécimal (Base 16) :

- **Symbole : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**
- **Fonctionnement :** Le système hexadécimal utilise seize symboles, de **0** à **9**, puis **A** à **F** pour représenter les valeurs **10** à **15**. Chaque chiffre représente une puissance de **16**. Par exemple, le nombre hexadécimal **2A3** peut être décomposé en $2\times16^2+A\times16^1+3\times16^0$, où A équivaut à **10** en décimal, ce qui donne un total de **675**en décimal.
- **Applications :** Le système hexadécimal est largement utilisé en programmation et en informatique pour représenter de manière compacte des valeurs binaires. Chaque chiffre hexadécimal représente exactement **quatre bits**.

II.4 Conversion d'un système de numération à une autre :

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 changement de base le plus simple est le passage entre la base 2 et les bases 8 ou 16. En effet, les valeurs 8 et 16 étant des puissances de 2 **($8 = 2^3$; $16 = 2^4$),** chaque bloc de 3 ou **4bits**correspond à un symbole **octal** ou **hexadécimal**. **[1]**

II.4.1 Représentation du Système Octal en bit :

Le système octal (base 8) est un système de numérotation qui utilise huit symboles distincts : 0, 1, 2, 3, 4, 5, 6, 7. Chaque chiffre dans un nombre octal peut être représenté en binaire (base 2) en utilisant exactement trois bits, car $2^3=8$, ce qui permet de représenter les 8 combinaisons possibles (000 à 111)

Tableau II.3: *Représentation des chiffres octaux en bits.*

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

Exemple de Conversion d'un Nombre Octal en Binaire :

Prenons un nombre octal, par exemple 345 en octal, et convertissons-le en binaire :

1. **3** en octal se convertit en **011** en binaire.
2. **4** en octal se convertit en **100**en binaire.
3. **5** en octal se convertit en **101**en binaire.

Ainsi, le nombre 345 en octal est représenté par 011100101 en binaire.

Conversion Inverse : Binaire à Octal :

Pour convertir un nombre binaire en octal, il suffit de regrouper les **bits** du nombre binaire en paquets de trois, en partant de la droite, et de convertir chaque groupe en son équivalent octal :

- Par exemple, le nombre binaire **110101010**en groupes de **trois** devient **110 101 010**.
- En convertissant chaque groupe, on obtient : 110 → 6, 101 → 5, 010 → 2.
- Ainsi, **110101010**en binaire correspond à **652**en octal.

II.4.2 Représentation du Système Hexadécimal en bit :

Le système hexadécimal (base 16) est un système de numérotation qui utilise seize symboles distincts : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Chaque chiffre hexadécimal peut être représenté en binaire (base2) en utilisant exactement quatre bits, car 2^4**=16**, ce qui permet de représenter les 16 combinaisons possibles (0000 à 1111). **[6]**

Tableau II.4: *Représentation des chiffres hexadécimaux en bits.*

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

Exemple de Conversion d'un Nombre Hexadécimal en Binaire

Prenons un nombre hexadécimal, par exemple **2F3** en hexadécimal, et convertissons-le en binaire :

1. 2 en hexadécimal se convertit en 0010 en binaire.
2. F en hexadécimal se convertit en 1111 en binaire.
3. 3 en hexadécimal se convertit en 0011 en binaire.

Ainsi, le nombre **2F3**en hexadécimal est représenté par **001011110011**en binaire.

Conversion Inverse : Binaire à Hexadécimal

Pour convertir un nombre binaire en hexadécimal, il suffit de regrouper les bits du nombre binaire en paquets de quatre, en partant de la droite, et de convertir chaque groupe en son équivalent hexadécimal :

- Par exemple, le nombre binaire 110110111011 en groupes de quatre devient 1101 1011 1011.
- En convertissant chaque groupe, on obtient : 1101 → D, 1011 → B, 1011 → B.
- Ainsi, 110110111011 en binaire correspond à **DBB** en hexadécimal.

II.4.3 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.

Le terme "non signé" signifie que le nombre est toujours positif ou nul. [4]

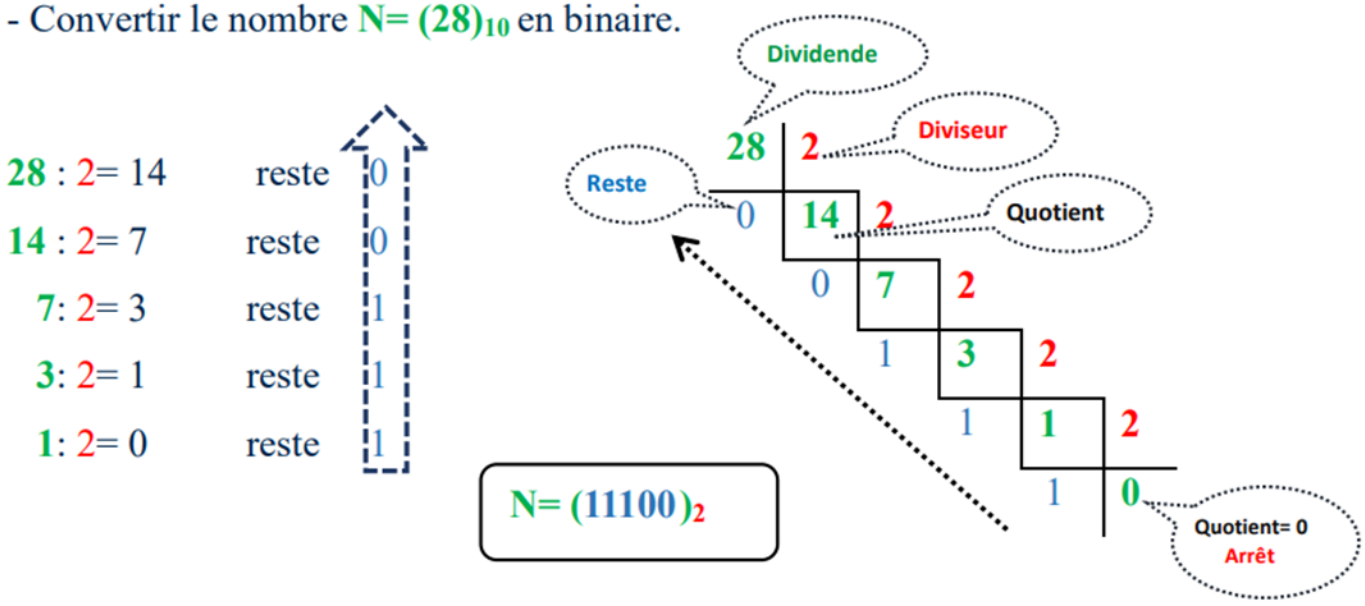
II.4.3.1 Codage :

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 »**). **[4]**

Example 1 :

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



II.4.3.2 Décodage :

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

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

Example 1 :

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

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

$N = (81)_{10}$

Example 2 :

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

$(1011)_2 = 1x2^3 +1x2^2 +0x2^1+1x2^0 = 13$

$N = (13)_{10}$

II.4.3.2 Transcodage :

Le transcodage d'un nombre est le passage entre deux systèmes autres que le décimal. Permet d'effectuer la conversion d'un nombre N de base b1 quelconque tel que $b1 \neq 10$ vers une autre baseb2 tel que $b2 \neq 10$.

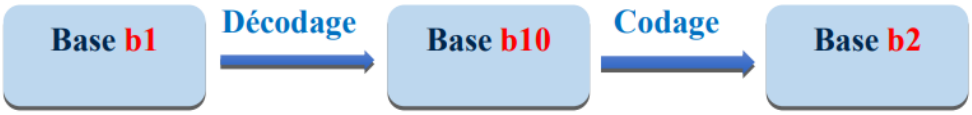
On a deux types de transcodage :

a. Transcodage Indirect (cas général) :

Cette méthode est valable quel que soit b1et b2. Un Transcodage Indirect passe d'abord par un décodage suivi d'un codage en passant par un format de référence ou intermédiaire.

➢ **Étapes Générales du Transcodage Indirect :**

1. **Décodage**: Convertir les données du format source (par exemple, binaire, octal, ou hexadécimal) en un format intermédiaire, souvent le système décimal.
2. **Recodage**: Convertir ensuite le format intermédiaire en le format cible souhaité.

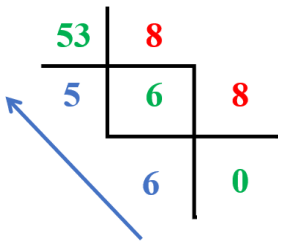


- **Exemple 1 :** Convertir le nombre binaire 110101 en octal via un transcodage indirect.

Décodage: Convertir le binaire en décimal.

110101en binaire = $1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0=32+16+4+1=53$en décimal.

Recodage: Convertir le décimal en octal.



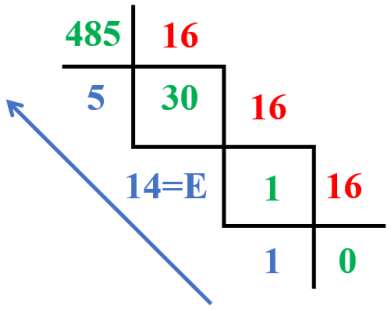
Résultat: **110101**en binaire = **65**en octal.

- **Exemple 2 :** Convertir le nombre octal **745**en hexadécimal via un transcodage indirect.

Décodage: Convertir l'octal en décimal.

745en octal= $7\times8^2+4\times8^1+5\times8^0=448+32+5=485$en décimal.

Recodage: Convertir le décimal en hexadécimal.



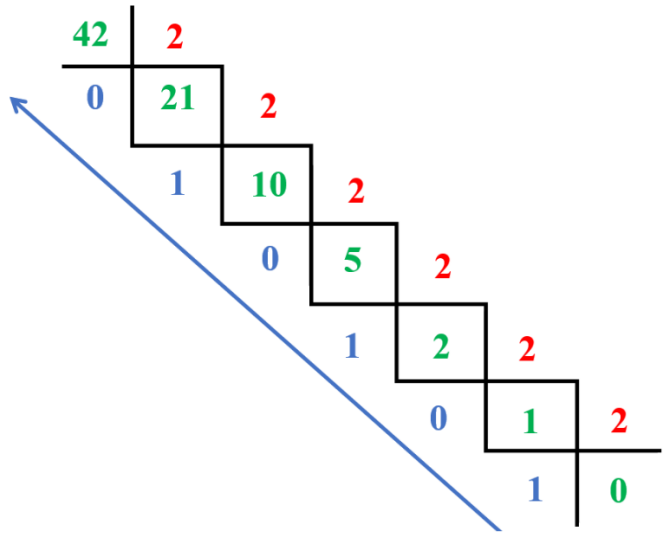
Résultat: **745**en octal = **1E5** en hexadécimal.

- **Exemple 3 :** Convertir le nombre hexadécimal **2A** en binaire via un transcodage indirect.

Décodage: Convertir l'hexadécimal en décimal.

2Aen hexadécimal =$2\times16^1+10\times16^0=32+10=42$en décimal.

Recodage: Convertir le décimal en binaire.



Résultat: **2A**en hexadécimal = **101010**en binaire

b. Transcodage Direct (cas particulier) :

Le transcodage direct est un processus qui consiste à convertir directement des informations codées d'un format à un autre sans passer par une étape intermédiaire de décodage dans un format de données non codé (comme le décimal).

- **Exemple 1 :** Convertir le nombre binaire **110110101** en octal via un transcodage direct.
  - Regrouper les **bits** par paquets de **trois**: **1 101 101 101**
  - Ajouter des zéros à gauche si nécessaire pour compléter le groupe de gauche : **001101101** 101
  - Convertir chaque groupe en octal : **001 (1), 101 (5), 101 (5), 101 (5)**

  Résultat : Le nombre binaire **110110101**est équivalent à **1555**en octal.

- **Exemple 2 :** Convertir le nombre binaire **110110101011**en hexadécimal via un transcodage direct.
  - Regrouper les **bits** par paquets de **quatre**: **1101 1010 1011**
  - Ajouter des zéros à gauche si nécessaire pour compléter le groupe de gauche et convertir chaque groupe en hexadécimal:**1101 (D), 1010 (A), 1011 (B)**

  Résultat : Le nombre binaire **110110101011**est équivalent à **DAB** en hexadécimal.

- **Exemple 3 :** Convertir le nombre octal 764 en hexadécimal via un transcodage direct.
  - Convertir chaque chiffre octal en binaire :
    - **7 → 111**
    - **6 → 110**
    - **4 → 100**
  - Le nombre binaire obtenu est donc **111110100**.
  - Regrouper les **bits** par **quatre**: **000111110100**(ajouter des zéros à gauche si nécessaire).
  - Convertir en hexadécimal : **0001(1), 1111(F), 0100 (4)**

  Résultat : Le nombre octal **764**est équivalent à **1F4**en hexadécimal.

➢ Confirmer les résultats de la conversion du nombre octal **764**en hexadécimal de l'**Exemple 3** précédent, avec l'utilisation d'un **transcodage indirect** en passant d'abord par le système décimal.

Étape 1 : Conversion de l'octal en décimal

Le nombre octal **764**est converti en décimal en utilisant la formule :

764 octal=$7\times8^2+6\times8^1+4\times8^0$

Calculons : $7\times82=7\times64=448$

$6\times81=6\times8=48$

$4\times80=4\times1=44$

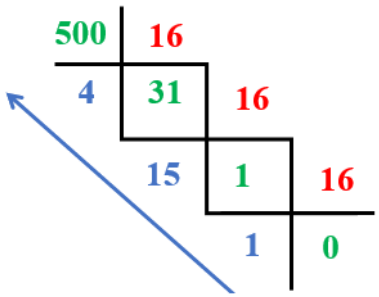
Additionnons ces résultats :

$448+48+4=500$

Ainsi, **764**en octal est équivalent à **500**en décimal.

Étape 2 : Conversion du décimal en hexadécimal

Nous convertissons maintenant le nombre décimal **500**en hexadécimal :



15est représenté par **F**en hexadécimal

En lisant les restes de bas en haut, on obtient le nombre hexadécimal **1F4**.

Conclusion

Le nombre octal **764**est équivalent à **1F4**en hexadécimal.

II.4.4 Représentation des nombres Fractionnaires non signé :

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

- Codage
- Décodage
- Transcodage

II.4.4.1 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).

- **Exemple 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.

➔ $N = (4,57)_{10} = (100,100)_2$

- **Exemple 2 :**

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

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

$0,732 \times 8 = 5\ ,\ 856$

$0,856 \times 8 = 6\ ,\ 848$

$0,848 \times 8 = 6\ ,\ 784$

$0,784 \times 8 = 6\ ,\ 272$

$0,272 \times 8 = 2\ ,\ 176$

➔ $N = (0,732)_{10} = (0,56662)_8$

II.4.4.2 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:**

Soit à déterminer la valeur décimale du nombre N

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

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

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

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

II.4.4.3 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. **[4]**

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

$(5)_8 = (101)_2$
$(7)_8 = (111)_2$
$(1)_8 = (001)_2$
$(4)_8 = (100)_2$
$(6)_8 = (110)_2$

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

- **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

$N = (1110011101,01110001)_2 = (\underbrace{0011}_{3}\ \underbrace{1001}_{9}\ \underbrace{1101}_{D}\ ,\ \underbrace{0111}_{7}\ \underbrace{0001}_{1})_2 = (39D,71)_{16}$

II.5 Opérations de base (Arithmétiques) dans le système binaire :

Les opérations arithmétiques de base dans le système binaire incluent l'addition, la soustraction, la multiplication et la division. Ces opérations suivent des règles similaires à celles utilisées dans le système décimal, mais adaptées au système binaire, qui ne comporte que deux chiffres : **0** et **1**.

II.5.1 Addition Binaire :

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

L'addition en binaire est assez simple et suit les règles suivantes :

- **0+0=0**
- **0+1=1**
- **1+0=1**
- **1+1=10 (ce qui équivaut à 0 avec une retenue de 1)**

Exemple :

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

Retenue : 1 1

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

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

Étapes de l'addition :

1. **Additionnez les bits les moins significatifs (à droite) :**

   1+0=1

2. **Additionnez le bit suivant :**

   1+1=10 (écrire 0 et retenue1)

3. **Additionnez le bit suivant avec la retenue :**

   1+1+1(la retenue) =11 (écrire 1 et retenue1)

4. **Additionnez le bit suivant avec la retenue :**

   0+0+1(la retenue) =1

5. **Additionnez le bit suivant :**

   1+0=1

6. **Additionnez les bits les plus significatifs :**

   1+1=10

II.5.2 Soustraction Binaire :

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

La soustraction en binaire est similaire à celle en base 10, mais ici, vous devrez "emprunter" si nécessaire :

- **0−0=0**
- **1−0=1**
- **1−1=0**
- **0−1=1 (emprunter 1 du bit à gauche, ce qui transforme 0 en 10)**

Exemple:

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

$$\begin{array}{r} 1\ {}_1 1\ {}_1 0\ {}_1 0\ 1\ 1\ 1 \\ =\ 1\ {}_1 1\ {}_1 0\ 1\ 1\ 0\ 0 \\ \hline = \ 0\ 1\ 1\ 1\ 0\ 1\ 1 \end{array}$$

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

II.5.3 Multiplication Binaire :

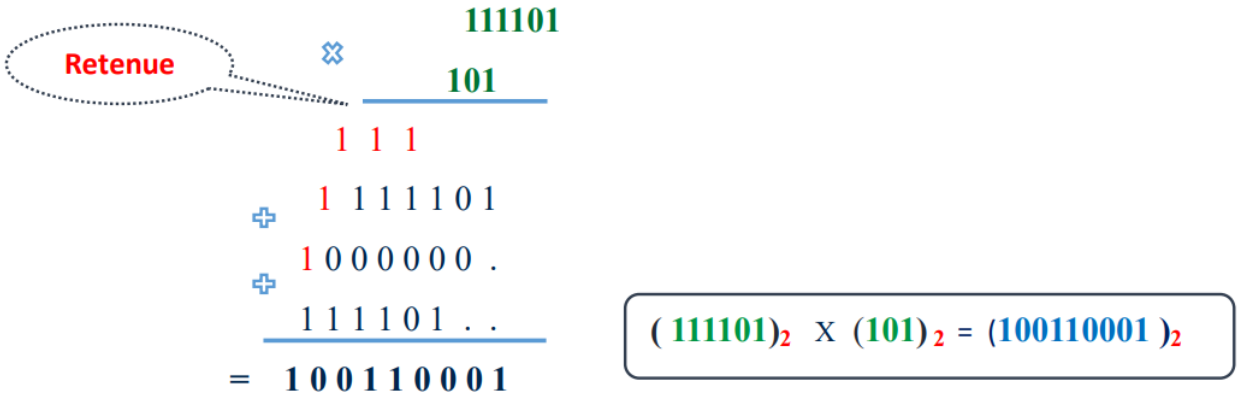
La multiplication binaire est une opération qui, à partir de deux nombres binaires **a** et b, donne un autre nombre binaire appelé produit. La multiplication permet d'éviter une addition répétée.

La **multiplication** binaire fonctionne comme la multiplication en base 10, en utilisant des décalages et des additions.

- **0×0=0**
- **0×1=0**
- **1×0=0**
- **1×1=1**

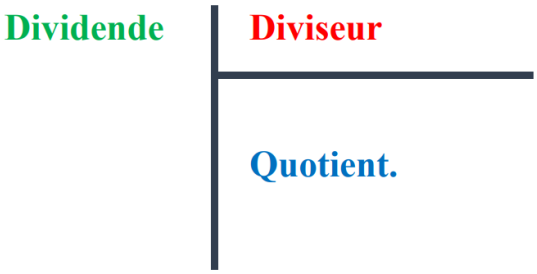
Exemple:

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



II.5.4 Division 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**.

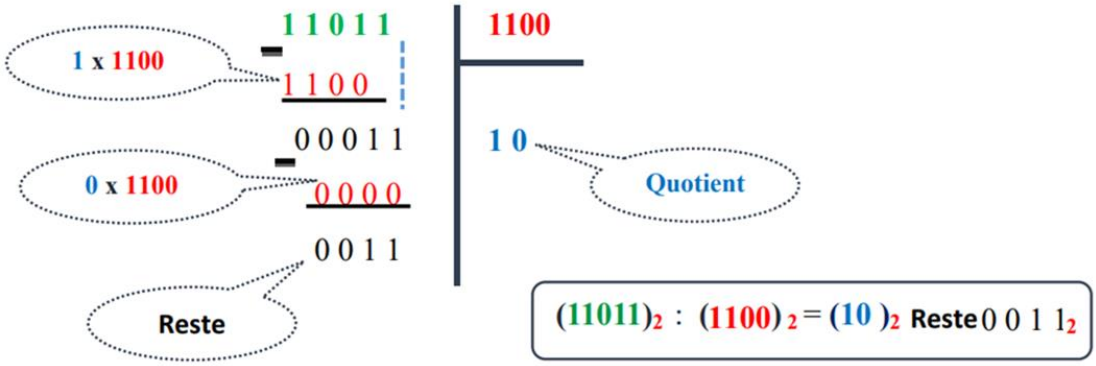


Remarque :

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

Exemple 1 :

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



Liste des Exercices :

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 :

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

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) $(111010)_2/(5)_{10}$

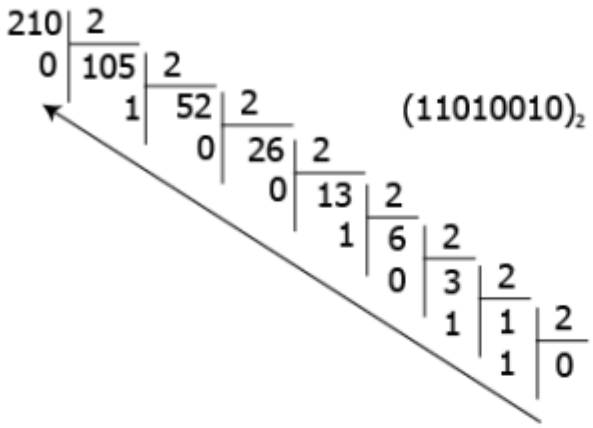
2) Effectuer l'addition suivante sur 1 octet

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

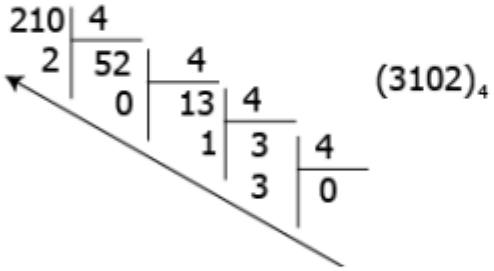
SOLUTION

Exercice 1:

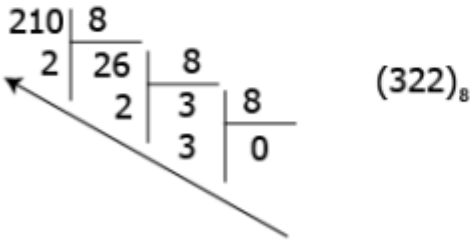
Base 2 :



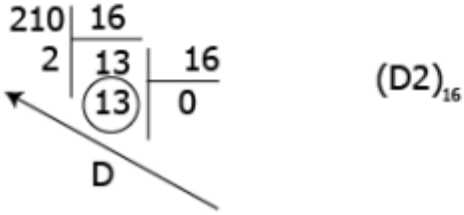
Base 4 :



Base 8 :



Base 16 :



Exercice 2 :

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

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

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

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

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

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

CHAPITRE-III- Représentation de l'information

III.1 Introduction :

La représentation de l'information dans une machine est la manière dont les données sont codées en binaire pour être manipulées et traitées efficacement par le matériel informatique. Les différents types de données (nombres, caractères, images, sons ou programmes) sont organisés et représentés selon des normes spécifiques qui permettent à la machine d'interpréter et d'exécuter des instructions en fonction des informations reçues.

III.2 Objectifs :

- Connaître le **codage binaire** en identifiant les différents types pour représenter des données dans des systèmes informatiques. **(Pur, BCD, GRAY, EXC-3).**
- Comprendre la manière dont les caractères sont traduits en informations binaires **(Code EBCDIC, Code ASCII, Code UTF-8)**.
- Apprendre à représenter des nombres entiers **(signés et non signés)** et fractionnaires **(Virgule fixe et flottante)** dans des systèmes binaires.

III.3 Le codage binaire (classification des codes) :

Il existe plusieurs méthodes pour représenter les informations en binaire. Chaque méthode a des applications spécifiques selon les besoins, comme la simplification des calculs ou l'amélioration de la transmission des données. On a plusieurs modes de représentation :

III.3.1 Codage Binaire Pur :

Le **codage binaire pur** (ou simplement **binaire**) est la représentation la plus directe des nombres en base 2. Chaque chiffre décimal est traduit en une séquence de bits correspondant à la valeur de ce nombre dans le système binaire.

- Chaque **bit** représente une puissance de 2.
- Il est utilisé de manière générale pour représenter les données dans les systèmes numériques, comme les nombres, les caractères, les instructions machine, etc.
- C'est la forme de codage binaire la plus couramment utilisée. **[5]**

Exemple :

Représentation des nombres en codage **binaire pur**:

- $5_{10} = (0101)_2$
- $13_{10} = (1101)_2$

Tableau III 1: *représente le code binaire Pur.*

Valeur Décimal	**A_0**	**A_1**	**A_2**	**A_3**
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
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Pour convertir un nombre décimal en binaire, on décompose le nombre en puissances de 2 et on utilise **0** et **1** pour indiquer la présence ou l'absence de chaque puissance.

Conversion de13_{10}:

$13 = 8 + 4 + 0 + 1 = 2^3 + 2^2 + 2^0 \Rightarrow 1101_2$

III.3.2 Codage Binaire BCD (Binary Coded Decimal) :

Le codage BCD (Binary Coded Decimal) est un système de représentation où chaque chiffre d'un nombre décimal est codé individuellement en binaire. Ce système est particulièrement utile pour afficher les nombres décimaux sur des écrans ou dans des systèmes où la conversion rapide entre binaire et décimal est nécessaire.

- Chaque chiffre décimal **(de 0 à 9)** est représenté par un groupe de **4 bits**.
- Un nombre entier est décomposé en chiffres décimaux, et chaque chiffre est converti en son équivalent binaire.

Tableau III 2: *représente le code binaire BCD.*

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

Exemple 1 :

Représentation du nombre59_{10} **en BCD :**

$5_{10} = 0101_2$

$9_{10} = 1001_2$

Ainsi, 59_{10} **en BCD** est représenté par :

$$59_{10}=01011001_{BCD}$$

Exemple 2 :

Un autre exemple avec 175_{10} :

$1_{10} = 0001_2$

$7_{10} = 0111_2$

$5_{10} = 0101_2$

Représentation **BCD**de175_{10} :

$$175_{10} = 000101110101_{BCD}$$

III.3.3 Codage Binaire Réfléchi (ou Code de Gray) :

Le **code de Gray** est un système de codage binaire dans lequel deux nombres consécutifs ne diffèrent que par un seul bit. Cela minimise les erreurs lors de la transition d'un nombre à un autre, notamment dans les systèmes où les erreurs de signal peuvent se produire.

III.3.3.1 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 du 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

En résumé Pour convertir un nombre en binaire en code de Gray, suivez ces étapes systématiques :

Étape 1 :

Le premier **bit** du **code de Gray** est identique au premier **bit** du nombre **binaire**.

- **Gray [0] = Binaire [0]**
- Le premier bit (bit du poids fort) du code de Gray est simplement copié à partir du premier bit du nombre binaire.

Étape 2 :

Appliquer l'opération **XOR** (OU **Exclusif**) entre chaque bit consécutif du nombre binaire pour obtenir le reste du **code de Gray**.

- Pour chaque bit i (à partir du 2ème), on fait un **XOR** entre le **bit i**et le **bit i−1**du nombre binaire :

$$Gray[i] = Binaire[i] \oplus Binaire[i-1]$$

L'opérateur **XOR** donne :

- **1⊕1=0**
- **1⊕0=1**
- **0⊕1=1**
- **0⊕0=0**

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

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ +\quad 1\ 0\ 1 \\ \hline = \ 1\ 1\ 1\ 1 \end{array}$$ en GRAY

N= (1010)$_2$ = (1111) $_{gray}$

Voici la conversion des nombres de 0 à 7 en code de Gray :

Tableau III 3: *représente le code de Gray.*

Nombre Décimal	Binaire pur	Code de Gray
0	**000**	**000**
1	**001**	**001**
2	**010**	**011**
3	**011**	**010**
4	**100**	**110**
5	**101**	**111**
6	**110**	**101**
7	**111**	**100**

III.3.3.2 Passage du Code GRAY au binaire :

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

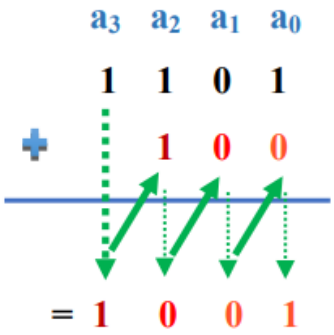
1. On garde a_3 le bit du poids fort
2. On ajoute le bit du poids fort a_3 à a_2
3. Le résultat trouvé dans le **2°/** est ajouté à a_1
4. Le résultat trouvé dans le **3°/** est ajouté à a_0

Remarque :

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

Exemple :

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



III.3.4Codage Binaire Excédé de Trois (Excess-3) :

Le **codage excès-3 (Excess-3)** est un système de codage binaire qui consiste à ajouter 3 à chaque chiffre décimal avant de le convertir en binaire. Il est utilisé pour simplifier certaines opérations arithmétiques et pour éviter les erreurs dans les systèmes numériques. **[2]**

- Chaque chiffre **décimal** est converti en ajoutant **3** avant de le représenter en binaire.
- Ce codage est principalement utilisé dans les circuits de calcul pour simplifier l'addition de nombres décimaux.

Tableau III 4: *représente le code de Excess-3.*

Décimal	Excédé de Trois (Excess-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

Exemple :

Représentation du nombre 5_{10}en code **Excès-3**:

- Ajoutez **3** à chaque chiffre : $5+3=8$
- Convertissez le résultat en binaire : $8_{10}=1000_2$

Représentation **Excès-3**du nombre 59_{10} :

- 5_{10} en **excès-3**: $5+3=8 \Rightarrow 1000_2$
- 9_{10}en **excès-3**: $9+3=12 \Rightarrow 1100_2$

Ainsi, 59_{10}est représenté en **excès-3**par :

$$59_{10} = 10001100_{\text{Excès-3}}$$

III.4 Représentation des caractères :

La représentation des caractères dans les systèmes informatiques permettre la manipulation, l'affichage et l'échange de texte.

III.4.1 Code ASCII (American Standard Code for Information Interchange) :

Le code **ASCII** est un standard de codage développé dans les années 1960 pour représenter des caractères textuels. Il est basé sur un ensemble de **128 caractères** qui incluent :

- Les lettres **majuscules** et **minuscules** de l'alphabet anglais.
- Les chiffres **(0-9).**
- Les **signes de ponctuation** et quelques **caractères de contrôle**.

- **7 bits (2^7 =128)** sont utilisés pour représenter chaque caractère **(128 caractères au total).**
- Un **octet (8 bits)** est souvent utilisé, avec un **bit de contrôle supplémentaire** pour d'autres fonctions.
- Il est largement utilisé dans les systèmes informatiques pour représenter du texte simple en anglais.

Tableau III 5*:Table ASCII.*

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]

Exemple :

Caractère	Code ASCII en décimal	Code ASCII en binaire
'A'	65	01000001
'a'	97	01100001
'0'	48	00110000

III.4.2 Code EBCDIC (Extended BinaryCodedDecimal Interchange Code) :

Le code EBCDIC est un jeu de caractères développé par **IBM** dans les années **1960** pour les gros systèmes (mainframes). Contrairement à l'**ASCII**, il utilise un jeu de 256 caractères, chacun codé sur 8 bits. **[3]**

- Chaque caractère est représenté par **8 bits** (1 octet), permettant de coder **256 caractères**.
- Utilisé principalement sur les systèmes **IBM**.
- Moins courant que **l'ASCII**, mais encore en usage dans certains systèmes traditionnels.

Exemple :

Caractère	Code EBCDIC en hexadécimal	Code EBCDIC en binaire
'A'	**C1**	**11000001**
'a'	**81**	**10000001**
'0'	**F0**	**11110000**

III.4.3 Code Unicode et UTF (Unicode Transformation Format) :

Le **Unicode** est un standard universel de codage des caractères qui permet de représenter des caractères provenant de presque toutes les langues du monde, ainsi que des symboles techniques, mathématiques, et bien d'autres. Il a été conçu pour résoudre les limitations des systèmes de codage comme **ASCII** et **EBCDIC**, qui ne peuvent pas représenter tous les caractères utilisés dans toutes les langues.

- **Unicode** peut coder plus de **1,1 million de caractères**, chacun étant représenté par un nombre unique appelé **code point**.
- Le format le plus courant de représentation d'Unicode est **UTF-8 (Unicode Transformation Format - 8 bits),** qui utilise entre **1 et 4 octets** pour coder chaque caractère.
  - **UTF-8**est compatible avec **l'ASCII** pour les **128 premiers caractères**.
  - **UTF-16**et **UTF-32**sont d'autres formats qui utilisent **2** ou **4 octets** fixes pour chaque caractère.

Exemple :

Le caractère **'A'** en **Unicode** a le code point **U+0041**, soit **65** en **décimal**, ce qui est identique à **l'ASCII**.
Le caractère **'€'** (symbole Euro) a le code point **U+20AC**, ce qui le représente en **UTF-8** par **3 octets**: **11100010 10000010 10101100**.

Unicode est aujourd'hui la norme dans la majorité des systèmes modernes, notamment sur le Web, dans les bases de données, et dans les applications multi-langues.

Tableau III 6*:Comparaison des systèmes de codage des caractères.*

Système	Taille (bits)	Nombre de caractères	Langues prises en charge	Utilisation principale
ASCII	7 bits	128	Anglais (caractères latins de base)	Textes simples
EBCDIC	8 bits	256	Anglais, quelques autres	Systèmes IBM
Unicode	1 à 4 octets (UTF-8)	+1,1 million	**Tous** les systèmes d'écriture	Web, bases de données, logiciels

III.5 Représentation des nombres :

Les systèmes informatiques doivent être capables de représenter et de manipuler différents types de nombres, tels que les **entiers**, les nombres **fractionnaires** ou à **virgule flottante**. La manière dont ces nombres sont codés dans un ordinateur est essentielle pour garantir des calculs précis et efficaces. Il existe plusieurs méthodes pour représenter les nombres en binaire, chacune étant adaptée à des besoins spécifiques. **[2]**

III.5.1 Représentation non signée :

Comme nous avons vu précédemment un aperçu sur les nombres entiers non signés (positifs) dans les conversions entre système. Sont représentés en binaire de la manière la plus simple, chaque bit correspond à une puissance de 2, et les valeurs sont simplement additionnées pour obtenir le nombre entier.

- Un nombre **non signé** n'a que des valeurs **positives** ou **nulles**.
- La plage des valeurs représentables dépend du nombre de **bits**. Par exemple, avec **n bits**, la valeur maximale est 2^n-1.

Exemple :

Avec **4 bits**:

- $0101_2=5_{10}$
- $1111_2=15_{10}$ (la valeur maximale avec **4 bits** est $2^4-1=15$).

III.5.2 Représentation avec signe et valeur absolue :

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**. **[9]**

III.5.3 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)**
- Complément à deux (**CP2**).

III.5.3.1 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 négatifs** 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 $(1111)_{SVA}$

III.5.3.2 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)

III.5.3.3 Approche complément à deux (CP2) :

Le complément à 2est la méthode la plus couramment utilisée pour représenter les nombres négatifs en binaire. 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**

Note :

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)	$[-(2^{n-1}), +(2^{n-1}-1)]$

III.5.3.4 Opérations en CP1 et CP2 :

- **Opérations arithmétiques 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

2ème 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**

3ème 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**

4ème 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**

❖ **Opérations arithmétiques 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

2ème 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**

3ème 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**

4ème 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**

III.5.4 Les nombres fractionnaires :

Un nombre réel (fractionnaires) **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.

III.5.4.1Représentation 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 ⟹ $N1_{max} = 2^n - 1$

- 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 :

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

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.

III.5.4.2 Virgule flottante :

a. 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

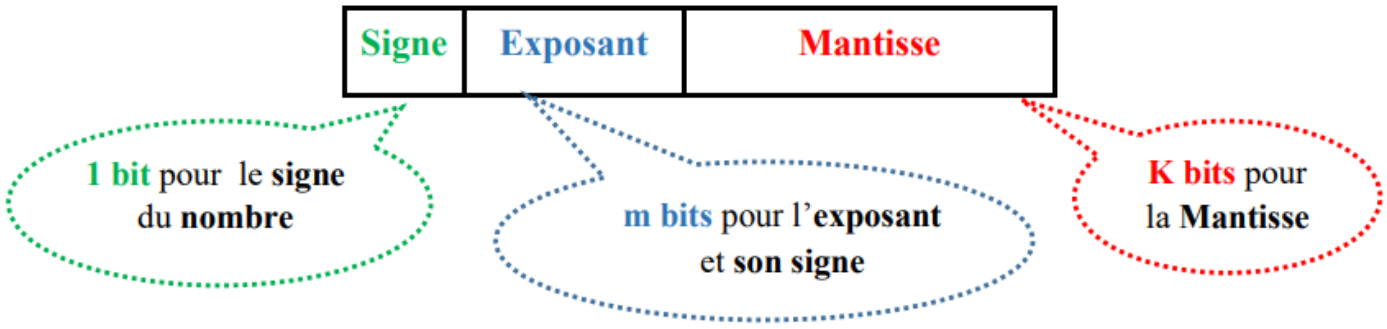
Note : On dit que la **mantisse** est **normalisée** si le premier chiffre après la **virgule** est différent de 0 et le premier chiffre avant la virgule est égale à 0

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 : $(100,01)_2 = +0,1000100 \times 2^{+3}$

0	0011	1000100

b. Représentation en virgule flottante selon la Norme IEEE -754 :

La Représentation en virgule flottante selon la **Norme IEEE -754** est un cas particulier de la représentation en virgule flottante des nombres réels.

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**

❖ **Norme IEEE-754 simple précision :**

Dans la Représentation simple précision on a :

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

Avec :

B= base (**B**=2)

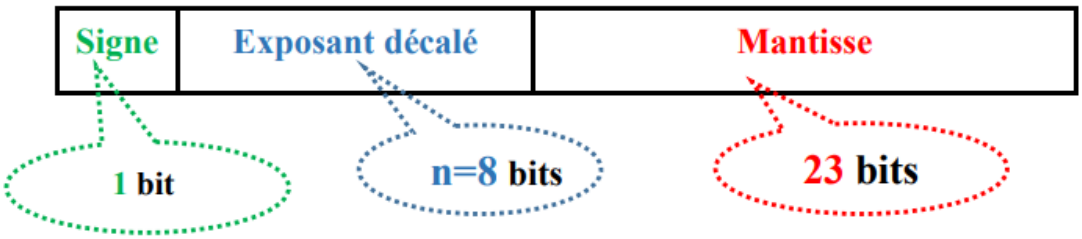
E= exposant réel

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 :

$$\text{Exposant Décalé} = \text{Exposant Réel} + \text{Décalage}$$

$$\text{Décalage} = 2^{n-1} - 1$$

Avec :

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

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

$$\text{Décalage} = 127$$

Note :

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

❖ **Norme IEEE-754 double précision :**

Dans la Représentation double précision on a :

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

Avec :

B= base (**B**=2)

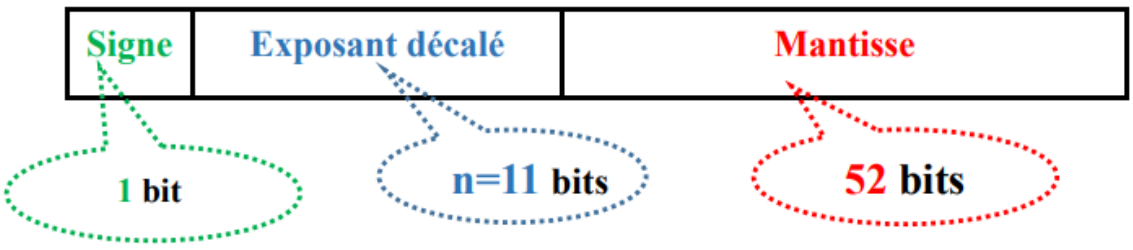
E= exposant réel

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

Note :

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

III.5.4 Comparaison entre Virgule Fixe et Virgule Flottante :

Virgule fixe est une représentation simple et efficace pour des systèmes qui n'ont pas besoin d'une large plage de valeurs, mais elle présente des limitations en termes de flexibilité.

Virgule flottante, en particulier avec la **norme IEEE 754**, est utilisée pour représenter des nombres très petits ou très grands avec une grande flexibilité et précision

Tableau III 7: *Comparaison entre Virgule Fixe et Virgule flottante.*

Caractéristiques	Virgule Fixe	Virgule Flottante (IEEE 754)
Plage de valeurs	Limitée par la taille des bits	Très large, adaptée à de très grandes ou petites valeurs
Précision	Fixe	Variable, dépend de l'exposant et de la mantisse
Utilisation	Systèmes embarqués, DSP	Calculs scientifiques, financiers, simulations numériques
Facilité de calcul	Simple, rapide	Plus complexe, nécessite des algorithmes spécifiques pour l'arithmétique
Représentation du zéro	Un seul zéro	Zéro positif et zéro négatif distincts
Complexité	Faible	Plus élevée

Liste des Exercices :

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 :

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

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

Exercice 4 :

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 5 :

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

Exercice 6 :

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

Exercice 7 :

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

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

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

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

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

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

Exercice 8 :

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

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

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

Exercice 9 :

Effectuer les conversions suivantes :

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

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

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

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

Exercice10: Traduire le message suivant codé en ASCII:

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

Solution :

Exercice 1 :

1) $(-7)_{10} = -(111)_2$ ⟹ 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) $-(107) = -(1101011)_2$ ⟹ 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'' = ''- ''

* $(1011011)_2 = 91$

⇨ $(11011011)_{sva} = (-91)_{10}$

2) Cp1

a) 00110110 MSB =0 ⇨nombre positif

⇨$+(00110110)_2 = (+54)_{10}$

b) 11011011 MSB =1 ⇨ nombre négatif

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

$= (-36)_{10}$

3) Cp2

a) 00110110 MSB =0 ⇨nombre positif

⇨$+(00110110)_2 = (+54)_{10}$

b) 11011011 MSB =1⇨ 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

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

Exercice 4 :

$(-29.774)_{10}$

* signe= - ⟹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 5 :

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

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

* **simple précision**

BS = 0

MN = 1.1111110011

E = +6 ⟹ 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 ⇨ ED=+6+1023=1029
$=(10000000101)_2$

0	10000000101	1111110011000……0

1BS 11bits ED 52bits MN

Exercice 6 :

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 :

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

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

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

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

Exercice 7 :

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

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

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

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

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

Exercice 8 :

En BCD :

a)

```
   67          0110 0111
 + 34      +   0011 0100
 ----        -----------
  101          1001 1011
                    0110
               ----------
                1010 0001
                0110
               ----------
               (0001 0000 0001)BCD
                 1    0    1
```

b)

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

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

```
  12          0001 0010
+ 15        + 0001 0101
----        -----------
  27          0010 0111
                2    7
```

En XS-3

a)

$$\begin{array}{rrrr} 67 & & (1001 & 1010)_{xs\text{-}3} \\ +\ 34 & & (0110 & 0111)_{xs\text{-}3} \\ \hline 101 & 0001 & 0000 & 0001 \\ & +\ 11 & +\ 0011 & +\ 0011 \\ \hline & (0100 & 0011 & 0100)_{xs\text{-}3} \\ & (1 & 0 & 1)_{10} \end{array}$$

b)

$$\begin{array}{rrrr} 12 & 0100 & 0101 & \\ +\ 15 & +\ 0100 & 1000 & \\ \hline 27 & 1000 & 1101 & \\ & -\ 0011 & -\ 0011 & \\ \hline & (0101 & 1010)_{xs\text{-}3} & \\ & (2 & 7)_{10} & \end{array}$$

Exercice 9 :

Effectuer les conversions suivantes :

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

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

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

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

Exercice10:

(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-IV- L'Algèbre de Boole binaire.

IV.1 Introduction :

L'algèbre de Boole est à la base de la logique binaire et de la conception des systèmes numériques (ordinateurs, téléphones, etc.), c'est un système mathématique qui traite de variables logiques prenant les valeurs **0 (faux)** ou **1 (vrai).** Elle a été développée par **George Boole** au XIXe siècle.

Applications :

- **Conception de circuits électroniques** : Les portes logiques **(ET, OU, NON)** suivent les règles de l'algèbre de Boole.
- **Optimisation des systèmes logiques** : Simplification des expressions logiques pour des circuits plus efficaces.
- **Programmation informatique** : Utilisée dans les conditions et les prises de décision.

IV.2 Objectifs :

- Acquérir une bonne maîtrise des concepts fondamentaux, comme les variables **binaires (0 et 1)** et les opérations logiques **(ET, OU, NON).**
- **Apprendre les axiomes et théorèmes de l'algèbre de Boole :** Maîtriser les lois et propriétés essentielles comme l'**associativité**, la **commutativité**, la **distributivité** et les **lois de Morgan**.
- **Manipuler les opérateurs logiques :** Apprendre à utiliser les opérateurs logiques de base **(ET, OU, NON)** et les opérateurs composés **(NAND, NOR, XOR, implication)**.
- **Simplifier des expressions logiques :** Acquérir des techniques de simplification des fonctions logiques, que ce soit par la méthode algébrique, les tableaux de **Karnaugh**, ou la méthode de **Quine-McCluskey**
- **Concevoir des circuits à partir de fonctions logiques :** Savoir exprimer des fonctions logiques en utilisant exclusivement des circuits **NAND** ou **NOR**.

IV.3 Définition et axiomes de l'algèbre de Boole (Notion de base) :

IV.3.1 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. La variable **binaire** est aussi appelée variable **booléenne**. Elle est utilisée pour décrire le comportement des circuits logiques.

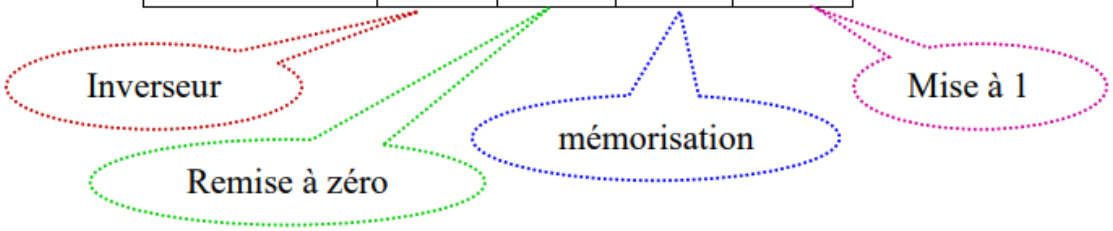
IV.3.2 Fonction logique :

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. **[4]**

Du fait qu'une variable logique ne peut prendre que 2 valeurs (0 ou 1), le nombre de fonctions s'en trouve limité. 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



Les axiomes de l'algèbre de Boole sont les lois fondamentales qui définissent les opérations sur ces variables :

Identité : A + 0 = A et A · 1 = A.

Annulation : A + 1 = 1 et A · 0 = 0.

Idempotence : A + A = A et A · A = A.

Complémentation : A + A' = 1 et A · A' = 0.

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

IV.4 Théorèmes et propriétés de l'algèbre de Boole :

Soient A, B et C, trois variables logiques. Les principaux théorèmes de l'algèbre de Boole incluent :

Loi de commutativité :

L'ordre est sans importance

A + B = B + A

A · B = B · A

Loi d'associativité :

(A + B) + C = A + (B + C)

(A · B) · C = A · (B · C)

Loi de distributivité :

$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

$A + (B \cdot C) = (A + B) \cdot (A + C)$

Théorème de Morgan :

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

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

Le produit logique complimenté de deux variables est égal à la somme logique des compléments des deux variables.

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

Example:

Déterminer le complément de la fonction suivante:

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

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

Note:

Pour chacune de ces combinaisons, la fonction peut prendre une valeur 0 ou 1.

L'ensemble de ces 2^n combinaisons et la valeur de la fonction correspondante représente la table de vérité.

Ces théorèmes facilitent la simplification des expressions logiques.

IV.5 Table de vérité :

Une table de vérité est un tableau qui montre toutes les combinaisons possibles des valeurs logiques des variables d'une fonction et leur résultat.

Pour une fonction à **n variables**, une table de vérité contient **2^N lignes**, **N= nombre** d'entrées (**variables**).

IV.6 Les opérateurs logiques de base :

- Il existe 3 operateurs logiques élémentaires en algèbre de Boole :

- **OU** logique "**addition**" : l'opérateur d'union ou addition logique, **A + B** est vrai si **A** ou **B** est vrai.

- **ET** logique "**multiplication**": l'opérateur d'intersection ou multiplication logique, **A** · **B** est vrai si **A** et **B** sont vrais.
- **NON** logique " **négation**" : l'opérateur qui inverse la valeur logique d'une variable, **-A** est vrai si A est faux

- **Représentation schématique :**

IV.6.1 "OU" logique :

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

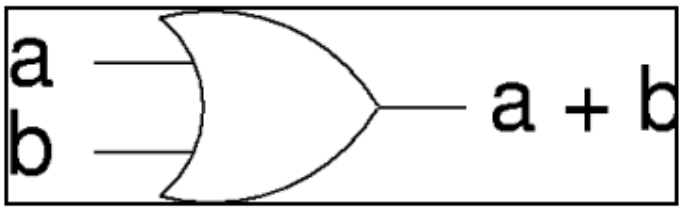


***Figure IV. 1**: Représentation schématique OR.*

***Tableau IV. 1**: Table de vérité OR.*

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

IV.6.2"ET" logique :

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

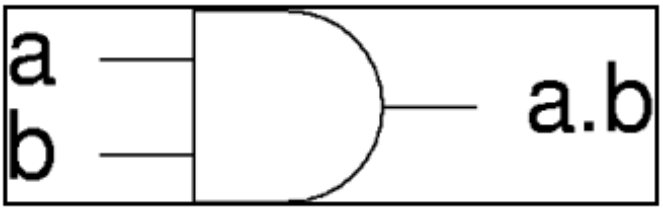


***Figure IV. 2**: Représentation schématique AND.*

***Tableau IV. 2**: Table de vérité AND.*

A	B	A. B
0	0	0
0	1	0
1	0	0
1	1	1

IV.6.3"NON" logique :

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

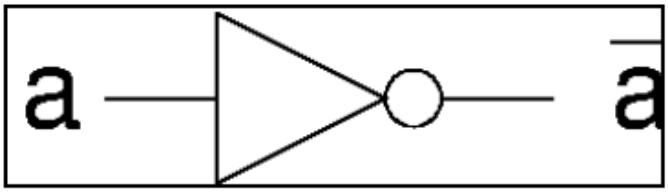


***Figure IV. 3**: Représentation schématique NOT.*

***Tableau IV. 3**: Table de vérité NOT.*

A	$\overline{A}$
0	1
1	0

Note :

Pour les opérateurs **(ET, OU)** nous avons juste donné la définition de base avec deux variables logiques, ils peuvent être réalisé avec plusieurs variables logiques **(A+B+C+D…).** **[7]**

IV.7Autres opérateurs logiques :

Il existe d'autres opérateurs logiques en **algèbre de Boole :**

IV.7.1 NON-OU(NOR) :

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

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

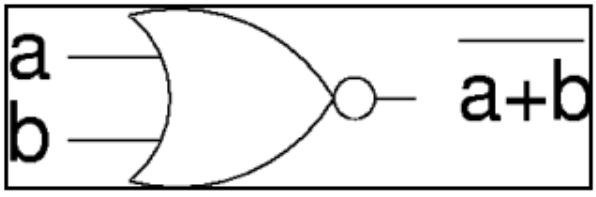


***Figure IV. 4**: Représentation schématique NOR.*

Tableau IV. 4: *Table de vérité NOR.*

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

IV.7.2 NON-ET (NAND) :

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

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

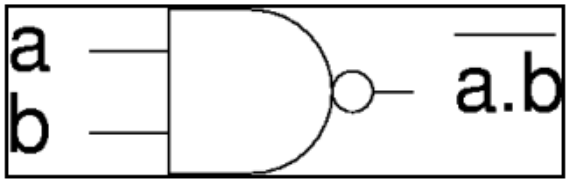


Figure IV. 5*: Représentation schématique NAND.*

Tableau IV. 5*: Table de vérité NAND.*

A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

IV.7.3 OU-Exclusif (XOR) :

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

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

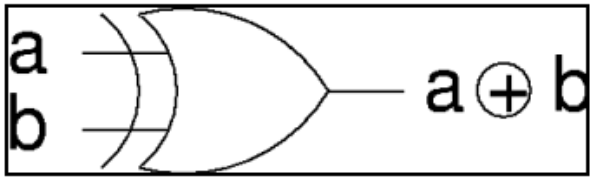


Figure IV. *6: Représentation schématique XOR.*

***Tableau IV. 6**: Table de vérité XOR.*

A	B	A⊕B
0	0	1
0	1	0
1	0	0
1	1	0

IV.8 Forme canonique :

On a deux expressions logiques à partir table de vérité :

IV.8.1 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.

IV.8.2 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. **[5]**

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

IV.9 Simplification d'une fonction logique :

IV.9.1 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.

Exemple :

Simplifier f

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

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

IV.9.2 Tableaux de Karnaugh :

Un tableau de Karnaugh est un outil graphique qui permet de simplifier une fonction logique en regroupant des termes adjacents dans une table représentant toutes les combinaisons possibles de variables.

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

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

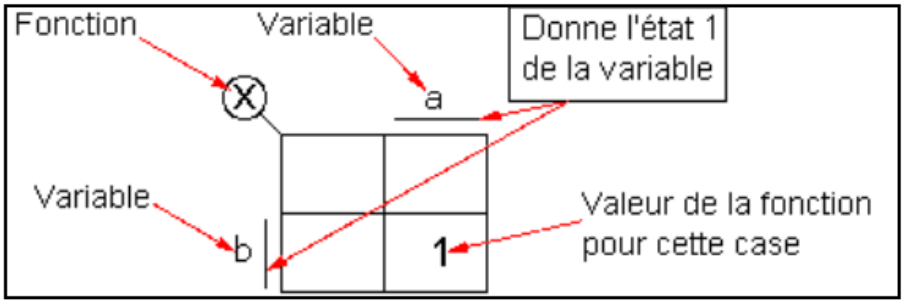


Figure IV. 7*: Table de KARNAUGH.*

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

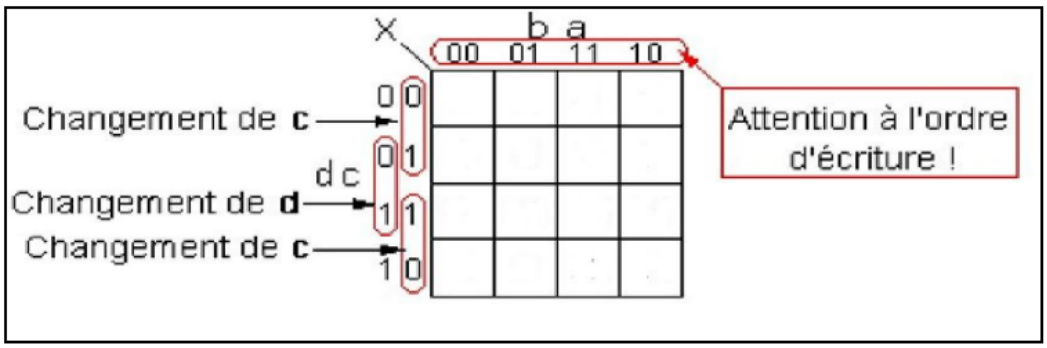


Figure IV. 8*: Variable d'entrées de la Table de KARNAUGH.*

IV.9.3 Simplification d'une fonction logique par méthode 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 $\mathbf{2^k}$ **= (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).

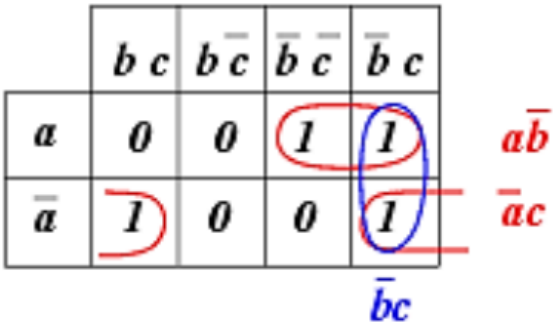
Exemple 1 :

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 :

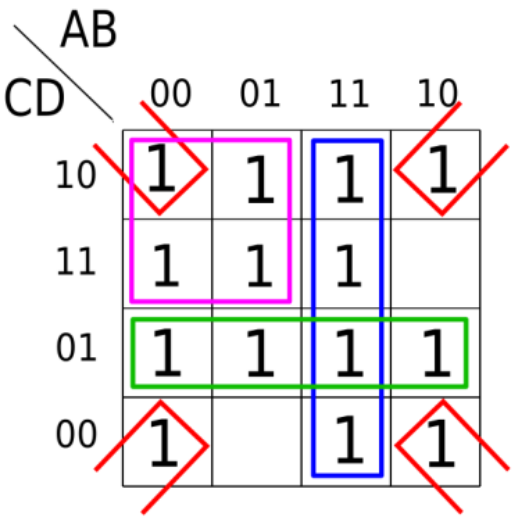
a	b	c		f(a,b,c)
0	0	0		**0**
0	0	1		**0**
0	1	0		**1**
0	1	1		**1**
1	0	0		**1**
1	0	1		**0**
1	1	0		**1**
1	1	1		**0**



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

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**



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

IV.9.4 Méthodes de Quine-McCluskey :

La méthode de **Quine-McCluskey** est une technique systématique pour la simplification des expressions logiques (ou des fonctions booléennes). Elle est similaire aux tableaux de **KARNAUGH**, mais elle est mieux adaptée à la simplification des fonctions comportant un grand nombre de variables.

- **Étapes de la méthode de Quine-McCluskey :**

1. **Représentation sous forme canonique** :

   Commencez par exprimer la fonction booléenne sous forme **disjonctive normale (SOP)**, c'est-à-dire une **somme de produits**. Utilisez pour cela les **mintermes** de la fonction, qui sont les combinaisons d'entrées où la fonction prend la valeur 1.

2. **Regroupement des mintermes** :

   Listez tous les mintermes dans une table et organisez-les en fonction du **nombre de 1** dans leur représentation binaire. Chaque minterme est écrit sous forme binaire (par exemple, pour trois variables, les mintermes sont représentés par des combinaisons de trois bits, comme 001, 010, etc.).

3. **Combinaison des mintermes** :

   Recherchez les mintermes qui ne diffèrent que par **un seul bit**. Si deux mintermes peuvent être combinés, remplacez le bit qui diffère par un **tiret (-)**, indiquant une indétermination (ou une simplification possible). Par exemple, **011** et **111** peuvent être combinés en **-11**.

   Répétez ce processus en combinant autant de mintermes que possible.

4. **Table des premiers implicants** :

   Les termes qui ne peuvent plus être combinés sont appelés **premiers implicants** (ou "implicants essentiels" lorsqu'ils couvrent des mintermes uniques). Notez ces premiers implicants dans une table.

5. **Table de couverture des mintermes** :

   Créez une table pour montrer quelles combinaisons de mintermes sont couvertes par chaque premier impliquant. Identifiez les **implicants essentiels**, c'est-à-dire ceux qui couvrent des mintermes que les autres implicants ne couvrent pas. Ces implicants doivent être inclus dans la solution.

6. **Simplification finale** :

   Utilisez les implicants essentiels, ainsi que des implicants supplémentaires si nécessaire, pour couvrir tous les mintermes de la fonction d'origine. La solution finale est une version simplifiée de la fonction booléenne. **[5]**

Exemple :

Prenons une fonction à trois variables **F (A, B, C)** définie par les **mintermes 1, 3, 7, 5** :

Minterme	Représentation binaire
1	**001**
3	**011**
7	**111**
5	**101**

Étape 1 : Organisez les mintermes par nombre de 1 :

- **1 bit à 1** : **001**(minterme 1)
- **2 bits à 1** : **011**(minterme 3), 101 (minterme 5)
- **3 bits à 1** : **111**(minterme 7)

Étape 2 : Combinaison des mintermes :

- **001**et **011**peuvent être combinés en **0-1**.
- **011**et **111**peuvent être combinés en **-11**.
- **101**et **111**peuvent être combinés en **1-1**.

Étape 3 : Les premiers implicants sont : 0-1, -11, 1-1.

Étape 4 : Créez la table des premiers implicants et identifiez les implicants essentiels qui couvrent les mintermes restants.

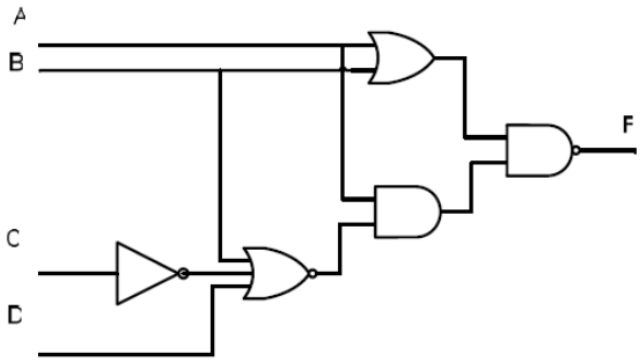
IV.9 Schéma d'un circuit logique (Logigramme) :

Schéma d'un circuit logique ou logigramme est la représentation graphique (schéma électronique) d'une fonction logique. Le principe consiste à remplacer chaque opérateur logique d'une fonction par son symbole logique. **[4]**

Exemple :

Donner le logigramme de F

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



Liste des Exercices :

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

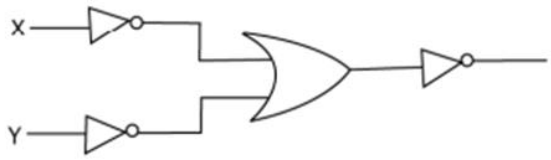
Pour chacune d'elles :

1 - Construire la table de KARNAUGH ;

2- Utiliser le diagramme pour simplifier les expressions.

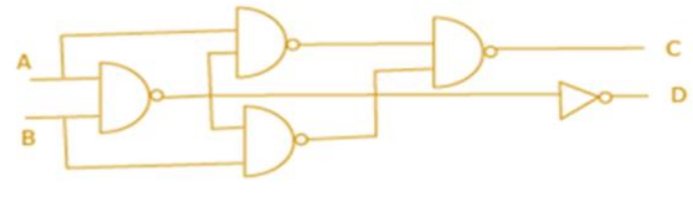
Exercice 6 :

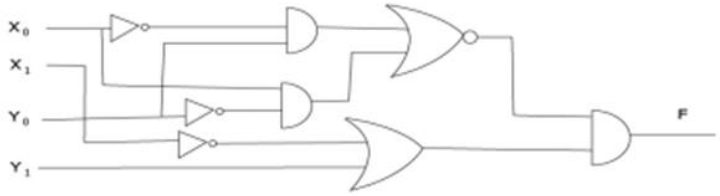
Quelle est la fonction logique réalisée par le circuit de la figure suivante :

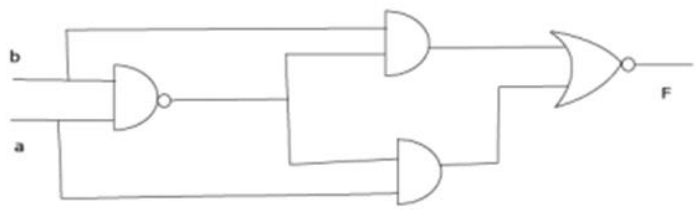


Exercice 7 :

Analyser les circuits suivants et précisez leur fonction







Solution :

Exercice 1 :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 :

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

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

$F_1(A,B,C) = AB + AC$

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

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

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

Exercice 6 :

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

Exercice 7:

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

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

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

$\equiv B \oplus A$

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

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

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

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

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

Bibliographie :

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

[2]. M. BEN AMARA & **K**. **GAALOUL**, Cours de systèmes logiques 1. ISET de Nabeul. 2019.

[3]. D. Rebaïne, La Présentation des données. Professeur, Département d'informatique et de mathématique Université du Québec, 2012.

[4]. S. GHERDAOUI, Structure Machine 1. Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF. 2020/2021.

[5]. M. KHALFI, Cours Structure Machine. Université Djilali Liabès De Sidi Bel-Abbès. 2015 – 2016.

[6]. M. LARBI, Structure Machine 1. Faculté des Sciences Exactes et Informatique (FSEI). 2022.2023.

[7]. M. MEFTAH, Codage et Représentation de l'Information. Université d'El-Oued Faculté de Sciences exactes.2018.

[8]. Z. TAHA, Structure Machine cours et exercices. Université de Bouira Faculty of Science PhD in computer science. 2021.

[9]. M. ISET, Architecture Des Ordinateurs Technologies et concepts. Institut Supérieur des Etudes Technologiques de Mahdia, 2010.