Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Normale Supérieure d'Oran (ENSO)

Département des Sciences Exactes

Filière d'Informatique

Architecture des Ordinateurs I

Cours et Exercices

Préparé par : Dr. Belayachi Naima

Table des matières

Table des figures

Liste des tableaux

Introduction générale

L'architecture des ordinateurs ne se limite pas à l'étude de l'organisation des différents éléments qui rentrent dans leurs compositions. Elle s'intéresse aussi aux fonctions et liaisons qui doivent être mises en place afin d'aboutir à des meilleures performances de la machine. Autrement dit, l'architecture des ordinateurs désigne la disposition des organes d'un système et les relations entre ces organes.

Ce polycopié constitue un manuel de cours d'Architecture des ordinateurs I et quelque exercices pour chaque chapitre. Il explique d'une façon simple et facile la structure et le fonctionnement de l'ordinateur en commençant par des notions et mécanismes de base.

Objectif du cours

L'objectif de ce support pédagogique est de permettre aux étudiants de la première année Professeur d'Enseignement Secondaire (PES) en Informatique d'acquérir certaines notions fondamentales en architecture des ordinateurs I, et cela pour une meilleure maitrise des notions et concepts fondamentaux appliqués dans le domaine d'informatique.

Ce module annuel qui est destiné aux étudiant de la 1^{ere} année PES Informatique, de coefficient 4 avec un volume horaire hebdomadière de 03 heures à raison d'une séance de cours et une séance de travaux dirigés par semaine, vise en premier lieu l'introduction des concepts de base liés à description de la machine de Von Neumann. Par la suite, les systèmes de numérotation et le codage des informations sont présentés, ainsi que la logique combinatoire et séquentielle. Finalement, les différents composants d'un ordinateur sont abordés suivis par la description des modes d'adressage qui sont nécessaires pour expliquer aux futurs enseignants de spécialiste Informatique le déroulement des programmes et le fonctionnement de l'ordinateur.

Organisation du polycopié

Le présent polycopié réalisé conformément au canevas destiné aux étudiants de la première année PES Informatique, est réparti en cinq chapitres :

- Le premier chapitre présente le principe et l'architecture de la machine de Von Neumann ainsi que son organisation logique.
- Le deuxième chapitre comporte les systèmes de numération. Il aborde les bases de numération ainsi que les conversions possibles des bases et les opérations arithmétiques sur la base binaire.
- Le troisième chapitre est consacré à la représentation et codage des informations.
- Le quatrième chapitre illustre les bases fondamentales sur la logique combinatoire et séquentielle.
- Le cinquième chapitre est réservé pour exposer et faciliter la compréhension de fonctionnement des éléments de l'ordinateur ainsi que les modes d'adressages utilisés.
- Quelques références bibliographiques sont données à la fin de ce manuscrit, grâce auxquelles nous avons pu élaborer le présent support.

Remerciements

Il est possible que ce support comporte quelques imperfections, je serais reconnaissante à tous ceux qui me ferait part de leurs remarques et suggestions.

Finalement, je tiens à exprimer mes remerciements aux professeurs qui ont bien voulu le juger et m'aider à l'améliorer.

Chapitre 1

ARCHITECTURE DE VON NEUMANN

Ce chapitre se concentre sur l'architecture de Von Neumann qui représente une architecture de base pour tous les ordinateurs d'aujourd'hui qui sont de plus en plus puissants et rapides. Il est important de noter que la structure des machines récentes reste en règle générale conforme à celle de la machine de Von Neumann. Le principe ainsi que l'organisation logique d'une telle architecture sont présentés dans ce chapitre.

Plan

1.1 Principe et Architecture

Dans les premiers ordinateurs, les différentes instructions et programmes nécessaires à l'exécution d'une tâche, étaient directement câblés dans l'unité de contrôle.

En 1945, le mathématicien Hongrois John Von Neumann a proposé de ranger les programmes (instructions) en mémoire avec les données en utilisant le système binaire basé sur deux valeurs (0 et 1) .

L'architecture dite architecture de von Neumann est un modèle pour un ordinateur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul. La séparation entre le stockage et le processeur est implicite dans ce modèle.

La machine de Von Neumann était composée des éléments suivants :

1. La Mémoire Centrale (MC).
2. L'unité centrale de traitement qui comprend l' :
   (a) Unité Arithmétique et Logique (UAL).
   (b) Unité de Contrôle et de Commande (UCC).
3. Les unités des Entrée / Sortie (E/S).
4. Les bus (unités de transfert).

Avant d'aller plus loin dans ce cours, il a été décidé de présenter d'abord les notions de programmes, instructions, et données.

1.2 Notions de programmes, instructions, données

Un programme est un ensemble d'instructions exécutées par le processeur dans un ordre bien déterminé. Il est généralement écrit dans un langage évolué (Pascal, C, Java, etc.).

Les instructions qui constituent un programme peuvent être classifiées en 4 catégories :

— Les instructions d'affectations (permettent de faire le transfert des données).
— Les instructions arithmétiques et logiques.
— Les instructions de branchement (conditionnelle et inconditionnelle)
— Les instructions des Entrées / Sorties.

Pour exécuter un programme par une machine, on passe par les étapes suivantes :

— Édition : on utilise généralement un éditeur de texte pour écrire un programme et le sauvegarder dans un fichier.
— Compilation : un compilateur est un programme qui convertit le code source (programme écrit dans un langage donné) en un programme écrit dans un langage machine (Binaire). Une instruction en langage évolué peut être traduite en plusieurs instructions machine.

— Chargement : charger le programme en langage machine dans la mémoire centrale afin de l'exécuter par le processeur.

Autrement dit, un programme est un ensemble d'actions (ou d'instructions) séquentielles et logiquement ordonnées, permettant de transformer des données en entrée (Inputs) en données de sorties (outputs ou les résultats), afin de résoudre un problème donné.

Les données manipulées par un programme sont soit des constantes ou des variables et peuvent être de différents types :

— Entier : représenté par l'ensemble ..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...

— Réel : représente les valeurs numériques fractionnels (avec des virgules : fixe ou flottante)

— Caractère : représente tous les caractères imprimables.

— Chaîne de caractères : une séquence d'un ou plusieurs caractères.

— Booléen (logique) : représente les deux valeurs TRUE (vrai) et FALSE (faux).

Aussi, les instructions d'un programme représente un type de données.

Pour comprendre le mécanisme d'exécution d'un programme, il faut comprendre le mécanisme de l'exécution d'une instruction. Ceci revient à étudier l'architecture de la machine sur la quelle va s'exécuter cette instruction.

1.3 Organisation logique d'une machine de Von Neumann

La conception d'une machine est une tâche complexe, qui nécessite une démarche méthodique. L'architecture de la machine proposée par Von Neumann est schématisée sur la Figure (1.1) :

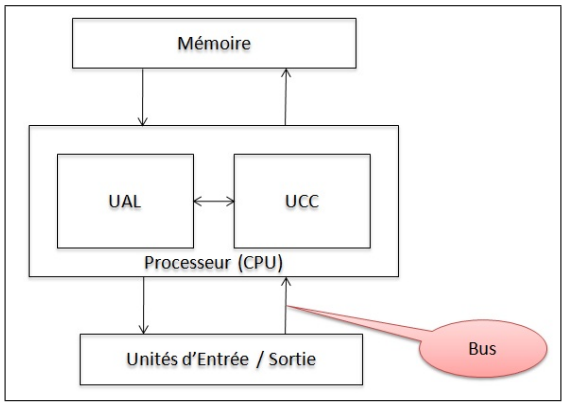


FIGURE 1.1 – Architecture de Von Neumann

1.3.1 Unité Centrale de Traitement

L'unité centrale (UC) appelée aussi processeur, microprocesseur, ou CPU (Central Processing Unit) à pour rôle d'exécuter les programmes et produire des résultats par la suite.

L'UC est composée d'une Unité Arithmétique et Logique (UAL) et d'une Unité de Contrôle et de Commande (UCC).

— L'unité arithmétique et logique (UAL) réalise les opérations élémentaires (addition, soustraction, multiplication, . . .). Elle regroupe l'ensemble des composants requis pour l'exécution des diverses instructions élémentaires et les liens entre ces composants.

— L'unité de contrôle et commande (UCC) gère les opérations sur la mémoire (lecture/écriture) et les opérations à réaliser par l'UAL selon l'instruction en cours d'exécution. Elle est responsable du séquençage des opérations à exécuter par l'UAL selon des entrées externes et les résultats des opérations. L'UCC dirige le fonctionnement de toutes les autres unités de l'ordinateur (UAL, MC, E/S)

Aussi, le processeur possède ses propres unités de stockage d'information, plus rapides que la mémoire, mais avec une capacité de stockage limitée. C'est un bloc de registres pour conserver les données à traiter et des résultats précédents de façon à pouvoir combiner toutes ces valeurs dans de nouveaux calculs par la suite. On trouve :

— Un registre Accumulateur.

— Un registre d'index.

— Un registre de base.

— Des registres généraux pour les calculs intermédiaires (R1, R2, R3, ...).

— Des registres de contrôle ou d'état gérés par la partie contrôle (comme le Registre d'Instruction RI, le Compteur Ordinal CO, le Stack Pointer SP , le Program Status Word PSW, . . .).

Un registre est un emplacement de mémorisation interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire : il s'agit de la mémoire la plus rapide d'un ordinateur, mais dont le coût de fabrication est le plus élevé car la place dans un microprocesseur est limitée.

La mesure de performance des processeurs n'est pas simple. Car la performance d'une machine peut s'exprimer de différentes manières.

Pour l'utilisateur final, le temps d'exécution d'un programme, de son début à sa fin, comprend le temps UC, les accès mémoires, les E/S et le temps utilisé pour les besoins du système d'exploitation. Par conséquent, cela dépend des performances de l'ensemble des composantes matérielles et logicielles du système.

Le temps d'exécution d'un programme sur une unité centrale exprimé en *Mips* (Millions d'instructions par seconde) est donné comme suit :

$$\text{TE} = \text{NI} \,.\, \text{CPI} \,.\, \text{Tc}$$

avec

TE : Temps d'exécution.

NI : Nombre d'instruction du programme. Il dépend du jeu d'instruction et la technologies des compilateurs.

CPI : Nombre de cycle horloge moyen par instruction. Il dépend du format de l'instruction et de sa complexité.

Tc : Temps de cycle (la période), qui dépend de la technologie du matériel. (1 / Tc) représente la fréquence du processeur qui exprimée en Hertz pour mesurer le nombre d'opérations que fait le processeur en une seconde.

1.3.2 Mémoire Centrale

La mémoire centrale (MC) représente l'espace de travail de l'ordinateur. C'est l'organe principal de rangement des informations utilisées par le processeur. Car pour exécuter un programme, il faut le charger (copier) dans la mémoire centrale de l'ordinateur.

Le temps d'accès à la mémoire centrale et sa capacité (la quantité d'informations qu'elle peut stoker qui est exprimée en bits ou en octet) sont deux éléments qui influencent sur le temps d'exécution d'un programme (performances d'une machine).

1.3.3 Sous-système des Entrées / Sorties

Ce sont les accessoires qui sont reliés à la machine pour faire entrer et recevoir les informations à traiter et diffuser les résultats une fois le traitement est terminé.

Principalement, on cite les unités d'entrée comme le clavier, la souris, le scanner et les unités de sortie comme l'écran et l'imprimante.

On trouve aussi des unités qui sont dites d'entrée et de sortie en même temps comme les périphériques de stockage par exemple.

1.3.4 Bus

Les bus représentent les composants qui permettent de véhiculer les informations (données, adresses, commandes) entre les différents éléments de l'ordinateur. Certaines architectures représentent un bus unique pour les trois types d'information, mais dans d'autres machines, on trouve des bus pour chacune des d'informations citées précédemment.

Bus de données

Il permet la circulation des données, y compris les instructions d'un programme entre un processeur et la mémoire.

Bus d'adresses

Il permet de véhiculer l'adresse qui désigne à chaque instant une information (opérande ou instruction) dans un espace mémoire.

Bus de commande

Il assure la transmission des signaux de commandes émis par l'unité de contrôle et de commande vers les autres unités de la machine.

1.4 Conclusion

L'architecture proposée par Von Neumann est un modèle pour un ordinateur de base qui utilise une unité de sauvegarde pour conserver les instructions et les données requises ou produites par l'unité de calcul.

Ce chapitre a permis de présenter l'architecture d'une machine dite de Von Neumann, ainsi que les notions de programmes, instructions, et données. L'organisation de cette machine a été aussi détaillée dans le présent chapitre.

Pour une meilleure compréhension du principe de l'architecture de la machine de Von Neumann, une série d'exercices est proposée à la fin de ce chapitre.

Le chapitre suivant est consacré à la description des systèmes de numération.

1.5 Exercices

Exercice 01

Dans une architecture de Von Neumann :

1. Où sont les données ?
2. Où sont les programmes ?
3. Quelle est le rôle de l'unité de commande ? de Bus ? de Mémoire centrale ? et de l'UAL ?.

Exercice 02

Soit une architecture de Von Neumann munie d'un processeur 24 bits (adresse) cadencé à une fréquence de 2.4 GHz.

1. Dessinez cette architecture en précisant l'emplacement des différents bus et des constituants de l'unité centrale.

2.Calculez le temps d'un cycle de ce processeur. La lecture d'un bloc 3 octets de la mémoire vers le processeur se fait en 4 cycles, le traitement des 3 octets par le processeur prend 9 cycles.

3. Soit un autre processeur cadencé à 1.3 GHz qui lit 4 octets en 2 cycles et il les traite en 5 cycles. Combien de temps lui faut-il pour traiter la même quantité de données que le processeur précédent ? Quel est le processeur le plus rapide ?

Exercice 03

Soit une architecture munie d'un processeur cadencé à 2 GHz, une mémoire centrale de 1 Go ayant un temps d'accès de 20 ns, un temps de cycle mémoire de 25 ns et un format de données de 32 bits.

Le processeur exécute un programme dans lequel il lit 8 octets en 6 cycles d'horloge et traite ces 8 octets en 10 cycles d'horloge.

NB : les 6 cycles de lecture représentent un temps supplémentaire pour organiser la donnée au niveau du processeur.

1. Combien de temps faudra-t-il au processeur pour traiter 1 Go de données ?

2. Combien est le temps d'attente (temps pendant lequel le processeur attend l'arrivée des données) ?

3. Quel est le taux d'attente par rapport au temps total du traitement ?

4. Que conclure de ce résultat ?

Chapitre 2

Systèmes de numération

Ce chapitre est dédié à la présentation des systèmes de numération à travers la description des différents bases de numération ainsi que les conversions possibles entre les bases suivie par la description des opérations arithmétiques sur la base binaire.

Plan

2.1 Bases de numération

Les informations traitées par les ordinateurs sont de différentes natures : nombres ; texte ; images ; sons ; vidéo ; instructions (programmes) ; ...

Dans un ordinateur, elles sont toujours représentées sous forme binaire (une suite de 0 et de 1).

Le système de numération décrit la façon avec laquelle les informations sont codées afin qu'elles soient facilement manipulées en machine. Autrement dit, cela revient à établir une correspondance pour passer sans ambiguïté d'une représentation externe d'une information à une autre représentation interne (sous forme binaire) de la même information, suivant un ensemble de règles précises pour pouvoir passer d'une base à une autre.

2.1.1 Base décimale

Le système décimal est le système de numération le plus utilisé (base = 10). Dans ce système, nous disposons de dix chiffres différents de 0 à 9 pour écrire tous les nombres.

Exemple :

Soit un nombre décimal $A = (2348)_{10}$. Ce nombre est la somme de 8 unités, 4 dizaines, 3 centaines et 2 milliers.

$$A = (2 * 1000) + (3 * 100) + (4 * 10) + (8 * 1)$$
$$A = (2 * 10^3) + (3 * 10^2) + (4 * 10^1) + (8 * 10^0)$$

Où 10 représente la base et les puissances de 0 à 3 représentent le rang de chaque chiffre.

Quelque soit la base, le chiffre de droite est celui des unités, et chiffre de gauche est celui qui a le poids le plus élevé.

D'une manière générale, toute base N est composée de N chiffre de 0 à N-1.

2.1.2 Base binaire

Dans le domaine de l'automatisme, de l'électronique et de l'informatique, nous utilisons la base binaire (base =2).

Dans cette base, tous les nombres s'écrivent avec deux chiffres uniquement (0 et 1).

Nous utilisons le binaire car les systèmes technologiques ont souvent deux états stables :

- Le courant passe = 1

- Le courant ne passe pas = 0

Le chiffre binaire qui peut prendre ces deux états est nommé « Bit » (Binary digit), Figure (2.1).

- Avec un bit nous pouvons coder deux états.

- Avec deux bits nous pouvons coder quatre états.

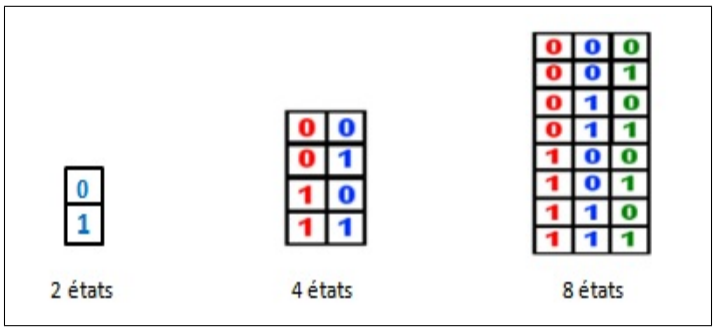


FIGURE 2.1 – Base binaire

- Avec trois bits nous pouvons coder huit états.

Une suite de huit bits est nommée un octet (byte). Avec un octet, nous pouvons écrire $2^8 = 256$ nombres binaires (de 0 à 255).

Un 1 dans une case représente la valeur décimale qui est au dessus, Figure (2.2).

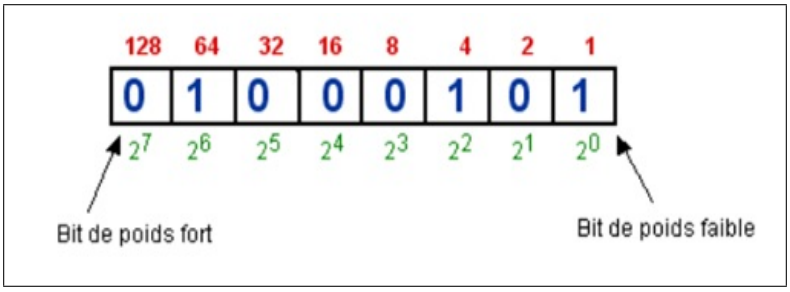


FIGURE 2.2 – Description d'un Octet (Byte)

2.1.3 Base octale

Le système octale (base = 8) utilise huit chiffres : {0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7}, utilisé en informatique il y'a un certain temps. Il permet de coder 3 bits par un seul symbole.

2.1.4 Base hexadécimale

Le système hexadécimal (base = 16) utilise 16 chiffres : $\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A = (10)_{10}; B = (11)_{10}; C = (12)_{10}; D = (13)_{10}; E = (14)_{10}; F = (15)_{10}\}$, il est très utilisé dans le monde de la micro-informatique, et permet de coder 4 bits par un seul symbole.

2.2 Conversion

La conversion consiste le passage d'une base vers une autre en respectant des règles pour garder et maintenir la cohérence des informations codées.

2.2.1 De la base décimale vers la base binaire

Pour passer de la base décimale vers la base binaire, on divise le nombre en base 10 par 2, puis on divise successivement le quotient de chaque division par 2 jusqu'à ne plus pouvoir diviser par 2, Figure (2.3).

Le nombre binaire s'obtient en relevant le reste de chaque division en partant de la dernière division vers la première (sens de lecture vers le haut), et l'écriture se fait de gauche à droite.

Exemple : Convertir le nombre $(230)_{10}$ en base 2.

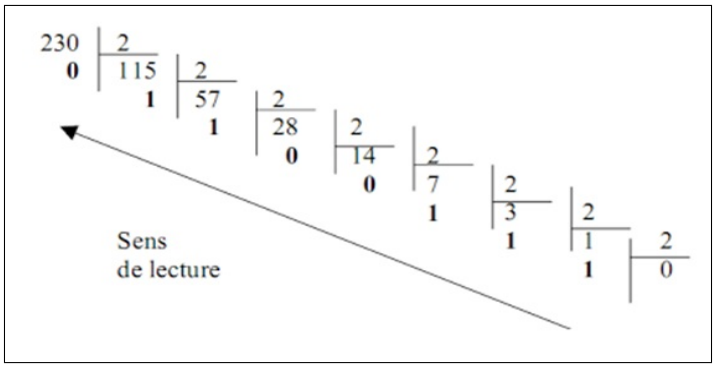


FIGURE 2.3 – Division successive par 2

Le résultat est donc : $(230)_{10} = (11100110)_2$

2.2.2 De la base binaire vers la base décimale

La conversion de la base binaire vers la base 10 se fait par la multiplication de chaque bit par le chiffre 2 élevé à une puissance, croissante par pas de 1, comptée à partir de Zéro en partant de la droite, puis on effectue la somme des résultats obtenus.

Ex : convertir le nombre $(110011)_2$ en décimal :

$(110011)_2 = 1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 0 * 2^3 + 1 * 2^4 + 1 * 2^5$

$= 1+2+0+0+16+32$

$= (51)_{10}$

Le résultat est donc : $(110011)_2 = (51)_{10}$

2.2.3 De la base décimale vers la base hexadécimale

On passe d'un nombre en base 10 à un nombre en base 16 par divisions successives (sur 16). On note les restes des divisions successives, puis on lit ces restes en remontant, et l'écriture se fait de gauche à droite, Figure (2.4).

Exemple : convertir le nombre $(728)_{10}$ en base 16.

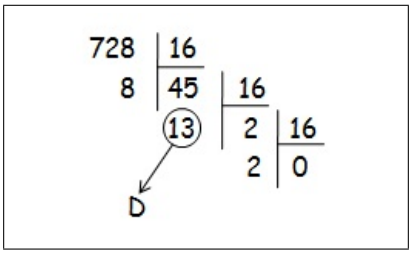


FIGURE 2.4 – Division successive par 16

Le résultat est donc $(728)_{10} = (2D8)_{16}$

2.2.4 De la base décimale vers la base octale

On passe d'un nombre en base 10 à son équivalent en base 8 par des divisions successives (sur 8). On note les restes des divisions successives puis on lit ces restes en remontant, et on écrit le résultat de gauche vers la droite, Figure (2.5).

Exemple : convertir le nombre $(728)_{10}$ en base 8.

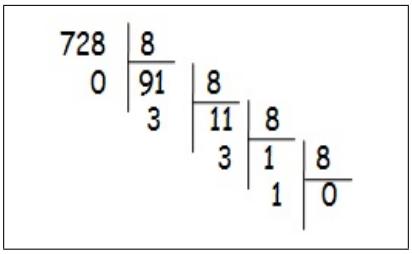


FIGURE 2.5 – Division successive par 8

Le résultat est donc $(728)_{10} = (1330)_8$

2.2.5 De la base octale / hexadécimale vers la base décimale

La conversion à partir du système octal (ou hexadécimal) se réduit à une addition de puissances de 8 (ou de 16) respectivement suivant le rang de chaque chiffre.

Exemple 1 : convertir le nombre $(123)_8$ en base 10.

$(123)_8 = 3 * 8^0 + 2 * 8^1 + 1 * 8^2$

$= 3 + 16 + 64$

$= (83)_{10}$

Exemple 2 : convertir le nombre $(6C5)_{16}$ en base 10.

$(6C5)_{16} = 5 * 16^0 + 12 * 16^1 + 6 * 16^2$

$= 5 + 192 + 1536$

$= (1733)_{10}$.

2.2.6 De la base octale / hexadécimale vers la base binaire

Cette conversion consiste à remplacer chaque chiffre octal (ou hexadécimal) par son équivalent binaire sur 3 (ou 4) bits respectivement.

Exemple 1 : convertir le nombre $(17)_8$ en base 2.

$(17)_8 = (001111)_2$ parce que $(1)_8 = (001)_2$ et $(7)_8 = (111)_2$

Exemple 2 : convertir $(2A)_{16}$ en base 2.

$(2A)_{16} = (00101010)_2$ parce que $(2)_{16} = (0010)_2$ et $(A)_{16} = (1010)_2$

2.3 Opérations arithmétiques sur la base binaire

2.3.1 Addition en binaire

L'addition est réalisée bit à bit, voir Figure (2.6).

$0 + 0 = 0$

$1 + 0 = 1$

$1 + 1 = 10 \rightarrow 0$ avec un report de 1 sur le rang de gauche.

$1 + 1 + 1 = 11 \rightarrow 1$ avec report de 1 sur le rang de gauche.

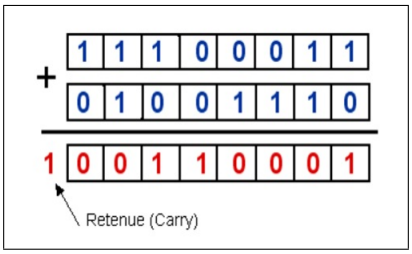


FIGURE 2.6 – Addition en binaire

2.3.2 Multiplication en binaire

On multiplie les nombres binaires de la même façon qu'on multiplie les nombres décimaux, voir Figure (2.7).

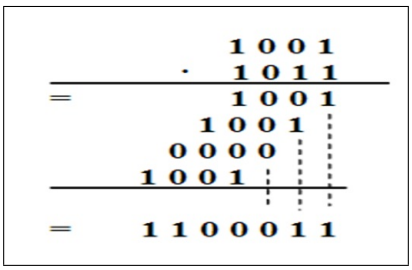


FIGURE 2.7 – Multiplication en binaire

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

2.3.3 Soustraction en binaire

La soustraction d'un nombre binaire (le diminuteur) d'un autre nombre (le diminuande) est semblable à la soustraction décimale.

On emprunte le 1 dans le cas où un bit du diminuteur est supérieur à celui de même rang du diminuande. Cet emprunt sera ajouté au bit du rang suivant du diminuteur, voir Figure (2.8).

0-0 = 0
1-0 = 1
1-1 = 0
0-1 = 1 avec un emprunt de 1

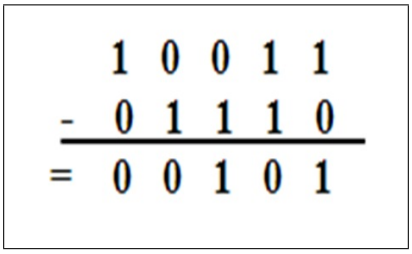


FIGURE 2.8 – Soustraction en binaire

2.3.4 Division en binaire

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux, voir Figure (2.9)

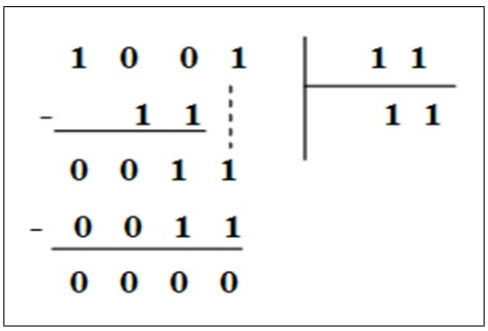


FIGURE 2.9 – Division en binaire

En réalité, la division en binaire est plus simple puisque pour déterminer combien de fois le diviseur entre dans le dividende, il n'y a que 2 possibilités 0 ou 1.

2.4 Conclusion

Les informations manipulées par un ordinateur peuvent être de différentes natures (texte, nombres, etc.), mais elles sont toujours représentées dans l'ordinateur sous forme d'un système numérique binaire.

Chaque information sera traitée comme une suite de 0 et de 1. L'unité indivisible d'information est le chiffre binaire (0 ou 1), que l'on appelle bit (binary digit).

Ce chapitre a été l'occasion de présenter le système de numération utilisé dans un ordinateur. Une série d'exercices est donnée à fin de ce chapitre pour mieux comprendre le cours.

Le codage des informations à l'intérieur de la machine est présenté dans le chapitre suivant.

2.5 Exercices

Exercice 01

Complétez le remplissage de la Table (2.1) en effectuant les conversions nécessaires :

Décimale	**Octale**	**Hexadécimale**	**Binaire**
211	.....	.....	.....
.....	.....	.....	101010101
.....	317	.....	.....
.....	.....	2B	.....
397	.....	.....	.....

TABLE 2.1 – Conversion binaire/octale/ décimale/ hexadécimale

Exercice 02

Convertissez les nombres suivants vers la base 16

a) 3167_{10}

b) 219_{10}

c) 6560_{10}

d) 1001011_2

e) 110_2

Exercice 03

Effectuez les opérations suivantes

1. 10110 * 101 =
2. 10111 + 10010 =
3. 1101 - 110 =
4. 1011 + 110 =
5. 11001 - 0111 =
6. 1011 * 11 =
7. 101100 / 100 =
8. 100100 / 11 =

Exercice 04

Convertissez les nombres suivants vers la base octale

a) 127_{16}.

b) 307_{10}.

c) ABC_{16}.

d) 01010000_2.

Chapitre 3

Représentation et codage des informations

Dans ce chapitre, nous introduisons la notion de codification des informations, en exposant la représentation des différents types d'information, spécialement les nombres entiers, réels, ainsi que les caractères.

Plan

3.1 Représentation des nombres entiers naturels

Un entier naturel est un nombre positif ou nul. Pour représenter un tel nombre, il faut déterminer le nombre des bits à utiliser pour le coder qui dépend de ce nombre que l'on désire codifier.

Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (= un octet) car $2^8 = 256$.

D'une manière générale un codage sur n bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et $2^n - 1$.

Exemples :

$9_{10} = (00001001)_2$

$128_{10} = (10000000)_2$

3.2 Représentation des nombres entiers relatifs

Un entier relatif est un entier pouvant être négatif ou positif.

Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées.

L'astuce consiste à utiliser un codage que l'on appelle complément à deux pour coder un nombre négatif.

Cette représentation permet d'effectuer les opérations arithmétiques usuelles naturellement.

3.2.1 Un entier relatif positif ou nul

Un entier relatif positif ou nul sera représenté en binaire (base 2) comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe.

Il faut donc s'assurer que pour un entier positif ou nul, le bit du poids fort est à zéro (0 correspond à un signe positif, 1 correspond à un signe négatif).

Sur 8 bits (1 octet), l'intervalle de codage est [-127, 127].

Sur 16 bits (2 octets), l'intervalle de codage est [-32767, 32767].

Sur 32 bits (4 octets), l'intervalle de codage est [-2147483647 ; 2147483647].

D'une manière générale le plus grand entier relatif positif codé sur N bits sera $2^{n-1} - 1$.

Exemple :

si on code un entier relatif positif sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).

3.2.2 Un entier relatif négatif (Complément à deux)

Un entier relatif négatif sera représenté grâce au codage en complément à deux.

Principe du complément à deux :

— Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.

— Inverser les bits : les 0 deviennent des 1 et vice versa (le complément à un).

— On ajoute 1 au résultat (les dépassements sont ignorés).

Cette opération correspond au calcul de $2^n - |x|$, où n est la longueur de la représentation et $|x|$ la valeur absolue du nombre à coder.

Exemple :

Pour coder le nombre -19 sur 8 bits, il suffit d' :

— Écrire 19 en binaire : 00010011

— Écrire son complément à 1 : 11101100

— Ajouter 1 au complement à 1 : 11101101

La représentation binaire de -19 sur 8 bits est donc : 11101101.

On remarquera qu'en additionnant un nombre et son complément à deux on obtient 0.

00010011

+ 11101101

=

00000000 (avec une retenue de 1 qui est éliminée).

3.2.3 Représentation en signe et valeur absolue

Le principe est de considérer que le bit du poids fort soit réservé pour coder le signe avec :

— 0 → entier positif.

— 1 → entier négatif.

Le Bit du poids fort est le plus à gauche, les autres bits codent le nombre en valeur absolue.

Il est nécessaire de savoir sur combien de bits on code le nombre pour déterminer quel bit code quoi

Exemple : un codage sur 4 bits :

$(0111)_2 = 7$ car le bit du poids fort est à 0.

$(1111)_2$ = -7 car le bit du poids fort est à 1.

3.3 Représentation des nombres réels

3.3.1 Virgule fixe

Une représentation d'un nombre en virgule fixe correspondant à une représentation possédant un nombre fixe de chiffres après la virgule.

La représentation des nombres réels en virgule fixe est utile pour représenter les parties fractionnaires dans un format utilisant le complément à deux quand le processeur de l'ordinateur ne dispose d'aucune unité de calcul en virgule flottante ou bien quand une virgule fixe permet d'augmenter la vitesse d'exécution ou d'améliorer l'exactitude des calculs.

Codage d'un nombre décimale en virgule fixe

Dans ce codage, la partie entière du nombre peut être traduite par des puissances positives de 2, et la partie fractionnaire va se traduire par des puissances négatives de 2.

Exemple :

$25 = \mathbf{1} * 2^{4} + \mathbf{1} * 2^{3} + \mathbf{0} * 2^{2} + \mathbf{0} * 2^{1} + \mathbf{1} * 2^{0}$.

$0{,}375 = \mathbf{0} * 2^{-1} + \mathbf{1} * 2^{-2} + \mathbf{1} * 2^{-3}$

Par conséquent le nombre 25.375 est traduit par 11001,011 en virgule fixe.

Codage d'un nombre binaire en virgule fixe

Exemple 1 : Mot de 8 bits

Mot de 8 chiffres avec (n,m) = (5,3) ⇒ n+m= 8

N =$(11001,011)_2 = (25.375)_{10}$

N =$(11001011)_{2(5,3)} = (25.375)_{10}$

Exemple 2 : Mot de 3 bits (m + n) = 3

$(010)_{2(3,0)} = 2_{10}$

$(010)_{2(2,1)} = 1_{10}$

$(010)_{2(1,2)} = 0.5_{10}$

$(010)_{2(0,3)} = 0.25_{10}$

3.3.2 Virgule flottante

Le principe est d'avoir une virgule flottante et une précision limitée, et ne coder que des chiffres significatifs

$$N = +/- M * B^e \text{ où}$$

— N = nombre codé

— +/- = codage du signe (positif ou négatif)

— M = mantisse (un nombre de X chiffres de la base B)

— e = exposant (nombre de Y chiffres de la base B)

Il faut arriver à une présentation sous une forme normalisée $1, Mantisse * B^e$ pour déterminer la mantisse et l'exposant

Pas de chiffre avant la virgule $0, XXXXX * B^e$, avec un bit caché (non représenté sur la machine)

Selon la norme standard IEEE 754 : le codage binaire d'un nombre réel en virgule flottante en simple précision (sur 32 bits) est comme suit, Figure (3.1) :

1 bit pour le signe, 8 bits pour l'exposant, et 23 bits pour la mantisse.

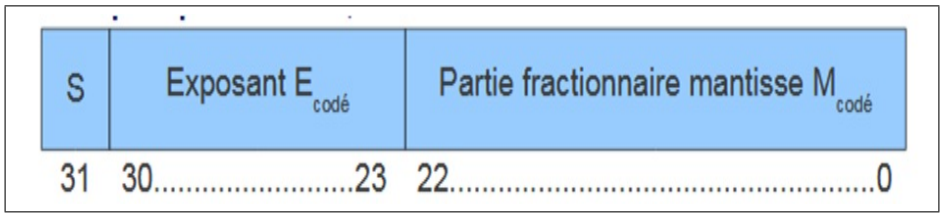


FIGURE 3.1 – Représentation des nombres réels en virgule Flottante IEEE754

Exemple 1 :

$(1011,101101)_2 = (1011,101101)_2 * 2^0$

$= (101,1101101)_2 * 2^1$

$= (10,11101101)_2 * 2^2$

$= (1,011101101)_2 * 2^3$

Exemple 2 :

$(0,00101)_2 = (0,00101)_2 * 2^0$

$= (0,0101)_2 * 2^{-1}$

$= (0,101)_2 * 2^{-2}$

$= (1,01)_2 * 2^{-3}$

Remarque :

Il faut pouvoir stocker indifféremment les exposants positifs ou négatifs. Pour cela, plutôt que d'utiliser un codage en complément à deux pour l'exposant, on applique **un décalage d'exposant** et on stocke la valeur décalée (exposant trouvé + décalage).

La valeur du décalage dépend du nombre n de bits utilisés pour stocker l'exposant :

$$\textbf{décalage} = 2^{n-1} - 1$$

Si l'exposant de la représentation normalisée vaut exp, la valeur stockée (exposant décalé) sera :

$$exp + decalage = exp + 2^{n-1} - 1$$

Exemple 1 :

pour un exposant stocké sur 5 bits :

$(1011,101101)_2 = (1,011101101)_2 2^3$

alors l'exposant stocké (décalé) = 3 + 15 = 18.

Exemple 2 :

pour un exposant stocké sur 5 bits :

$(0,00101)_2 = (1,01)_2 2^{-3}$
alors l'exposant stocké = -3 + 15 = 12.

Conversion d'un réel du décimal en binaire

C'est une conversion d'une manière approchée. Il faudra donc donner la précision voulue.
• Pour la partie entière, on fait comme pour les entiers.
• Pour la partie décimale, on procède comme suit :
— On multiplie la partie décimale par la base B ;
— On note la partie entière obtenue ;
— On recommence avec la partie décimale restante ;
— On s'arrête quand la partie décimale est nulle ou quand la précision souhaitée est atteinte.

La partie décimale est la concaténation des parties entières obtenues dans l'ordre de leur calcul.

Exemple : conversion de 28,8625 en binaire
– Conversion de 28 : $(11100)_2$
– Conversion de 0,8625 comme suit :
0,8625 * 2 = 1,725 = **1** + 0,725
0,725 * 2 = 1,45 = **1** + 0,45
0,45 * 2 = 0,9 = **0** + 0,9
0,9 * 2 = 1,8 = **1** + 0,8 ←
0,8 * 2 = 1,6 = **1** + 0,6
0,6 * 2 = 1,2 = **1** + 0,2
0,2 * 2 = 0,4 = **0** + 0,4
0,4 * 2 = 0,8 = **0** + 0,8 ←
...
28,8625 peut être représenté par $(11100, 11011100...)_2$

Conversion d'un réel du binaire vers le décimal

Exemple :
$110,101_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 0 * 2_{-2} + 1 * 2^{-3}$
= 4 + 2 + 0,5 + 0,125
$= 6,625_{10}$

Synthèse

Pour trouver la représentation en virgule flottante d'un nombre décimal, il faut :
— Convertir le nombre en binaire.

— Déterminer la représentation normalisée du nombre.

— Ajouter le décalage à l'exposant trouvé et convertir ce résultat en binaire.

Pour trouver la valeur décimale d'un nombre codé en virgule flottante, on effectue les étapes précédentes dans le sens inverse.

Exemple :

Donnez la représentation en virgule flottante de $(-0,28125)_{10}$ selon la norme IEEE754.

$(-0,28125)_{10} = (-0,01001)_2 = (-1,001)_2 2^{-2}$

Décalage $= 2^{n-1} - 1 = 2^{8-1} - 1 = 127$

Exposant décalé = 127-2 = $125_{10} = 01111101_2$

La représentation est comme suit, Figure (3.2) :

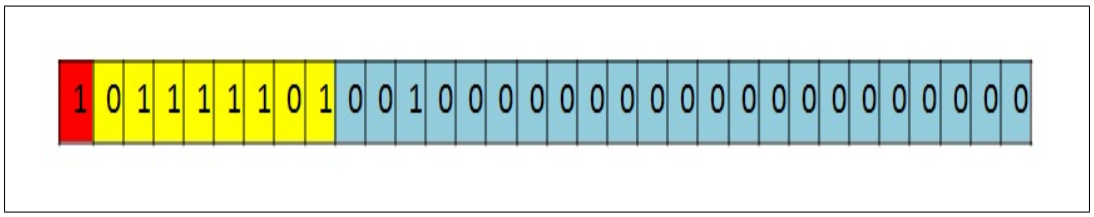


FIGURE 3.2 – Représentation du nombre $(-0,28125)_{10}$ selon la norme IEEE754

3.4 Codage des caractères

Dans un ordinateur, les données sont toujours représentées sous forme binaire (une suite de 0 et de 1).

Cependant, l'être humain ne parle généralement pas couramment le langage binaire. Il doit donc tout "traduire" pour que la machine puisse exécuter les instructions relatives aux programmes installés.

Comment sont codés les textes ?

Un texte est une suite de caractères, nous allons donc plutôt nous poser la question, comment sont stockés les caractères à l'intérieur de la machine ?

La réponse est simple, à chaque caractère on associe un code binaire.

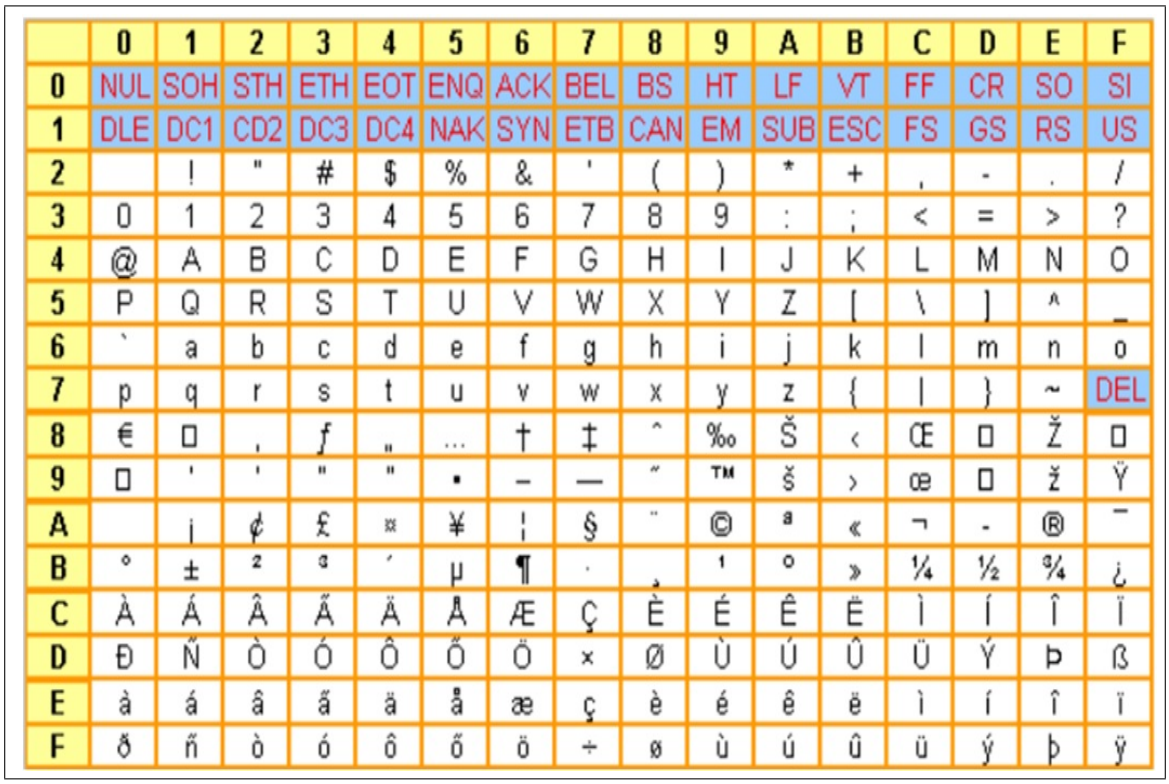
Historiquement, une des premières méthodes de codage des caractères est appelée ASCII (American Standard Code for Information Interchange).

Dans le système de codage ASCII, chaque caractère est codé sur un octet. En réalité sur les 8 bits seuls 7 sont utilisés pour coder les caractères (le 8^{ieme} bit, appelé bit de parité, est utilisé pour détecter les erreurs). $2^7 = 128$ caractères peuvent être codés en ASCII.

Par exemple le caractère 'A' est représenté par le code binaire 1000001, le caractère '4' est représenté par le code binaire 0110100.

Au début de l'histoire de l'informatique cela ne posait pas trop de problèmes, mais avec l'arrivée des outils de bureautiques (traitement de texte,...), cela est devenu problématique : par exemple, pour le français, les caractères accentués ('é', 'à', '@', ...) ne sont pas codés dans le système ASCII.

Pour pallier à cette difficulté, le code ASCII a été étendu à un code 8 bits, Figure (3.3).



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8	€	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	Ž	□
9	□	'	'	"	"	•	–	—	˜	™	š	›	œ	□	ž	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

FIGURE 3.3 – Table ASCII

Afin d'assurer une compatibilité avec l'ASCII, les caractères codés en ASCII ont exactement le même code en UTF-8.

Pour dépasser la limite des 128 caractères de l'ASCII, en UTF-8, certains caractères sont codés sur plus d'un octet.

La norme ASCII établit une correspondance entre une représentation binaire des caractères de l'alphabet latin et les symboles, les signes, qui constituent cet ensemble.

Exemple :

le caractère 'a' est codé en 1100001 (code ASCII = 97).

le caractère 'A' est codé en 1000001 (code ASCII = 65).

La norme ASCII permet ainsi a toutes sortes de machines de stocker, analyser et communiquer de l'information textuelle.

3.5 Conclusion

Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information, et sa représentation interne dans la machine, sous forme d'une suite de bits (0 et 1).

Dans ce chapitre, nous avons abordé la représentation des différents types d'informations, à savoir les entiers naturels, entiers relatifs, réels, caractères.

Pour mieux comprendre le principe de la codification des informations, quelques exercices sont proposés à la fin de chapitre.

Le chapitre qui suit, sera consacré à la présentation générale de l'ordinateur.

3.6 Exercices

Exercice 01

1. Indiquez la valeur codée par la suite 1101100101110101 qui représente un entier signé en complément à 2 sur 16 bits.

2. Même question avec la suite 1001000011101101.

Exercice 02

Codez sur 4 bits les entiers 7, 2, -2, -7 , et -5 avec les représentations suivantes :

1. Signe et valeur absolue ;

2. Complément à 2.

Exercice 03

Codez les entiers 61 et - 61 sur un octet en utilisant la représentation par le signe et la valeur absolue.

1. Montrez que l'addition binaire de ces entiers ainsi codés produit un résultat incorrect.

2. Montrez qu'en revanche le résultat est correct si ces entiers sont codés en utilisant la représentation par le complément à 2.

Exercice 04

Combien de nombres entiers naturels peut-on représenter en binaire sur n bits ?

Exercice 05

En mémoire, on trouve la séquence suivante 01101010 10010101.

Indiquez la valeur de cette mémoire (base 10) dans les cas où cette suite est constituée de :

— deux nombres entiers naturels codés chacun sur 8 bits.

— deux nombres entiers relatifs en représentation signe et valeur absolue sur 8 bits.

— deux nombres entiers relatifs représentés sur 8 bits en complément à 2.

Exercice 06

1. Codez sur 8 bits les parties fractionnaires suivantes : $(0.578125)_{10}$ et $(0.85)_{10}$
2. Décodez les nombres suivants : $(0.10110000)_2$ et $(0.11011001)_2$
3. Qu'est-ce-que vous remarquez ?

Exercice 07

Coder les réels suivants (représentés en base 10) selon la norme IEEE 754 simple précision : - 0.078125, - 13.625, - 87.375.

Exercice 08

Décoder les réels suivants (donnés en simple précision) : $(41FE8000)_{16}$, $(3EA80000)_{16}$, $(C5E00000)_{16}$.

Exercice 09

Voici les codes ASCII, sur un octet chacun, de quelques caractères alphabétiques :

'A' : 01000001 ; 'B' : 01000010 ; 'C' : 01000011

'a' : 01100001 ; 'b' : 01100010 ; 'c' : 01100011

1. Écrivez les valeurs en base 10 correspondant à ces informations binaires.
2. Donnez les codes ASCII de 'D' et de 'd' ? De 'Z' et de 'z' ?

Chapitre 4

Présentation générale de l'ordinateur

Dans le présent chapitre, nous commençons par introduire les principaux organes qui constituent un ordinateur. Ensuite, nous nous attachons à la présentation d'une instruction et les machines à une, deux, et trois adresses. Les modes d'adressage sont aussi présentés dans ce chapitre, pour pouvoir passer au déroulement d'exécution d'une instruction.

Plan

4.1 Les différents organes

Les ordinateurs sont des machines programmables capables d'effectuer un nombre important d'opérations avec précision en un temps très court.

La structure de base d'un ordinateur comprend les éléments fondamentaux qui sont représentés sur la Figure (4.1).

— Une unité centrale de traitement.
— Une unité de mémoire, appelée mémoire centrale.
— Des unités de stockage.
— Des unités d'entrée/sortie.
— Un système de bus permettant de véhiculer l'information entre l'unité centrale et les autres unités.

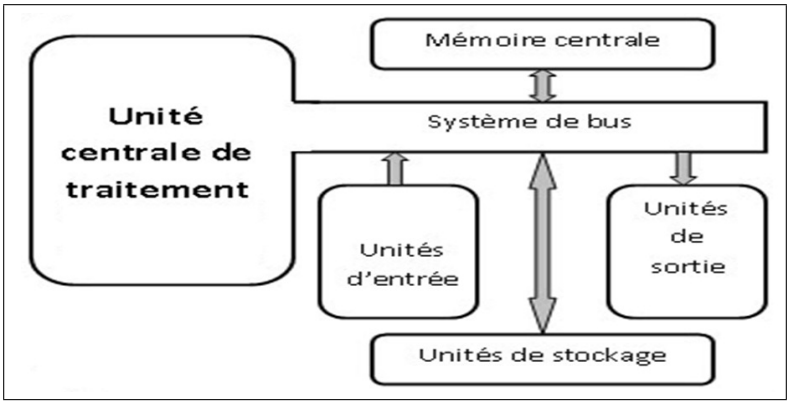


FIGURE 4.1 – Représentation des principaux eléments d'un ordinateur

4.1.1 Unité Centrale de Traitement

L'unité centrale de traitement ou CPU (Central Processing Unit) est le centre de calcul et de contrôle d'un ordinateur. Elle constitue le cerveau de l'ordinateur.

L'unité centrale réalise toutes les opérations demandées, elle est matérialisée physiquement par un circuit électronique appelé microprocesseur , qui est caractérisé par :

— Sa marque (exemple : Intel).
— Sa fréquence d'horloge : le nombre d'opérations que le microprocesseur peut effectuer en seconde. Exemple : 1 GHz = 2^{30} Hz.

L'unité centrale comporte les éléments suivants :

— Unité de contrôle et de commande.
— Unité arithmétique et logique.

Unité Arithmétique et Logique (UAL)

L'UAL est l'organe qui permet d'effectuer des opérations arithmétiques (addition, soustraction, multiplication, division), des opérations logiques (par exemple des comparaisons). La Figure (4.2) permet de représenter l'UAL.

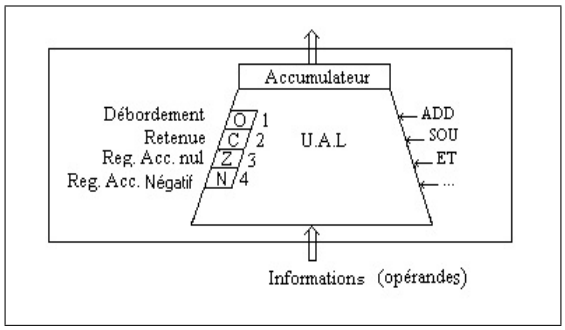


FIGURE 4.2 – Représentation de l'Unité Arithmétique et Logique

En plus des unités fonctionnelles qui effectuent les traitements, l'UAL comporte :

— Un registre accumulateur (ACC) : c'est un registre de travail qui sert a stocker une opérande (données) au début d'une opération et le résultat à la fin.

— Un registre d'état : Ce registre nous indique l'état du déroulement de l'opération . Ce registre est composé d'un ensemble de bits. Ces bits s'appellent indicateurs (drapeaux ou flags), qui sont mis à jours (modifiés) après la fin de l'exécution d'une opération dans l'UAL.

  Les principaux indicateurs sont :

  — Retenue : ce bit est mis à 1 si l'opération génère une retenue.

  — Signe :ce bit est mis à 1 si l'opération génère un résultat négative.

  — Débordement :ce bit est mis à 1 s'il y a un débordement.

  — Zéro : ce bit est mis à 1 si le résultat de l'opération est nul.

Unité de contrôle et de commande

Cette unité gère le déroulement des instructions, décode et donne l'ordre à l'UAL pour exécuter les instructions demandées. Elle comporte :

— Un registre instruction (RI) : qui contient l'instruction en cours d'exécution. Chaque instruction est décodée selon son code opération grâce à un décodeur.

— Un décodeur : pour déterminer quelle opération doit être effectuée.

— Un registre compteur ordinal (CO) ou le compteur de programme (CP) : contient l'adresse de la prochaine instruction à exécuter. Initialement, il contient l'adresse de le première instruction du programme à exécuter.

— Un séquenceur : il organise et synchronise l'exécution des instructions selon le rythme de l'horloge, en générant les signaux nécessaires à l'exécution une instruction.

Remarque :

Le microprocesseur peut contenir d'autres registres autre que CO, RI et ACC, qui sont considérés comme une mémoire interne (registre de travail) du microprocesseur.

Ces registres sont plus rapide que la mémoire centrale, mais le nombre de ces registres et leurs capacités sont limités. Généralement ils sont utilisés pour sauvegarder les données avant d'exécuter une opération

4.1.2 Unité de mémoire (Mémoire Centrale)

La mémoire centrale (MC) est un organe de l'ordinateur permettant d'enregistrer, de stocker et de restituer les informations. La mémoire centrale d'un ordinateur est séparée en deux sections : la mémoire vive et la mémoire morte.

— La mémoire vive (RAM : Random Access Memory) : est une mémoire où on peut lire et écrire à volonté. Cette mémoire est dite volatile, c'est-à-dire qu'elle perd son contenu dés qu'elle est hors tension.

La mémoire vive contient en plus des programmes servant à la gestion de l'ordinateur, le programme relatif à un traitement spécifique ainsi que les données qu'il requiert et les résultats qu'il génère, voir Figure (4.3)

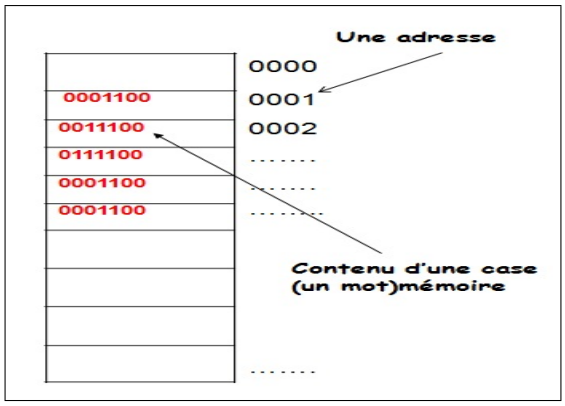


FIGURE 4.3 – Représentation d'une mémoire

La mémoire centrale peut être vu comme un large vecteur (tableau) de mots ou octets. Un mot mémoire stocke une information sur n bits. Chaque mot possède sa propre adresse. Cette mémoire peut contenir des programmes et les données utilisées par les programmes en cours d'exécution.

— La mémoire morte (ROM : Read Only Memory) : est destinée uniquement à être lue. Son contenu n'est pas altéré par une coupure de courant. La mémoire morte,

programmée par le constructeur, est essentiellement utilisée pour contenir les logiciels de base servant au démarrage de l'ordinateur.

4.1.3 Unités de stockage

— Les disques magnétiques :

1. Disque dur : est un support de stockage composé d'un regroupement de disques magnétiques protégé par un boîtier. Le disque dur est le principal outil de stockage de données d'un ordinateur. Il permet de sauvegarder une grande masse de données d'une manière permanente.
2. Disquette : est composée d'un plateau circulaire en matière plastique qui tourne à l'intérieur d'un emballage de protection carré. La capacité d'une disquette est 1.44 Mo soit 1444 octets.

— Les disques optiques compacts : Un disque optique compact (CD-ROM en Anglais) offre une capacité de stockage importante et une très grande sécurité (capacité entre 650 et 700 Mo). L'écriture sur les disques optique se fait par un graveur, la lecture se fait par un lecteur de CD-ROM.

4.1.4 Unités d' Entrée / Sortie

— Les périphériques d'entrée :

1. Le clavier (permet la saisie des informations textuelles et numériques).
2. La souris (est un périphérique de pointage servant à déplacer un curseur sur l'écran, elle permet de transmettre des ordres grâce à ses boutons).
3. Scanner.
4. Microphone.
5. ...

— Les périphériques de sortie :

1. L'écran (permet de visualiser les informations).
2. L'imprimante (permet d'imprimer des documents sur papiers).
3. ...

4.2 Codage d'une instruction

Chaque microprocesseur possède un certain nombre limité d'instructions qu'il peut exécuter. Ces instructions sont appelées jeu d'instructions.

Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur peut exécuter.

Les instructions peuvent être classifiées en 4 catégories : Instruction d'affectation (transfert des données entre les registres et la mémoire avec les opérations de lecture et d'écriture) ; Instructions arithmétiques et logiques (ET, OU, ADD,...) ; Instructions de branchement (conditionnelle et inconditionnelle) ; Instructions d'entrées /sorties.

Les instructions d'un programme en cours d'exécution et leurs opérandes (données) sont stockées dans la mémoire.

La taille d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type de l'instruction et du type de l'opérande manipulée.

L'instruction est découpée en deux parties, voir Figure (4.4) :

1. Code opération (code instruction) : un code sur N bits qui indique quelle opération doit être effectuée.
2. Opérande : qui contient la donnée ou la référence (adresse) à la donnée.

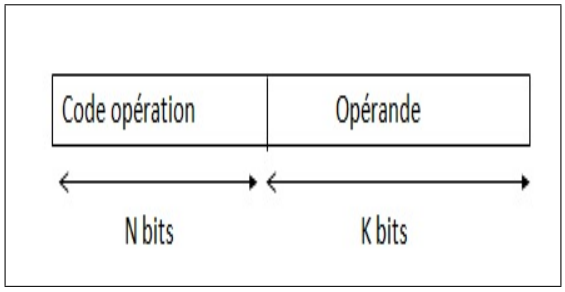


FIGURE 4.4 – Représentation d'une instruction

Remarque :

Le format d'une instruction peut ne pas être le même pour toutes les instructions.

Le champ opérande peut être découpé à sont tours en plusieurs champs.

4.3 La machine à 3 adresses, à 2 adresses, et à 1 adresse

4.3.1 Machine à 3 adresses

Dans ce type de machine, pour chaque instruction il faut préciser :

— L'adresse de la première opérande

— L'adresse de la deuxième opérande

— L'emplacement du résultat

Exemple :

ADD A,B,C (C $\leftarrow$ A+B)

Dans ce type de machine la taille de l'instruction est grande.

4.3.2 Machine à 2 adresses

Dans ce type de machine, pour chaque instruction il faut préciser :
— L'adresse de la première opérande
— L'adresse de la deuxième opérande ,
L'adresse de résultat est implicitement l'adresse de la deuxième opérande .
Exemple :

$$\text{ADD A,B (B} \leftarrow \text{A+B)}$$

4.3.3 Machine à 1 adresse

Dans de type de machine pour chaque instruction il faut préciser uniquement l'adresse de la deuxième opérande. La première opérande existe dans le registre accumulateur (ACC) de l'UAL. Le résultat est mis dans le registre accumulateur. Ce type de machine est le plus utilisé.

Exemple :

$$\text{ADD A (ACC} \leftarrow \text{ACC + A)}$$

4.4 Les modes d'adressage

Le mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande.
Les modes d'adressage les plus utilités sont :
— Adressage Immédiat
— Adressage Direct
— Adressage Indirect
— Adressage Indexé
— Adressage Relatif
— Adressage Basé

4.4.1 Adressage Immédiat

L'opérande existe dans le champs adresse de l'instruction
Exemple :
ADD 150 Cette commande va additionner le Contenu de l'ACC avec la valeur 150.

$$\text{ACC} \leftarrow \text{(ACC)+ 150}$$

Si le registre accumulateur contient la valeur 200, alors après l'exécution de l'instruction précédente son contenu sera égale à 350.

4.4.2 Adressage Direct

Le champs opérande de l'instruction contient l'adresse de l'opérande (emplacement de la donnée dans la mémoire).

Pour réaliser l'opération, il faut récupérer (lire) l'opérande à partir de la mémoire.

Exemple :

ADD 150 sa revient à additionner le contenu de l'ACC avec le contenu de la case mémoire d'adresse 150

$$\text{ACC} \leftarrow (\text{ACC}) + (150)$$

On suppose que l'accumulateur continent la valeur 20 et la case mémoire d'adresse 150 contient la valeur 30.

A la fin de l'exécution, nous allons avoir la valeur 50 dans l'ACC (50 = 20 + 30).

4.4.3 Adressage Indirect

Le champs adresse (opérande) contient l'adresse de l'adresse de l'opérande.

Pour réaliser l'opération il faut récupérer l'adresse de l'opérande à partir de la mémoire. Par la suite, il faut chercher l'opérande à partir de la mémoire, comme illustré sur la Figure (4.5).

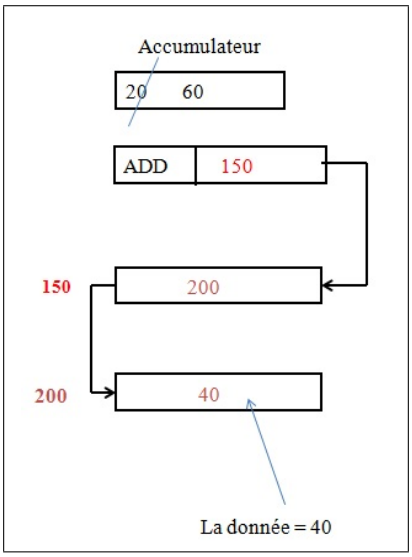


FIGURE 4.5 – Adressage indirect

$$ACC \leftarrow (ACC) + ((Adresse))$$

Exemple :

ADD 150

Supposant que initialement l'accumulateur contient la valeur 20. Pour exécuter l'instruction précédente, il faut :

- Récupérer l'adresse de l'opérande à partir de l'adresse 150 (la valeur 200).
- Récupérer la valeur de l'opérande (la donnée) à partir de l'adresse 200 (la valeur 40).
- Additionner la valeur 40 avec le contenu de l'accumulateur (20) et nous allons avoir la valeur 60, voir Figure (4.5).

4.4.4 Adressage Indexé

L'adresse effectif de l'opérande est relatif à une zone mémoire. L'adresse de cette zone se trouve dans un registre spécial appelé registre d'index, voir Figure (4.6).

Adresse opérande = champs opérande + (registre d'index)

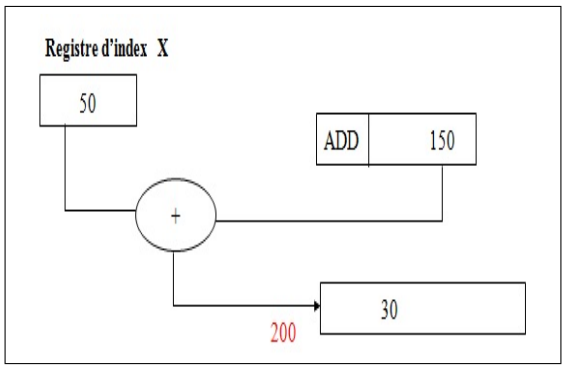


FIGURE 4.6 – Adressage indexé

Exemple :

$$ADD150_{indexepar X}$$

ACC ← (ACC) + (150 + (X))

ACC ← (ACC) + (200)

ACC ← (ACC) + 30

Remarque : Si le champs opérande ne contient pas une valeur immédiate alors l'adresse de l'opérande = (champs opérande) + (X) = (150) + 50

4.4.5 Adressage Relatif

L'adresse effectif de l'opérande est obtenue en additionnant le contenu du compteur ordinal (CO) au contenu du champ adresse de l'instruction.

En général, ce type d'adressage est utilisé dans les instruction de branchement.

Exemple :

Supposant que le CO contient la valeur 100

L'instruction : BR 150 aura comme effet un branchement vers l'instruction d'adresse : 100 + 150 = 250, voir Figure (4.7).

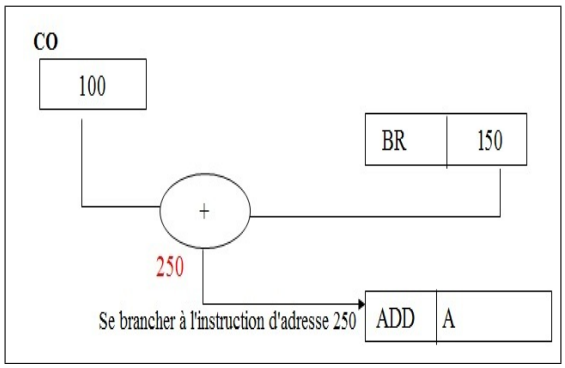


FIGURE 4.7 – Adressage relatif

4.4.6 Adressage Basé

Avec ce mode d'adressage, l'adresse de l'opérande est obtenue en additionnant le contenu du registre de base au contenu du champ adresse de l'instruction.

Le registre de base indique l'adresse du début d'un bloc de 2^k mots mémoires.

Le champ adresse de l'instruction contient le déplacement dans le bloc.

4.5 Déroulement d'exécution d'une instruction

Le traitement d'une instruction est décomposé en trois phases :

— Phase 1 : Recherche de l'instruction à traiter et décodage.

— Phase 2 : Recherche de l'opérande et exécution de l'instruction.

— Phase 3 : Passage à l'instruction suivante (mise à jour du compteur ordinal CO).

Chaque phase comporte un certain nombre d'opérations élémentaires (microcommandes) exécutées dans un ordre bien précis (elle sont gérées par le séquenceur).

Remarque :

Les phases 1 et 3 ne changent pas pour l'ensemble des instructions, par contre la phase 2 change selon l'instruction et le mode d'adressage utilisé.

Exemple :

Déroulement de l'instruction d'addition en mode immédiat ACC ← (ACC) + Valeur

Phase 1 : Rechercher l'instruction à traiter.

— Mettre le contenu du CO dans le Registre d'Adresse Mémoire : RAM ← (CO).

— Lire l'instruction à partir de la mémoire.

— Transférer l'instruction à partir de la MC vers le Registre d'Instruction RI : RI ← (MC RAM).

— Analyser et décoder l'instruction.

Phase 2 : Traitement

— Transférer l 'opérande à partir du RI vers l'UAL : UAL ← (RI).ADR

— Exécuter l'opération suivant le code opération de l'instruction.

Phase 3 : Passer à l'instruction suivante

— Incrémenter le CO : CO ← CO + 1

Remarque :

Le CO (Compteur Ordinal) contient l'adresse de la prochaine instruction à exécuter.

4.6 Conclusion

Le présent chapitre a abordé la présentation générale des organes de l'ordinateur à savoir les unités de traitement, de stockage et mémorisation, ainsi que les unités des Entrées / Sorties.

Par la suite, le codage d'instruction, les machines à 1 adresse, 2 adresses, et 3 adresses ont été présentés, suivie par les modes d'adressage et le déroulement d'exécution d'une instruction.

Quelques exercices sont proposés à la fin de ce chapitre pour une bonne maitrise des notions présentées dans ce cours.

Le prochain chapitre fera l'objectif de la logique combinatoire et séquentielle.

4.7 Exercices

Exercice 01

Déterminer la condition d'adressage après l'exécution de l'instruction de chargement dans chacun des trois cas donnés par la suite, sachant que :

(reg de base) = 150, (reg indexe) = 25, (PC) = 01, (ACC) = 20, et que le contenu de la mémoire principale soit comme suit : (@4) = 25, (@120) = 100, (@125) = 5, (@200) =120, (@250) = 40.

— LOAD 200 qui a pour effet le chargement de l'accumulateur avec la valeur 100.

— LOAD 4 qui a pour effet le chargement de l'accumulateur avec la valeur 4.

— LOAD 100 qui a pour effet le chargement de l'accumulateur avec la valeur 40.

Exercice 02

Supposons qu'un ordinateur ayant un format d'instruction à une adresse, Figure (4.8).

Donnez le déroulement des instructions suivantes :

— Chargement du contenu de l'@ 45.

— Addition du contenu de l'@46.

— Rangement dans la cellule d'@ 60.

00	LOAD 45	45	126
01	ADD 46	46	-15
02	STORE 60	47	12
...		...	
...		60	143
...		....	
...		100	PRINT 60

FIGURE 4.8 – Machine avec un format d'instruction à 1 adresse

Exercice 03

On dispose d'un ordinateur ayant un format d'instruction à 1 adresse.

Déroulez l'instruction de chargement (LOAD 100) selon le contenu de la mémoire centrale illustrée sur la Figure (4.9), en utilisant un :

— L'adressage direct.

— L'adressage indirect.

— L'adressage immédiat.

— L'adressage relatif avec (registre de base) = 150.

— L'adressage indexé avec (registre indexe) = 50.

@100	200

@150	15

@200	20

@250	25

FIGURE 4.9 – Représentation du contenu de la mémoire

Exercice 04

Soit le programme suivant :

ADD 45, 46
ADD 46,47
MOUVE 47, 60

— Donnez l'exécution le déroulement) de ce programme avec un adressage direct en utilisant le contenu de la mémoire illustrée sur la Figure (4.8).
— Calculez le nombre d'accès à la mémoire centrale.

Chapitre 5

La logique combinatoire et séquentielle

Dans ce chapitre, nous commençons par introduction à l'algèbre de Boole à travers la présentation des variables et fonctions booléennes ainsi que quelque propriétés appliquées sur l'algèbre de Boole. Aussi, nous abordons les circuits combinatoires avec des exemples de ces circuits. Par la suite, nous passons aux circuits séquentiels.

Plan

5.1 Algèbre de Boole

Les machines numériques sont constituées d'un ensemble de circuits électroniques.

De nombreux dispositifs électroniques, électromécanique fonctionnent en tout ou rien.

Ceci dit qu'ils peuvent prendre 2 états. Exemple :

- Arrêt / Marche
- Ouvert / Fermé
- Avant / Arrière
- Vrai / Faux

Chaque circuit fournit une fonction logique bien déterminée (addition, comparaison, ...).

Pour concevoir et réaliser ce circuit, on doit avoir un modèle mathématique de la fonction réalisée par ce circuit.

Ce modèle doit prendre en considération le système binaire.

Le modèle mathématique utilisé est celui de Boole.

5.1.1 Définition

L'algèbre de Boole est applicable au raisonnement logique qui traite des fonctions à variables binaires (deux valeurs), et produit un résultat booléen, c'est-à-dire vrai ou faux. Elle ne s'applique pas aux systèmes à plus de deux états d'équilibre, et permet d'étudier les circuits logiques (un système logique sert à modifier des signaux).

L'algèbre de Boole permet de manipuler des valeurs logiques, sachant qu'une valeur logique n'a que deux états possibles : Vrai(1) ou Faux(0).

Elle possède trois opérateurs (fonctions de base) booléens :

NOT (NON),

AND (ET),

OR (OU).

5.1.2 Variables et fonctions Booléennes

La variable logique est une grandeur qui peut prendre 2 valeurs qui sont repérées habituellement 0 ou 1, voir Figure (5.1).

La variable 0 sera associée à un niveau bas (une tension nulle).

La variable 1 sera associée à un niveau haut (une tension positive).

Une variable logique se note par une lettre comme en algèbre x, y, z, ...

Une fonction logique F des n variables logiques $(x_1, x_2, ...x_n)$, notée par exemple $Fx_1, x_2, ...x_n)$, associe une valeur 0 ou 1 aux différentes combinaisons possibles des n variables logiques $(x_1, x_2, ...x_n)$.

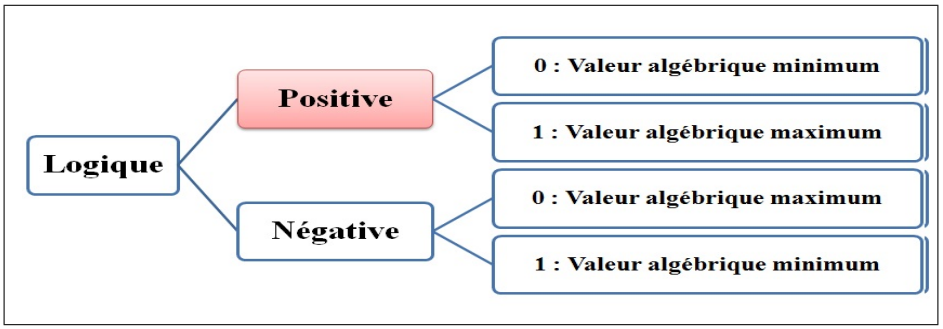


FIGURE 5.1 – Variables logiques : positive / négative

Chaque variable logique x_i peut prendre la valeur 0 ou la valeur 1.

Au total, il y a 2^n combinaisons possibles des n variables logiques $(x_1, x_2, ...x_n)$.

Toutes les fonctions logiques sont formées des 3 fonctions de base : et, ou, non. On aura donc :

— x et y ;

— x ou y ;

— non x.

Une fonction logique se note par une lettre comme en algèbre. Elle possède une ou plusieurs variables logiques d'entrée et une variable logique de sortie.

Exemple :

Considérons une fonction F de deux variables x et y. Il y a donc 2^2= 4 combinaisons possibles de ces deux variables.

Elle peut être définie par une expression logique ou par une table de vérité donnant la valeur de la fonction pour chaque combinaison des variables logiques.

Exemple de fonction logique :

$F(x, y, z, t) = (x \ et \ y) \ ou \ z \ et \ (non \ t)$

$F(x, y, z, t) = (x * y) + z * (\bar{t})$

La table de vérité permet la connaissance de la sortie (d'un circuit logique) en fonction des diverses combinaisons des valeurs des entrées.

Le nombre de colonnes est le nombre total des variables d'entrées plus la colonne qui représente la sortie.

Le nombre de lignes est 2^n sachant que n est le nombre des variables logiques.

Exemple :

Une fonction de trois entrées (3 variables logiques) et une sortie est représentée par

une table de 4 colonnes et 8 lignes.

5.1.3 Fonctions logiques de base

Fonction inversion NON (NOT)

Cette fonction est également appelée complément, notée : $F = \bar{x}$, voir Table (5.1).

x	$F(x) = \bar{x}$
0	1
1	0

TABLE 5.1 – Table de vérité de la fonction inversion (Non)

Relation caractéristique :

$$\bar{\bar{x}} = x \forall x$$

Fonction OU (OR)

C'est une fonction de deux variables également appelée somme logique, notée : $F = x + y$, voir Table (5.2).

x	y	$F(x, y) = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 5.2 – Table de vérité de la fonction Ou

La fonction OU vaut 1 si au moins une des variables vaut 1.

Relations caractéristiques :

$$x + 0 = x$$
$$x + 1 = 1$$
$$x + x = x$$

Fonction ET (AND)

C'est une fonction de deux variables également appelée produit logique, notée : $F = x * y$, voir Table (5.3). La fonction Et ne vaut 1 que si toutes les variables valent 1.

Relations caractéristiques :

$$x * 1 = x.$$
$$x * 0 = 0.$$
$$x * x = x.$$
$$x * \bar{x} = 0.$$

x	y	$F(x,y) = x * y$
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 5.3 – Table de vérité de la fonction Et

5.1.4 Propriétés des fonctions logiques de base

1. Les fonctions ET et OU sont commutatives :

$$x + y = y + x$$
$$x * y = y * x$$

2. Les fonctions ET et OU sont distributives l'une par rapport à l'autre

$$x * (y + z) = (x * y) + (x * z)$$
$$x + (y * z) = (x + y) * (x + z)$$

3. Ordre de priorité

Dans les expressions logiques (formules ne faisant intervenir que des variables logiques et les trois lois ci-dessus) il existe un ordre de priorité qui est le suivant en décroissant : NON, ET, OU.

Ces règles de priorité dispensent d'un certain nombre de parenthèses. Par exemple :

$$x + (y + z) = (x + y) + z = x + y + z$$
$$x * (y * z) = (x * y) * z = x * y * z$$
$$x + (y * z) = x + y * z \neq (x + y) * z$$

4. Théorème de De Morgan

C'est l'une des propriétés les plus importantes des fonctions logiques.

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

Démonstration du théorème par table de vérité, voir Table (5.4) :

5.1.5 Simplification des fonctions logiques

Afin d'assurer la réalisation physique d'une fonction logique d'une façon plus simple et économique, il est nécessaire de chercher l'expression la plus simple de cette fonction.

Simplifier une expression booléenne revient à réduire le nombre des opérateurs, et le nombre des entrées sur les opérateurs réalisant la fonction logique pour minimiser le temps de propagation de l'information à travers les circuits.

x	y	$x+y$	$\overline{(x+y)}$	$\bar{x}$	$\bar{y}$	$\bar{x}*\bar{y}$	$\overline{(x*y)}$	$\bar{x}+\bar{y}$
0	0	0	1	1	1	1	1	1
0	1	1	0	1	0	0	1	1
1	0	1	0	0	1	0	1	1
1	1	1	0	0	0	0	0	0

TABLE 5.4 – Démonstration du théorème de De Morgan

Méthode algébrique

Pour obtenir une expression plus simple de la fonction par cette méthode, il faut utiliser :

— Les théorèmes et les propriétés de l'algèbre de Boole.

— La multiplication par un : $(X + \bar{X})$.

— L'addition d'un terme nul : $(X * \bar{X})$.

Exemple : Simplification de la fonction F(x,y)

$$\begin{aligned} F(x,y) &= x + x * y \\ &= x * (1 + y) \\ &= x * 1 \\ &= x \end{aligned}$$

Remarque :

Les règles et propriétés de l'algèbre de Boole permettent de simplifier les fonctions mais reste une méthode relativement lourde. La méthode algébrique ne permet jamais de savoir si on aboutit ou pas à une expression minimale de la fonction. Nous pourrons alors utiliser la méthode du tableau de KARNAUGH (méthode graphique)

Méthode graphique

Le tableau de KARNAUGH permet de visualiser une fonction et d'en tirer intuitivement la forme simplifiée de la fonction. Pour une fonction à n variables, le tableau aura 2^n cases. Chaque case représente la valeur de la fonction pour une combinaison de variables précise.

Le principe est de faire les regroupements des cases adjacentes, voir Figure(5.2). Les cases portant le même chiffre sont des exemples de cases adjacentes , et qui peuvent être regroupées.

Il s'agit d'un tableau à double entrées dans lequel chaque combinaison des variables d'entrée est associée à une case qui contient la valeur de la fonction.

Les groupements des cases adjacentes doivent être de taille maximale (nombre max de cases dans le groupement).

On cesse d'effectuer les groupements lorsque tous les uns appartiennent au moins à l'un d'eux.

Remarque :

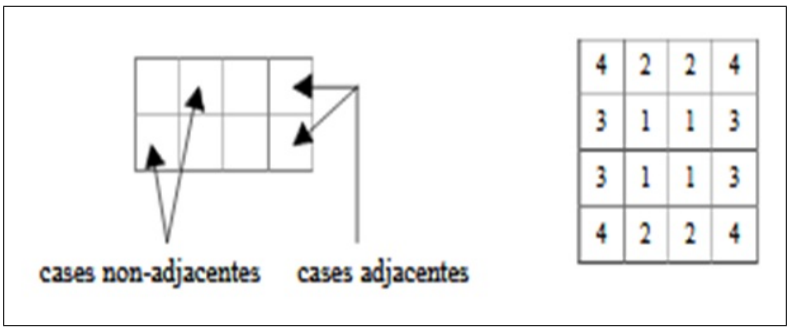


FIGURE 5.2 – Regroupement des cases adjacentes

Avant de tirer la fonction à partir du tableau de KARNAUGH, il faut respecter les règles suivantes :

- Grouper tous les uns.
- Grouper le maximum des uns dans un seul groupement.
- Il est interdit de faire des groupements en diagonale, donc il n'y a que des groupements carrés et rectangles ;
- Le nombre des uns dans un groupement est une puissance de 2 (est égal à 2^k).
- Un 1 peut figurer dans plus qu'un groupement.
- Il faut imaginer le tableau comme une sphère, c'est à dire que un "1" étant placé tout en haut à droite pourra se regrouper avec un "1" placé tout en haut à gauche ou bien avec un "1" placé tout en bas à droite.
- Ne prenez que l'état des variables qui ne changent pas.
- Ne prenez que l'état des variables qui ne changent pas.
- Il faut séparer les groupements par un "ou" (le symbole "+").

Exemples :

La Figure (5.3) illustre la simplification de quelques fonctions logiques en utilisant le tableau de Karnaugh :

$F1 = A\bar{B}\bar{C} + A\bar{B}C$

$F2 = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D$

$F3 = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}\bar{C}D$

$F4 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$

5.2 Circuits combinatoires

Tout ordinateur est conçu à partir des circuits intégrés qui ont chacun une fonction spécialisée (Unité Arithmétique et Logique, mémoire, circuit décodant les instructions etc.).

Ces circuits sont fait à partir de circuits logiques dont le but est d'exécuter des opérations sur des variables logiques (binaires).

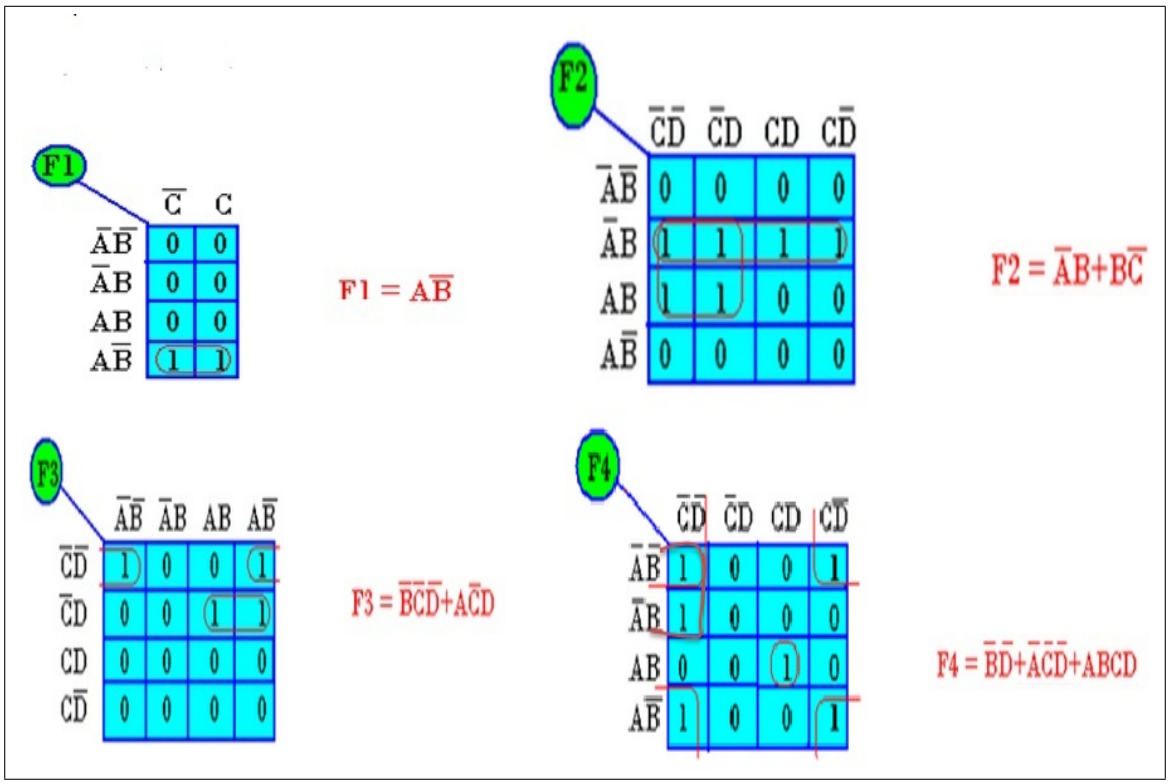


FIGURE 5.3 – Exemples de simplification graphique des fonctions logiques

Les fonctions de sortie s'expriment selon des expressions logiques des seules variables d'entrée.

Un circuit combinatoire est défini par une ou plusieurs fonctions logiques. C'est un circuit numérique dont les sorties dépendent uniquement des entrées.

Les circuits combinatoires sont souvent vus comme des boîtes noires régies par un fonctionnement déterministe avec un ensemble d'entrées/sorties, voir Figure (5.4).

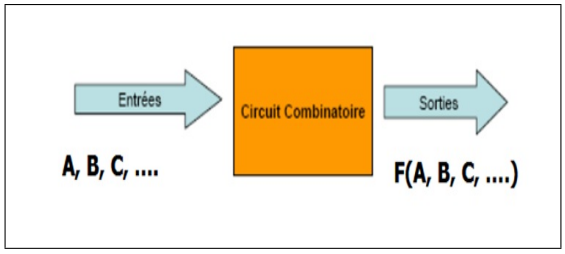


FIGURE 5.4 – circuit combinatoire

Étant donné la représentation d'une fonction (table de vérité ou expression booléenne), il faut savoir construire le circuit réalisant la fonction uniquement à l'aide des portes logiques.

5.2.1 Représentation des fonctions logiques de base (Portes logiques)

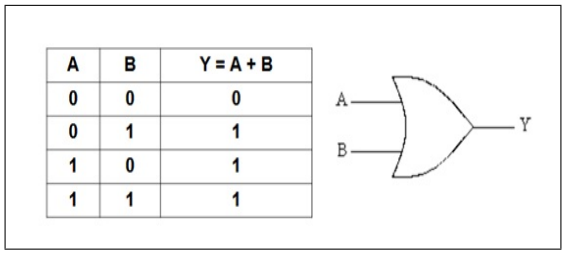
Sur les schémas de circuits électroniques, les fonctions logiques sont représentées par des symboles que l'on appelle généralement ' portes logiques '.

Les fonctions NON, ET et OU sont associées aux symboles représentés comme suit :

Porte OU logique

— Au moins deux entrées.

— La sortie d'une fonction OU est dans l'état 1 si au moins une de ses entrées est dans l'état 1, voir Figure (5.5).



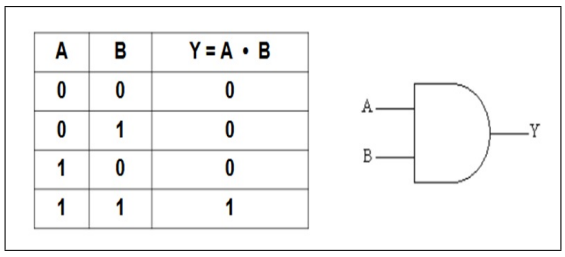
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

FIGURE 5.5 – Porte OU logique

Porte Et logique

— Au moins deux entrées.

— La sortie d'une fonction AND est dans l'état 1 si et seulement si toutes ses entrées sont dans l'état 1, voir Figure (5.6).



A	B	Y = A • B
0	0	0
0	1	0
1	0	0
1	1	1

FIGURE 5.6 – Porte ET logique

Porte Non logique (Inverseur)

— Une seule entrée et une seule sortie

— La sortie d'une fonction NON prend l'état 1 si et seulement si son entrée est dans l'état 0, voir Figure (5.7).

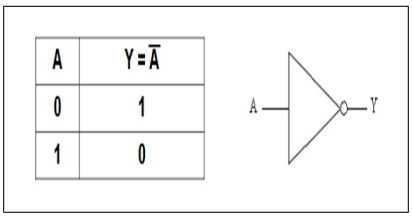


FIGURE 5.7 – Porte Non logique

Porte Non Et

— Elle est constituée par un inverseur à la sortie d'une porte ET, voir Figure (5.8).

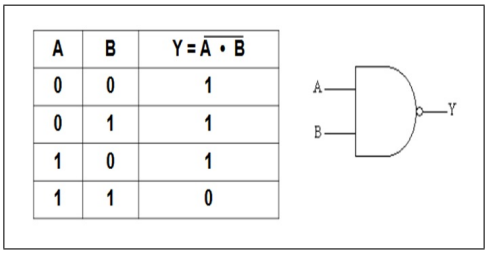


FIGURE 5.8 – Porte Non ET logique

Porte Non OU

— Une négation à la sortie d'une porte OU constitue une fonction NON OU (NOR : NOT OR), voir Figure (5.9).

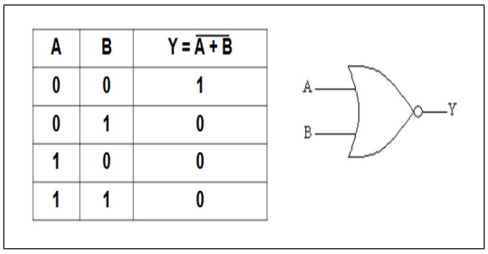


FIGURE 5.9 – Porte Non OU logique

Porte OU-EXCLUSIF (XOR)

— Deux entrées

— La sortie d'une fonction XOR est dans l'état 1 si les deux entrées sont différentes, voir Figure (5.10).

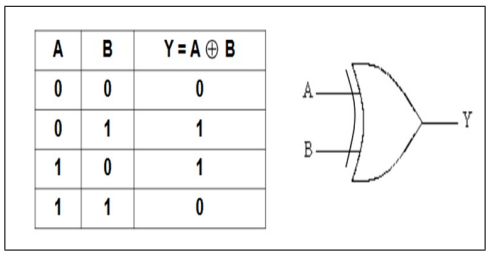


FIGURE 5.10 – Porte XOR

5.2.2 Conception d'un circuit combinatoire

Toute fonction logique peut être réalisée à l'aide des portes logiques. Pour cela, il faut :

— Écrire l'équation de la fonction à partir de sa table de vérité.
— Simplifier l'équation.
— Réaliser l'équation à l'aide des portes disponibles.

En combinant entre les différentes portes logiques, on peut à priori réaliser n'importe quelle fonction logique.

On appelle ' logigramme ' la réalisation d'une fonction à laide des portes logiques de base, voir Figure (5.11).

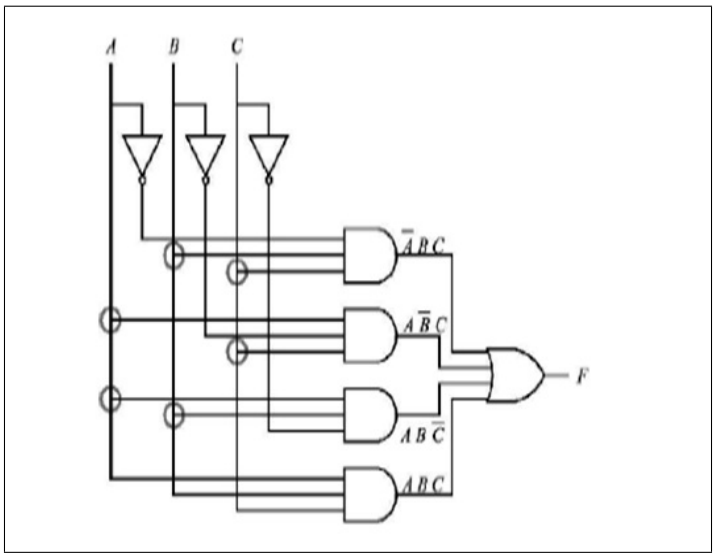


FIGURE 5.11 – Fonction logique réalisée à l'aide de portes logiques

5.2.3 Exemples de circuits combinatoires

Codeur

C'est un dispositif qui effectue l'opération inverse du décodeur. On l'appelle aussi un encodeur. Une seule entrée est activée à la fois, ce qui correspond à un nombre binaire en sortie.

La Figure (5.12), illustre un exemple d'un codeur de 4 entrées et 2 sorties.

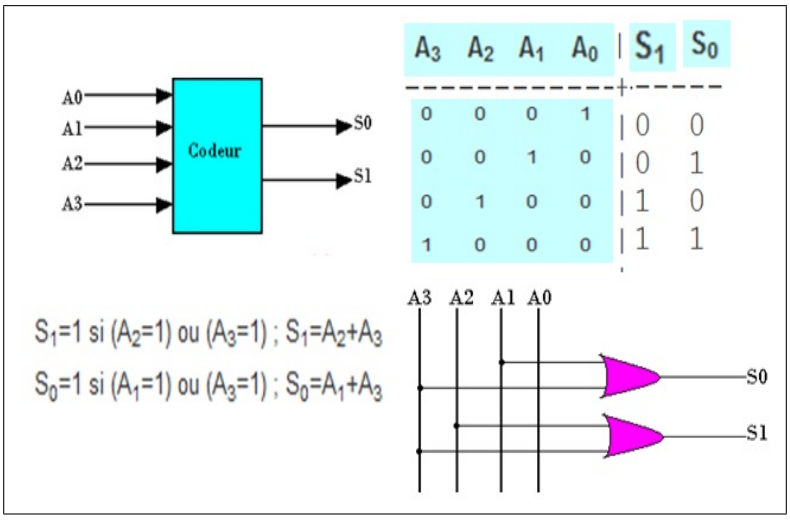


FIGURE 5.12 – Codeur de 4 entrées et 2 sorties

Décodeur

Le décodeur réalise la fonction inverse du codeur. C'est un circuit logique comportant n entrée et 2^n sorties.

Lorsque le signal de validation est actif, seule la sortie dont le numéro correspond à la valeur binaire affichée sur l'entrée est activée et toutes les autres sorties sont inactives.

La Figure (5.13), illustre un exemple d'un décodeur de 2 entrées et 4 sorties.

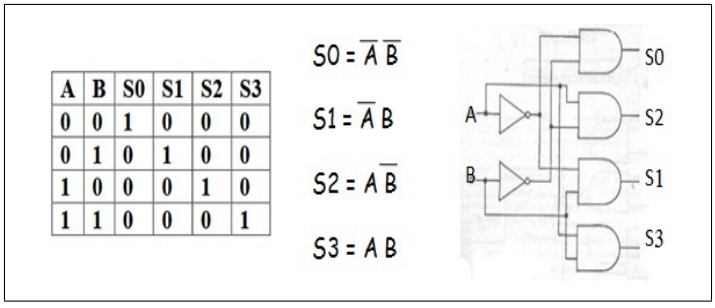
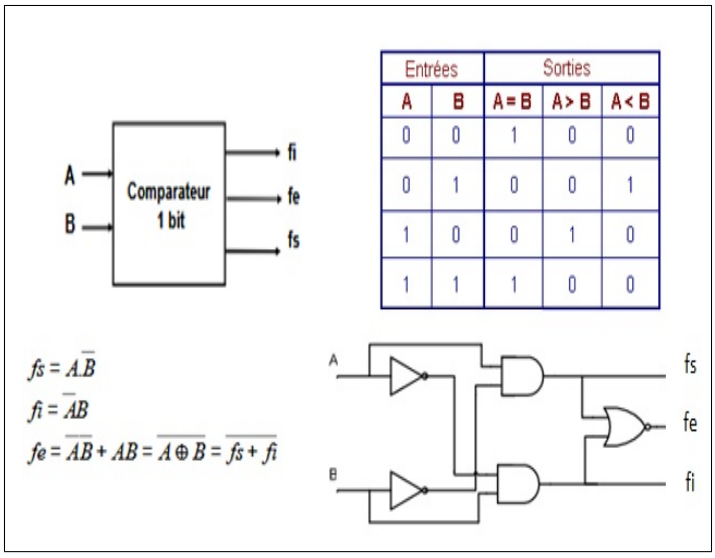


FIGURE 5.13 – Décodeur de 2 entrées et 4 sorties

Comparateur

C'est un circuit combinatoire qui permet de comparer entre deux nombres binaire A et B. La Figure (5.14), illustre un exemple de comparateur de deux bits.



Entrées		Sorties		
A	B	A=B	A>B	A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

FIGURE 5.14 – comparateur de deux bits A et B

Il possède 2 entrées : A (sur un bit) et B (sur un bit), et 3 sorties :

— fe : égalité (A=B)

— fi : inférieur (A < B)

— fs : supérieur (A > B)

5.3 Circuits séquentiels

Tous les circuits présentés précédemment ont été réalisés avec des portes logiques les unes derrière les autres. A aucun moment la sortie d'une porte logique n'a été rebouclée, plus ou moins directement, vers son entrée, voir Figure (5.15(a)). Ce type de circuit est décrit dans le cadre de la logique combinatoire.

Pour ces circuits une même combinaison des entrées donnera toujours la même valeur de sortie.

Un circuit combinatoire est un circuit numérique dont les sorties dépendent uniquement des entrées. L'état du système ne dépend pas de l'état interne du système, avec aucune mémoration de l'état du système.

La situation est complètement différente lorsque la sortie d'une porte est rebouclée sur son entrée, voir Figure (5.15(b)).

Dans cette situation l'état de sortie du circuit à un instant donné dépend de :

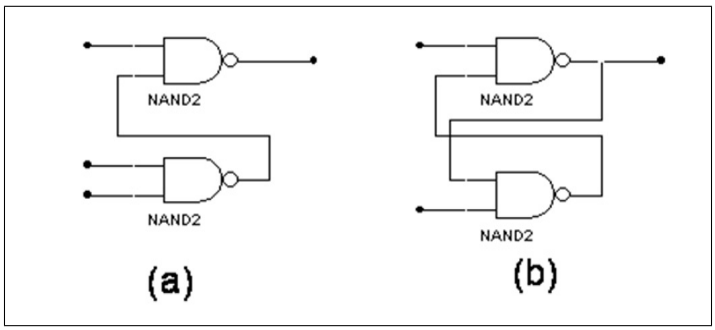


FIGURE 5.15 – Circuit combinatoire (a) / circuit séquentiel (b)

— La valeur des entrées à cet instant.

— La valeur de la (ou des) sortie(s) aux instants antérieurs.

Ces circuits sont évidemment déterministes mais leur état présent est fixé par toute la séquence des entrées, c'est-à-dire par les valeurs qui ont précédé les valeurs actuelles.

Une étude temporelle est donc indispensable pour en comprendre le fonctionnement d'un tel système.

Ces circuits doivent être décrits et étudiés dans le cadre de la logique séquentielle.

Le terme 'séquentiel' fait ici référence à une succession d'événements dans le temps et s'applique aux systèmes où le temps joue un rôle à part entière pour la détermination de l'état de sortie.

Autrement dit, un circuit séquentiel est un circuit numérique (logique) dont l'état à l'instant (t+1) est une fonction des entrées en même instant (t+1) et de l'état précédent du système (l'instant t), Figure (5.16).

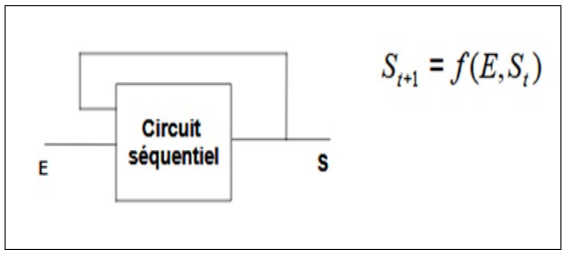


FIGURE 5.16 – Représentation d'un circuit séquentiel

En théorie des circuits électroniques, la logique séquentielle est un type de logique dont les résultats ne dépendent pas seulement des données actuellement traitées mais aussi des données traitées précédemment.

Elle s'oppose à la logique combinatoire, dont les résultats sont fonction et seulement fonction des données actuellement traitées.

En d'autres termes, la logique séquentielle utilise la notion de mémoire de stockage (Bascules, registres, etc.) alors que la logique combinatoire n'en a pas.

De manière générale, les circuits séquentiels font apparaître des retours qui permettent de mémoriser des informations relatives aux états antérieurs appliqués sur le circuit.

La sortie d'un circuit séquentiel est ainsi fonction de variables internes également appelées variables d'état.

L'élément de base de la logique séquentielle est la bascule.

5.3.1 Bascule

Un bascule est un circuit logique capable, dans certaines circonstances, de maintenir les valeurs de ses sorties malgré les changements de valeurs d'entrées (mémoire). Elle permet le passage de la logique combinatoire à la logique séquentielle.

Les bascules sont les circuits séquentiels élémentaires permettant de mémoriser une information binaire (bit) sur leur sortie. Elles constituent le point mémoire élémentaire.

On distingue deux catégories principales de bascules :

— Les bascules asynchrones (verrous latch).

— Les bascules synchrones (bascules flip-flop).

Bascule RS

La bascule RS est la plus simple des bascules. Elle est réalisée à partir de deux portes NOR ou de deux portes NAND. Le schéma de la bascule RS avec des portes NOR est illustré sur la Figure (5.17). La bascule possède deux entrées notées R et S ainsi que deux sorties conventionnellement notées Q et $\bar{Q}$.

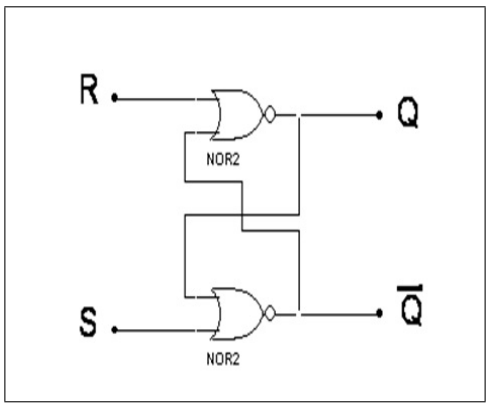


FIGURE 5.17 – Bascule RS

Pour comprendre le fonctionnement de la bascule RS, on va étudier le comportement des variables de sorties Q et $\bar{Q}$ en fonction des variables d'entrée R et S. On désigne :

— Q_t La variable de sortie à l'instant t (état présent de la bascule).

— Q_{t+1} La variable de sortie à l'instant t+1 (état futur de la bascule).

Cas 1 : Quand S = R = 0

• On suppose que $Q_t = 0$ alors on aura le schéma illustré sur la Figure (5.18 (a)).

On remarque que $Q_t = 0$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{0 + 0} = 1$

Du moment où $\bar{Q}_t = 1$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \bar{Q}_t + R = 1 + 0 = 0$

On aura donc $Q_t = Q_{t+1}$ L'état de la bascule est stable.

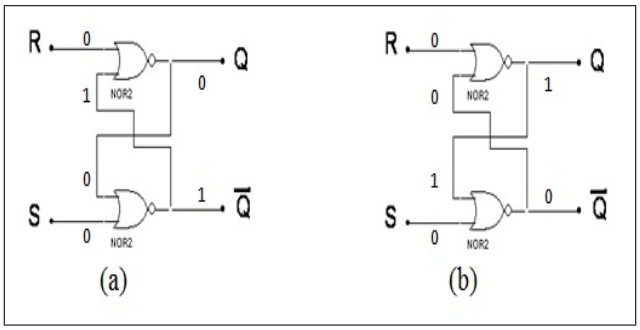


FIGURE 5.18 – Bascule RS avec R=S=0

• On suppose que $Q_t = 1$ alors on aura le schéma représenté sur la Figure (5.18 (b)).

On remarque que $Q_t = 1$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{1 + 0} = 0$

Du moment où $\bar{Q}_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 0} = 1$

On aura donc $Q_t = Q_{t+1}$ L'état de la bascule est stable

Cas 2 : Quand S = 1 et R = 0

• On suppose que $Q_t = 0$ alors on aura le schéma illustré sur la Figure (5.19 (a)).

On remarque que $Q_t = 0$ on a $barQ_t = \overline{S + Q_t} = \overline{1 + 0} = 0$

Du moment où $Q_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 0} = 1$

On aura donc $Q_t = 0$ et $Q_{t+1} = 1$

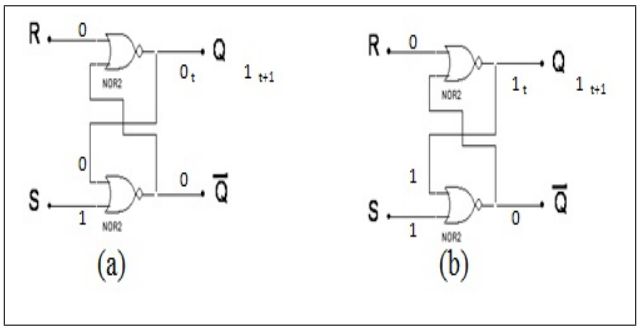


FIGURE 5.19 – Bascule RS avec S=1 et R=0

• On suppose que $Q_t = 1$ alors on aura le schéma de la Figure (5.19 (b)) :

On remarque que $Q_t = 1$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{1 + 1} = 0$

Du moment où $\bar{Q}_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 0} = 1$

On aura donc $Q_t = Q_{t+1}$

Remarque :

Quand la variable S = 1, la variable de sortie Q garde son état si elle est à 1, sinon elle prend la valeur 1.

Cas 3 : Quand S = 0 et R = 1

• On suppose que $Q_t = 0$ alors on aura le schéma montré sur la Figure (5.20(a)).

On remarque que $Q_t = 0$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{0 + 0} = 1$

Du moment où $\bar{Q}_t = 1$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{1 + 0} = 0$

On aura donc $Q_t = 0$ et $Q_{t+1} = 0$

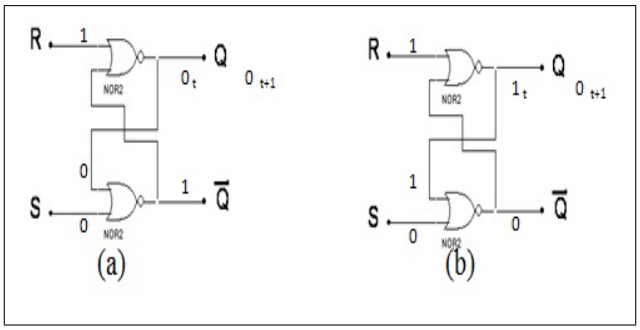


FIGURE 5.20 – Bascule RS avec S=0 et R=1

• On suppose que $Q_t = 1$ alors on aura le schéma de la Figure (5.20(b)).

On remarque que $Q_t = 1$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{0 + 1} = 0$

Du moment où $\bar{Q}_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 1} = 0$

On aura donc $Q_t = 1$ et $Q_{t+1} = 0$.

Remarque :

Quand la variable R = 1, la variable de sortie Q garde son état si elle est à 0, sinon elle prend la valeur 0.

Cas 4 : Quand S = R = 1

• On suppose que $Q_t = 0$ alors on aura le schéma illustré sur la Figure (5.21(a)).

On remarque que $Q_t = 0$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{1 + 0} = 0$

Du moment où $\bar{Q}_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 1} = 0$

On aura donc $Q_t = 0$ et $Q_{t+1} = 0$

• On suppose que $Q_t = 1$ alors on aura le schéma représenté sur la Figure (5.21(b)).

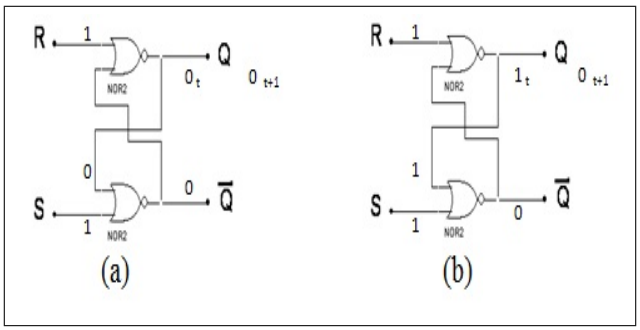


FIGURE 5.21 – Bascule RS avec S = R = 1

On remarque que $Q_t = 1$ on a $\bar{Q}_t = \overline{S + Q_t} = \overline{1 + 1} = 0$

Du moment où $\bar{Q}_t = 0$, la variable de sortie Q à l'instant t+1 :

$Q_{t+1} = \overline{\bar{Q}_t + R} = \overline{0 + 1} = 0$

On aura donc $Q_t = 0$ et $Q_{t+1} = 0$

Après l'étude des quatre cas précédents , nous constatons que le problème se pose lorsque l'une des deux entrées passe à 1.

En pratique les deux entrées ne peuvent pas changer rigoureusement au même instant, l'une substitue nécessairement avant l'autre.

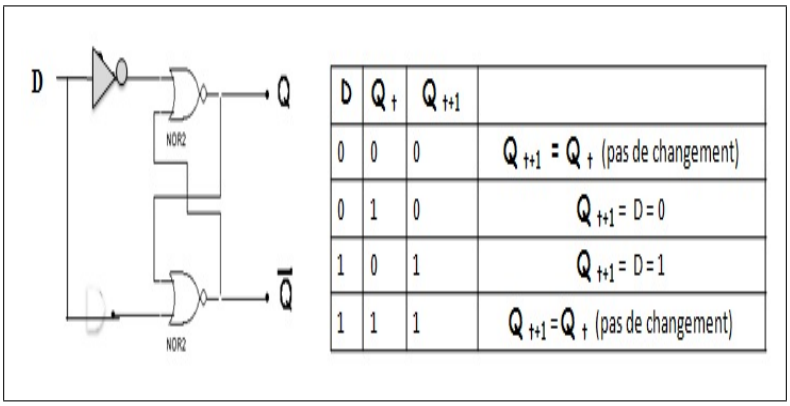
A ce moment, la bascule prend l'un des état (0 ou 1) selon que ce soit S ou R qui change d'état en premier. On dit que l'état de la bascule est indéterminé, il peut être à 1, comme il peut être à 0, voir Figure (5.22).

R	S	Q_t	Q_{t+1}	
0	0	0	0	$Q_t = Q_{t+1}$
0	0	1	1	
0	1	0	1	$Q_{t+1} = 1$ (mise à 1)
0	1	1	1	
1	0	0	0	$Q_{t+1} = 0$ (Remise à 0)
1	0	1	0	
1	1	0	X	Etats indéterminés
1	1	1	x	

FIGURE 5.22 – Synthèse de la bascule RS

Bascule D

Une autre manière de résoudre le problème d'ambiguité rencontré avec la bascule RS, lorsque R = S = 1, est de faire en sorte que ce cas ne se présente jamais à l'entrée de la bascule. Pour cela, on utilise qu'une seule variable d'entrée externe D, et on parle alors de la bascule D, voir Figure (5.23).



D	Q_t	Q_{t+1}	
0	0	0	$Q_{t+1} = Q_t$ (pas de changement)
0	1	0	$Q_{t+1} = D = 0$
1	0	1	$Q_{t+1} = D = 1$
1	1	1	$Q_{t+1} = Q_t$ (pas de changement)

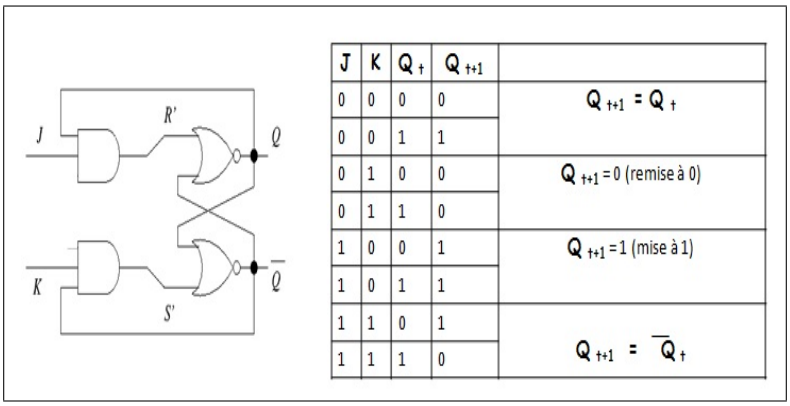
FIGURE 5.23 – Bascule D

L'expression simplifiée de Q_{t+1}

$$Q_{t+1} = D$$

Bascule JK

La situation JK diffère de la bascule RS du fait que, quand les deux variables d'entrée passent simultanément à 1, l'état de la bascule n'est pas indéterminé mais la bascule passe à l'état opposé, voir Figure (5.24).



J	K	Q_t	Q_{t+1}	
0	0	0	0	$Q_{t+1} = Q_t$
0	0	1	1	
0	1	0	0	$Q_{t+1} = 0$ (remise à 0)
0	1	1	0	
1	0	0	1	$Q_{t+1} = 1$ (mise à 1)
1	0	1	1	
1	1	0	1	$Q_{t+1} = \overline{Q}_t$
1	1	1	0	

FIGURE 5.24 – Bascule JK

L'expression simplifiée de Q_{t+1}

$$Q_{t+1} = \bar{K}Q_t + J\bar{Q}_t$$

Bascule T

La bascule T ressemble à la bascule JK à une seule entrée, voir Figure (5.25).

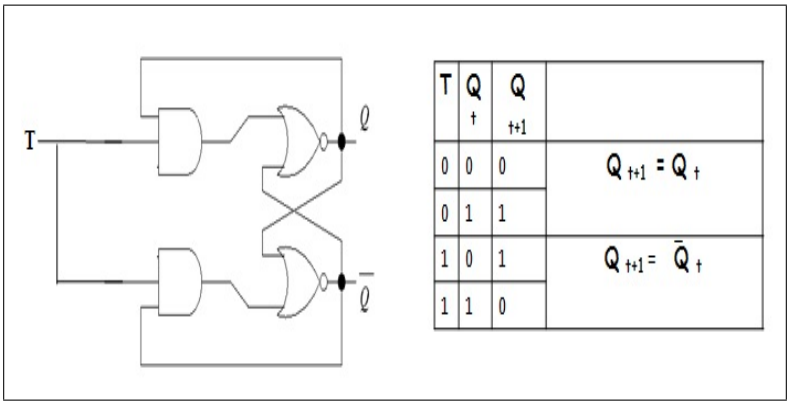


FIGURE 5.25 – Bascule T

L'expression simplifiée de Q_{t+1}

$$Q_{t+1} = T \bigoplus Q_t$$

5.3.2 Déclenchement d'une bascule

Le déclenchement d'une bascule se traduit pas un changement momentané de ses variables d'entrée. Le déclenchement d'une bascule synchrone diffère d'une bascule asynchrone.

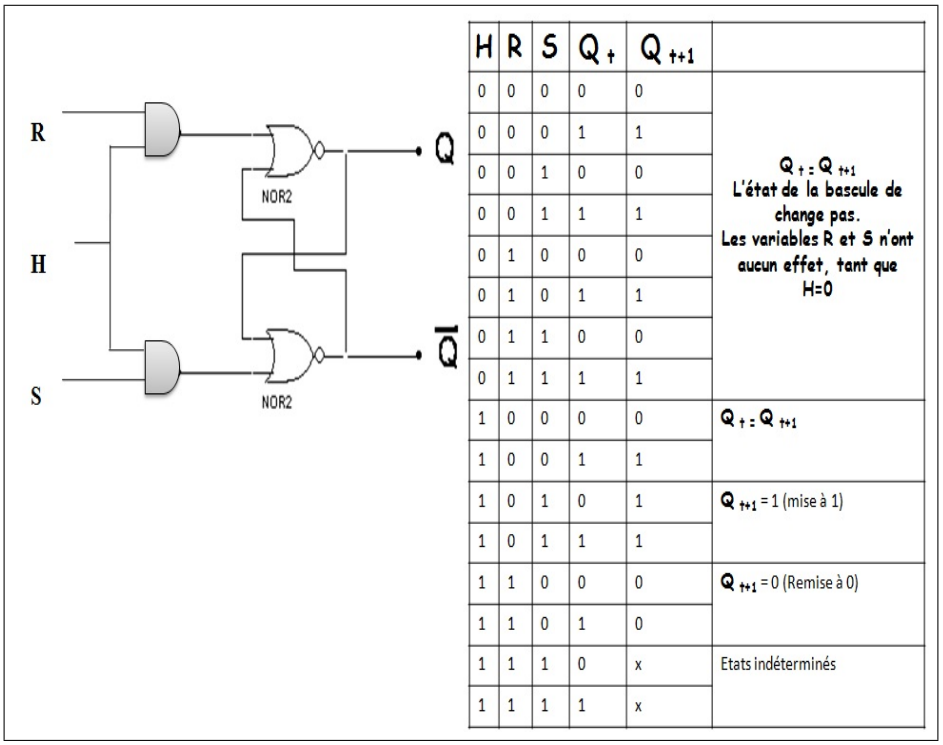
— La bascule asynchrone est déclenchée lorsque les signaux appliqués en entrée changent.

— La bascule synchrone est pilotée par une horloge. Par conséquent, le déclenchement d'une telle bascule est provoqué par des impulsions. Les bascules synchrones peuvent être classifiées selon deux catégories :

1. Bascule synchrone Latch : Réagisse sur un niveau d'horloge. Elle est déclenchée quand H = 1 ou quand H =0
2. Bascule synchrone Flip-flop : Change d'état non pas quand H = 1 ou H =0 mais pendant la transition du signal d'horloge de l'état 0 à 1 (front montant) ou de l'état 0 à 1 (front descendant).

Synchronisation de la bascule RS

L'utilisation d'une horloge (H) permet de synchroniser les changements d'état de la bascule. En effet, il est souvent utile de ne faire changer l'état à une bascule qu'à des instants bien précis, d'où l'intérêt de l'horloge.



H	R	S	Q_t	Q_{t+1}	
0	0	0	0	0	$Q_t = Q_{t+1}$ L'état de la bascule de change pas. Les variables R et S n'ont aucun effet, tant que H=0
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	$Q_t = Q_{t+1}$
1	0	0	1	1	
1	0	1	0	1	$Q_{t+1} = 1$ (mise à 1)
1	0	1	1	1	
1	1	0	0	0	$Q_{t+1} = 0$ (Remise à 0)
1	1	0	1	0	
1	1	1	0	x	Etats indéterminés
1	1	1	1	x	

FIGURE 5.26 – bascule RS synchrone

A partir de la Figure (5.26) on constate que :

— Si H = 0, les variables R et S n'ont aucun effet Sur la bascule.

— Si H = 1, la bascule se retrouve sous le contrôle des deux variables R et S.

5.3.3 Système séquentiel synchrone / asynchrone

Les systèmes séquentiels peuvent être différenciés en fonction de leur mode de fonctionnement qui peut être synchrone ou asynchrone, voir Figure (5.27).

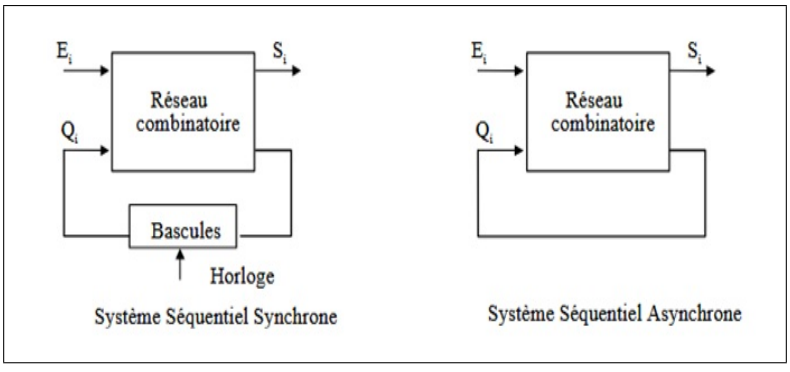


FIGURE 5.27 – Système séquentiel synchrone / asynchrone

Dans le mode synchrone, les éléments de mémorisation sont des bascules. Les modifications d'état du système ne peuvent donc intervenir qu'à des instants très précis, déterminés par des signaux d'horloge.

Par contre, dans le mode asynchrone, la fonction de mémorisation est réalisée par de simples boucles de rétroaction. L'évolution des états ne dépend donc que des modifications intervenant sur les entrées E_i de la machine.

Système asynchrone

On appelle circuits séquentiels asynchrones des circuits séquentiels sans signaux d'horloge. Ce type de système change d'état à tout moment, chaque fois qu'une ou plusieurs entrées changent.

Système synchrone

On appelle circuits séquentiels synchrones des circuits séquentiels qui possèdent une horloge, et dont l'état interne se modifie précisément après chaque front (montant ou descendant) de l'horloge. Ce type de système ne se déclenche et ne se change d'état que lorsqu'un signal le commande (signal d'horloge), suivant des impulsions périodique, voir Figure (5.28).

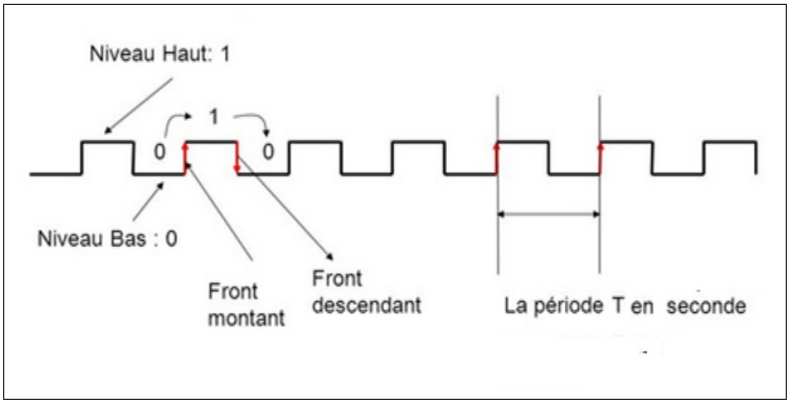


FIGURE 5.28 – Impulsions d'horloge

.

Chronogramme

Pour représenter une fonction logique, nous avons vu les équations logiques, les tables de vérité et les tableaux de Karnaugh. Ces représentations sont suffisantes en logique combinatoire mais n'intègrent pas la notion de temps et ne permettent donc pas de visualiser facilement des systèmes où le temps intervient de façon essentielle. D'où l'intérêt du chrono-

gramme, qui consiste à dessiner un graphique sur lequel le temps sera représenté en abscisse et le niveau logique (0 ou 1) en ordonnée, voir Figure (5.29).

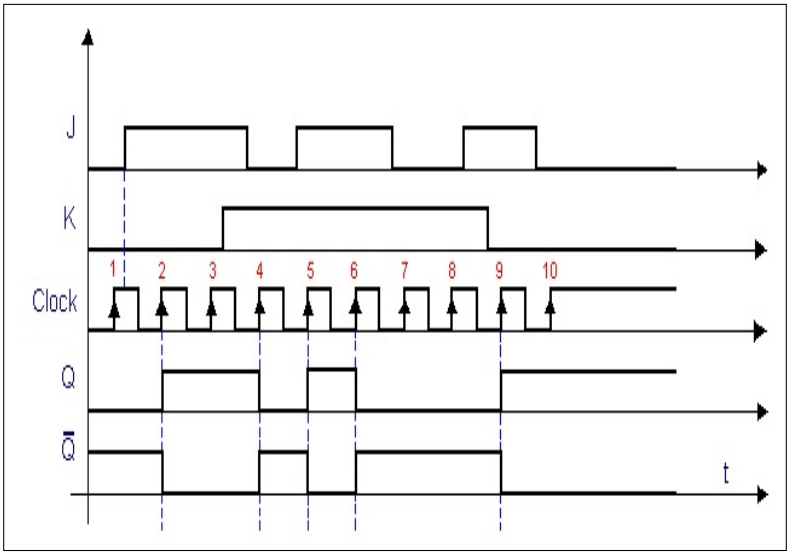


FIGURE 5.29 – Chronogramme

.

5.3.4 Conception d'un système séquentiel

5.3.5 Compteurs

Le compteur est un dispositif séquentiel constitué à l'aide des bascules interconnectées .

Il possède deux fonctions :

— Le comptage (il faut déterminer le pas).

— La mémorisation .

Un compteur peut être synchrone ou asynchrone, voir Figure (5.30) :

— Compteur asynchrone : un compteur est dit asynchrone si les impulsions à compter sont appliquées seulement sur l'horloge H de la première bascule, et l'état de chaque bascule est fonction des états des bascules précédentes.

— Compteur synchrone : un compteur est dit synchrone si les impulsions à compter sont appliquées simultanément sur l'horloge H de toutes les bascules.

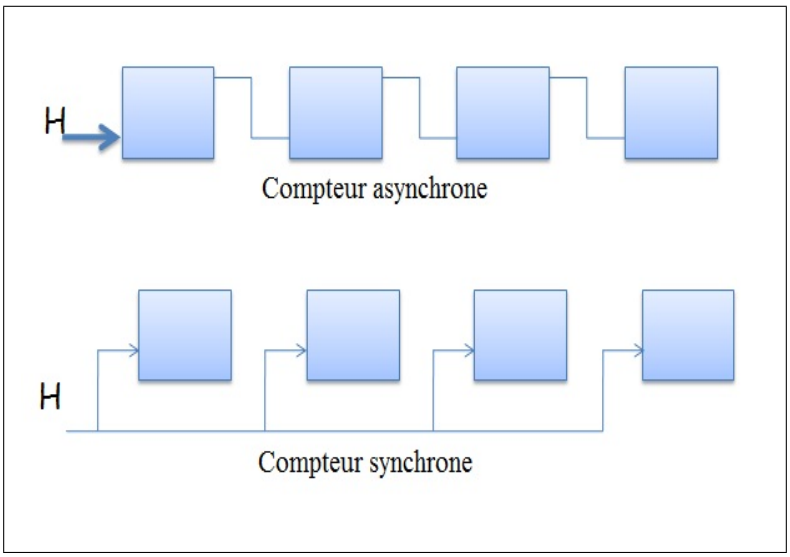


FIGURE 5.30 – Compteur synchrone / asynchrone

5.3.6 Types de compteurs

Compteur binaire

Le compteur binaire est un circuit séquentiel composé d'une succession de bascules, capables de sauvegarder le nombre d'impulsions délivrées par une horloge ou autre signal, en transformant un nombre binaire pour un usage ou affichage ultérieur.

Exemple :

Pour mesurer la fréquence d'un signal, le nombre d'impulsions de ce signal est compté pendant une durée égale à une seconde.

Compteur progressif

Le compteur binaire est dit progressif si son contenu passe d'une valeur binaire m à une valeur binaire $m+1$ (sens croissant) après l'application d'une impulsion d'horloge.

Exemple :

Un compteur binaire formé de trois bascules, et en comptant de 0 jusqu'à 7. La table de vérité des transitions d'un tel compteur après chaque impulsion d'horloge est illustrée sur la Table (5.5).

valeur en décimale	Contenu du compteur
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

TABLE 5.5 – Table de vérité d'un compteur progressif

Compteur régressif

Le compteur binaire est dit régressif si son contenu passe d'une valeur binaire m à une valeur binaire m-1 (sens décroissant) après l'application d'une impulsion d'horloge.

Exemple :

Un compteur binaire formé de trois bascules, et en décomptant de 7 jusqu'à 0. La table de vérité des transitions d'un tel compteur après chaque impulsion d'horloge est illustre par la Table (5.6) :

valeur en décimale	Contenu du compteur
7	111
6	110
5	101
4	100
3	011
2	010
1	001
0	000

TABLE 5.6 – Table de vérité d'un compteur régressif

Compteur modulo N

Un compteur binaire constitué de n bascules est dit modulo N (tel que $N \leq 2^n$), s'il peut compter de 0 jusqu'à $n-1$. La N^{ieme} impulsion le remet, obligatoirement à 0.

Les n étages constituant un tel compteur permettent de présenter tous les états possibles.

Si un certain nombre d'états ne seront jamais utilisés, en fonctionnement normal ($N = 2^n$), on parle d'un compteur incomplet.

Si $N = 2^n$ alors n est le nombre de bascules nécessaires.

Exemple 1 : Compteur modulo 16 : (compter de 0 jusqu'à 15)

$16 = 2^4 \Longrightarrow$ 4 bascules nécessaires

Exemple 2 : Compteur modulo 6 : (compter de 0 jusqu'à 5)

$4 = 2^2 < 6 \Longrightarrow$ 2 bascules de suffisent pas

$8 = 2^3 > 6 \Longrightarrow$ 3 bascules nécessaires (compteur incomplet), voire Table (5.7).

valeur en décimale	**Contenu du compteur**
0	000
1	001
2	010
3	011
4	100
5	101
6	XXX
7	XXX

TABLE 5.7 – Table de vérité d'un compteur modulo 6

Exemple d'un Compteur Asynchrone Progressif

Considérons un compteur asynchrone progressif formé de trois bascules type JK à front descendant. La table des transitions, après chaque impulsion d'horloge H, est donnée par la Table (5.8) :

H	**valeur en décimale**	Q_3	Q_2	Q_1	Q_{3+}	Q_{2+}	Q_{1+}
↓	0	0	0	0	0	0	1
↓	1	0	0	1	0	1	0
↓	2	0	1	0	0	1	1
↓	3	0	1	1	1	0	0
↓	4	1	0	0	1	0	1
↓	5	1	0	1	1	1	0
↓	6	1	1	0	1	1	1
↓	7	1	1	1	0	0	0

TABLE 5.8 – Table de vérité d'un compteur asynchrone progressif

• A chaque front descendant de H, on a : $Q_{1+} = \bar{Q}_1$

D'après la table caractéristique de la bascule JK : $Q_{1+} = \bar{Q}_1 \Rightarrow J_1 = K_1 = 1$

Sachant que la bascule 1 commute au front descendant, on peut écrire : $H = Ck_1$ (Ck_1 est l'horloge de la bascule 1).

D'où : $Ck_1 = H$ et $J_1 = K_1 = 1$

• A chaque front descendant de Q_1, on a : $Q_{2+} = \bar{Q}_2$

D'après la table caractéristique de la bascule JK : $Q_{2+} = \bar{Q}_2 \Rightarrow J_2 = K_2 = 1$ Sachant que la bascule 2 commute au front descendant, on peut écrire : $Q_1 = Ck_2$ (Ck_2 est l'horloge de la

bascule 2)

D'où : $Ck_2 = Q_1$ et $J_2 = K_2 = 1$

• A chaque front descendant de Q_2, on a : $Q_{3+} = \bar{Q}_3$

D'après la table caractéristique de la bascule JK : $Q_{3+} = \bar{Q}_3 \Rightarrow J_3 = K_3 = 1$

Sachant que la bascule 3 commute au front descendant, on peut écrire : $Q_2 = Ck_3$ (Ck_3 est l'horloge de la bascule 3)

D'où : $Ck_3 = Q_2$ et $J_3 = K_3 = 1$

Le schéma de ce compteur est visualisé sur la Figure (5.31). .

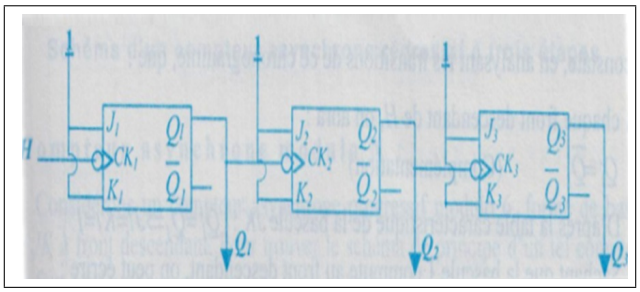


FIGURE 5.31 – Schéma d'un compteur asynchrone progressif

5.4 Conclusion

Dans ce chapitre nous avons présenté les principes et les règles de calcul de l'algèbre de Boole, puis la manipulation des fonctions logiques. Les circuits logiques combinatoires et séquentiels ont aussi été présentés par la la suite.

Ce cours est suivi par une série d'exercices, donnée à la fin de ce chapitre pour un meilleur entrainement sur le fonctionnement de ces systèmes.

5.5 Exercices

Exercice 1

Démontrez les relations suivantes en utilisant les règles de calculs de l'algèbre de Boole

$X + XY = X$

$X + \bar{X}Y = X + Y$

$X(X + Y) = X$

$X(\bar{X} + Y) = XY$

$XA + \bar{X}B + AB = XA + \bar{X}B$

Exercice 2

Trouvez les expressions logiques simplifiées des fonctions F et G définies par les tables de vérités illustrées sur la Figure (5.32) :

x	y	z	F
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	0
1	1	1	1

x	y	z	G
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

FIGURE 5.32 – Fonctions F et G

Exercice 3

Simplifiez algébriquement les expressions logiques suivantes :
$\bar{X}Y + XY$
$(X + Y)(X + \bar{X})$
$X + \bar{X}Y + \bar{X}\bar{Y}$

Exercice 4

Simplifiez les fonctions logiques suivantes en utilisant la méthode graphique :
$F1 = a * b + a * b * d + a * b * c * d$
$F2 = a * b * \bar{c} + \bar{a} * \bar{c} * d + \bar{a} * \bar{b} * \bar{d} + a * c + b * c * \bar{d}$
$F3 = a * b * \bar{c} + \bar{c} * d * a + a * \bar{b} * c * d$

Exercice 5

On considère le montage suivant :
1. Quelle est la fonction logique F réalisée par ce montage ?
2. Simplifiez la fonction F(utiliser le théorème de De Morgan).
3. Proposez un montage plus simple permettant de réaliser la fonction F.

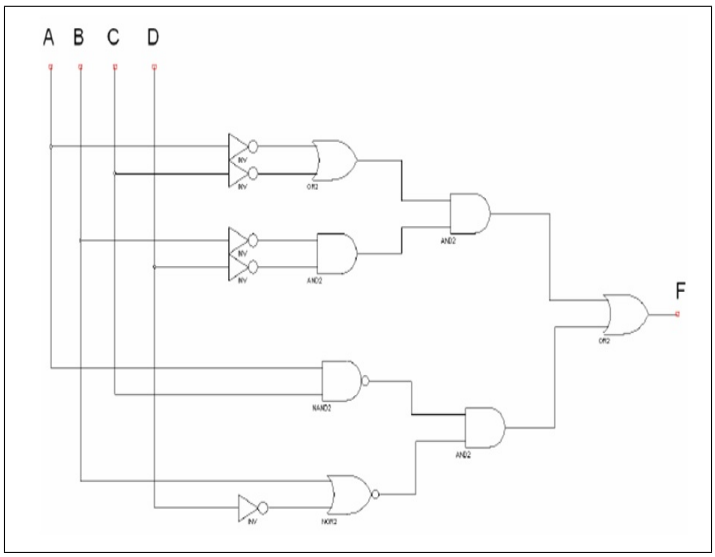


FIGURE 5.33 – Montage d'un circuit logique

Exercice 6

En binaire, un chiffre décimal (compris entre 0 et 9) est codé sur 4 bits a b c d dans l'ordre des poids décroissants. Ce chiffre est visualisé sur un afficheur 7 segments représenté sur la Figure (5.34).

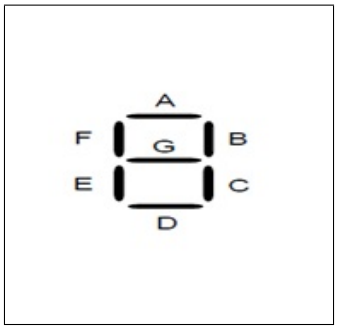


FIGURE 5.34 – Afficheur 7 Segments

Chaque segment est représenté par une lettre allant de A à G. Lors de l'affichage du chiffre 6 (respectivement 9) le segment (respectivement D) est Allumé.

1. Donner les expressions logiques, en fonction de a b c d , des fonctions logiques fA et fD valant 1 lorsque les segments A et D de l'afficheur sont allumés.

2. Simplifier les fonctions précédentes en utilisant des tables de Karnaugh.

3. Donner le schéma de la fonction fA avec un minimum de portes logiques.

Remarque : Le symbole Ø peut prendre indifféremment la valeur 0 ou 1 : on remplace donc par 1 uniquement ceux qui permettent d'augmenter le nombre des cases d'un regroupement

et ceux qui réduit le nombre des regroupements.

Exercice 7

Réalisez la fonction équivalence $f = \overline{x \oplus y} = \bar{x} * \bar{y} + x * y$ à l'aide de 4 portes logiques NOR uniquement.

Exercice 8

Construire un circuit logique capable de comparer deux nombre de 3 bits chacun ($A_0A_1A_2$) et ($B_0B_1B_2$) En sortie, on voudrai avoir :

— 1 si $A_0A_1A_2 = B_0B_1B_2$.

— 0 sinon.

Exercice 9

On considère le schéma de la Figure (5.35) réalisé avec des bascules RS asynchrones à base de portes NOR.

1. Donner les expressions logiques des entrées R1, S1, R2 et S2 des deux bascules.
2. Rappeler la table de vérité et le fonctionnement d'une bascule RS asynchrone à base de portes NOR.
3. Que peut-on dire des bascules 1 et 2 lorsque E = 0 ? Même question lorsque E = 1.

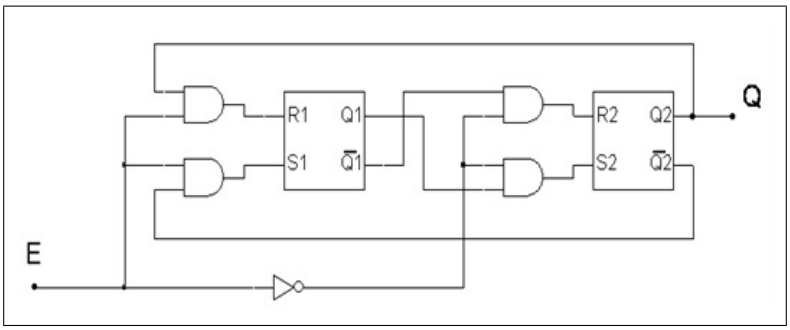


FIGURE 5.35 – Circuit réalisé avec des bascules RS asynchrones

Exercice 10

Le montage de la Figure (5.36) représente une bascule réalisée à partir de portes NAND.

1. Que valent les sorties Q et $\bar{Q}$ dans les deux situations suivantes : R = 1, S = 0 et R = 0, S = 1 ?

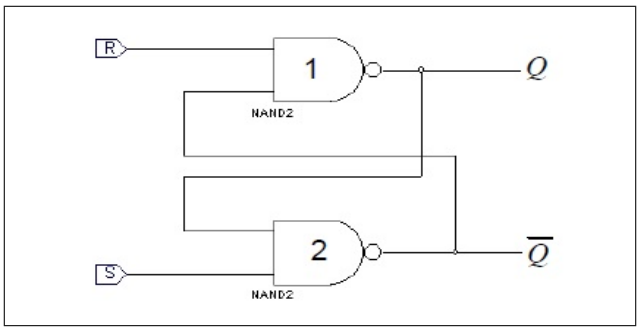


FIGURE 5.36 – Bascule réalisée à partir de portes NAND

2. Le circuit est dans un état définit par R = 1, S = 0, Q = 0, $\bar{Q}$ =1. Quelle est l'évolution des sorties lorsque l'entrée S commute vers le niveau haut (R = 1, S :0 → 1) ?
3. Le circuit est dans un état définit par R = 0, S = 1, Q = 1, $\bar{Q}$ = 0 . Quelle est l'évolution des sorties lorsque l'entrée R commute vers le niveau haut (S = 1, R :0 → 1) ?

Exercice 11

Soit le circuit de la Figure (5.37), composé de bascules type JK à front montant :

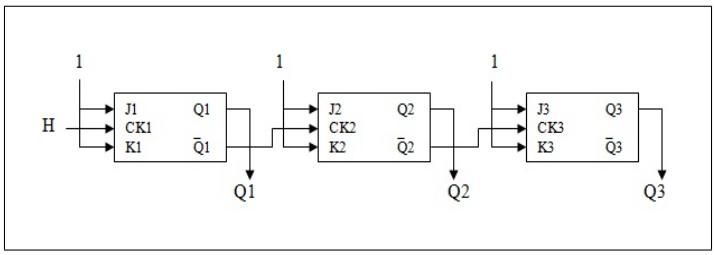


FIGURE 5.37 – Circuit composé de bascules JK à front montant

1. Décrire le comportement des différentes bascules de ce circuit.
2. Le circuit est-il synchrone ou asynchrone ? pourquoi ?
3. Donner le chronogramme de fonctionnement de ce circuit.
4. En déduire la table des transitions après chaque impulsion d'horloge.
5. Que fait ce circuit ?

Exercice 12

On considère une bascule JK' qui n'est autre que la bascule JK avec un inverseur entre l'entrée interne K et l'entrée K', voir le schéma de la Figure (5.38).

1. Établir la table caractéristique de cette bascule.

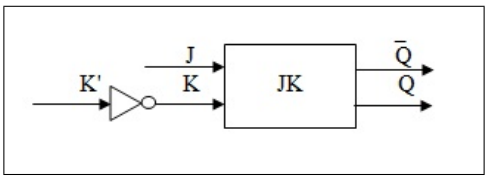


FIGURE 5.38 – Bascule JK'

2. Exprimer l'état futur de la bascule Qt+1 en fonction des variables d'entrée (J et K') et de la variable de sortie Qt
3. Montrer qu'en joignant les deux entrées J et K', on obtient une bascule D.

Exercice 13

Faire la synthèse d'un compteur synchrone progressif modulo 5, formé de trois bascules types D à front montant.

Bibliographie

[1] S. Ait Aoudia. *Architecture des systèmes informatiques.* Office des Publications Universitaires, ISBN 996100924-X, 2005.

[2] T. Andrew. *Architectures des ordinateurs.* Pearson Education, 2005.

[3] J. P. Hayes. *Computer architecture and organization.* McGraw-Hill, Inc, 2002.

[4] J. L. Hennessy and D. A. Patterson. *Computer architecture : a quantitative approach.* Elsevier, 2011.

[5] K. Hwang and A. Faye. *Computer architecture and parallel processing.* Pearson Education, 1984.

[6] M. Koudil and helifatiS. L. K. *Structure des ordinateurs, Autour du processeur.* Office des Publications Universitaires, ISBN 9961009215, 2005.

[7] E. Lazard. *Architecture de l'ordinateur.* Pearson Education, 2006.

[8] Y. Ligier and P. Zanella. *Architecture et technologie des ordinateurs.* Dunod, 1995.

[9] W. Stallings. *Organisation et architecture de l'ordinateur.* Pearson Education, 2003.