**République Algérienne Démocratique et Populaire**
**Ministère de lôEnseignement Supérieur et de la Recherche Scientifique**
**Université A.MIRA-BEJAIA**


Faculté des sciences exactes
Département Informatique

# Cours: architecture des ordinateurs

Dr Tighidet soraya épouse Aloui

# Architecture des Ordinateurs



**Cours : Architecture des ordinateurs**
Support destiné aux étudiants de L1, L2 en Informatique.

**Dr. Soraya TIGHIDET**
Enseignante au Département d'Informatique
Faculté des Sciences Exactes
Université Abderrahmane Mira de Bejaia

# Avant-propos

Ce polycopie de cours d'architecture des ordinateurs est destinée aux étudiants d'Informatique en Lience1 et Licence2.

Dans ce manuscrit, il est abordé :

Les systèmes de numération et code

Les Opérateur logique et Algèbre de Boole

La logique combinatoire

Vue d'ensemble de l'ordinateur

Ce cours vise à introduire les concepts de bases pour comprendre le fonctionnement d'un ordinateur, ses composants de base, la logique de traitement des données, instructions,etc.

# Chapitre 1

# Systèmes de numération et codes

Le système de numération binaire est le plus important de ceux utilisés dans les circuits numériques. Il est le seul que ces circuits soit capable d'utiliser .il ne faut pour autant pas négliger l'importance des autres systèmes. Le système décimal revêt de l'importance en raison de son utilisation universelle pour représenter les grandeurs du monde courant. De ce fait, il faudra parfois convertir des valeurs décimales en valeurs binaires avant de pouvoir les traiter dans un circuit numérique. Par exemple, lorsque vous composez un nombre décimal sur votre calculatrice ou sur le clavier de votre ordinateur, les circuits internes convertissent ce nombre décimal en une valeur binaire.

De même, il y aura des situations ou des valeurs binaires données par un circuit numérique devront être converties en valeurs décimales pour qu'on puisse les lire. Par exemple, votre calculatrice (ou votre ordinateur) calcule la réponse a un problème au moyen du système binaire puis convertit ces réponses en des valeurs décimales avant de les afficher.

Nous connaissons les systèmes binaire et décimal, étudions maintenant deux autres systèmes de numération très répondus dans les circuits numériques. Il s'agit des systèmes de numération octale base de 8 et hexadécimales bases de 16 qui servent tous les deux au même but, soit celui de constituer un outil efficace pour représenter de gros nombres binaires.

Comme nous le verrons, ces systèmes de numération ont l'avantage d'exprimer les nombres de façon que leur conversion en binaire, et vice versa , soit très facile.

Dans un système numérique, il peut arriver que trois ou quatre de ces systèmes de numération cohabitent, d'où l'importance de pouvoir convertir un système dans un autre . le présent chapitre se propose de vous montrer comment effectuer de telles conversions. Certaines de ces conversions ne seront pas appliquées immédiatement dans l'étude des circuits numériques, mais vous constaterez au moment de l'étude des microprocesseurs que c'est une connaissance dont vous avez besoin.

Ce chapitre se veut également une introduction a certains des codes binaires utilisés pour représenter divers genres d'information. Ces codes agencent les 0 et les 1 de manière différente du système binaire.

## 1-1 Représentation des nombres

Le nombre de symboles utilisés caractérise le numéro de la base.

EX :

- En base 10, nous avons les 10 symboles (0.1…..,9)
- En base 2, nous avons les 2 symboles (0,1)
- En base 3, nous avons les 3symboles (0,1,2)
- En base 16, nous avons besoin de 16 symboles,
- Nous utiliserons les 10 chiffres plus les lettres de A a F , soit
  0,1,2,3,4,5,6,7,8,9 , A,B,C,D,E,F

**N.B :** il faut remarquer que le choix de symboles chiffres facilite grandement les choses, cette facilité découlant de notre grande habitude du système décimal ( 10 symboles).

Le poids d'un chiffre dépend de sa position dans le nombre. Nous parlons de numération de position, soit :

Un nombre dans une base ''b'' entière positive s'écrit :

$N_b = a_n\, a_{n-1} \ldots a_1 a_0 , a_{-1} \ldots a_{-m}$ (1)

Ce qui correspond aux opérations :

$N_B = a_n . b^n + a_{n-1} . b^{n-1} + \ldots + a_1 . b^1 + a_0 . b^0 + a_{-1} . b^{-1} + \ldots + a_{-m} . b^{-m}$ (2)

L'indice de b indique la base dans laquelle le nombre est calculé.

**N.B :** la formule (2) donne N dans la base dans laquelle on effectue les opérations (ici la base B) . pour nous, ce sera généralement la base 10.

### 1-2 Conversion Binaire – Décimal

Le système de numération binaire est un système de numération de position ou le poids de chaque bit est un multiple de puissance de 2 (base) . le affecté d'un certain poids qui dépend de son rang par rapport au bit de poids le plus faible. Ainsi tout nombre binaire peut être transformé en son équivalent décimal simplement en additionnant les poids des diverses positions ou se trouve une valeur 1. Voici une illustration :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | | 0 | | 1 | | 1 | | | (binaire) |
| $1.2^4$ | + | $1.2^3$ | + | $0.2^2$ | + | $1.2^1$ | + | $1.2^0$ | | | |
| 16 | + | 8 | + | 0 | + | 2 | + | 1 | = | $27_{10}$ | ( décimal ) |

Exemple 2-1 : conversion de nombre binaire en décimal

Voyons un autre exemple pour un nombre ayant un plus grand nombre de bits.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | binaire |
| $1.2^{7}$ + | 0.26 | + 1.25 + | $1.2^{4}$ | + | $0.2^{3}$ | + | $1.2^{2}$ + | $0.2^{1}$ + $1.2^{0}$ |
| 128 + | 0 | + 32 + | 16 | + | 0 | + | 4 + | 0 + 1 $=181_{10}$ décimal |

Exemple 2-2 : autre exemple de conversion

Vous remarquez que la méthode consiste à trouver les poids (les puissances de 2) pour chaque position du nombre ou il y a 1, puis à additionner le tout. Remarquez que le bit de poids le plus fort a un poids de $2^7$ même s'il s'agit du huitième bit ; il en est ainsi parce que le bit de poids le plus faible est le premier bit et que son poids est toujours $2^0$.

## 1-3 Conversion Décimal - Binaire

Nous pouvons essayer d'appliquer la même démarche que précédemment. Nous allons exprimer le nombre décimal en binaire en utilisant la formule de représentation de nombre en numération de position. L'exemple 2-3 nous montre la démarche pour le nombre 158.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 5 | | 8 | | | décimal |
| $1.10^{2}$ | + | $5.10^{1}$ | + | $8.10^{0}$ | | | num position |
| | | | | | | | en décimal |
| $0001.(1010)^{10}$ | + | 0101 | . | $(1010)^{01}$ | + | 1000 . $(1010)^{00}$ $= ???_2$ | binaire |

Exemple 2-3 : conversion décimal – binaire avec représentation de numération de position

L'exemple 2-3 n'est pas utilisable pour nous. Nous devrions être capables, pour cela, de faire le calcul en base 2 ! Cette méthode n'est pas utilisable pour nous car nous savons faire les calculs uniquement en décimal.

Il existe deux façons de convertir un nombre décimal en son équivalent binaire. Une méthode qui convient bien aux petits nombres est une démarche qui est basée sur la numération de position en binaire. Le nombre décimal est simplement exprimé comme une somme de puissance de 2, puis on inscrit des 1 et des 0 vis-à-vis des positions binaires appropriées. Voici un exemple :

$45_{10} = 32+8+4+1=2^5+0+2^3+2^2+0+2^0 =1\ 0\ 1\ 1\ 0\ 1_2$

Notons qu'il y a un 0 vis-à-vis des positions $2^1$ et $2^4$ , puisque ces positions ne sont pas utilisées pour trouver la somme en question. Il s'agit d'une méthode par essais successifs (tâtonnement).

### 1-3.1 Conversion de la partie entière

L'autre méthode convient mieux aux grands nombres décimaux, il s'agit de répéter la division par 2. La partie entière d'un nombre peut s'exprimer comme suit :

$$N_B = a_n . b^n + a_{n-1} . b^{n-1} + \dots + a_1 b^1 + a_0 . b^0$$

Si nous divisons NB par la base b , nous obtenons l'expression suivante :

— = ( . + . + …+ . ) ... . ... ( )

$A_{0\ apparait}$ comme le reste de la division de Nentier par b ; $a_1$ est le reste de la division du quotient par b ; $a_2$ est le reste de la division du nouveau quotient par b. on opère donc par divisions successives par b.

Cette méthode de conversion est illustrée ci-après pour le nombre $25_{10.}$

Nous utilisons des divisions répétitives par 2 du nombre décimal à convertir. A chaque division nous obtenons un quotient et un reste. Nous devons effectuer les divisions jusqu'à obtenir un quotient nul. Il est important de noter que le nombre binaire résultant s'obtient en écrivant le premier reste a la position du bit de poids le plus faible (LSB) et le dernier reste a la position du bit de poids le plus fort (MSB).

---

25/2 = 12 reste 1 poids faible (LSB)

12/2=6 reste 0

6/2=3 reste 0

3/2=1 reste 1

½=0 reste 1 poids fort (MSB)

$25_{10} = 1\ 1\ 0\ 0\ 1_2$

---

Exemple 2-4 : conversion de 25 décimal en binaire

### 1-3.2 Conversion de la partie fractionnaire

La conversion de la partie fractionnaire s'obtient par l'operateur inverse, soit la multiplication. La partie fractionnaire d'un nombre peut s'exprimer comme suit :

$N_B = a_{-1}.b^{-1} + a_{-2}.b^{-2} + \dots + a_{-m+1}.b^{-m+1} + a_{-m}.b^{-m}$

Si nous multiplions $N_B$ par la base b, nous obtenons l'expression suivante :

b. $N_B = a_{-1}\dots$ ()

$a_{-1}$ apparaît comme la partie entière de la multiplication fractionnaire par b ; $a_{-2}$ est la partie entière de la multiplication par b du reste ; $a_{-3}$ est la partie entière de la multiplication du nouveau reste par b. On opère donc par multiplication successive par b.

Cette méthode de conversion est illustrée ci-après pour le nombre $0{,}375_{10}$. Nous utilisons des multiplications successives par 2 du nombre décimal à convertir. A chaque multiplication nous obtenons une partie entière et un reste. Il est important de noter que le nombre binaire résultant s'obtient en écrivant le premier chiffre à la position du bit de poids le plus fort (MSB).

| | | | | | |
|---|---|---|---|---|---|
| 0.375 | x 2 | =0.75 | Partie entière = 0 | Poids fort (MSB) | Reste = 0.75 |
| 0.75 | x 2 | =1.5 | Partie entière = 1 | | Reste = 0.5 |
| 0.5 | x 2 | =1.0 | Partie entière = 1 | | Reste = 0 |

$N_{10}$ =0.375 correspond à $N_2$ =0.011

*Exemple 2-5 : Conversion du nombre fractionnaire décimal 0.375 en binaire*

On peut remarquer qu'un nombre fini dans une base peut conduire à une suite infinie dans une autre.

### 1- 4 Le système de numération octal

Le système de numération octal a comme base huit ; ce qui signifie qu'il comprend huit symboles possibles ; soit0, 2, 3,4 ,5 ,6et 7.ainsi ; chaque chiffre dans un nombre octal q un nombre octal a une valeur comprise entre 0et7. Voici les poids de chacune des positions d'un nombre octal.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ……. | $8^3$ | $8^2$ | $8^1$ | . | $8^0$ | $8^{-1}$ | $8^{-2}$ | $8^{-3}$ | …….. |

#### 1- 4.1 Conversion Octal-Décimal

On convertit un nombre octal en son équivalent décimal en multipliant chaque chiffre octal par son poids positionne. Voici un exemple :

$372_8=3.8^2+7.8^1+2.8^0$

$= 3.64+7.8+2.1$

$= 250_{10}$

Exemple2-6 conversion octal-décimal

1-4-2 conversion octal-décimal

Il est possible de convertir un nombre décimal entier en son équivalent octal en employant la répétition de divisions, la même qu'on a utilisée pour la conversion décimal-binaire, mais cette fois-ci divisant par 8 plutôt que par2.voici un exemple :

266/8 =33reste2

33/8 =4reste1

4/8 =0reste4

$266_6 = 412_8$

---

Exemple2-7 conversion octal-décimal

Notez que le premier reste devient le chiffre de poids plus faible du nombre octal et que le dernier reste devient le chiffre de poids le plus fort.

Si on utilise une calculatrice pour faire les divisions, on aura comme résultatun nombre avec une partie fractionnaire plutôt qu'un reste .on calcule toute foi les reste en multipliant la fraction décimale par 8. Par exemple, avec la calculatrice, la réponse de la division 266/8, est33, 25.enmultipliant la partie décimale par 8, ont trouvé un reste de 0 ,25x8=1

**1-4-3 Conversion Octal-Binaire**

Le principal avantage du système de numération octal réside dans la facilite avec laquelle il est possible de passer d'un nombre octal a un nombre binaire. Cette conversion s'effectue en transformant en chaque chiffre du nombrer octal en son équivalent binaire de trois chiffres. Voyez dans le tableau ci-dessous les huit symboles octaux exprimés en binaire.

| Chiffre octal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Equivalent binaire | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

Au moyen de ce tableau, tout nombre octal est converti en binaire par la transformation de chacun des chiffres. Par exemple, la conversion de 4728 va comme suit :

4 7 2

100 111 010

Donc le nombre octal $472_8$ est équivalent au nombre binaire 100111010.

**1-4.4 Conversion Binaire –Octal**

La conversion d'un nombre binaire en un nombre octal est simplement l'inverse de la marche à suivre précédente. Il suffit de faire avec le nombre binaire des groupes de trois bits en partant du chiffre de poids le plus faible , puis de convertir ces triplets en leur équivalent octal ( voir tableau 2 1). à titre d'illustration, convertissons $100111010_2$ en octal.

011 010 110

$472_8$

Parfois, il arrivera que le nombre ne forme pas un nombre juste de groupes de trois .dans ce cas, on pourra a jouter un ou deux zéros à gauche du bit de poids le plus fort pour former le dernier triple (si on lit de droite à gauche). Voici une illustration de ceci avec le nombre binaire 11010110.

011 010 110

3 2 $6_8$

Notez l'ajout d'un zéro à gauche du bit de poids le plus fort pour obtenir un nombre juste de triplets.

**1-5 Système de numération hexadécimal**

---

Le système hexadécimal a comme base 16, ce qui symboles de chiffres possibles, qui, dans ce cas sont les dix chiffres 0 à 9 plus les lettres majuscules A, B, C, D, F et F. le tableau 2-1expose les rapports entre les systèmes hexadécimal, décimal et binaire. Remarquez que chaque chiffre hexadécimal a comme équivalent binaire un groupe de quatre bits.

Il ne faut surtout pas les chiffres hexadécimal A à F correspondent aux valeurs décimales 10 à 5.

| hexadécimal | Décimal | Binaire |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 111 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

Tableau 2-1 : rapport entre hexadécimal décimal et binaire

La représentation hexadécimale est principalement utilisée pour représenter un nombre binaire sous forme plus compact. Un nombre en hexadécimal comprend 4foi moins de chiffres !

**1-5-1 Conversion Hexadécimal-Décimal**

Un nombre hexadécimal peut être converti en son équivalentdécimal en exploitant le fait qu'à chaque position d'un chiffre hexadécimal est attribué un poids ; dans ce cas le nombre 16 élevé une certaine puissance. Le chiffre de poids le plus faible de$16^0$=1, le chiffre immédiatement à gauche a un poids de $16^1$=16, l'autre chiffre immédiatement à gauche, un poids de $19^2$=256, et de suite. Voici un exemple sur la façon dont fonctionne ce processus de conversion.

---

$356_{15}$ =$3,16^2$ +$5.16^1$+$6.16^0$

=768+80+6

=$854_{10}$

$2AF_{16}$=$2016^2$+$10.16^1$+$15.16^0$

=512+160+15

=$687^{10}$

---

Exemple 2-8 :1-5-1 conversion hexadécimal-décimal

**1-5-2 Conversion Décimal-Hexadécimal**

Vous vous rappelez peut-être que pour la conversion décimal-binaire nous avons eu recours à la répétition de division par 8, que la conversion décimal en un nombre hexadécimal, il faut procéder de la même façon, mais cette fois en divisant par 16.les exemples qui suivent illustrent cette technique. Remarquez comment les restes des divisions deviennent les chiffres du nombre hexadécimal ; de plus, voyez, comment les restes supérieurs à 9 sont exprimés au moyen des lettres A à Exemple, conversion de (423) 10 en hexadécimal :

---

423/16 =26 reste 7

26/16 =1 reste 10

1/16 = 0reste 1

$423_{10}$=$1A7_{16}$

---

Exemple 2-9 : conversiondécimal-hexadécimal

**1-5.3 Conversion Hexadécimal Binaire**

Comme le système de numération octal, le système de numération hexadécimal se veut une façon abrégée de représenter les nombres binaires. La conversion d'un nombre hexadécimal en un nombre binaire ne pose vraiment pas de difficulté, puisque chaque chiffre hexadécimal est remplacé par son équivalent binaire de 4bits (tableau 2 2). Voice un exemple avec 9F2016.

---

$9F2_{16}$ =9 F 2

1001 1111 0010

$=100111110010_2$

Exemple 2-10 : conversiondécimal-hexadécimal

**1-5.4 Conversion Binaire-Hexadécimal**

Cette conversion est tout simplement l'inverse de la précédente. Le nombre binaire est divisé en groupes de quatre bits, puis on substitue à chaque groupe son chiffre hexadécimal équivalent. Au besoin, on ajoute des zéros à gauche pour obtenir un dernier groupe de 4 bit.

1110100110 = 0011 1010 0110

3 A 6

$= 3A6_{16}$

Exemple 2-10 : conversion binaire-hexadécimal

Pour passer d'un nombre hexadécimal à son équivalent binaire, il faut connaitre la suite des nombres binaires de quatre (0000à1111) ainsi que le nombre correspondant en hexadécimal. Des que cette correspondant ce devient un réflexe automatique, les conversions se font rapidement sans calculs. C'est ce qui explique pourquoi le système hexadécimal est si pratique pour représenter de grands nombres binaires.

**1-5-5 Comptage hexadécimal**

Lorsque l'on compte selon le système de numération hexadécimal, la valeur dans une position du nombre croit par pas de 1 depuis 0jusqu' à F. quand le chiffre dans une position est F, le chiffre suivant dans cette position est 0 et le chiffre immédiatement à gauche est augmenté de1.Cest ce qu'on voit dans les suites de nombres hexadécimal suivantes

a.38, 39,3A ,3B, 3C, 3D, 3E, 3F, 40, 41,42

b.6F8, 6F9, 6FA, 6FB, 6FC, 6FE, 6FE, 6FF, 700

Notez que le chiffre qui suit 9dans une position est A

**1-5.6 Utilité du système hexadécimal**

La facilite avec laquelle se font les conversions entre les systèmes binaire et hexadécimal explique pourquoi le système hexadécimal est devenu une façon abrégée d'exprimer de grands binaires. Dans un ordinateur, il n'est pas rare de retrouver des nombres binaires ayant jusqu' à 64 bits de longueur. Ces nombres binaires, comme nous le verrons, ne sont pas toujours des valeurs numériques. Mais peuvent correspondre à un certain code représentant des renseignements non numérique .dans un ordinateur, un nombre binaire peut être :1) un vrai nombre ;2) un nombre

correspondant à un emplacement (adresse) en mémoire ; 3) un code d'instruction ;4) un code correspondant à un caractère alphabétique ou non numérique ; ou5) un groupe de bits indiquant la situation dans laquelle se trouvent des dispositions internes et externes de l'ordinateur.

Quand on doit travailler avec beaucoup de nombres binaires très longs, il est plus commode et plus rapide d'écrire ces nombres en hexadécimal plutôt qu'en binaire toutefois, ne perdez pas de vue que les circuits et les systèmes numérique fonctionnent exclusivement en binaire et que c'est par pur souci de commodité pour les opérateurs qu'on emploie la notation hexadécimal.

## 1-6 Code BCD, Soit Binary Coded Décimal

L'action de faire correspondre à des nombres, des lettres des mots un groupe spécial de symboles s'appelle codage et le groupe de symboles un code. Un des codes que vous connaissez peut-être le mieux est le code morse dans lequel on utilise une série de points et de traits pour représenter les lettres de l'alphabet.

Nous avons vu que tout nombres, décimal pouvait être converti en son équivalent binaire. Il est possible de considère le groupe de 0 et de 1 du nombre binaire comme un code qui représente le nombre décimal .quand on fait correspondre à un nombre décimal son équivalent binaire, on dit qu'on fait un codage binaire pur.

Les circuits numériques fonctionnent avec des nombres binaires exprimés sous forme autre durant leurs opérations internes, malgré que le monde extérieur soit un monde décimal. Cela implique qu'il faut effectuer fréquemment des conversions entre les systèmes binaire et décimal. Nous savons que pour les grands nombres, les conversions de ce genre peuvent être longues peuvent être longues et laborieuses. C'est la raison pour laquelle on utilise dans certaines situations un codage des nombresdécimaux qui combine certaines caractéristiques de système décimal.

Le BCD s'appelle en français code décimal codé binaire (CDB). Si on représente chaque chiffre d'un nombre décimal par son équivalent binaire, on obtient le code dit décimal codé binaire (abrégé dans le reste du texte par BCD). Comme le plus élevé des chiffres décimaux est 9, il faut donc4 bits pour coder les chiffres.

Illustrons le code BCD en prenant le nombre décimal 874 et en changeant chaque chiffre pour son équivalent binaire ; cela donne

| | | | |
|---|---|---|---|
| 8 | 7 | 4 | décimal |
| 100 | 0111 | 0100 | BCD |

De nouveau, on voit que chaque chiffre a été converti en son équivalent binaire pur. Notez qu'on fait toujours 4 bits. Evidemment, seuls les groupes binaire 0000 à 1011sont utilisés. Le code BCD ne fait pas usage des groupes 1010, 1011, 1101, 1110 et1111. Autrement dit, seuls dix des 16 combinaisons des 4 bits sont utilisant .si l'une "inadmissibles'' le code apparait dans une machine utilisant le code BCD, c'est généralement le signe qu'une erreur s'est produite.

### 1-6.1 Comparaison entre code BCD et nombre binaire

Il importe de bien réaliser que le code BCD n'est pas autre système de numération comme les systèmes octal, décimal ou hexadécimal. En fait, ce code est le système décimal dont on a converti les chiffres en leur équivalent binaire. En outre, il faut bien comprendre qu'un nombre BCD n'est pas un nombre binaire pur. Quand on code selon le système binaire pur, on prend le nombredécimal dans son intégralité et on le convertit en binaire, sans le fractionner, qui individuel qui est remplacé par son équivalent BCD :

$137_{10}=10001001_2$ (binaire)

$137_{10}=00010010111$ (BCD)

Le code BCD nécessite 12bits pour représenter 137tanis que le nombre binaire pur n'a besoin que 8 bits.il faut plus de bits en BCD qu'en binaire pur pour représenter les nombres décimaux de plus d'un chiffre. Comme vous le savez, il en est ainsi pace que le code BCD n'utilise pas toutes les combinaisons possibles de groupes de 4 bits ; c'est donc un code peu efficace.

Le principal avantage du code BCD provient de la facilité relative avec laquelle on passe de ce code à un nombre décimal, et vice versa. Il ne faut retenir que les groupes de 4 bits des chiffres 0 à 9. C'est un avantage non négligeable du point de vue du matériel, puisque dans un système numérique ce que sont des circuits logique qui ont la charge d'effectuer ces conversions.

### 6-1.2 Conversion BCD-Binaire

Une conversion est nécessaire pour convertir un nombre exprimé en BCD en binaire.la seule possibilité est de passer par la valeur décimal.

Voici la démarche à suivre :

$N_{BCD}$ l'exprime en décimal convertir en binaire (voir§ 2-3 ? page 7)

Nous verrons plus tard comment une telle conversion est réalisée dans un système numérique qui fait tous les calculs en binaire !

### 1-6.3 Conversion Binaire- BCD

unconversion d'une valeur binaire en BCD demande de passer aussi par la valeur décimale. Voici la démarche à suivre :

$N_2$ convertir en décimal(voir §2-2, page7) l'exprimé en BCD

## 1-7 Récapitulatif des différents codes

Nous donnerons un tableau des principaux codes. Il faut toutefois mentionner le code GRAY ou binaire réfléchi. Ce code présente l'avantage qu'l n'y a qu'un seul bit qui change à la foi .il offre des lors de multiples utilisation.

| décimal | Binaire | octal | hexadécimal | Gray ou BR | Excédent3 | AIKEN |
|---|---|---|---|---|---|---|
| 00 | 0000 | 00 | 0 | 0000 | 0011 | 0000 |
| 01 | 0001 | 01 | 1 | 0001 | 0100 | 0001 |
| 02 | 0010 | 02 | 2 | 0011 | 0101 | 0010 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 03 | 0011 | 03 | 3 | 0010 | 0110 | 0100 |
| 04 | 0100 | 04 | 4 | 0110 | 0111 | 1011 |
| 05 | 0101 | 05 | 5 | 0111 | 0101 | 1100 |
| 06 | 0110 | 06 | 6 | 0101 | 0100 | 1101 |
| 07 | 0111 | 07 | 7 | 0100 | 1100 | 1111 |
| 08 | 1000 | 08 | 8 | 1101 | Sur deux | décades |
| 09 | 1001 | 09 | 9 | 1111 | Sur deux | décades |
| 10 | 1010 | A | A | 1110 | Sur deux | décades |
| 11 | 1011 | B | B | 1010 | Sur deux | décades |
| 12 | 1100 | C | C | 1011 | Sur deux | décades |
| 13 | 1101 | D | D | 1011 | Sur deux | décades |
| 14 | 1110 | E | E | 1001 | Sur deux | décades |
| 15 | 1111 | F | F | 1000 | Sur deux | décades |

Tableau2-2 : table des codes

Les codes excédent 3et AIKEN ne sont pratiquement plus utilisés.

## 1-8 Les codes Alphanumériques

Un ordinateur ne serait pas d'une bien grande utilité s'il était incapable de traiter l'information non numérique. On veut dire par-là qu'un ordinateur doit reconnaitre des codes qui correspondent à des nombre, des lettres, des signes de ponctuation et des caractères spéciaux. Les codes de ce genre sont dit alphanumériques. Un ensemble de caractères complet doit renfermer les 26 lettre minuscules, les 26 lettres majuscules, les dix chiffres, 7signes de ponctuation et entre 20 à40 caractères spéciaux comme+, #, %, on peut conclure qu'un code alphanumérique reproduit tous les caractères et les diverses fonctions que l'on retrouve sur un clavier standard de machine à écrire ou d'ordinateur.

### 1-8.1 Code ASCII

Le code alphanumérique le plus répandu est le code ASCII (American standard Code for Information Interchange) ; on le retrouve dans la majorité des micro-ordinateurs et des mini-ordinateurs et dans beaucoup de gros ordinateurs. Le code ASCII (prononce''aski'') standard est un code sur 7bit, on peut donc représenter grâce à lui $2^7$=128 éléments codés. C'est amplement suffisant pour reproduire toutes courantes d'un clavier et les fonctions de contrôle comme (RETOUR) et(INTERLIGE).le tableau 2-3 contient le code ASCII standard. Dans ce dernier, en plus du groupe binaire de chaque caractère, on a donné l'équivalent hexadécimal.

| 0...3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | NUL | DLE | Space | 0 | Nat | P | , | p |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 2 | STX | DC2 | “ | 2 | B | R | b | r |
| 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 6 | ACK | SYN | § | 6 | F | V | f | v |
| 7 | BEL | ETB | , | 7 | G | WW | g | w |
| 8 | BS | CAN | ( | 8 | H | X | h | x |
| 9 | HT | EM | ) | 9 | I | Y | i | y |
| A | LF | SUB | * | : | J | Z | j | z |
| B | VT | ESC | + | ; | K | Nat | k | Nat |
| C | FF | FS | , | < | L | Nat | l | Nat |
| D | CR | GS | - | = | M | Nat | m | Nat |
| E | SO | RS | . | < | N | ^ | n | ~ |
| F | SI | US | / | ? | O | _ | o | DEL |

Tableau2-3 : liste partielle de code ASCII

Légende :

| | | | |
|---|---|---|---|
| SOH | Début d’en tête | US | Séparateur de sous-article |
| STX | Début de texte | RS | Séparateur d’article |
| ETX | Fin de texte | GS | Séparateur de groupe |
| EOT | Fin de transmission | RS | Séparateur de fichiers |
| ENQ | Demande | | |
| ACK | Accusé de réception | BEL | Sonnerie |
| DLE | Echappement de transmission | SO | Hors code |
| NAK | Accusé de réception négatif | SI | En code |
| SYN | Synchronisation | CAN | Annulation |
| ETB | Fin de bloc de transmission | EM | Fin de support |
| BS | Espace arrière | SUB | substitution |
| HT | Tabulateur horizontal | ESC | Echappement |
| LF | Interligne | SP | Espace |
| CR | Retour de chariot | NUL | Nul |
| DC1 | Marche lecteur | DEL | Oblitération |

DC2 Embrayage perforateur

DC3 Arrêt lecteur

DC4 Débrayage perforateur Nat Usage national

Le code ASCII standard n'inclus pas des caractères comme les lettres avec les accents é, è ; à etc. Le nombre de bits utilisé a donc été augmenté à 8. Nous disposons ainsi 256élément codés .l'extension du code ASII n'est pas standard. En français nous avons besoin des lettres avec de accents comme é, è, à,… En allemand, il y a d'autres caractères comme ë, à …. Il existe donc autant de variantes du code ASCII étendu sur 8bits qu'il y a de pays voir de région !

## SERIE DE TD N°01

<<je vous le dis : vous n'avez le droit d'éviter un effort qu'un autre effort, car vous devez grandir. >> Antoine de Saint-Exupéry

<<je n'ai jamais rencontré d'homme si ignorant qu'il n'eut quelque chose à m'apprendre>>

**Galilée**

Combien de fois abandonnons-nous notre

chemin, attirés par l'éclat trompeur du chemin d'à coté

**Paulo Coelho**

**Exercice n° 1 :**effectuer les conversions suivantes

a. /$(10110,11)_{2=}$(-) $_{10}$b. /$(1254,1)_8$= (-10) c. / (F5B, A) $_{16}$=(-) $_{10}$

d./$(52,38)_{10}$ =(-)$_2$ e./$(225)_{10}$= (-)$_8$

# Chapitre 2

# Opérateur logique et algèbre de Boole

---

## II.1 Systèmes binaires :

Actuellement, alors que les ordinateurs analogiques restent du domaine de la recherche, les informations traitées par les systèmes informatiques sont codées sous forme binaire. Un système binaire (signal, circuit etc….) est un système qui ne peut exister que dans deux états autorisés. Le circuit de la figure 1 est un exemple plus que simpliste de circuit binaire : selon que l'interrupteur S est ouvert ou fermé la tension Vo ne peut être égale qu'à +5V ou 0V .

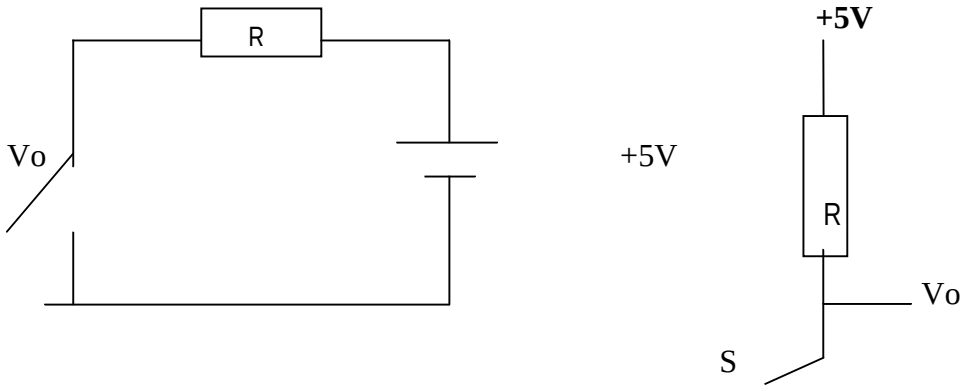



Figure 1.

La réalité technique est un peu plus complexe avec des interrupteurs commandés réalisés par des transistors. Diverses notations peuvent être utilisées pour représenter ces deux états :

Numérique : 1 et 0 (bit : binary digit)

Logique : vrais et faux (true et false)

Oui ou non (yes et no)

Physique : ouvert et fermé

ON et OFF

Haut et bas (HI et LO, H et L, H et B)

Pour étudier les fonctions de variables binaires on utilise une algèbre développée au XIX$^{éme}$ siècle par un mathématicien anglais : Georges Boole. Dans ce chapitre nous nous proposons de présenter les fonctions de base de l'algèbre booléenne ainsi que leurs représentations symboliques en électronique. Nous rappellerons également,

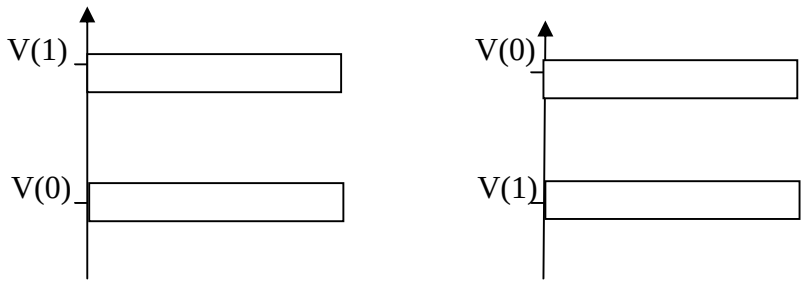
sans prétendre à la rigueur mathématique, les quelques notions élémentaires nécessaires à l'étude des circuits électroniques.

L'algèbre de Boole concerne la logique des systèmes binaires. Une variable booléenne ne peut prendre que deux valeurs possibles 0 ou 1. En électronique les deux états d'une telle variable peuvent être associés à deux niveaux de tension : V(0) et V(1) pour les états 0 et 1 respectivement. On distingue les logiques positive et négative selon que V(1)> V(0) ou V(1) < V(0). Ce que nous pouvons résumer dans la table suivante donnant la signification logique des niveaux physiques :

| Niveau | Logique positive | Logique négative |
|---|---|---|
| **H** | **1** | **0** |
| **L** | **0** | **1** |

**Table 1**

En pratique un niveau est défini par un domaine en tension ou en courant. Par exemple en technologie TTL, un niveau sera dit haut s'il est compris entre +2V et +5V et un niveau sera bas s'il est inférieur à +0.8V . Dans la plage intermédiaire, l'état est indéterminé. Si les transitions sont inévitables, il est indispensable de traverser cette plage intermédiaire le plus rapidement possible. D'autre part, les signaux soivent être stabilisés avavt d'être pris en compte par les circuits. Il y a donc des contraintes temporelles, spécifiées par les chronogrammes des feuilles des données (data sheets) fournis par les constructeurs. Dans ce cours, destiné à des informaticiens, nous n'aborderons que très peu cet aspect pratique de l'électronique.




Logique positive logique négative

Figure2

Nous verrons que dans certains cas (supports magnétiques, lignes de transmission, disques optiques, etc……) l'information peut être portée par les transitions entre deux états.

## II.2 Opérateur OU (inclusif) :

L'opération OU (OR), encore appelée addition logique, a 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. La fonction OU , notée + est représentée par le symbole indiqué sur la figure3 et est définie par la table de vérité suivante :

| A | B | Y= A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Table 2**

Une table de vérité donne pour toutes les combinaisons possibles des variables logiques en entrée X, ici X= (A , B), la valeur de la fonction logique f (X).

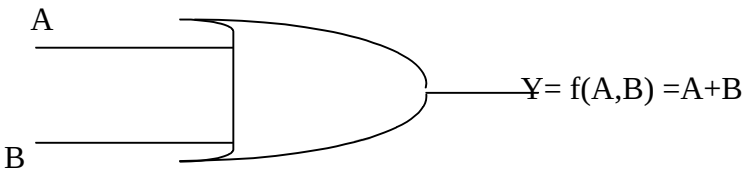



**Figure 3**

Il est facile de vérifier les propriétés suivantes de la fonction OU :

$$\begin{cases} (A+B) + C = A + (B + C) = A+B+C & \text{Associativité} \\ A + B = B+A & \text{Commutativité} \\ A + A = A & \text{Idempotence} \\ A + 0 = A & \text{Elément neutre} \\ A + 1 = 1 & \text{Elément absorbant} \end{cases}$$

## II.3 Opérateur ET

L'opérateur ET (AND), encore dénommée produit logique ou intersection, a 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. La fonction ET , notée $\bullet$, est représentée par le symbole indiqué sur la figure 4 et est définie par la table de vérité suivante :

| A | B | Y= A•B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Table 3**

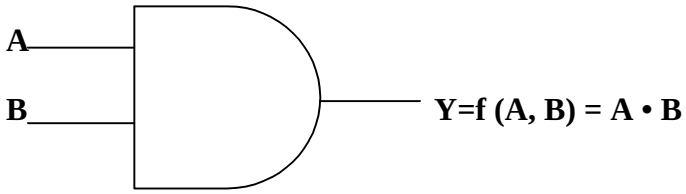



**Figure 4**

Il est facile de vérifier les propriétés suivantes de la fonction ET :

$$\begin{cases} (A \bullet B)\bullet c = A\bullet(B \bullet C) = A \bullet B \bullet C & \text{Associativité} \\ A \bullet B = B \bullet A & \text{Commutativité} \\ A \bullet A = A & \text{Idempotence} \\ A \bullet 1 = A & \text{Elément neutre} \\ A \bullet 0 = 0 & \text{Elément absorbant} \end{cases}$$

D'autre part, les opérations ET et OU sont distributives l'une par rapport à l'autre :

$$\begin{cases} A \bullet (B+C) = (A \bullet B) + (A \bullet C) & \text{Distributivité du ET sur le OU} \\ A + (B \bullet C) = (A+B) \bullet (A+C) & \text{Distributivité du OU sur le ET} \end{cases}$$

Mentionnons également les propriétés d'absorption :

$$\begin{cases} A + (A \bullet B) = A \\ A \bullet (A+B) = A \end{cases}$$

En effet :

$$A + (A \bullet B) = (A \bullet 1) + (A \bullet B) = A \bullet (1+B) = A \bullet 1 = A$$

$$A \bullet (A+B) = (A \bullet A) + (A \bullet B) = A + (A \bullet B) = A$$

## II.4 Inverseur : porte NON

L'opération NON (NOT) a 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. La négation logique est symbolisée par un petit cercle dessiné à l'endroit où une ligne en entrée ou en sortie rejoint symbole logique, comme par exemple sur la figure 5. La table 4 donne la table de vérité correspondante.

| **A** | **$Y = \overline{A}$** |
|---|---|
| **0** | **1** |
| **1** | **0** |

**Table 4**

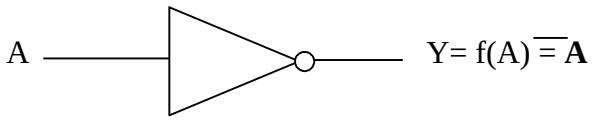



**Figure 5**

A partir des définitions des fonctions NON, OU et ET nous pouvons déduire :

$$\begin{cases} \overline{\overline{A}} = A \\ \overline{A} + A = 1 \\ \overline{A} \bullet A = 0 \\ \overline{A} + (A \bullet B) = A+B \end{cases}$$

## II.5 Théorèmes de DE Morgan

DE Morgan a exprimé deux théorèmes qui peuvent se résumer sous la forme suivante :

$$\begin{cases} \overline{A \bullet B \bullet C \bullet \ldots\ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots\ldots \\ \overline{A + B + C + \ldots\ldots} = \overline{A} \bullet \overline{B} \bullet \overline{C} \bullet \ldots\ldots \end{cases}$$

Pour vérifier le premier théorème nous remarquons que si toutes les entrées sont à 1 les deux membres de l'équation sont nuls. Par contre si une au moins des entrées est à 0 les deux membres de l'équation sont égaux à 1. Il y a donc égalité quels que soient les états des diverses entrées. Le second théorème se vérifie de la même manière : si toutes les entrées sont à 0 les deux membres de l'équation sont à 1. Par contre si au moins une des entrées est à 1 les deux expressions sont à 0.

Les théorèmes de DE Morgan montrent qu'une fonction ET peut être fabriquée à partir des fonctions OU et NON. De même une fonction OU peut être obtenue à partir des fonctions ET et NON. La figure 6 montre la conversion d'une porte OU en porte ET et réciproquement, utilisant le fait que :

$$\begin{cases} \overline{(\overline{A} \bullet \overline{B})} = \overline{\overline{A}}\,\overline{\overline{B}} = A + B \quad (1) \\ \overline{A + B} = \overline{\overline{A}} \bullet \overline{\overline{B}} = A \bullet B \quad (2) \end{cases}$$

De même à partir des théorèmes de DE Morgan nous pouvons montrer qu'une porte ET en logique positive fonctionne comme une porte OU en logique négative et vice versa.

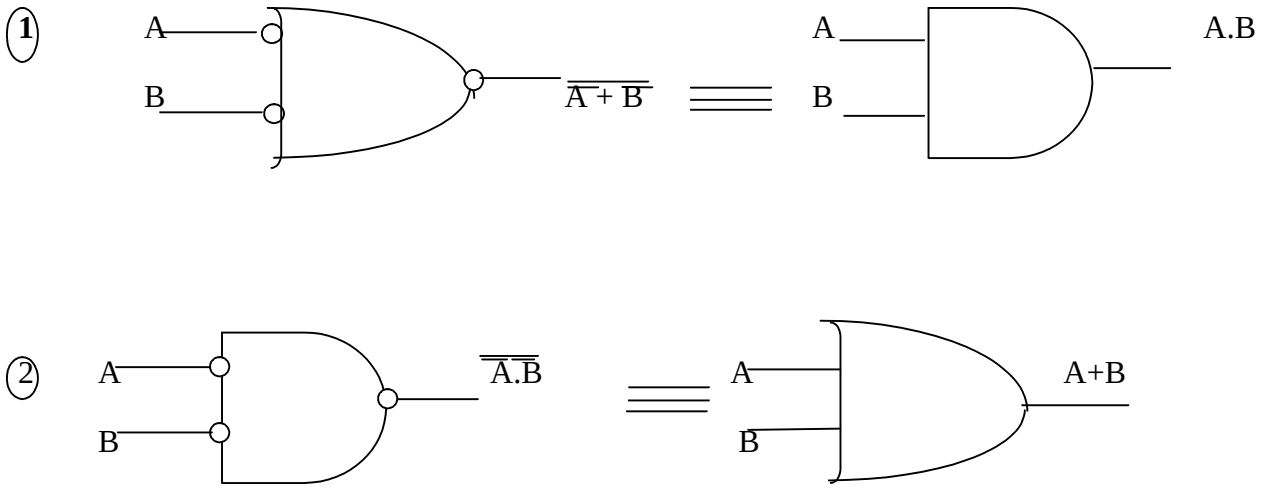



Figure 6

## II.6 Opérateurs NON ET et NON OU

Une porte NON ET ( NAND :NOT AND) est constituée par un inverseur à la sortie d'une porte ET (fig7). Une négation à la sortie d'une porte OU constitue une fonction NON OU (NOR :NOT OR)symbolisée sur la figure 8. Leurs tables de vérité respectives sont données par les tables 5 et 6 :

| A | B | $Y = \overline{A \bullet B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Table 5**

| A | B | $Y = \overline{A + B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Table 6**

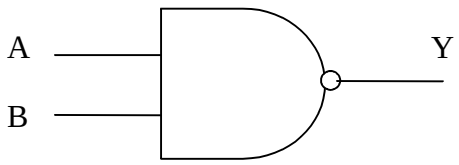



**Figure 7**

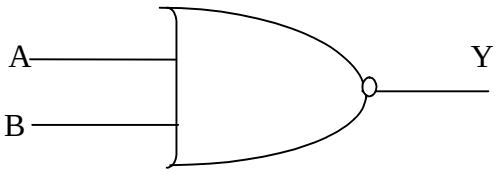



**Figure 8**

Comme les transistors qui interviennent comme éléments de base des portes sont par essence des inverseurs, les portes NAND et NOR sont très usitées dans la réalisation des circuits logiques. Grâce aux lois de De Morgan il est possible de réaliser des systèmes logiques avec uniquement des portes NAND ou NOR. La figure 9 montre, par exemple, comment les portes NOT, OR,AND peuvent être obtenues à partir de portes NOR.

Porte NOR

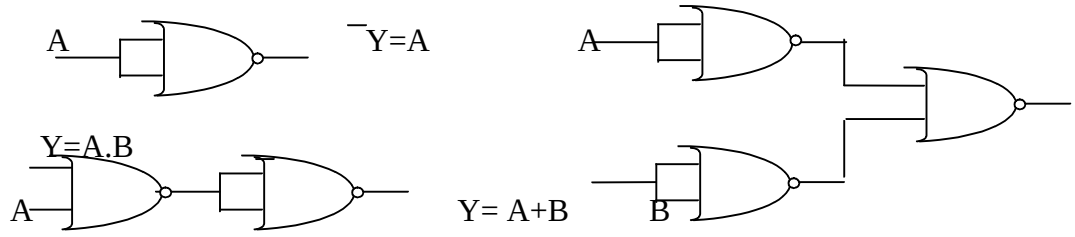



**figure 9**

## II.7 Opérateur OU exclusif

La sortie d'une fonction OU exclusif (XOR) à deux entrées est dans l'état 1 si une entrée et seulement une est dans l'état 1.la représentation symbolique d'une fonction XOR (notée $\oplus$ ) est donnée sur la figure 10 et sa table de vérité est la suivante :

| A | B | Y= A +B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

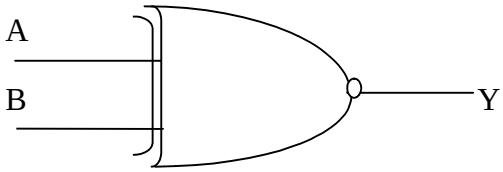



**Figure 10**

Nous pouvons formuler de diverses manières la définition précédente : Y=A $\oplus$ B est égal à 1 si et seulement si A = 1 ou B=1 mais pas simultanément. Ce que nous pouvons écrire : **$A \oplus B = (A+B)\bullet \overline{(A \bullet B)}$**

Nous pouvons encore dire Y= A ⊕ B est égal à 1 si A= 1 et B=0 ou si B=1 et A=0. Soit :

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

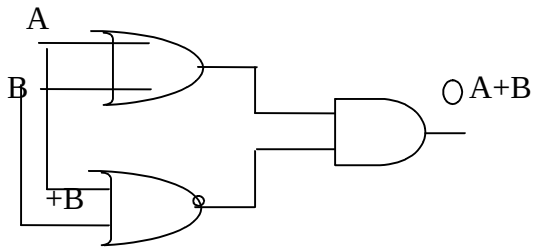
Une fonction XOR fournit un comparateur d'inégalité : Y= A ⊕ Bne vaut 1 que si A et B sont différents. Si A et B sont égaux à 1 ou si A et B sont égaux à 0 alors Y=0. Ce qui s'écrit :

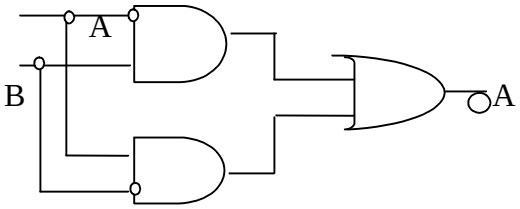
$$A \oplus B = \overline{(A \cdot B)} + \overline{(A \cdot B)}$$

la fonction $Z=\overline{Y}= \overline{A \oplus B}$ correspond à un détecteur d'égalité. Nous avons encore la relation suivante qui peut être démontrée en utilisant les théorèmes de DE Morgan :

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

A ces quatre relations logiques correspondent quatre circuit réalisant la fonction XOR à partir de portes OR et AND.



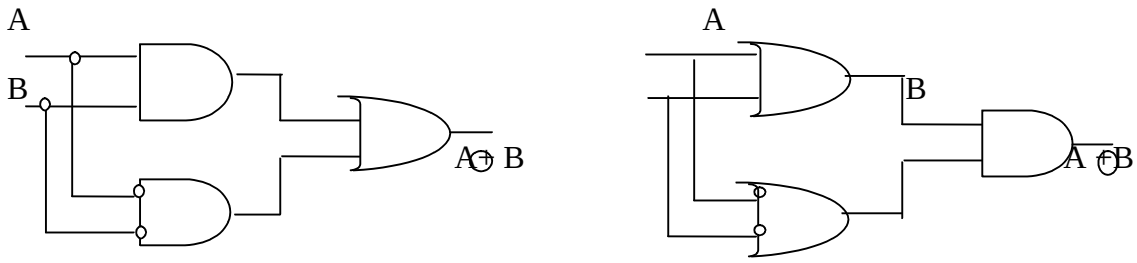



**Figure11**

Mentionnons également quelques propriétés faciles à vérifier :

**A ⊕A = 0**

**⊕ A = 1**

**A ⊕ 0 = A**

**A ⊕ 1 = $\overline{A}$**

**$\overline{A}$ ⊕ $\overline{B}$ = A ⊕ B**

⊕ = ⊕ = ⊕

## II.8 Opérateurs à trois Etats

La porte « 3 états », ou « tri-state », ,'est pas une porte logique au sens strict. Elle est principalement utilisée pour connecter une sortie sur une ligne commune à plusieurs circuits (un bus par exemple). Elle remplace généralement une porte ET. En effet, la mise en parallèle sur une même ligne de plusieurs portes ET introduit des capacités parasites. Ceci augmente les constantes de temps et a pour effet de détériorer les fronts de montée et de descente des signaux. Cela peut perturber le fonctionnement d'un système. Une porte 3 états est schématisée sur la figure suivante :

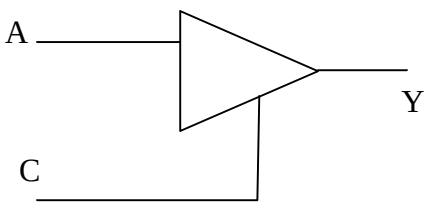



**Figure 12**

| C | A | Y | Sortie |
|---|---|---|---|
| **1** | **0** | **0** | **Faible impédance** |
| **1** | **1** | **1** | **Faible impédance** |
| **0** | **X** | **0** | **Haute impédance** |

**Table 8**

Lorsque la commande C est à 0 l'impédance de sortie est très grande : pratiquement déconnectée d'autre part, ces portes « 3 état »fournissent une amplification de puissance.

## II.9 Résumé des identités booléennes de base

Il est possible de montrer que toute fonction booléenne d'un nombre quelconque de variables peut s'écrire avec les trois fonctions de base ET, OU et NON. Nous avons rassemblé dans la table 9 des relations de base de l'algèbre de Boole qui nous seront utiles par la suite.

| | | |
|---|---|---|
| **OU** | **(A+B)+C = A+ (B+C) = A+B+C**<br>**A + B= B + A**<br>**A + A = A**<br>**A + 0 = A**<br>**A + 1 = 1** | **Associativité**<br>**Commutativité**<br>**Idempotence**<br>**Elément neutre**<br>**Elément absorbant** |
| **ET** | **(A • B)•C = A•(B•C) = A•B•C**<br>**A•B = B•A**<br>**A•A =A**<br>**A•0 =A**<br>**A•1 = 0** | **Associativité**<br>**Commutativité**<br>**Idempotence**<br>**Elément neutre**<br>**Elément absorbant** |
| **Distributivité** | **A • (B+C) = (A•B) + (A•C)**<br>**A+ (B•C) = (A+B) • (A+C)** | |

| | |
|---|---|
| **NON** | $\overline{\overline{A}} = A$<br>$\overline{A} + A = 1$<br>$\overline{A} \cdot A = 0$ |
| | $A+(A \cdot B)=A$<br>$A \cdot (A+B)= A$<br>$(A+B) \cdot (A+\overline{B}) = A$<br>$A+(\overline{A} \cdot B) = A+B$ |
| **De Morgan** | $\overline{A \cdot B \cdot C \cdot \ldots} = \overline{A}+\overline{B}+C+\ldots\ldots$<br>$\overline{A+B+C+\ldots\ldots} = \overline{A} \cdot \overline{B} \cdot C \cdot \ldots\ldots\ldots$ |
| **OU** | $A \ominus B = (A+B) \cdot \overline{(A \cdot B)}$<br>$A \ominus B = (A \cdot \overline{B}) + (B \cdot A)$<br>$A \oplus B = \overline{(A \cdot B)} + \overline{(\overline{A} \cdot B)}$<br>$A \oplus B = (A+B) \cdot (\overline{A}+B)$<br>$A \oplus A = 0$<br>$\overline{A} \oplus A = 1$<br>$A \oplus 0 = A$<br>$A \oplus 1 = \overline{A}$<br>$\overline{A} \oplus \overline{B} = A \oplus$<br>$\oplus = \oplus = \oplus$<br>○ ○ ○ |

Table 9

## II.10 Ecritures canoniques d'une fonction logique

### II.10a. Somme canonique de produits

Considérons trois variables booléennes x, y et z. A partir de ces trois variables nous pouvons construire huit produits logiques ( ou minterms) $P_{i=0,7}$ faisant intervenir x ou $\bar{x}$, y ou $\bar{y}$ et z ou $\bar{z}$. pour chacun des huit combinaisons $C_{i=0,7}$ (000, 001, 010,etc……) des variables x, y et z, nous pouvons calculer les valeurs de ces produits. Celles-ci sont rassemblées dans la table 10. Chacun de ces produits prend la valeur 1 pour une et une seule combinaison : $P_i$ vaut 1 uniquement pour la combinaison $C_i$ et 0 pour les autres combinaisons.

| | | | | $P_0$ | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_i$ | x | y | z | $\bar{x}\,\bar{y}\,\bar{z}$ | $\bar{x}\,\bar{y}\,z$ | $\bar{x}\,\bar{y}\,z$ | $\bar{x}\,y\,z$ | $x\,\bar{y}\,\bar{z}$ | $x\,\bar{y}\,z$ | $x\,y\,\bar{z}$ | $x\,y\,z$ |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Table 10

Pour toute fonction logique de trois variables x, y et z, nous pouvons écrire sa table de vérité, c'est-à-dire expliciter sa valeur pour chacune des huit combinaisons $C_i$. Considérons, par exemple, la fonction F dont la de vérité est donnée dans la table 11 :

| $C_i$ | x | y | z | F | $P_1+P_3+P_4$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 |

Table 11

Cette fonction F prend la valeur 1 pour la combinaison $C_1$ comme le produit $P_1$. La combinaison $C_3$ comme $P_3$ et la combinaison $C_4$ comme $P_4$. La fonction F prenant la valeur 0 pour toutes les autres combinaisons comme les produits $P_1$, $P_3$, $P_4$, nous pouvons donc écrire que F est égale à la fonction :

$$F = P_1 + P_3 + P_4$$

Nous pouvons vérifier cette identité dans la table 11. Nous pouvons donc exprimer F en fonction des variables x, y et z sous la forme :

$$F = \bar{x}\,\bar{y}\,z + \bar{x}\,y\,z + x\,\bar{y}\,\bar{z}$$

Cette façon, très générale, d'écrire une fonction booléenne est appelée somme canonique de produits.

II.10.b Produit canonique de sommes

Soient encore trois variables binaires x, y et z. Nous pouvons définir huit sommes logiques des trois variables faisant intervenir x ou $\bar{x}$, y ou $\bar{y}$ et z ou $\bar{z}$. La table 12 donne les tables de vérité de ces sommes. Nous constatons que chacune de ces fonctions prend la valeur 0 pour une et une seule combinaison.

| | | | | $S_0$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_i$ | X | y | z | $x+y+z$ | $x+y+\bar{z}$ | $x+\bar{y}+z$ | $x+\bar{y}+\bar{z}$ | $\bar{x}+y+z$ | $\bar{x}+y+\bar{z}$ | $\bar{x}+\bar{y}+z$ | $\bar{x}+\bar{y}+\bar{z}$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 5 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Table 12

Reprenons l'exemple précédent de la fonction F. Celle-ci vaut 0 pour les combinaisons $C_0$, $C_2$, $C_5$, $C_6$ et $C_7$ en même temps que $S_0$, $S_2$, $S_5$, $S_6$ et $S_7$. La fonction F peut donc être vue comme le produit logique de ces cinq sommes, ce qui est vérifié dans la table 13. Nous pouvons donc exprimer la fonction F sous la forme suivante :

$$F = (x + y + z) \bullet (x + \overline{y} + z) \bullet (\overline{x} + y + \overline{z}) \bullet (\overline{x} + \overline{y} + z) \bullet (\overline{x} + \overline{y} + \overline{z})$$

Cette écriture est appelée produit canonique de sommes. Celle-ci est moins utilisée que la somme canonique de produits.

| Ci | x | y | z | t | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 |

Table 13

## II.11 Simplification de l'écriture des fonctions logiques

II.11.a. Simplification algébrique Simplifier une expression booléenne c'est lui trouver une forme plus condensée, faisant intervenir moins d'opérateurs et conduisant à une réalisation matérielle plus compacte. On peut simplifier une fonction par manipulation algébrique en utilisant par exemple les relations rassemblées dans la table 9. Considérons la fonction F définie par la table de vérité suivante :

| x | y | z | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Table 14

Nous en déduisons sa forme canonique somme produits :

$$F = \bar{y}\, z + x\, \bar{z} + x\, y\, \bar{} + x\, y\, z$$

Nous pouvons écrire :

$$F = \bar{y}\, z + x\, \bar{z} + x\, y\, \bar{} + x\, y\, z$$

$$= (\bar{y}\, z + x\, y\, z) + (x\, \bar{z} + x\, y\, z) + (x\, y\, \bar{} + x\, y\, z) \quad \text{car } A+A=A$$

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

$$= x\, y + y\, z + z\, x$$

Cependant cette méthode, qui demande astuce et chance, n'est pas toujours très aisée à mettre en œuvre. Nous allons maintenant décrire une méthode graphique très utile pour un nombre de variables inférieur à 6.

## II.11.b Tableaux de Karnaugh

La méthode de simplification de Karnaugh repose sur l'identité :

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

Elle est basée sur l'inspection visuelle de tableaux disposés de façon telle que deux cases adjacentes en ligne et en colonne ne diffèrent que par l'état d'une variable et une seule.

Si une fonction dépend de n variables il y a $2^n$ produits est représenté par une case dans un tableau. Les figures suivantes donnent la structure des tableaux de Karnaugh pour 2, 3, 4 et 5 variables. Observez comment sont numérotées les lignes et les colonnes : d'une case à sa voisine une seule variable change d'état. Pour 5 variables, deux représentations sont possibles. Dans ce cas le tableau de Kernaugh peut être traité comme deux tableaux 4 x 4 superposés ( fig16) ou un seul tableau de 4 x 8 ( fig17)

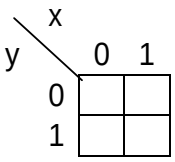


Tableau à 2 variables

Figure 13

Chaque case d'un tableau correspond au seul minterm prenant la valeur 1 pour la combinaison identifiée par la ligne et la colonne. Par exemple les trois cases coloriées dans les tableaux de la figure 18 correspondent respectivement aux produits suivants :

$$x\,y\,z\,t, \quad \bar{x}\,\bar{y}\,\bar{z}\,\bar{t} \quad \text{et} \quad x\,y\,\bar{z}\,\bar{t}$$

Il faut comprendre chaque ligne et chaque colonne comme une structure cyclique continue :

Chaque case a toujours quatre voisins qu'il faut éventuellement chercher à l'autre extrémité de la ligne ou de la colonne. Les tableaux de la figure 18 illustrent ce concept, les croix y matérialisent les voisins des cases coloriées :

Dans le cas de 5 variables chaque case possède cinq voisins. Dans la représentation en tableaux superposés, quatre voisins se situent dans le même plan et le cinquième «  à la vertical » dans l'autre plan. Pour la représentation plane de 8 colonnes il faut « replier » le tableau par rapport à la ligne médiane séparant les colonnes 010 et 110, qui définit un axe de symétrie. La figure 19 illustre les 5 cases voisines de la case 10101.

Le passage de la table de vérité au tableau de Karnaugh consiste à remplir chaque case avec la valeur de la fonction pour le produit correspondant. Il est possible de ne copier que les 1.

La méthode de simplification de Karnaugh consiste à rassembler les cases adjacentes contenant des 1 par groupe de 2, 4 ou 8 termes. Considérons en effet le groupement vertical de deux cases, en rouge, de la figure 20. Il correspond à la somme de deux termes :

$$G = x\,y\,t + x\,y\,\bar{}$$

Il est possible de factoriser le produit x y :

$$G = x\,y\,(t+ t) = x\,y$$

La variable t qui prend les deux valeurs 0 et 1 dans le groupement disparait. Il ne reste que le produit des variables x et y, qui garde ici la valeur 1.

Dans un groupement de deux termes en élimine donc la variable qui change

d'état et on conserve le produit des variables qui ne changent pas. Dans un groupement de quatre on élimine les deux variables qui changent d'état. Dans un groupement de huit on élimine trois variables, ….

On cherche à avoir le minimum de groupement, chaque groupement rassemblant le maximum de termes. Une même case peut intervenir dans plusieurs groupements car C + C = C. c'est la cas de la case jaune sur la figure 20.

Pour les cases isolées on ne peut éliminer aucune variable. On concerve donc le produit caractérisant la case. L'expression logique finale est la réunion des groupements après élimination des variables qui changent d'état.

Reprenons l'exemple de la fonction F définie par la table de vérité 14. La figure 20 donne le tableau de Karnaugh correspondant :

Nous y observons trois groupements de deux termes, nous pouvons écrire pour la fonction :

$$F = x\,y + y\,z + z\,x$$

Nous retrouvons le résultat précédent.

Considérons une fonction F de quatre variables x, y , z et t définie par la table 15. La figure 21 donne le tableau de Karnaugh équivalent. Sur cette figure nous avons également matérialisé les trois groupements possibles : deux groupement de quatre termes, dont un contenant les quatre coins, et un groupement de deux termes. Cette méthode nous permet d'écrire :

$$F = x^{\_} + {}^{\_}{}^{\_} + y^{\_} t$$

| x | y | z | t | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

# Chapitre 3
# Logique combinatoire.

Un circuit gouverné par les règles de la logique combinatoire possède une ou plusieurs entrées, et une ou plusieurs sorties, et obéit à la propriété suivante :

L'état de la (ou des) sortie(s) à un instant donné ne dépend que du circuit et de la valeur des entrées à cet instant.

Une même combinaison des entrées donnera ainsi toujours la même valeur des sorties. Cette propriété, qui peut paraître de bon sens, n'est pas systématiquement vérifiée.

## 3.1. *Représentation schématique des fonctions logiques de base.*

### 3.1.1. Les fonctions NON, ET, OU.

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 sur la Figure 1.

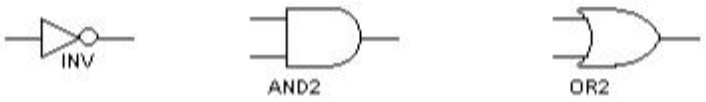



Figure 1 Symboles associés aux fonctions logiques NON, ET, OU

On rencontre aussi d'autres fonctions logiques réalisées à partir des 3 fonctions précédentes.

### 3.1.2. La fonction NON ET (NAND).

La fonction NON ET est obtenue en complémentant la fonction ET $F = \overline{x \bullet y}$ . La table de vérité et le symbole associés à cette fonction sont :

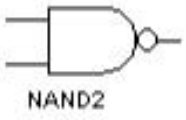



Figure 2 Symbole associé à la fonction NON ET (NAND)

| $x$ | $y$ | $F = \overline{x \bullet y}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |

### 3.1.3. La fonction NON OU (NOR).

La fonction NON OU est obtenue en complémentant la fonction OU $F = \overline{x + y}$ . La table de vérité et le symbole associés à cette fonction sont :

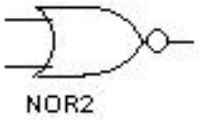



Figure 3 Symbole associé à la fonction NON OU (NOR)

| $x$ | $y$ | $F = \overline{x + y}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 1 | 0 |

### 3.1.4. La fonction OU EXCLUSIF (XOR).

La fonction OU EXCLUSIF ne vaut 1 que si les deux entrées sont différentes. Elle s'écrit $F = \overline{x} \oplus y = x \bullet \quad + \quad \overline{} \bullet y$. La table de vérité et le symbole associés à cette fonction sont :


Figure 4 Symbole associé à la fonction OU EXCLUSIF

(XOR)

| $x$ | $y$ | $F = x \oplus y$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |

**3.2. *Réalisation matérielle d'une fonction logique.***

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 complexe à l'aide des portes de base. A titre d'exemple réalisons la fonction OU EXCLUSIF (XOR) en n'utilisant que des portes NON, ET, OU.

- *1ére méthode :*

On réalise la fonction $F = x \oplus y = \overline{} \bullet \quad + \quad \bullet$ telle qu'elle est écrite. Obtient alors le schéma de la Figure 5

Cette solution n'est pas satisfaisante pour au moins deux raisons

1. On n'a pas cherché à minimiser le nombre de portes utilisées.

2. On utilise 3 types de portes différents donc 3 boîtiers différents sur le montage. Sachant qu'un boîtier contient plusieurs portes (4 ou 6 généralement) on peut facilement gagner de la place en n'utilisant qu'un seul type de portes.

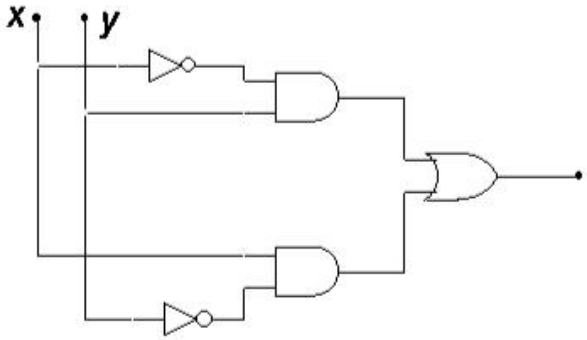



Figure 5 Réalisation de la fonction OU EXCLUSIF (XOR)

*2$^{eme}$ méthode :*

Pour simplifier la réalisation, on cherche à n'utiliser qu'un seul type de portes, par exemple des portes NAND.

La fonction s'écrit alors

$$F = x \oplus y = x \bullet \overline{y} + \overline{x} \bullet y = \overline{\overline{x \bullet \overline{y}} \bullet \overline{\overline{x} \bullet y}}$$

et on obtient le schéma suivant (Figure 6), avec 5 portes NAND, en notant que $\overline{\overline{x}} \bullet \overline{x} = x$.

Il faut noter ici que la fonction NAND, comme d'ailleurs la fonction NOR, est dite complète car elle permet de réaliser à elle seule toutes les fonctions logiques.

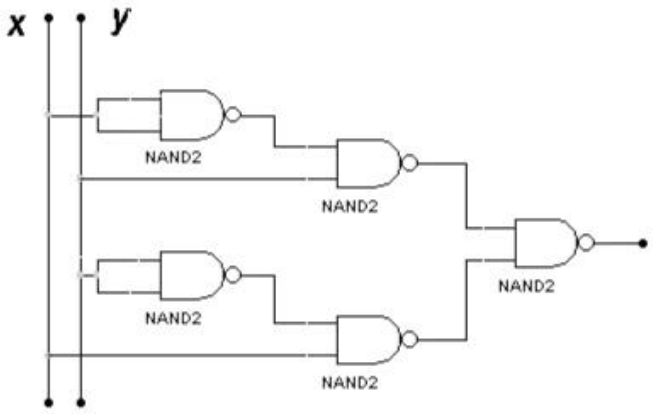


Figure 6 Réalisation de la fonction OU EXCLUSIF uniquement avec des portes NAND

*3$^{eme}$ méthode :*

En étant astucieux on remarque que

$$x \bullet \bar{y} = x \bullet (\bar{y} + \bar{x}) = x \bullet \overline{x \bullet y}$$
$$\bar{x} \bullet y = y \bullet (\bar{y} + \bar{x}) = y \bullet \overline{x \bullet y}$$

La fonction *F* peut alors se réécrire sous la forme

$$F = \overline{\overline{x \bullet \overline{x \bullet y}} \bullet \overline{y \bullet \overline{x \bullet y}}}$$

ce qui conduit au schéma de la Figure 7 qui ne comporte que 4 portes NAND!

Une remarque s'impose : Ce n'est pas nécessairement l'expression la plus simplifiée de la fonction logique *F* qui donne le logigramme (et donc le circuit électronique) le plus simple ou le plus optimisé. Ici nous avons gagné une porte logique en écrivant $F = x \bullet (\bar{y} + \bar{x}) + y \bullet (\bar{y} + \bar{x})$ au lieu de $F = x \bullet \bar{y} + \bar{x} \bullet y$.

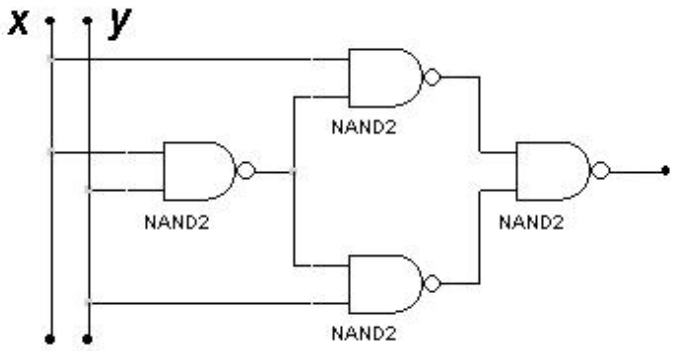



Figure 7 Réalisation de la fonction OU EXCLUSIF avec le minimum de portes logiques.

## 3.4. *Quelques circuits logiques "complexes".*

En pratique il n'est pas nécessaire de réaliser des fonctions logiques complexes en utilisant les portes logiques de base que nous venons de décrire. On trouve dans les catalogues des différents constructeurs un grand nombre de circuits logiques intégrés réalisant des fonctions logiques complexes. Nous verrons simplement ici le multiplexeur, l'encodeur et le décodeur.

### 3.4.1. Le multiplexeur (sélecteur de données).

- ***Principe du multiplexeur.***

Un multiplexeur possède plusieurs entrées et une seule sortie. Il agit comme un sélecteur de données en orientant vers sa sortie la donnée présente sur l'une de ses entrées. Pour fixer les idées, considérons le multiplexeur le plus simple représenté sur la Figure 12. Il s'agit d'un circuit logique à deux entrées, $D_0$ et $D_1$, permettant d'afficher sur sa sortie $F$ la donnée présente sur une des deux entrées. Ceci n'est réalisable que si il existe une entrée supplémentaire $S_0$ , dite entrée de sélection ou d'adresse, telle que

$$F = D_0 \text{ si } S_0 = 0$$
$$F = D_1 \text{ si } S_0 = 1$$

Un multiplexeur comporte donc deux types d'entrées : les entrées de données (ou d'informations), et les entrées de sélection (ou d'adresse) dont les combinaisons servent à numéroter les entrées d'informations

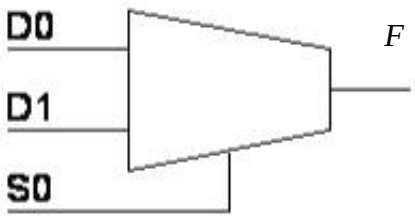



**Figure 12** Multiplexeur à deux entrées de données $D_0$ et $D_1$, et une entrée de sélection $S_0$ (entrée d'adresse)

***Généralisation.***

Pour sélectionner parmi quatre entrées il faut impérativement disposer de deux entrées de sélection: chacune des quatre combinaisons possibles des entrées de sélection correspondra à l'aiguillage d'une entrée, et d'une seule, vers la sortie. De la même façon pour sélectionner parmi huit entrées données, trois entrées de sélections sont nécessaires ($2^3=8$). De façon générale, un multiplexeur possédant $n$ entrées de sélection permet de sélectionner une entrée parmi $2^n$ .

La sélection de l'entrée est réalisée en affectant un poids aux entrées d'adresse $S_n,\ldots S_0$ . On peut, par exemple, associer $S_n$ au bit de poids fort et $S_0$ a celui de poids faible. La combinaison binaire $S_n$ K$S_0$ ainsi obtenue est appelée **adresse** et le multiplexeur aiguillera vers la sortie $F$ l'entrée $D_i$ dont l'indice décimal $i$ correspondra à l'adresse binaire $i = (S_n\ S_0\ )_2$ . Ainsi, dans un multiplexeur à 8 entrées de données, l'entrée $E_5$ sera aiguillée vers la sortie si l'adresse binaire écrite sur les entrées de sélection est[4]:

$S_2$ $S_1$ $S_0$
1 0 1

Les multiplexeurs ont de nombreuses applications. Ils peuvent par exemple être utilisés comme :

- sélecteur de données.
- convertisseur parallèle-série. Le multiplexeur reçoit en parallèle des données qu'il peut transmettre l'une après l'autre sur sa sortie.
- générateur de fonctions logiques.

***Utilisation en générateur de fonction logique.***

La transmission de la donnée présente sur l'entrée $D_5$, dont l'adresse est 101, (voir plus haut) consiste pour le multiplexeur à réaliser un **ET** logique entre $D_5$ et le monôme $S_2 \bullet S_1 \bullet S_0$ . En fait un multiplexeur à $n$ entrées d'adresses (et donc $2^n$ entrées de données) peut réaliser toutes les fonctions logiques combinatoires de $n+1$ variables.

Considérons par exemple, la réalisation de la fonction $F = x \bullet y \bullet \bar{\ } + \bar{\ } \bullet \ \bullet z + y \bullet z$ avec un multiplexeur à 2 entrées d'adresse.

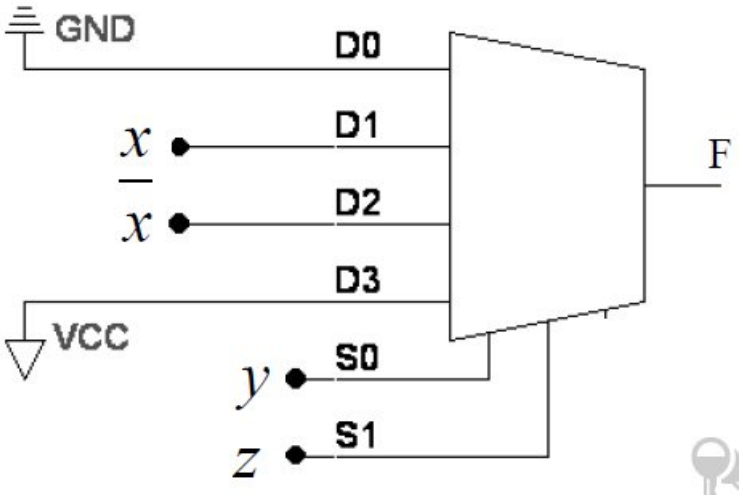



**Figure 13** Réalisation de la fonction logique avec un $F = x \bullet y \bullet \bar{} + \bar{} \bullet \ \bullet z + y \bullet z$
Multiplexeur à deux entrées d'adresse

On connecte 2 des variables aux entrées d'adresse ou de sélection. Par exemple $y$ à $S_0$ et $z$ à $S_1$. On doit ensuite connecter convenablement les 4 entrées de données de façon à reproduire les différents termes de la fonction $F$.

1. Pour le terme $x \cdot y \cdot \bar{z}$ on doit relier l'entrée $D_1$ dont l'adresse est 01 ($z = 0, y = 1$) à $x$.

2. Pour le terme $\bar{x} \cdot \bar{y} \cdot z$ on doit relier l'entrée $D_2$ dont l'adresse est 10 ($z = 1, y = 0$)à $\bar{x}$.

3. Pour le terme $z \cdot y$ on relie l'entrée dont l'adresse est 11 ($z = 1, y = 1$) au niveau logique 1 (par exemple 5volts en technologie TTL).

4. Toutes les autres entrées sont connectées au niveau logique 0 (la masse) puisque $F = 0$ pour les valeurs de $y$ et $z$ correspondantes.

Le schéma correspondant est représenté sur la Figure 13.

### 3.4.2. Encodeur prioritaire.

C'est un circuit à $m$ entrées et $n$ sorties. Les sorties délivrent le code de l'entrée active si il n'y en a qu'une ou de l'entrée prioritaire si il y en a plusieurs. Si le code est le code binaire standard alors on a $m=2^n$ et l'encodeur est dit binaire. Il existe bien entendu des encodeurs pour les codes BCD, GRAY, ASCII ... Dans le cas du code ASCII l'encodeur est utilisé pour coder les touches d'un clavier, c'est à dire pour générer le code ASCII

associé au caractère ou à la fonction de la touche enfoncée. Le fonctionnement d'un encodeur prioritaire a quatre entrée est décrit par la table de vérité suivante. Les entrées sont actives au niveau haut (1), lorsque plusieurs entrées $E_i$ sont actives, seul le code correspondant à l'entrée d'indice le plus élevé est présent sur les sorties $a_1$ $a_0$ . Dans cet exemple l'entrée $E_3$ a la plus forte priorité.

| $E_0$ | $E_1$ | $E_2$ | $E_3$ | $a_1$ | $a_0$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| x | 1 | 0 | 0 | 0 | 1 |
| x | x | 1 | 0 | 1 | 0 |
| x | x | x | 1 | 1 | 1 |

### 3.4.3. Le décodeur-démultiplexeur.

Un démultiplexeur est un aiguilleur à une entrée de donnée, *n* entrées d'adresse et *m* sorties. La valeur de l'entrée se retrouve sur la sortie dont le numéro est codé par l'adresse. Dans cette fonction le circuit joue le rôle inverse du multiplexeur.

Ces circuits sont aussi des décodeurs : si l'entrée est maintenue active, le numéro de la sortie reflète le code de l'adresse. Dans le cas d'un code binaire on a $m = 2^n$ . Mais il existe aussi des décodeurs pour les codes BCD, GRAY , …

Un démultiplexeur dont l'entrée E est maintenue active au niveau 1 peut également servir de générateur de fonctions logiques. En effet, puisque l'on retrouve sur les sorties toutes les combinaisons possibles des entrées d'adresse, une porte OU suffit pour fabriquer une fonction logique sous sa première forme canonique.

Soit la fonction logique de trois variables $f = \bar{a} \bullet b \bullet c + a \bullet \bar{b} \bullet c + a \bullet b \bullet \bar{c} + a \bullet b \bullet c$. Le codage des sorties avec l'adresse (*c b a*) correspond aux sorties 6, 5 , 3 et 7. D'où le schéma suivant

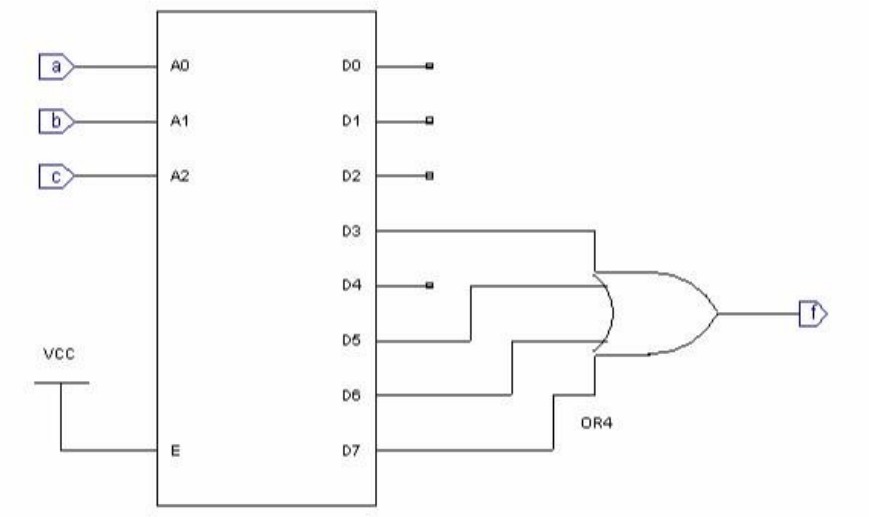



Figure 14 Réalisation d'une fonction logique avec un décodeur/démultiplexeur.

## 3.5 Réseaux logiques programmables

### 3.5 .1 Définition

• Un réseau logique programmable (circuit logique programmable) est un circuit qui peut être configuré par l'utilisateur pour avoir une ou plusieurs fonctions logiques. • Un circuit programmable est constitué d'un ensemble d'opérateurs ET et OU organisés sous forme de deux matrices. • La matrice des ET est un ensemble de portes AND qui permet de relier les différentes variables d'entrées . • La matrice des OU est un ensemble de portes OR qui permet de relier les différents termes AND. • Une matrice peut être programmable ( paramétrable ) ou figée ( préconfigurée ). • La programmation consiste a faire bruler (sauter) les fusibles des termes ( ou des variables ) qu'on veut pas utiliser laisser les fusibles utiles .

Remarques • La programmation se fait une seule fois : une fois les fusibles brulés on peut pas les réparer. • La programmation est réalisée grâce à un dispositif spécial .

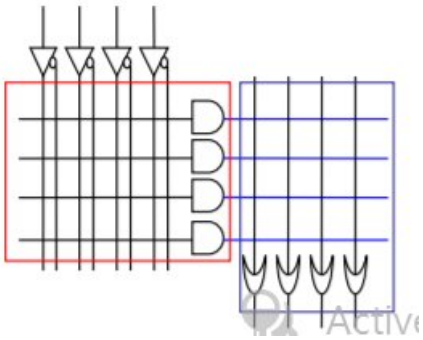


Figure 15 Schéma général d'un réseau logique programmable

3.6 Classification des réseaux programmables •

Selon le type des deux matrices on peut distinguer les trois types suivants :

– Matrice ET figée et OU programmable PROM (Programmable Read-Only Memory)

– Matrice ET programmable et OU figée PAL(Programmable Array Logic)

– Matrice ET programmable et OU programmable FPLA (Field ProgrammableArray Logic)

### 3.6.1 Les PROM

La matrice ET est figée : les produits sont déterminés La matrice des ET nous permet de générer toutes les combinaisons possibles La programmation consiste a choisir des termes et les relier par des OU.

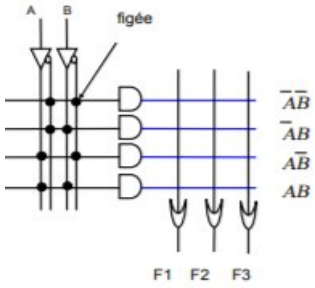


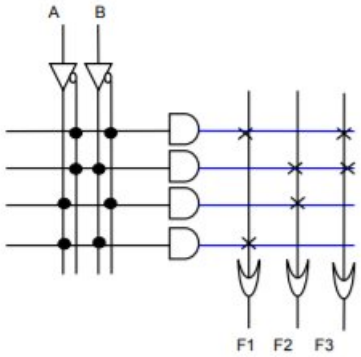

Figure 16 : Schéma général d'un

Les PROM : exemple

$f1 = \overline{A}.\overline{B} + A.B$

$f2 = A.\overline{B} + \overline{A}.B$

$f2 = \overline{A}.\overline{B} + \overline{A}.B$




### 3.6.2 Les PAL

La matrice OR est figée : chaque terme ou comporte un nombre déterminé de termes ET La matrice ET est programmable

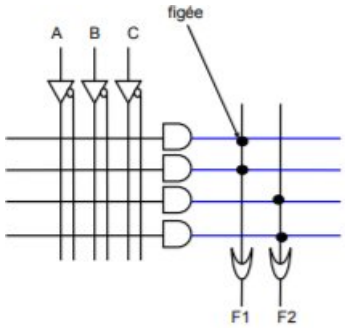



Figure 17 Schéma général d'un PAL

### 3.6.3 Les FPLA

Les deux matrices sont programmables, c'est le cas général des PROM et PAL

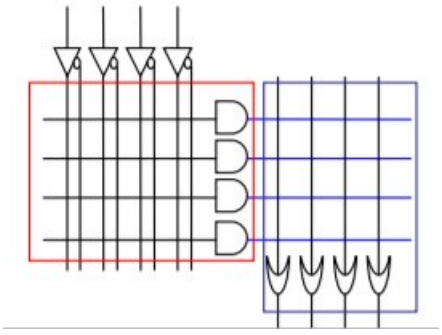


Figure 17 Schéma général d'un FPLA

3.7 Les réseaux programmables séquentiels •

Les PROM, PAL et les FPLA nous permet de réaliser uniquement des circuits combinatoire. • Il existe des réseaux programmable sequentiels : ces réseaux sont constitué d'une partie combinatoire et d'une partie sequentiels ( un ensemble de bascules en sortie). •

C'est possible d'utiliser ces réseaux sequentils pour résaliser des registres, des compteurs,.....

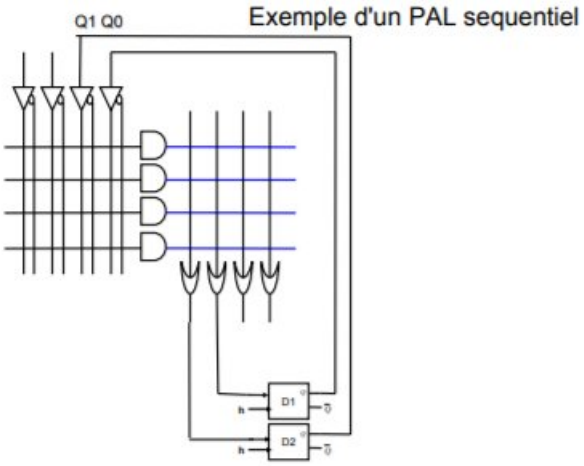



Figure 18 Schéma générale d'un réseaux programmables séquentiels

Reference:

- amrouche.esi.dz/doc/ch6_pal_fpla.pdf

Luc MUSEUR, «cours Électronique Numérique »,2 Université Paris 13, Institut Galilée.

# Chapitre 4
# Vue d'ensemble de l'ordinateur

Un ordinateur est une machine conçue pour exécuter au moins un programme formé d'une séquence d'instructions.
Nous décrivons ici l'organisation d'un ordinateur répondant au modèle de Von Neumann : c'est un modèle mis au point dans les années 1950, et qui est toujours utile pour comprendre le fonctionnement des ordinateurs.

## 4.1Architecture de Von Neumann

L'architecture de Von Neumann (cf. figure 1) met en fonctionnement quatre éléments fondamentaux : l'unité de traitement, l'unité de contrôle, l'unité d'échange (ou unité d'entrée/sortie) et la mémoire centrale. Cette dernière stocke deux types d'informations : les instructions du programme que la machine devra exécuter et les données sur lesquelles la machine effectuera les traitements dictés par les instructions. A ces deux types d'informations se chargent deux unités distinctes : l'unité de contrôle (appelée aussi unité de commande) et l'unité de traitement. L'unité de contrôle permet d'extraire à partir de la mémoire centrale l'instruction à exécuter ; l'analyser et établir les connexions électriques correspondantes ; extraire à partir de la mémoire centrale les données sur lesquelles porte l'instruction ; déclencher le traitement de ces données dans l'unité de traitement ; et ranger éventuellement les résultats de calcul dans la mémoire centrale. L'unité de traitement effectue sur les données qu'elle reçoit les traitements commandés par l'unité de contrôle.

L'unité de contrôle et l'unité de traitement forment en général un seul ensemble dans un ordinateur. On appelle cet ensemble par unité centrale ou CPU (*Central Processing Unit*). Quelques fois, on désigne par unité centrale l'ensemble : unité de contrôle, unité de traitement et la mémoire centrale.

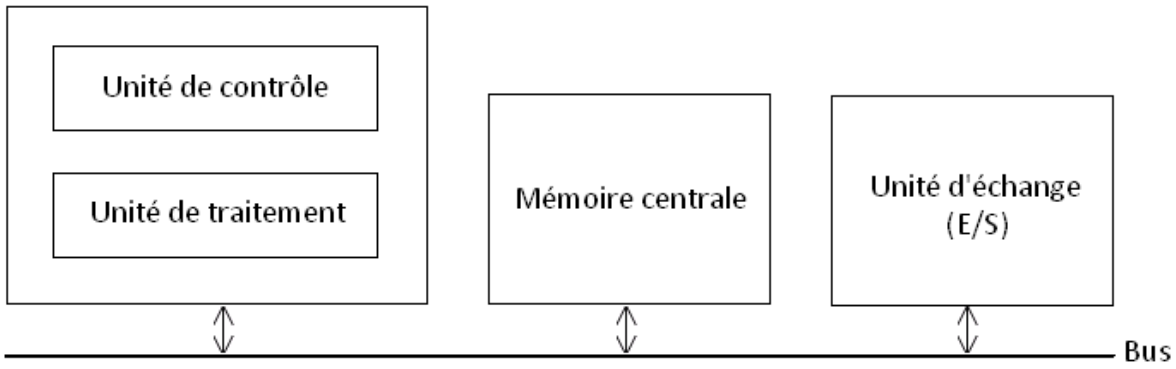



**Figure 1 : Architecture de Von Neumann**

### 4.1.1 CPU

Le CPU constitue le cœur de l'ordinateur. Il se charge de l'exécution des programmes et de la coordination entre les différents organes de l'ordinateur. Il est composé de deux parties : la partie opérative qui correspond à l'unité de traitement, et la partie de commande qui correspond à l'unité de contrôle.

### 4.1.2Unité de traitement

L'unité de traitement (cf. figure 2) se charge d'effectuer les traitements dictés par les instructions. Ces traitements se résument en des opérations arithmétiques (addition, soustraction, multiplication, etc.) et logiques (comparaison, ET logique, etc.). Elle est essentiellement composée de : (1) un ensemble de circuits logiques permettant de réaliser les différentes opérations arithmétiques et logiques. Ces circuits composent ce qu'on appelle UAL (Unité Arithmétique et Logique) ; (2) registre accumulateur (ACC) permettant de contenir l'opérande de l'instruction en cours d'exécution et les résultats intermédiaires ; (3) registre indicateur (IND) permettant de mémoriser l'état du registre ACC ; et (4) un ensemble de registres généraux servant généralement à contenir les opérandes.

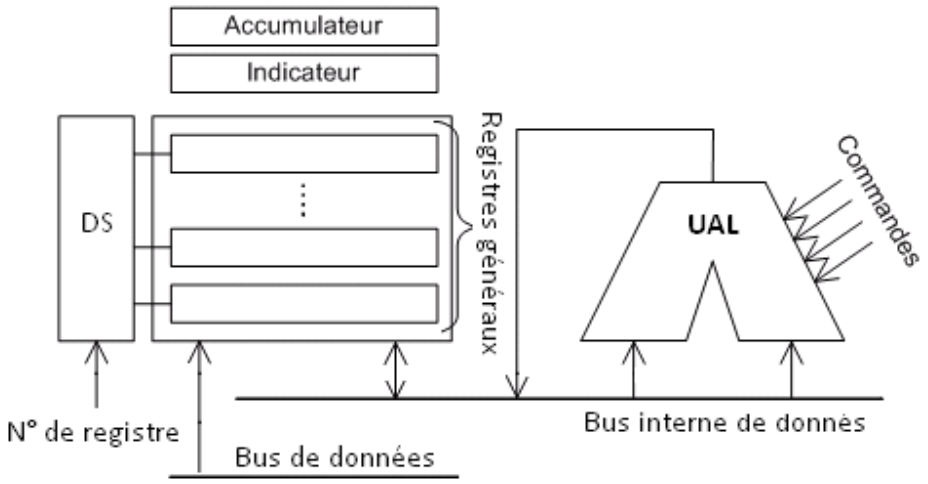



**Figure 2 : Unité de traitement**

Pour effectuer les différents calculs, l'unité de traitement dispose de plusieurs registres permettant de contenir provisoirement les opérandes (données) requis par les calculs. Une fois, les opérandes sont stockés dans les registres de cette unité, un signal de commande (provenant du bus de commande) précisant l'opération à effectuer va activer le circuit logique permettant de réaliser l'opération. Par exemple s'il s'agit d'une addition, c'est le circuit additionneur qui sera activé. Etant donné que cette unité dispose de plusieurs registres généraux, un dispositif de sélection (DS) permet de sélectionner celui qui est concerné par l'opération pour soit lui affecter un opérande en provenance de la mémoire centrale ou y accéder pour récupérer un opérande pour le stocker en mémoire centrale ou bien l'utiliser dans l'UAL pour obtenir un résultat de traitement. Le registre ACC est également utilisé par les instructions de branchement pour réaliser des ruptures de séquences au niveau de l'exécution des programmes. Ceci est réalisé grâce à des tests sur le contenu de ce registre. C'est le registre IND qui indique l'état du registre ACC. Ce registre est relié à l'unité de contrôle qui s'en sert pour exécuter les instructions de branchement. Certaines lignes de commandes proviennent de l'unité de contrôle. Ces lignes sont utilisées pour préciser le type d'opération à réaliser (addition, opération logique, etc.).

### 4.1.3Unité de contrôle

Le rôle de l'unité de contrôle (cf. figure 3) est l'exécution des instructions. Elle est composée essentiellement du registre RI (Registre Instruction) utilisé pour contenir l'instruction à exécuter ; registre CO (Compteur Ordinal) utilisé pour contenir l'adresse de la prochaine instruction à exécuter ; et un circuit logique appelé séquenceur qui se charge d'analyser le code opération COP de l'instruction (COP est une partie de l'instruction indiquant le type et la nature du traitement à effectuer) et générer un ensemble de signaux pour activer le circuit concerné à l'exécution de l'instruction.

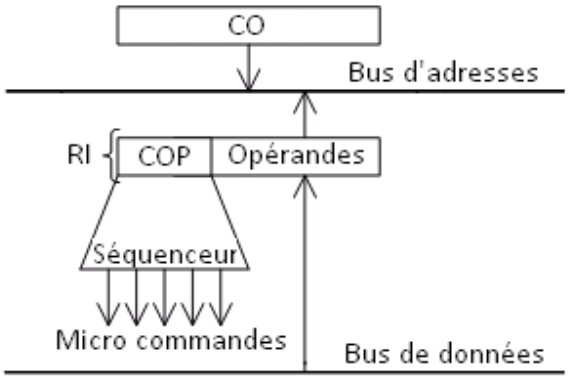



**Figure 3 : Unité de contrôle**

### 1.1.4Mémoire centrale

La mémoire centrale est un dispositif capable d'acquérir des informations, de les stocker (en binaire) et de les restituer à la demande. La représentation binaire de l'information fait que cette dernière soit codée suivant une configuration de bits ou chaque bit représente une information élémentaire de soit « 1 » ou « 0 ». La figure 4 illustre la composition de la mémoire centrale.

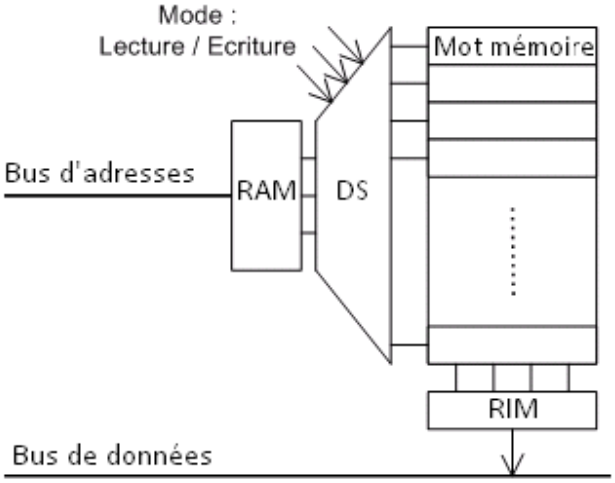



**Figure 4 : Mémoire centrale**

La mémoire centrale est composée du Registre Adresse Mémoire (RAM), Registre Information Mémoire (RIM), un dispositif de sélection (SD) et un ensemble de mots mémoire composant la mémoire centrale. Chaque mot mémoire est composé à son tour d'un ensemble de cellules chacune permettant de stocker un bit. Afin de repérer les mots mémoires, ces derniers sont disposés d'une façon ordonnée de sorte qu'à chacun soit associé une adresse.

Pour effectuer une lecture d'une information se trouvant dans un mot mémoire, il est nécessaire d'indiquer l'adresse de ce mot. Cette adresse, indiquée dans le registre RAM, doit être décodée pour sélectionner le mot mémoire qui lui correspond (ceci est réalisé grâce au Dispositif de Sélection DS). Le signal de sélection doit être positionné pour indiquer que la mémoire centrale est choisie. Dès que le signal de lecture/écriture indique qu'il s'agit d'une lecture, l'information se trouvant dans le mot sélectionné est dupliquée dans le registre RIM. Une fois dans le registre RIM, l'information peut être mise sur le bus de données pour qu'elle soit conduite vers une autre unité de l'ordinateur.

### 1.1.5Unité d'échange

Lorsque l'unité contrôle rencontre une instruction d'échange d'informations avec l'extérieur (instruction d'entrée/sortie), elle commande les unités d'échange. Il existe deux types d'unités d'échange : les mémoires auxiliaires et les unités de communication. Les mémoires auxiliaires sont aussi appelées mémoires secondaires par opposition à la mémoire centrale, et dont les capacités de stockage sont de loin plus importantes que celle de la mémoire centrale. Elles sont utilisées pour stocker d'une manière permanente l'information (disque dur, lecteur DVD, disque amovible, etc.). Les unités de communication ont pour rôle de communiquer l'information soit depuis l'ordinateur (mémoire centrale) vers l'utilisateur ou l'inverse (clavier, souris, écran, imprimante, etc.).

## 1.2 Cycle d'exécution d'une instruction

L'exécution d'un programme s'effectue en exécutant instruction par instruction jusqu'à la fin du programme. L'exécution d'une instruction se fait à travers trois étapes : (1) recherche et décodage de l'instruction, (2) exécution de l'instruction, et (3) préparation de l'instruction suivante.

La phase de recherche d'instruction est la même pour tous les différents types d'instructions, elle suppose que le CO contient l'adresse de l'instruction :

1. Transférer l'adresse se trouvant dans le CO vers le registre RAM.
2. Commander à la mémoire centrale une lecture de l'information se trouvant dans le mot mémoire indexé par la valeur contenue dans le registre RAM. A l'issue de la commande de lecture, la mémoire centrale va délivrer dans le registre RIM l'instruction demandée.
3. L'instruction va être ensuite conduite vers le registre RI.

La phase de décodage et d'exécution dépend de la nature de l'instruction à exécuter (instruction de calcul, transfert, appel système, etc.). En général, elle se déroule comme suit :

1. Le séquenceur analyse et décode le code opération de l'instruction.

2. Après le décodage, le séquenceur génère en plusieurs étapes dans le temps un ensemble de signaux de commandes qui vont permettre d'activer les circuits concernés par le traitement.
3. La dernière phase est la préparation de la prochaine instruction à exécuter. Cette phase consiste à mettre à jour le contenu de CO par l'adresse de la prochaine instruction à exécuter. Cette adresse soit elle est obtenue par une incrémentation du CO si l'exécution se fait d'une manière séquentielle, ou soit elle provient de l'actuelle instruction s'il s'agit d'une rupture de séquence.

### 1.3 Modes d'adressage

Il y a généralement, plusieurs types de donnée représentés par les opérandes : donnée immédiate stockée dans l'instruction machine, donnée dans un registre de l'unité de contrôle, ou donnée stockée à une adresse en mémoire centrale. L'accès à ces données nécessite l'usage d'un mécanisme d'adressage. Il en existe principalement quatre modes :

- Adressage immédiat : la valeur de la donnée est stockée dans l'instruction. Cette valeur est donc copiée de la mémoire centrale vers l'unité de traitement lors de la phase de chargement de l'instruction.
- Adressage direct : la valeur de la donnée est stockée à une adresse en mémoire centrale. C'est cette adresse qui est représentée dans l'instruction.
- Adressage indirect : la valeur de la donnée est stockée à une adresse $\alpha$ en mémoire centrale. Cette adresse est stockée dans un registre R ou à une autre adresse $\beta$. C'est R ou $\beta$ qui est stocké dans l'instruction ($\beta$ est appelé pointeur). L'adressage indirect par registre est présent dans la totalité des unités de contrôle, par contre, l'adressage indirect par mémoire est moins fréquent.
- Adressage indexé (ou basé) : il est parfois nécessaire d'accéder à des données situées à des adresses consécutives en mémoire centrale. En adressage indexé, on charge un registre d'index avec l'adresse de début de cette zone de données, ensuite on spécifie dans l'instruction le déplacement à réaliser à partir de cet index. L'adresse réelle de la donnée est donc égale à l'adresse de l'index ajouté au déplacement. Certaines unités de contrôle exécutent automatiquement l'incrémentation ou la décrémentation de leurs registres d'index ce qui permet de réaliser des transferts ou d'autres opérations sur des zones (chaînes de caractères).

Référence
- S Bensaid, cours architecture des ordinateurs, université de bejaia, 20014
-Omar mawloud, cours architecture des ordinateurs, université de bejaia, 20014

J-ean pierre Meinadier, ‘structure et fonctionnement des ordinateurs, livre, librairie Larousse, 1971