

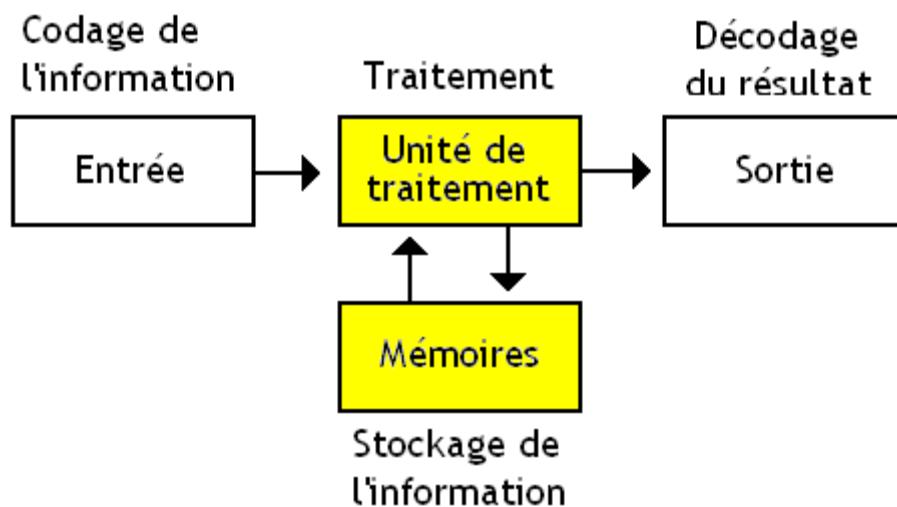
REPRESENTATION DE L'INFORMATION

I. Introduction

Un ordinateur est une machine de traitement de l'information. Il est capable d'acquérir de l'information, de la stocker, de la transformer en effectuant des traitements quelconques, puis de la restituer sous une autre forme. Un ordinateur est un ensemble de composants électroniques modulaires, c'est-à-dire des composants pouvant être remplacés par d'autres composants ayant éventuellement des caractéristiques différentes, capables de faire fonctionner des programmes informatiques.

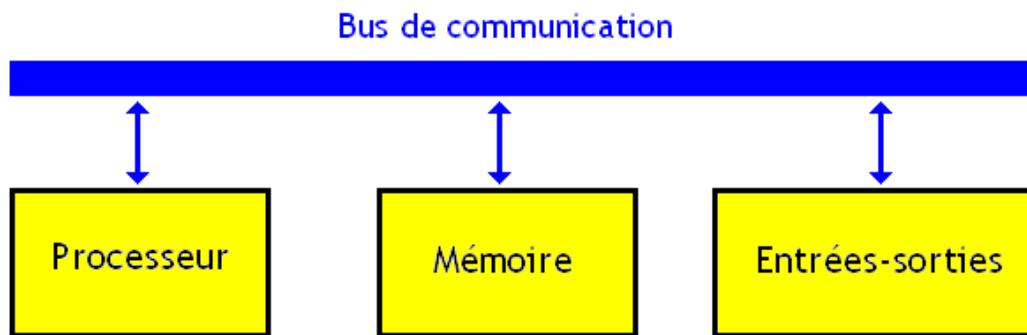
Un ordinateur n'est pas qu'un amoncellement de circuits et est organisé d'une manière bien précise. Il est structuré autour de trois composants principaux :

- Les entrées/sorties, qui permettent à l'ordinateur de communiquer avec l'extérieur
- Une mémoire qui mémorise les données à manipuler
- Un processeur, qui manipule l'information et donne un résultat



Pour faire simple, le processeur est un circuit qui s'occupe de faire des calculs et de traiter des informations. La mémoire s'occupe purement de la mémorisation des informations. Les entrées-sorties permettent au processeur et à la mémoire de communiquer avec l'extérieur et d'échanger des informations avec des périphériques. Tout ce qui n'appartient pas à la liste

du dessus est obligatoirement connecté sur les ports d'entrée-sortie et est appelé périphérique. Ces composants communiquent via un bus, un ensemble de fils électriques qui relie les différents éléments d'un ordinateur.



Parfois, on décide de regrouper la mémoire, le CPU et les ports d'entrée-sortie dans un seul composant électronique nommé microcontrôleur. Dans certains cas, qui sont plus la règle que l'exception, certains périphériques sont carrément inclus dans le microcontrôleur ! On peut ainsi trouver dans ces microcontrôleurs, des compteurs, des générateurs de signaux, des convertisseurs numériques-analogiques, etc. On trouve des microcontrôleurs dans les disques durs, les baladeurs mp3, dans les automobiles, et tous les systèmes embarqués en général. Nombreux sont les périphériques ou les composants internes à un ordinateur qui contiennent des microcontrôleurs.

Le mot « informatique » vient de la contraction des mots « information » et « automatique ». Nous appelons information tout ensemble de données. On distingue généralement différents types d'informations : textes, nombres, sons, images, etc., mais aussi les instructions composant un programme. Une information est une connaissance qui fait sens pour les personnes concernées et qui peut être représentée par des symboles. Bien sûr, la notion de sens n'est pas pertinente pour un ordinateur, qui n'est qu'une machine, et c'est uniquement la représentation de l'information sur laquelle il va agir. Ainsi un traitement d'informations par un ordinateur consiste en des transformations de leurs représentations. Ces transformations peuvent être des calculs, des réagencements, des ajouts, des suppressions, des copies et toute autre opération consistant en des manipulations de symboles. Pour éviter l'expression « représentation d'informations », un peu longue, on parlera parfois de manière équivalente de « donnée ».

II. Architecture de base d'un ordinateur

Il existe deux architectures de base d'un ordinateur : architecture de Von Neumann et l'architecture Harvard.

1. Architecture de Von Neumann

L'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. De telles machines sont aussi connues sous le nom d'ordinateur à programme enregistré. La séparation entre le stockage et le processeur est implicite dans ce modèle.

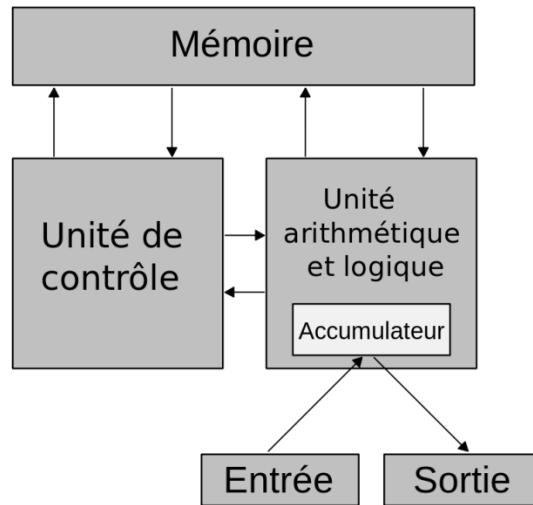
Cette architecture est appelée ainsi en référence au mathématicien John Von Neumann qui a élaboré en juin 1945 dans le cadre du projet EDVAC1 la première description d'un ordinateur dont le programme est stocké dans sa mémoire.

Le modèle de Von Neumann est caractérisé par :

- 1 seule mémoire qui contient les instructions et les données
- 1 bus d'adresse, 1 bus (bidirectionnel) de donnée
- Logique complexe pour lire/écrire tour à tour les données et les instructions

L'architecture de Von Neumann décompose l'ordinateur en 4 parties distinctes :

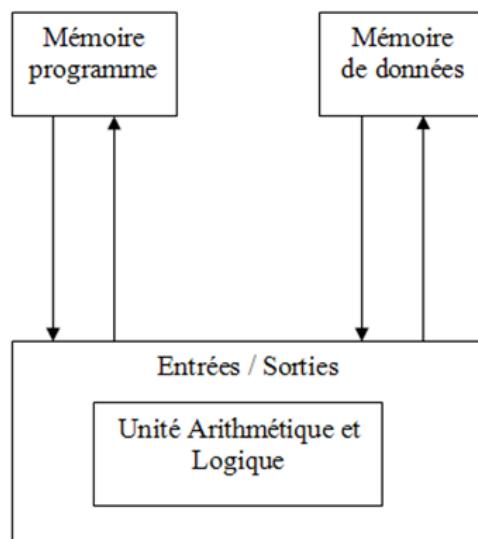
- **L'unité arithmétique et logique** (UAL ou ALU en anglais) ou unité de traitement : son rôle est d'effectuer les opérations de base ;
- **L'unité de contrôle** : chargée du « séquençage » des opérations ;
- **La mémoire** : qui contient à la fois les données et le programme qui indiquera à l'unité de contrôle quels sont les calculs à faire sur ces données. La mémoire se divise entre mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine) ;
- **Les dispositifs d'entrée-sortie** : qui permettent de communiquer avec le monde extérieur.



2. Architecture Harvard

L'architecture de type Harvard est une conception des processeurs qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. Le nom de cette structure vient du nom de l'université Harvard où une telle architecture a été mise en pratique pour la première fois avec le Mark I en 1944.

Avec deux bus distincts, l'architecture dite de Harvard permet de transférer simultanément les données et les instructions à exécuter. Ainsi, l'unité de traitement aura accès simultanément à l'instruction et aux données associées.



Le modèle Harvard est caractérisé par :

- 2 mémoires : 1 pour les instructions, 1 pour les données
- 2 bus d'adresse, 2 bus (bidirectionnels) de donnée

Cette architecture peut se montrer plus rapide à technologie identique que l'architecture de Von Neumann. Le gain en performance s'obtient cependant au prix d'une complexité accrue de structure.

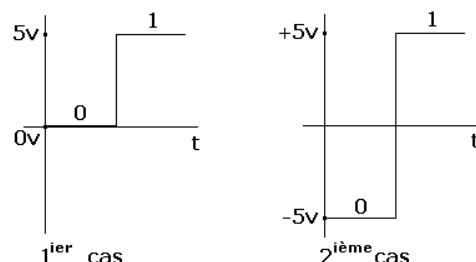
III. Systèmes de numération

L'ordinateur est une machine électronique qui fonctionne avec le courant électrique. Ce dernier possède principalement deux états représentés dans deux cas:

1^{er} cas : Présence ou absence d'une tension

2^{ième} cas : Tension positive ou négative.

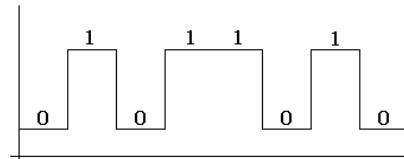
Exemple



De ce fait conventionnellement et quel que soit le cas, il était associé pour le premier état le nombre 1 et pour le deuxième état le nombre 0. Donc, en associant une suite de ces nombres on peut coder les informations dans un ordinateur.

Exemple

Si on a la suite binaire 01011010 sa représentation électrique sera la suivante :



On peut classer les informations dans un ordinateur sous trois ensembles :

- Caractères alphabétique : A...Z, a...z
- Caractères numérique : 0,1,...9
- Caractères spéciaux : +,* , !, ?...

Chaque caractère possède une suite de code (0 et 1) appelée code binaire qui est en réalité une suite de signaux électrique. Ce code binaire contient 8 cases appelées dans ce contexte bits.

Exemple : le caractère A est représenté par la suite binaire 01000001

Remarque : un caractère est un octet donc on a 1 octet (byte) =8 bits

Cette suite de bits appartient à un système de numération dit binaire. Toutefois, il existe plusieurs autres systèmes de numération et il existe aussi des règles de passage entre eux.

Tout système de numération nécessite un ensemble limité de symbole pour représenter n'importe quel nombre. Le nombre de ces symboles constitue la base du système de numération. Ainsi, si un nombre ($a_n a_{n-1} a_{n-2} \dots a_0$) appartient à la base B alors on a a_i dans $\{0, \dots, B-1\}$.

1. Système de numération décimal :

C'est le système le plus utilisé quotidiennement, ses caractéristiques sont les suivantes :

Symbol : {0..9} Base : nombre des symboles est 10

Exemple : $(1012)_{10}$ $(456)_{10}$ $(25)_{10}$

2. Système de numération binaire :

C'est le système utilisé par les ordinateurs pour coder les informations.

Symbol : {0,1} Base : 2

Exemple : $(101)_2$ $(10001)_2$ $(1110001)_2$

3. Système de numération octal :

Utilisé dans le domaine de l'informatique.

Symbol : {0..7} Base : 8

Exemple : $(45)_8$ $(745)_8$ $(6237)_8$

4. Système de numération hexadécimal :

Utilisé aussi dans le domaine de l'informatique.

Symbol : {0..9, A, B, C, D, E, F} avec A=10 jusqu'à F=15 Base : 16

Exemple : $(45A)_{16}$ $(101B)_{16}$ $(637)_{16}$

IV. Conversion des systèmes de numération

En informatique, le système utilisé est le binaire mais dans la vie courante on utilise le système décimale pour cela il existe des opérations qui nous permettent de passer d'une base à une autre.

1. Passage de toutes les bases vers la base décimale :

Si on a le nombre N suivant $(a_n a_{n-1} a_{n-2} \dots a_0)_b$ et on veut le convertir vers la base décimale, on doit appliquer la formule suivante :

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

Avec N : nombre décimal résultat

a : un digit du nombre N

n : ordre du digit dans le nombre N

b : base du nombre N

Exemple :

$$(101)_2 = (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = (5)_{10}$$

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

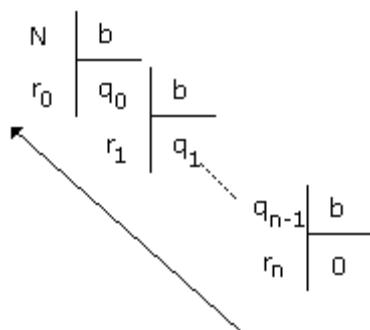
$$(1A)_{16} = (1 * 16^1) + (10 * 16^0) = (26)_{10}$$

2. Passage de la base décimale vers les autres bases :

Si on a un nombre dans la base décimale et on veut le convertir soit vers le binaire, l'octale ou l'hexadécimale, on doit faire une suite de division entière par la base voulue jusqu'à arriver à un résultat égale à 0, puis récupérer tous les restes à partir de la fin.

Soit

- N : le nombre décimal
- b : la base voulue
- q_i : résultat de la division
- r_i : reste de la division



Donc le nombre dans la base b sera : $(r_n \dots r_1 r_0)_b$

Exemple : $(13)_{10} = (1101)_2$

$$(25)_{10} = (31)_8$$

$$(31)_{10} = (1F)_{16}$$

3. Passage entre binaire et octal :

Pour passer du binaire vers l'octal, on doit regrouper les bits du nombre binaire en des paquets de trois bits en commençant à partir de la droite, puis convertir chaque paquet vers le décimal. Les résultats des paquets seront les digits du nouveau nombre octal.

Exemple :

Soit $(\underline{101} \underline{100} \underline{011})_2$

Donc on aura (5 4 3)₈

Remarque : si dans le dernier paquet de gauche on a un nombre de bit inférieur à trois on complète par des zéros.

Pour passer de l'octale vers le binaire on doit faire l'opération inverse, c'est à dire on doit transformer chaque digit du nombre octal en trois bits binaire.

Exemple :

Soit $(7 \quad 3)_8$

$(111 \ 011)_2$

4. Passage entre binaire et hexadécimal :

Pour passer du binaire vers l'hexadécimal, on doit regrouper les bits du nombre binaire en des paquets de quatre bits en commençant à partir de la droite et faire l'opération inverse lorsque on veut passer de l'hexadécimal vers le binaire.

Exemple 1 : $(\underline{1010} \underline{0011} \underline{0110 })_2$

$(A \quad 3 \quad 6)_{16}$

Exemple 2 : $(7 \quad F \quad 1)_{16}$

$(0111 \ 1111 \ 0001)_2$

Remarque : il n'existe pas d'opération directe qui nous permet de passer entre l'octal et l'hexadécimal, pour cela on doit utiliser une base intermédiaire qui sera généralement la base binaire.

La table de correspondance suivante est très importante à apprendre pour faciliter les opérations de conversion entre base.

Base 10	Base 16	Base 2
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

V. Arithmétique binaire

Les diverses opérations arithmétiques qui interviennent dans les ordinateurs et les calculatrices portent sur des nombres exprimés en notation binaire. Dans cette partie, nous allons nous concentrer sur les principes de base qui nous permettent de comprendre comment les machines numériques réalisent les opérations arithmétiques de base en essayant de montrer comment effectuer manuellement ces opérations.

1. Addition binaire

L'addition de deux nombres binaires est parfaitement analogue à l'addition de deux nombres décimaux. En fait, l'addition binaire est plus simple puisqu'il y a moins de cas à apprendre. On commence par additionner les chiffres du rang de poids faible, les chiffres du deuxième rang sont ensuite additionnés, et ainsi de suite. Les mêmes règles s'appliquent à l'addition binaire. Cependant, il n'y a que cinq cas, qui peuvent survenir lorsqu'on additionne deux chiffres binaires et cela quel que soit le rang. Ces cinq cas sont:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{avec une retenue de 1}$$

$$1 + 1 + 1 = 1 \quad \text{avec une retenue de 1}$$

Grâce à ces règles, il devient possible d'effectuer certaines opérations d'addition de la même façon que cela se fait couramment en base dix.

$$\begin{array}{r}
 & 173 \\
 + & 44 \\
 \hline
 217
 \end{array}
 \qquad
 \begin{array}{r}
 & \color{red}{1} \color{black} \color{red}{1} \color{black} \color{red}{1} \\
 + & 1010 \ 1101 \\
 & 0010 \ 1100 \\
 \hline
 1101 \ 1001
 \end{array}$$

$$\begin{array}{r}
 & 190 \\
 + & 141 \\
 \hline
 331
 \end{array}
 \qquad
 \begin{array}{r}
 & \color{red}{1} \color{black} \color{red}{1} \color{black} \color{red}{1} \\
 + & 1011 \ 1110 \\
 & 1000 \ 1101 \\
 \hline
 1\ 0100 \ 1011
 \end{array}$$

2. Représentation des nombres négatifs

a. Représentation des nombres négatifs par signe

Un entier relatif sera représenté en binaire comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe : 0 correspond à un signe positif, 1 à un signe négatif. Ainsi, si on code un entier naturel sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).

Exemple : représentation sur 8 bits. Le bit situé à l'extrême gauche est le bit de signe.

$$(+19)_{10} = (0001\ 0011)_2$$

$$(-19)_{10} = (\textcolor{red}{1}001\ 0011)_2$$

b. Représentation des nombres négatifs par complément à deux

Le complément est une méthode qui permet de représenter les nombres négatifs en utilisant un codage qui suit le principe suivant :

1. Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.
2. Inverser les bits : les 0 deviennent des 1 et vice versa. On fait ce qu'on appelle le complément à un.
3. On ajoute 1 au résultat (les dépassemens sont ignorés).

Exemple : On désire coder la valeur -19 sur 8 bits, il suffit :

1. Écrire 19 en binaire : 00010011
2. Écrire son complément à 1 : 11101100
3. Ajouter 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 zéro.

En effet, $00010011 + 11101101 = 00000000$ (avec une retenue de 1 qui est éliminée).

3. Soustraction binaire

La soustraction de deux nombres binaires est réalisée en suivant les règles suivantes :

$$0 - 0 = 0$$

$$0 - 1 = 1 \quad \text{avec un emprunt de 1}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Exemple :

$$\begin{array}{r} 173 \\ - 44 \\ \hline 129 \end{array} \qquad \begin{array}{r} 1010\ 1101 \\ - 0010\ 1100 \\ \hline 1000\ 0001 \end{array}$$

$$\begin{array}{r} 190 \\ - 141 \\ \hline 49 \end{array} \qquad \begin{array}{r} 1011\ 1110 \\ - 1000\ 1101 \\ \hline \ 1 \\ 0011\ 0001 \end{array}$$

On peut faire la soustraction binaire en utilisant le complément à deux.

$$A - B \rightarrow A + (-B)$$

Donc, la soustraction devient une addition avec le complément à deux du nombre négatif.

$$\begin{array}{r} 173 \\ + (-44) \\ \hline 129 \end{array} \qquad \begin{array}{r} 1010\ 1101 \\ + (-0010\ 1100) \\ \hline 1000\ 0001 \end{array} \qquad \begin{array}{r} 1111\ 1 \\ + 1010\ 1101 \\ + 1101\ 0100 \\ \hline \boxed{1} 1000\ 0001 \\ \text{Eliminée} \end{array}$$

4. Multiplication binaire

La multiplication de deux nombres binaires est réalisée en suivant les règles suivantes :

$$0 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 0 = 0$$

$$1 * 1 = 1$$

Exemple :

$$\begin{array}{r}
 * \quad 9 \\
 * \quad 5 \\
 \hline
 45
 \end{array}
 \qquad
 \begin{array}{r}
 * \quad 1011 \\
 * \quad 101 \\
 \hline
 1011 \\
 + \quad 0000 \cdot \\
 + \quad 1011 \dots \\
 \hline
 110111
 \end{array}$$

5. Division binaire

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux. En réalité, la division binaire est plus simple pour déterminer combien de fois le diviseur entre dans le dividende. Il n'y a que deux possibilités soit 0 ou 1.

Exemple : on veut diviser 74 par 5

$$\begin{array}{r}
 1001010 \\
 - 101 \\
 \hline
 01000 \\
 - 101 \\
 \hline
 00111 \\
 - 101 \\
 \hline
 0100
 \end{array}
 \qquad
 \begin{array}{r}
 101 \\
 \hline
 1110
 \end{array}$$

Le résultat est $(1110)_2$ qui $(14)_{10}$ et le reste est $(100)_2$ qui est $(4)_{10}$

VI. Représentation des nombres fractionnaires

1. Représentation en virgule fixe

Un nombre réel est constitué de deux parties : la partie entière et la partie fractionnelle. Si on a le nombre réel N suivant :

$$N = \underbrace{(a_n a_{n-1} a_{n-2} \dots a_0)}_{\text{Partie entière}}, \underbrace{a_{-1} a_{-2} \dots a_{-m}}_b$$

composé de n digits dans la partie entière et m digits dans la partie fractionnelle et on veut le convertir vers la base décimale, on doit appliquer la méthode polynomial :

$$N = (a_n b^n) + (a_{n-1} b^{n-1}) + \dots + (a_0 b^0) + (a_{-1} b^{-1}) + \dots + (a_{-m} b^{-m})$$

Exemple 1

Représentation du nombre binaire $(100,101)_2$

$$1 * 2^2 + 0 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = (4,625)_{10}$$

Pour le passage de la base décimale vers les autres bases, on doit :

1. La partie entière est transformée en effectuant des divisions successives par la base.
2. La partie fractionnelle est transformée en effectuant des multiplications successives par la base.

Exemple 2

On veut convertir $(35,625)_{10}$ vers la base 2.

$$\text{Partie entière} = (35)_{10} = (100011)_2$$

$$\text{Partie fractionnelle} = (0,625)_{10}$$

$$\begin{array}{rcl}
 0,625 & * 2 & = 1,25 \\
 0,25 & * 2 & = 0,5 \\
 0,5 & * 2 & = 1,0 \\
 \end{array}$$

↓

$(0,625)_{10} = (0,101)_2$

$$\text{Donc } (35,625)_{10} = (100011,101)_2$$

Exemple 3

On veut convertir le nombre $(738,49)_{10}$ vers la base 2 avec une précision de 0.005. Donc si on a m digits dans la partie fractionnelle, alors :

$$\begin{aligned} 2^{-m} &\leq 0.005 \\ \log 2^{-m} &\leq \log 0.005 \\ -m \log 2 &\leq \log 0.005 \\ m &\geq \frac{-\log 0.005}{\log 2} \\ m &\geq 7.64 \text{ on va prendre } m = 8 \end{aligned}$$

$$\text{P.E.} = (738)_{10} = (10\ 1110\ 0010)_2$$

$$\text{P.F.} = (0,49)_{10}$$

$$\begin{aligned} 0.49 * 2 &= 0,98 \\ 0.98 * 2 &= 1,96 \\ 0.96 * 2 &= 1,92 \\ 0.92 * 2 &= 1,84 \\ 0.84 * 2 &= 1,68 \\ 0.68 * 2 &= 1,36 \\ 0.36 * 2 &= 0,72 \\ 0.72 * 2 &= 1,44 \end{aligned}$$

↓

$$(0,49)_{10} = (0,01111101)_2$$

→

Donc $(738,49)_{10} = (10\ 1110\ 0010, 01111101)_2$ avec une précision de 0.005.

2. Représentation en virgule flottante

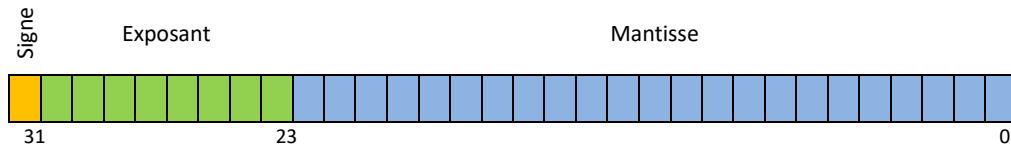
Il y a plusieurs représentations d'un même nombre :

$$(71)_{10} = 7,1 * 10 = 0,71 * 10^2$$

$$(101,1)_2 = 0,1011 * 2^3 = 1011 * 2^{-1} = 1,011 * 2^2$$

La norme IEEE 754 simple précision définit la façon de coder un nombre réel. Cette norme se propose de coder le nombre sur 32 bits et définit trois composants :

- Le signe est représenté par un seul bit de poids fort.
- L'exposant est codé sur les 8 bits consécutifs au signe.
- La mantisse (les bits situés après la virgule) sur les 23 bits restants.



Un nombre binaire N est écrit en virgule flottante comme suit :

$$N = \text{signe} (1, \text{mantisse}) * 2^{\text{exposant}}$$

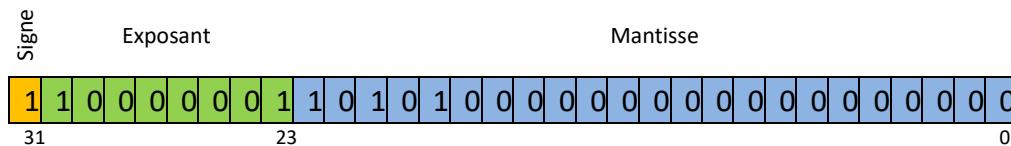
Certaines conditions sont toutefois à respecter pour les exposants :

- L'exposant 0000 0000 est interdit.
- L'exposant 1111 1111 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie « Not a number ».
- Il faut rajouter 127 (0111 1111) à l'exposant pour conversion de décimal vers un nombre réel binaire. Les exposants peuvent aller de -254 à 255.

Exemple

On veut coder le nombre $N = (-6,625)_{10}$ en utilisant la norme IEEE 754 :

- On code la valeur absolue en binaire : $(6,625)_{10} = (110,1010)_2$
- On transforme le nombre sous la forme : 1, partie fractionnaire
- $110,1010 = 1,101010 * 2^2$
- La partie fractionnaire étendue sur 23 bits est donc 101 0100 0000 0000 0000 0000.
- Exposant = $127 + 2 = (129)_{10} = (1000 0001)_2$



VII. Codage ASCII

Le code ASCII (American Standard Code for information interchange) est un code alphanumérique, devenu une norme internationale. Il est utilisé pour la transmission entre ordinateurs ou entre un ordinateur et des périphériques. Sous sa forme standard, il utilise 7 bits. Ce qui permet de générer $2^7=128$ caractères. Ce code représente les lettres alphanumériques majuscules et minuscules, les chiffres décimaux, des signes de ponctuation et des caractères de commande.

Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que :

- Retour à la ligne (CR)
- Bip sonore (BEL)
- Les codes 65 à 90 représentent les majuscules
- Les codes 97 à 122 représentent les minuscules

Il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère il faut recourir à un autre code. Le code ASCII a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...).

Ce code attribue les valeurs 0 à 255 (donc codées sur 8 bits, soit 1 octet) aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (caractères accentués dans le cas du code iso-latin1).

ASCII TABLE

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

ARITHMETIQUE BINAIRE

I. Introduction

Les diverses opérations arithmétiques qui interviennent dans les ordinateurs et les calculatrices portent sur des nombres exprimés en notation binaire. Dans cette partie, nous allons nous concentrer sur les principes de base qui nous permettent de comprendre comment les machines numériques réalisent les opérations arithmétiques de base en essayant de montrer comment effectuer manuellement ces opérations.

II. Addition binaire

L'addition de deux nombres binaires est parfaitement analogue à l'addition de deux nombres décimaux. En fait, l'addition binaire est plus simple puisqu'il y a moins de cas à apprendre. On commence par additionner les chiffres du rang de poids faible, les chiffres du deuxième rang sont ensuite additionnés, et ainsi de suite. Les mêmes règles s'appliquent à l'addition binaire. Cependant, il n'y a que cinq cas, qui peuvent survenir lorsqu'on additionne deux chiffres binaires et cela quel que soit le rang. Ces cinq cas sont:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{avec un retenu de 1}$$

$$1 + 1 + 1 = 1 \quad \text{avec un retenu de 1}$$

Grâce à ces règles, il devient possible d'effectuer certaines opérations d'addition de la même façon que cela se fait couramment en base dix.

$$\begin{array}{r}
 + 173 \\
 44 \\
 \hline
 217
 \end{array}
 \quad
 \begin{array}{r}
 + \textcolor{red}{1\ 1\ 1} \\
 1010\ 1101 \\
 0010\ 1100 \\
 \hline
 1101\ 1001
 \end{array}$$

$$\begin{array}{r}
 + 190 \\
 141 \\
 \hline
 331
 \end{array}
 \quad
 \begin{array}{r}
 + \textcolor{red}{111\ 1} \\
 1011\ 1110 \\
 1000\ 1101 \\
 \hline
 1\ 0100\ 1011
 \end{array}$$

III. Représentation des nombres négatifs

1. Représentation des nombres négatifs par signe

Un entier relatif sera représenté en binaire comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe : 0 correspond à un signe positif, 1 à un signe négatif. Ainsi, si on code un entier naturel sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).

Exemple : représentation sur 8 bits. Le bit situé à l'extrême gauche est le bit de signe.

$$(+19)_{10} = (\textcolor{red}{0}001\ 0011)_2$$

$$(-19)_{10} = (\textcolor{red}{1}001\ 0011)_2$$

Cette représentation peut causer des calculs erronés. Si on fait la somme de ses deux nombres :

$$\begin{array}{r}
 (+19)_{10} \\
 + (-19)_{10} \\
 \hline
 (0)_{10}
 \end{array}
 \neq
 \begin{array}{r}
 (\textcolor{red}{0}001\ 0011)_2 \\
 + (\textcolor{red}{1}001\ 0011)_2 \\
 \hline
 (\textcolor{red}{1}010\ 0110)_2
 \end{array}$$

Pour résoudre ce problème, on doit coder les nombres négatifs en utilisant le complément à deux.

2. Représentation des nombres négatifs par complément à deux

Le complément est une méthode qui permet de représenter les nombres négatifs en utilisant un codage qui suit le principe suivant :

1. Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.
2. Inverser les bits : les 0 deviennent des 1 et vice versa. On fait ce qu'on appelle le complément à un.
3. On ajoute 1 au résultat (les dépassemens sont ignorés).

Exemple : On désire coder la valeur -19 sur 8 bits, il suffit :

1. Écrire 19 en binaire : 00010011
2. Écrire son complément à 1 : 11101100
3. Ajouter 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 zéro. En effet, $00010011 + 11101101 = 00000000$ (avec un retenu de 1 qui est éliminé).

IV. Soustraction binaire

La soustraction de deux nombres binaires est réalisée en suivant les règles suivantes :

$$0 - 0 = 0$$

$$0 - 1 = 1 \quad \text{avec un emprunt de 1}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Exemple

$$\begin{array}{r}
 173 \\
 - 44 \\
 \hline
 129
 \end{array}
 \quad
 \begin{array}{r}
 1010\ 1101 \\
 - 0010\ 1100 \\
 \hline
 1000\ 0001
 \end{array}$$

$$\begin{array}{r}
 190 \\
 - 141 \\
 \hline
 49
 \end{array}
 \quad
 \begin{array}{r}
 1011\ 1110 \\
 - 1000\ 1101 \\
 \hline
 \textcolor{blue}{1} \\
 0011\ 0001
 \end{array}$$

On peut faire la soustraction binaire en utilisant le complément à deux.

$$A - B \rightarrow A + (-B)$$

Donc, la soustraction devient une addition avec le complément à deux du nombre négatif.

$$\begin{array}{r}
 + \begin{array}{r} 173 \\ (-44) \end{array} & + \begin{array}{r} 1010\ 1101 \\ (-0010\ 1100) \end{array} & + \begin{array}{r} 1111\ 1 \\ 1010\ 1101 \\ 1101\ 0100 \\ \hline 1 \end{array} \\
 \hline 129 & 1000\ 0001 & 1000\ 0001
 \end{array}$$

Eliminé

Pour que le calcul soit exact dans tous les cas, on doit ajouter le bit de signe à chaque nombre. Si le signe du nombre obtenu est négatif (c'est-à-dire égale à 1) alors on doit calculer le complément à deux (Comp_2) du nombre obtenu pour trouver le résultat.

Exemple

Les opérations suivantes sont faites sur 5 bits (le bit poids fort est le signe et 4 bits pour les données). Si le nombre est négatif alors on prend son Comp₂ puis on fait une addition.

$$\begin{array}{r}
 +9 & & 0 & 1001 \\
 +4 & + & 0 & 0100 \\
 \hline
 +13 & & 0 & 1101 \\
 & & \text{Positif} &
 \end{array}
 \qquad
 \begin{array}{r}
 +9 & & 0 & 1001 \\
 -4 & & 1 & 1100 \\
 \hline
 +5 & & \boxed{1} & 0101 \\
 & & \text{Eliminé} & \text{Positif}
 \end{array}$$

Si le nombre obtenu est négatif (bit de signe égal à 1) alors il faut calculer son Comp₂ pour trouver le résultat.

$$\begin{array}{r}
 -9 \\
 -4 \\
 \hline
 -13
 \end{array}
 \quad
 \begin{array}{r}
 -(1001) \\
 -(0100) \\
 \hline
 \text{Eliminé Négatif}
 \end{array}
 \quad
 \begin{array}{r}
 + 1\ 0111 \\
 + 1\ 1100 \\
 \hline
 \boxed{1} \ 1\ 0011
 \end{array}$$

Résultat = - Comp₂(1 0011) = -(01101) = -(13)₁₀

V. Multiplication binaire

La multiplication de deux nombres binaires est réalisée en suivant les règles suivantes :

$$0 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 0 = 0$$

$$1 * 1 = 1$$

Exemple

$$\begin{array}{r}
 * \begin{array}{r} 11 \\ 5 \end{array} \\
 \hline
 55
 \end{array}
 \quad
 \begin{array}{r}
 * \begin{array}{r} 1011 \\ 101 \end{array} \\
 \hline
 \begin{array}{r}
 + \begin{array}{r} 1011 \\ 0000 \end{array} \\
 + \begin{array}{r} 1011 \\ \dots \end{array} \\
 \hline
 110111
 \end{array}
 \end{array}$$

VI. Division binaire

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux. En réalité, la division binaire est plus simple pour déterminer combien de fois le diviseur entre dans le dividende. Il n'y a que deux possibilités soit 0 ou 1.

Exemple : on veut diviser 74 par 5

$$\begin{array}{r}
 \begin{array}{r}
 1001010 \\
 - 101 \\
 \hline
 01000 \\
 - 101 \\
 \hline
 00111 \\
 - 101 \\
 \hline
 0100
 \end{array}
 \quad \left| \begin{array}{l} 101 \\ 1110 \end{array} \right.
 \end{array}$$

Le résultat est $(1110)_2$ qui est $(14)_{10}$ et le reste est $(100)_2$ qui est $(4)_{10}$

VII. Représentation des nombres fractionnaires

1. Représentation en virgule fixe

Un nombre réel est constitué de deux parties : la partie entière et la partie fractionnelle. Si on a le nombre réel N suivant :

$$N = (\underbrace{a_n a_{n-1} a_{n-2} \dots a_0}_\text{Partie entière}, \underbrace{a_{-1} a_{-2} \dots a_{-m}}_\text{Partie fractionnelle})_b$$

N est composé de $n+1$ digits dans la partie entière et m digits dans la partie fractionnelle et on veut le convertir vers la base décimale, on doit appliquer la méthode polynomiale :

$$N = (a_n b^n) + (a_{n-1} b^{n-1}) + \dots + (a_0 b^0) + (a_{-1} b^{-1}) + \dots + (a_{-m} b^{-m})$$

Exemple 1

Représentation du nombre binaire $(100,101)_2$

$$1*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = (4,625)_{10}$$

Pour le passage de la base décimale vers les autres bases, on doit :

1. La partie entière est transformée en effectuant des divisions successives par la base.
2. La partie fractionnelle est transformée en effectuant des multiplications successives par la base.

Exemple 2

On veut convertir $(35,625)_{10}$ vers la base 2.

Partie entière = $(35)_{10} = (100011)_2$

Partie fractionnelle = $(0,625)_{10}$

$$\begin{array}{rcl}
 0,625 * 2 & = & 1,25 \\
 0,25 * 2 & = & 0,5 \\
 0,5 * 2 & = & 1,0 \\
 (0,625)_{10} = (0,101)_2
 \end{array}$$

Donc $(35,625)_{10} = (100011,101)_2$

Exemple 3

On veut convertir le nombre $(738,49)_{10}$ vers la base 2 avec une précision de 0.005. Donc si on a m digits dans la partie fractionnelle, alors :

$$\begin{aligned}
 2^{-m} &\leq 0.005 \\
 \log 2^{-m} &\leq \log 0.005 \\
 -m \log 2 &\leq \log 0.005 \\
 m &\geq \frac{-\log 0.005}{\log 2} \\
 m &\geq 7.64 \text{ on va prendre } m = 8
 \end{aligned}$$

P.E. = $(738)_{10} = (1011100010)_2$

P.F. = $(0,49)_{10}$

$$\begin{array}{rcl}
 0.49 * 2 = 0,98 & & \\
 0.98 * 2 = 1,96 & & \\
 0.96 * 2 = 1,92 & & \\
 0.92 * 2 = 1,84 & & \\
 0.84 * 2 = 1,68 & & \\
 0.68 * 2 = 1,36 & & \\
 0.36 * 2 = 0,72 & & \\
 0.72 * 2 = 1,44 & & \\
 (0,49)_{10} = (0,01111101)_2
 \end{array}$$

Donc $(738,49)_{10} = (1011100010,01111101)_2$ avec une précision de 0.005.

2. Représentation en virgule flottante

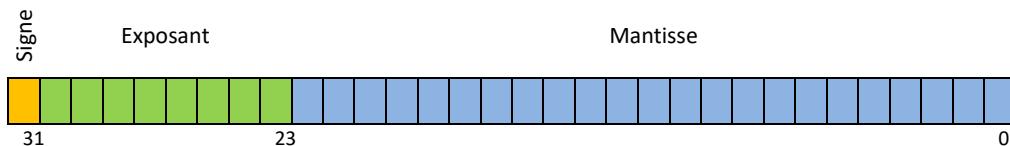
Il y a plusieurs représentations d'un même nombre :

$$(71)_{10} = 7,1 * 10 = 0,71 * 10^2$$

$$(101,1)_2 = 0,1011 * 2^3 = 1011 * 2^{-1} = 1,011 * 2^2$$

La norme IEEE 754 simple précision définit la façon de coder un nombre réel. Cette norme se propose de coder le nombre sur 32 bits et définit trois composants :

- Le signe est représenté par un seul bit de poids fort.
- L'exposant est codé sur les 8 bits consécutifs au signe.
- La mantisse (les bits situés après la virgule) sur les 23 bits restants.



Un nombre binaire N est écrit en virgule flottante comme suit :

$$N = \text{signe} (1, \text{mantisse}) * 2^{\text{exposant}}$$

Certaines conditions sont toutefois à respecter pour les exposants :

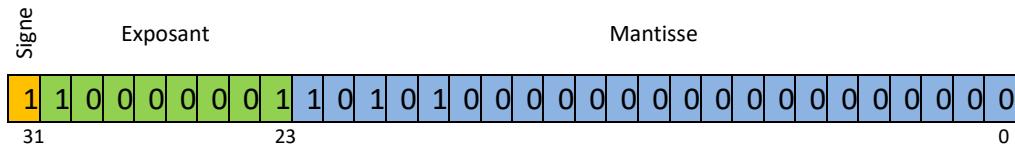
- L'exposant 0000 0000 est interdit.
- L'exposant 1111 1111 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie « Not a number ».
- Il faut rajouter 127 (0111 1111) à l'exposant pour conversion de décimal vers un nombre réel binaire. Les exposants peuvent aller de -128 à 127.

Exemple

On veut coder le nombre $N = (-6,625)_{10}$ en utilisant la norme IEEE 754 :

- On code la valeur absolue en binaire : $(6,625)_{10} = (110,101)_2$
- On transforme le nombre sous la forme : 1, partie fractionnaire

- $110,101 = 1,10101 \times 2^2$
 - La partie fractionnaire étendue sur 23 bits est donc 101 0100 0000 0000 0000 0000.
 - Exposant = $127 + 2 = (129)_{10} = (1000\ 0001)_2$



VIII. Codage ASCII

Le code ASCII (American Standard Code for Information Interchange) est un code alphanumérique, devenu une norme internationale. Il est utilisé pour la transmission entre ordinateurs ou entre un ordinateur et des périphériques. Sous sa forme standard, il utilise 7 bits. Ce qui permet de générer $2^7=128$ caractères. Ce code représente les lettres alphanumériques majuscules et minuscules, les chiffres décimaux, des signes de ponctuation et des caractères de commande.

Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que :

- Retour à la ligne (CR)
 - Bip sonore (BEL)
 - Les codes 65 à 90 représentent les majuscules
 - Les codes 97 à 122 représentent les minuscules

Il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère il faut recourir à un autre code. Le code ASCII a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...).

Ce code attribue les valeurs 0 à 255 (donc codées sur 8 bits, soit 1 octet) aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (caractères accentués dans le cas du code iso-latin1).

ASCII TABLE

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

ARITHMÉTIQUE BINAIRE

CODAGE EN BINAIRE

L'ordinateur code les informations sous **forme binaire**

❑ Les **opérations Arithmétique** sont réalisées en binaire:

- Addition
- Soustraction
- Multiplication
- Division

❑ La représentation des **nombres négatif**

❑ La représentation des **nombres à virgule** (virgule fixe ou flottante)

❑ Le **codage des données** que manipule l'ordinateur (Codage ASCII)

PRINCIPE DE L'ADDITION

- Elle est analogue à l'addition de deux nombres décimaux
- On commence par additionner les chiffres du rang de poids faible, les chiffres du deuxième rang sont ensuite additionnés, et ainsi de suite

Il y a cinq cas possible :

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{avec un retenu de 1}$$

$$1 + 1 + 1 = 1 \quad \text{avec un retenu de 1}$$

EXEMPLES

Règles

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{avec un retenu de 1}$$

$$1 + 1 + 1 = 1 \quad \text{avec un retenu de 1}$$

$$\begin{array}{r} 173 \\ + 44 \\ \hline 217 \end{array}$$

$$\begin{array}{r} 1\ 1\ 1 \\ + 1010\ 1101 \\ 0010\ 1100 \\ \hline 1101\ 1001 \end{array}$$

$$\begin{array}{r} 190 \\ + 141 \\ \hline 331 \end{array}$$

$$\begin{array}{r} 111\ 1 \\ + 1011\ 1110 \\ 1000\ 1101 \\ \hline 1\ 0100\ 1011 \end{array}$$

REPRÉSENTATION PAR SIGNE

- Un entier relatif sera représenté en binaire **comme un entier naturel**,
- Le **bit de poids fort** (le bit situé à l'extrême gauche) représente le **signe** :
 - 0 ➔ signe positif
 - 1 ➔ signe négatif

Exemple

Coder un entier relatif sur 4 bits:



REPRÉSENTATION PAR SIGNE

Exemple

Représentation sur 8 bits: Le bit situé à l'extrême gauche est le bit de signe

$$(+19)_{10} = (0\ 001\ 0011)_2$$

$$(-19)_{10} = (1\ 001\ 0011)_2$$

Si on fait l'addition de ces deux nombres, on doit trouver 0 :

$(+19)_{10}$	$(0001\ 0011)_2$
+	+
$(-19)_{10}$	$(1001\ 0011)_2$
<hr/>	<hr/>
$(0)_{10}$	\neq
	$(1010\ 0110)_2$



On doit coder les nombres négatifs en utilisant le complément à deux

COMPLÉMENT À DEUX

Règle

- Écrire la **valeur absolue** du nombre en base 2 (bit de poids fort doit être égal à 0)
- Inverser les **bits** (complément à un)
- On **ajoute 1** au résultat

$$\text{Comp}_2(N) = \text{Comp}_1(N) + 1$$

COMPLÉMENT À DEUX**Exemple**

On désire coder la valeur -19 sur 8 bits, il suffit :

1. Écrire 19 en binaire :

00010011

2. Écrire son complément à 1 :

11101100

3. Ajouter 1 :

11101101

$(-19)_{10}$ sur 8 bits $\rightarrow (11101101)_2$

$ \begin{array}{r} + (19)_{10} \\ + (-19)_{10} \\ \hline (0)_{10} \end{array} $	$ \begin{array}{r} + \overset{1}{\cancel{1}} \overset{1}{\cancel{1}} \overset{1}{\cancel{1}} \overset{1}{\cancel{1}} \overset{1}{\cancel{1}} \\ + 00010011 \\ \hline \boxed{1}00000000 \end{array} $ <p style="margin-top: 10px;">Eliminé</p>
--	--

PRINCIPE DE LA SOUSTRACTION

Il y a quatre cas possible :

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ avec un emprunt de } 1$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

EXEMPLE 1

Soustraction avec des nombres positifs et le résultat est positif

Règles

$$0 - 0 = 0$$

0 - 1 = 1 avec un emprunt de 1

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$\begin{array}{r} 173 \\ - 44 \\ \hline 129 \end{array}$	$\begin{array}{r} 1010\ 1101 \\ - 0010\ 1100 \\ \hline 1000\ 0001 \end{array}$
$\begin{array}{r} 190 \\ - 141 \\ \hline 49 \end{array}$	$\begin{array}{r} 1011\ 1110 \\ - 1000\ 1101 \\ \hline \quad \quad \quad 1 \\ 0011\ 0001 \end{array}$

EXEMPLE 2

Soustraction :

$$\begin{array}{r}
 173 \\
 - 44 \\
 \hline
 129
 \end{array}
 \quad
 \begin{array}{r}
 1010\ 1101 \\
 - 0010\ 1100 \\
 \hline
 1000\ 0001
 \end{array}$$

Addition avec le complément à deux (sans bit de signe)

$$\begin{array}{r}
 + 173 \\
 + (-44) \\
 \hline
 129
 \end{array}
 \quad
 \begin{array}{r}
 + 1010\ 1101 \\
 + (-0010\ 1100) \\
 \hline
 1000\ 0001
 \end{array}
 \quad
 \begin{array}{r}
 1111\ 1 \\
 + 1010\ 1101 \\
 + 1101\ 0100 \\
 \hline
 \boxed{1} 1000\ 0001
 \end{array}$$

Eliminé

EXEMPLE 3

Pour que le calcul soit exact dans tous les cas

- On doit ajouter le **bit de signe** à chaque nombre

$ \begin{array}{r} +9 \\ +4 \\ \hline +13 \end{array} $	$ \begin{array}{r} +0\ 1001 \\ +0\ 0100 \\ \hline 0\ 1101 \end{array} $ <p style="color: green;">Positif</p>	$ \begin{array}{r} +9 \\ -4 \\ \hline +5 \end{array} $ $ \begin{array}{r} 0\ 1001 \\ 1\ 1100 \\ \hline 10\ 0101 \end{array} $ <p style="color: red;">Eliminé</p> <p style="color: green;">Positif</p>
--	---	--

- Si le signe du **résultat obtenu est négatif** (c'est-à-dire égale à 1) alors on doit calculer le **complément à deux** du nombre obtenu pour trouver le résultat final

$ \begin{array}{r} -9 \\ -4 \\ \hline -13 \end{array} $	$ \begin{array}{r} -(1001) \\ -(0100) \\ \hline \end{array} $	$ \begin{array}{r} +1\ 0111 \\ +1\ 1100 \\ \hline \end{array} $
$ \begin{array}{r} \boxed{1}\ 1\ 0011 \end{array} $		
Eliminé Négatif		
Résultat = - Comp₂(1 0011)=-(01101)=-(13)₁₀		

PRINCIPE DE LA MULTIPLICATION

La multiplication de deux nombres binaires est réalisée en suivant les règles suivantes :

$$0 * 0 = 0$$

$$0 * 1 = 0$$

$$1 * 0 = 0$$

$$1 * 1 = 1$$

Exemple

$$\begin{array}{r} \begin{array}{r} * \quad 11 \\ \quad 5 \\ \hline \quad 55 \end{array} & \begin{array}{r} * \quad 1011 \\ \quad 101 \\ \hline \end{array} \\ + \quad 1011 \\ + \quad 0000 \\ + \quad 1011.. \\ \hline 110111 \end{array}$$

PRINCIPE DE LA DIVISION

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

En réalité, la division binaire est plus simple pour déterminer combien de fois le diviseur entre dans le dividende: il n'y a que deux possibilité soit 0 ou 1.

Exemple

On veut diviser 74 par 5 :

- Le résultat est $(1110)_2$ qui $(14)_{10}$
- Le reste est $(100)_2$ qui est $(4)_{10}$

$ \begin{array}{r} 1001010 \\ - 101 \\ \hline 01000 \\ - 101 \\ \hline 00111 \\ - 101 \\ \hline 0100 \end{array} $	101 $\underline{\quad}$ 1110
---	--

REPRÉSENTATION EN VIRGULE FIXE

Un nombre réel est constitué de deux parties :

$$N = (a_n \ a_{n-1} \ a_{n-2} \dots \ a_0, \ a_{-1} \ a_{-2} \ \dots \ a_{-m})_b$$

Partie entière
Partie fractionnelle

Passage vers la base 10 : Appliquer la méthode polynomial

$$N = (a_n b^n) + (a_{n-1} b^{n-1}) + \dots + (a_0 b^0) + (a_{-1} b^{-1}) + \dots + (a_{-m} b^{-m})$$

REPRÉSENTATION EN VIRGULE FIXE

Exemple 1 : Passage vers la base 10

Représentation du nombre binaire ($100,101$)₂

$$1*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = (4,625)_{10}$$

Passage de la base décimale vers les autres bases:

- La **partie entière** est transformée en effectuant des **divisions successives** par la base.
- La **partie fractionnelle** est transformée en effectuant des **multiplications successives** par la base.

REPRÉSENTATION EN VIRGULE FIXE**Exemple 2: Passage vers la base 2**

On veut convertir $(35,625)_{10}$ vers la base 2 :

➤ Partie entière = $(35)_{10} = (100011)_2$

➤ Partie fractionnelle = $(0,625)_{10}$

$$0,625 * 2 = \boxed{1},25$$

$$0,25 * 2 = \boxed{0},5$$

$$0,5 * 2 = \boxed{1},0$$



$$(0,625)_{10} = (0,101)_2$$

$$(35,625)_{10} = (100011,101)_2$$

Exemple 3

On veut convertir le nombre $(738,49)_{10}$ vers la base 2 avec une précision de 0.005

Si on a m digits dans la partie fractionnelle, alors :

$$\begin{aligned} 2^{-m} &\leq 0.005 \\ \log 2^{-m} &\leq \log 0.005 \\ -m \log 2 &\leq \log 0.005 \\ m &\geq \frac{-\log 0.005}{\log 2} \end{aligned}$$

$$m \geq 7.64 \text{ on va prendre } m = 8$$

REPRÉSENTATION EN VIRGULE FIXE

Exemple 3

$$P.E. = (738)_{10} = (10\ 1110\ 0010)_2$$

$$P.F. = (0,49)_{10}$$

$$0.49 * 2 = \boxed{0},98$$

$$0.98 * 2 = \boxed{1},96$$

$$0.96 * 2 = \boxed{1},92$$

$$0.92 * 2 = \boxed{1},84$$

$$0.84 * 2 = \boxed{1},68$$

$$0.68 * 2 = \boxed{1},36$$

$$0.36 * 2 = \boxed{0},72$$

$$0.72 * 2 = \boxed{1},44$$

$$(0,49)_{10} = (0,01111101)_2$$

Donc $(738,49)_{10} = (10\ 1110\ 0010, 01111101)_2$ avec une précision de 0.005

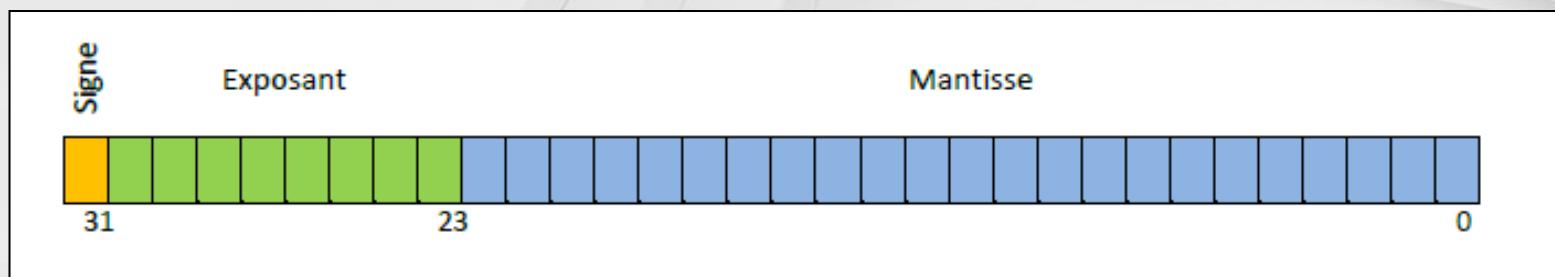
REP. EN VIRGULE FLOTTANTE

Il y a plusieurs représentations d'un même nombre :

$$(71)_{10} = 7,1 * 10 = 0,71 * 10^2$$

$$(101,1)_2 = 0,1011 * 2^3 = 1011 * 2^{-1} = 1,011 * 2^2$$

La **norme IEEE 754** simple précision (32 bits) définit la façon de coder un nombre réel :



REP. EN VIRGULE FLOTTANTE

Un nombre binaire N est écrit en **virgule flottante** comme suit :

$$N = \text{signe} (1, \text{mantisso}) * 2^{\text{exposant}}$$

Certaines conditions sont toutefois à respecter pour les exposants :

- L'exposant 0000 0000 est interdit.
- L'exposant 1111 1111 est interdit
- Il faut rajouter 127 (0111 1111) à l'exposant pour conversion de décimal vers un nombre réel binaire. Les exposants peuvent aller de -254 à 255.

Exemple

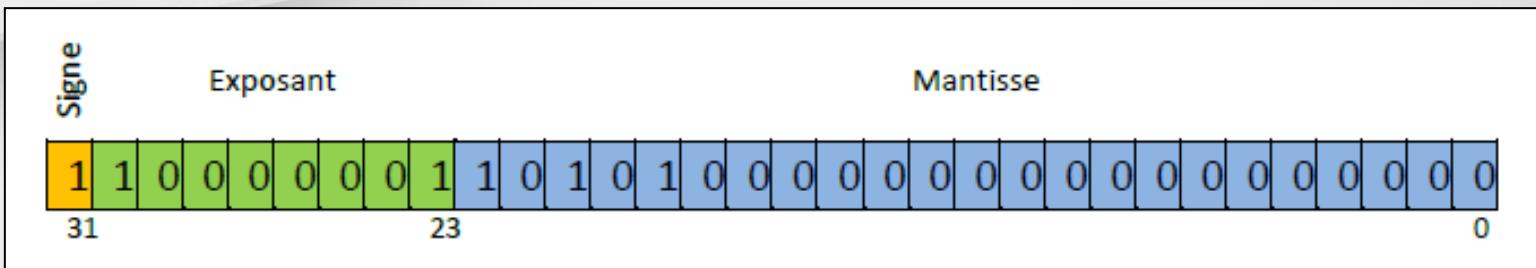
On veut coder le nombre $N = (-6,625)_{10}$ en utilisant la norme IEEE 754 :

- Le nombre est négatif donc on met 1 dans le bit de signe
 - On code la valeur absolue en binaire : $(6,625)_{10} = (110,101)_2$
 - On transforme le nombre sous la forme : 1, partie fractionnaire

$$110,101 = 1,\textcolor{red}{10101} * 2^2$$

Mantisse sur 23 bits : 101 0100 0000 0000 0000 0000

Exposant : $127 + 2 = (129)_{10} = (1000\ 0001)_2$



ASCII

Le code ASCII (American Standard Code for Information Interchange) :

- Une norme internationale
- Utiliser pour la transmission entre ordinateurs ou périphériques
- Coder sur 7 bits sous sa forme standard ($2^7=128$ caractères)
- Puis 8 bits (ASCII étendu)
- Représente les majuscules et minuscules, les chiffres décimaux, des signes de ponctuation et des caractères de commande.

TABLE ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	.	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	1	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	-	127	7F	[DEL]

CIRCUITS LOGIQUES

I. Introduction

L'ordinateur est un dispositif électronique sophistiqué conçu à partir de circuits intégrés qui traite l'information mise sous forme d'impulsions électriques traduisant les chaînes binaires utilisées pour représenter les symboles qu'on y introduit codés sous forme d'une suite de bits. Rappelons qu'un ordinateur ne comprend que les impulsions électriques.

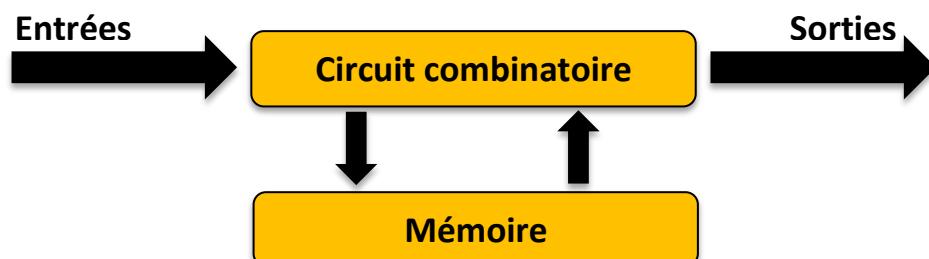
Les traitements, pour leur part, sont essentiellement réalisés à l'aide d'opérations telles l'addition, la soustraction, la multiplication, la division, la comparaison. Plus fondamentalement, les opérations sont composées de fonctions logiques qui sont effectuées par des circuits logiques de base.

Il existe deux grands types de fonctions logiques :

- Les fonctions logiques combinatoires, bases du calcul booléen, elles résultent de l'analyse combinatoire des variations des grandeurs d'entrées uniquement.



- Les fonctions logiques séquentielles ou bascules, qui résultent de l'association de plusieurs fonctions logiques « combinatoires » dont les résultats ne dépendent pas seulement des données en train d'être traitées, mais aussi des données traitées précédemment.



Les fonctions logiques combinatoires directement issues des mathématiques (algèbre de Boole) sont les outils de base de l'électronique numérique. Elles sont mises en œuvre en électronique sous forme de portes logiques. Ainsi les circuits électroniques calculent des fonctions logiques de l'algèbre de Boole. Ces portes électroniques sont construites à partir de plusieurs transistors connectés de manière adéquate.

II. Portes logiques

Tous les circuits digitaux sont constitués d'éléments de base appelés portes logiques que l'on peut combiner de plusieurs façons dont le comportement fonctionnel est décrit par l'algèbre de Boole. Les informations traitées par un ordinateur sont représentées par des variables logiques (variables booléennes) ne peuvent prendre que deux valeurs logiques représentées en binaire par 0 et 1. Ainsi, chaque symbole binaire correspond à un état physique du dispositif.

Niveau électrique	Etat	Tension	Binaire	Proposition
Bas	Ouvert (OFF)	0 volt	0	Faux
Haut	Fermé (ON)	5 volts	1	Vrai

1. Fonction logique

Une fonction logique F (ou booléenne) est une fonction de n variables logiques dont la valeur appartient à l'ensemble $E = \{0, 1\}$.

$$F: \begin{matrix} E^n \rightarrow E \\ \{0, 1\} \rightarrow \{0, 1\} \end{matrix}$$

Ainsi, si on n variables alors on aura 2^n combinaisons possibles, et pour chaque combinaison correspond une valeur. Le tableau qui représente les 2^n valeurs s'appelle « **la table de vérité** » de la fonction.

Exemple

Soit la fonction $F(X, Y)$ donc $n=2$ alors on a $2^2=4$ combinaisons possibles. Donc la table de vérité de F sera comme suit :

X	Y	F
0	0	
0	1	
1	0	
1	1	

} 0 ou 1

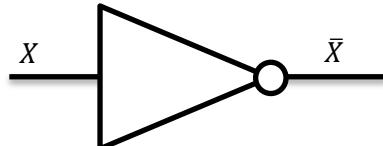
2. Fonctions logiques de base

Toute fonction logique peut être réalisée à l'aide d'un nombre de fonctions logiques de base appelées portes. On distingue trois fonctions logiques de base qui sont NOT, AND, OR.

a. Porte NOT ou NON

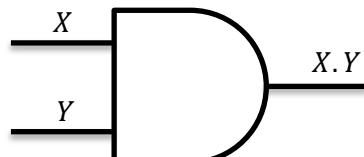
Elle possède une seule entrée et une seule sortie. Elle consiste à changer l'état de 0 à 1 et inversement.

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

**b. Porte AND ou ET**

Elle possède au moins deux entrées et une seule sortie dont la valeur correspond au produit des deux variables booléennes d'entrées. La sortie ne passe à 1 que si toutes les entrées passent à 1.

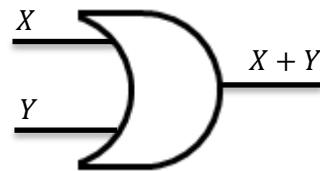
X	Y	$F(X, Y) = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1



c. Porte OR ou OU

Elle possède au moins deux entrées et une seule sortie dont la valeur correspond à la somme des deux variables booléennes d'entrées. Il suffit qu'une variable logique en entrée passe à 1 pour que la sortie passe à 1.

X	Y	$F(X, Y) = X + Y$
0	0	0
0	1	1
1	0	1
1	1	1



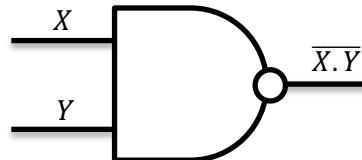
3. Fonctions logiques combinées

Les trois opérations de base permettent d'écrire toutes les fonctions de l'ordinateur.

a. Porte NAND ou NON-ET

Le NOT appliqué au AND.

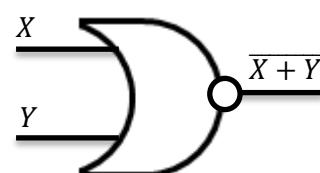
X	Y	$F(X, Y) = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0



b. Porte NOR ou NON-OU

Le NOT appliqué au OU.

X	Y	$F(X, Y) = \overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0



c. Porte XOR ou OU-exclusif

Le résultat de la fonction est à 1 si et seulement si une seule entrée vaut 1.

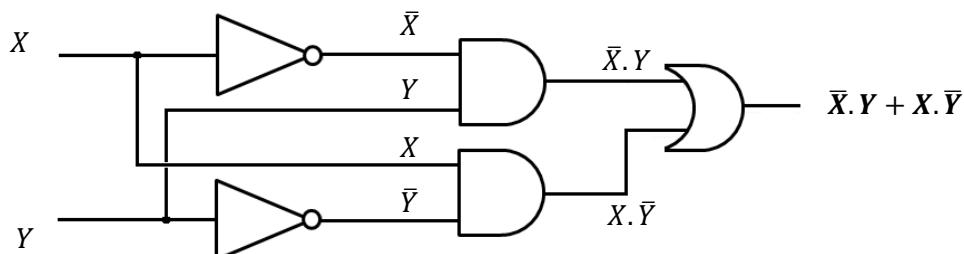
X	Y	$F(X, Y) = X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0



$$F(X, Y) = X \oplus Y = \bar{X} \cdot Y + X \cdot \bar{Y}$$

X	Y	\bar{X}	\bar{Y}	$\bar{X} \cdot Y$	$X \cdot \bar{Y}$	$\bar{X} \cdot Y + X \cdot \bar{Y}$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

Le circuit du XOR à partir des circuits de base est le suivant :



4. Exemples de circuits intégrés logiques

Le tableau suivant donne une idée sur les circuits intégrés logiques contenant les portes logiques de base ou combinées.

Porte	Circuit	Brochage	Aperçu
NOT	CMOS 4069		

AND	CMOS 4081		
OR	CMOS 4071		
NAND	CMOS 4011		
NOR	CMOS 4001		
XOR	CMOS 4030		

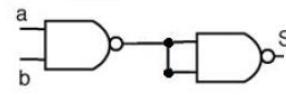
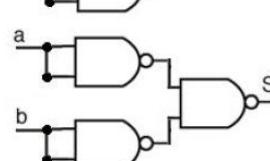
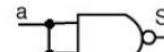
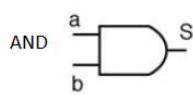
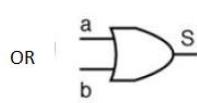
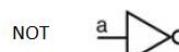
Exercice

Les portes logiques NAND et NOR sont appelées universelles, car avec elles seules on peut réaliser toutes les autres portes logiques.

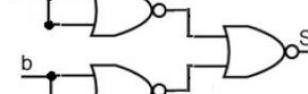
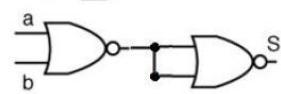
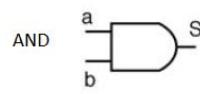
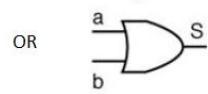
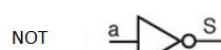
- A l'aide des portes NAND uniquement réaliser les trois portes logiques de bases : NOT, OR, AND
- A l'aide des portes NOR uniquement réaliser les trois portes logiques de bases : NOT, OR, AND

Solution

- Construction des portes NOT, OR, AND à l'aide de portes NAND



- Construction des portes NOT, OR, AND à l'aide de portes NOR



CIRCUITS LOGIQUES

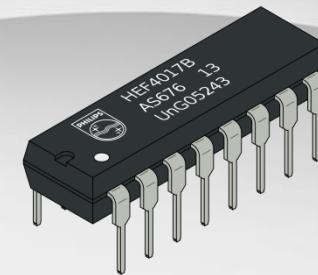
DONNÉES BINAIRE

- Un ordinateur fonctionne avec des **impulsions électriques**
- L'ordinateur est un dispositif conçu à partir de **circuits intégrés**
- Les **impulsions électriques** sont représentées par des **chaînes binaires**
- Chaque **symbole** est codé sous forme d'une **suite de bits (0100 0001)**



Ordinateur

Se compose de



Circuit intégré

DONNÉES BINAIRE

Les **traitements des données binaire** sont réalisés à l'aide d'opérations :

- Addition
- Soustraction
- Multiplication
- Division
- Comparaison

Les **opérations** sont composées de **fonctions logiques** qui sont effectuées par des **circuits logiques de base**

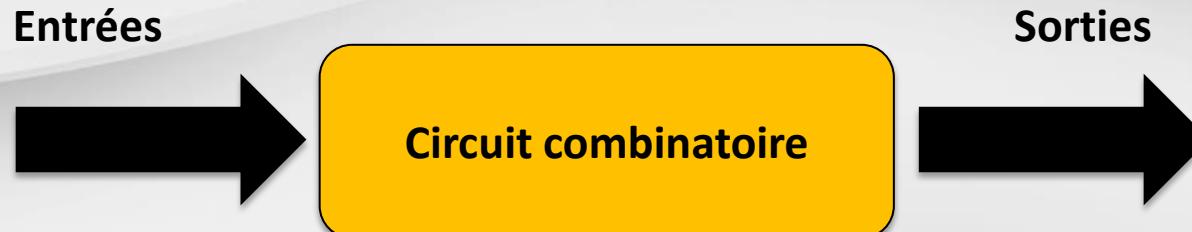
FONCTIONS LOGIQUES

Il existe deux grands types de **fonctions logiques** :

- Les fonctions logiques combinatoires
- Les fonctions logiques séquentielles

□ **Les fonctions logiques combinatoires:**

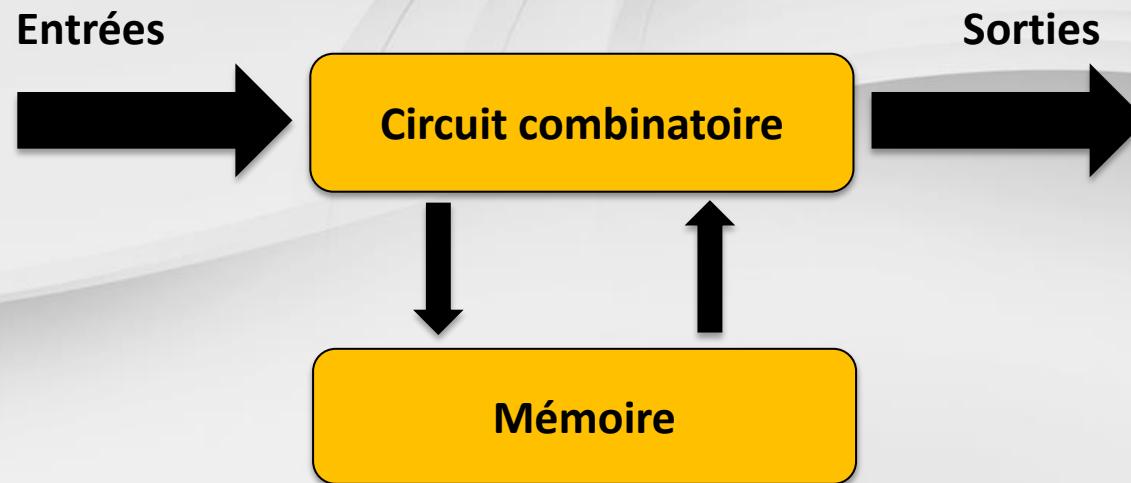
bases du **calcul booléen**, elles résultent de l'analyse combinatoire des variations des grandeurs d'entrées uniquement



FONCTIONS LOGIQUES

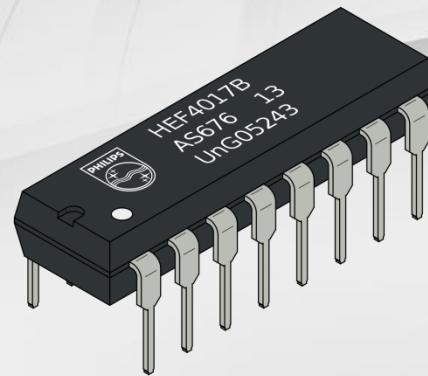
□ Les fonctions logiques séquentielles ou bascules:

Elle résultent de l'association de plusieurs fonctions logiques « combinatoires » dont les résultats ne dépendent pas seulement des données en train d'être traitées, mais aussi **des données traitées précédemment**



INTRODUCTION

- Les **fonctions logiques** directement issues des **mathématiques** (**algèbre de Boole**) sont les outils de base de l'**électronique numérique**
- Elles sont mises en œuvre en **électronique** sous forme de **portes logiques**
- Ainsi les **circuits électroniques** calculent des **fonctions logiques** de l'**algèbre de Boole**



INTRODUCTION

- Les **informations** traitées par un ordinateur sont représentées par des **variables logiques** (variables booléennes)
- Elles ne peuvent prendre que **deux valeurs logiques en binaire** par 0 et 1.
- Ainsi, chaque symbole binaire correspond à **un état physique du dispositif**.

Niveau électrique	Etat	Tension	Binaire	Proposition
Bas	Ouvert (OFF)	0 volt	0	Faux
Haut	Fermé (ON)	5 volts	1	Vrai

CARACTÉRISTIQUES D'UNE F. LOG

- Une **fonction logique** F (ou booléenne) est une fonction de n **variables logiques** dont la valeur appartient à l'ensemble $E = \{0, 1\}$.

$$\begin{aligned} F: \quad & E^n \rightarrow E \\ & \{0, 1\} \rightarrow \{0, 1\} \end{aligned}$$

- Si on n variables alors on aura 2^n combinaisons possibles, et pour chaque combinaison correspond une valeur.
- Le tableau qui représente les 2^n valeurs s'appelle « **la table de vérité** » de la fonction.

CARACTÉRISTIQUES D'UNE F. LOG

Exemple

- Soit la fonction $F(X,Y)$ donc $n=2$ alors on a $2^2=4$ combinaisons possibles.
- Donc la **table de vérité** de F sera comme suit :

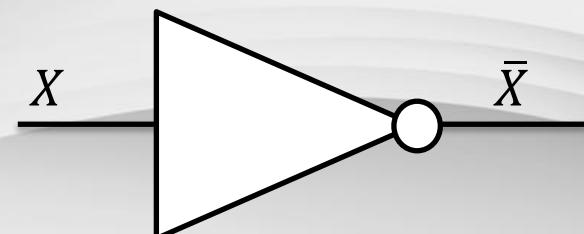
X	Y	$F(X,Y)$
0	0	
0	1	
1	0	
1	1	

0 ou 1

Porte NOT ou NON

- Elle possède une **seule entrée** et une **seule sortie**.
- Elle consiste à changer l'état de 0 à 1 et inversement.

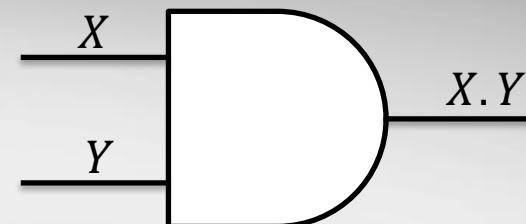
X	$F(X) = \bar{X}$
0	1
1	0



Porte AND ou ET

- Elle possède au moins deux entrées et une seule sortie
- La valeur de sortie correspond au produit des deux variables booléennes d'entrées.
- La sortie ne passe à 1 que si toutes les entrées passent à 1.

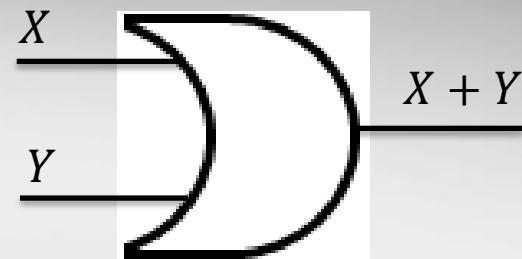
X	Y	$F(X, Y) = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1



Porte OR ou OU

- Elle possède **au moins deux entrées et une seule sortie**
- La valeur de la sortie correspond à la **somme** des deux variables d'entrées.
- Il suffit qu'une variable logique en entrée passe à 1 pour que la sortie passe à 1.

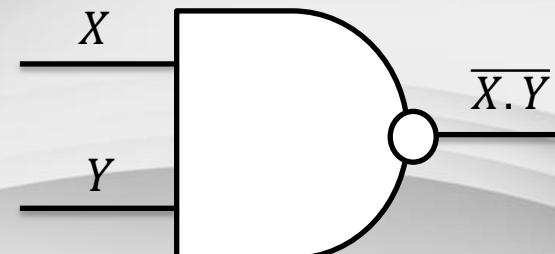
X	Y	$F(X, Y) = X + Y$
0	0	0
0	1	1
1	0	1
1	1	1



Porte NAND ou NON-ET

- Le NOT appliqué au AND

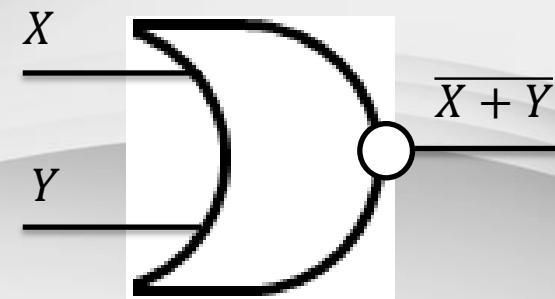
X	Y	$F(X, Y) = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0



Porte NOR ou NON-OU

- Le NOT appliqué au OR

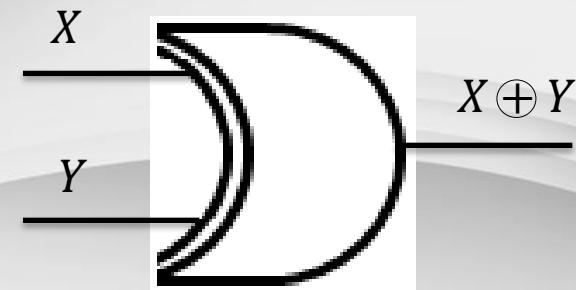
X	Y	$F(X, Y) = \overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0



Porte XOR ou OU-exclusif

- Le résultat de la fonction est à 1 si et seulement si une seule entrée vaut 1

X	Y	$F(X,Y)=X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

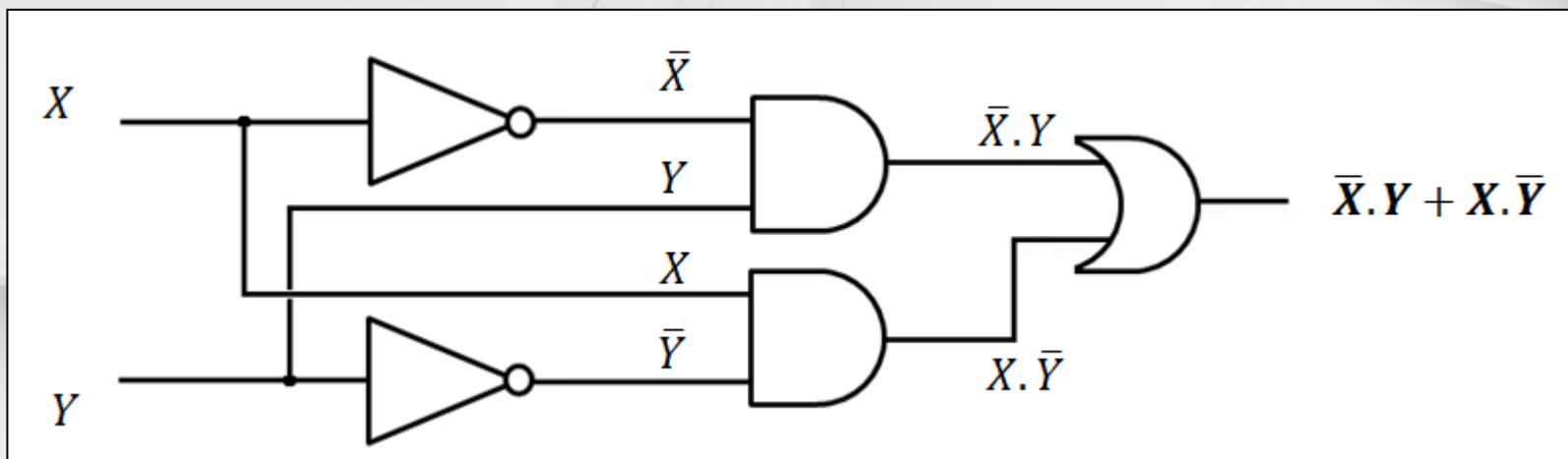


Porte XOR ou OU-exclusif

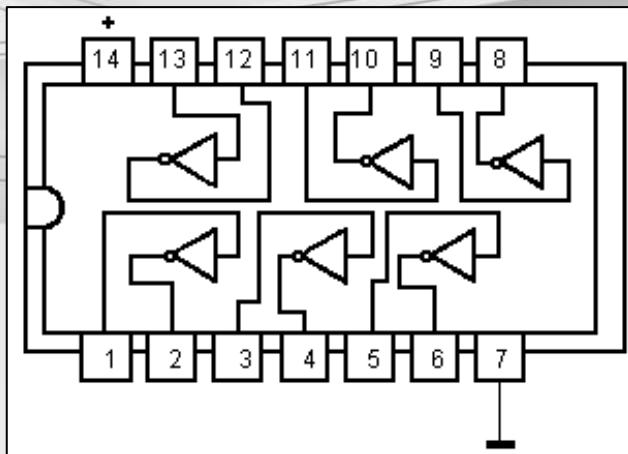
$$F(X, Y) = X \oplus Y = \bar{X} \cdot Y + X \cdot \bar{Y}$$

X	Y	\bar{X}	\bar{Y}	$\bar{X} \cdot Y$	$X \cdot \bar{Y}$	$\bar{X} \cdot Y + X \cdot \bar{Y}$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

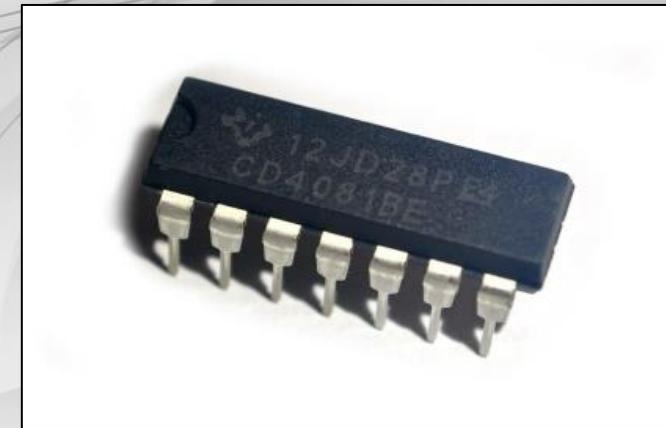
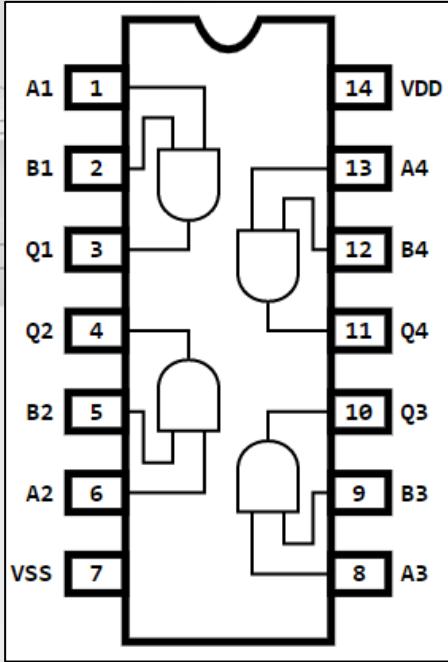
Le circuit du XOR à partir des circuits de base est le suivant :



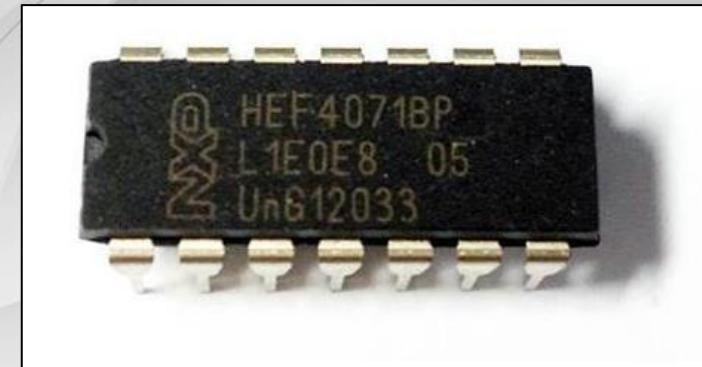
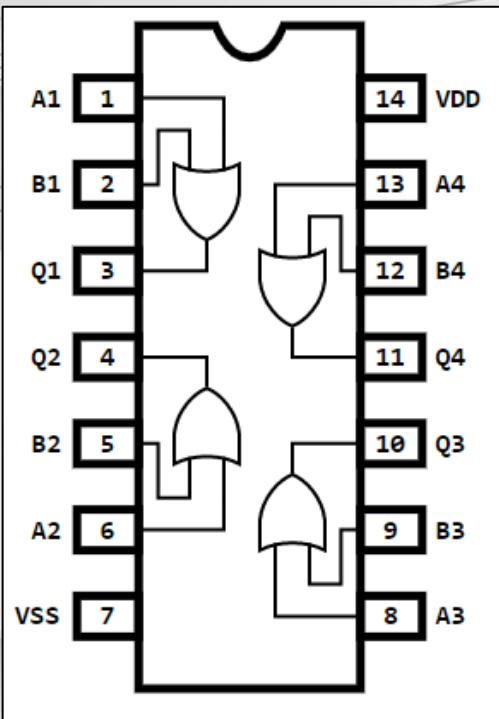
Porte NOT : CMOS 4069



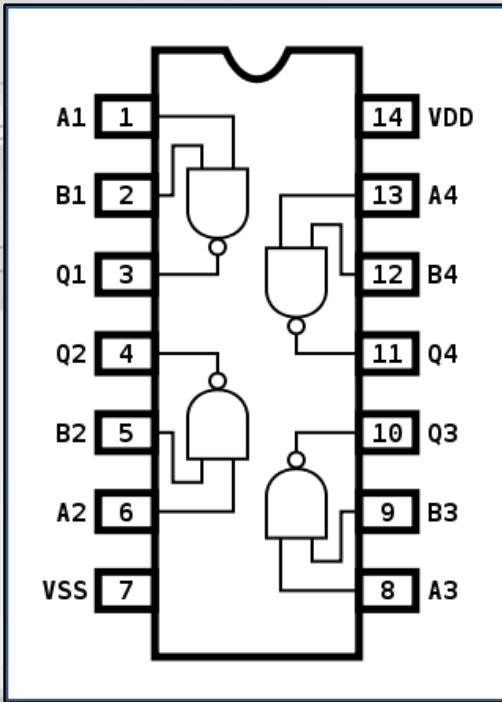
Porte AND : CMOS 4081



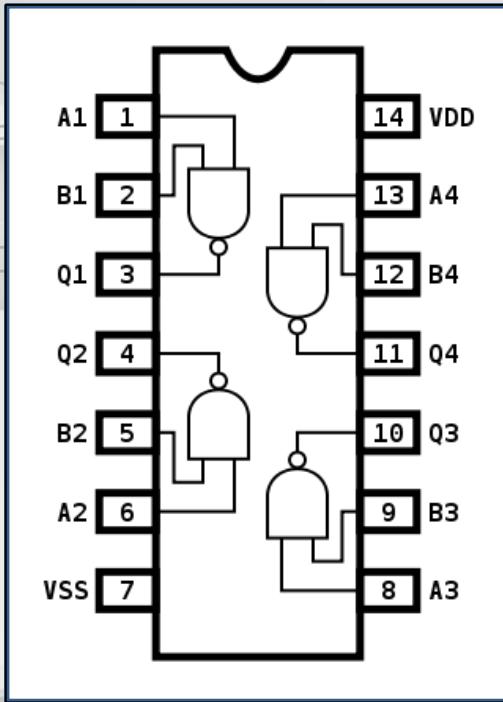
Porte OR : CMOS 4071



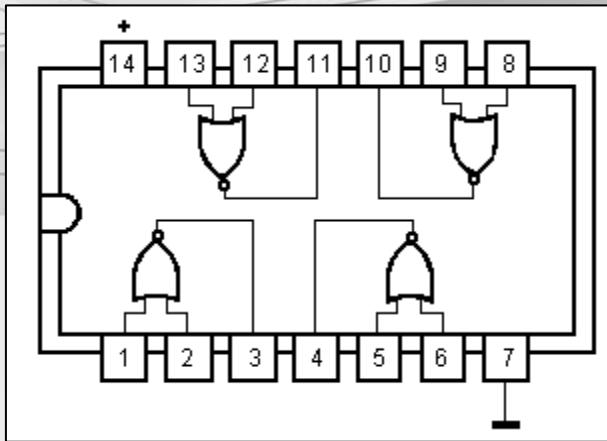
Porte NAND : CMOS 4011



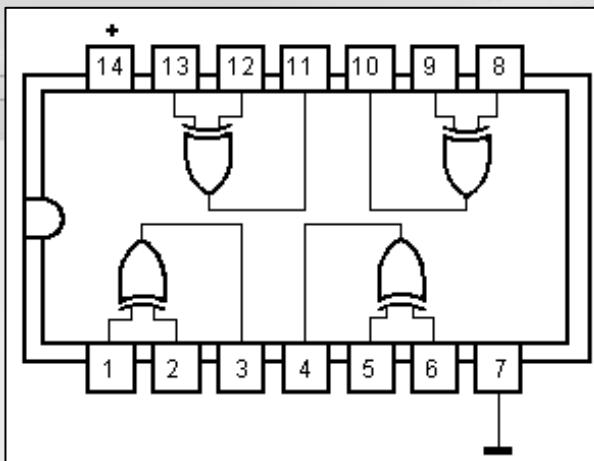
Porte NAND : CMOS 4011



Porte NOR : CMOS 4001



Porte XOR : CMOS 4030



Exercice

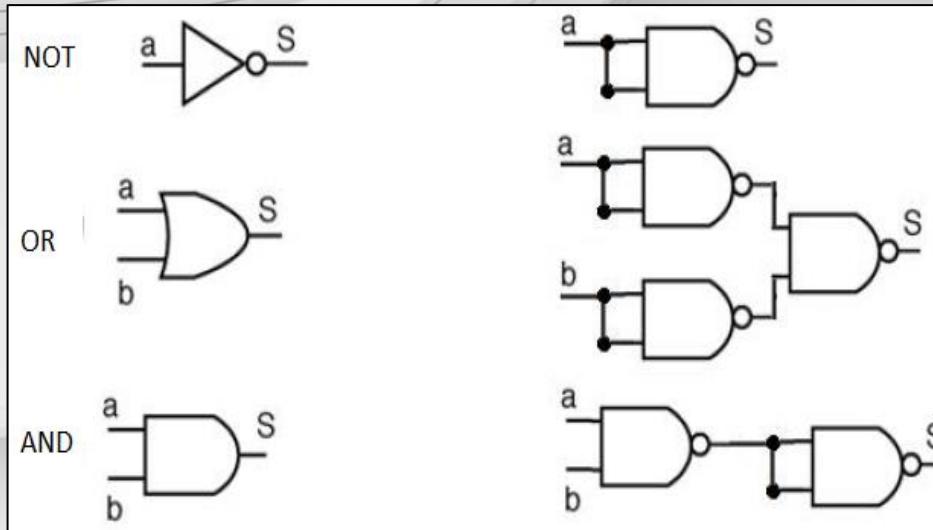
Les portes logiques NAND et NOR sont appelées universelles, car avec elles seules on peut réaliser toutes les autres portes logiques.

- 1) A l'aide des portes NAND uniquement réaliser les trois portes logiques de bases : NOT, OR, AND
- 2) A l'aide des portes NOR uniquement réaliser les trois portes logiques de bases : NOT, OR, AND

Exercice

Solution

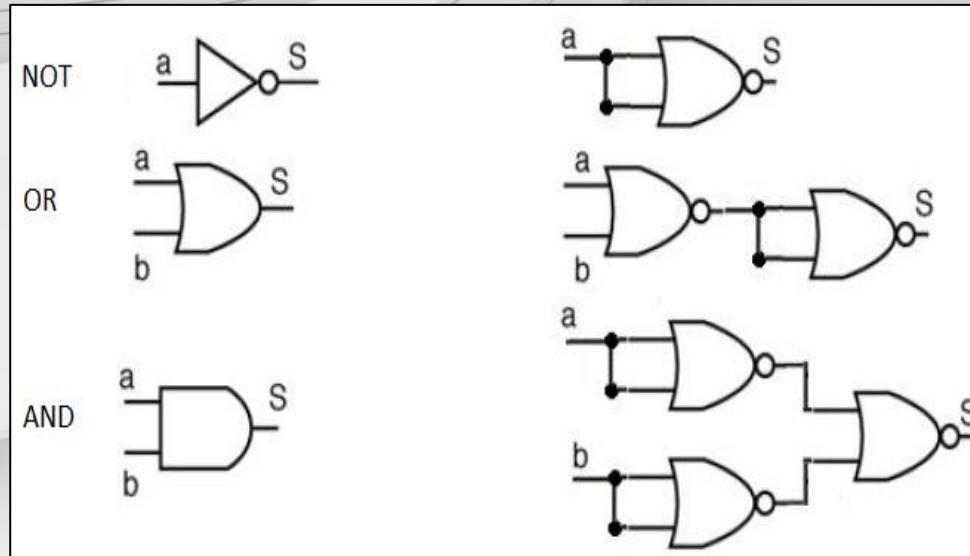
1) Construction des portes NOT, OR, AND à l'aide de portes NAND



Exercice

Solution

2) Construction des portes NOT, OR, AND à l'aide de portes NOR



EQUATIONS LOGIQUES

I. Théorèmes de l'algèbre de Boole

L'algèbre de Boole contient un ensemble de théorèmes mathématiques qui précisent les fondements théoriques de la logique binaire ou booléenne. Il se greffe autour d'une liste de 22 théorèmes fondamentaux. Il est intéressant de noter que la moitié (ou presque) d'entre eux sont le dual de l'autre : en remplaçant l'opérateur ET(.) par l'opérateur OU(+) et vice-versa, et en changeant les « 0 » par les « 1 » et vice-versa, on retrouve l'autre portion des théorèmes du tableau.

Identité	
1) $A \cdot 1 = A$	5) $A + 0 = A$
2) $A \cdot 0 = 0$	6) $A + 1 = 1$
3) $A \cdot A = A$	7) $A + A = A$
4) $A \cdot \bar{A} = 0$	8) $A + \bar{A} = 1$
Commutativité	
9) $A \cdot B = B \cdot A$	10) $A + B = B + A$
Associativité	
11) $A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$	12) $A + (B + C) = (A + B) + C = A + B + C$
Distributivité	
13) $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	14) $A + (B \cdot C) = (A + B) \cdot (A + C)$
Absorption	
15) $A \cdot (A + B) = A$	16) $A + (A \cdot B) = A$
Adjacence logique	
17) $A \cdot B + A \cdot \bar{B} = A$	18) $(A + B)(A + \bar{B}) = A$
Allégement	
19) $A \cdot (\bar{A} + B) = A \cdot B$	20) $A + \bar{A} \cdot B = A + B$
Loi de De Morgan	
21) $\overline{A \cdot B} = \bar{A} + \bar{B}$ $A \cdot B \dots \bar{X} = \bar{A} + \bar{B} + \dots + \bar{X}$	22) $\overline{A + B} = \bar{A} \cdot \bar{B}$ $A + B + \dots + \bar{X} = \bar{A} \cdot \bar{B} \dots \bar{X}$

Exemple : Simplifier algébriquement la fonction suivante :

$$\begin{aligned}
 F(X, Y) &= \bar{X} \cdot \bar{Y} + X \cdot \bar{Y} + X \cdot Y \\
 &= \bar{Y} \cdot (\bar{X} + X) + X \cdot Y \quad (\text{Théorème 13})
 \end{aligned}$$

$$\begin{aligned}
 &= \bar{Y} \cdot 1 + X \cdot Y && (\text{Théorème 8}) \\
 &= \bar{Y} + X \cdot Y && (\text{Théorème 1}) \\
 &= \bar{Y} + X && (\text{Théorème 20})
 \end{aligned}$$

II. Etablissement d'équations logiques

1. Problématique

Soit l'équation d'un circuit logique $S = \overline{\bar{X} \cdot \bar{Y}} + \bar{Y} \cdot Z$ (1)

A l'aide des théorèmes de l'algèbre de boule, on peut écrire :

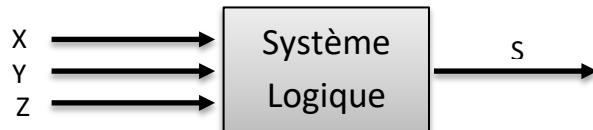
$$S = (\bar{X} \cdot \bar{Y}) \cdot (\bar{Y} \cdot Z) = (X \cdot Y) \cdot (\bar{Y} \cdot Z) \quad (\text{Théorème 22})$$

$$S = (X \cdot Y) \cdot (Y + \bar{Z}) \quad (\text{Théorème 21})$$

$$S = X \cdot Y + X \cdot Y \cdot \bar{Z} \quad (2) \quad (\text{Théorème 13})$$

$$S = X \cdot Y \quad (3) \quad (\text{Théorème 16})$$

Le schéma général du système logique est le suivant :



Le même système qui fournit une sortie S en fonction des valeurs des entrées X, Y, Z peut être réalisé de trois manières différentes :

Equation (1) a pour coût :

- deux portes AND à deux entrées
- trois portes NOT
- une porte OR

Equation (2) a pour coût :

- trois portes AND à deux entrées

- une porte OR à deux entrées
- une porte NOT

Equation (3) a pour coût :

- une porte AND à deux entrées

D'où la nécessité de simplifier au maximum la fonction logique d'un circuit afin de minimiser son coût.

2. Représentation d'une fonction logique

Les fonctions décrites par les tables de vérité peuvent être synthétisées d'une façon systématique par leur expression booléenne de la forme d'une somme de produit (**SDP**) ou d'un produit de sommes (**PDS**). Pour chaque combinaison de variables on a un terme unique qui peut être représenté par un minterme ou un maxterme :

- **Un minterme** est définie comme étant le produit logique des variables booléennes considérées avec la convention suivante :
 - Si la variable égale à 0 alors inscrire son complément.
 - Si la variable égale à 1 alors inscrire la variable elle-même.
- **Un maxterme** est définie comme étant la somme logique des variables booléennes considérées avec la convention suivante :
 - Si la variable égale à 0 alors inscrire la variable elle-même.
 - Si la variable égale à 1 alors inscrire son complément.

Exemple

Soit une fonction logique F à trois variables x, y, z.

Si $(x,y,z)=(0,0,0)$ alors minterme= $m_0=(\bar{x} \cdot \bar{y} \cdot \bar{z})$

Maxterme= $M_0=(x + y + z)$

Si $(x,y,z)=(1,0,1)$ alors minterme= $m_5=(x \cdot \bar{y} \cdot z)$

Maxterme= $M_0=(\bar{x} + y + \bar{z})$

x	y	z	Minterme	Maxterme
0	0	0	m_0	$\bar{x} \cdot \bar{y} \cdot \bar{z}$
0	0	1	m_1	$\bar{x} \cdot \bar{y} \cdot z$
0	1	0	m_2	$\bar{x} \cdot y \cdot \bar{z}$
0	1	1	m_3	$\bar{x} \cdot y \cdot z$
1	0	0	m_4	$x \cdot \bar{y} \cdot \bar{z}$
1	0	1	m_5	$x \cdot \bar{y} \cdot z$
1	1	0	m_6	$x \cdot y \cdot \bar{z}$
1	1	1	m_7	$x \cdot y \cdot z$

Toute fonction logique peut être écrite comme une somme logique de mintermes pour lequel la fonction vaut 1 ou bien un produit logique de maxtermes pour lequel la fonction vaut 0. Cette forme est appelée forme canonique d'une fonction logique :

- SDP = $\sum_{i=0}^n m_i / F=1$
- PDS = $\prod_{i=0}^n M_i / F=0$

Exemple : Soit la fonction majoritaire à 3 variables

- $F=0$ si la majorité de variable est égale à 0
- $F=1$ si la majorité de variable est égale à 1

x	y	z	F	m_i	M_i
0	0	0	0	m_0	M_0
0	0	1	0	m_1	M_1
0	1	0	0	m_2	M_2
0	1	1	1	m_3	M_3
1	0	0	0	m_4	M_4
1	0	1	1	m_5	M_5
1	1	0	1	m_6	M_6
1	1	1	1	m_7	M_7

- **1^{er} cas : SDP**

$$\text{Forme canonique} = \sum_{i=0}^n m_i / F=1$$

$$F(x,y,z) = m_3 + m_5 + m_6 + m_7$$

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

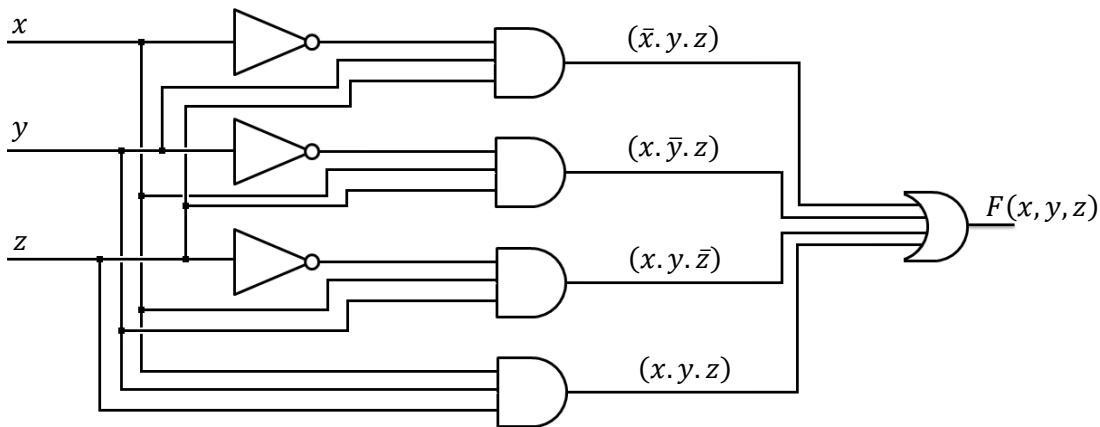
- 2^{ième} cas : PDS

Forme canonique = $\sum_{i=0}^n M_i / F=0$

$$F(x,y,z) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

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

Le circuit logique (logigramme) de la fonction F en utilisant le SDP est le suivant : (Ce circuit peut être simplifié)



3. Simplification des fonctions logiques

a. Simplification algébrique

Dans cette méthode, on se base essentiellement sur les théorèmes de l'algèbre de Boole pour simplifier les expressions logiques. Malheureusement, il n'est pas toujours facile de savoir quel théorème faut-il évoquer pour obtenir la simplification minimale.

Exercice : simplifier les équations suivantes

$$F_1 = \bar{a}bc + abc + ab\bar{c} + \bar{a}b\bar{c}$$

$$F_2 = ab + \bar{a}b + a\bar{b}$$

$$F_3 = abc + ab\bar{c} + \bar{a}b\bar{c} + ab\bar{c}$$

Solution

$$F_1 = \bar{a}bc + abc + ab\bar{c} + \bar{a}b\bar{c} = \bar{a}bc + ab(c + \bar{c}) + \bar{a}b\bar{c}$$

$$F_1 = \bar{a}b(c + \bar{c}) + ab = \bar{a}b + ab = b(a + \bar{a}) = b$$

$$F_2 = ab + \bar{a}b + a\bar{b} = b(a + \bar{a}) + a\bar{b}$$

$$F_2 = b + a\bar{b} = b + a \quad (\text{Théorème d'allégement})$$

$$F_3 = abc + ab\bar{c} + \bar{a}b\bar{c} + ab\bar{c} = ab(c + \bar{c}) + b\bar{c}(a + \bar{a}) = ab + b\bar{c} = b(a + \bar{c})$$

b. Simplification par la méthode de Karnaugh

La méthode du tableau de Karnaugh permet de visualiser une fonction et d'en tirer intuitivement une fonction simplifiée. L'élément de base de cette méthode est la table de Karnaugh qui est représenté sous forme d'un tableau formé par des lignes et des colonnes. Chaque case représente une combinaison des variables qui contient la valeur de la fonction correspondante.

La méthode consiste à réaliser des groupements des cases adjacentes :

- On regroupe les 1 si on considère les mintermes
- On regroupe les 0 si on considère les maxtermes

Remarque : Dans ce qui suit on considère les mintermes, ainsi on regroupe les 1.

Ces groupements des cases doivent être de taille maximale (nombre max de casse) et égale à 2^k (c'est-à-dire 2, 4, 8, 16, ...). On cesse d'effectuer les groupements lorsque tous les uns appartiennent au moins à l'un d'eux. Avant de tirer les équations du tableau de Karnaugh, il faut respecter les règles suivantes :

- Grouper tous les uns.
- Grouper le maximum des uns dans un seul groupement. Un groupement a une forme rectangulaire.

- Le nombre des uns dans un groupement est une puissance de 2 et égal à 2^k . Un 1 peut figurer dans plus qu'un groupement.
- Un groupement doit respecter les axes de symétries du tableau de Karnaugh.

Regroupement des 2 cases adjacentes

La réunion de deux cases adjacentes contenant 1 chacune élimine une seule variable celle qui change d'état en passant d'une case à l'autre.

Simplification de la fonction Majorité de 3 variables $F(x, y, z)$

x\yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Groupement bleu : $x\bar{y}z + xyz = xz$

Groupement rouge : $\bar{x}yz + xyz = yz$

Groupement vert : $xyz + xy\bar{z} = xy$

Donc la forme simplifiée de la fonction Majorité est $F(x, y, z) = xy + yz + xz$

Regroupement des 4 cases adjacentes

Lorsqu'on regroupe 4 cases adjacentes alors 2 variables disparaissent, on peut alors remplacer la somme des 4 cases par un seul terme qui comporte que 2 variables uniquement.

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

AB/CD	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	1	1	0	1
10	0	0	0	1

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

AB/CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	0	0	1
10	1	0	0	1

- Fonction $F_3(A, B, C, D) = \bar{C}\bar{D} + AB + \bar{B}C$

AB/CD	00	01	11	10
00	1	0	1	1
01	1	0	0	0
11	1	1	1	1
10	1	0	1	1

Regroupement des 8 cases adjacentes

Un groupement de 8 cases adjacentes ou symétriques remplies des 1 logiques va simplifier 3 variables logiques dans l'expression canonique de la fonction logique.

- Fonction $F_4(A, B, C, D) = \bar{D}$

AB/CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

- Fonction $F_5(A, B, C, D) = \bar{B}$

AB/CD	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

Traitements des problèmes à 5 variables

Pour résoudre ce problème on va le décomposer en 2 problèmes à 4 variables en appliquant le théorème d'expansion (théorème de Shannon).

On a: $F_7(A, B, C, D, E) = \bar{E} \cdot F_7(A, B, C, D, 0) + E \cdot F_7(A, B, C, D, 1)$

- Fonction $F_7(A, B, C, D, 0) = C\bar{D}$

AB/CD	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	0	1
10	0	0	0	1

- Fonction $F_7(A, B, C, D, 1) = \bar{C}D$

AB/CD	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

Ce qui donne : $F_7(A, B, C, D, E) = \bar{E}C\bar{D} + E\bar{C}D$

Les valeurs indifférentes ou indéfinies

Le symbole \emptyset (ou X ou *) peut prendre indifféremment la valeur 0 ou 1 : on considère que la valeur est 1 uniquement pour ceux qui permettent d'augmenter le nombre des cases d'un regroupement et ceux qui réduisent le nombre de regroupement.

- Fonction $F_8(A, B, C, D) = \bar{A}C$

AB/CD	00	01	11	10
00	\emptyset	0	\emptyset	1
01	0	0	1	\emptyset
11	0	0	0	0
10	0	0	0	0

- Fonction $F_9(A, B, C, D) = C\bar{D} + B\bar{D} + A\bar{D}$

AB/CD	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	\emptyset	\emptyset	\emptyset	\emptyset
10	1	0	\emptyset	\emptyset

EQUATIONS LOGIQUES

INTRODUCTION

- L'algèbre de Boole contient un ensemble de théorèmes mathématiques qui précisent les fondements théoriques de la logique binaire ou booléenne.
- Il se greffe autour d'une liste de **22 théorèmes** fondamentaux.
- Il est intéressant de noter que la moitié (ou presque) d'entre eux sont le dual de l'autre :
 - en remplaçant l'opérateur ET(.) par l'opérateur OU(+) et vice-versa,
 - en changeant les « 0 » par les « 1 » et vice-versa, on retrouve l'autre portion des théorèmes du tableau.

THÉORÈMES

Identité

- 1) $A \cdot 1 = A$
 2) $A \cdot 0 = 0$
 3) $A \cdot A = A$
 4) $A \cdot \bar{A} = 0$

- 5) $A + 0 = A$
 6) $A + 1 = 1$
 7) $A + A = A$
 8) $A + \bar{A} = 1$

Commutativité

9) $A \cdot B = B \cdot A$

10) $A + B = B + A$

Associativité

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

12) $A + (B + C) = (A + B) + C = A + B + C$

Distributivité

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

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

Absorption

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

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

Adjacence logique

17) $A \cdot B + A \cdot \bar{B} = A$

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

Allégement

19) $A \cdot (\bar{A} + B) = A \cdot B$

20) $A + \bar{A} \cdot B = A + B$

Loi de De Morgan

21) $\overline{A \cdot B} = \bar{A} + \bar{B}$
 $\overline{A \cdot B \dots X} = \bar{A} + \bar{B} + \dots + \bar{X}$

22) $\overline{A + B} = \bar{A} \cdot \bar{B}$
 $\overline{A + B + \dots + X} = \bar{A} \cdot \bar{B} \dots \bar{X}$

EXEMPLE

Simplifier algébriquement la fonction suivante :

$$F(X, Y) = \bar{X} \cdot \bar{Y} + X \cdot \bar{Y} + X \cdot Y$$

$$= \bar{Y} \cdot (\bar{X} + X) + X \cdot Y \quad (\text{Distributivité - Théorème 13})$$

$$= \bar{Y} \cdot 1 + X \cdot Y \quad (\text{Théorème 8})$$

$$= \bar{Y} + X \cdot Y \quad (\text{Théorème 1})$$

$$= \bar{Y} + X \quad (\text{Allégement - Théorème 20})$$

PROBLÉMATIQUE

Soit l'équation d'un circuit logique $S = \overline{X \cdot Y} + \bar{Y} \cdot Z$ (1)

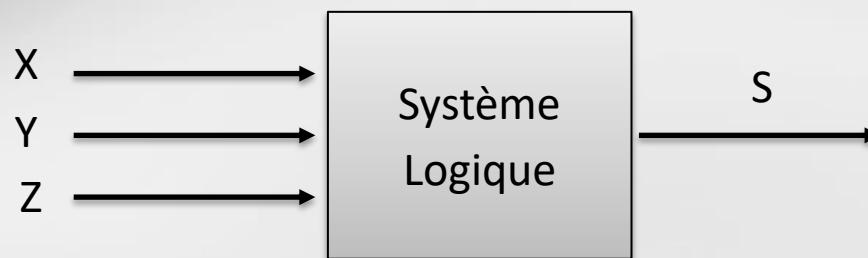
A l'aide des théorèmes de l'algèbre de boule, on peut écrire :

$$S = (\overline{X \cdot Y}) \cdot (\bar{Y} \cdot Z) = (X \cdot Y) \cdot (\overline{\bar{Y} \cdot Z}) \quad (\text{Théorème 22})$$

$$S = (X \cdot Y) \cdot (Y + \bar{Z}) \quad (\text{Théorème 21})$$

$$S = X \cdot Y + X \cdot Y \cdot \bar{Z} \quad (2) \quad (\text{Théorème 13})$$

$$S = X \cdot Y \quad (3) \quad (\text{Théorème 16})$$



PROBLÉMATIQUE

Le même système peut être réalisé de trois manières différentes :

Equation (1) a pour coût :

- deux portes AND à deux entrées
- trois portes NOT
- une porte OR

Equation (2) a pour coût :

- trois portes AND à deux entrées
- une porte OR à deux entrées
- une porte NOT

Equation (3) a pour coût :

- une porte AND à deux entrées



D'où la nécessité de simplifier au maximum la fonction logique d'un circuit afin de minimiser son coût.

REPRÉSENTATION

Les fonctions décrites par les **tables de vérité** peuvent être synthétisées :

- somme de produit (**SDP**)
- produit de sommes (**PDS**)

Chaque combinaison de variables on a un terme unique qui peut être représenté par :

- **Un minterme** est définie comme étant **le produit logique** des variables booléennes considérées avec la convention suivante :
 - Si la variable égale à 0 alors inscrire son complément.
 - Si la variable égale à 1 alors inscrire la variable elle-même.
- **Un maxterme** est définie comme étant **la somme logique** des variables booléennes considérées avec la convention suivante :
 - Si la variable égale à 0 alors inscrire la variable elle-même.
 - Si la variable égale à 1 alors inscrire son complément.

REPRÉSENTATION

Exemple général

Soit une fonction logique F à trois variables x, y, z.

Si $(x,y,z)=(0,0,0)$ alors

$$\text{minterme} = m_0 = (\bar{x} \cdot \bar{y} \cdot \bar{z})$$

$$\text{Maxterme} = M_0 = (x + y + z)$$

Si $(x,y,z)=(1,0,1)$ alors

$$\text{minterme} = m_5 = (x \cdot \bar{y} \cdot z)$$

$$\text{Maxterme} = M_0 = (\bar{x} + y + \bar{z})$$

REPRÉSENTATION**Exemple général**

x	y	z	Minterme	Maxterme
0	0	0	m_0	$\bar{x} \cdot \bar{y} \cdot \bar{z}$
0	0	1	m_1	$\bar{x} \cdot \bar{y} \cdot z$
0	1	0	m_2	$\bar{x} \cdot y \cdot \bar{z}$
0	1	1	m_3	$\bar{x} \cdot y \cdot z$
1	0	0	m_4	$x \cdot \bar{y} \cdot \bar{z}$
1	0	1	m_5	$x \cdot \bar{y} \cdot z$
1	1	0	m_6	$x \cdot y \cdot \bar{z}$
1	1	1	m_7	$x \cdot y \cdot z$

Toute fonction logique peut être écrite comme

- une **somme logique de mintermes** pour lequel la fonction vaut 1
- un **produit logique de maxtermes** pour lequel la fonction vaut 0

Cette forme est appelée **forme canonique** d'une fonction logique :

$$SDP = \sum_{i=0}^n m_i / F=1$$

$$PDS = \prod_{i=0}^n M_i / F=0$$

REPRÉSENTATION

Exemple

Soit la fonction majoritaire à 3 variables

- $F=0$ si la majorité de variable est égale à 0
- $F=1$ si la majorité de variable est égale à 1

1^{ier} cas : SDP

$$\text{Forme canonique} = \sum_{i=0}^n m_i / F=1$$

$$F(x,y,z) = m_3 + m_5 + m_6 + m_7$$

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

2^{ième} cas : PDS

$$\text{Forme canonique} = \sum_{i=0}^n M_i / F=0$$

$$F(x,y,z) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

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

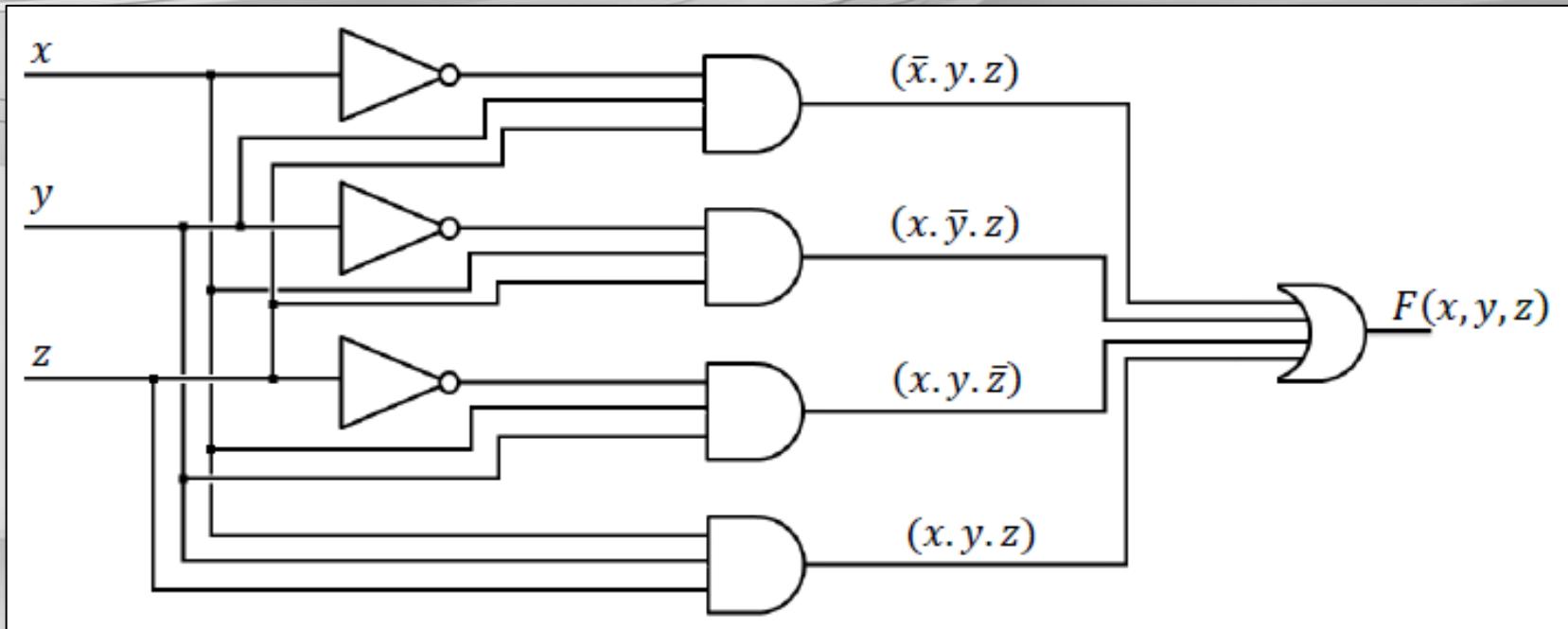
x	y	z	F	m_i	M_i
0	0	0	0	m_0	M_0
0	0	1	0	m_1	M_1
0	1	0	0	m_2	M_2
0	1	1	1	m_3	M_3
1	0	0	0	m_4	M_4
1	0	1	1	m_5	M_5
1	1	0	1	m_6	M_6
1	1	1	1	m_7	M_7

REPRÉSENTATION

Exemple

Le circuit logique (logigramme) de la fonction F en utilisant le **SDP** est le suivant :

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



Ce circuit peut être simplifié

SIMPLIFICATION ALGÉBRIQUE

- On se base essentiellement sur les théorèmes de l'**algèbre de Boole**
- il n'est pas toujours **facile de savoir quel théorème** faut-il évoquer pour obtenir la simplification minimale.

Exercice :

simplifier les équations suivantes :

$$F_1 = \bar{a}bc + abc + ab\bar{c} + \bar{a}b\bar{c}$$

$$F_2 = ab + \bar{a}b + a\bar{b}$$

$$F_3 = abc + ab\bar{c} + \bar{a}b\bar{c} + a\bar{b}\bar{c}$$

SIMPLIFICATION ALGÉBRIQUE

Solution

$$F_1 = \bar{a}bc + abc + ab\bar{c} + \bar{a}b\bar{c} = \bar{a}bc + ab(c + \bar{c}) + \bar{a}b\bar{c}$$

$$F_1 = \bar{a}b(c + \bar{c}) + ab = \bar{a}b + ab = b(a + \bar{a}) = b$$

$$F_2 = ab + \bar{a}b + a\bar{b} = b(a + \bar{a}) + a\bar{b}$$

$$F_2 = b + a\bar{b} = b + a \quad (\text{Théorème d'allégement})$$

$$F_3 = abc + ab\bar{c} + \bar{a}b\bar{c} + ab\bar{c} = ab(c + \bar{c}) + b\bar{c}(a + \bar{a}) = ab + b\bar{c} = b(a + \bar{c})$$

MÉTHODE DE KARNAUGH

- ❑ L'élément de base de cette méthode est la **table de Karnaugh**
- ❑ Chaque case représente une combinaison des variables (**minterme**)
- ❑ La méthode consiste à réaliser des **groupements des cases adjacentes**
- ❑ Ces groupements des cases doivent être de taille maximale (**nombre max de casse**) et **égale à 2^k** (c'est-à-dire 2, 4, 8, 16, ...).
- ❑ On cesse d'effectuer les groupements lorsque tous **les uns appartiennent au moins à l'un d'eux**.

MÉTHODE DE KARNAUGH

Règles

Avant de tirer les équations du tableau de Karnaugh, il faut respecter les règles suivantes :

- Grouper tous les uns.
- Grouper le maximum des uns dans un seul groupement. Un groupement a une forme rectangulaire.
- 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.
- Un groupement doit respecter les axes de symétries du tableau de Karnaugh.

MÉTHODE DE KARNAUGH

Construction de la table de Karnaugh

Fonction Majorité:

Table de vérité

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 de Karnaugh

x\yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

MÉTHODE DE KARNAUGH

Regroupement des 2 cases adjacentes

La réunion de deux cases adjacentes contenant 1 chacune élimine une seule variable celle qui change d'état en passant d'une case à l'autre.

- Simplification de la fonction Majorité de 3 variables $F(x, y, z)$

$x \setminus yz$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Groupement bleu : $x\bar{y}z + xyz = xz$

Groupement rouge : $\bar{x}yz + xyz = yz$

Groupement vert : $xyz + xy\bar{z} = xy$

Donc la forme simplifiée de la fonction Majorité est $F(x, y, z) = xy + yz + xz$

MÉTHODE DE KARNAUGH

Regroupement des 4 cases adjacentes

Lorsqu'on regroupe **4 cases adjacentes** alors **2 variables disparaissent**, on peut alors remplacer la somme des 4 cases par un seul terme qui comporte que 2 variables uniquement.

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

AB/CD	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	1	1	0	1
10	0	0	0	1

The Karnaugh map shows two adjacent 2x2 groups of cells. The first group, located at the top-right corner (01 row, 10 column), is highlighted with a blue border. The second group, located at the bottom-right corner (11 row, 10 column), is highlighted with a green border. These groups represent the terms $B\bar{C}$ and $C\bar{D}$ respectively.

MÉTHODE DE KARNAUGH

Regroupement des 4 cases adjacentes

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

AB/CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	0	0	1
10	1	0	0	1

MÉTHODE DE KARNAUGH

Regroupement des 4 cases adjacentes

- Fonction $F_3(A, B, C, D) = \bar{C}\bar{D} + AB + \bar{B}C$

$\bar{A}\bar{B}/\bar{C}\bar{D}$	00	01	11	10
00	1	0	1	1
01	1	0	0	0
11	1	1	1	1
10	1	0	1	1

MÉTHODE DE KARNAUGH

Regroupement des 8 cases adjacentes

Un groupement de **8 cases adjacentes** ou symétriques remplies des 1 logiques va simplifier **3 variables logiques** dans l'expression canonique de la fonction logique.

- Fonction $F_4(A, B, C, D) = \bar{D}$

AB/CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

MÉTHODE DE KARNAUGH

Regroupement des 8 cases adjacentes

- Fonction $F_5(A, B, C, D) = \bar{B}$

AB/CD	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

MÉTHODE DE KARNAUGH

Traitements des problèmes à 5 variables

Pour résoudre ce problème on va le décomposer en **2 problèmes à 4 variables** en appliquant le théorème d'expansion (théorème de Shannon).

$$\text{On a : } F_7(A, B, C, D, E) = \bar{E} \cdot F_7(A, B, C, D, 0) + E \cdot F_7(A, B, C, D, 1)$$

- Fonction $F_7(A, B, C, D, 0) = CD$

AB/CD	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	0	1
10	0	0	0	1

MÉTHODE DE KARNAUGH

Traitement des problèmes à 5 variables

- Fonction $F_7(A, B, C, D, 1) = \bar{C}D$

AB/CD	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

Ce qui donne : $F_7(A, B, C, D, E) = \bar{E}\bar{C}\bar{D} + E\bar{C}D$

MÉTHODE DE KARNAUGH

Les valeurs indifférentes ou indéfinies

- Le symbole \emptyset (ou X ou *) peut prendre **indifféremment la valeur 0 ou 1**
 - On considère que la valeur est 1 uniquement pour ceux qui permettent d'augmenter le nombre des cases d'un regroupement et ceux qui réduisent le nombre de regroupement
- Fonction $F_8(A, B, C, D) = \bar{A}C$

AB/CD	00	01	11	10
00	∅	0	∅	1
01	0	0	1	∅
11	0	0	0	0
10	0	0	0	0

MÉTHODE DE KARNAUGH

Les valeurs indifférentes ou indéfinies

- Fonction $F_9(A, B, C, D) = C\bar{D} + B\bar{D} + A\bar{D}$

AB/CD	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	\emptyset	\emptyset	\emptyset	\emptyset
10	1	0	\emptyset	\emptyset

The Karnaugh map shows the function $F_9(A, B, C, D)$. The rows and columns are labeled with AB/CD values. The map contains the following values:

- (00, 00) = 0
- (01, 00) = 1
- (11, 00) = \emptyset
- (10, 00) = 1
- (00, 01) = 0
- (01, 01) = 0
- (11, 01) = \emptyset
- (10, 01) = 0
- (00, 11) = 0
- (01, 11) = 0
- (11, 11) = \emptyset
- (10, 11) = \emptyset
- (00, 10) = 1
- (01, 10) = \emptyset
- (11, 10) = \emptyset
- (10, 10) = \emptyset

Two rectangles are drawn on the map: one red rectangle covers the (01,00) and (10,00) cells, and one blue rectangle covers the (10,00) and (10,10) cells.