
Université Claude Bernard Lyon 1

Licence STS L1

# Bases de l'architecture pour la programmation - LIFBAP -

Hamid LADJAL

hamid.ladjal@univ-lyon1.fr

hamid.ladjal@liris.cnrs.fr

•http://perso.univ-lyon1.fr/hamid.ladjal/LIFBAP/

# COORDONNÉES ET SITE WEB

## Responsable de L'UE :

**Hamid LADJAL**
Bâtiment Nautibus (2ème étage)
Tel : 04 72 43 16 36
Mél : hamid.ladjal@liris.cnrs.fr
hamid.ladjal@univ-lyon1.fr

## Responsables d'amphi, TD et TP :

**Hamid LADJAL** (mercredi matin et après midi ) + d'autres intervenants

Site WEB de l'UE (pour infos pratiques, supports, corrections …)
http://perso.univ-lyon1.fr/hamid.ladjal/LIFBAP/
https://perso.liris.cnrs.fr/hamid.ladjal/LIFBAP/supports.html

# Détail des enseignements de l'UE

**CM : 5 séances de 1h30**

- Présentation des concepts fondamentaux
- Notions de base de l'architecture des ordinateurs
- Illustration par des exemples

**TD : 7 séances de TD de 1h30**

- Mieux assimiler les notions de bases
- Analyser un problème et le formaliser
- Apprendre à poser et résoudre des problèmes
- Posséder certaines démarches pour le résoudre

**TP : 4 séances de TP de 3h00**

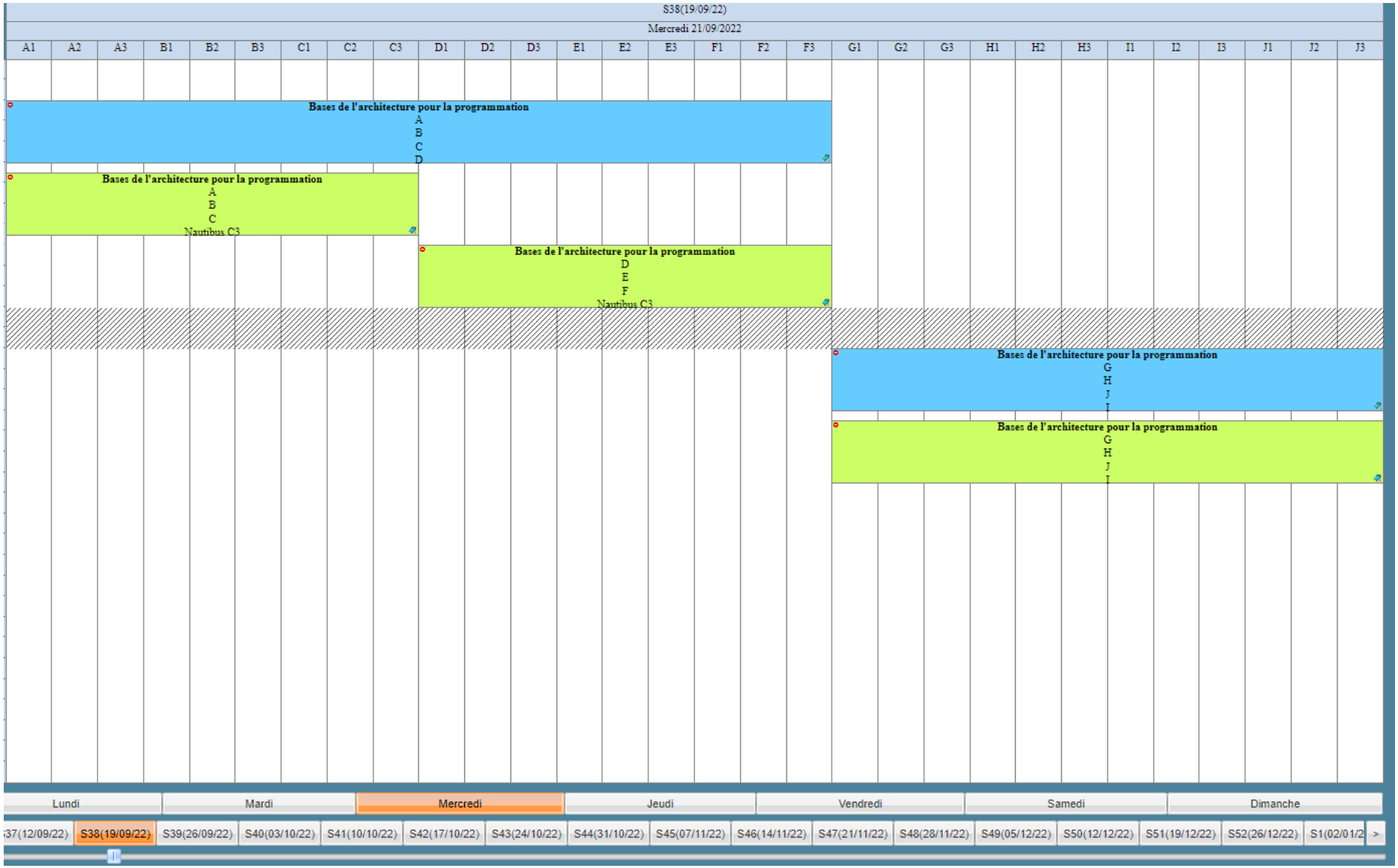
- Réaliser et programmer des circuits combinatoires et séquentiels simples
- Savoir réaliser et mettre en pratique les notions vues en cours et en TD

# Emploi du temps

| | Mercredi Matin (Grp: A, B, C, D, E, F, G ,H, I, J) | | |
|---|---|---|---|
| **Mercredi** | **CM** | **TD** | **TP** |
| 14/09/2022 | CM_1 (11h30 -13h00) (groupes A, B, C D, E, F )<br>CM_1 (14h00 -15h30) (groupes G, H, I , J ) | | |
| 21/09/2022 | CM_2 (8h00 -9h30) (groupes A, B, C D, E, F )<br>CM_2 (14h00 -15h30) (groupes G, H, I , J ) | TD1 (9h45 - 11h15)(A,B, C)<br>TD1(11h30 - 13h00)(D, E, F)<br>TD1(15h45 - 17h15)(G, H, I, J) | |
| 28/09/2022 | CM_3 (8h00 -9h30) (groupes A, B, C D, E, F )<br>CM_3 (14h00 -15h30) (groupes G, H, I , J ) | TD2 (9h45 - 11h15)(A,B, C)<br>TD2(11h30 - 13h00)(D, E, F)<br>TD2(15h45 - 17h15)(G, H, I, J) | |
| 05/10/2022 | CM_4 (14h00 -15h30) (groupes A, B, C D, E, F G, H, I , J ) | TD3 (9h45 - 11h15)(A,B, C)<br>TD3(11h30 - 13h00)(D, E, F)<br>TD3(15h45 - 17h15)(G, H, I, J) | |
| 12/10/2022 | CM_5 (14h00 -15h30) (groupes A, B, C D, E, F G, H, I , J ) | TD4 (9h45 - 11h15)(A,B, C)<br>TD4(11h30 - 13h00)(D, E, F)<br>TD4(15h45 - 17h15)(G, H, I, J) | |
| 19/10/2022 | | | TP_1(9h45-13h00)<br>TP_1(14h00-17h15) |
| 26/10/2022 | | TD5 (9h45 - 11h15)(A,B, C)<br>TD5(11h30 - 13h00)(D, E, F)<br>TD5(14h00 - 15h30)(G, H, I, J) | Séance soutien 15h45 -17h15 |
| 02/11/2022 | Banalisé | Banalisé | Banalisé |
| 09/11/2022 | | | TP_2(9h45-13h00)<br>TP_2(14h00-17h15) |
| 16/11/2022 | | TD6 (9h45 - 11h15)(A,B, C)<br>TD6(11h30 - 13h00)(D, E, F)<br>TD6(14h00 - 15h30)(G, H, I, J) | |
| 23/11/2022 | | | TP_3(9h45-13h00)<br>TP_3(14h00-17h15) |
| **30/11/2022** | | TD7 (9h45 - 11h15)(A,B, C)<br>TD7(11h30 - 13h00)(D, E, F)<br>TD7(14h00 - 15h30)(G, H, I, J) | Séance soutien 15h45 -17h15 |
| 07/12/2022 | | | TP4_(9h45-13h00)<br>TP_4(14h00-17h15) |

**ADE WEB: http://adelb.univ-lyon1.fr/direct/**

# Emploi du temps




**ADE WEB: http://adelb.univ-lyon1.fr/direct/**

# MODALITÉ DE CONTRÔLE DES CONNAISSANCES (MCC)

**En TD (30% de la note finale):**

- Un contrôle de présence à chaque séance
- 2 interrogations de 30 min environ (15% chacune)

**En TP (30% de la note finale) :**

- 1 TP noté de 1h00 en fin se semestre

**Contrôle final (40% de la note finale)**

- Épreuve de 1h30 sans document, anonyme
- Questions de cours, exercices….

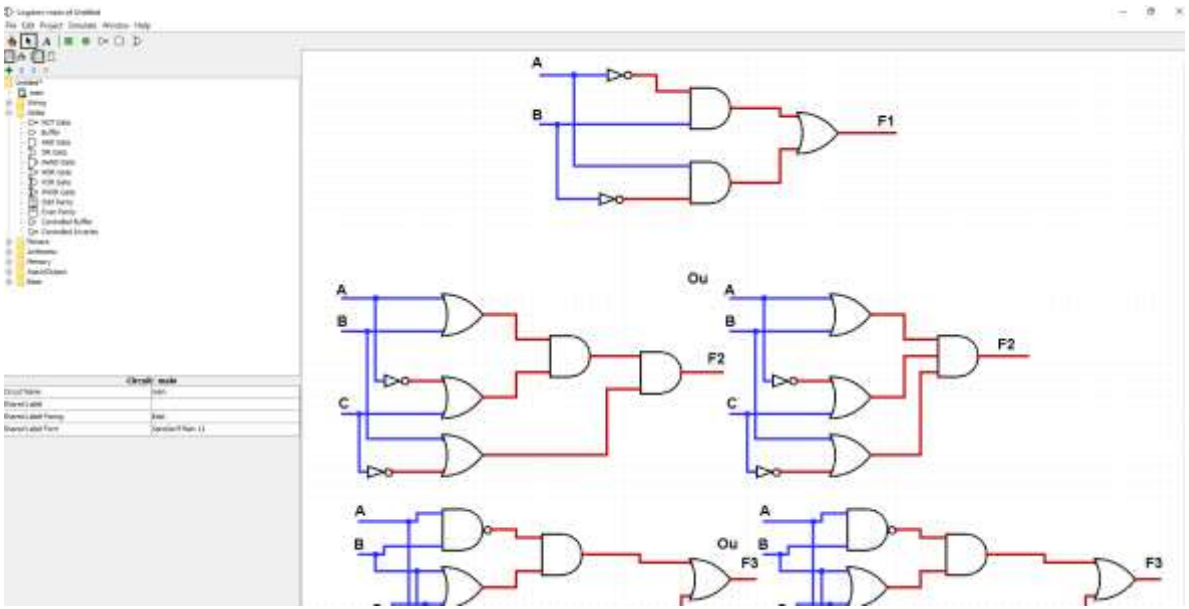
**Absences seront contrôlées** à chaque séance de TD et TP

- Justificatif en cas d'absence (=>enseignant de TD/TP validé par la scolarité)
- Une influence sur la note de l'UE

# INFOS PRATIQUES

## Environnement et outils de travail

- Linux / windows
- **Logisim**: Un outil pour le design et la simulation de circuits logiques numériques
- **https://logisim.fr.uptodown.com**



# Conseils sur la méthode de travail

**Pdf = uniquement copies des transparents**

**=>**Prendre des notes

(en particulier exercices)

Pour vous aider : transparents numérotés

- **Savoir refaire les exercices et les TP**

Temps de travail estimé :

– Après un CM 1h - 1h30

– Après un TD 1h30 - 2h00

• **Cours avec complexité croissante**

# Plan

1) L’algèbre de Boole, la logique combinatoire et les circuits combinatoires

2) Circuits séquentiels

3) Représentation et codage des données


Université Claude Bernard Lyon 1

Licence STS
L1

# CM1: Logique combinatoire et les circuits combinatoires

hamid.ladjal@univ-lyon1.fr
hamid.ladjal@liris.cnrs.fr

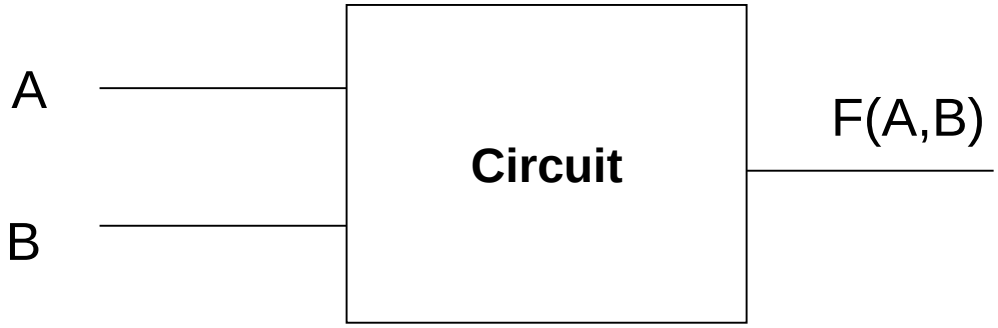
•http://perso.univ-lyon1.fr/hamid.ladjal/LIFBAP/

# Logique combinatoire

- L’algèbre de Boole
- Opérateurs de base
- Propriétés et les fonctions combinatoires
- Circuits combinatoires:
    - Multiplexeur et démultiplexeur
    - Codeur, décodeur et transcodeur
    - Additionneur et comparateur….

# Introduction

- Les machines numériques (ordinateur, tablette, téléphone…) sont constituées d'un ensemble de circuits électroniques.

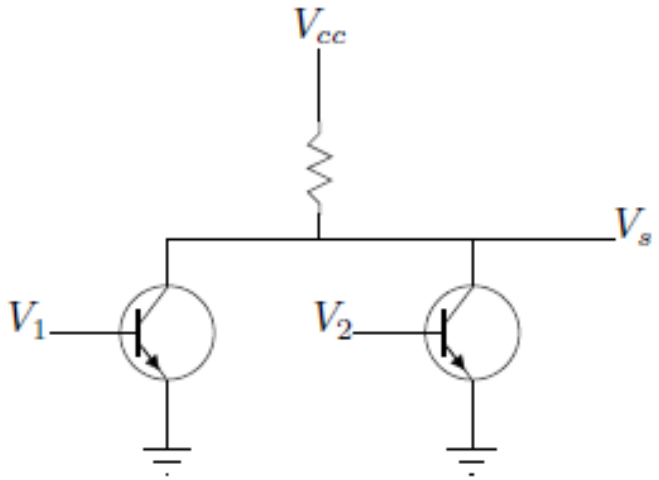


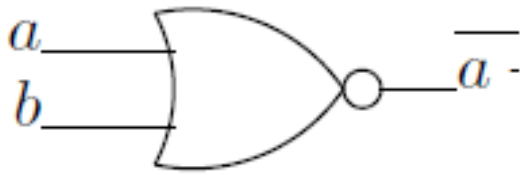

- Chaque circuit fournit une fonction logique bien déterminée; opérations logiques ou arithmétiques (addition, soustraction, comparaison ,….).

# Introduction

- Une fonction logique de base est réalisée à l'aide des portes logiques qui permettent d'effectuer des opérations élémentaires.

- Ces portes logiques sont aujourd'hui réalisées à l'aide de transistors.



# Introduction

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

- Ce modèle doit prendre en considération le système binaire.

- Le modèle mathématique utilisé est celui de Boole.

# Algèbre de Boole

1854 : Georges Boole propose une algèbre

Propositions vraies ou fausses
et opérateurs possibles ⟶ Algèbre de Boole

↓

Étude des systèmes binaires :
Possédant deux états s'excluant mutuellement

C'est le cas des systèmes numériques
(des sous ensembles : les circuits logiques)

# Algèbre binaire

On se limite : Base de l'algèbre de Boole
Propriétés indispensables aux systèmes logiques

Définitions :

- États logiques : 0 et 1, Vrai et Faux, H et L
  (purement symbolique)

- Variable logique : Symbole pouvant prendre
  comme valeur des états logiques (A,b,c, Out ...)

- Fonction logique : Expression de variables et d'opérateurs
  ( f = not(a)^ (c OR r.t) )

# Calcul propositionnel

Algèbre de Boole sur [0,1] = algèbre binaire
Structure d'algèbre de Boole

- 2 lois de composition interne (LCI)
- 1 application unaire

2 LCI : ET, OU

- Somme (OU, Réunion, Disjonction)

  $s = a + b = a \vee b$
- Produit (ET, intersection, Conjonction)

  $s = a \cdot b = ab = a \wedge b$

Application unaire :

- Not (complémentation, inversion, négation, non) $s = \overline{a} = not(a) = \neg a$

# Fonctions logiques

Fonction logique à n variables f(a,b,c,d,...,n)

$[0,1]^n \longrightarrow [0,1]$

- Une fonction logique ne peut prendre que deux valeurs
- Les cas possibles forment un ensemble fini ( $2^n$)
- La table de fonction logique = table de vérité

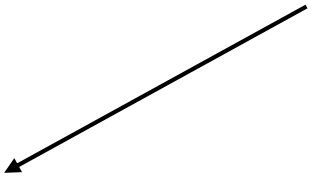
# Opérateurs logiques de base

# OU ( OR )

- Le OU est un opérateur binaire ( deux variables) , à pour rôle de réaliser la somme logique entre deux variables logiques.
- Le OU fait la disjonction entre deux variables.
- Le OU est défini par F(A,B)= A + B ( il ne faut pas confondre avec la somme arithmétique )

| A | B | A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

table de vérité



# ET ( AND )

- Le ET est un opérateur binaire ( deux variables) , à pour rôle de réaliser le Produit logique entre deux variables booléennes.
- Le ET fait la conjonction entre deux variables.

- Le ET est défini par : F(A,B)= A . B

| A | B | A . B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# NON ( négation )

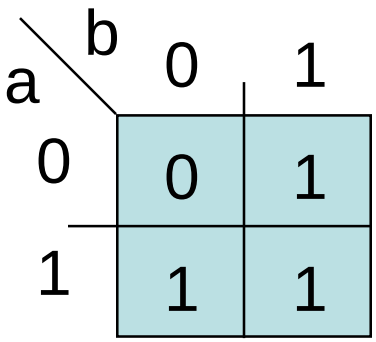
- NON : est un opérateur unaire ( une seule variable) qui à pour rôle d'inverser la valeur d'une variable .

$$F(A)= \text{Non } A = \overline{A}$$

( lire : A barre )

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

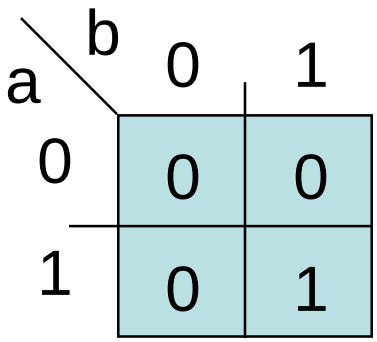
# Tables de vérité de ET, OU, NON




$s = a + b$

S est vrai si a OU b est vrai.

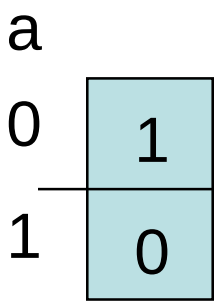
| a | b | s |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




$s = a \cdot b$

S est vrai si a ET b sont vrais.

| a | b | s |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




$s = \overline{a}$

S est vrai si a est faux

| a | s |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Deux autres opérateurs : NAND,NOR

| a \ b | 0 | 1 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 0 |

$s = a \uparrow b = \overline{a.b}$

S est vrai si a OU b est faux.

NAND (Not-AND)

| a \ b | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 0 |

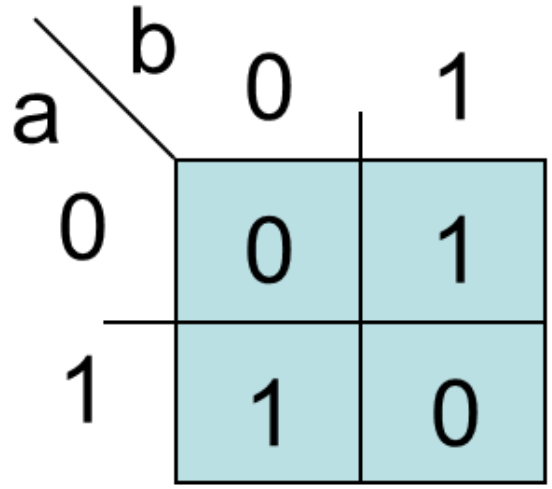
$s = a \downarrow b = \overline{a+b}$

S est vrai si ni a, ni b ne sont vrais.

NOR (Not-OR)

NAND et NOR ne sont pas associatifs

# Encore un opérateur : XOR




$$s = a \oplus b = a.\overline{b} + \overline{a}.b$$

S est vrai si a OU b est vrai mais pas les deux.

XOR (Ou-Exclusif) vaut 1 si a est différent de b
Opérateur de différence (disjonction)

# Encore un opérateur : XOR

XOR est associatif $s = a \oplus b \oplus c \quad .....\oplus n$

vaut 1 si le nombre de variables à 1 est impair.

$$s = \overline{a \oplus b} = \overline{a} \oplus b = a \oplus \overline{b} = a \text{ XNOR } b$$

XNOR = $\overline{\text{XOR}}$ vaut 1 si $a = b$

Inverseur programmable : (le programme vaut 0 ou 1)

$a \oplus 1 = \overline{a} \qquad a \oplus 0 = a$

# Simplification des fonctions logiques

# Simplification /optimisation ?

Méthodes «classiques» de simplifications :

- pas de solution unique
- indépendant de la technologie
- le temps n'est pas pris en compte

La simplification «mathématique» n'est pas toujours optimale en regard des critères d'optimisation technologiques.

# Simplification des fonctions logiques

- L'objectif de la simplification des fonctions logiques est de :
  - réduire le nombre de termes dans une fonction
  - et de réduire le nombre de variables dans un terme

- Cela afin de réduire le nombre de portes logiques utilisées → réduire le coût du circuit

- Plusieurs méthodes existent pour la simplification :

1) **Les méthodes algébriques**
2) **Les méthodes graphiques : ( ex : tableaux de karnaugh )**

# Propriétés de ET,OU,NON

## 1) Les méthodes algébriques

- **Commutativité**

$a+b = b+a$

$a.b = b.a$

- **Associativité**

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

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

- **Distributivité**

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

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

- **Idempotence**

$a+a = a$

$a.a = a$

- **Absorption**

$a+a.b = a$

$a.(a+b) = a$

- **Involution**

$\overline{\overline{a}} = a$

# Propriétés de ET,OU,NON

## Les méthodes algébriques

- **Elément neutre**

$a+0 = a$

$a.1 = a$

- **Elément absorbant**

$a+1 = 1$

$a.0 = 0$

- **Inverse**

$a+\overline{a} = 1$

$a.\overline{a} = 0$

- **Théorème de "De Morgan"**

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

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

$$\overline{\sum_i x_i} = \prod_i \overline{x_i}$$

$$\overline{\prod_i x_i} = \sum_i \overline{x_i}$$

- **Théorème du Consensus**

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

$(a+x)(\overline{b}+x)(a+b)=(a+x)(\overline{b}+x)$

# Propriétés de ET,OU,NON

**Exercice 1:**

Démontrez la proposition suivante :

$$ABC + AB\overline{C} + A\overline{B}CD = AB + ACD$$

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

# Propriétés de ET,OU,NON

## Correction

$$ABC + AB\overline{C} + A\overline{B}CD = AB(C + \overline{C}) + A\overline{B}CD$$
$$= AB + A\overline{B}CD$$
$$= A(B + \overline{B}(CD))$$
$$= A(B + CD)$$
$$= AB + ACD$$

$$A.B.C + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} =$$
$$A.B.C + \overline{A}.B.C + A.B.C + A.\overline{B}.C + ABC + A.B.\overline{C} =$$
$$B.C + A.C + A.B$$

# Simplification par la table de Karnaugh

# Description de la table de karnaugh

• La méthode consiste à mettre en évidence par une méthode graphique (un tableau ) tous les termes qui sont adjacents (qui ne différent que par l'état d'une seule variable).

• Un tableau de Karnaugh = table de vérité de $2^n$ cases avec un changement unique entre 2 cases voisines d'où des codes cycliques (Gray ou binaire réfléchi).

• La méthode peut s'appliquer aux fonctions logiques de 2,3,4,5 et 6 variables.

• Les tableaux de Karnaugh comportent $2^n$ cases ( n: est le nombre de variables ).

# Description de la table de karnaugh

## Règles de regroupement :

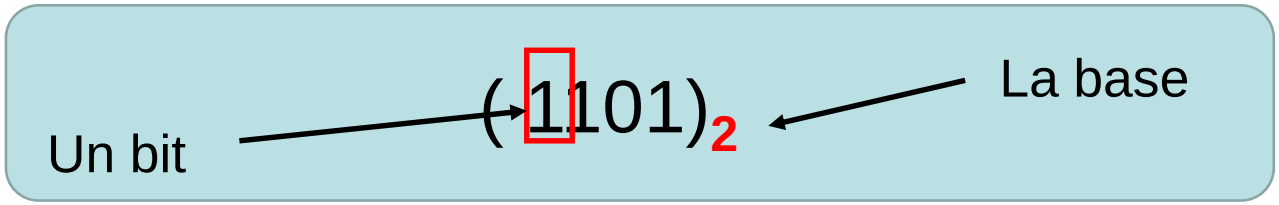
- groupe de $2^n$ cases : 1,2,4 ou 8
- en ligne, colonne, rectangle, carré, mais pas diagonale
- tous les 1, mais pas les 0 au moins une fois dans les groupements

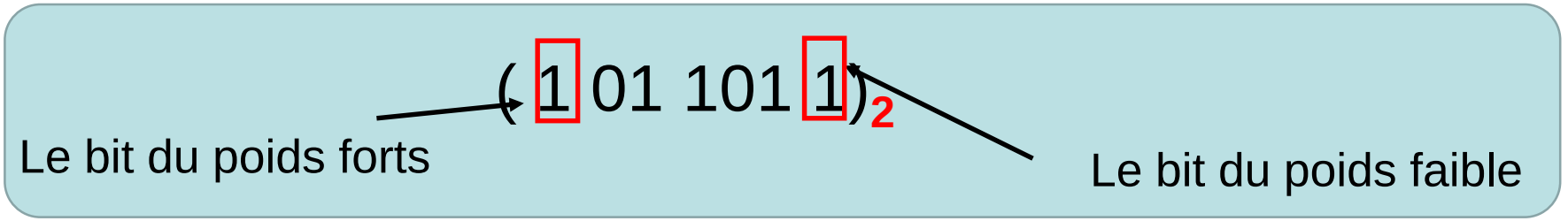
## Règles de minimisation de la fonction :

- rechercher les groupements en commençant par les cases qui n'ont qu'une seule façon de se grouper
- rechercher les groupements les plus grands
- les groupements doivent contenir au moins un 1 non utilisé par les autres groupements
- L'expression logique finale est la réunion ( la somme ) des groupements après simplification et élimination des variables qui changent d'état.

# Système binaire

- Dans le système binaire, pour exprimer n'importe quelle valeur on utilise uniquement 2 symboles : { 0 , 1}




. Un nombre dans la base 2 peut être écrit aussi sous la forme polynomiale

# Système binaire

**Exemple**

- Sur **un seul bit** : 0 , 1

- **Sur 2 bits :**

| Binaire | Décimal |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

$2^1 2^0$

**4 combinaisons= $2^2$**

**Sur 3 Bits**

$2^2 2^1 2^0$

| Binaire | Décimal |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

**8 combinaisons= $2^3$**

**Sur 4 Bits**

| Binaire | Décimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

**16 combinaisons= $2^4$**

# Description de la table de karnaugh

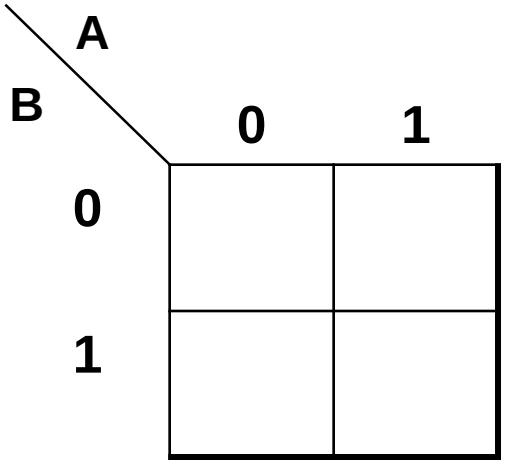
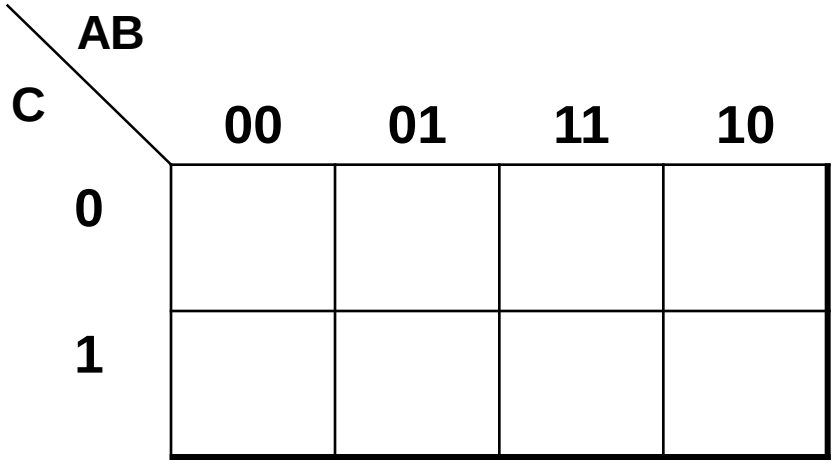



**Tableau à 2 variables**

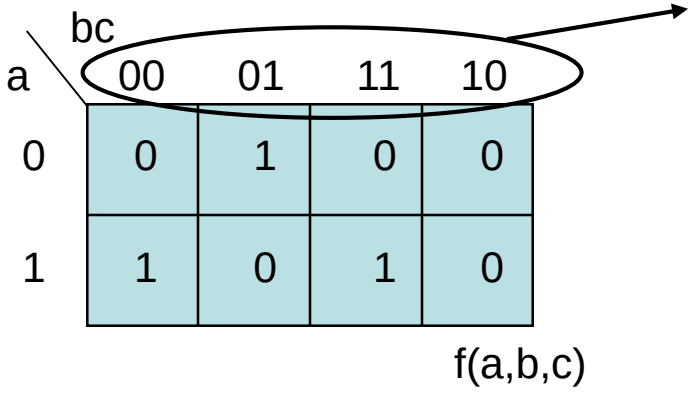



**Tableaux à 3 variables**

# Tableaux de Karnaugh

f (a,c,d, ..,n) fonction logique à N entrées sera représentée par
une table à $2^N$ lignes
un tableau à $2^N$ cases

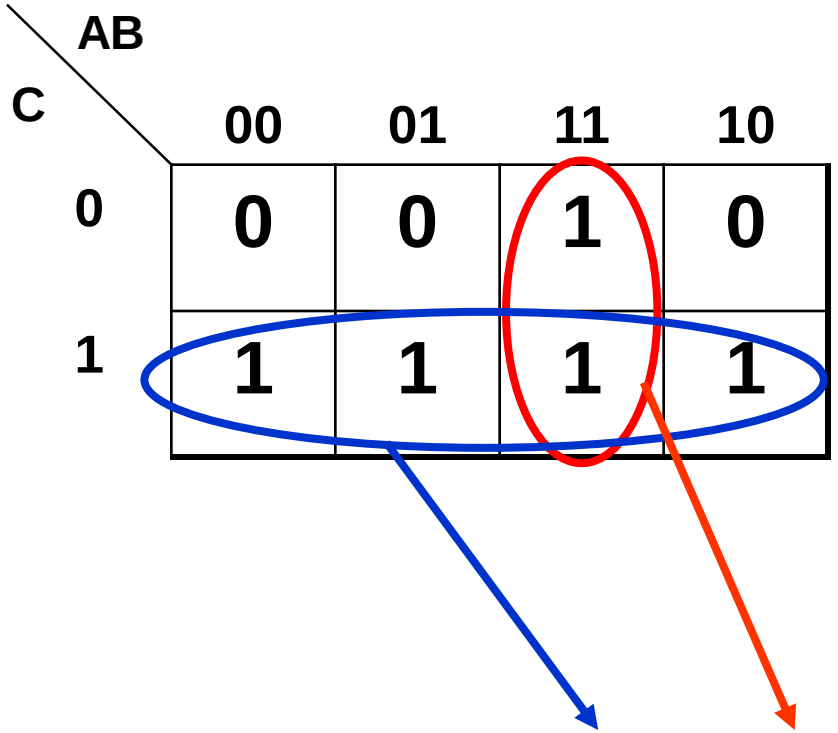
| a b c | f(a,b,c) |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 1 |
| 0 1 0 | 0 |
| 0 1 1 | 0 |
| 1 0 0 | 1 |
| 1 0 1 | 0 |
| 1 1 0 | 0 |
| 1 1 1 | 1 |




Code Gray ou
binaire réfléchi
=
1 seul changement
entre 2 codes
successifs

# Tableaux de Karnaugh

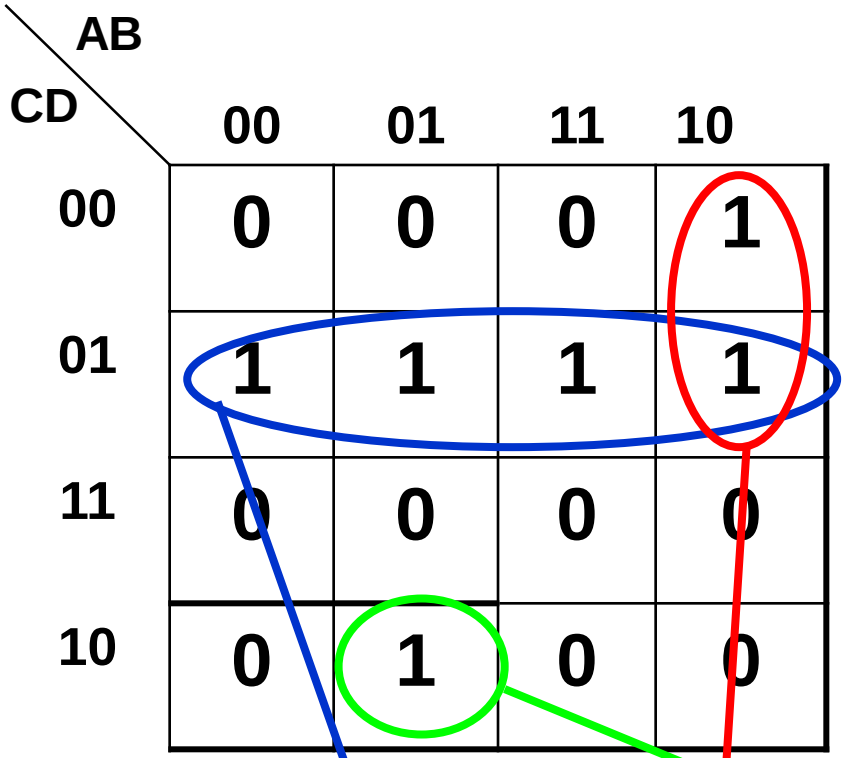
## Exemple 1 : 3 variables




$$F(A,B,C) = C + AB$$

# Tableaux de Karnaugh

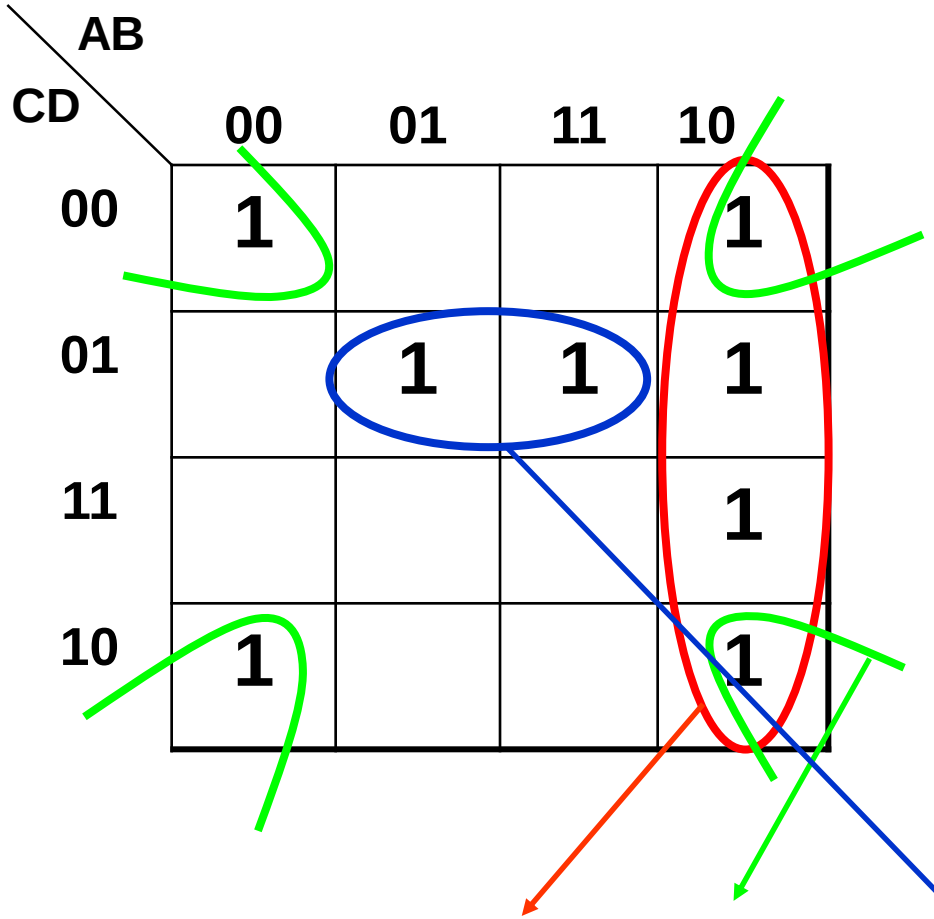
## Exemple 2 : 4 variables




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

# Tableaux de Karnaugh

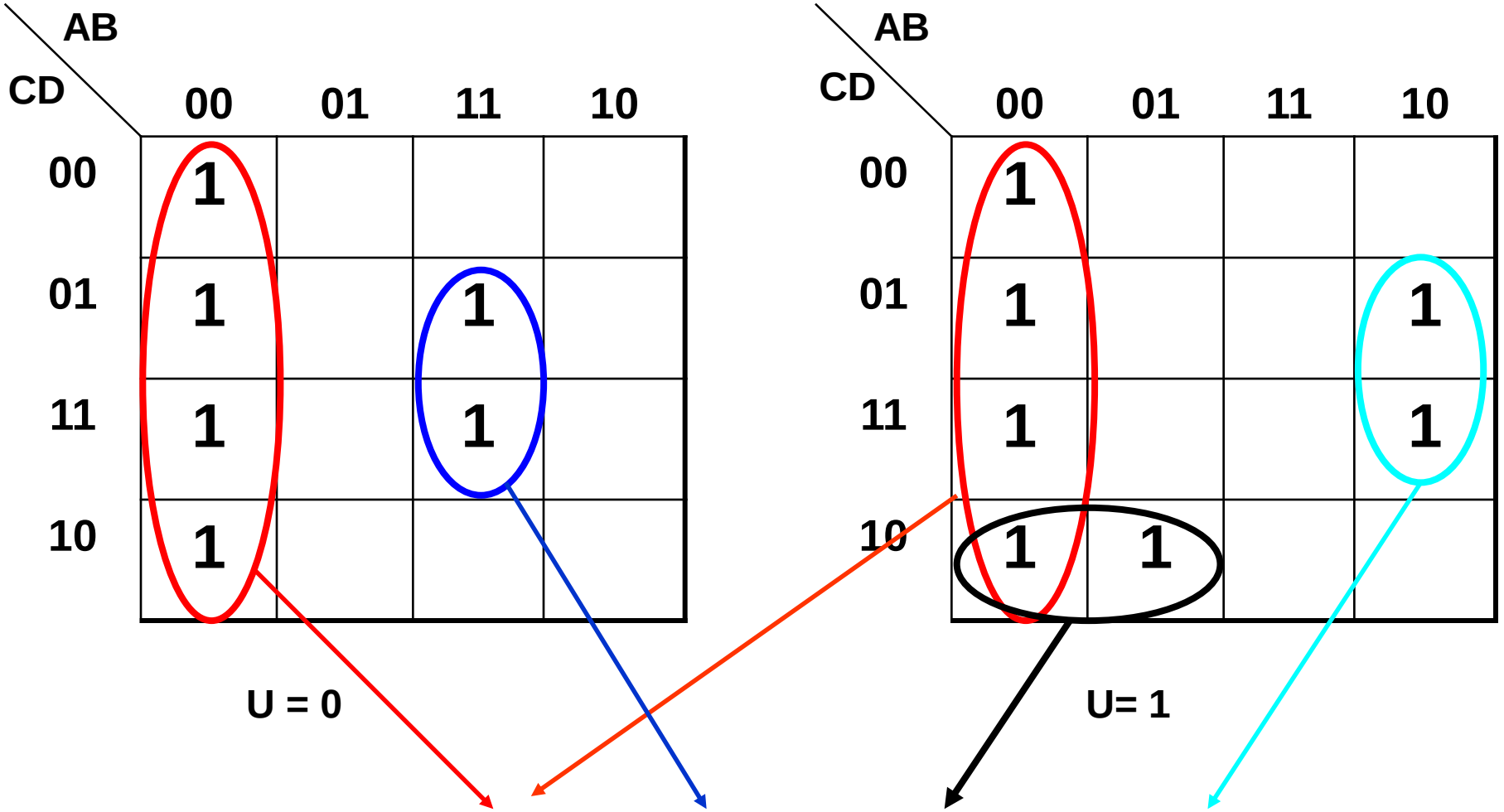
## Exemple 3 : 4 variables




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

# Tableaux de Karnaugh

## Exemple 4 : 5 variables




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

# Exercice

**Trouver la forme simplifiée des fonctions à partir des deux tableaux ?**

| C \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | 1 | 1 | 1 |
| 1 | 1 | | 1 | 1 |

| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | 1 | 1 |
| 01 | | | | |
| 11 | | | | |
| 10 | 1 | 1 | 1 | 1 |

# Logique multi-niveaux

On peut généraliser l'algèbre binaire à plus de 2 niveaux

| a \ b | 0 | 1 | Z | X |
|---|---|---|---|---|
| 0 | 0 | X | 0 | X |
| 1 | X | 1 | 1 | X |
| Z | 0 | 1 | Z | X |
| X | X | X | X | X |

0 logique

1 logique

Z déconnecté

X inconnu

# Logique multi-niveaux

•Pour les cas impossibles ou interdites

il faut mettre un X dans la T.V .

•Les cas impossibles sont représentées

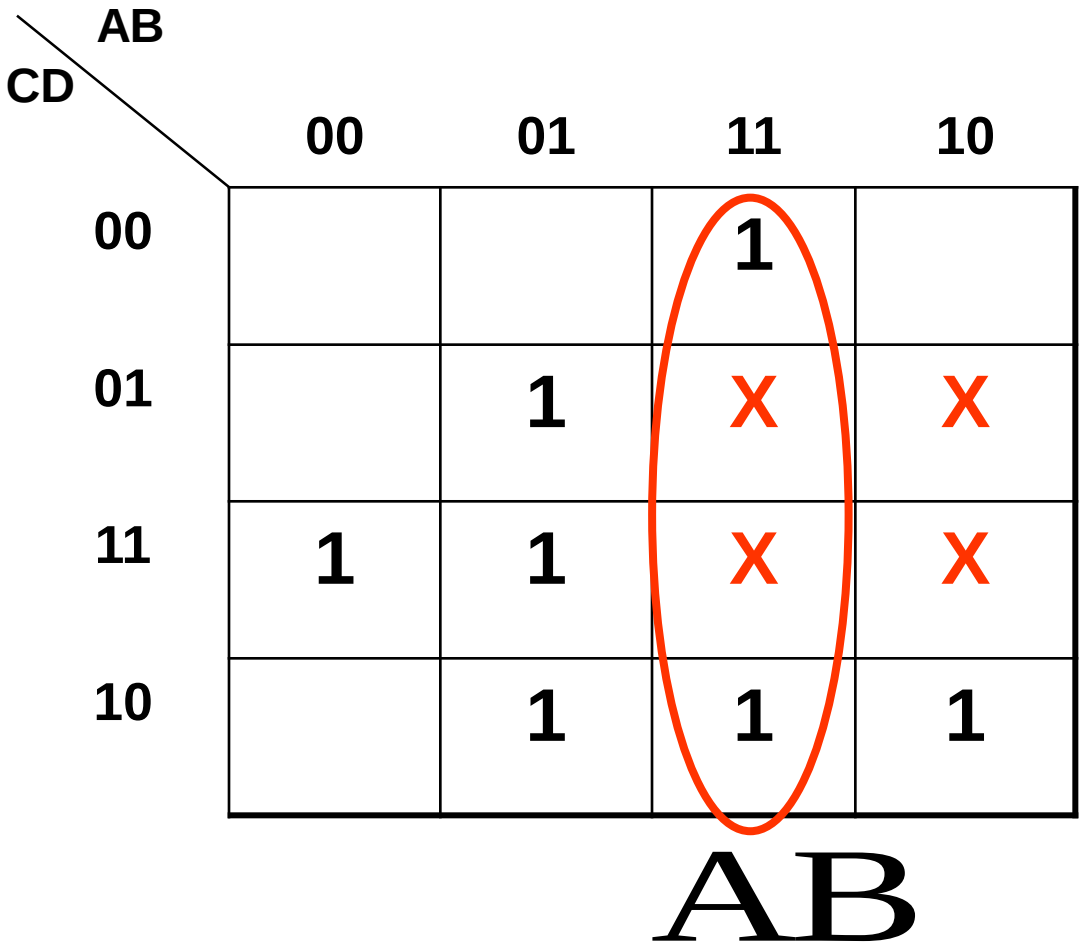
aussi par des X dans la table de karnaugh

| AB / CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | |
| 01 | | 1 | X | X |
| 11 | 1 | 1 | X | X |
| 10 | | 1 | 1 | 1 |

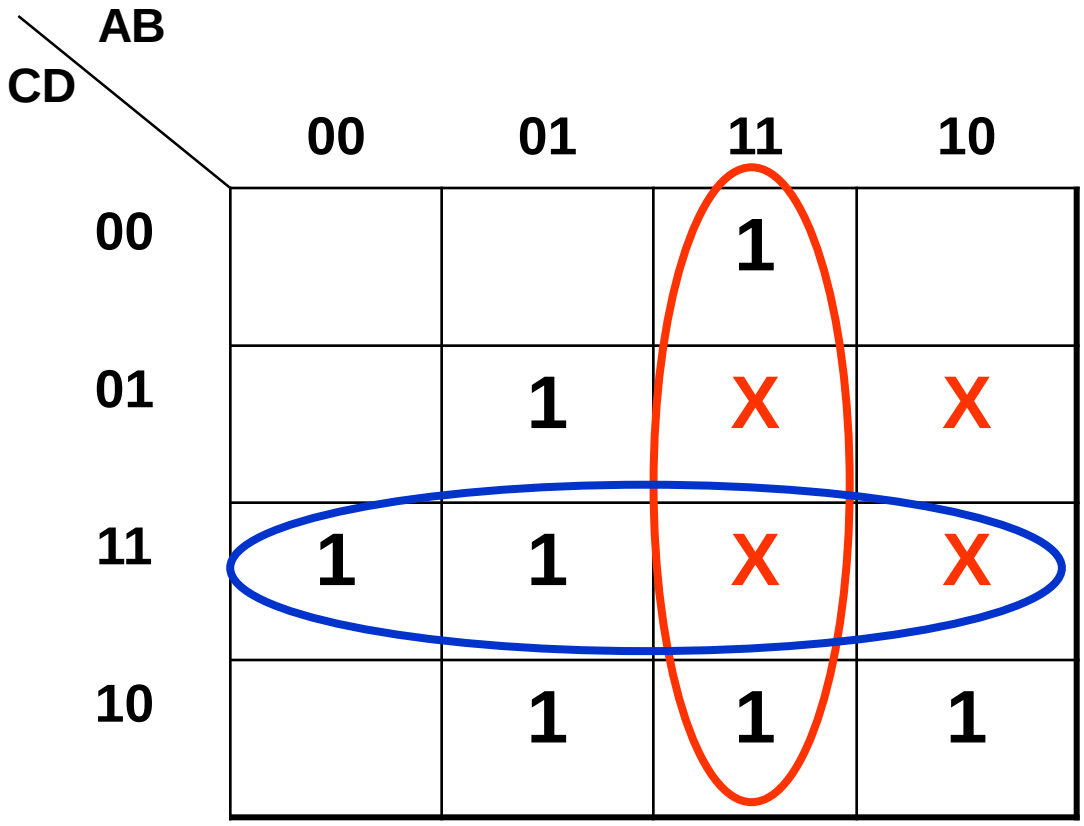
| A | B | C | D | | S |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 0 |
| 0 | 0 | 0 | 1 | | 0 |
| 0 | 0 | 1 | 0 | | 0 |
| 0 | 0 | 1 | 1 | | 1 |
| 0 | 1 | 0 | 0 | | 0 |
| 0 | 1 | 0 | 1 | | 1 |
| 0 | 1 | 1 | 0 | | 1 |
| 0 | 1 | 1 | 1 | | 1 |
| 1 | 0 | 0 | 0 | | 0 |
| 1 | 0 | 0 | 1 | | X |
| 1 | 0 | 1 | 0 | | 1 |
| 1 | 0 | 1 | 1 | | X |
| 1 | 1 | 0 | 0 | | 1 |
| 1 | 1 | 0 | 1 | | X |
| 1 | 1 | 1 | 0 | | 1 |
| 1 | 1 | 1 | 1 | | X |

# Tableaux de Karnaugh

- Il est possible d'utiliser les X dans des regroupements :
  - Soit les prendre comme étant des 1
  - Ou les prendre comme étant des 0
- Il ne faut pas former des regroupement qui contient uniquement des X

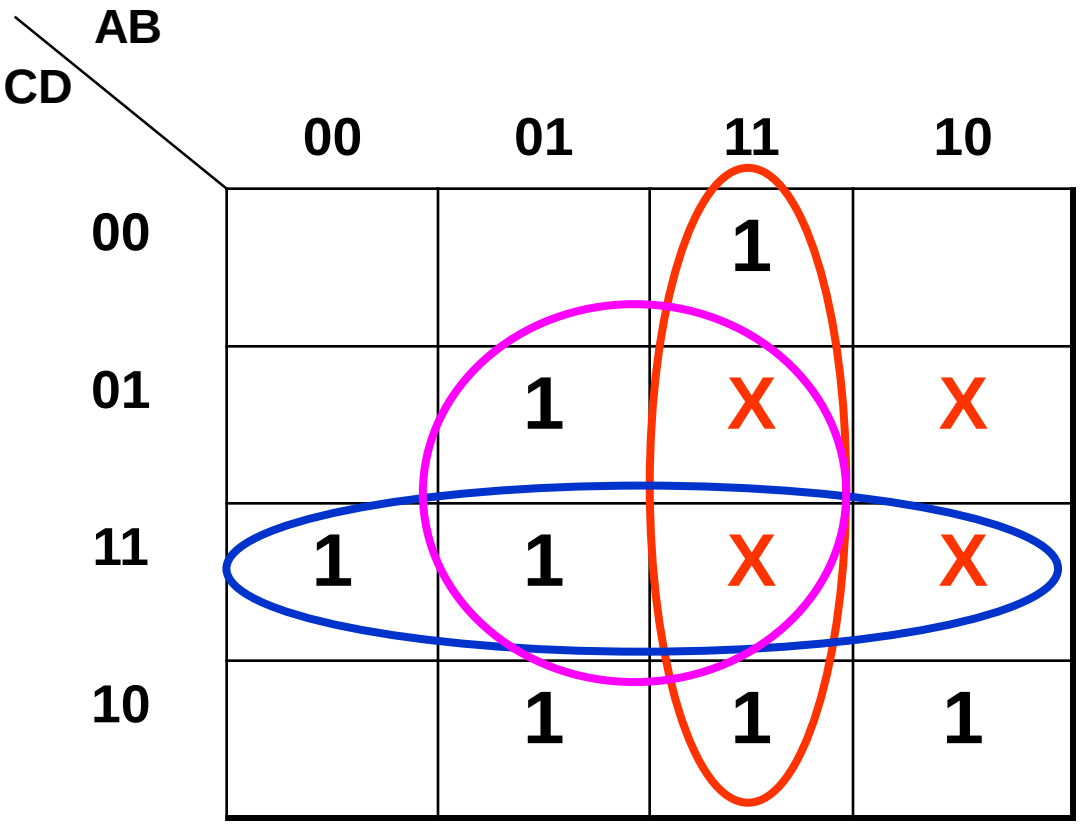


# Tableaux de Karnaugh



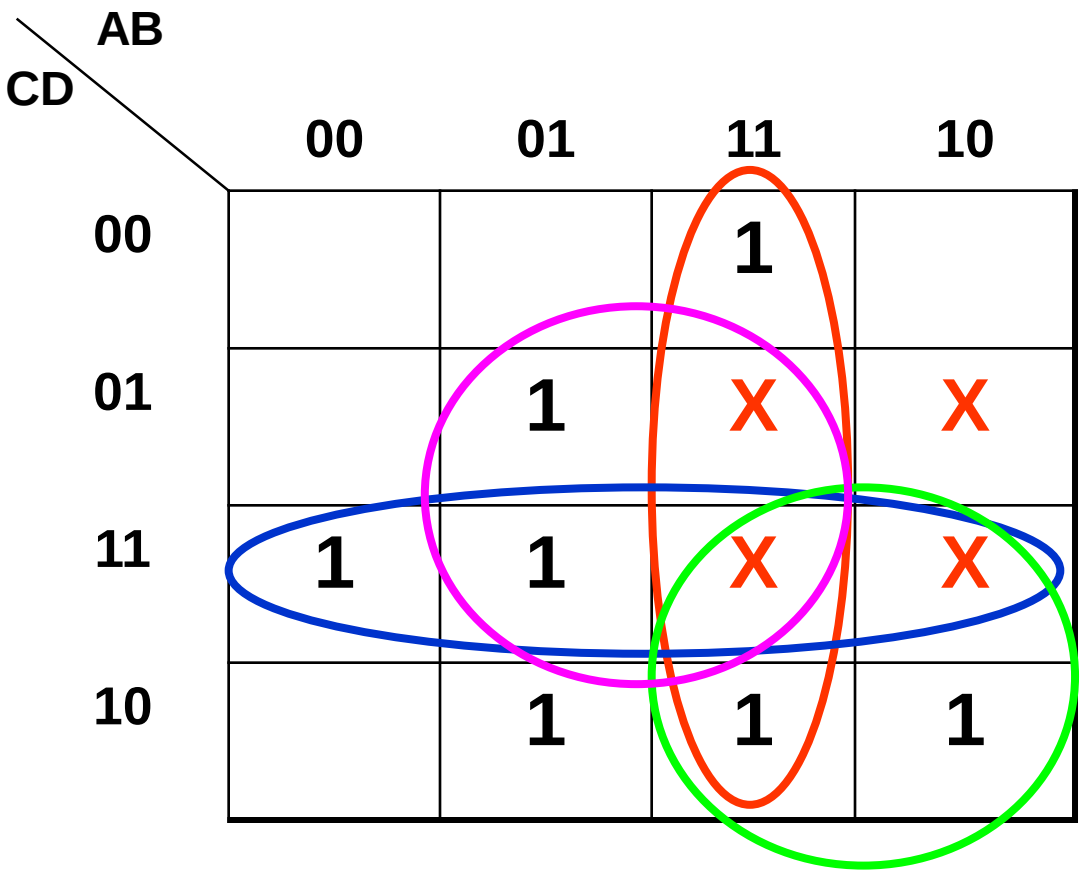

$$AB + CD$$

# Tableaux de Karnaugh



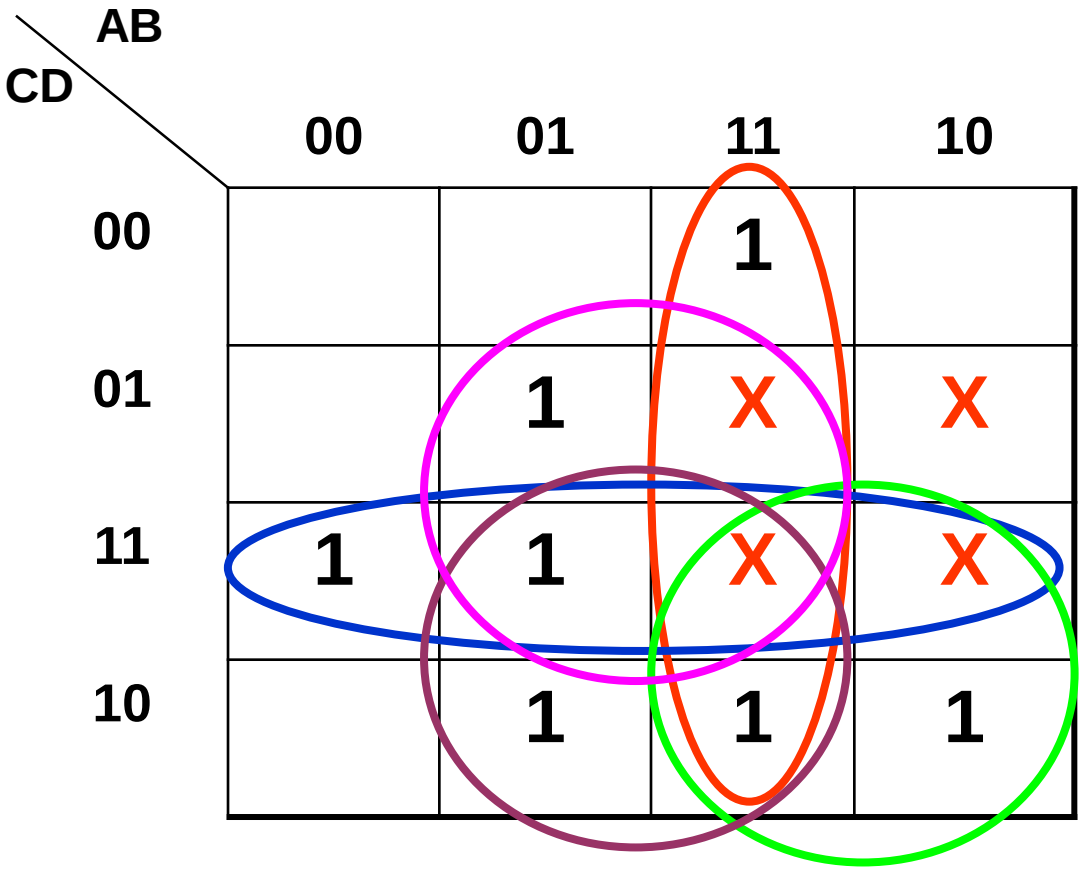

$$AB + CD + BD$$

# Tableaux de Karnaugh




$$AB + CD + BD + AC$$

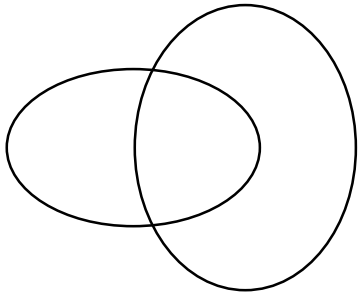
# Tableaux de Karnaugh



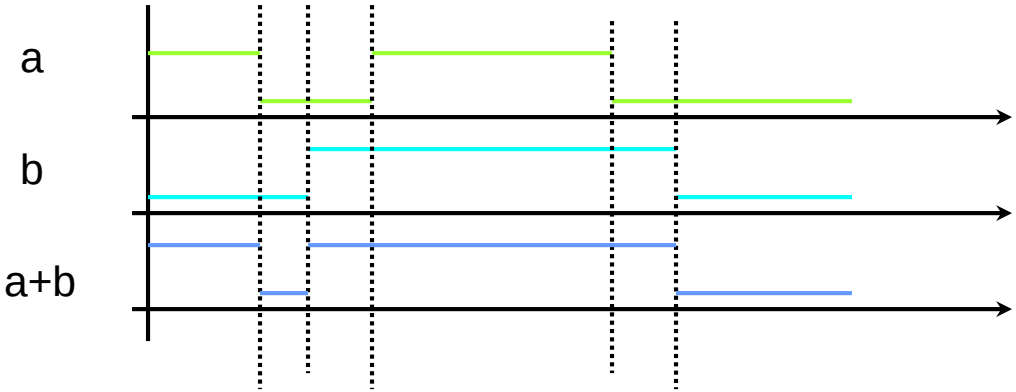

$$AB + CD + BD + AC + BC$$

# Représentation des fonctions

- Diagramme de Venn ou d'Euler
  vue ensembliste



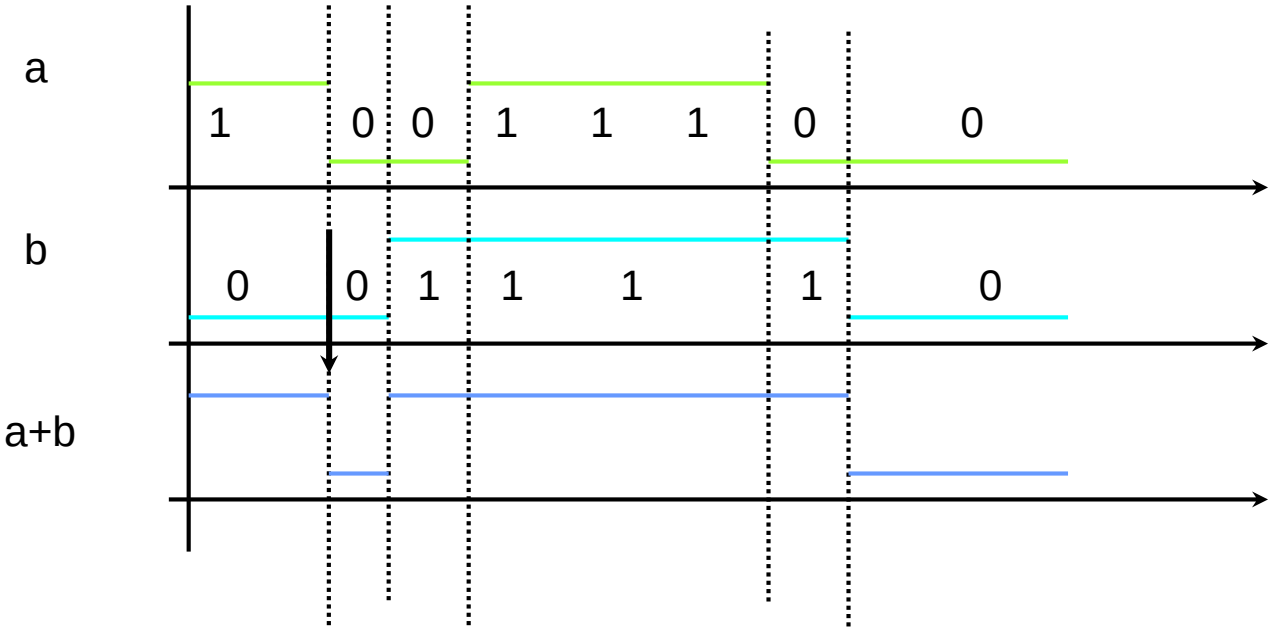
- Table de vérité
- Tableau de Karnaugh
- Équation logique ex: **f(a,b)=a+b**
- Chronogramme : Graphe d'évolution temporelle



# Chronogrammes

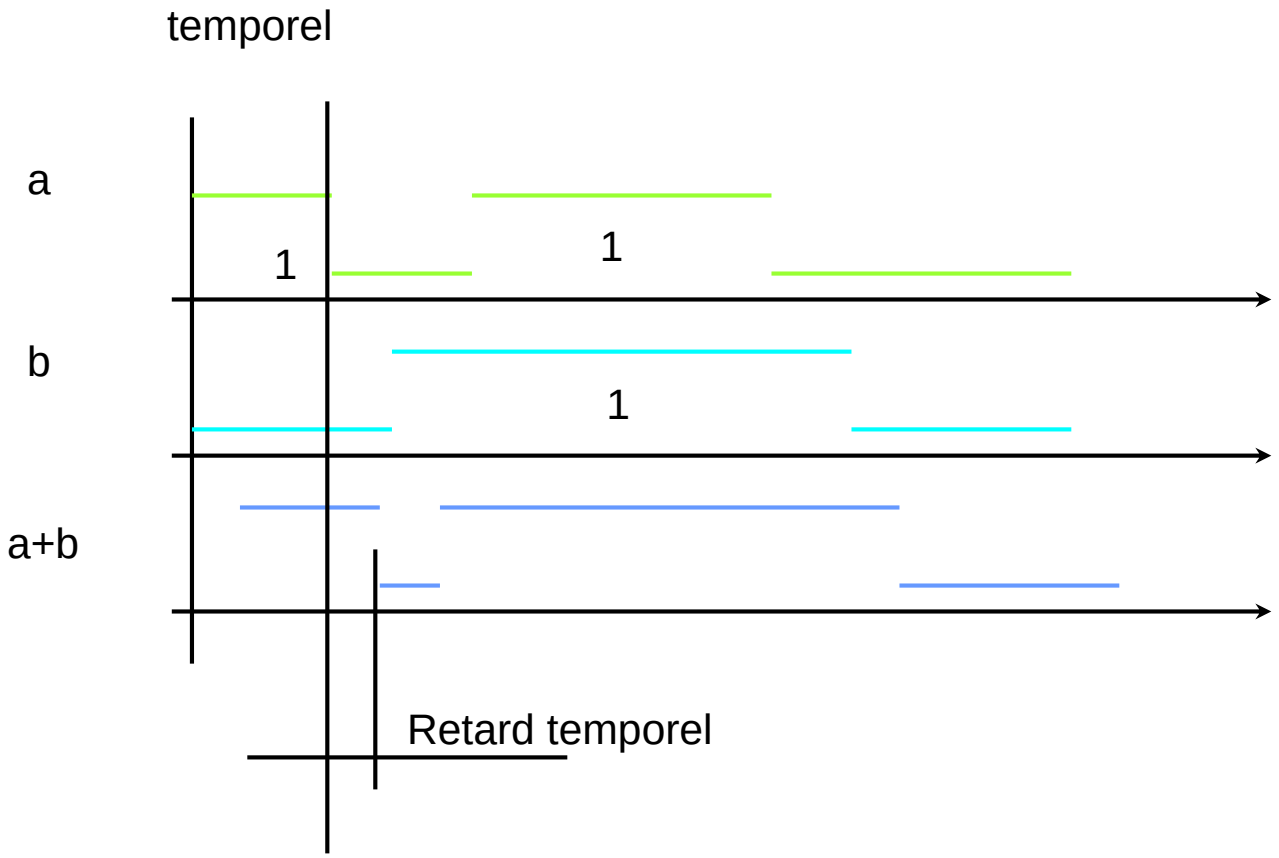
Plusieurs niveaux d'abstraction :

fonctionnel,



# Chronogrammes

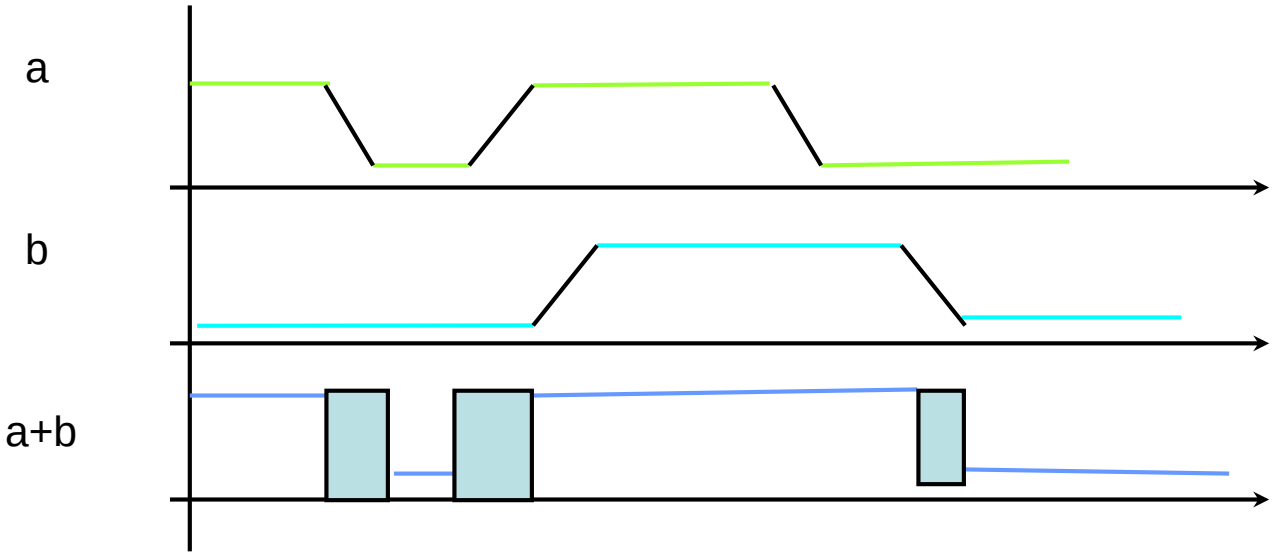
Plusieurs niveaux d'abstraction :



# Chronogrammes

Plusieurs niveaux d'abstraction :

analogique symbolique



# Réalisation en électronique

0/1 représentés par des tensions, courants, charges, fréquences, ....

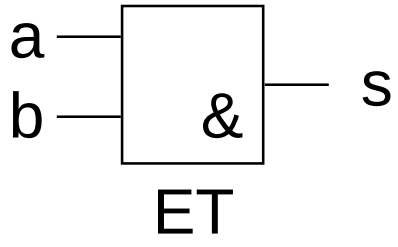
Classiquement TENSIONS : Niveau haut = H (le plus positif)
Niveau bas = L (B) (le plus négatif)

Association d'une information binaire à un niveau :

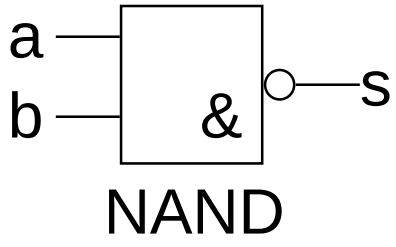
| | |
|---|---|
| Convention positive | H ⟶ 1 |
| (ou logique positive) | L ⟶ 0 |
| Convention négative | H ⟶ 0 |
| (ou logique négative) | L ⟶ 1 |

# Représentation graphique : Norme française

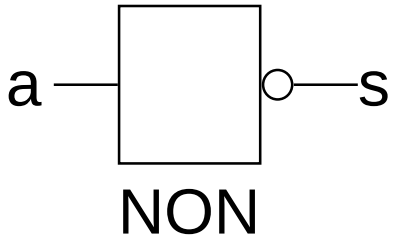
**Les portes logiques:**



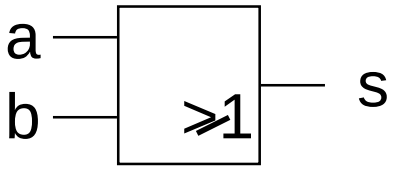
ET



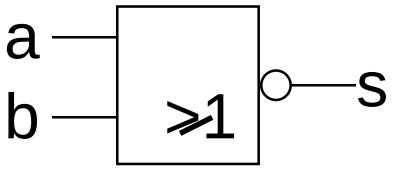
NAND



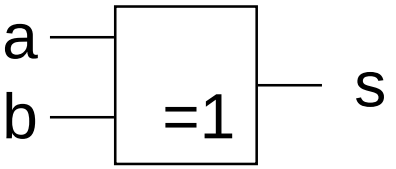
NON



OU



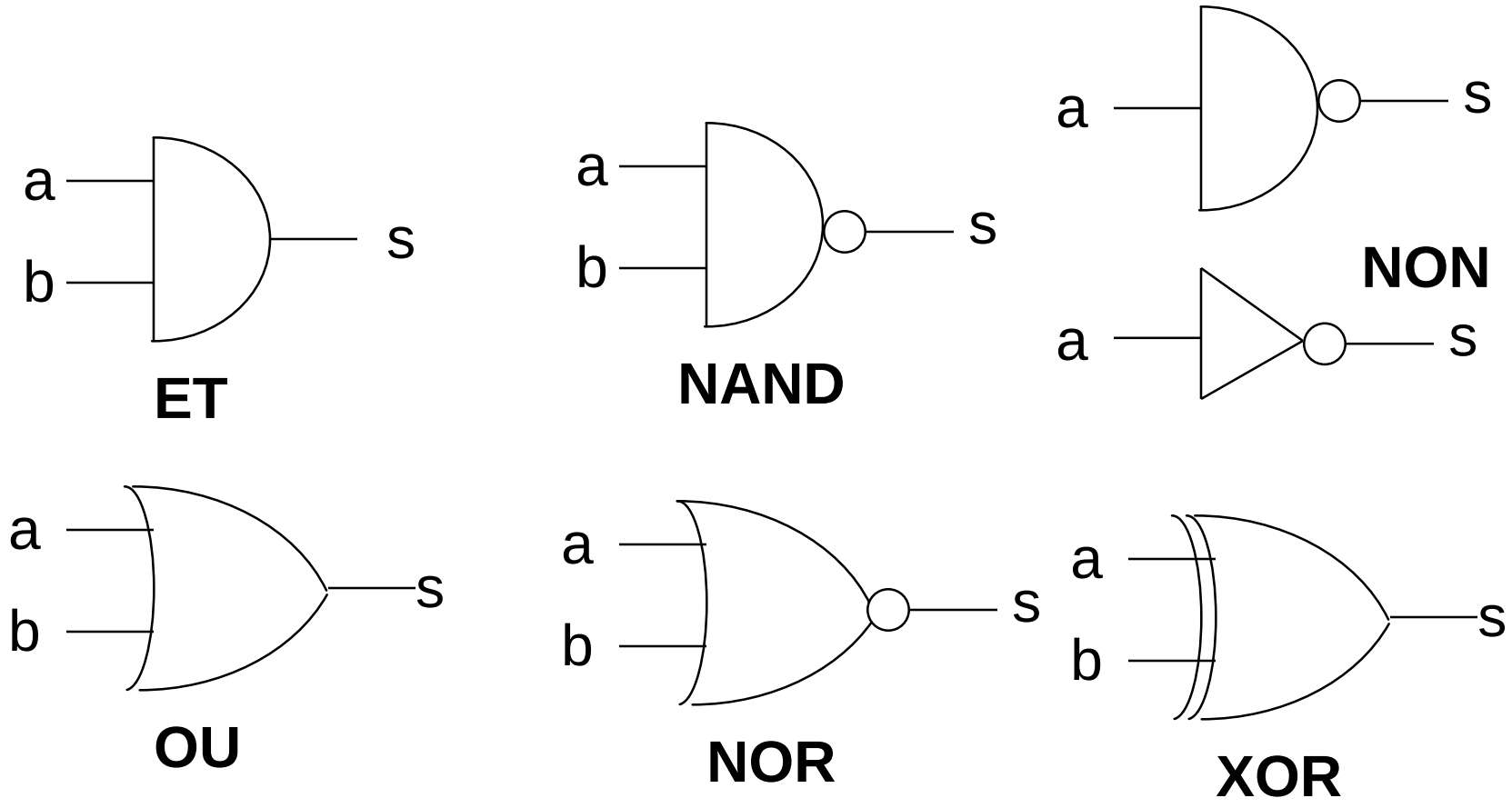
NOR



XOR

# Représentation graphique : Norme américaine

**Les portes logiques:**



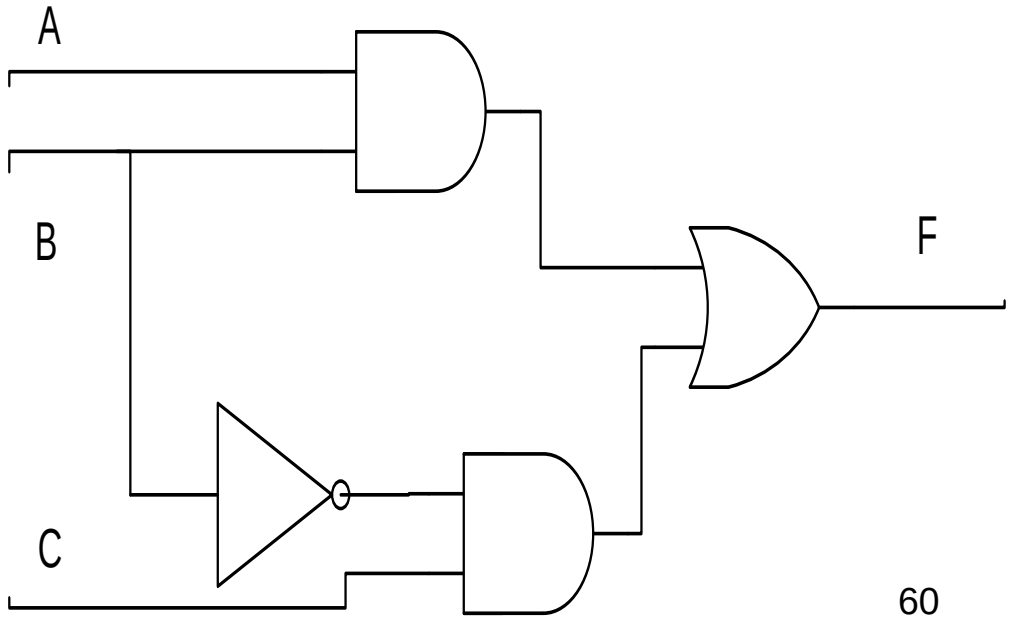
# Schéma d'un circuit logique ( Logigramme)

•Un logigramme est la traduction de la fonction logique en un schéma électronique.

•Le principe consiste à remplacer chaque opérateur logique par la porte logique qui lui correspond.

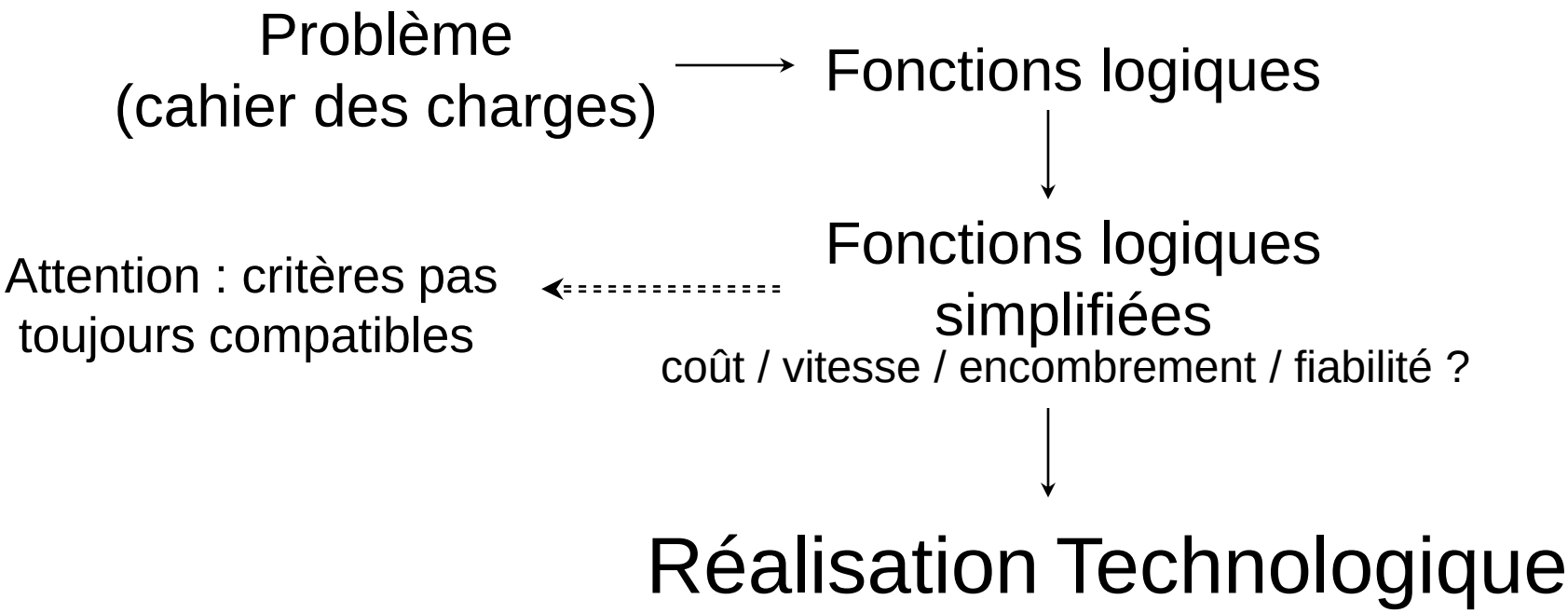
**Exemple 1**

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



# Les circuits combinatoires

# Moyens physiques de réalisation des fonctions logiques



# Les circuits combinatoires

**Objectifs**

- Apprendre la structure de quelques circuits combinatoires souvent utilisés ( multiplexeur, codeur et decodeur, demi additionneur , additionneur complet,…….).

- Apprendre comment utiliser des circuits combinatoires pour concevoir d'autres circuits plus complexes.

# Circuits combinatoires

- Un circuit combinatoire est un circuit numérique dont les sorties dépendent uniquement des entrées.
- $S_i=F(E_i)$
- $S_i=F(E_1,E_2,\ldots,E_n)$

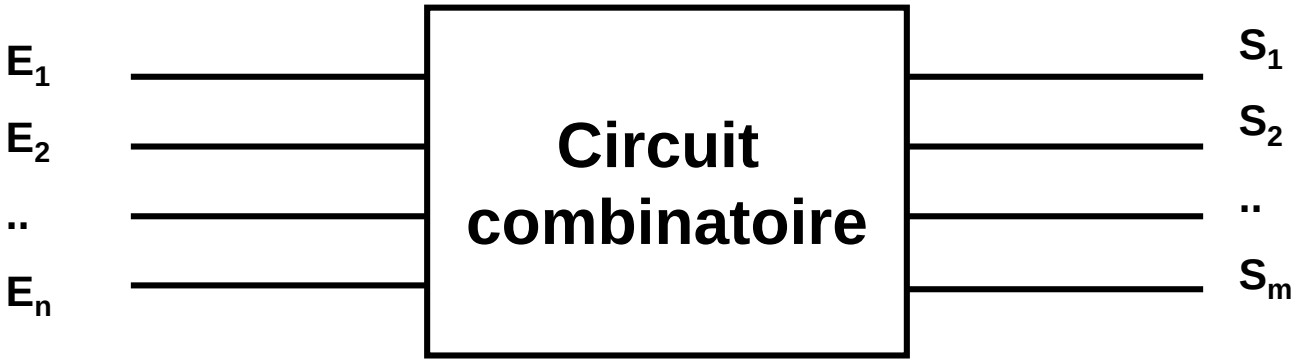



**Schéma Bloc**

- C'est possible d'utiliser des circuits combinatoires pour réaliser d'autres circuits plus complexes.

# Composants combinatoires

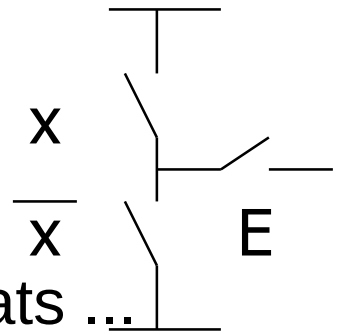
- Inverseurs
- Multiplexeur / démultiplexeur
- Codeurs / Décodeurs
- Transcodeurs
- Additionneur, comparateurs
- Unité arithmétique et logique UAL

# Portes intégrées

Options technologiques : familles logiques
(TTL,CMOS, BiCMOS, ECL ..)

Entrées : classique, triggée

Sorties : collecteur (drain) ouvert, sortie 3 états ...




**Remarque 1 :**

10 entrées = $2^{10}$ fonctions possibles

⟶ Choix des meilleures fonctions

# Portes intégrées

**Remarque 2:**

Problème du nombre de boîtiers pour réaliser une fonction logique ⟶ INTEGRATION

SSI (*small scale integration*) petite : inférieur à 12 portes

MSI (*medium*) moyenne : 12 à 99

LSI (*large*) grande : 100 à 9999

VLSI (*very large*) très grande : 10 000 à 99 999

ULSI (*ultra large*) ultra grande : 100 000 et plus

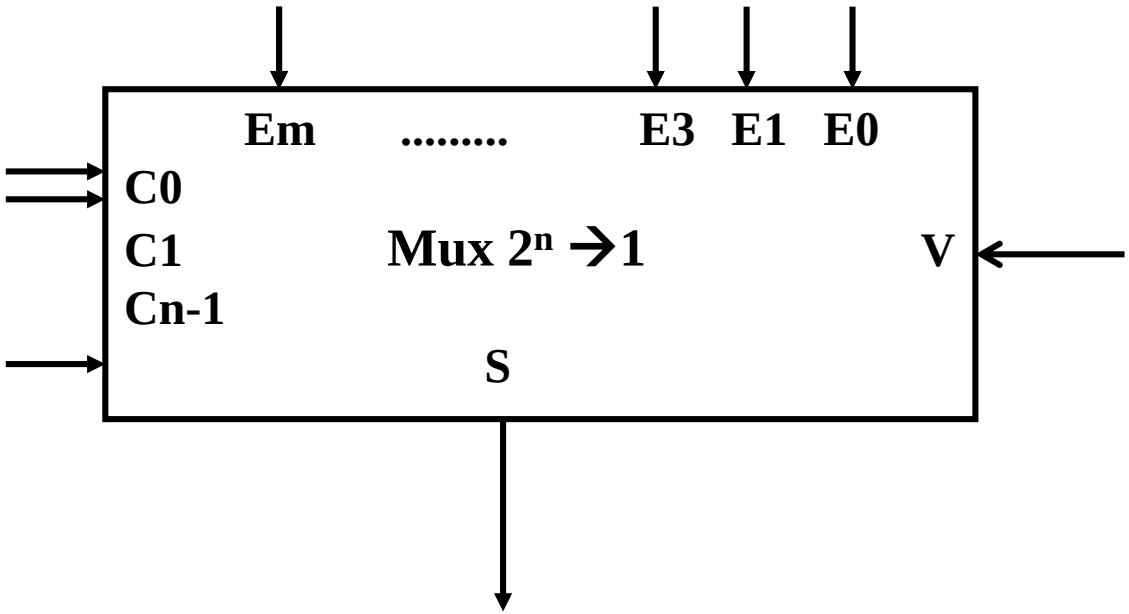
**Remarque 3:**

Une manière d'augmenter la puissance de traitement est de construire des CI dédiés à une application

(**ASIC** pour *Application Specific Integrated Circuit*)

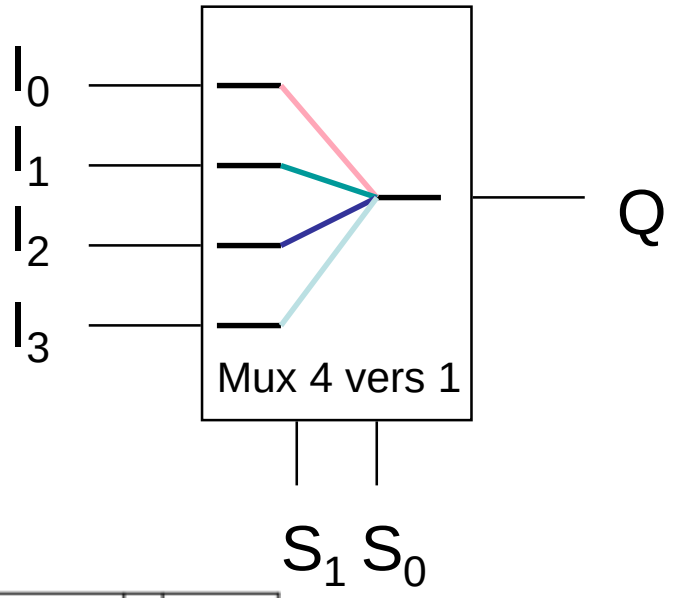
# Multiplexeur

- Un multiplexeur est un circuit combinatoire qui permet de sélectionner une information (1 bit) parmi $2^n$ valeurs en entrée.
- Il possède :
  - $2^n$ entrées d'information
  - Une seule sortie
  - N entrées de sélection ( commandes)



# Multiplexeur 4 → 1

***Sélection d'une voie parmi $2^N$ par N bits de commande***




Si $(S_1S_0)_2 = (0)_{10}$ alors $Q = I_0$

$$Q = \overline{S_1}.\overline{S_0}.I_O$$

Si $(S_1S_0)_2 = (1)_{10}$ alors $Q = I_1$

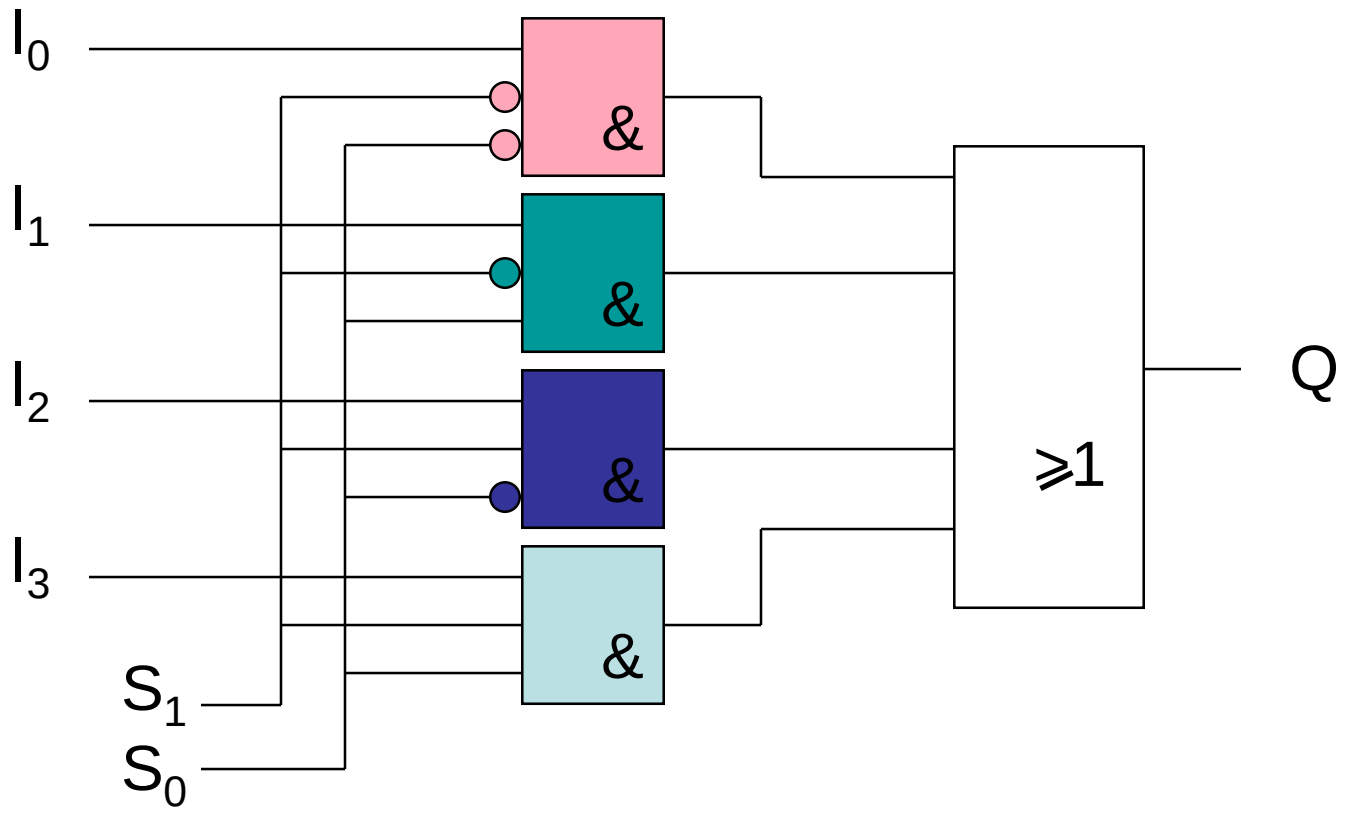
$$Q = \overline{S_1}.S_0.I_1$$

| S1 | S0 | | Q |
|---|---|---|---|
| 0 | 0 | | $I_0$ |
| 0 | 1 | | $I_1$ |
| 1 | 0 | | $I_2$ |
| 1 | 1 | | $I_3$ |

$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

# Multiplexeur (logigramme)

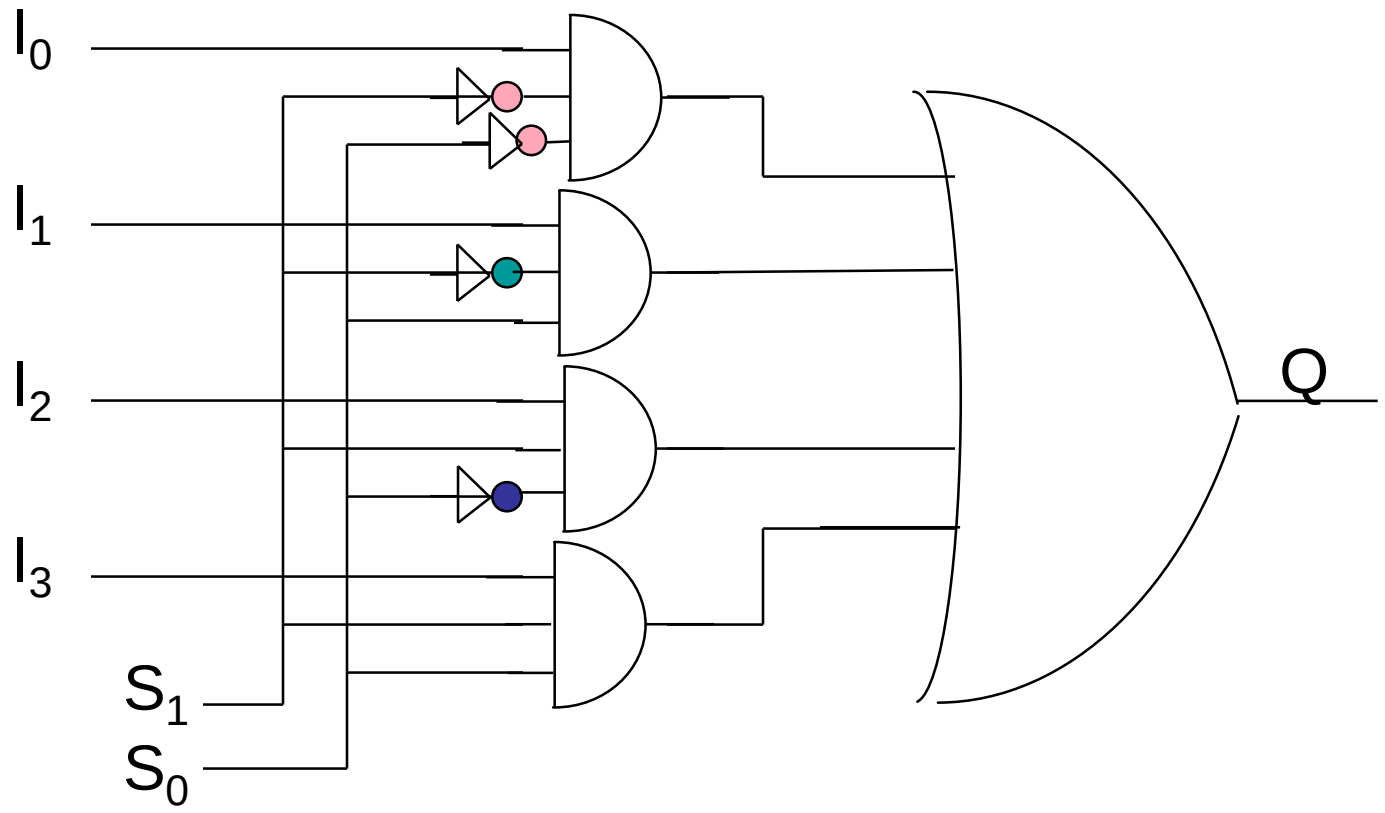
$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$




**Applications :** La conversion parallèle / série d'informations

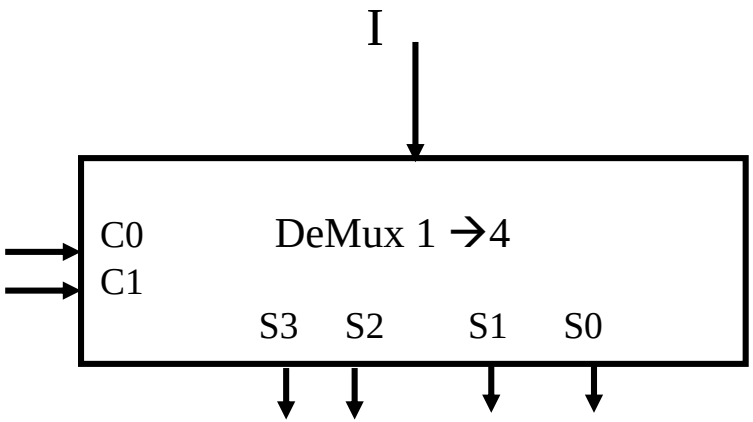
# Multiplexeur (logigramme)

$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

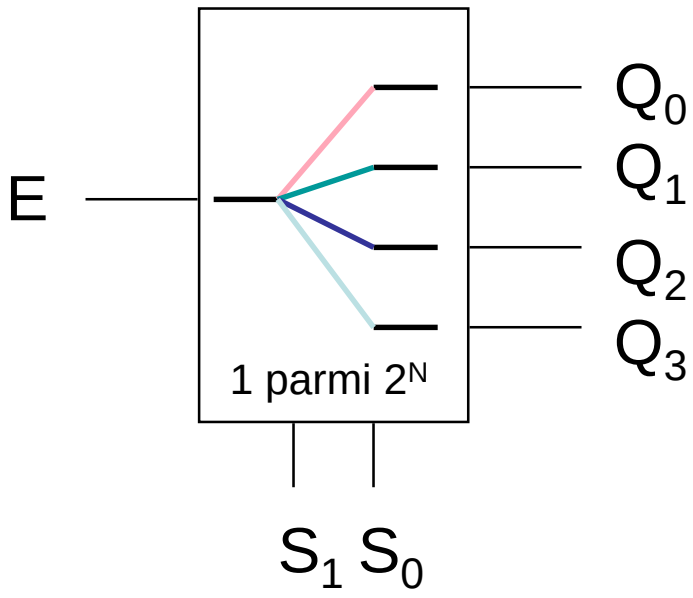


# Démultiplexeur

- Il joue le rôle inverse d'un multiplexeurs, il permet de faire passer une information dans l'une des sorties selon les valeurs des entrées de commandes.
- Il possède :
  - une seule entrée
  - $2^n$ sorties
  - N entrées de sélection ( commandes)



# Démultiplexeur : 1 parmi $2^n$

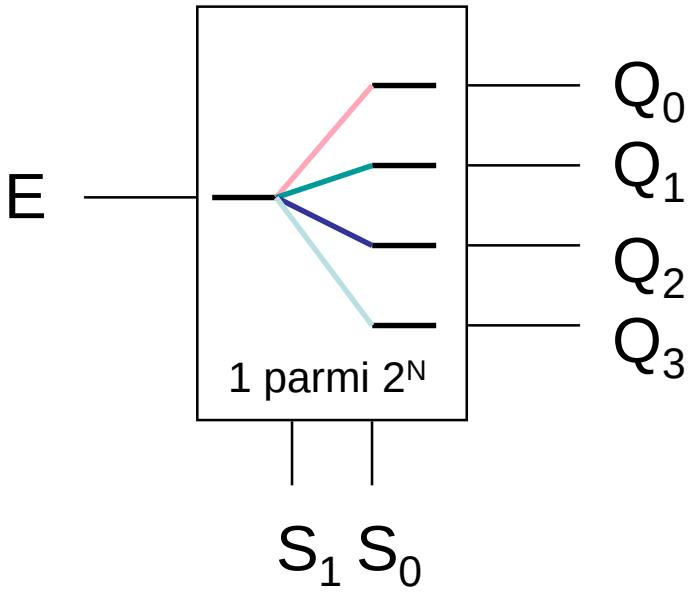



$Q_0$ = E si $(S_1S_0)_2$=0
0 sinon

$Q_1$ = E si $(S_1S_0)_2$=1
0 sinon

Remarque : E peut ne pas être «disponible»
Sortie sélectionnée = 1 les autres 0
ou Sortie sélectionnée = 0 les autres 1

# Démultiplexeur : 1 →4




$$Q0 = \overline{S1}.\overline{S0}.(E)$$

$$Q1 = \overline{S1}.S0.(E)$$

$$Q2 = S1.\overline{S0}.(E)$$

$$Q3 = S1.S0.(E)$$

| $S_1$ | $S_0$ | Q3 | Q2 | Q1 | Q0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | E |
| 0 | 1 | 0 | 0 | E | 0 |
| 1 | 0 | 0 | E | 0 | 0 |
| 1 | 1 | E | 0 | 0 | 0 |

**Table de vérité**

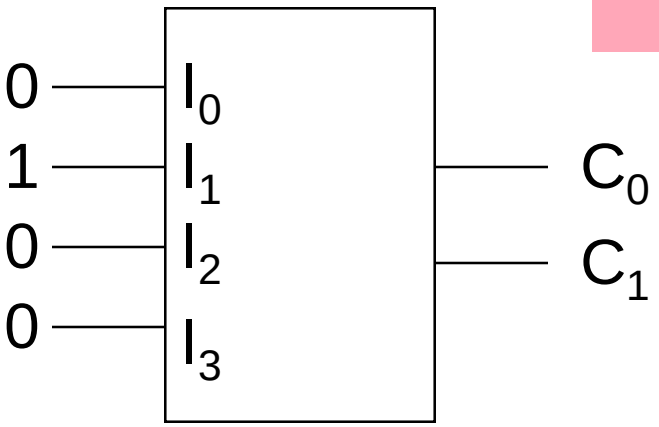
# Codeur (ou Encodeur)

***Faire correspondre un mot code à un symbole***

**$2^N$ entrées**

**N sorties**

1 entrée parmi $2^N$




Le code de l'entrée

**Mot**

**Code**

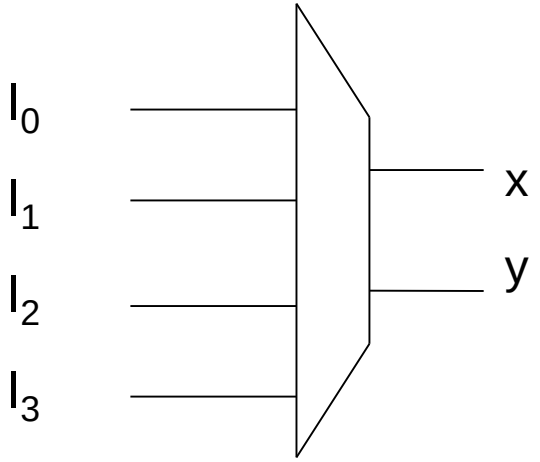
Traduit le rang de l'entrée active en un code binaire

Exemple : Clavier / Scan code
Caractère / Code ASCII

# L'encodeur binaire ( 4→2)

## Table de vérité

| $I_0$ | $I_1$ | $I_2$ | $I_3$ | | x | y |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 0 | 0 |
| 1 | x | x | x | | 0 | 0 |
| 0 | 1 | x | x | | 0 | 1 |
| 0 | 0 | 1 | x | | 1 | 0 |
| 0 | 0 | 0 | 1 | | 1 | 1 |



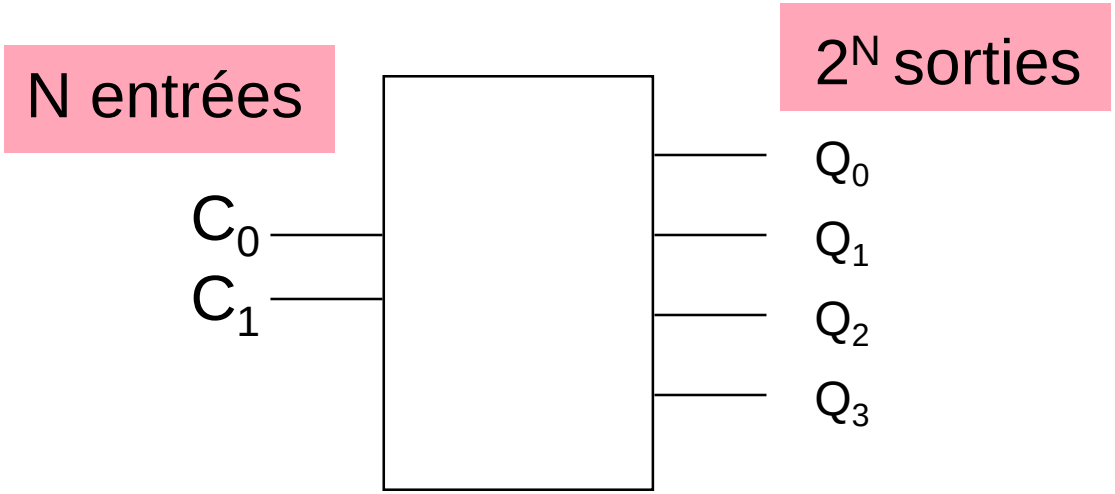

## Equations

$$X = \overline{I0}.\overline{I1}.(I2 + I3)$$

$$Y = \overline{I0}.(I1 + .\overline{I2}.I3)$$

# Le décodeur binaire

- C'est un circuit combinatoire qui est constitué de :
  - N : entrées de données
  - $2^n$ sorties
  - Pour chaque combinaison en entrée une seule sortie est active à la fois

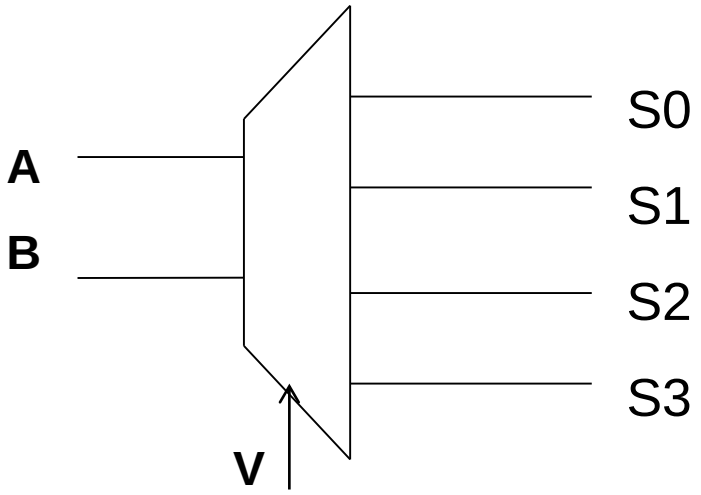



Active la ligne de sortie correspondant au code binaire présent en entrée

# Décodeur 2→4

**Table de vérité**

| V | A | B | | S0 | S1 | S2 | S3 |
|---|---|---|---|---|---|---|---|
| 0 | X | X | | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | | 0 | 0 | 0 | 1 |




$$S_0 = (\overline{A}.\overline{B}).V$$

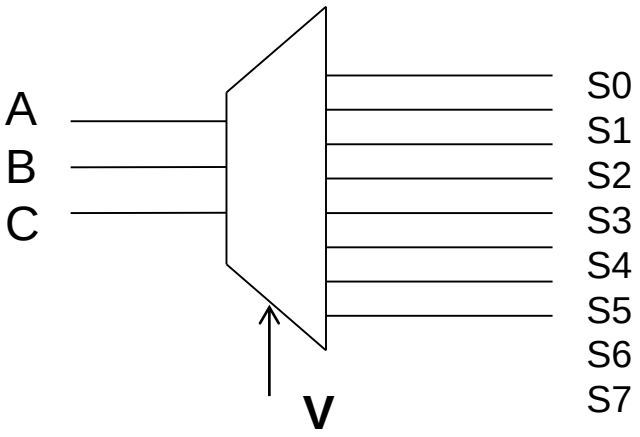
$$S_1 = (\overline{A}.B).V$$

$$S_2 = (A.\overline{B}).V$$

$$S_3 = (A.B).V$$

# Décodeur 3→8

| A | B | C | | S0 | S1 | S2 | S3 | S4 | S5 | S6 | S7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |




$$S_0 = \overline{A}.\overline{B}.\overline{C}$$
$$S_1 = \overline{A}.\overline{B}.C$$
$$S_2 = \overline{A}.B.\overline{C}$$
$$S_3 = \overline{A}.B.C$$
$$S_4 = A.\overline{B}.\overline{C}$$
$$S_5 = A.\overline{B}.C$$
$$S_6 = A.B.\overline{C}$$
$$S_7 = A.B.C$$

Remarque :

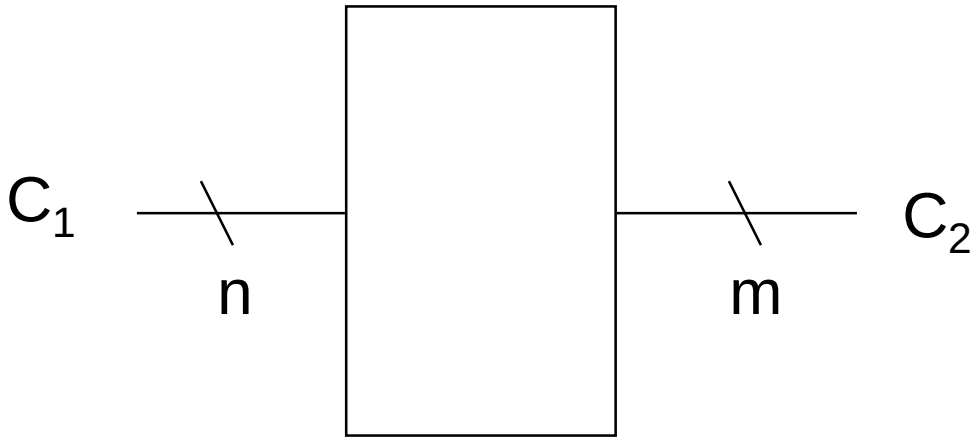
Multiplexeur ⟺ Démultiplexeur

Codeur ⟺ Décodeur

# Transcodeur

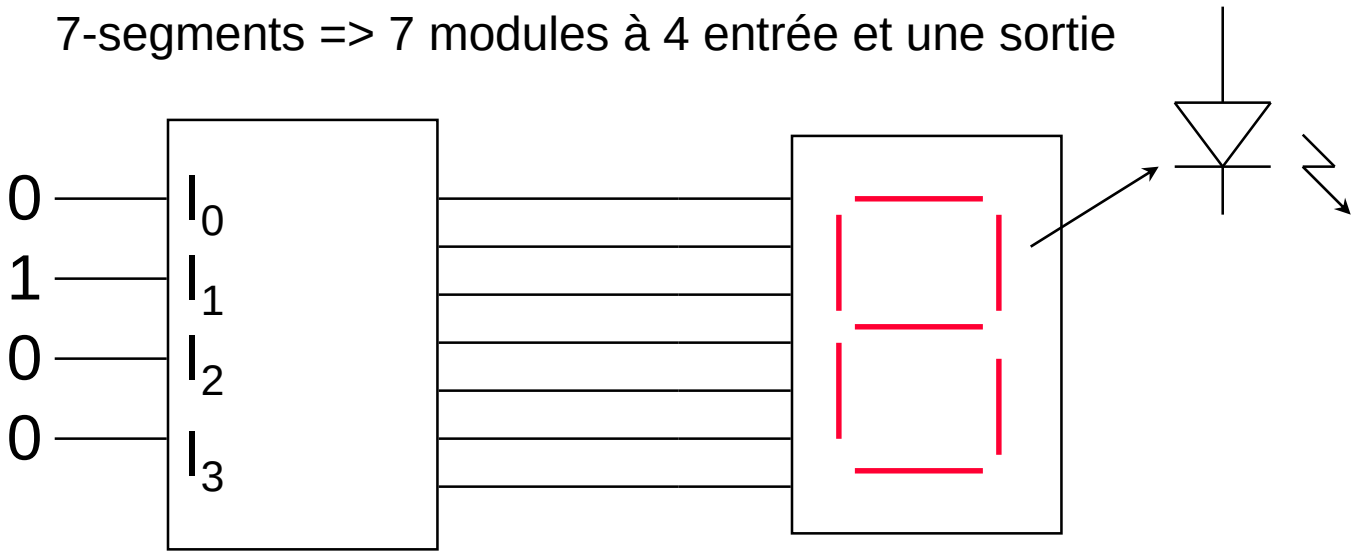
C’est un circuit combinatoire qui permet de transformer un code X ( sur n bits) en entrée en un code Y ( sur m bits) en sortie.

Passage d’un code $C_1$ à un code $C_2$



# Transcodeur : exemple

7-segments => 7 modules à 4 entrée et une sortie




Code binaire 0 à 9

Configuration alimentation des diodes (ou LCD)

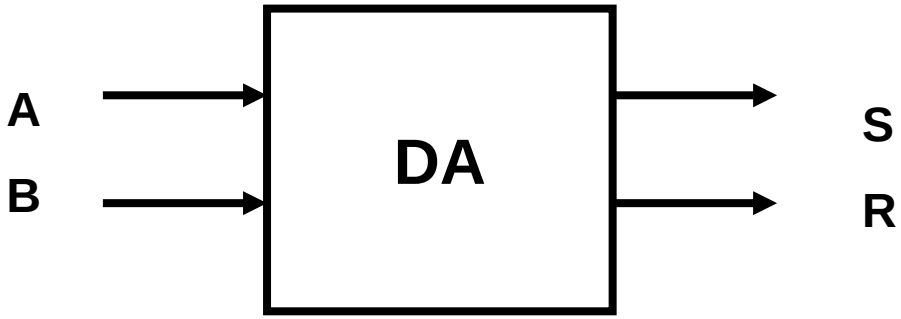
Exemples de code :

Binaire, binaire réfléchi, 7-segments, BCD, …

# Additionneur

- Le demi additionneur est un circuit combinatoire qui permet de réaliser la somme arithmétique de deux nombres A et B chacun sur un bit.

- A la sotie on va avoir la somme S et la retenu R ( Carry).




Pour trouver la structure ( le schéma ) de ce circuit on doit en premier dresser sa table de vérité

# Demi Additionneur

- En binaire l'addition sur un seul bit se fait de la manière suivante:

$$\begin{cases} 0+0 = 00 \\ 0+1 = 01 \\ 1+0 = 01 \\ 1+1 = 10 \end{cases}$$

•La table de vérité associée :

| A | B | | R | S |
|---|---|---|---|---|
| 0 | 0 | | 0 | 0 |
| 0 | 1 | | 0 | 1 |
| 1 | 0 | | 0 | 1 |
| 1 | 1 | | 1 | 0 |

De la table de vérité on trouve :

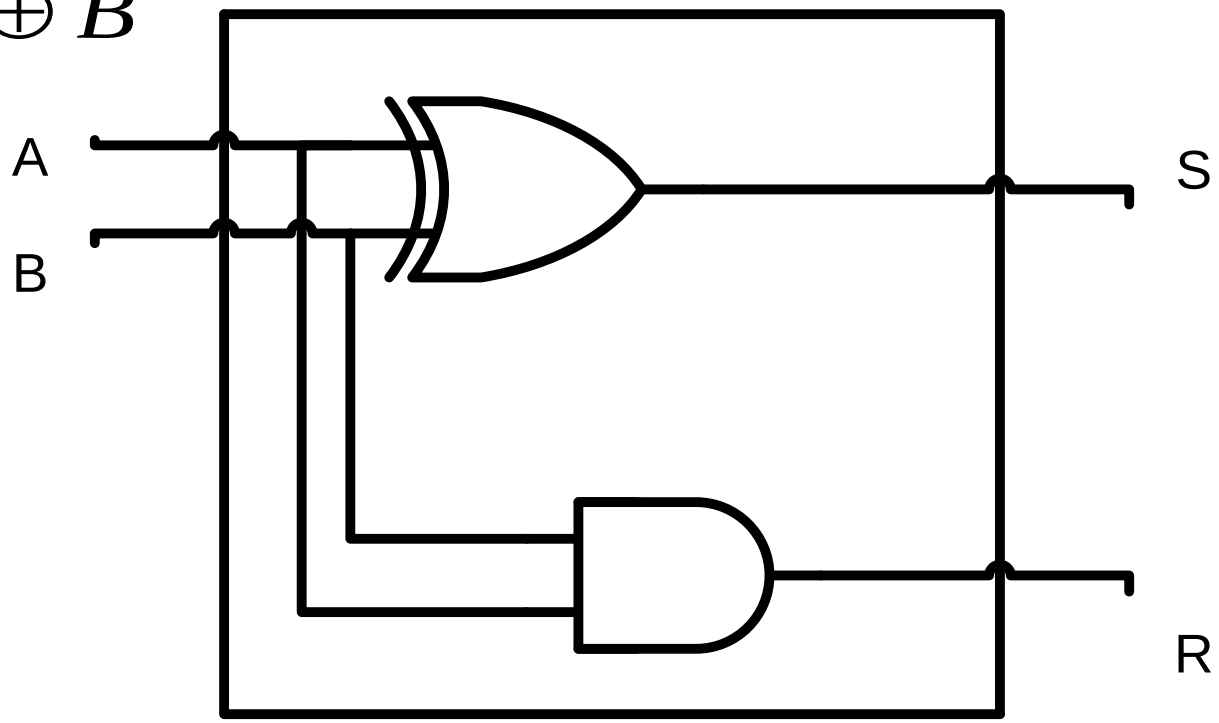
$$R = A.B$$

$$S = \overline{A}.B + A.\overline{B} = A \oplus B$$

# Demi Additionneur

$$R = A.B$$

$$S = A \oplus B$$




Logigramme Demi-Additionneur

# Additionneur complet

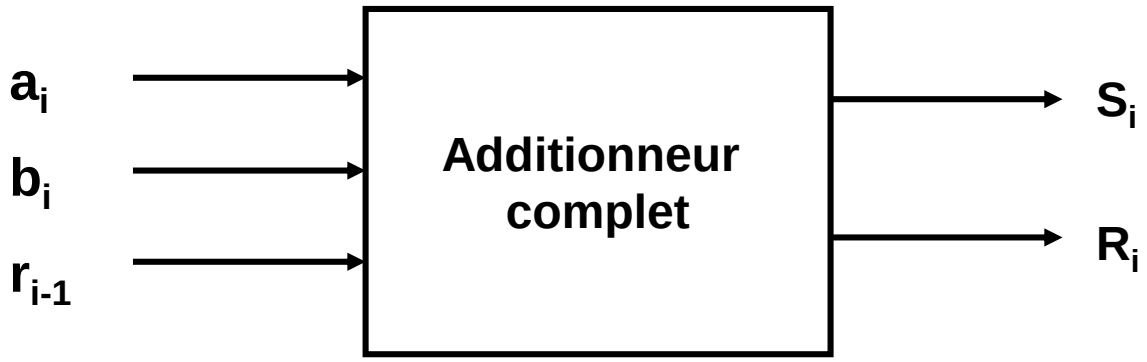
- Lorsque on fait une addition (binaire) il faut tenir en compte de la retenue entrante.

| | | | | |
|---|---|---|---|---|
| $r_4$ | $r_3$ | $r_2$ | $r_1$ | $r_0 = 0$ |
| | $a_4$ | $a_3$ | $a_2$ | $a_1$ |
| + | $b_4$ | $b_3$ | $b_2$ | $b_1$ |
| $r_4$ | $s_4$ | $s_3$ | $s_2$ | $s_1$ |

| | | |
|---|---|---|
| | $r_{i-1}$ | |
| | $a_i$ | |
| + | $b_i$ | |
| $r_i$ | $s_i$ | |

# Additionneur complet 1 bit

- L'additionneur complet à un bit possède 3 entrées :
  - $a_i$ : le premier nombre sur un bit.
  - $b_i$ : le deuxième nombre sur un bit.
  - $r_{i-1}$ : le retenue entrante sur un bit.
- Il possède deux sorties :
  - $S_i$ : la somme
  - $R_i$ la retenue sortante



# Additionneur complet 1 bit

**Table de vérité d'un additionneur complet sur 1 bit**

| $a_i$ | $b_i$ | $r_{i-1}$ | | $r_i$ | $s_i$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | | 0 | 0 |
| 0 | 0 | 1 | | 0 | 1 |
| 0 | 1 | 0 | | 0 | 1 |
| 0 | 1 | 1 | | 1 | 0 |
| 1 | 0 | 0 | | 0 | 1 |
| 1 | 0 | 1 | | 1 | 0 |
| 1 | 1 | 0 | | 1 | 0 |
| 1 | 1 | 1 | | 1 | 1 |

Equations

$$S_i = \overline{A}_i.\overline{B}_i.R_{i-1} + \overline{A}_i.B_i.\overline{R}_{i-1} + A_i.\overline{B}_i.\overline{R}_{i-1} + A_i.B_i.R_{i-1}$$

$$R_i = \overline{A}_i B_i R_{i-1} + A_i \overline{B}_i R_{i-1} + A_i B_i \overline{R}_{i-1} + A_i B_i R_{i-1}$$

# Additionneur complet 1 bit

Si on veut simplifier les équations on obtient :

$$S_i = \overline{A_i}.\overline{B_i}.R_{i-1} + \overline{A_i}.B_i.\overline{R_{i-1}} + A_i.\overline{B_i}.\overline{R_{i-1}} + A_i.B_i.R_{i-1}$$

$$S_i = \overline{A_i}.(\overline{B_i}.R_{i-1} + B_i.\overline{R_{i-1}}) + A_i.(\overline{B_i}.\overline{R_{i-1}} + B_i.R_{i-1})$$

$$S_i = \overline{A_i}(B_i \oplus R_{i-1}) + A_i.(\overline{B_i \oplus R_{i-1}})$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

$$R_i = \overline{A_i}B_iR_{i-1} + A_i\overline{B_i}R_{i-1} + A_iB_i\overline{R_{i-1}} + A_iB_iR_{i-1}$$

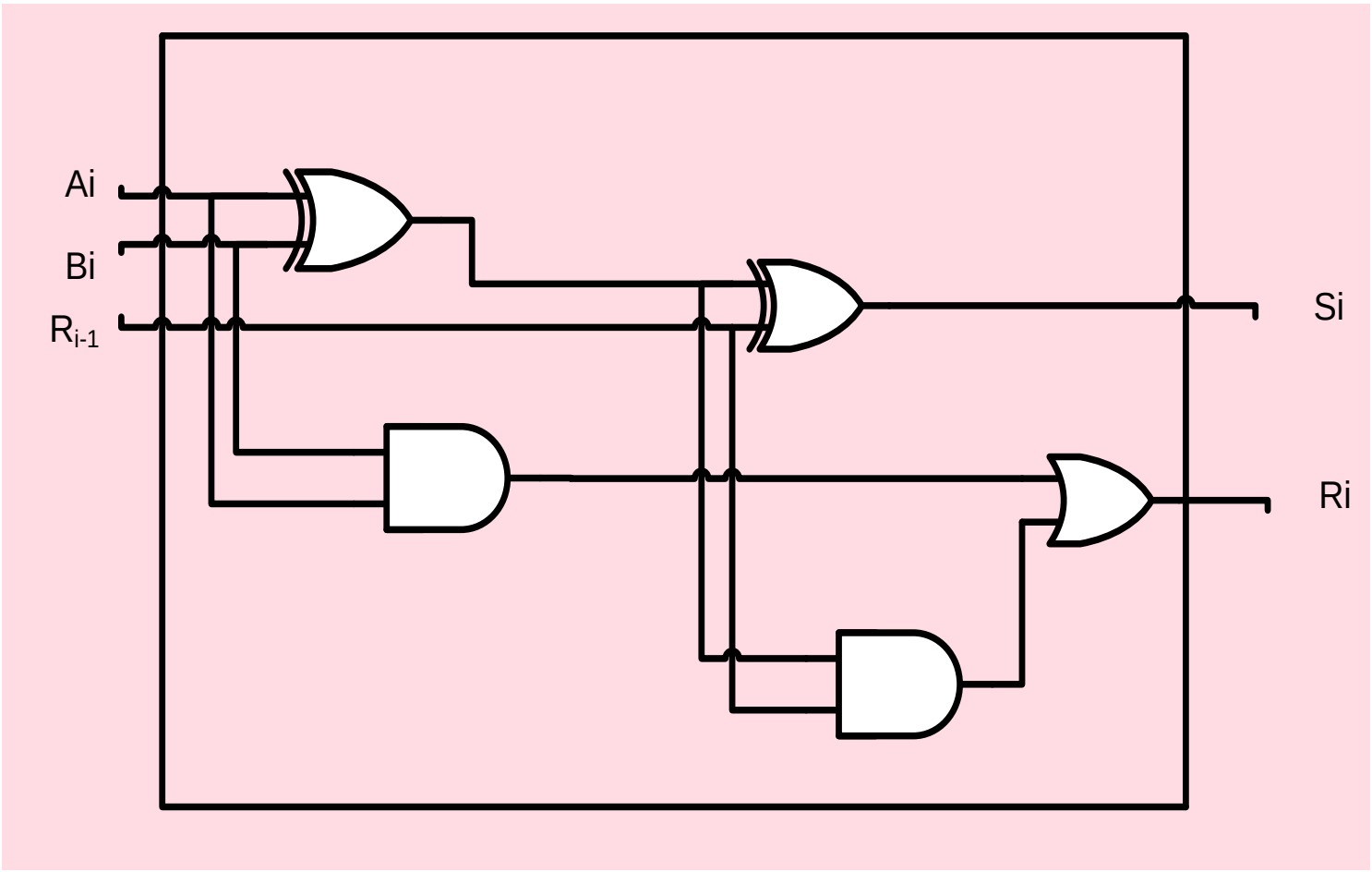
$$R_i = R_{i-1}.(\overline{A_i}.B_i + A_i.\overline{B_i}) + A_iB_i(\overline{R_{i-1}} +_i R_{i-1})$$

$$R_i = R_{i-1}.(A_i \oplus B_i) + A_iB_i$$

# Schéma d'un additionneur complet

$$R_i = A_i.B_i + R_{i-1}.(B_i \oplus A_i)$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$



# Additionneur sur 4 bits

- Un additionneur sur 4 bits est un circuit qui permet de faire l'addition de deux nombres A et B de 4 bits chacun
  - $A(a_3a_2a_1a_0)$
  - $B(b_3b_2b_1b_0)$

  ⇨ En plus il prend en compte de la retenu entrante

- En sortie on va avoir le résultat sur **4 bits ainsi que la retenu ( 5 bits en sortie )**

- Donc au total le circuit possède 9 entrées et 5 sorties.

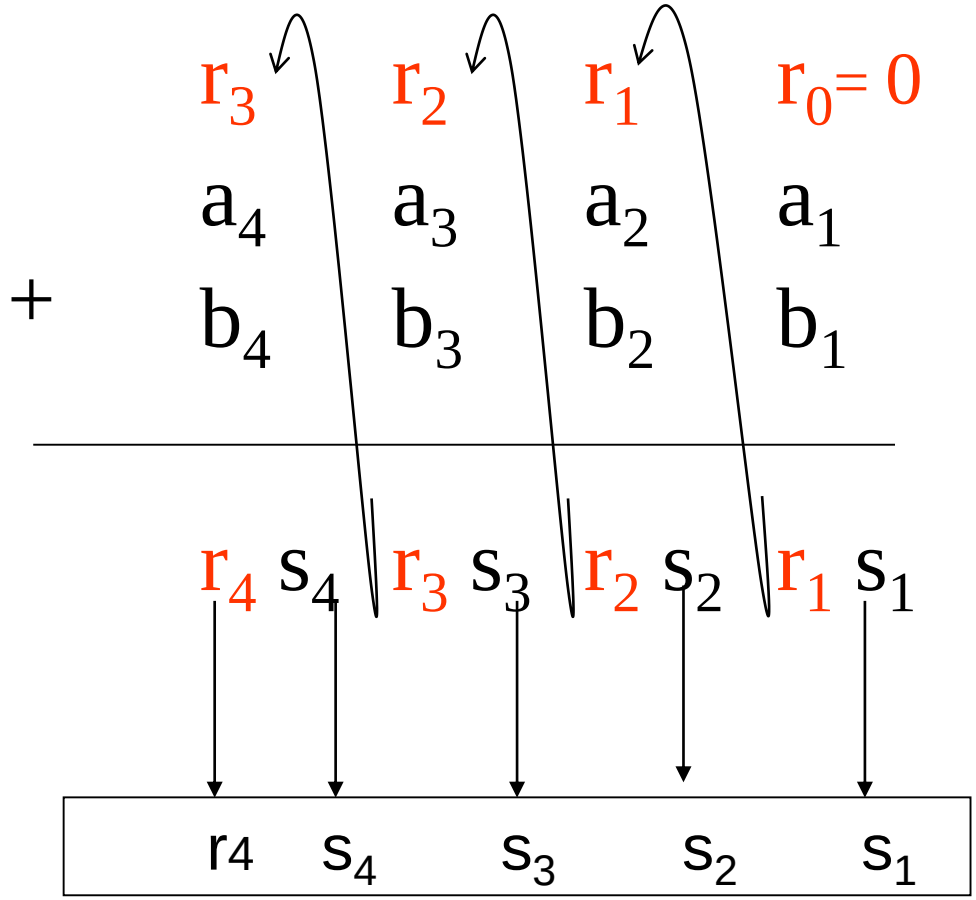
- Avec 9 entrées on a $2^9$=**512 combinaisons** !!!!!! Comment faire pour représenter la table de vérité ?????

- Il faut trouver une solution plus facile et plus efficace pour concevoir ce circuit ?

# Additionneur sur 4 bits

•Lorsque on fait l'addition en binaire , on additionne bit par bit en commençant à partir du poids fiable et à chaque fois on propage la retenue sortante au bit du rang supérieur.

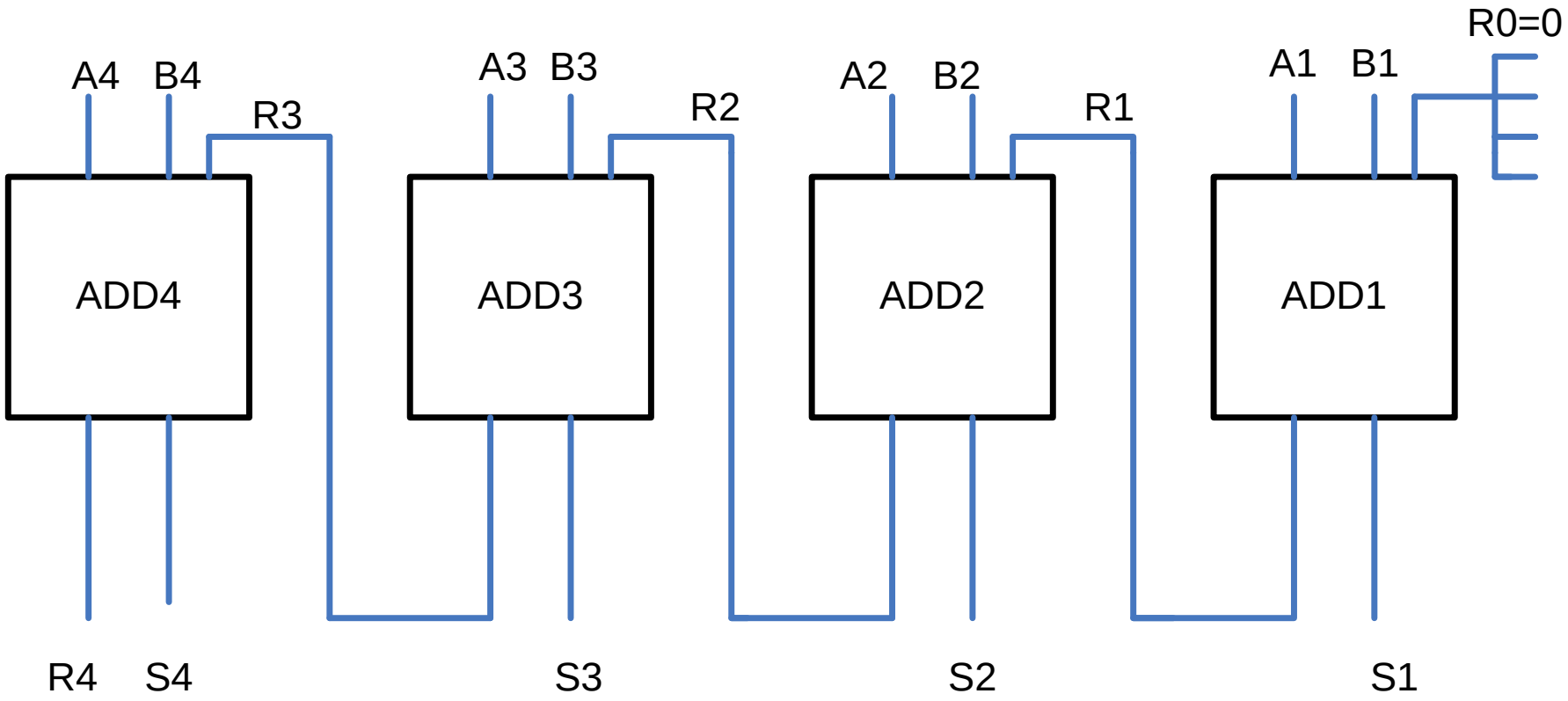
L'addition sur un bit peut se faire par un additionneur complet sur 1 bits.




**Résultat final**

# Additionneur 4 bits ( schéma )

Le premier mot $A(a_3a_2a_1a_0)$
Le deuxième mot $B(b_3b_2b_1b_0)$

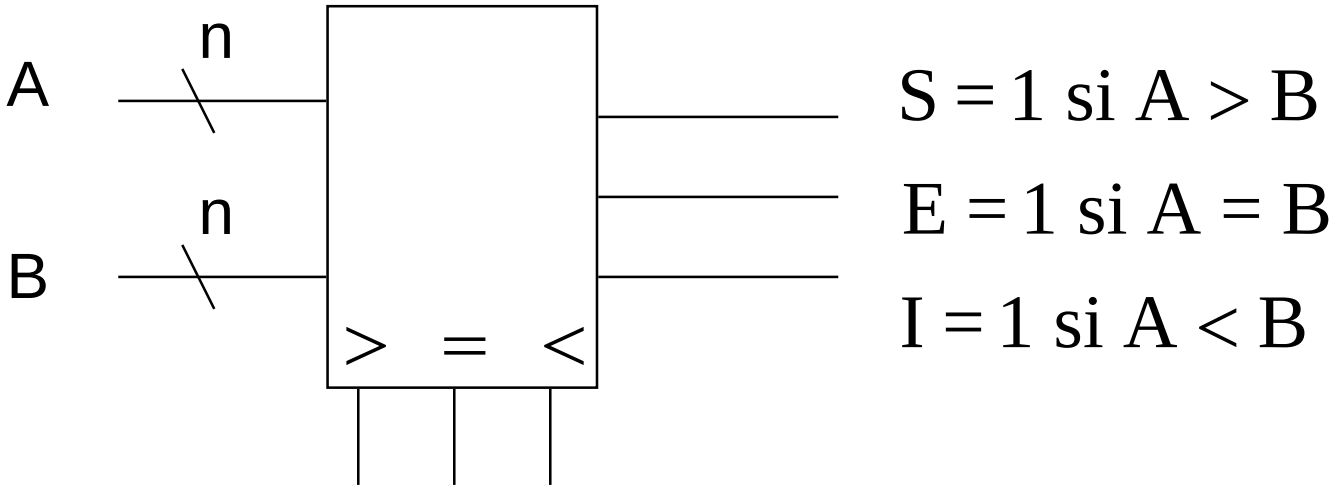


# Comparateur

- C'est un circuit combinatoire qui permet de comparer entre deux nombres binaire A et B.
- Il possède 2 entrées :
  - A : sur n bit
  - B : sur n bit

Il possède 3 sorties

E : égalité ( A=B)
I : inférieur ( A < B)
S : supérieur (A > B)




Entrées de cascadage

Pour une comparaison à n autres bits

# Comparateur sur un bit

Il possède 2 entrées :

A : sur un bit

B : sur un bit

Il possède 3 sorties

fe : égalité ( A=B)

fi : inférieur ( A < B)

fs : supérieur (A > B)

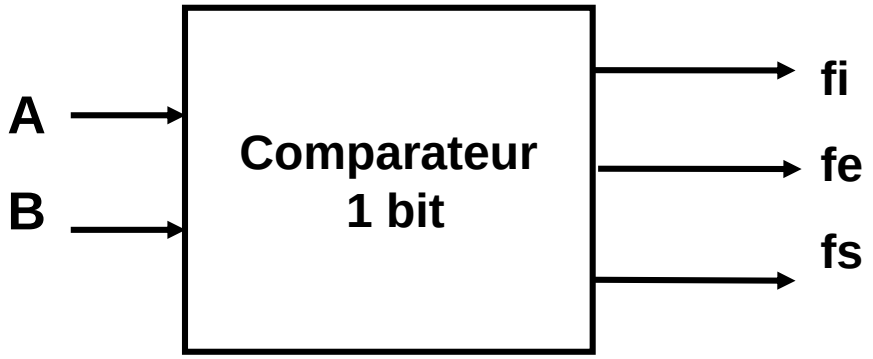



Table de vérité

| A | B | | fs | fe | fi |
|---|---|---|---|---|---|
| 0 | 0 | | 0 | 1 | 0 |
| 0 | 1 | | 0 | 0 | 1 |
| 1 | 0 | | 1 | 0 | 0 |
| 1 | 1 | | 0 | 1 | 0 |

$$fs = A.\overline{B}$$

$$fi = \overline{A}B$$

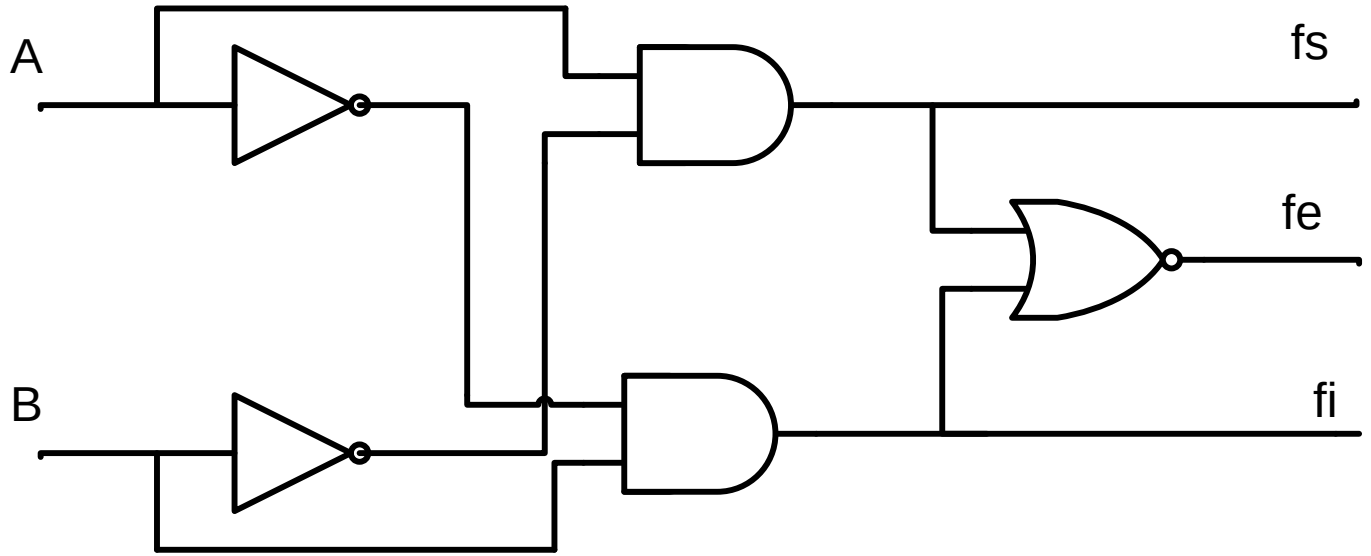
$$fe = \overline{A}\,\overline{B} + AB = \overline{A \oplus B} = \overline{fs + fi}$$

# Logigramme comparateur sur un bit

$$fs = A.\overline{B}$$

$$fi = \overline{A}B$$

$$fe = \overline{fs + fi}$$




**Logigramme comparateur sur 1 bit**

# Exemple 2 : Comparateur sur 2 bits

Il possède 2 entrées :

A : sur 2 bits ($A_2A_1$)

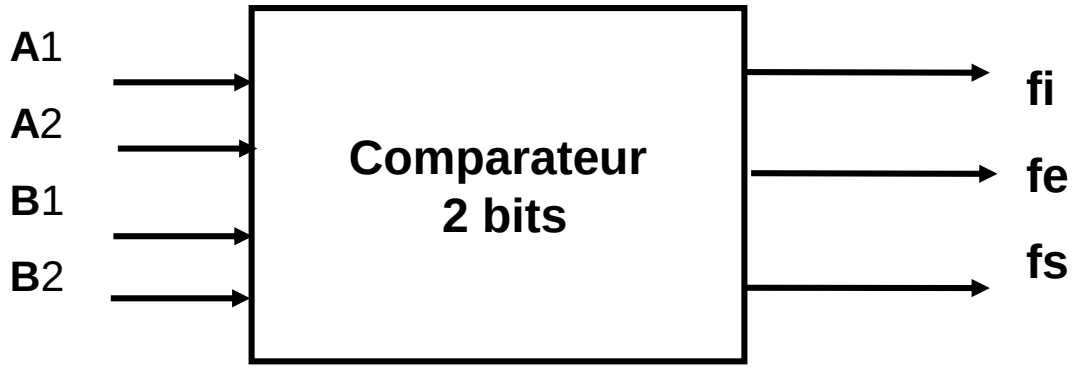
B : sur 2 bits ($B_2B_1$)

Il possède 3 sorties

fe : égalité

fi : inférieur

fs : supérieur



# Comparateur 2 bits

**A=B** si **A2=B2** et **A1=B1**

$$fe = \overline{(A2 \oplus B2)}.\overline{(A1 \oplus B1)}$$

**A>B si**

**A2 > B2 ou (A2=B2 et A1>B1)**

$$fs = A2.\overline{B2} + \overline{(A2 \oplus B2)}.(A1.\overline{B1})$$

**A<B si**

**A2 < B2 ou (A2=B2 et A1<B1)**

$$fi = \overline{A2}.B2 + \overline{(A2 \oplus B2)}.(\overline{A1}.B1)$$

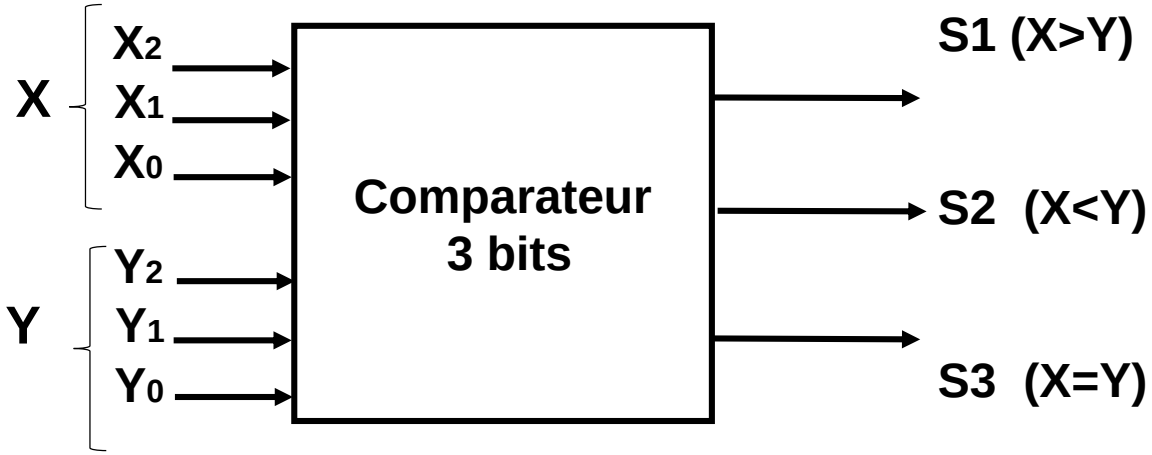
| A2 | A1 | B2 | B1 | | fs | fe | fi |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | | 0 | 1 | 0 |

# Comparateur 3 bits

- Un circuit combinatoire qui permet de comparer entre deux nombres binaire X et Y.

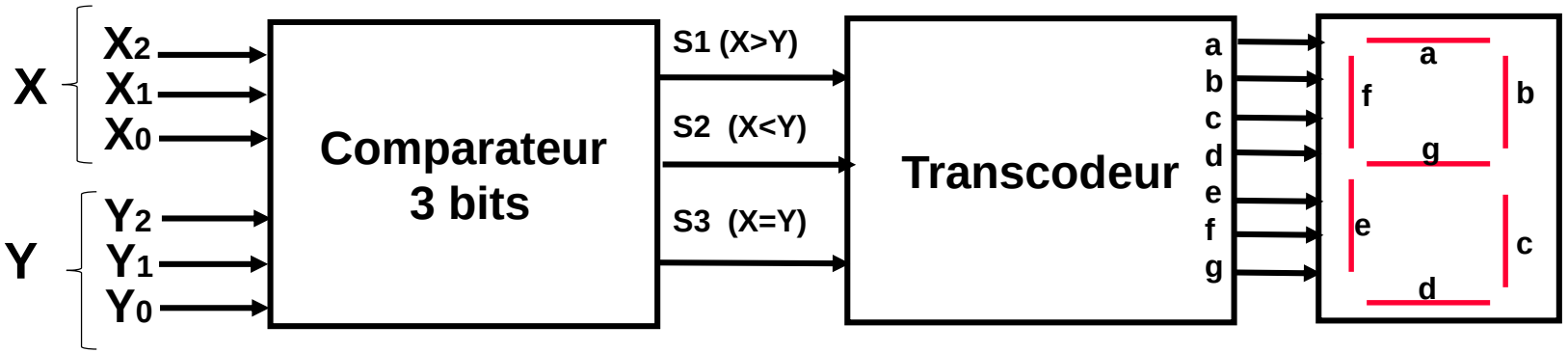
- Il possède 2 entrées :
  - A : sur 3 bits
  - B : sur 3 bits

Il possède 3 sorties
fe : égalité ( X=Y)
fi : inférieur ( X < Y)
fs : supérieur (X> Y)

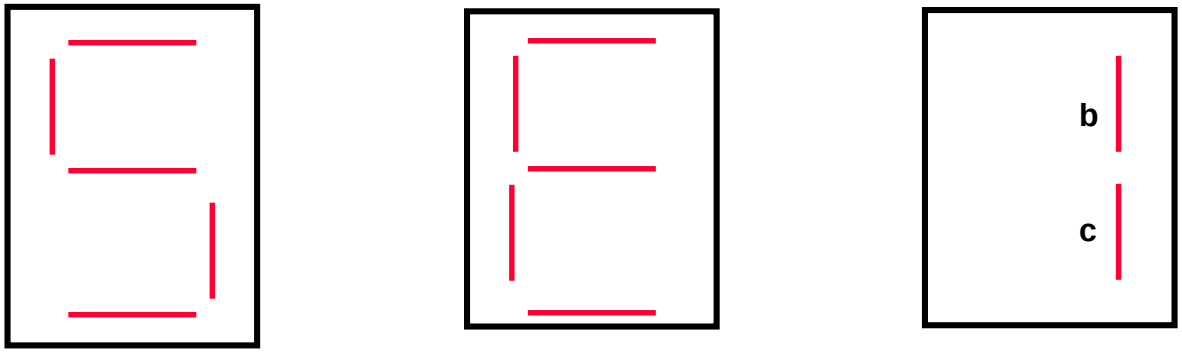


# Deux circuits combinatoires

Exemple: Circuit plus complexe = Comparateur + Transcodeur

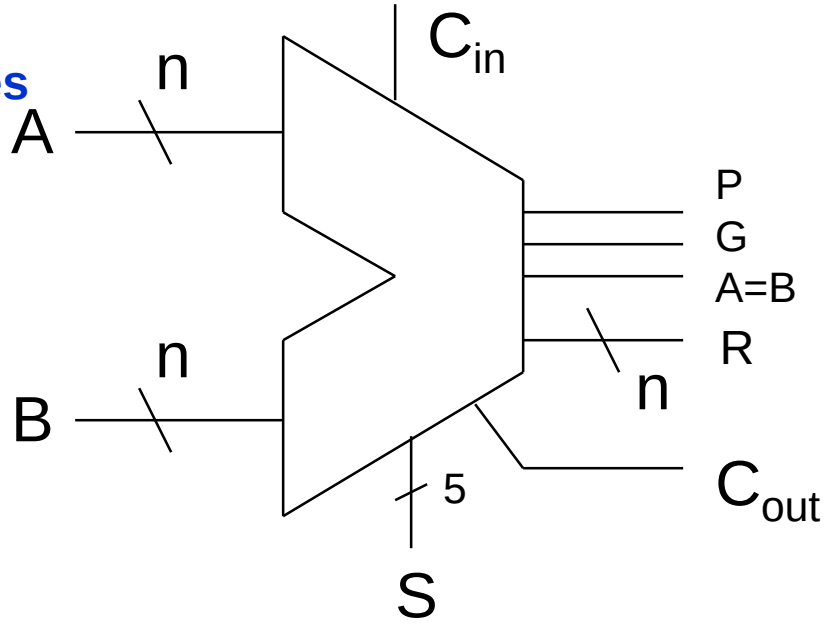



Par exemple




(Si X>Y) (Si X=Y) (Si X<Y)

# ALU (ou UAL)
# Unité Arithmétique et Logique




Exemple :

Résultat

$R = A + \overline{B}$

R = A + B

R = A + B + 1

...

R = A ou B

R = A nand B

...

# Merci pour votre attention