# Architecture des ordinateurs

## *Chapitre 2 : Logique booléenne et circuits combinatoires*

*
* *

Clément Moreau, Olivier Ploton

{clement.moreau, olivier.ploton}@univ-tours.fr

Université de Tours ~ Département informatique de Blois

Licence 2 - Informatique

# Sommaire


1. Algèbre de Boole
2. Logique combinatoire
3. Circuits logiques

# Algèbre de Boole

## Définition (Algèbre de Boole)

L'*algèbre de Boole* est une structure algébrique $(B, \{\vee, \wedge, \neg\}, \{0, 1\})$ avec :

- $B = \{b_0, b_1, ...\}$ est un ensemble fini ou dénombrable de variables booléennes,
- $\vee, \wedge, \neg$, respectivement les opérateurs de disjonction, conjonction et négation,
- 0, la constante de valeur FAUX,
- 1, la constante de valeur VRAI.

# Algèbre de Boole

Il est possible d'établir une lecture logique de l'algèbre de Boole selon la logique des propositions. (cf. cours de L1 de *Logique pour l'informatique*). À noter qu'il existe plusieurs conventions d'écriture des opérateurs.

| Disjonction | Conjonction | Négation |
|---|---|---|
| $\cup$ | $\cap$ | $\sim$ |
| $\vee$ | $\wedge$ | $\neg$ |
| $+$ | $\cdot$ | $-$ |
| $\mid$ | & | ! |
| $OR$ | $AND$ | $NOT$ |

| VRAI | FAUX |
|---|---|
| $\mathcal{V}$ | $\mathcal{F}$ |
| $\top$ | $\perp$ |
| 1 | 0 |

Nous conserverons dans ce chapitre les notations $\vee, \wedge, \neg$ et $0, 1$.

Les opérateurs s'expriment comme des lois de composition interne et peut donc se présenter sous forme de table. Une telle table est appelée *table de vérité*. En particulier, soit $(x, y) \in B^2$, deux variables booléennes alors :

| $x$ | $y$ | $x \vee y$ | $x \wedge y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

| $x$ | $\neg x$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

On peut composer les opérateurs et les variables pour donner des *fonctions booléennes.*

On note $\mathrm{Pr}(\circ)$ la priorité d'un opérateur $\circ$, alors : $\mathrm{Pr}(\vee) < \mathrm{Pr}(\wedge) < \mathrm{Pr}(\neg)$.

# Algèbre de Boole

Soient $(x, y, z) \in B^3$. On note les propriétés des opérateurs suivants :

| | | | |
|---|---|---|---|
| *Éléments neutres* | : | $x \vee 0 = x$ | $x \wedge 1 = x$ |
| *Commutativité* | : | $x \vee y = y \vee x$ | $x \wedge y = y \wedge x$ |
| *Associativité* | : | $(x \vee y) \vee z = x \vee (y \vee z)$ | |
| | | $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ | |
| *Idempotence* | : | $x \vee x = x$ | $x \wedge x = x$ |
| *Double négation* | : | $\neg\neg x = x$ | |
| *Distributivité* | : | $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ | |
| | | $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ | |
| *Contradiction* | : | $\neg x \wedge x = 0$ | |
| *Tiers-exclu* | : | $\neg x \vee x = 1$ | |
| *Loi d'absorption* | : | $x \vee (x \wedge y) = x$ | $x \wedge (x \vee y) = x$ |
| *Loi d'allégement* | : | $x \wedge (\neg x \vee y) = x \wedge y$ | |
| | | $x \vee (\neg x \wedge y) = x \vee y$ | |
| *Loi de de Morgan* | : | $\neg(x \wedge y) = \neg x \vee \neg y$ | |
| | | $\neg(x \vee y) = \neg x \wedge \neg y$ | |

Il existe d'autres opérateurs binaires. On peut montrer qu'en tout, il en existe $2^4$ différents.

Les plus connus sont : l'implication $\Rightarrow$, l'équivalence $\Leftrightarrow$, le ou exclusif $\oplus$, le Nor $\downarrow$, le Nand $\uparrow$.

| $x$ | $y$ | $x \Rightarrow y$ | $x \Leftrightarrow y$ | $x \oplus y$ | $x \downarrow y$ | $x \uparrow y$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

# Algèbre de Boole

On notera les équivalences (notées $\equiv$) suivantes entre les opérateurs :

- $x \Rightarrow y \equiv \neg x \vee y$
- $x \Leftrightarrow y \equiv (x \Rightarrow y) \wedge (y \Rightarrow x) \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$
- $x \oplus y \equiv (x \wedge \neg y) \vee (\neg x \wedge y) \equiv (x \vee y) \wedge (\neg x \vee \neg y) \equiv ((x \vee y) \wedge \neg(x \wedge y)$
- $x \uparrow y \equiv \neg(x \wedge y)$
- $x \downarrow y \equiv \neg(x \vee y)$

# Algèbre de Boole

## Définition (Syntaxe)

On appelle *fonction booléenne* toute application $f : \{0,1\}^n \rightarrow \{0,1\}$ dont la formule s'écrit au moyen d'opérateurs de l'algèbre de Boole.
Ainsi, la *syntaxe* définissant l'ensemble des fonctions booléennes $\mathcal{B}$ sur $B$ est le langage d'alphabet $\Sigma = B \cup \{\neg, \vee, \wedge, (,)\} \cup \{0,1\}$ et de règles inductives suivantes :

- Toute variable booléenne $b \in B$ est une fonction booléenne,
- Si $f \in \mathcal{B}$, alors $\neg f \in \mathcal{B}$
- Si $f, g \in \mathcal{B}$, alors $f \vee g \in \mathcal{B}$
- Si $f, g \in \mathcal{B}$, alors $f \wedge g \in \mathcal{B}$

En logique, il est nécessaire de distinguer la syntaxe de la sémantique. La syntaxe décrit la construction des formules (i.e. fonctions) tandis que la sémantique décrit leur valeur de vérité (i.e. sens).

# Algèbre de Boole

## Définition (Sémantique)

Soit une fonction booléenne $f$.
La *sémantique* d'une fonction booléenne $f$ consiste en la valeur de vérité de cette fonction.
Elle est définie à l'aide d'une *fonction d'interprétation* $I : B \rightarrow \{0,1\}^n$ qui est une distribution des valeurs de vérité aux variables booléennes.
Intuitivement, une fonction d'interprétation correspond à une ligne dans la table de vérité de $f$.
Une interprétation $I$ qui rend une fonction $f$ vraie, càd telle que $f(I) = 1$ est appelée un modèle de $f$ et est notée $I \models f$.

## Exemple

Soit l'ensemble de variables booléennes $B = \{x, y, z\}$.

On pose la fonction booléenne $f(x, y, z) = \neg((x \wedge y) \Rightarrow z)$ et la fonction d'interprétation $I(x, y, z) = (1, 1, 0)$

Ainsi,
$$f(I) = f(1,1,0) = \neg((1 \wedge 1) \Rightarrow 0) \\ = \neg(1 \Rightarrow 0) \\ = \neg(0) \\ = 1$$

Dès lors $I$ est un modèle de $f$. On a $I \models f$.

| $x$ | $y$ | $z$ | $f(x,y,z)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Algèbre de Boole

## Définition (Système complet de connecteurs)

Un *système complet de connecteurs* est un ensemble d'opérateurs $S$ tel que pour toute fonction $f \in \mathcal{B}$, il existe une fonction équivalente $f'$ construite uniquement à partir des connecteurs de $S$.

L'ensemble $\{\vee, \wedge, \neg\}$ est le système complet de connecteurs standard. Pour montrer qu'un ensemble de connecteurs $S$ est complet, on exprime les opérateurs $\{\vee, \wedge, \neg\}$ à l'aide de ceux de $S$.

## Exemple

Soit $B = \{x, y\}$. L'ensemble $S = \{\neg, \Rightarrow\}$ forme un système complet de connecteurs car :

- $x \vee y \equiv \neg x \Rightarrow y$
- $$\begin{aligned} x \wedge y &\equiv \neg(\neg x \vee \neg y) \\ &\equiv \neg(x \Rightarrow \neg y) \end{aligned}$$

# Algèbre de Boole

## Théorème (Complétude de $\uparrow$ et de $\downarrow$)

*Les opérateurs Nand $\uparrow$ et Nor $\downarrow$ forment chacun un système complet de connecteurs.*

## Démonstration.

On démontre pour $\{\uparrow\}$. On tente de reconstruire le système complet de connecteurs standard uniquement à l'aide $\{\uparrow\}$.
Soit $(x, y) \in B$.
On rappelle que $x \uparrow y \equiv \neg(x \wedge y)$.

- $\neg x \equiv \neg(x \wedge x) \equiv x \uparrow x$
- $x \wedge y \equiv \neg\left(\neg(x \wedge y)\right) \equiv \neg\left(x \uparrow y\right) \equiv (x \uparrow y) \uparrow (x \uparrow y)$
- $x \vee y \equiv \neg\left(\neg x \wedge \neg y\right)\right) \equiv \neg x \uparrow \neg y \equiv (x \uparrow x) \uparrow (y \uparrow y)$

$\square$

L'utilité des fonctions booléennes est que, tout circuit d'un processeur (quelque soit l'opération à réaliser) peut s'exprimer sous la forme d'une fonction booléenne.

Connaissant le résultat devant être obtenu par une fonction booléenne. Comment déterminer son expression ?

- Mise sous forme normale,
- Déterminer les mintermes / maxtermes,
- Simplification par tableau de Karnaugh.

## Définition (Littéral)

Un *littéral* est une variable booléenne ou sa négation (i.e. de la forme $b$ ou $\neg b$, pour $b \in B$).

## Définition (Forme normale)

Une *forme normale disjonctive* (FND) est une disjonction de la forme :

$$\bigvee_{i=1}^{k} \left( \bigwedge_{j=1}^{\ell_i} b_j \right)$$

où les $\ell_i \in \mathbb{N}^*$ dépendent de $i$ et les $b_j$ sont des littéraux.
Une *forme normale conjonctive* (FNC) est une conjonction de la forme :
$\bigwedge_{i=1}^{k} \left( \bigvee_{j=1}^{\ell_i} b_j \right)$

## Exemple

Soit $B = \{x, y, z\}$.

- $x \wedge (y \vee \neg x) \wedge z$ est une FNC
- $(x \wedge \neg z \wedge y) \vee y \vee \neg x$ est une FND
- $((x \vee \neg y) \wedge y) \vee \neg z$ est... rien du tout, on transforme :
  $((x \vee \neg y) \wedge y) \vee \neg z = (x \wedge y) \vee \neg z$ est une FND

On peut automatiquement écrire la formule logique correspondant à une fonction booléenne dès lors qu'on possède sa table de vérité.

### Définition (Minterme)

Soit $f(x_1, x_2, ..., x_n)$, alors, il existe $2^n$ interprétations de la fonction $f$.
Soit $\mathcal{M}$ l'ensemble des modèles de $f$ (i.e des interprétations $I$ telles que $I \models f$).

Soit $I \in \mathcal{M}$, un *minterme* $m_I$ de $f$ est une conjonction $\bigwedge_{i=1}^{n} x_i^{(I)}$ où

$$x_i^{(I)} = \begin{cases} x_i & \text{Si } I(x_i) = 1 \\ \neg x_i & \text{Si } I(x_i) = 0 \end{cases}.$$

La formule algébrique qui décrit $f$ est une FND de la forme :

$$f(x_1, x_2, ..., x_n) = \bigvee_{I \in \mathcal{M}} m_I$$

## Exemple

Soit la fonction $f$ de table de vérité suivante :

| $x$ | $y$ | $f(x,y)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Les modèles de $f$ sont $I_1(x,y) = (0,1)$, $I_2(x,y) = (1,1)$. On a $\mathcal{M} = \{I_1, I_2\}$.
Les mintermes de $f$ sont :

- $m_{I_1} = \neg x \wedge y$
- $m_{I_2} = x \wedge y$

Dès lors :

$$f(x,y) = (\neg x \wedge y) \vee (x \wedge y)$$

## Définition (Maxterme)

Soit $f(x_1, x_2, ..., x_n)$, alors, il existe $2^n$ interprétations de la fonction $f$.
Soit $\mathcal{F}$ l'ensemble des interprétations qui falsifient $f$ (i.e des interprétations $I$ telles que $I \not\models f$).

Soit $I \in \mathcal{F}$, un *maxterme* $M_I$ de $f$ est une disjonction $\bigvee_{i=1}^{n} x_i^{(I)}$ où

$$x_i^{(I)} = \begin{cases} \neg x_i & \text{Si } I(x_i) = 1 \\ x_i & \text{Si } I(x_i) = 0 \end{cases}$$

La formule algébrique qui décrit $f$ est une FNC de la forme :

$$f(x_1, x_2, ..., x_n) = \bigwedge_{I \in \mathcal{F}} M_I$$

## Exemple

On reprend la précédente fonction $f$ de table de vérité suivante :

| $x$ | $y$ | $f(x,y)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Les falsifications de $f$ sont $I_1(x,y) = (0,0)$, $I_2(x,y) = (1,0)$. On a $\mathcal{F} = \{I_1, I_2\}$.
Les maxtermes de $f$ sont :

- $M_{I_1} = x \vee y$
- $M_{I_2} = \neg x \vee y$

Dès lors :

$$f(x,y) = (x \vee y) \wedge (\neg x \vee y)$$

## Corollaire

*Toute fonction $f \in \mathcal{B}$ peut s'exprimer sous forme de mintermes (resp. sous forme de maxtermes).*

On va chercher maintenant à simplifier les expressions obtenues à partir des mintermes/maxtermes à l'aide de la méthode des tableaux de Karnaugh.

# Logique combinatoire

- La méthode de Karnaugh consiste à présenter les états d'une fonction logique sous la forme d'un *tableau à double entrée.*
- Chaque case du tableau correspond à une combinaison des variables d'entrées, c'est-à-dire à une ligne de la table de vérité.
- Le tableau de Karnaugh aura autant de cases que la table de vérité possède de lignes.
- Les lignes et les colonnes du tableau sont numérotées selon le code de Gray : *à chaque passage d'une case à l'autre, une seule variable change d'état.*

| $cd \backslash ab$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |

| $a \backslash bc$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |

| $a \backslash b$ | 0 | 1 |
|---|---|---|
| 0 | | |
| 1 | | |

Soit la table de Karnaugh d'une fonction booléenne $f$.

- Regrouper les cases adjacentes de “1” selon des paquets de taille $2^n$ avec $n$ le plus grand possible.
- Une même case peut faire partie de plusieurs regroupements.
- Les regroupements peuvent se faire au delà des bords : les côtés/coins ont des codes Gray voisins.
- Toute case contenant un 1 doit faire partie d'au moins un regroupement, mais aucun 0 ne doit y être.
- Pour chaque rectangle, on élimine les variables qui changent d'état, l'on ne conserve que celles qui restent fixes.
  On met en conjonction les variables fixes afin d'obtenir des mintermes de $f$.
- Les mintermes obtenus sont ensuite mis en disjonction afin d'obtenir une simplification de la FND de $f$.

# Logique combinatoire

| $ab \backslash cd$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | 1 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 0 | 1 | 1 |

| $a \backslash bc$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $\varphi$ |
| 1 | 1 | $\varphi$ | 0 | 0 |

| $a \backslash b$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

FIGURE - Quelques exemples de regroupements

# Logique combinatoire

## Exemple

Tableaux de Karnaugh à deux dimensions :

| $a \backslash b$ | 0 | 1 | |
|---|---|---|---|
| 0 | 1 | 0 | |
| 1 | 1 | 1 | $a$ |
| | $\neg b$ | | |

$$f(a,b) = a \vee \neg b$$

| $a \backslash b$ | 0 | 1 | |
|---|---|---|---|
| 0 | 1 | 0 | $a \vee \neg b$ |
| 1 | 1 | 1 | |

## Exemple

Tableaux de Karnaugh à trois dimensions :

$$f(a,b,c) = b \vee (\neg a \wedge \neg c)$$

| $a \backslash bc$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |

| $a \backslash bc$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |

$$f(a,b,c) = (b \vee \neg c) \wedge (\neg a \vee b)$$

# Circuits logiques

Quel rapport entre la logique booléenne et les ordinateurs ?

Les *circuits logiques* sont les composants de base des ordinateurs !

## Définition (Circuit logique)

Un *circuit logique* peut être vu comme une boîte noire ayant $n \geq 1$ ports d'entrée $e_1, e_2, ..., e_n$ et $m \geq 1$ ports de sortie $s_1, s_2, ..., s_m$.
Il traite des informations codées sur $n$ bits et donne des informations codées sur $m$ bits.
Ainsi, le comportement d'un circuit logique est analogue à celui d'une fonction booléenne $f : \{0,1\}^n \rightarrow \{0,1\}^m$.

- Ces circuits électroniques sont qualifiées de "logiques" car un bit d'information 0 est assimilé à la valeur de vérité faux et 1 à vrai.
- Le codage de l'information, en entrée ou sortie, est représenté par l'absence (0) ou la présence (1) d'une tension électrique grâce aux *transistors*.

# Circuits logiques

On cherche à réaliser le circuit logique d'un additionneur binaire.

On considère le schéma fonctionnel suivant :

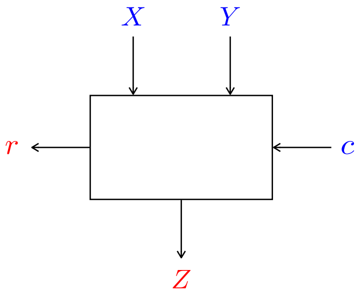



FIGURE - Abstraction d'un additionneur binaire

- Les entrées sont les bits $X$ et $Y$ et le bit de report $c$,
- Les sorties sont le bits de sortie $Z$ et la retenue $r$,
- Le résultat de $\langle X + Y + c \rangle_2$ est donné par $Z$ et $r$.

| $X$ | $Y$ | $c$ | $Z$ | $r$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

On obtient :

- $Z = (\neg X \wedge \neg Y \wedge c) \vee (\neg X \wedge Y \wedge \neg c) \vee (X \wedge \neg Y \wedge \neg c) \vee (X \wedge Y \wedge c)$
- $r = (\neg X \wedge Y \wedge c) \vee (X \wedge \neg Y \wedge c) \vee (X \wedge Y \wedge \neg c) \vee (X \wedge Y \wedge c)$

# Circuits logiques

On dresse les tableaux de Karnaugh pour simplifier $Z$ et $r$.

$Z$

| $c \backslash XY$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

$r$

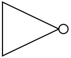
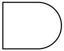
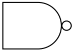
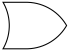
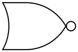
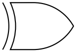
| $c \backslash XY$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

- On déduit la FND simplifiée pour $r$ suivante :
  $r = (X \wedge Y) \vee (X \wedge c) \vee (Y \wedge c) = (X \wedge Y) \vee [(X \oplus Y) \wedge c]$
- Grâce à l'opérateur $\oplus$, on peut écrire $Z$ plus simplement telle que :
  $Z = (X \oplus Y) \oplus c$

On rappelle que $x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y) = (x \vee y) \wedge \neg(x \wedge y)$

# Circuits logiques

Un circuit logique est représenté grâce à des portes logiques.

| Porte logique | Opération | Nom |
|---|---|---|
|  | $\neg x$ | NON (NOT) |
|  | $x \wedge y$ | ET (AND) |
|  | $x \uparrow y$ | NON-ET (Nand) |
|  | $x \vee y$ | OU (OR) |
|  | $x \downarrow y$ | NON-OU (Nor) |
|  | $x \oplus y$ | OU exclusif (XOR) |

# Circuits logiques

- Une porte peut posséder plus d'entrées que l'arité de l'opération qu'elle implémente. Il faut que l'opération soit *associative* pour ça.
- On peut répéter une variable à l'aide d'une *bifurcation* représentée par un $\bullet$ au sein du circuit.

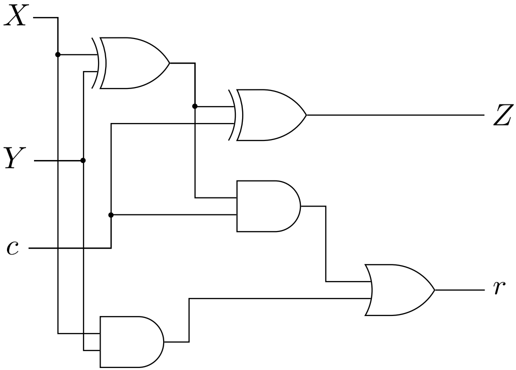



FIGURE - Circuit logique de l'additionneur binaire

Si on reprend les formules de notre additionneur binaire :

- $r = (X \wedge Y) \vee [(X \oplus Y) \wedge c]$
- $Z = (X \oplus Y) \oplus c$

On obtient le circuit logique ci-contre.

Pour additionner deux entiers naturels $(X, Y)$ codés sur 16 bits, on a alors :

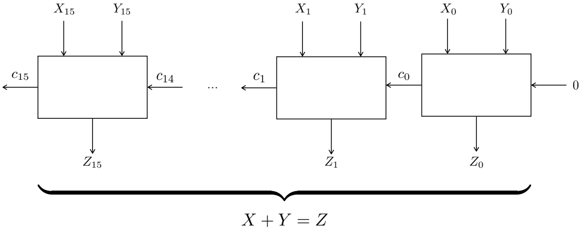
- 16 entrées $X_i$ pour la représentation binaire,
- 16 entrées $Y_i$ pour la représentation binaire,
- 16 sorties $Z_i$ pour la représentation binaire de $Z = X + Y$,
- 1 bit de sortie supplémentaire au cas d'un éventuel dépassement de capacité.

On a au total 32 entrées et 17 sorties ! Une telle spécification par table de vérité n'est pas envisageable. De même, écrire les 17 formules spécifiant les 17 sorties sous forme normale est très laborieux !

# Circuits logiques

## Solution

On construit un circuit additionneur 1 bit. On composera autant de "circuit 1 bit" que nécessaire. Le circuit sera alors un *circuit en série* : une partie du circuit doit attendre le résultat d'une autre partie du circuit, d'où le besoin d'une horloge.



# Circuits logiques

## Coût d'un circuit logique

Un circuit logique est "*bien modélisé*" si et seulement si :

1. Il minimise le nombre d'opérateurs différents,
2. Il minimise le nombre total d'opérateurs.

Besoin de minimiser les coûts de commande des composants, la taille des circuits, la consommation électrique, etc.

## Décodeur

Un *décodeur* traduit un nombre codé en binaire en activant la ligne correspondant à ce nombre. Il comprend $k$ entrées et $2^k$ sorties. La $i$-ème sortie du décodeur vaut 1 si les $k$ entrées forment l'entier $i$, (i.e. $\langle e_{k-1} ... e_0 \rangle_2 = \langle i \rangle_{10}$).

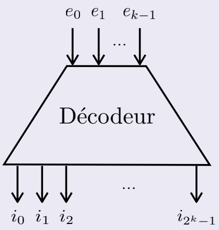


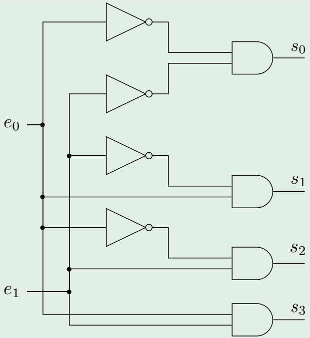

FIGURE - Symbole du décodeur à $k$ entrées

# Circuits logiques

## Exemple

On donne l'exemple de l'implémentation d'un décodeur à 2 bits. Sa table de vérité est telle que :

| $e_1$ | $e_0$ | $s_3$ | $s_2$ | $s_1$ | $s_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |



# Circuits logiques

## Quelles utilités ?

- Dans une UAL : Supposons que nous ayons une puce qui implémente quatre opérations (Par exemple $\vee, \wedge, \neg, +$), en attribuant un code opérationnel à chacune, par exemple $+$ : 00, un décodeur peut servir à activer le circuit correspondant à l'opération adéquate.
- Gestion de la mémoire : Comme la mémoire est organisée sous forme matricielle où chaque case possède une adresse. Un décodeur peut servir, connaissant l'adresse à accéder, à activer la cellule mémoire correspondante.

## Définition (Multiplexeur)

Un *multiplexeur* est [un peu] l'inverse d'un décodeur. Un multiplexeur $k$ bits permet de sélectionner une entrée parmi $2^k$ disponibles. Un multiplexeur $k$ bits possède $k + 2^k$ entrées et une seule sortie. Les $k$ premières entrées $b_0, ..., b_{k-1}$ sont appelées bits d'adresses et donnent le numéro de l'entrée à sélectionner parmi $e_0, ...e_{2^k-1}$. La sortie $s = e_i$ telle que $\langle b_{k-1}, ...b_0\rangle_2 = \langle i\rangle_{10}$.

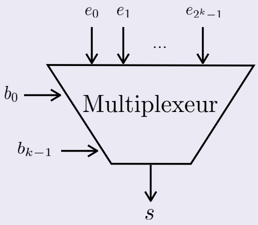


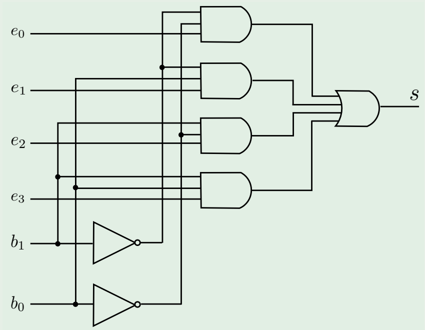

FIGURE - Symbole du multiplexeur à $k + 2^k$ entrées

# Circuits logiques

## Exemple

Soit la fonction $s$ résultante de la sortie d'un multiplexeur 2 bits. On a $s$ telle que :

$$s = (b_1 \wedge b_0 \wedge e_3) \vee (b_1 \wedge \neg b_0 \wedge e_2) \vee (\neg b_1 \wedge b_0 \wedge e_1) \vee (\neg b_1 \wedge \neg b_0 \wedge e_0)$$



# Circuits logiques

L'unité arithmétique et logique (UAL) est représentée comme un ensemble de modules permettant les opérations logiques classiques $\wedge, \vee, \neg$ et $+$ (addition usuelle).
Un processeur $n$ bits implémente alors une UAL composée de $n$ circuits comme celui-ci à droite, d'une manière analogue à l'additionneur vu précédemment. Cette mise en séquence permet alors de traiter des mots de $n$ bits.

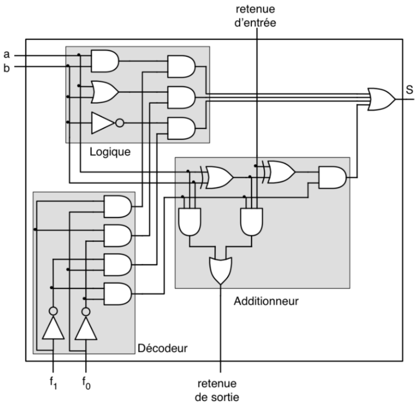



FIGURE - UAL 1 bit

Image extraite de *L'architecture de l'ordinateur*, Tanenbaum (2005)

# Conclusion

Pour terminer cette partie, j'aimerais revenir sur l'importance qu'une fonction logique a d'être normalisée pour réduire les coûts. Si l'on considère la fonction suivante :

$$f(x, y, z) = \neg(x \wedge y) \Rightarrow \neg z$$

- Cette fonction demande l'implémentation de 4 opérateurs, dont 3 différents, alors que si l'on cherche une version équivalente à l'aide des tables de vérité, on montre que $f \equiv (x \uparrow y) \uparrow z$.
- Une telle représentation est beaucoup plus économe ! Elle n'utilise qu'un seul type de porte logique et minimise le nombre de composants.

# Conclusion

- Les *circuits logiques* sont les composants de base des ordinateurs !
- Un circuit logique est une représentation matérielle du concept de *fonction logique*.
- L'*algèbre booléenne* permet de manipuler les fonctions logiques :
  - On cherche à les simplifier à l'aide des *tableaux de Karnaugh* et des *tables de vérité*.
  - On *réduit alors les coûts* pour l'implémentation des circuits.
- Pour réaliser des circuits complexes, on compose plusieurs circuits élémentaires.
  - Cas de l'*additionneur* et du *UAL* 1 bit.
- Aujourd'hui, les circuits ne sont plus implémentés porte par porte mais sont gravés sur des galettes de silicium (ou *wafer*). Les composants élémentaires actuels sont généralement des modules qui regroupent plusieurs portes selon une fonction donnée.
  - Par exemple, on trouve les *décodeurs* et *multiplexeurs*, les *additionneurs*, etc.