Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 4
Algèbre de Boole et circuits logiques

Yann Thorimbert

Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. Codage des nombres
3. Codage des médias
4. **Circuits logiques** ←
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Rappel

- On peut coder les informations qui nous intéressent sous forme d'états binaires (cf. chapitres précédents).

- On veut réaliser des raisonnements logiques qui portent sur ces données : autrement dit, on veut effectuer un **traitement** de ces données.

Exemple de table de vérité

- Considérons pour l'exemple un cas simplifié d'infraction routière en Suisse :

  "*Une personne est en excès de vitesse si elle roule à plus de 50 km/h à l'intérieur d'une localité.*"

- Nous voulons qu'un ordinateur puisse **traiter** ce type de **proposition logique**.

- Pour formaliser une telle phrase, on peut :
  1) Repérer les sous-parties : *être en excès de vitesse*, *être hors localité*.
  2) Résumer le tout sous la forme d'une **table de vérité**.

Exemple de table de vérité

Limitation de vitesse dans les localités en Suisse (où 1 = "oui", 0 = "non").

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

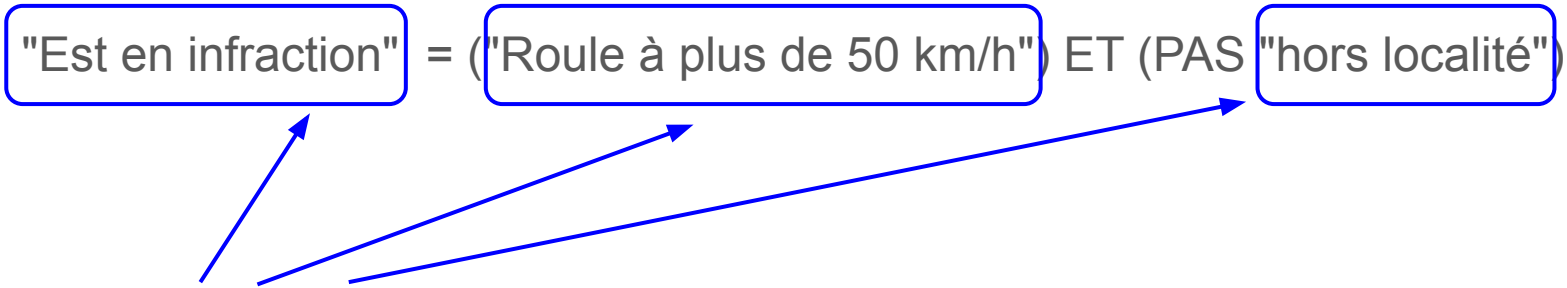
"Est en infraction" est vrai si et seulement si :
"roule à + de 50 km/h" est vrai et "est hors localité" est faux.

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

Les éléments d’une expression logique

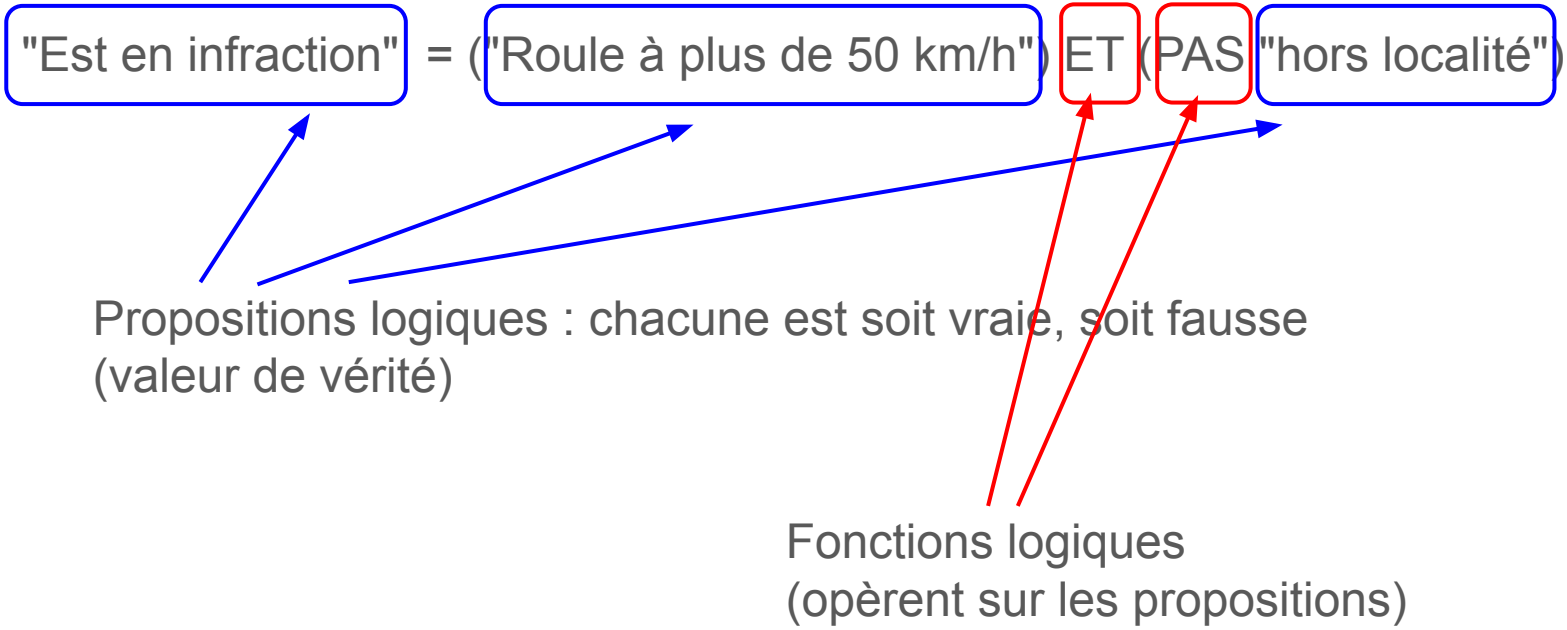
"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

Les éléments d'une expression logique



Propositions logiques : chacune est soit vraie, soit fausse (valeur de vérité)

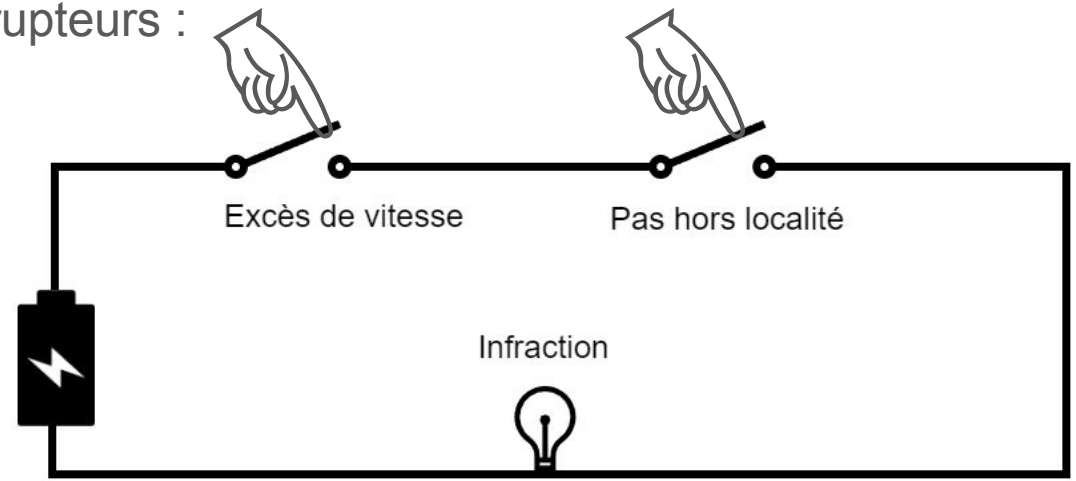
Les éléments d'une expression logique



De table de vérité à circuit logique

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

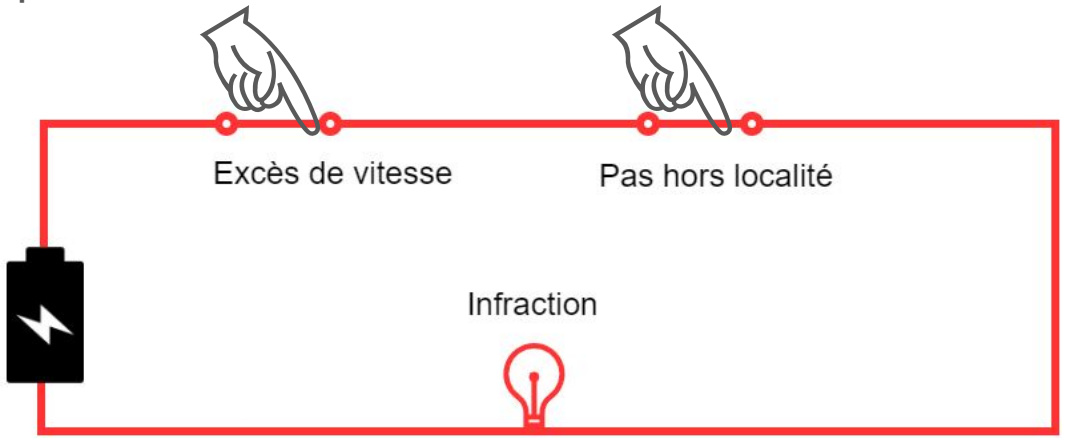
On pourrait **réaliser** cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs :



De table de vérité à circuit logique

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

On pourrait réaliser cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs :



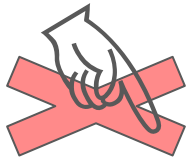
De table de vérité à circuit logique

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

On pourrait réaliser cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs.

Cependant, on veut **automatiser** le processus.

De plus, on veut **abstraire** le processus.



De table de vérité à circuit logique

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

- Une table de vérité peut tout aussi bien être résumée par un **schéma**.
- Le schéma doit incorporer les **fonctions logiques** que sont ET et PAS.

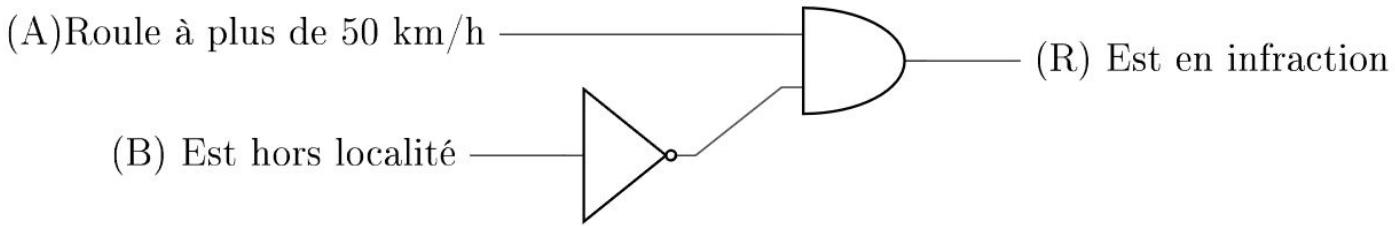
De table de vérité à circuit logique

"Est en infraction" = ("Roule à plus de 50 km/h") ET (PAS "hors localité")

- Une table de vérité peut tout aussi bien être résumée par un **schéma**.
- Utilisons les symboles arbitraires suivants, nommés **portes logiques** :
  - ET est arbitrairement symbolisé par le symbole 
  - PAS est arbitrairement symbolisé par le symbole 

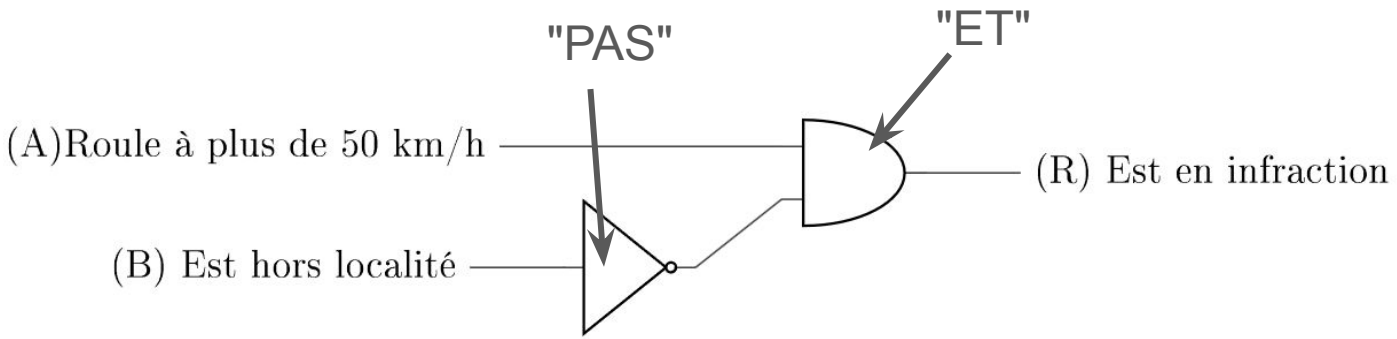
De table de vérité à circuit logique

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0



De table de vérité à circuit logique

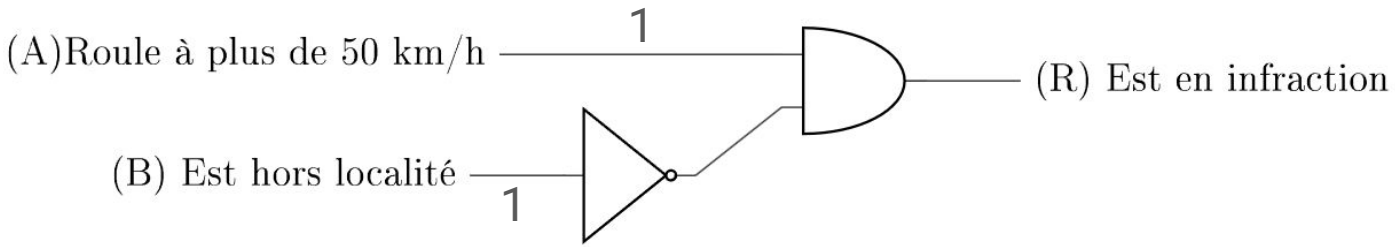
Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0



De table de vérité à circuit logique

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

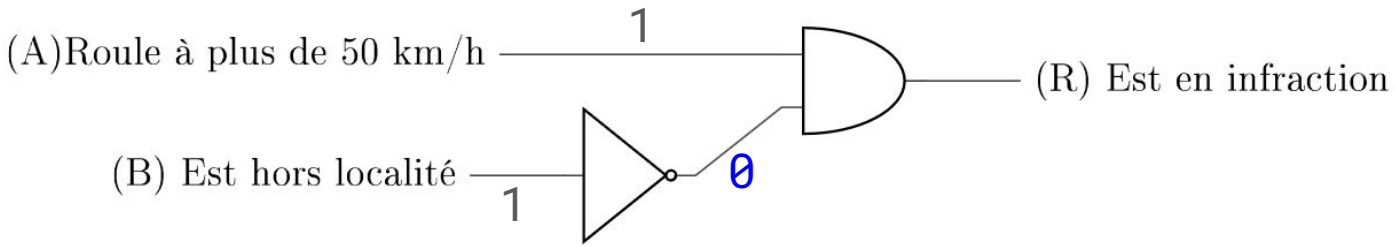
Dans cet exemple, la personne roule à plus de 50 km/h hors localité :



De table de vérité à circuit logique

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

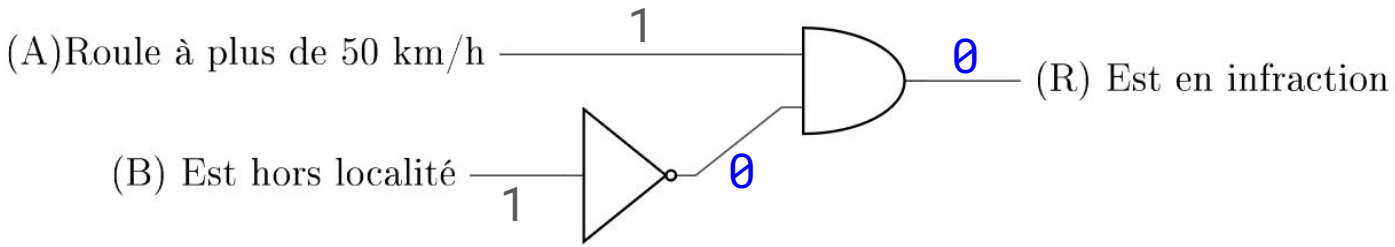
Dans cet exemple, la personne roule à plus de 50 km/h hors localité :



De table de vérité à circuit logique

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

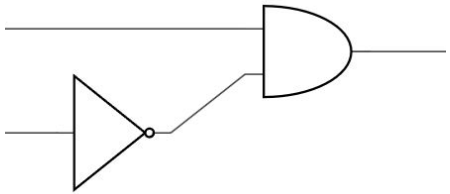
Dans cet exemple, la personne roule à plus de 50 km/h hors localité :



De table de vérité à circuit logique

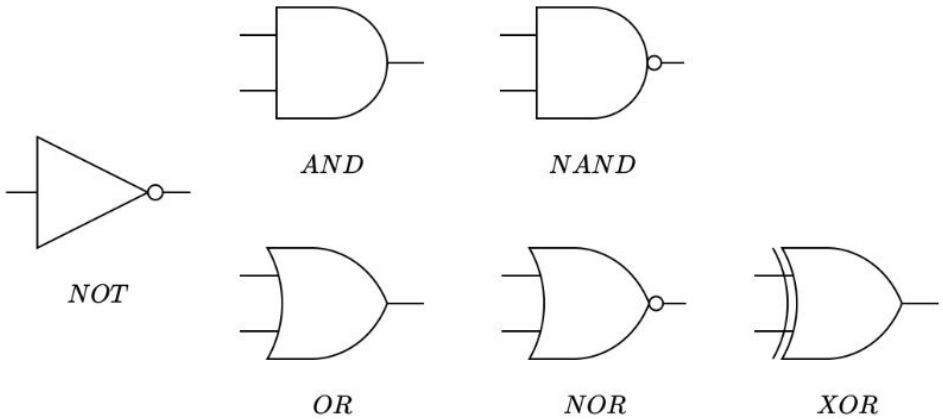
- Table de vérité faite de **propositions logiques binaires**.
- Circuit logique fait de **portes logiques** prenant des valeurs binaires en entrée et retournant des valeurs binaires en sortie.
- Le circuit **réalise** la table de vérité.

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

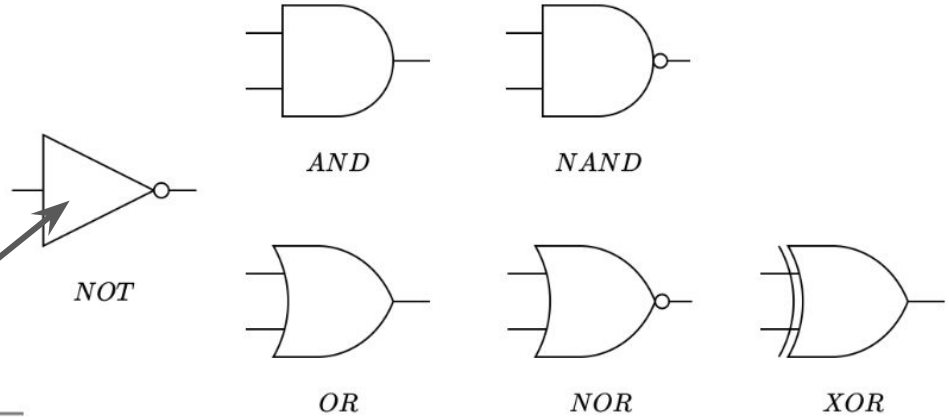


Différents types de portes

Chaque porte correspond à une table de vérité.



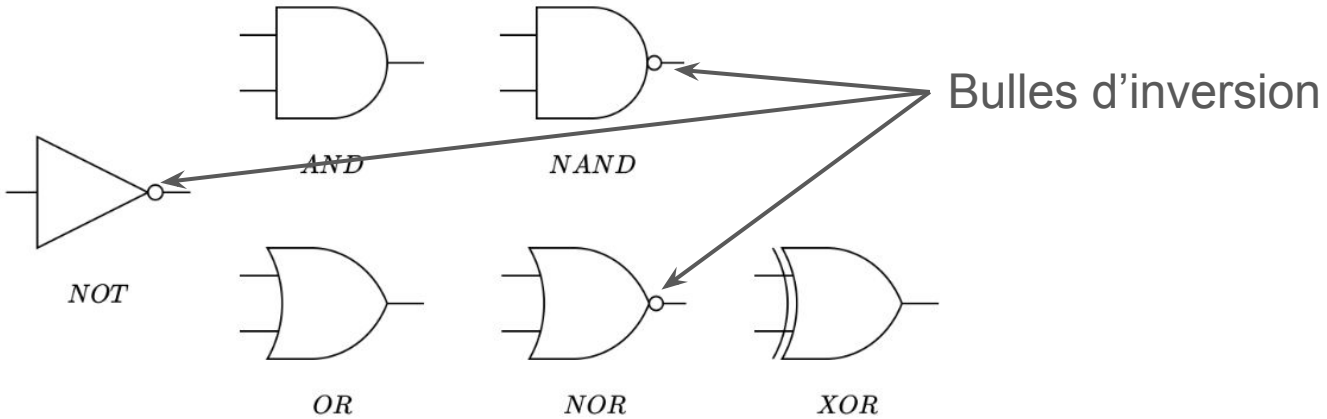
Différents types de portes | **NOT**



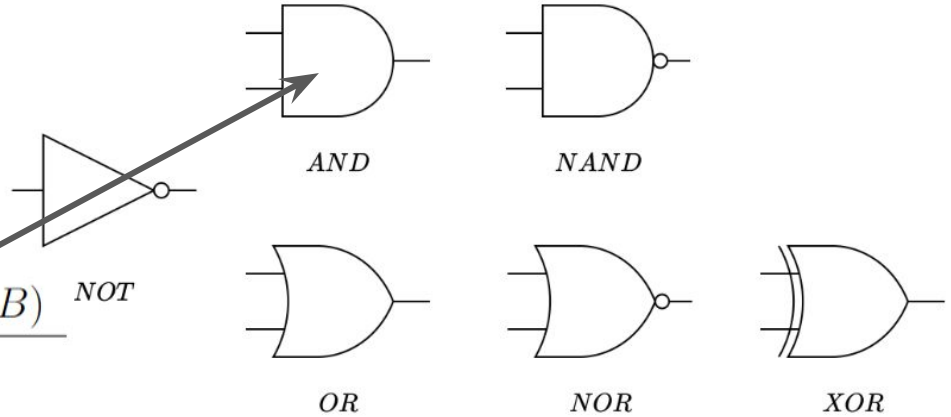
A	$NOT(A)$
0	1
1	0

Différents types de portes | **Bulles d'inversion**

Les portes suivies d'une bulle d'inversion sont l'inverse de leur pendant sans bulle.



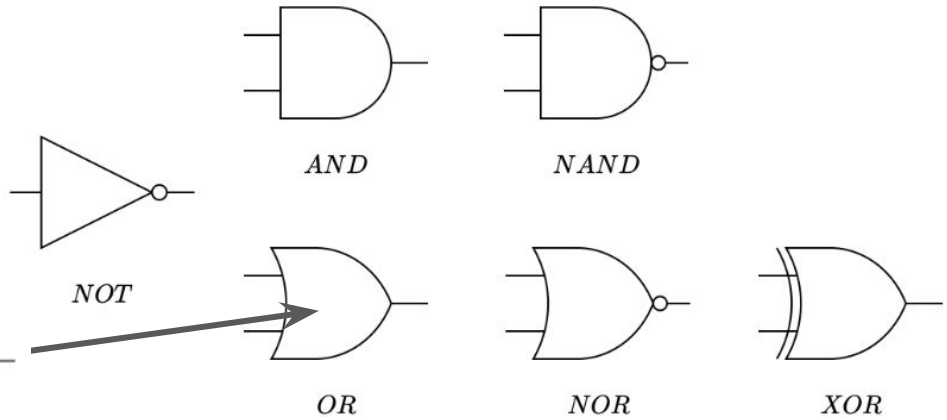
Différents types de portes | **AND**



A	B	$AND(A, B)$
0	0	0
0	1	0
1	0	0
1	1	1

Différents types de portes | **OR**

Chaque porte possède une table de vérité.

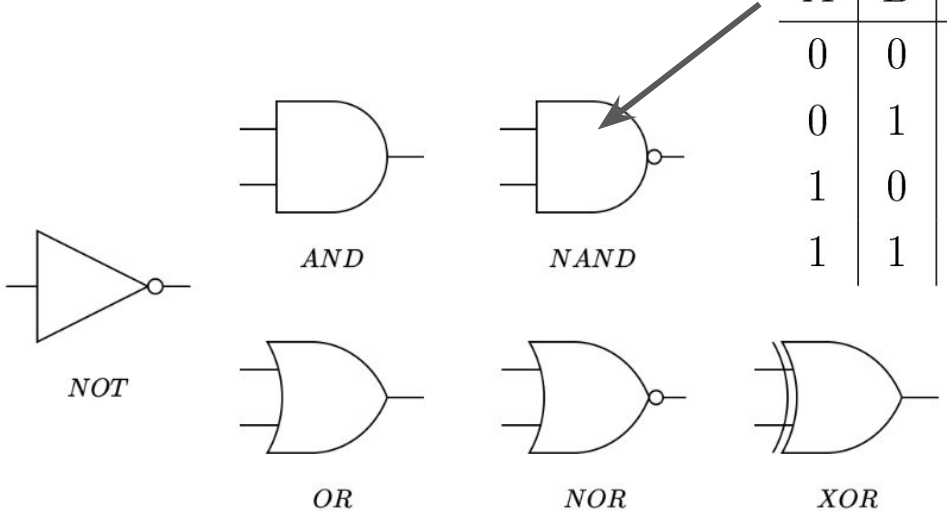


A	B	$OR(A, B)$
0	0	0
0	1	1
1	0	1
1	1	1

Différents types de portes | **NAND**

Not AND

"Tous"

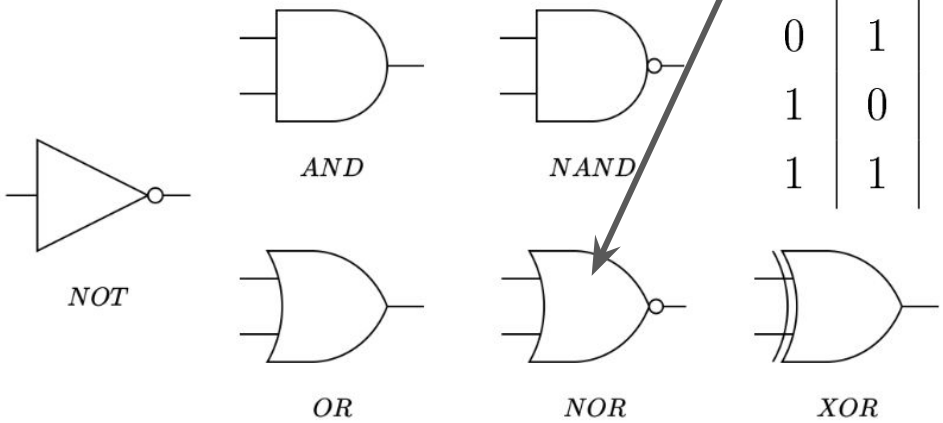


A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0

Différents types de portes | **NOR**

Not **OR**

"Ni"



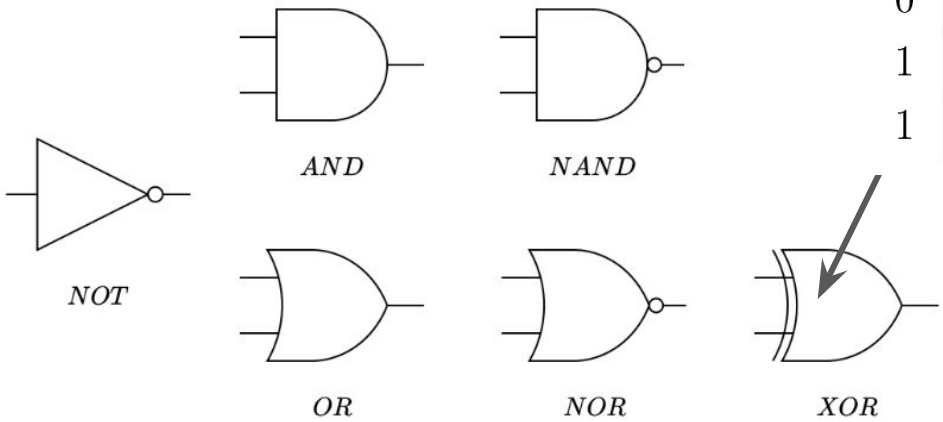
A	B	$NOR(A, B)$
0	0	1
0	1	0
1	0	0
1	1	0

Différents types de portes | **XOR**

E**x**clusive **OR**

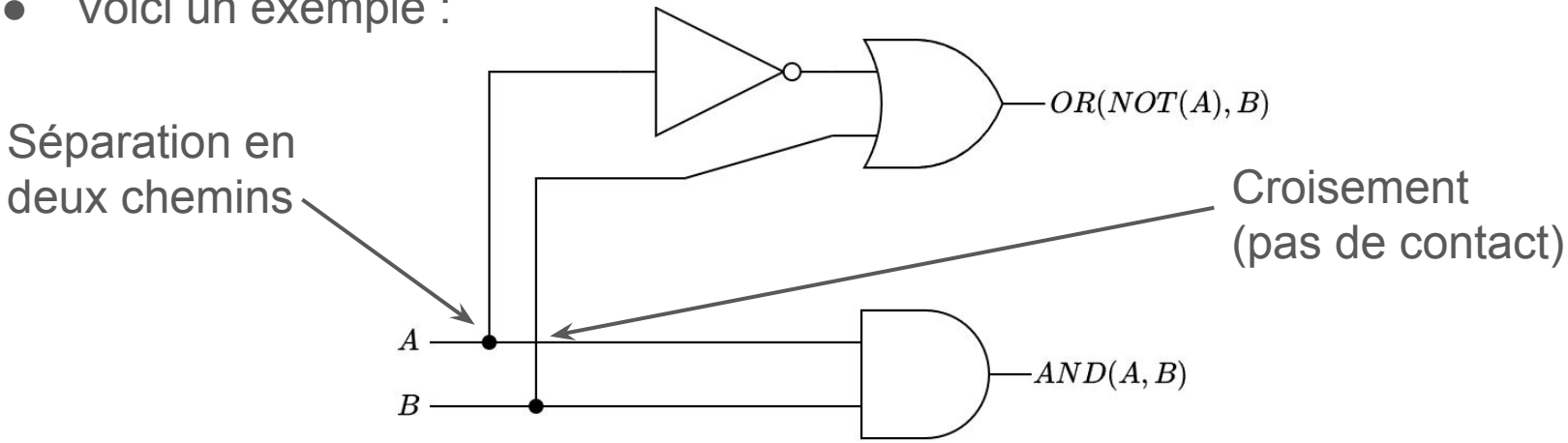
"Ou bien"

A	B	$XOR(A, B)$
0	0	0
0	1	1
1	0	1
1	1	0



Note sur les schémas de circuits logiques

- On représente les noeuds ou jonctions avec un point.
- On représente les fils qui se croisent sans point.
- Parfois, nous écrirons des expressions sur le schéma.
- Voici un exemple :



Votamatic

- On considère la table de vérité suivante,
  traduite en français par "(PAS A ET PAS B) OU (A ET B)" :
- Quel circuit logique lui correspond ?

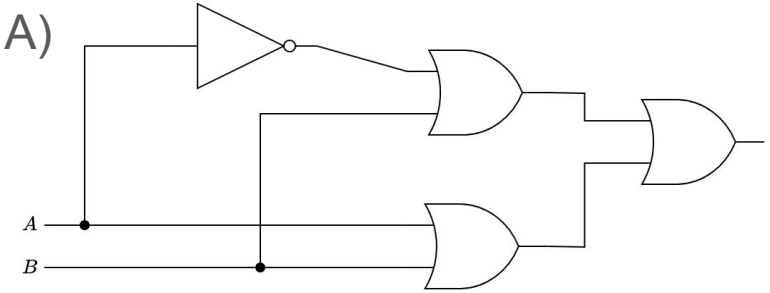
A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

Votamatic.unige.ch

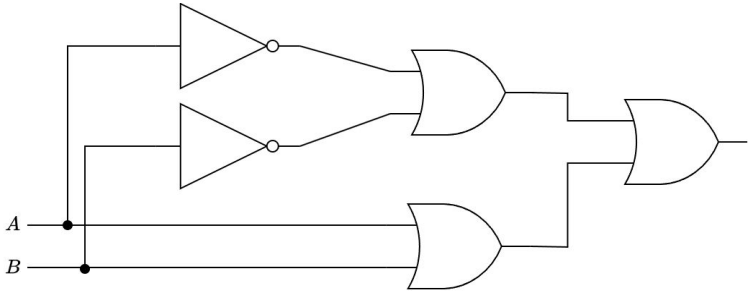
Code VCVT

Votamatic code VCVT - (PAS A ET PAS B) OU (A ET B)

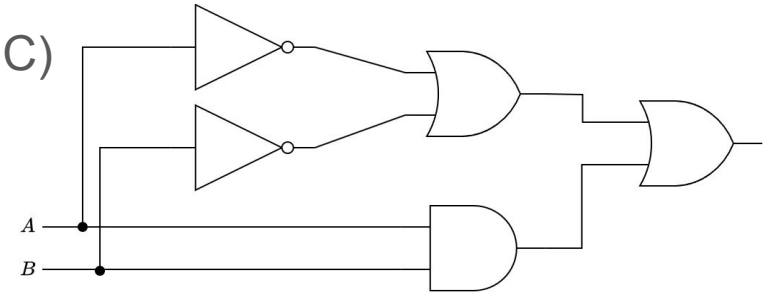
A)



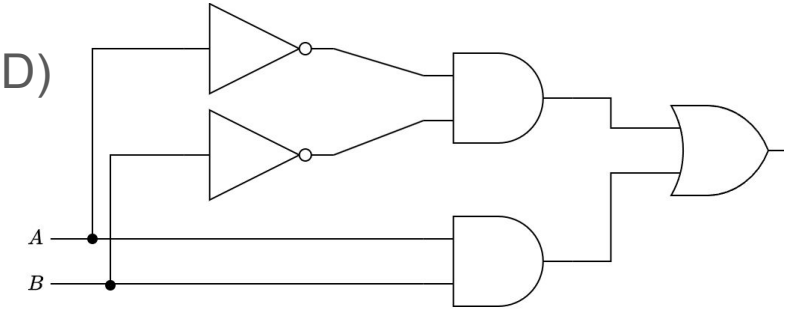
B)



C)



D)



Rappel : transistors

- Les transistors sont les éléments électroniques de base des ordinateurs actuels.
- Robinets à électrons : laissent (ou pas) passer le courant électrique entre deux bornes en fonction de la tension à une troisième borne.
- Si l’on peut réaliser les schémas dans la vie réelle, alors on peut simuler un raisonnement logique grâce à une machine !

Aparté physique

- Quand un courant électrique traverse une résistance, on mesure une différence de tension aux bornes de la résistance.
- Loi d'Ohm : $\Delta U = U_1 - U_2 = R \cdot I$
- On dit que la résistance fait chuter la tension.

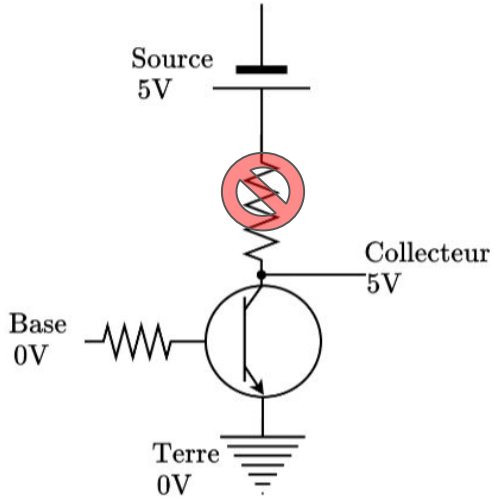
De transistors à portes logiques

Fonction logique NOT :

Base **n'active pas** le transistor :

⇒ Le courant ne s'établit pas

⇒ Pas de chute de tension.



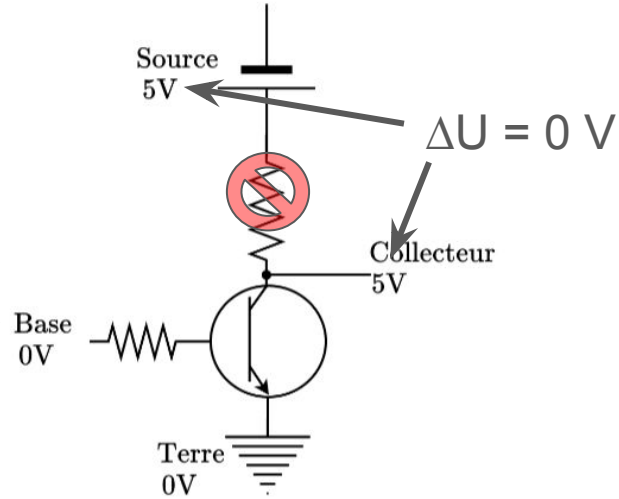
De transistors à portes logiques

Fonction logique NOT :

Base **n'active pas** le transistor.

⇒ Le courant ne s'établit pas.

⇒ Pas de chute de tension.



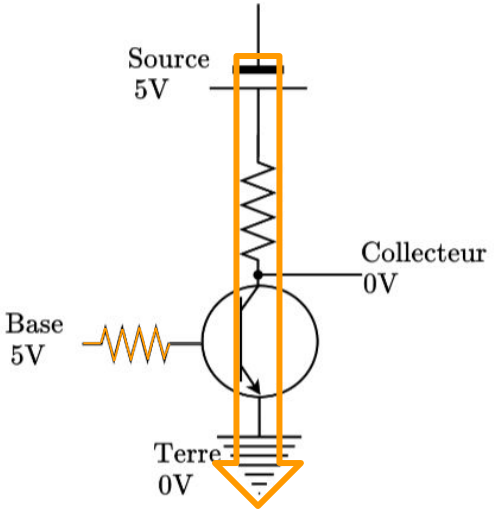
De transistors à portes logiques

Fonction logique NOT :

Base **active** le transistor.

⇒ Le courant s'établit.

⇒ Chute de tension.



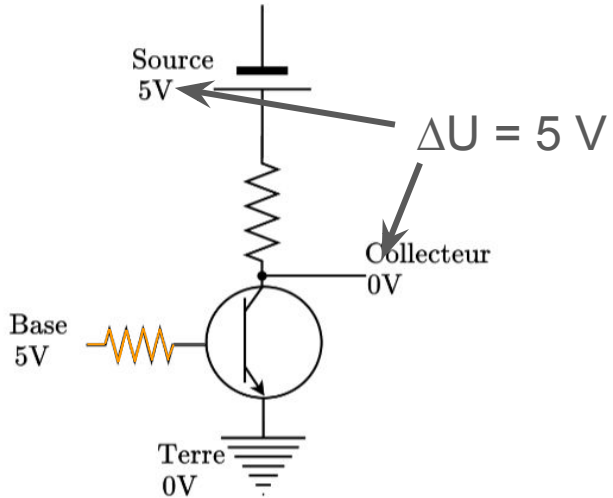
De transistors à portes logiques

Fonction logique NOT :

Base **active** le transistor.

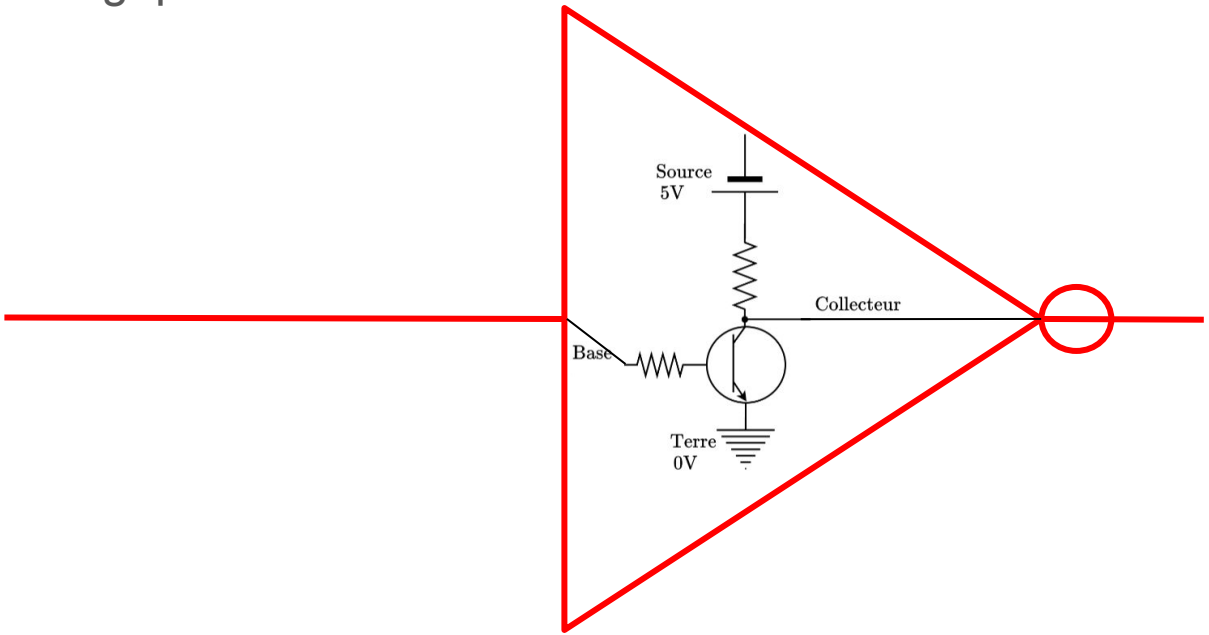
⇒ Le courant s'établit.

⇒ Chute de tension.



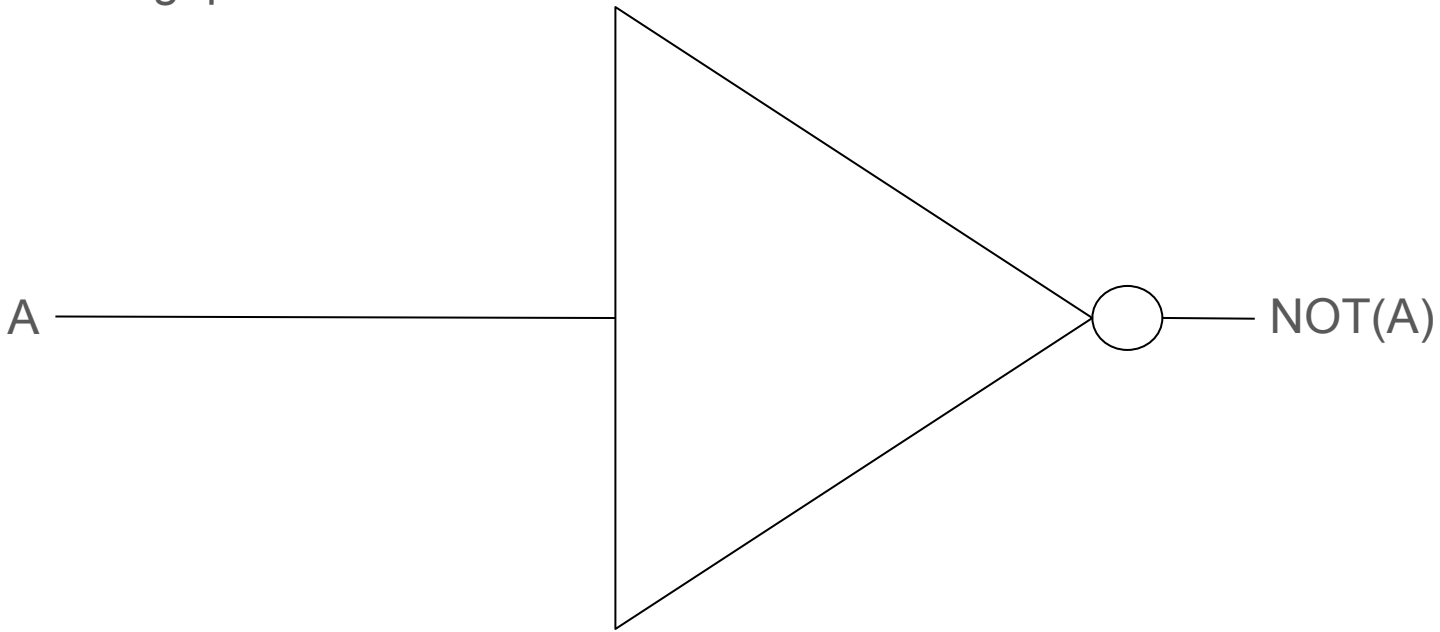
De transistors à portes logiques | **Abstraction**

Fonction logique NOT :



De transistors à portes logiques | **Abstraction**

Fonction logique NOT :

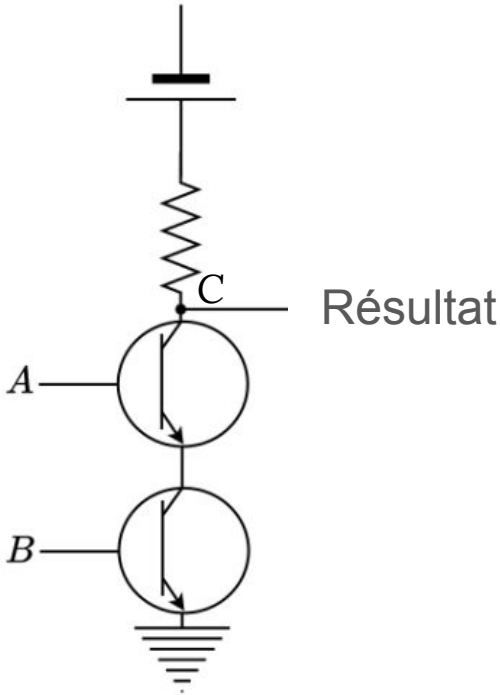


Une première porte : transistors en série

- Il y a chute de tension au collecteur si le courant s' établit à travers l'un ET l'autre des transistors.

- Autrement dit : $U_c > 0$ seulement si au moins un des deux transistors n'est pas activé.

A	B	Résultat

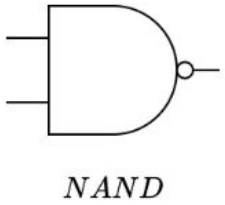


Porte NAND

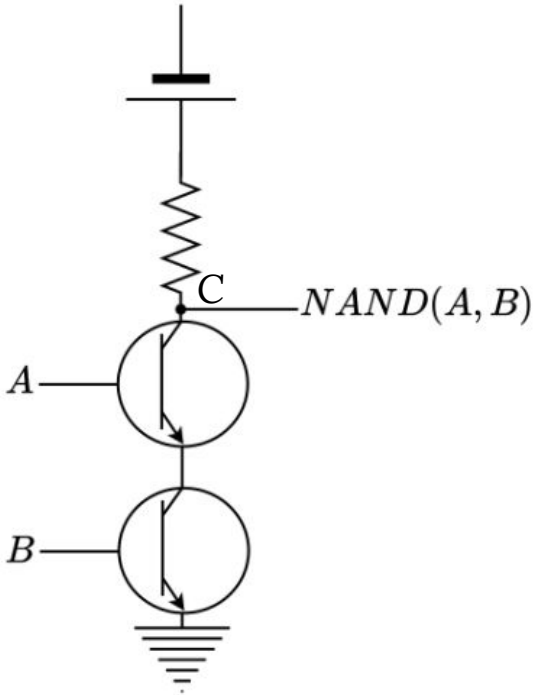
- Il y a chute de tension au collecteur si le courant s' établit à travers l'un ET l'autre des transistors.

- Autrement dit : $U_c > 0$ seulement si au moins un des deux transistors n'est pas activé.

A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0



NAND

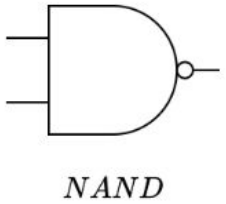


Porte NAND

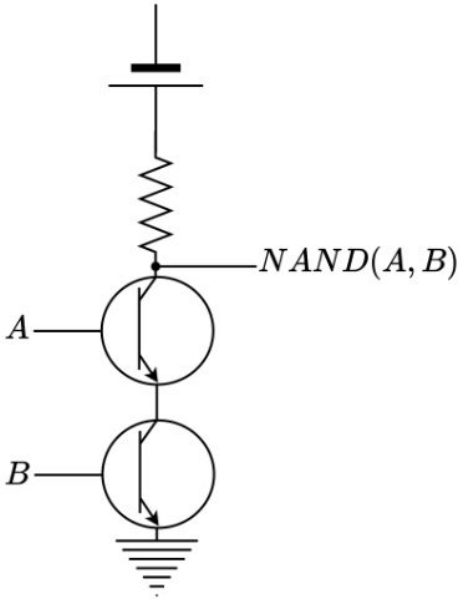
Fonction logique

A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0

Porte logique



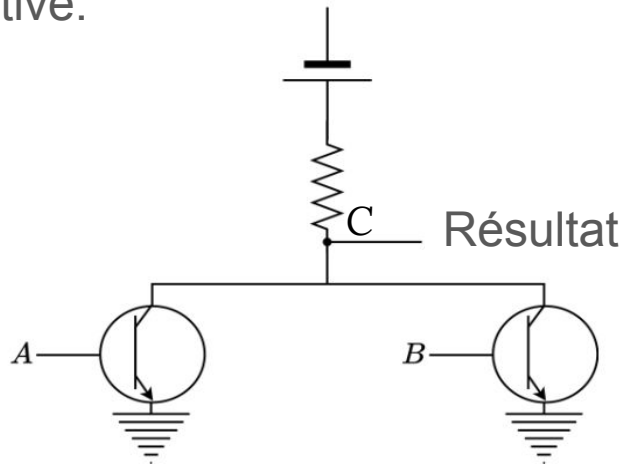
Circuit électronique



Une seconde porte : transistors en parallèle

- Il y a chute de tension dès que A OU B est activé.
- $U_c > 0$ seulement si aucun transistor n'est activé.

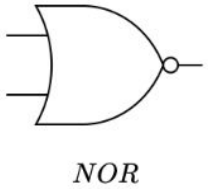
A	B	Résultat

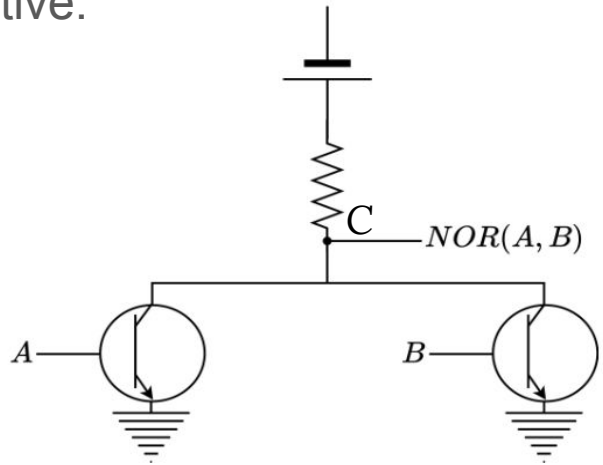


Porte NOR

- Il y a chute de tension dès que A OU B est activé.
- $U_c > 0$ seulement si aucun transistor n'est activé.

A	B	$NOR(A, B)$
0	0	1
0	1	0
1	0	0
1	1	0





Combinaison de fonctions logiques

On observe que NAND est une inversion de AND au même titre que NOR est une inversion de NOR.

A	B	AND	NAND	OR	NOR
0	0				
0	1				
1	0				
1	1				

Combinaison de fonctions logiques

On observe que NAND est une inversion de AND au même titre que NOR est une inversion de NOR.

A	B	AND	NAND	OR	NOR
0	0	0		0	
0	1	0		1	
1	0	0		1	
1	1	1		1	

Combinaison de fonctions logiques

On observe que NAND est une inversion de AND au même titre que NOR est une inversion de NOR.

A	B	AND	NAND	OR	NOR
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

Combinaison de fonctions logiques

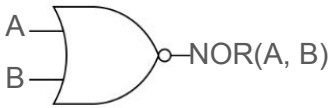
NAND(A, B) = NOT(AND(A, B))

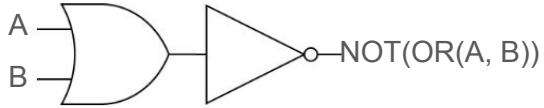
NOR(A, B) = NOT(OR(A, B))

A	B	AND	NAND	OR	NOR
0	0	**0**	1=not(**0**)	**0**	1=not(**0**)
0	1	**0**	1=not(**0**)	**1**	0=not(**1**)
1	0	**0**	1=not(**0**)	**1**	0=not(**1**)
1	1	**1**	0=not(**1**)	**1**	0=not(**1**)

Combinaison de fonctions/portes logiques

Exemple pour NOR(A, B) = NOT(OR(A, B))

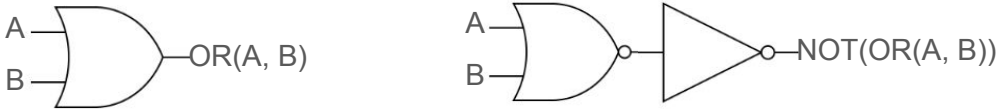




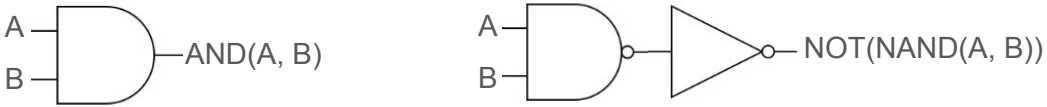
A	B	OR	NOR
0	0	**0**	1=not(**0**)
0	1	**1**	0=not(**1**)
1	0	**1**	0=not(**1**)
1	1	**1**	0=not(**1**)

Combinaison de fonctions/portes logiques

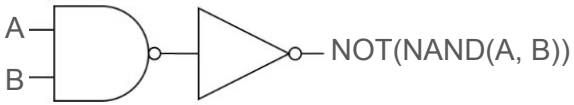
Exemple pour OR(A, B) = NOT(NOR(A, B))

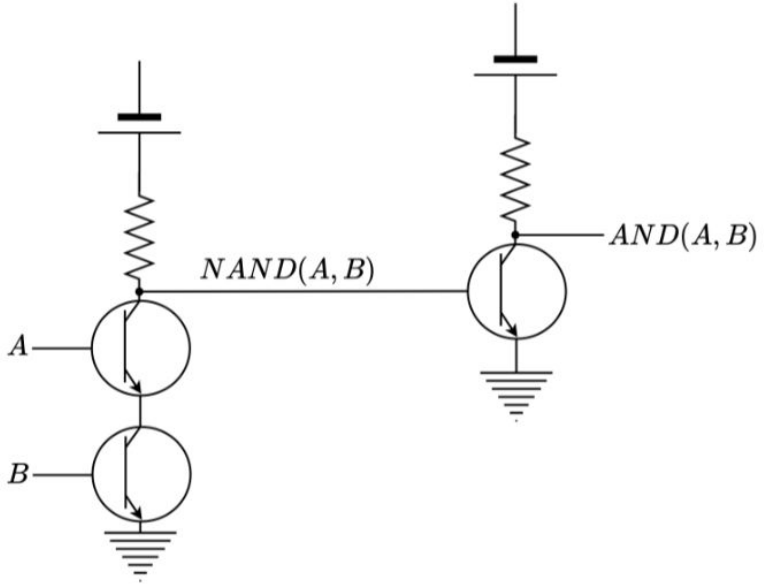


Exemple pour AND(A, B) = NOT(NAND(A, B))

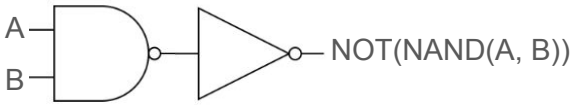


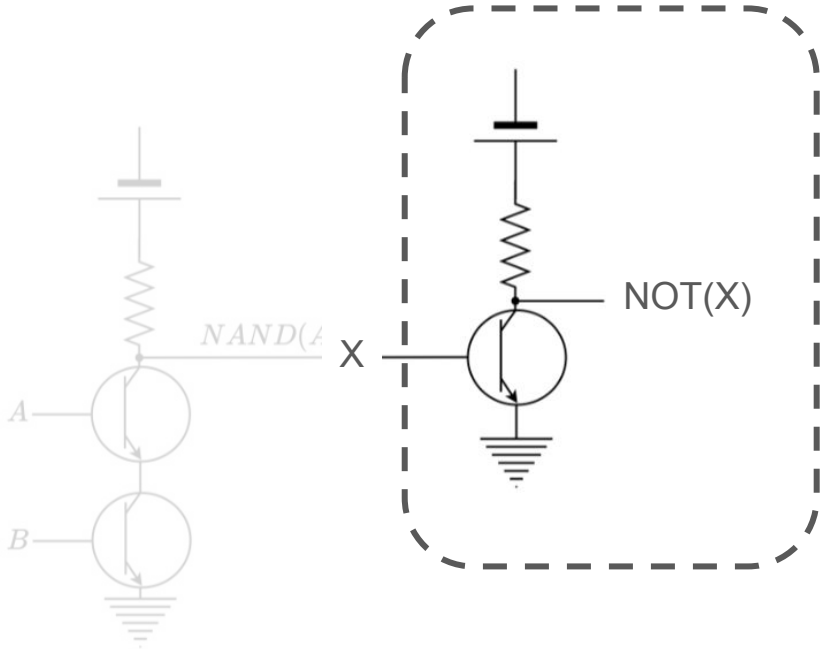
Exemple de porte AND à base de transistors



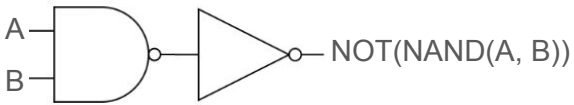


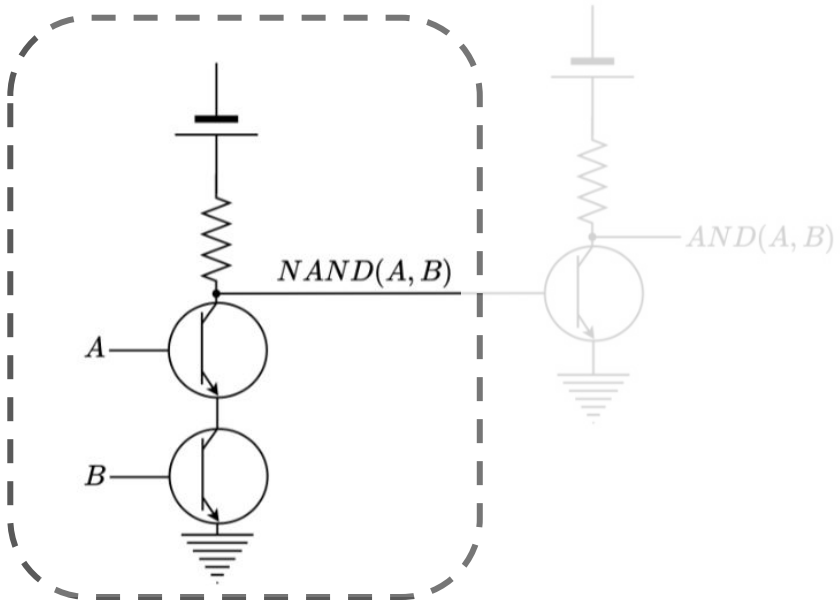
Exemple de porte AND à base de transistors



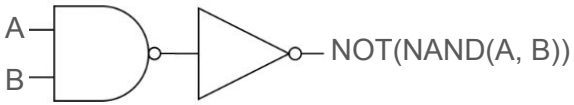


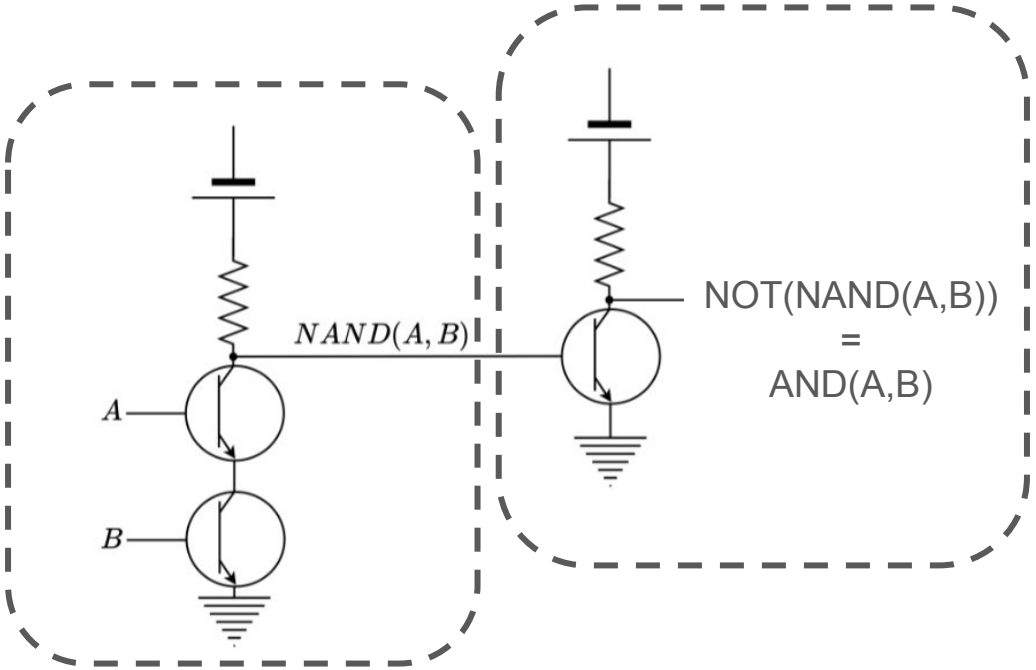
Exemple de porte AND à base de transistors





Exemple de porte AND à base de transistors





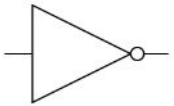
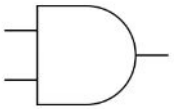
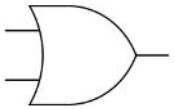
Algèbre de Boole

- Nous avons vu qu'un raisonnement logique pouvait être exprimé :
  - Comme une phrase du langage courant.
  - Comme une table de vérité.
  - Comme un schéma. Cette dernière méthode peut s'implémenter comme un circuit logique dans le monde physique, par exemple grâce à des transistors.

- Afin de simplifier l'expression des raisonnements logiques, nous allons maintenant utiliser une **notation et des règles mathématiques** : l'algèbre de Boole.

Fonctions logiques | **Notation**

`1 = TRUE et 0 = FALSE`

Fonction	Algèbre	Logique	Programmation	Diagrammes logiques
NOT	!	$\neg$ ou $\sim$	!	
AND	$\cdot$	$\wedge$	&& ou *and*	
OR	$+$	$\vee$	|| ou *or*	

Logique booléenne | **Fonctions de base**

- Exemple :

  OR(TRUE, TRUE) = TRUE s'écrit 1 + 1 = 1 en notation algébrique.

- Exemple :

  AND(TRUE, FALSE) = FALSE s'écrit 1 · 0 = 0 en notation algébrique.

Résumé avant d’aller plus loin

- Une fonction logique est **définie** par une table de vérité.
- Une fonction logique **porte sur des propositions** logiques vraies ou fausses.
- Une fonction logique peut être exprimée à l’aide de l’algèbre de Boole.
- Une fonction logique est symbolisée par une **porte logique** sur un schéma de circuit logique.
- Une porte logique peut être **réalisée** au moyen de **transistors**.

Résumé avant d'aller plus loin | **Exemple avec NAND**

Exemple : La fonction logique NAND est définie par sa table de vérité, et peut être représentée de différentes façon, voire même être réalisée à l'aide de transistors.

Notation algébrique

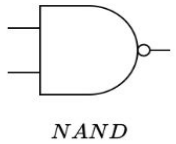
!(AB)

Propositions logiques

Table de vérité

A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0

Symbole de porte logique



NAND

Logique booléenne | **Relations générales**

- Comme on l'a vu, on peut dégager certaines relations générales.
  Par exemple : OR(A, B) = NOT(NOR(A, B))
  Ou encore (équivalent) : A + B = !(!(A + B))

- L'algèbre de Boole (vers 1850) formalise le calcul de propositions binaires.
  Permet de transformer des phrases en expressions algébriques.

- Nécessaire car :
  - Tous les raisonnements ne sont pas aussi simples que celui de l'excès de vitesse.
  - **Simplification** d'expressions en apparence complexes → impact sur le circuit correspondant !
  - On veut pouvoir fournir des preuves formelles.

Logique booléenne | **Priorité des opérations**

- PAS(**rouler trop vite)** ET (**être hors localité**) s'écrit : !A · B

  A B

  La négation est prioritaire sur les autres fonctions logiques.

- Pour le reste, on suit l'ordre des opérations habituel en mathématiques.
  ⇒ AND est prioritaire sur OR.

Logique booléenne | **Règles - Double négation**

- Pour l’exemple, revenons sur la correspondance obtenue par "force brute" :
  OR(A, B) = NOT(NOR(A, B))

  C’est-à-dire : A + B = !(!(A + B))

- À l’évidence, X = !(!X)

- Analogue à la double négation dans la langue française :
  "ce n’est pas irréaliste" signifie "c’est réaliste".

Logique booléenne | **Preuve des règles**

- Une façon de procéder : vérification exhaustive.
- Exemple pour !(x + y) = !x !y

x	**y**	**!(x + y)**	**!x !y**
0	0		
0	1		
1	0		
1	1		

Logique booléenne | **Preuve des règles**

- Une façon de procéder : vérification exhaustive.
- Exemple pour !(x + y) = !x !y

x	**y**	**!(x + y)**	**!x !y**
0	0	!(0 + 0) = !0 = **1**	!0 !0 = 1·1 = **1**
0	1	!(0 + 1) = !1 = **0**	!0 !1 = 0·1 = **0**
1	0	!(1 + 0) = !1 = **0**	!1 !0 = 1·0 = **0**
1	1	!(1 + 1) = !1 = **0**	!0 !0 = 0·0 = **0**

Logique booléenne | **Règles importantes**

Nom	Version AND	Version OR
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$

Il y en a d'autres…

Logique booléenne | **Règles importantes**

Nom	Version AND	Version OR
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributivité	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Idempotence	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + (xy) = x$
Théorème de De Morgan	$!(xy) =!x+!y$	$!(x + y) =!x!y$

Exemple de simplification

XY + X = ?

Nom	Version *AND*	Version *OR*
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributivité	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Idempotence	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + (xy) = x$
Théorème de De Morgan	$!(xy) =!x+!y$	$!(x + y) =!x!y$

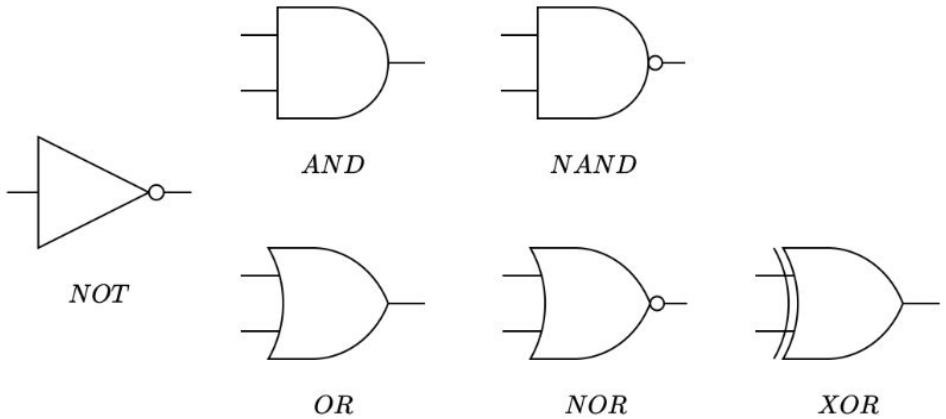
Exemple de simplification

XY + X = X·Y + X·1 = X(Y + 1) = X

Nom	Version *AND*	Version *OR*
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributivité	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Idempotence	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + (xy) = x$
Théorème de De Morgan	$!(xy) =!x+!y$	$!(x + y) =!x!y$

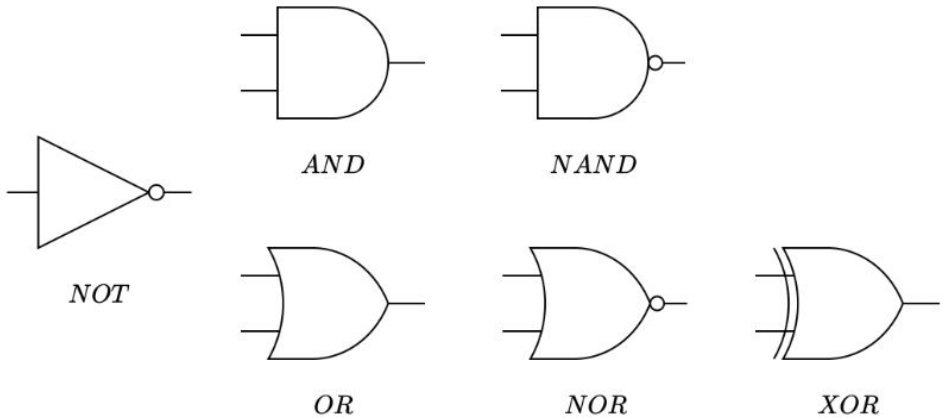
Logique booléenne | **Fonction logique universelle**

- Existe-t-il une fonction qui permette à **elle-seule** de formuler **toutes les autres** ?



Logique booléenne | **Fonction logique universelle**

- Existe-t-il une fonction qui permette à **elle-seule** de formuler **toutes les autres** ?
- Réponse : oui. Il en existe même deux.



Logique booléenne | **Fonction logique universelle**

- Partons des trois fonctions qui nous semblent fondamentales : NOT, AND, OR.

- Voyons si l'on peut réduire ce triplet de fonctions.
  Arbitrairement, commençons par examiner OR:

$$x + y = !!x + !!y = !(!x \cdot !y)$$

Logique booléenne | **Fonction logique universelle**

- Partons des trois fonctions qui nous semblent fondamentales : NOT, AND, OR.

- Voyons si l'on peut réduire ce triplet de fonctions :

$$x + y = !(!x) + !(!y) = !(!x \cdot !y)$$

Double négation — De Morgan

⇒ OR peut être exprimé avec NOT et AND, c'est-à-dire NAND.

Logique booléenne | **Fonction logique universelle**

1) On vient de montrer que x + y = !(!x · !y) = NAND(!x, !y)

2) Quid de la fonction NOT elle-même ?
!x = !(x · x) ≡ NAND(x, x)

Idempotence

3) En combinant 1) et 2) :
x + y = NAND(!x, !y) = NAND(NAND(x, x) , NAND(y, y)).

!x !y

On vient de montrer que OR peut être exprimé **exclusivement** à partir de NAND.

Logique booléenne | **Fonction logique universelle**

- On vient de montrer que OR peut être exprimé **exclusivement** à partir de NAND.
- Faisons pareil avec AND :

$$x \cdot y = !!x \cdot !!y = !(!x + !y)$$

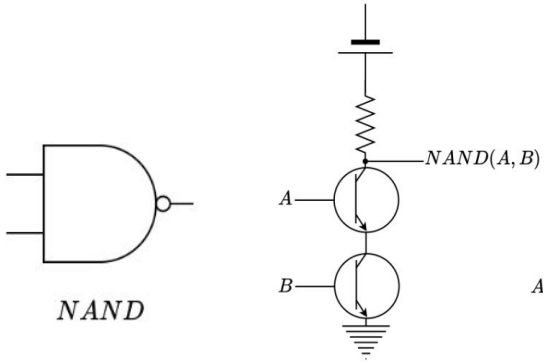
Double négation — de Morgan

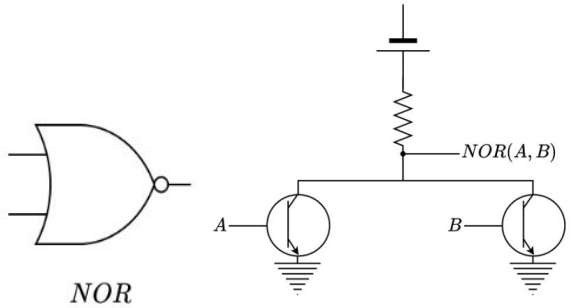
- Autrement dit : AND est exprimable avec NOT et OR.
- Comme on vient de montrer que OR et NOT sont exprimables avec NAND, alors AND est également exprimable avec NAND.

Logique booléenne | **Fonctions logiques universelles**

- NOR est universelle au même titre que NAND (démonstration similaire).

- Il se trouve que ce sont justement les portes les plus faciles à construire avec des transistors.





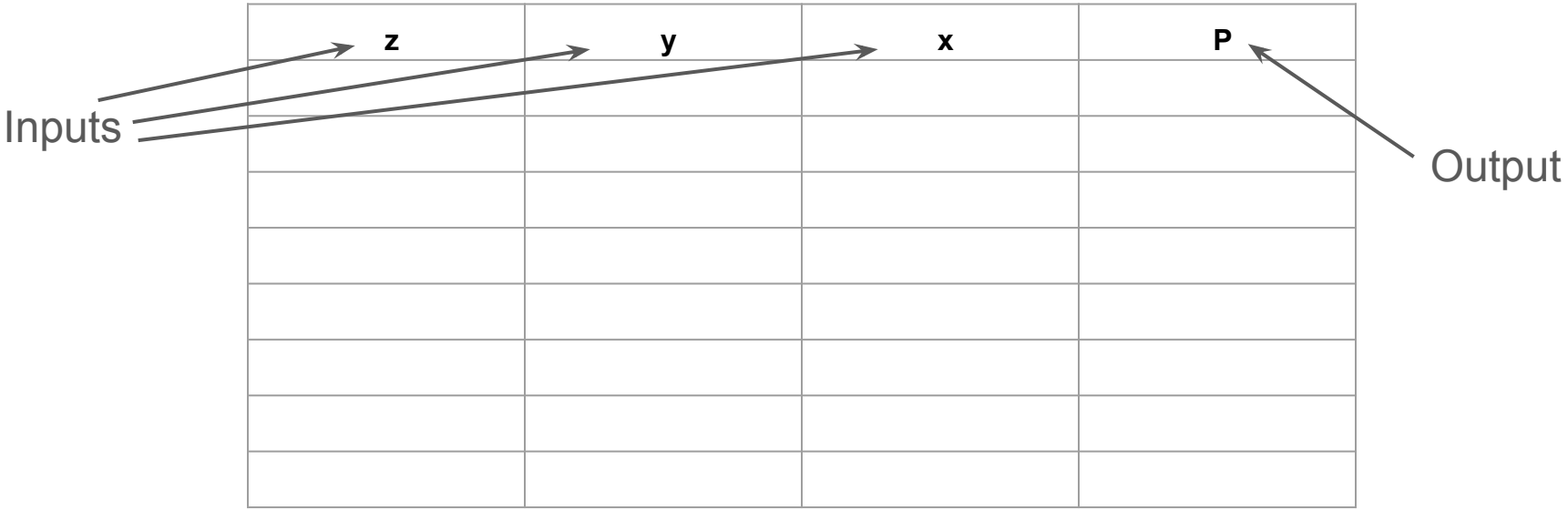
Un exemple non trivial : nombres premiers à 3 bits

- Objectif : obtenir la table de vérité correspondant à la fonction logique P(x,y,z) telle que P(x,y,z) = 1 si et seulement si le nombre "zxy"$_2$ est premier.

- Exemples :
  - P(1,0,0) = 0 car 100_2 = 4 n'est pas premier.
  - P(1,0,1) = 1 car 101_2 = 5 est premier.

Un exemple non trivial : nombres premiers à 3 bits

Objectif : obtenir la table de vérité correspondant à la fonction logique P(x,y,z) telle que P(x,y,z) = 1 si et seulement si le nombre $(zxy)_2$ est premier.



Un exemple non trivial : nombres premiers à 3 bits

Objectif : obtenir la table de vérité correspondant à la fonction logique P(x,y,z) telle que P(x,y,z) = 1 si et seulement si le nombre $(zxy)_2$ est premier.

z	y	x	P

Un exemple non trivial : nombres premiers à 3 bits

Objectif : obtenir la table de vérité correspondant à la fonction logique P(x,y,z) telle que P(x,y,z) = 1 si et seulement si le nombre $(zxy)_2$ est premier.

z	y	x	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Un exemple non trivial : nombres premiers à 3 bits

Maintenant, comment obtenir une expression algébrique pour P, de la forme P = f(z,y,x) ?

z	y	x	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

Condition	z	y	x	P
	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
	1	0	0	0
	1	0	1	1
	1	1	0	0
	1	1	1	1

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

Condition	z	y	x	P
	0	0	0	0
	0	0	1	0
!zy!x	**0**	**1**	**0**	**1**
!zyx	**0**	**1**	**1**	**1**
	1	0	0	0
z!yx	**1**	**0**	**1**	**1**
	1	1	0	0
zyx	**1**	**1**	**1**	**1**

P = !zy!x + !zyx + z!yx + zyx

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

$$P = !zy!x + !zyx + z!yx + zyx$$

Simplifions :

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

$$P = !zy!x + !zyx + z!yx + zyx$$

Simplifions :

$$P = \mathbf{!zy}!x + \mathbf{!zy}x + \mathbf{z}!y\mathbf{x} + \mathbf{zyx}$$

Distributivité AND

$$= \mathbf{!zy}(!x + x) + \mathbf{zx}(!y + y)$$

Complément OR

$$= \mathbf{!zy} + \mathbf{zx}$$

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

$$P = !zy!x + !zyx + z!yx + zyx = \mathbf{!zy + zx}$$

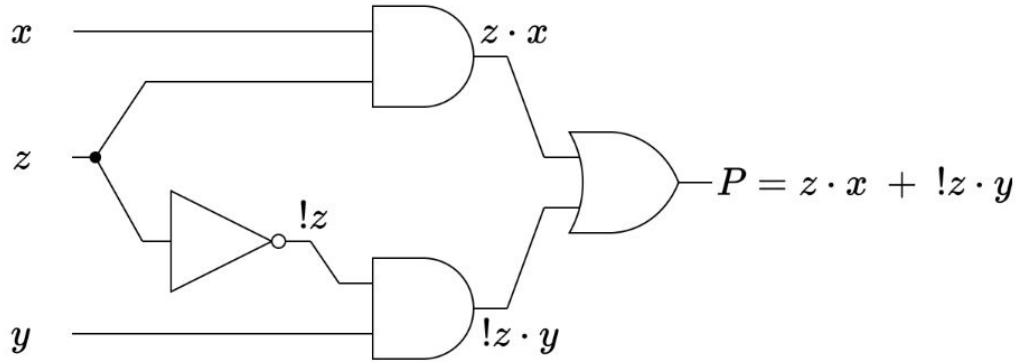
Ce qui correspond au circuit :

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir P = 1 :

$$P = !zy!x + !zyx + z!yx + zyx = \mathbf{!zy + zx}$$

Ce qui correspond au circuit :

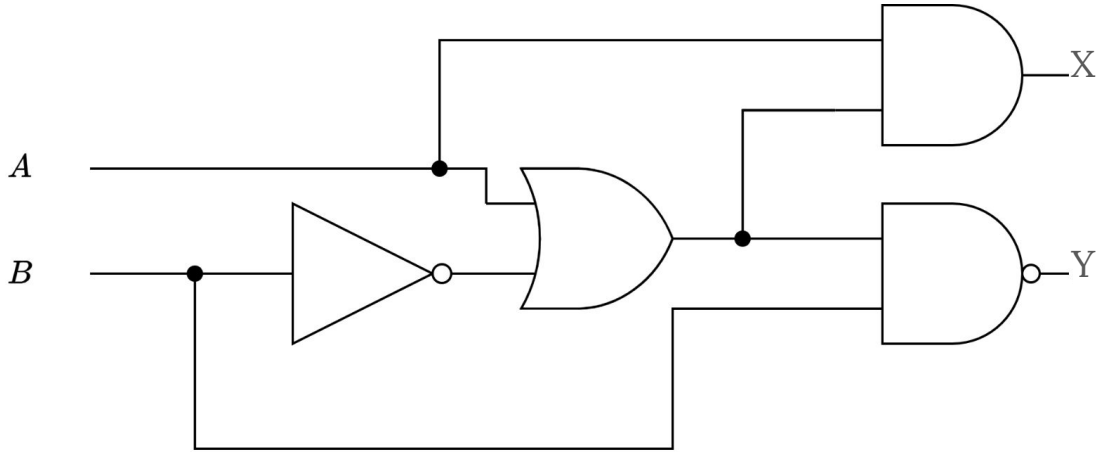


Votamatic - 2 questions

votamatic.unige.ch code JZVB

Votamatic code JZVB (Question 2)

Pour que X = 1 et Y = 1, que doivent valoir A et B ?



Une approche méthodique pour l'expression d'une table

- Précédemment (nombre premier à 3 bits), nous avons listé toutes les façons de produire la valeur 1 en sortie, puis avons simplifié l'expression.
- Ces différentes façons de produire 1 sont combinées avec des OU logiques.
- Méthode : output = expression_1 + expression_2 + … + expression_n.
- Cette méthode se nomme la méthode des **mintermes**.

Mintermes et maxtermes

- Mintermes : lister toutes les façons de produire la valeur 1 en sortie (combinées avec des OU).

- Maxtermes : lister toutes les façons de ne pas produire la valeur 1 en sortie (combinées avec des OU), puis en effectuer la négation.

Exemple pour les maxtermes | **Nombre premier à 3 bits**

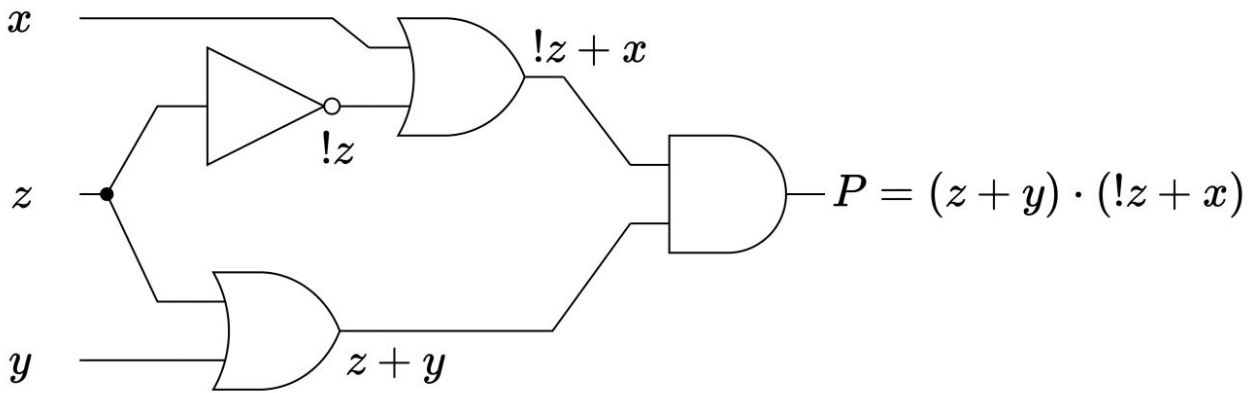
- Les entrées qui donnent P = 0 sont 000, 001, 100 et 110.
- Cela correspond à l'expression !P = !z!y!x + !z!yx + z!y!x + zy!x

Exemple pour les maxtermes | **Nombre premier à 3 bits**

- Les entrées qui donnent P = 0 sont 000, 001, 100 et 110.
- Cela correspond à l'expression !P = !z!y!x + !z!yx + z!y!x + zy!x
- Simplification : !P = !z!y(!x + x) + z!x(!y + y) = !z!y + z!x
- Négation : P = !!P = ! (!z!y + z!x) = !(!z!y) · !(z!x) = (!!z+!!y) · (!z+!!x)
  = (z + y) · (!z + x)

Exemple pour les maxtermes | **Nombre premier à 3 bits**

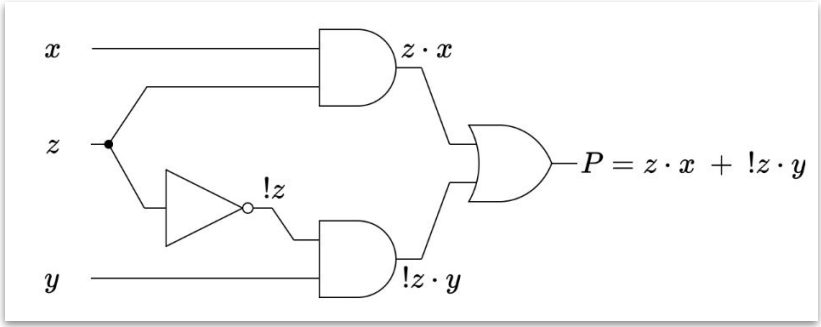
Finalement P = (z + y) · (!z + x)



Note sur l'équivalence des deux derniers circuits

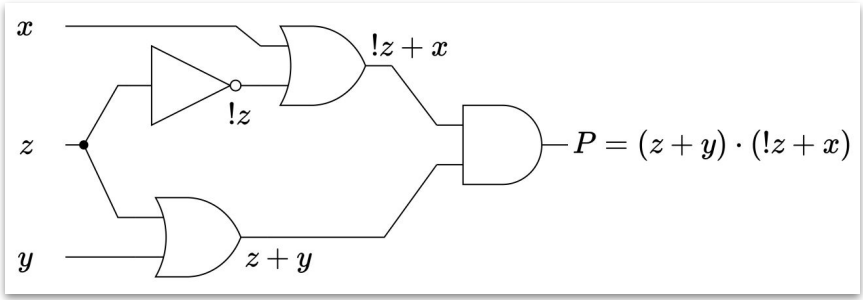
Circuit obtenu avec les mintermes :

P_m = zx + !zy



Circuit obtenu avec les maxtermes :

P_M = (z + y) · (!z + x)



Note sur l’équivalence des deux derniers circuits

- Si nous n’avons pas fait d’erreur : $P_m = P_M$
- $P_M = (z + y) \cdot (!z + x) = z!z + zx + !zy + xy = zx + !zy + \mathbf{xy} = P_m + \mathbf{xy}$

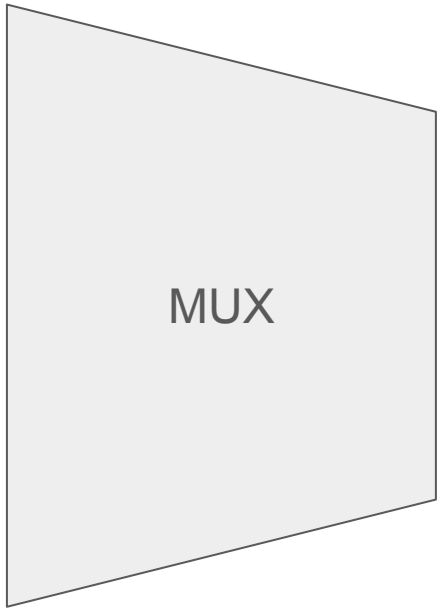
Note sur l'équivalence des deux derniers circuits

- Si nous n'avons pas fait d'erreur : $P_m = P_M$
- $P_M = (z + y) \cdot (!z + x) = z!z + zx + !zy + xy = zx + !zy + \mathbf{xy} = P_m + \mathbf{xy}$
- Deux possibilités :
  - Si xy = 0, on a l'équivalence.
  - Si xy = 1, alors x = 1 et y = 1. Dans ce cas, $P_m = zx + !zy = z + !z = 1$ et, de toute manière, $P_m + xy = P_m$.
- Note : en fait, zx + !zy + xy = zx + !zy correspond au théorème du consensus, ou théorème de la redondance.

Circuits combinatoires

- Circuits combinant plusieurs portes logiques (déjà vu !)
- Limite entre porte logique et circuit logique dépend des auteurs.
- Nous allons ici voir deux circuits combinatoires importants :
  - Le multiplexeur
  - L’additionneur

Le multiplexeur



Multiplexeur à 2 voies

- Il est courant de vouloir exprimer des **conditions** du type : "*Si S est faux, alors la valeur de sortie est A, sinon la valeur de sortie est B.*"

- Permet d'effectuer un **choix** entre A et B grâce à S.

- Équivalent électronique du if/else de programmation :

```
if S == 0:
    Out = A
else:
    Out = B
```

Multiplexeur à 2 voies

- Il est courant de vouloir exprimer des conditions du type : "*Si S est faux, alors la valeur de sortie est A, sinon la valeur de sortie est B.*"

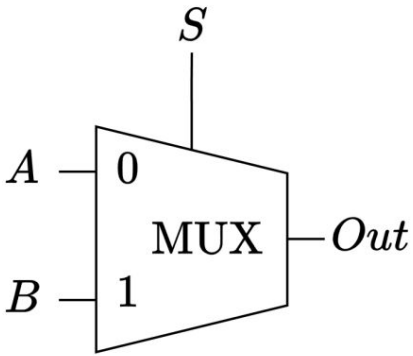
- Permet d'effectuer un choix entre A et B grâce à S.

- Bien entendu, on peut exprimer cela au travers d'une table de vérité.
  Trois entrées : S, A, B.
  Une sortie : Out.

- Vocabulaire : S est la **ligne de contrôle**, A et B sont les **lignes de données**.

Multiplexeur à 2 voies

- On décide que S=0 correspond à la sélection de A et S=1 correspond à la sélection de B.
- On obtient en sortie une copie de l'entrée sélectionnée.
- Symbole :

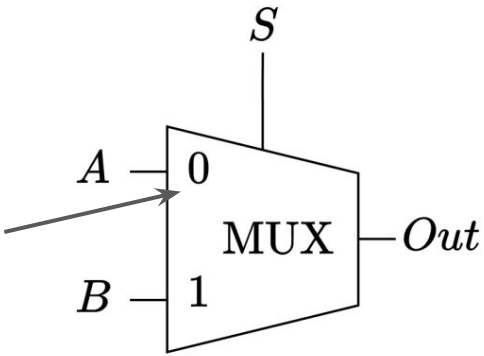


S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à 2 voies

- On décide que S=0 correspond à la sélection de A et S=1 correspond à la sélection de B.
- On obtient en sortie une copie de l'entrée sélectionnée.
- Symbole :

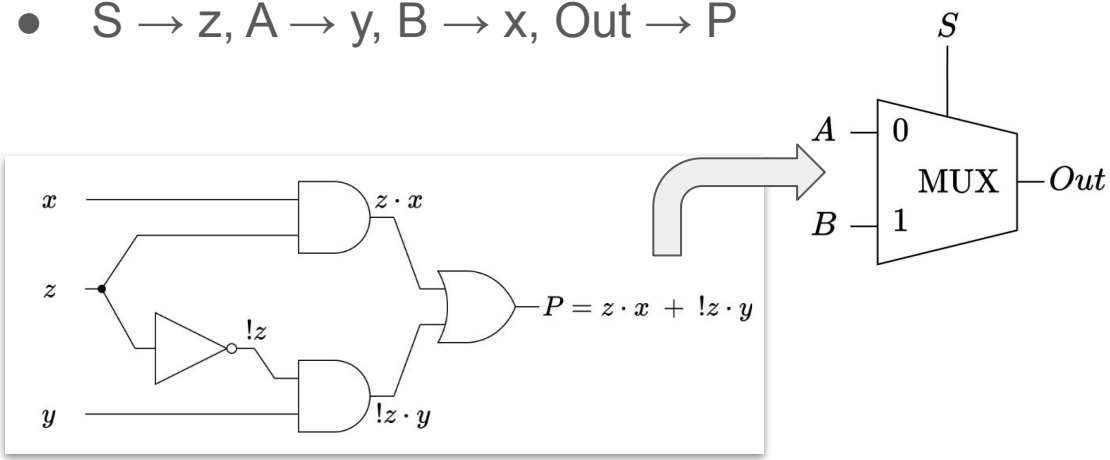
Ne veut pas dire que A = 0



S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à 2 voies

- Cette table de vérité est équivalente à celle du problème du nombre premier à 3 bits !
- S → z, A → y, B → x, Out → P

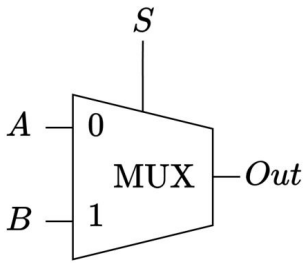


S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à 2 voies

- Cette table de vérité est équivalente à celle du problème du nombre premier à 3 bits !

- Out = SB + !SA

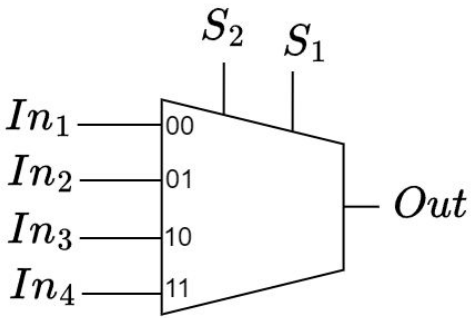


S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à *k* voies

Exemple pour *k* = 4 lignes de données

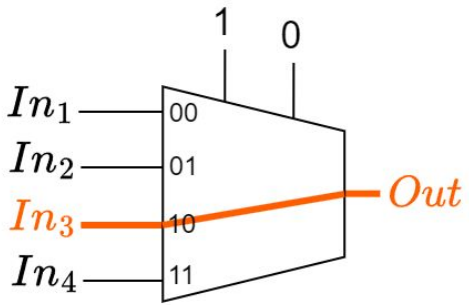
S	Out
00	In_1
01	In_2
10	In_3
11	In_4



Multiplexeur à *k* voies

Exemple pour *k* = 4 lignes de données.

S	Out
00	In_1
01	In_2
10	In_3
11	In_4



Multiplexeur à *k* voies | **Taille de la ligne de contrôle**

- Sélectionne l'une des *k* entrées, selon le signal de contrôle *S*.

- Question : quelle doit être la taille (nombre de bits) de *S* afin de pouvoir sélectionner n'importe laquelle des *k* entrées ?

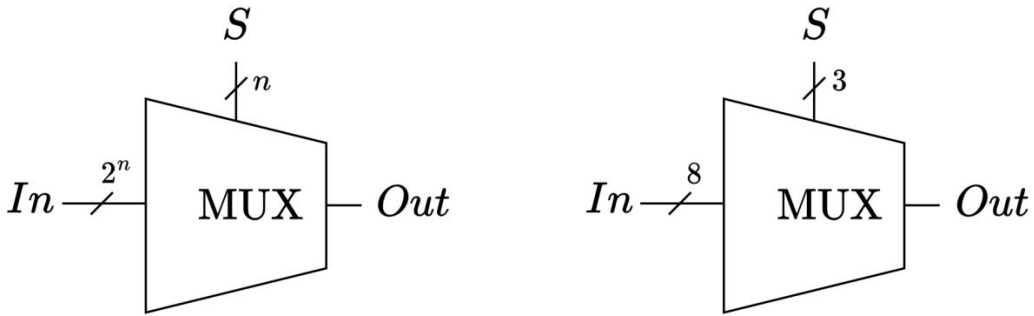
Multiplexeur à *k* voies | **Taille de la ligne de contrôle**

- Sélectionne l'une des *k* entrées, selon le signal de contrôle *S*.
- Question : quelle doit être la taille (nombre de bits) de *S* afin de pouvoir sélectionner n'importe laquelle des *k* entrées ?
- Réponse : il faut $\log_2(k)$ bits. Autrement dit, si la ligne de contrôle comprend *n* bits, on peut sélectionner une entrée parmi $k = 2^n$.
- Exemple : avec 3 bits pour la sélection, on peut avoir 8 voies.

Multiplexeur à *k* voies | **Symboles**

- Si la ligne de contrôle comprend *n* bits, on peut sélectionner une entrée parmi $k = 2^n$.

- Plutôt que de dessiner toutes les voies sur les schémas, on indique le nombre de bits concernés avec une barre oblique. Exemples :

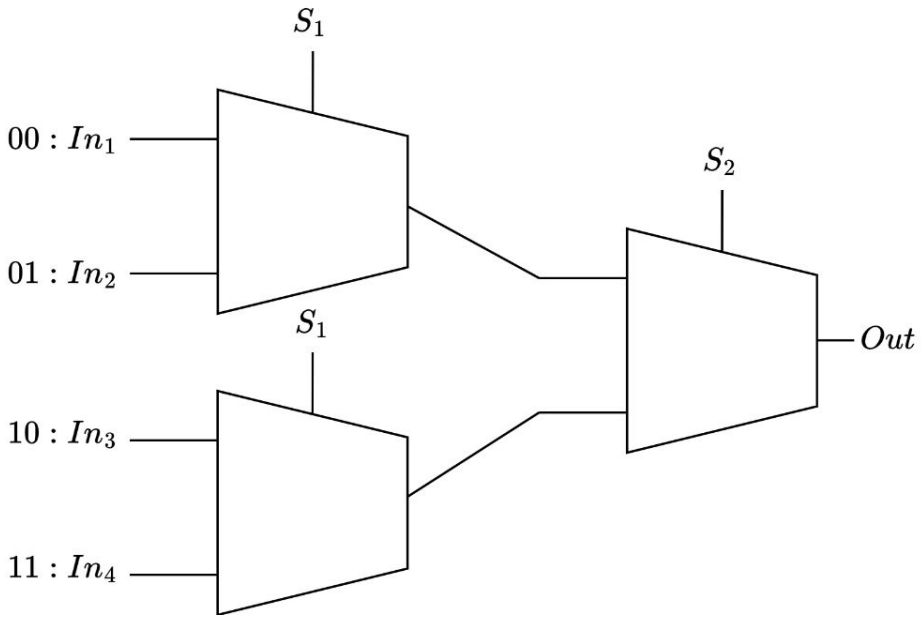


Multiplexeur à *k* voies | **Parcours du signal**

- Question : combien de portes logiques le signal doit-il traverser pour générer un output ?

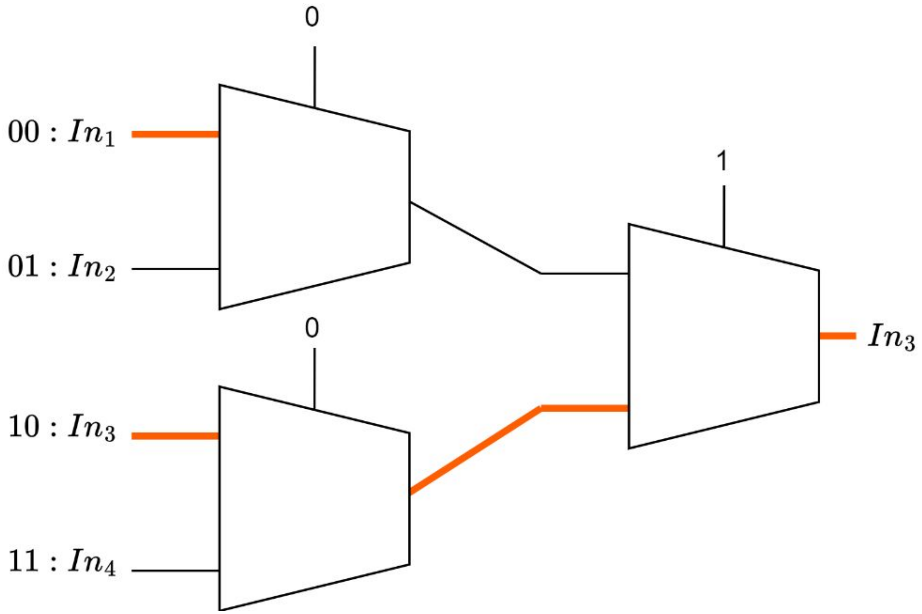
Multiplexeur à *k* voies | **Parcours du signal - Intuition**

- Question : combien de portes logiques le signal doit-il traverser pour générer un output ?



Multiplexeur à *k* voies | **Parcours du signal**

- Question : combien de portes logiques le signal doit-il traverser pour générer un output ?



Multiplexeur à *k* voies | **Parcours du signal**

- Question : combien de portes logiques le signal doit-il traverser pour générer un output ?

- Réponse : ce nombre est proportionnel au nombre de bits de *S*.

- Raison : multiplexeur à *k* voies peut être vu comme un arbre de multiplexeurs à 2 voies. Le nombre d'étages de cet arbre est de $\log_2(k)$.

Multiplexeur à *k* voies | **Parcours du signal**

- Le signal doit traverser un nombre de portes proportionnel à log(k).

- Ce fait trouvera son importance plus tard dans le cours, car des multiplexeurs sont utilisés pour sélectionner des données dans la mémoire de l'ordinateur.

  À savoir : la vitesse d'accès aux éléments de la mémoire est d'autant plus lente que les adresses sont longues.

Multiplexeur à *k* voies **(non traité en cours)**

- Question : quel est le nombre de lignes de la table de vérité d'un multiplexeur à 3 voies ?

(non traité en cours)

Multiplexeur à *k* voies **(non traité en cours)**

- Question : quelle est le nombre de lignes de la table de vérité d'un multiplexeur à 3 voies ?

- Début de la table de vérité pour *k* = 4 voies :

S_1	S_2	In_1	In_2	In_3	In_4	Out
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0

. . .

Multiplexeur à *k* voies **(non traité en cours)**

- Question : quelle est le nombre de lignes de la table de vérité d'un multiplexeur à 3 voies ?

- Réponse : il y a $k + \log_2(k)$ colonnes, donc $2^{k + \log2(k)} = k2^k$

- Cependant, le nombre de portes logiques que le signal doit traverser est proportionnel au nombre de bits de *S*.

. . .

L’additionneur

Le circuit additionneur

- Objectif : prendre deux mots (états binaires) en input, et obtenir un mot en output qui représente la somme des inputs.

- Inputs :
  - Bits du nombre a : a_0, a_1, … ,a_{n-1}
  - Bits du nombre b : b_0, b_1, … ,b_{n-1}

- Outputs :
  - Bits de la somme s_0, s_1, … ,s_{n-1}
  - Bit de la retenue R.

Rappel sur l'addition binaire

- Exemple de $011001_2 + 110011_2$

0^1	0^1	1	1	0^1	0^1	1
0	1	1	0	0	1	1
1	**0**	**0**	**1**	**1**	**0**	**0**

- Difficulté : la retenue

Addition de deux bits

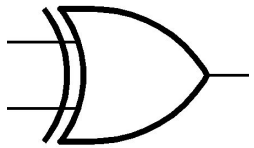
- Quatre cas possibles pour **une** colonne donnée.
  - `0 + 0 =  0`
  - `0 + 1 =  1`
  - `1 + 0 =  1`
  - `1 + 1 = 10`

Pas présent sur la colonne concernée

- La table de vérité pour l'addition sur une colonne correspond donc au XOR.

Addition de deux bits

- La table de vérité pour l'addition sur une colonne correspond donc au XOR.
- Ne prend pas en compte la retenue !
- XOR(a,b) = a ⊕ b

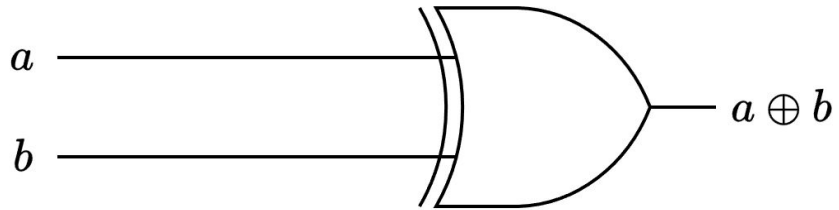


a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

Addition de deux bits

- Ne prend pas en compte la retenue !
- Comment modifier ce circuit pour également avoir un output de retenue ?

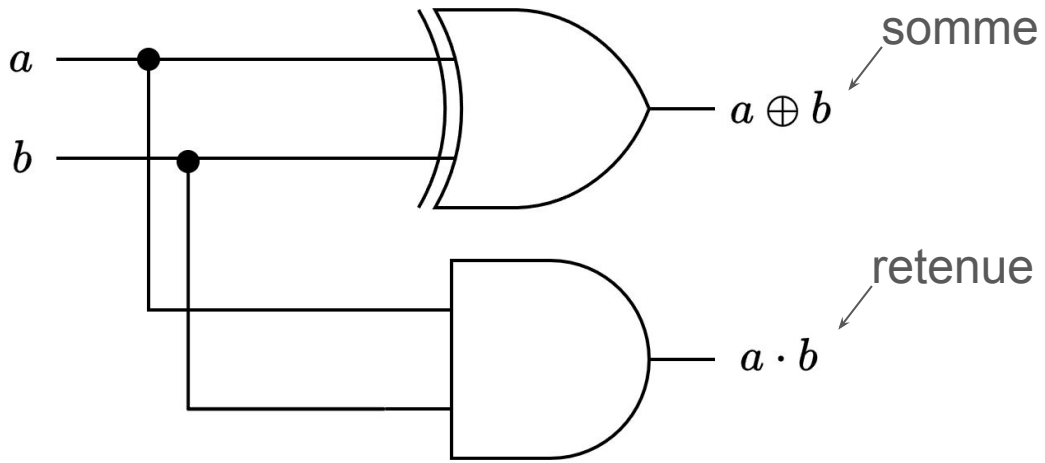
a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0



Addition de deux bits | **Prise en compte de la retenue**

- Pour le cas où *a* et *b* valent 1, une retenue existe !
- Circuit souvent nommé "demi-additionneur".
- Reste à la transmettre à la colonne de gauche, et récupérer celle de droite.

a	b	a ⊕ b	ab
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

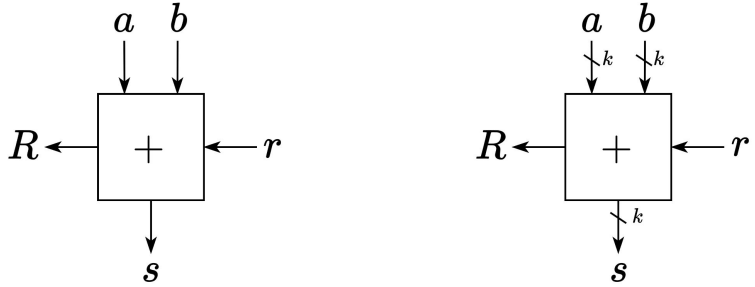


Additionneur | **Cas général (additionneur complet)**

- En plus de a_i et b_i, Le circuit associé à chaque colonne reçoit en input le reste de la colonne de droite, $r_i = R_{i-1}$.

- Le circuit de chaque colonne output la somme s_i et transmet le reste R_i à la colonne de gauche.

- Symbole de l'additionneur complet :

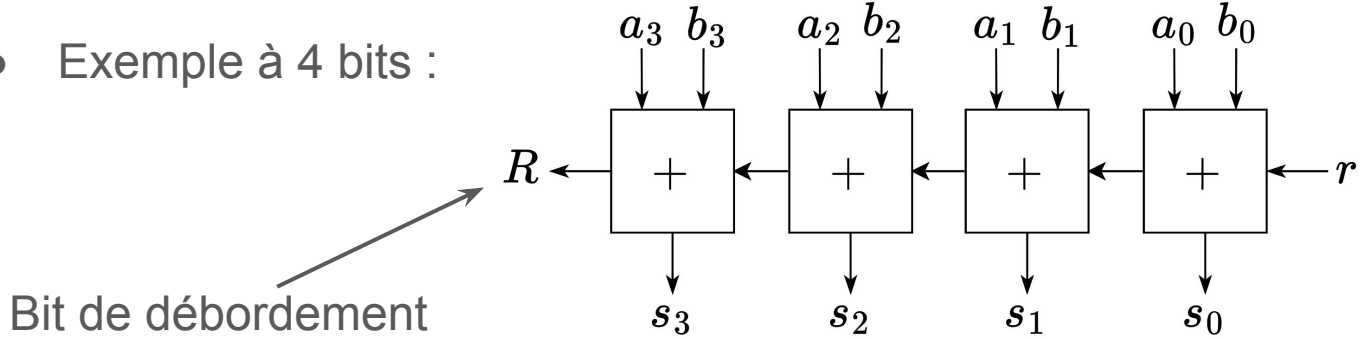


Additionneur | **Cas général**

- En plus de a_i et b_i, Le circuit associé à chaque colonne reçoit en input le reste de la colonne de droite, $r_i = R_{i-1}$.

- Le circuit de chaque colonne output la somme s_i et transmet le reste R_i à la colonne de gauche.

- Exemple à 4 bits :



Additionneur | **Table de vérité pour le cas général 1 bit**

r	a	b	s	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

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour R :
  - Soit *a* et *b* valent 1 tous les deux, peu importe *r* ;
  - Soit *r* vaut 1 et uniquement l'un des bits *a* et *b* vaut 1.

r	a	b	s	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**

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour R :
  - Soit *a* et *b* valent 1 tous les deux, peu importe *r* ;
  - Soit *r* vaut 1 et uniquement l'un des bits *a* et *b* vaut 1.

$\Rightarrow R = ab + r(a \oplus b)$

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour s :
  - Soit r vaut 0 et uniquement l'un des deux bits a et b vaut 1 ;
  - Soit r vaut 1 et a = b.

r	a	b	s	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

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour s :
  - Soit r vaut 0 et uniquement l'un des deux bits a et b vaut 1 ;
  - Soit r vaut 1 et a = b.

S'exprime a ⊕ b = 1

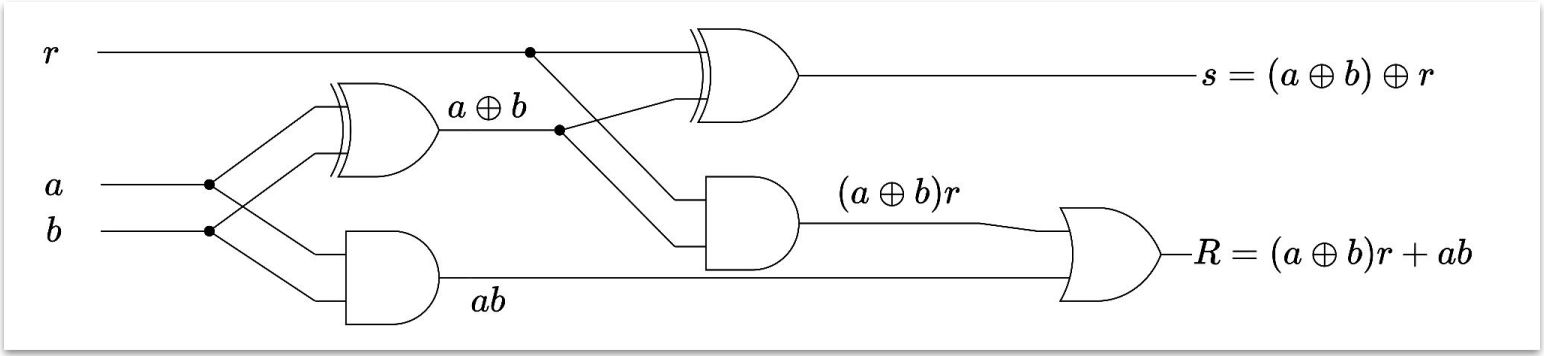
S'exprime a ⊕ b = 0

⇒ s = !r(a ⊕ b) + r!(a ⊕ b).

Mais comme x ⊕ y ≡ x!y + !xy, on remarque finalement que :
s = r ⊕ (a ⊕ b)

Additionneur | **Table de vérité pour le cas général**

- R = ab + r(a ⊕ b)
- s = r ⊕ (a ⊕ b)
- Additionneur complet :



Soustracteur | **Rappel sur le complément à 2**

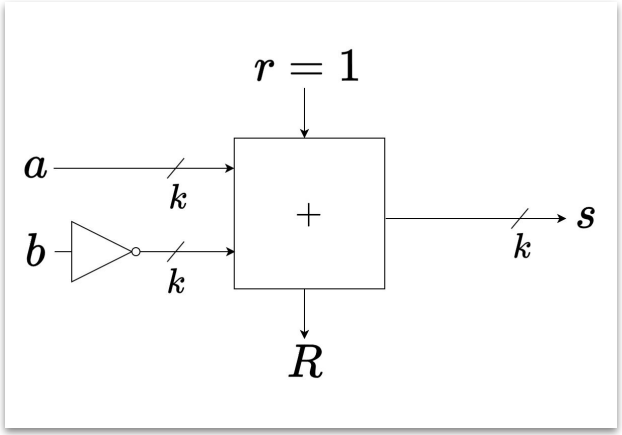
- a - b = a + (-b).
- Profite des avantages du complément à 2 !
- Le codage de (-a) est obtenu comme flip(a) + 1.

Soustracteur

- a - b = a + (-b) = a + flip(b) + 1

Fourni au circuit comme un "reste initial"

- Soustracteur :

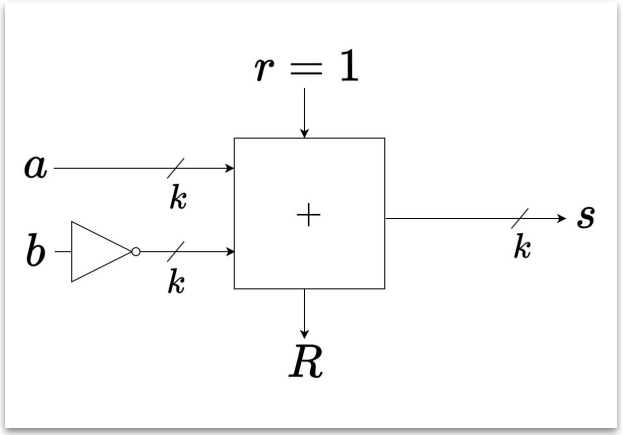


Soustracteur

- a - b = a + (-b) = a + flip(b) + 1

Bit à bit, revient à !b

- Soustracteur :



Note sur les optimisations

- Dans notre additionneur complet, chaque additionneur 1-bit doit attendre le résultat de son voisin avant d'effectuer son calcul, car il a besoin de la retenue.

- Dans les ordinateurs modernes, les additionneurs utilisent des techniques avancées pour anticiper les retenus et gagner du temps.

Logique séquentielle et éléments de mémoire

- Après avoir discuté de l'architecture des ordinateurs, nous reviendrons sur la façon de réaliser des circuits capables de **mémoriser des valeurs** (après la semaine 7).

- Cela se fera grâce à des circuits dits "**séquentiels**", où une notion de temps intervient.