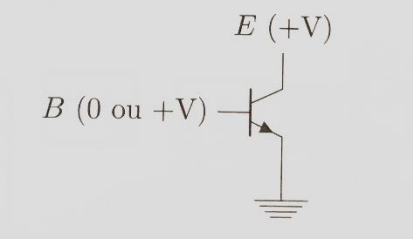
Circuit et logique booléenne

Portes logiques

Les circuits d'un ordinateur (mémoire, microprocesseur, etc...) manipulent uniquement des chiffres binaires 0 et 1, qui en interne, sont simplement représentés par des tensions électriques. Ainsi, le chiffre 0 est représenté par une tension basse (proche de 0 Volt) et le chiffre 1 par une tension haute (que l'on notera +V volts, car cette tension varie selon les circuits).

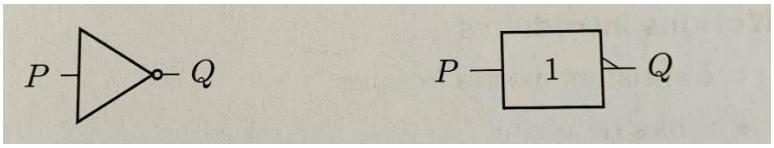
Les opérateurs (logiques ou arithmétiques) sur ces nombres binaires sont construits à partir de *circuits électroniques* dont les briques élémentaires seront appelées *transistors*. Les transistors que l'on trouve dans les circuits des ordinateurs se comportent comme des interrupteurs qui laissent ou non passer un courant électrique selon le mode du *tout ou rien*, comme représenté graphiquement de la manière suivante.



Dans ce schéma, la commande de l'interrupteur est jouée par la broche B. Lorsqu'elle est sous tension haute, la broche B laisse passer le courant entre la broche E (sous tension haute) et la masse (le sens du courant est indiqué par la petite flèche), ce qui a pour effet de passer E sous la tension basse (en pratique, il y a des résistances placées au bon endroits pour ne pas avoir de courts-circuits). Inversement, quand B est sous tension basse la broche E reste sous tension haute.

Ce simple transistor permet de réaliser une opération élémentaire appelée *porte logique* NON (appelée NOT en anglais). Une porte logique est une fonction qui prend un ou plusieurs *bits* en entrée et qui produit un *bit* en sortie.

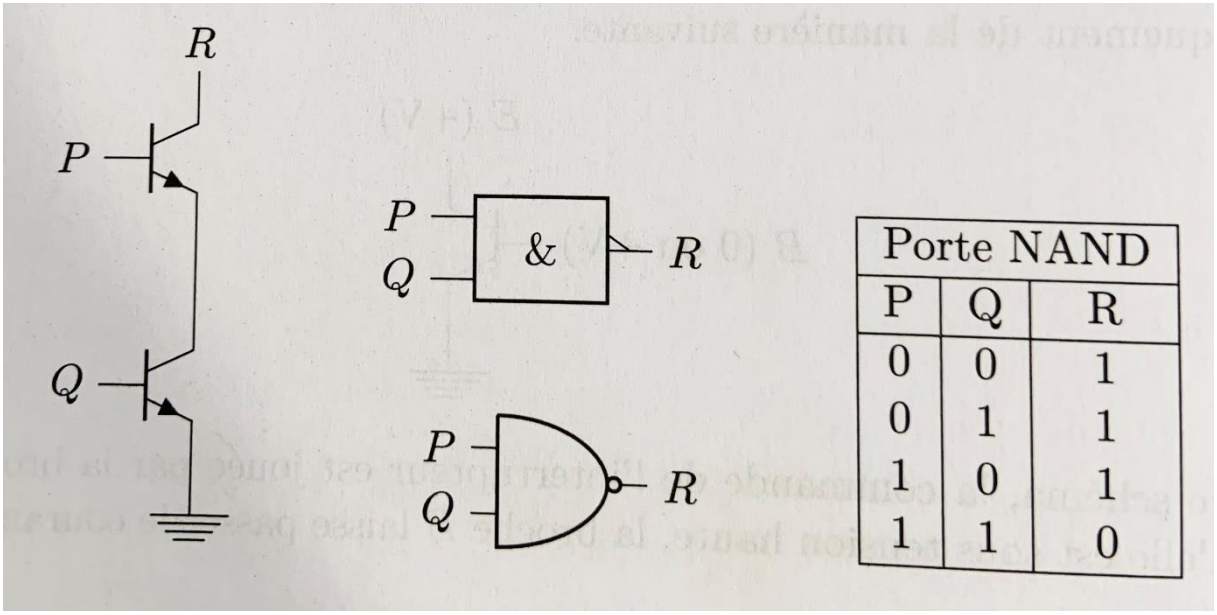
La porte NOT implantée par un transistor est la plus simple de toutes les portes. Elle n'a qu'un seul *bit* en entrée (P) et sa sortie(Q) vaut 0 quand l'entrée vaut 1 et inversement elle vaut 1 quand son entrée est à 0. Graphiquement, on représente la porte NOT comme dans le schéma ci-dessous, avec à gauche la notation américaine et à droite la notation européenne.



Pour représenter le calcul réalisé par une porte logique, on utilise une *table logique* qui relie les valeurs des entrées à la valeur du résultat. La table logique de la porte NOT est donnée ci-dessous.

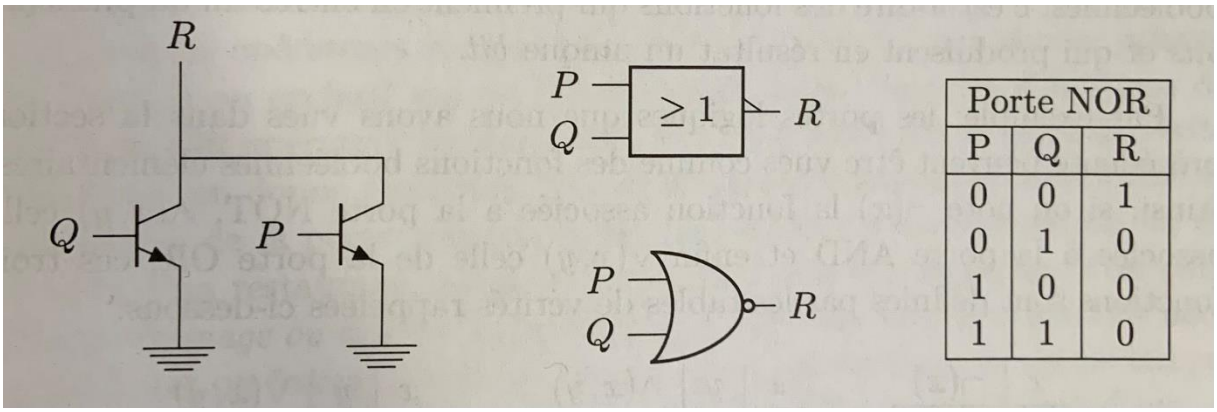
Porte NOT	
P	Q
0	1
1	0

On peut fabriquer d'autres portes logiques en combinant plusieurs transistors. Par exemple, en combinant deux transistors en série comme ci-dessous (schéma de gauche) on peut fabriquer la porte NON ET (appelée NAND en anglais) qui, pour deux entrées P et Q, produit un résultat R dont le calcul est donné par la table de vérité à droite. On donne également ci-dessous les schémas européen (en haut) et américain (en bas) pour cette porte.



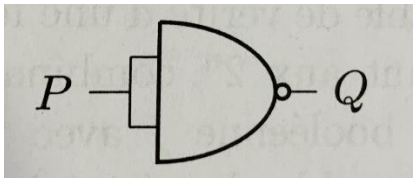
Porte NAND		
P	Q	R
0	0	1
0	1	1
1	0	1
1	1	0

De la même manière, en combinant deux transistors en parallèle, on obtient la porte NON OU (NOR en anglais) donnée ci-dessous (avec les schémas pour la représenter et sa table de vérité).



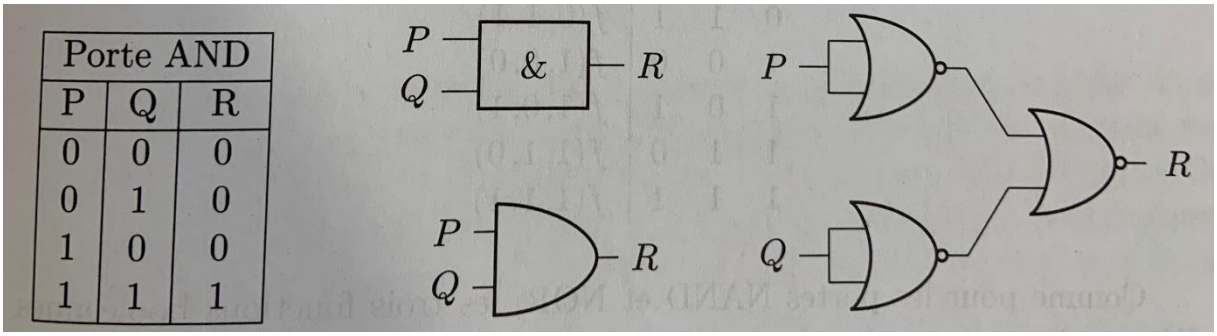
Porte NOR		
P	Q	R
0	0	1
0	1	0
1	0	0
1	1	0

Les portes NAND et NOR sous fondamentales dans les circuits électroniques car elles sont *complètes*, c'est-à-dire que n'importe quel circuit peut être conçu en utilisant *uniquement* ces deux portes. Par exemple, la porte NOT peut être fabriquée à partir d'une porte NAND en reliant les deux entrées de cette porte, comme ci-dessous.



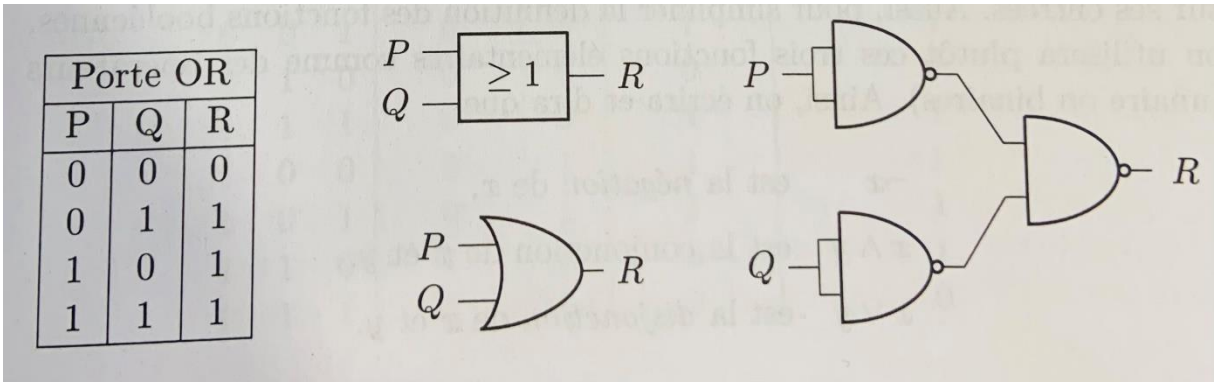
Une autre porte logique très importante est la porte ET (AND en anglais). Elle peut aussi être construite avec plusieurs portes NOR. Voici ci-dessous la table de vérité de la porte ET, ses représentations symboliques américaine (en bas) et européenne (en haut) ainsi qu'un schéma de construction avec des portes NOR.

Porte AND		
P	Q	R
0	0	0
0	1	0
1	0	0
1	1	1



De la même manière, la porte OU (OR en anglais) peut aussi être construite avec plusieurs portes NAND.

Porte OR		
P	Q	R
0	0	0
0	1	1
1	0	1
1	1	1



Fonctions booléennes

Certains circuits électroniques peuvent être décrits comme des fonctions booléennes, c'est-à-dire des fonctions qui prennent en entrée un ou plusieurs *bits* et qui produisent en résultat un unique *bit*.

Par exemple, les portes logiques que nous avons vues dans la section précédente peuvent être vues comme des fonctions booléennes élémentaires.

Les trois fonctions (NOT, AND, OR) sont définies par les tables de vérité ci-dessous :

x	$\neg(x)$
0	1
1	0

x	y	$\wedge(x,y)$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$\vee(x,y)$
0	0	0
0	1	1
1	0	1
1	1	1

Plus généralement, la table de vérité d'une fonction avec *n* bits en entrée aura 2^n lignes correspondant aux 2^n combinaisons possibles des entrées. Par exemple, une fonction booléenne *f* avec trois entrées *x, y, z* sera entièrement définie par une table de vérité à $2^3 = 8$ lignes, de la forme suivante.

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

Pour simplifier la définition des fonctions booléennes, on utilisera plutôt les trois fonctions élémentaires comme des opérateurs. Ainsi on écrira et dira que :

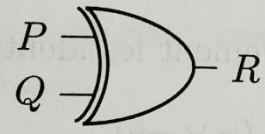
$\neg x$ est la *négation* de x,

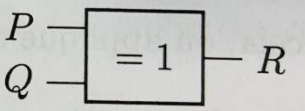
$x \wedge y$ est la conjonction de x et y,

$x \vee y$ est la *disjonction* de x et y.

Parmi les opérateurs élémentaires que nous n'avons pas encore présentés figure le ***ou exclusif*** qui est fréquemment utilisé dans les définitions de fonctions. Il correspond au *ou* de la carte d'un restaurant, quand il est indiqué dans un menu que l'on peut prendre *fromage* ou *dessert*. Comme chacun sait, cela veut dire que l'on peut prendre soit un fromage, soit un dessert, mais pas les deux en même temps.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0





Pour calculer la table de vérité associée à une fonction *f*, il suffit de procéder comme pour un calcul arithmétique ordinaire en calculant les résultats pour toutes les sous-expressions, en commençant par les calculs en profondeur puis en remontant.

Voici une fonction f :

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

La table de vérité associée à une fonction *f :*

x	y	z	$(x \wedge y)$	$\neg y$	$(\neg y \vee z)$	$(x \wedge y) \oplus (\neg y \vee z)$
0	0	0	0	1	1	1
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Une expression booléenne peut aussi être manipulée comme n'importe quelle expression arithmétique. Pour cela, le calcul avec les opérateurs booléens obéit à quelques identités élémentaires dont certaines sont données dans le tableau ci-dessous.

involutif :	$\neg(\neg x) = x$	
neutre :	$1 \wedge x = x$	$0 \vee x = x$
absorbant :	$0 \wedge x = 0$	$1 \vee x = 1$
idempotence :	$x \wedge x = x$	$x \vee x = x$
complément :	$x \wedge \neg x = 0$	$x \vee \neg x = 1$
commutativité :	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
associativité :	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
distributivité :	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
De Morgan :	$\neg(x \wedge y) = \neg x \vee \neg y$	$\neg(x \vee y) = \neg x \wedge \neg y$

Par exemple, en utilisant ces lois, on peut montrer l'égalité suivante.

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

Pour cela, on applique successivement les identités indiquées à droite.

$\neg(y \wedge (x \vee \neg y))$	$=$	$\neg y \vee \neg(x \vee \neg y)$	De Morgan
	$=$	$\neg y \vee (\neg x \wedge y)$	De Morgan et involutif
	$=$	$(\neg y \vee \neg x) \wedge (\neg y \vee y)$	distributivité
	$=$	$(\neg y \vee \neg x) \wedge 1$	complément
	$=$	$(\neg y \vee \neg x)$	neutre
	$=$	$(\neg x \vee \neg y)$	commutativité