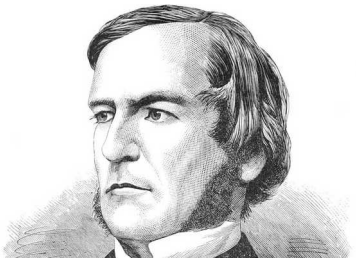
DU TRANSISTOR AU MICROPROCESSEUR



Le fonctionnement des ordinateurs est basé sur la logique dite ***booléenne***.

On appelle « logique à deux états », ou logique *booléenne*, un système d'opérations qu'on peut faire sur des objets qui ne peuvent prendre que deux valeurs, appelées **FAUX / VRAI** ou **0 / 1**. Elle fut lancée en 1854 par le mathématicien britannique George Boole.

Grâce à l'évolution technologique, tubes à vide puis transistors, l'algèbre de Boole trouve aujourd'hui de nombreuses applications en informatique et dans la conception des circuits électroniques.

I. IMPLEMENTATION PHYSIQUE DE LA LOGIQUE BOOLEENNE

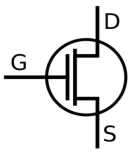
Un transistor (et anciennement un tube à vide), est un dispositif électronique qui ne peut prendre que deux ***états*** :

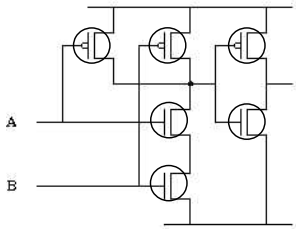
- soit laisser passer le courant (état « haut », noté 1)
- soit ne pas le laisser passer (état « bas », noté 0).

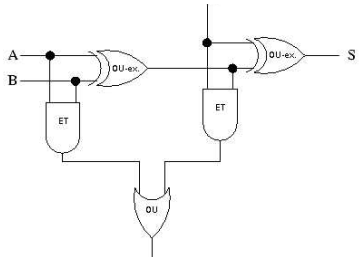
Le transistor est donc un petit interrupteur électronique ; si on branche plusieurs transistors ensemble, alors le comportement de l'ensemble du montage dépend de l'état de chaque transistor (bloquant ou passant).
En combinant un ensemble de transistors, on peut ainsi réaliser des circuits électroniques appelés ***portes logiques,*** qui prennent en entrée un ou des signaux électriques (chaque entrée est dans un état "haut" ou état "bas"), et donne en sortie un ou des signaux électriques (chaque sortie est aussi dans un état "haut" ou à un état "bas").

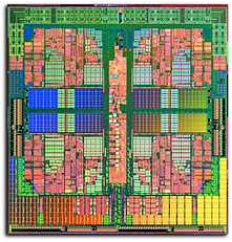
Le fonctionnement de ces portes logiques suit l'algèbre de Boole ; en les agençant correctement entre elles, on réalise alors des ***circuits logiques*** permettant d'effectuer toutes les opérations de base nécessaires au fonctionnement d'une machine informatique.
Deux types de circuits logiques existent, les ***circuits combinatoires*** (l'état de leur sortie ne dépend que de l'état de leurs entrées), et les *circuits séquentiels* (l'état de leur sortie dépend en plus du temps) ; seuls les premiers seront étudiés par la suite.

Les circuits logiques les plus complexes que l'on trouve actuellement dans les microprocesseurs ou les circuits mémoire RAM comportent plusieurs milliards de transistors...







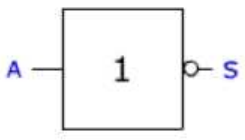


TRANSISTORS → PORTES LOGIQUES → CIRCUITS LOGIQUES → CPU, MEMOIRE, ...

II. PORTES LOGIQUES USUELLES

Pour tout circuit logique, on peut établir sa ***table de vérité***, tableau qui recense les valeurs que peut prendre l'état de sortie en fonction de tous les états possibles en entrée. *Nous considérerons pour simplifier que des portes à deux entrées.*
Établir ainsi la table de vérité des portes logiques ci-dessous, qui correspondent aux opérateurs booléens « de base » :

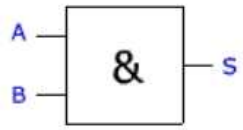
Porte NON (ou inverseur)
→ sa sortie est à l'état opposé de celui de son entrée :

Symbole international	Table de vérité
	A | S 0 | 1 |

Porte ET

→ sa sortie n'est à l'état haut que si son entrée A ET son entrée B sont à l'état haut :

Symbole international	Table de vérité

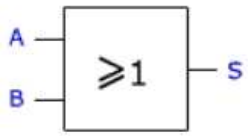


A	B	S
0	0	
1	0	
0	1	
1	1	

Porte OU

→ sa sortie est à l'état haut si son entrée A OU son entrée B est à l'état haut :

Symbole international	Table de vérité



A	B	S
0	0	
1	0	
0	1	
1	1	

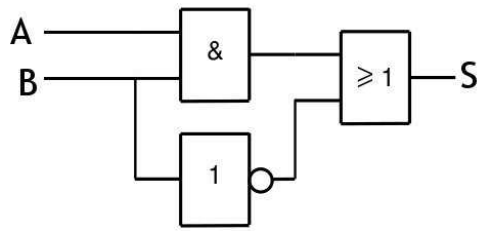
III. ASSOCIATION DE PORTES LOGIQUES : CIRCUITS COMBINATOIRES

A partie de ces portes, on peut alors en les associant réaliser n'importe quel ***circuit combinatoire*** pouvant effectuer diverses ***fonctions booléennes*** .

Vous allez déterminer d'abord sur papier la table de vérité de divers assemblages de portes logiques, puis vous vérifierez votre réponse à l'aide d'un simulateur en ligne à l'adresse : https://logic.ly/demo

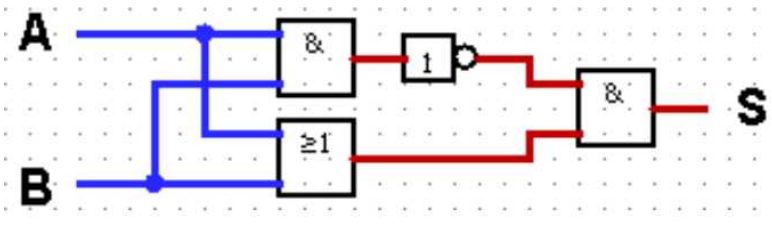
1. Table de vérité

Compléter la table de vérité du circuit combinatoire suivant :



A	B	S
0	0	
1	0	
0	1	
1	1	

Compléter la table de vérité du circuit combinatoire ci-dessous ; il réalise la fonction booléenne appelé OU EXCLUSIF, noté XOR :



A	B	S
0	0	
1	0	
0	1	
1	1	

2. <u>Vérification à l'aide du simulateur en ligne</u>

- La prise en main du simulateur est assez intuitive : on place les portes logiques en les « cliquer-déplaçant » depuis le menu de droite, puis on relie leurs entrées et leur sorties selon le circuit combinatoire que l'on veut réaliser.
- Pour avoir les symboles internationaux des portes logiques, aller dans le menu `Edit > Applications Settings`, et dans `Symbol Style`, choisir `ANSI /IEEE`
- Pour les entrées, vous pouvez y relier un interrupteur qui permettra de basculer l'état de l'entrée de 0 à 1.
- Pour visualiser les sorties, vous pouvez directement y relier des ampoules : une ampoule éteinte indiquera une sortie à 0, une ampoule allumée une sortie à 1 !

→ réalisez les deux circuits combinatoires précédents, et à l'aide de la simulation, vérifiez les tables de vérité que vous avez déterminées à la question précédente.

IV. <u>DES CIRCUITS PLUS COMPLEXES : L'ADDITIONNEUR BINAIRE</u>

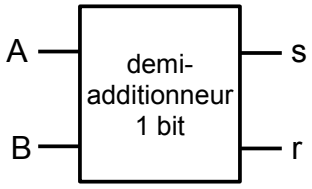
C'est un circuit combinatoire de base dans un microprocesseur, puisqu'il permet l'addition bit à bit de deux nombres, ou leur soustraction si ils sont représentés en complément à deux (*voir chapitre sur la représentation des entiers naturels et relatifs*)

1. <u>Rappels sur l'addition binaire</u> : effectuer l'addition suivante en tenant compte des retenues éventuelles ; vérifiez ensuite votre résultat en convertissant les nombres en base 10 :

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1\ 1_2 \\ +\ 1\ 1\ 1\ 0\ 1\ 0_2 \\ \hline \end{array}$$

2. <u>Demi-additionneur 1 bit</u> : on dispose de deux entrées A et B représentant chacune 1 bit (de valeur 0 ou 1), et on voudrait obtenir la somme *s* de ces 2 bits suivant leur valeur, ainsi que la retenue éventuelle *r.*
→ compléter la table de vérité du circuit combinatoire qui réaliserait cette opération :

A	B	s	r
0	0		
1	0		
0	1		
1	1		



- à quelle fonction booléenne correspond alors *s* ?
- même question pour *r* ?
- Proposer alors un circuit combinatoire simple qui permette de réaliser l'addition de deux bits ; ce circuit est appelé ***demi-additionneur 1 bit*** :

- Vérifiez votre proposition de circuit à l'aide du simulateur.

3. <u>Additionneur complet 1 bit</u> :

Le circuit précédent permet uniquement l'addition de deux nombres à 1 bit ; pour des nombres comportant un plus grand nombre de bit, il ne peut pas être utilisé, car il ne tient pas compte des éventuelles retenues précédentes (*voilà pourquoi il est appelé « demi »-additionneur...*)
→ un additionneur complet doit donc comporter une entrée supplémentaire dont l'état indique une éventuelle retenue précédente.

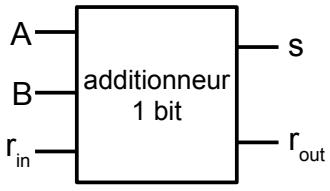


Table de vérité :

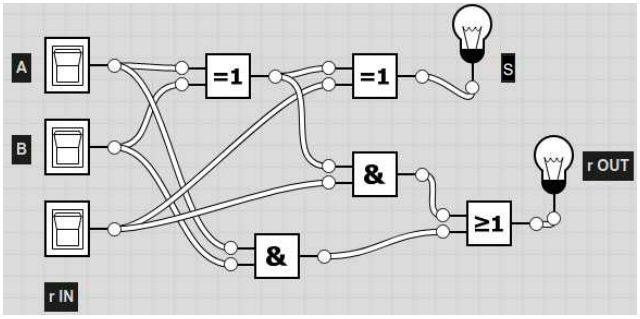
Compléter la table de vérité que doit vérifier le circuit combinatoire d'un additionneur 1 bit complet en vous inspirant de l'addition binaire que vous avez faite précédemment :

A	B	r_{in}	s	r_{out}
0	0	0		
0	1	0		
1	0	0		
1	1	0		
0	0	1		
0	1	1		
1	0	1		
1	1	1		

Simulation du circuit d'un additionneur 1 bit :

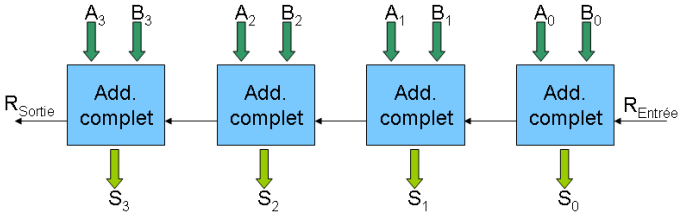
On montre que le circuit combinatoire réalisant l'addition binaire complète de deux bits en tenant compte de la retenue est celui schématisé ci-contre.

Vérifier à l'aide du simulateur que ce circuit a bien la table de vérité ci-dessus...



4. Circuit additionneur *n* bits :

En « chaînant » plusieurs additionneurs 1 bit les uns à la suite des autres, la retenue du précédent étant « envoyée » à l'entrée du suivant, on réalise un ***additionneur à propagation de retenue*** qui permet l'addition de nombre comportant un plus grand nombre de bits.



V. LOGIQUE BOOLEENNE SOUS PYTHON

Le langage Python possède un type de données de base utilisable en logique booléenne : le type `bool` (= *boolean*).
Les variables de type `bool` ne peuvent donc prendre que deux valeurs : `True` (= VRAI) ou `False` (= FAUX).

Les opérateurs logiques de base pour manipuler les booléens sont en Python les suivants :
NON = `not` ET = `and` OU = `or`

1. Expression de quelques opérations sur des booléens

Prévoir le résultat (`True` ou `False`) des opérations suivantes (`a, b, c et d`) sont des variables de type `bool` :

a	b	c	d	Opération	Résultat
`True`	`True`			`not ( a and b )`	
`True`	`False`	`True`		`a or ( b and not c )`	
`False`	`True`	`True`	`True`	`( a and not b ) or ( not c and d )`	

Vérifiez vos résultats à l'aide de Python.

2. L'opération OU EXCLUSIF

- Établir l'instruction Python correspondant à un OU EXCLUSIF entre deux booléens a et b.
- Vérifiez votre résultat avec Python.