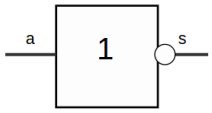
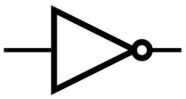
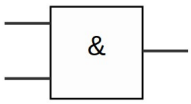
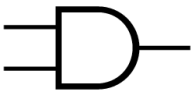
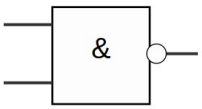
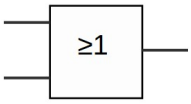
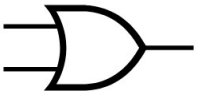
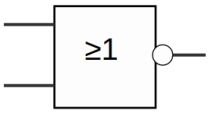
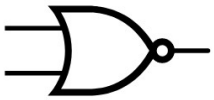
Les compositions de portes logiques NAND

I – Rappels sur les fonctions logiques de base

NON	**NOT**	**On inverse l'entrée**		
ET	**AND**	**Vrai si TOUTES les entrées sont vraies**		
NON-ET	NAND	Faux si TOUTES les entrées sont vraies		
OU	**OR**	**Vrai si au moins UNE des entrées est vraie**		
NON-OU	NOR	Faux si au moins UNE des entrées est vraie.		

TABLES DE VERITE AVEC DEUX ENTREES uniquement :

a	**not a**
1	0
1	0
0	**1**
0	**1**

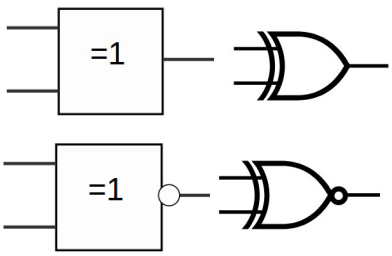
a	b	**a and b**	**a nand b**
1	**1**	**1**	0
1	0	0	**1**
0	**1**	0	**1**
0	0	0	**1**

a	b	**a or b**	**a nor b**
1	**1**	**1**	0
1	0	**1**	0
0	**1**	**1**	0
0	0	0	**1**

II – XOR ou OU exclusif →

Exercice 01/ 02° Avec la définition, remplir les tables du **XOR**, XNOR à 2 entrées puis du XOR à 3 entrées

a	**b**	**a xor b**	**a xnor b**
1	1		
1	0		
0	1		
0	0		



a	**b**	**c**	**a xor b xor c**
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

Remarque :

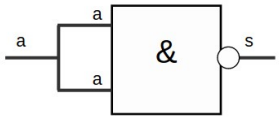
`a xor b` est équivalent à **`a != b`**
`a xnor b` est équivalent à **`a == b`**

III – Technologie NAND

AND : VRAI uniquement si les entrées sont toutes VRAIES
NAND : FAUX uniquement si les entrées sont toutes VRAIES

On peut réaliser toutes les fonctions logiques à l'aide d'une association de plusieurs NAND.
On dit que la fonction NAND est **complète** ou **universelle**.

Question 04 : Fonction NOT

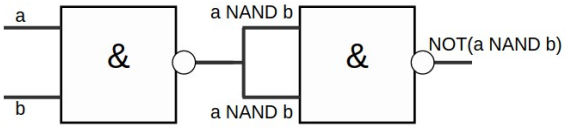


`not a = a nand a`

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

a	a and a	a nand a
1		
0		

Question 05 : Fonction AND

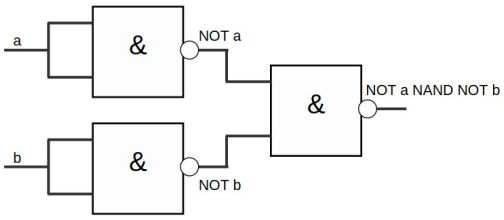


`a and b = not(a nand b)`

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

a	b	a nand b	not(a nand b)
1	1		
1	0		
0	1		
0	0		

Question 06 : Fonction OR



`a or b = not a nand not b`

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

a	b	not a	not b	not a nand not b
1	1			
1	0			
0	1			
0	0			

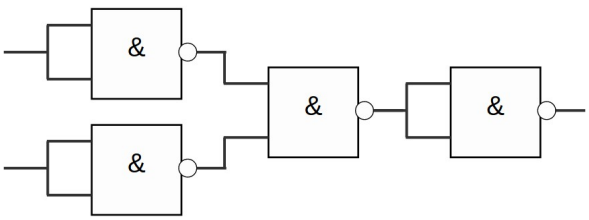
Régles de priorité : Le plus prioritaire est le **not**. Ensuite le **AND (.)**. Puis le **OR(+)**.

Exercice 07 **Un élève écrit ceci en python :** `not a or not b and c.`

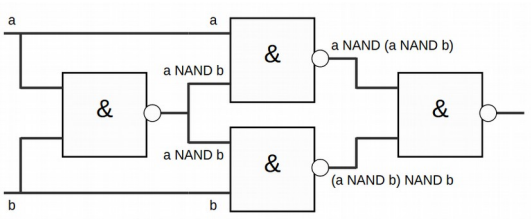
A quoi correspond en réalité cette opération ?

- A : `(not a) or (not (b and c))`
- B : `not (a or not (b and c))`
- C : `not (a or not b) and c`
- D : `(not a) or ((not b) and c)`

Faire un NOR consiste juste à complémenter (rajouter un NOT) à la fin d'un circuit OR :



Question 08 : Fonction XOR



a xor b =
(a NAND (a NAND b)) NAND ((a NAND b) NAND b)

a	b	a nand b	X = a nand (a nand b)	Y = (a nand b) nand b	X nand Y
1	1				
1	0				
0	1				
0	0				

Pour créer un XNOR, on rajoute juste une porte NOT en plus à la fin.

IV – Test d'égalité

ll faut appliquer bit à bit un XNOR puisque
a xnor b est équivalent à **a == b .**

Si tous les bits de sortie sont à 1, c'est que les deux entrées sont identiques !

0	1	0	1	0	0	1	1
xnor	xnor	xnor	xnor	xnor	xnor	xnor	xnor
0	1	0	0	1	1	0	0
donne	donne	donne	donne	donne	donne	donne	donne

Un processeur 8 bits devra donc faire deux opérations de comparaison si on lui demande de comparer deux contenus sur 2 octets (16 bits).

Exercice 09 Un processeur 16 bits (bus, mémoire, UAL, tout fonctionne avec 16 bits d'un coup) est-il meilleur qu'un processeur 8 bit pour traiter des données de 8 bits maximum ?

Exercice 10 Un processeur 16 bits (bus, mémoire, UAL, tout fonctionne avec 16 bits d'un coup) est-il meilleur qu'un processeur 8 bit pour traiter des données de 16 bits maximum ?

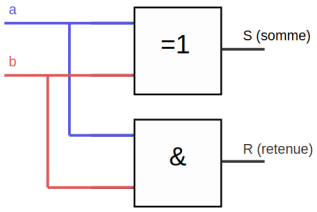
Exercice 11 Vous ne travaillez qu'avec des données de 64 bits maximum. On vous propose un ordinateur double-coeur (2 processeurs) de 64 bits ou un ordinateur de 128 bits. Que choisir ? On considère que les deux ordinateur ont la même fréquence de traitement.

Exercice 12 Vous ne travaillez qu'avec des données de 64 bits maximum. On vous propose un ordinateur de 64 bits 3 Ghz ou un ordinateur de 128 bits 2 GHz. Que choisir ?

V – Addition

Rappel : une addition binaire est telle que :
0+0 = 0
0+1 = 1
1+1 = 10

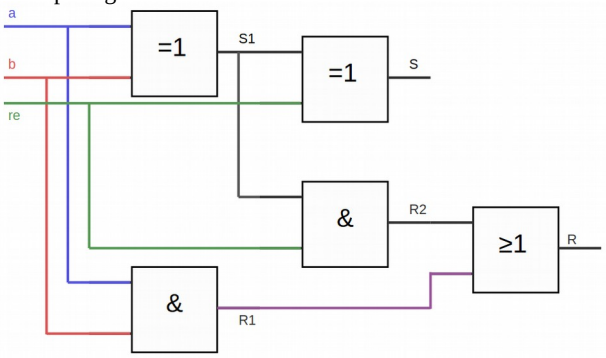
L'unité de a + b correspond également à **a XOR b**
La retenue de a + b correspond à **a AND b**



a	b	Retenue (carry) de a+b	Unité de a+b
1	1	**1**	**0**
1	0	**0**	**1**
0	1	**0**	**1**
0	0	**0**	**0**

Ceci est un demi-additionneur car il ne sait pas gérer la présence éventuelle d'une retenue en entrée en plus de a et b.

Voici le schéma d'un additionneur complet POUR UN BIT uniquement. Il en faut 8 pour gérer un octet.



re	a	b	S1= a XOR b	R1= a AND b	R2= S1 AND re	R= R1 OR R2	S= S1 XOR re
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	1
1	0	1	1	0	1	1	0
1	1	0	1	0	1	1	0
1	1	1	0	1	0	1	1