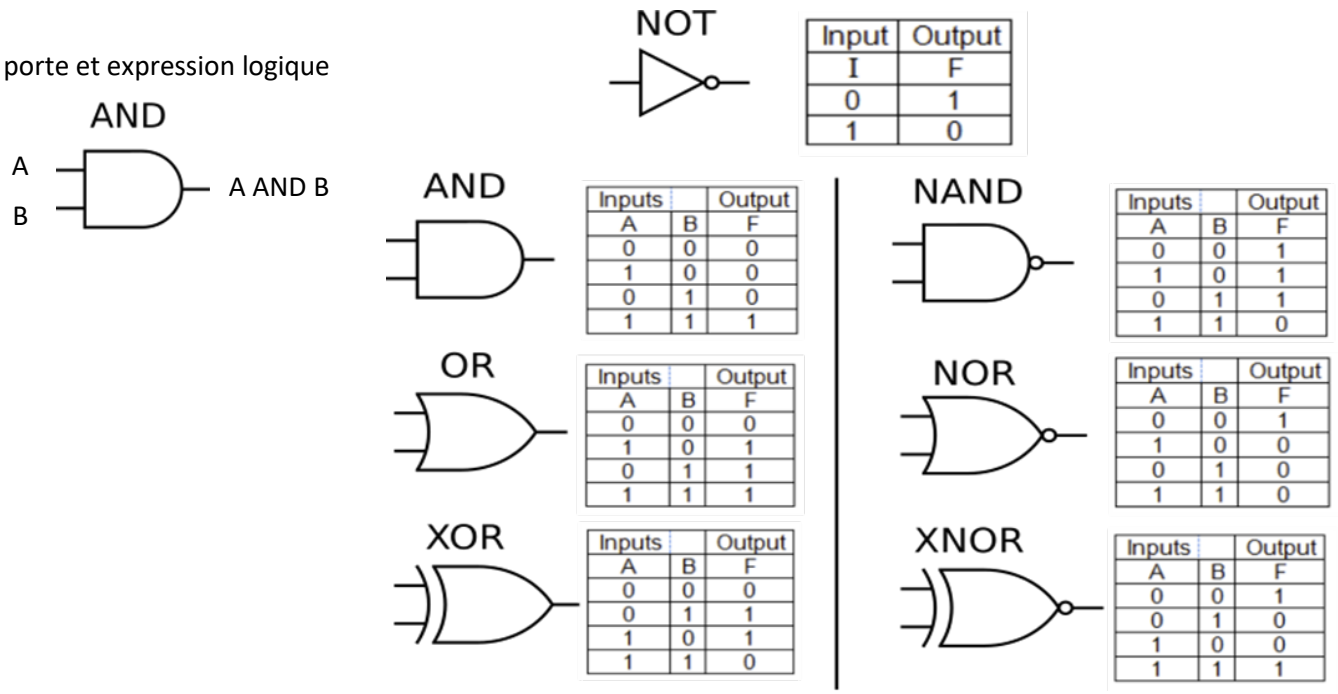
M3.L1: Série d'exercices sur l'architecture de l'ordinateur

Pour les exercices de cette série, nous serons amenés à manipuler des *variables logiques* prenant seulement deux niveaux de tension notés 0 et 1. De telles variables sont équivalentes aux booléens que nous avons déjà rencontrés et les opérateurs logiques mentionnés dans cette série sont les opérateurs logiques bien connus auxquels s'ajoutent quelques autres. Nous utilisons les noms français ou anglais de ces opérateurs.

Notation : Soit x une variable logique, la notation $\bar{x}$ signifie la négation de *x* : si x=0 alors $\bar{x}$ =1 et vice versa.

1. Introductions aux portes logiques

Voici les symboles à connaître pour les portes logiques qui réalisent les fonctions logiques familières (négation, et, ou, ou-exclusif) ainsi que la *négation de ces fonctions logiques* (non-et, non-ou, non-ou-exclusif). Ces dernières portent la lettre N en préfixe: NAND veut dire non-et, NOR veut dire non-ou etc… De même le symbole des portes logiques comportant une composant de "négation" a un petit cercle au niveau de la sortie :



Symbole et **table de vérité** des portes logiques standards:
les entrées A et B sont à gauche, peu importe laquelle, et la sortie F est à droite du symbole.
La table de vérité indique la sortie de la porte logique pour toutes les combinaisons possibles des entrées A et B.

1.1) l'opérateur XOR produit un 1 en sortie pour seulement deux combinaisons des entrées A et B :

- Cas 1 : si A vaut 0 et B vaut 1
- Cas 2 : si A vaut 1 et B vaut 0

1.1.1) construire un circuit avec une porte AND et un inverseur qui produit le **cas1**, c'est-à-dire 1 en sortie pour cette combinaison de A et de B et 0 pour toutes les autres combinaisons de A et B.
1.1.2) construire un circuit avec une porte AND et un inverseur qui produit le **cas2**, c'est-à-dire 1 en sortie pour cette combinaison de A et de B et 0 pour toutes les autres combinaisons de A et B.

1.1.3) Comment faites-vous ensuite avec une porte OR pour assembler ces deux circuits en un circuit dont la sortie est celle de XOR ?

1.2) On aurait pu aussi bien commencer par exprimer XOR sous forme d'une expression logique en fonction de AND, OR et NOT. Quelle est cette expression logique ?

2. Tables de vérité d'une expression logique quelconque

1.1. Un circuit logique a 5 entrées A, B, C, D, et E.
La table de vérité de ce circuit devra montrer la sortie du circuit pour toutes les combinaisons distinctes des entrées. Combien d'états distincts ce circuit peut-il avoir en entrée ?

1.2. cas général pour la construction d'une table de vérité. Il suffit d'étendre le modèle vu en page précédente avec une colonne par variable logique en entrée et une colonne par expression logique intermédiaire. Si on a N variables en entrées on aura 2^N lignes correspondants aux 2^N combinaisons distinctes des entrées. Pour écrire ces 2^N lignes il suffit d'énumérer les 2^N premières valeurs binaires sur N bits.

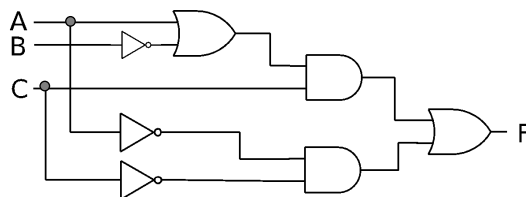
1.2.1) Donnez la table de vérité et dessinez le circuit logique correspondant à l'expression booléenne :

$$Y = (\bar{A}\ et\ \bar{B}) et\ (B\ et\ \bar{C})$$

1.2.2) Aurait-on pu le déduire le résultat de ce circuit simplement en analysant l'expression logique ?

1.3) Ecrivez l'équation logique booléenne pour le circuit suivant:

Pour ce circuit, on a rajouté des points gris pour supprimer d'éventuelle ambiguïté de connection.



1.4) Dessinez le circuit qui implémente l'expression logique suivante:

F = not (A or B) and (C or not B)

3. Compréhension des programmes en assembleur

On considère le programme assembleur suivant (voir cours M3.L1 pour les instructions):

```
1: charge       r4, r1
2: charge       r3, 0
3: somme        r4, r4, -1
4: cont_neg     r4, 7
5: somme        r3, r2, r3
6: continue     3
7: stop
```

Si r1 = 3 et r2 = 8, quelle est la sortie de ce programme dans le registre r3 ?

En général, que fait ce programme?

4. Programme de multiplication de nombres complexes

Ecrivez un programme assembleur pour calculer le produit de deux nombres complexes y et z.

Utilisez les instructions similaires à celles vues au cours (p.ex. **multiplie r1, r2, r3** pour déposer dans r1 le résultat de la multiplication de r2 et r3).

Vous pouvez utiliser les registres de r0 à r9. Initialement, le registre r0 contient Réel(y), r1 contient Imag(y), r2 contient Réel(z), r3 contient Im(z). A la fin de l'exécution du programme, la partie réelle du résultat doit se trouver dans r4 et la partie imaginaire dans r5.

{ Pour rappel: (a + bi)(c + di) = (ac - bd) + (ad + bc)i }

5. Programme de comparaison de valeurs horaires

Ecrivez un programme assembleur qui détermine laquelle de deux heures, A et B, exprimées en heures et en minutes est la plus petite (c.à.d. arrive le plus tôt dans la journée).

Toutes les heures données sont entre 00:00 et 24:00 heures (donc p.ex. 13:45 et pas 1:45).

Vous pouvez utiliser les registres de r0 à r9. Les registres r0 à r3 contiennent les informations suivantes:

r0	r1	r2	r3	...
A heure	A minutes	B heure	B minutes	

A la fin de l'exécution du programme, r9 doit contenir 1 si A est une heure strictement plus petite que B et 0 sinon.

Exemple: Si on doit comparer 8h10 et 21h45, on vous donne r0 = 8, r1 = 10, r2 = 21 et r3 = 45 et à la fin de l'exécution, r9 devra contenir 1.

notation: L'instruction **continue_pp a, b, c** fait continuer l'exécution à la ligne **c** si **a** est un nombre strictement plus petit que **b**.

6. Circuit

Considerez le circuit suivant. Quelle fonction logique réalise-t-il ?