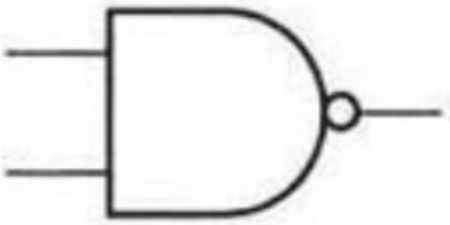
MODÈLE D'ARCHITECTURE SÉQUENTIELLE - MODÈLE DE VON NEUMANN - CORRECTION

Exercice 1

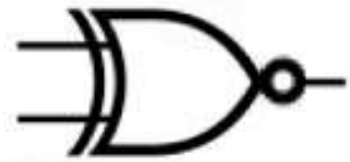
1. Déterminer : le symbole (américain), la table de vérité et la notation en algèbre de Boole de la porte **NON ET** ou **NAND** (qui est le contraire de la porte ET).

Symbole	Table de vérité	Notations
	A | B | A NAND B 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0	NON(A.B) NON A OU NON B NON A + NON B

2. Déterminer : le symbole (américain), la table de vérité et la notation en algèbre de Boole de la porte **NON OU** ou **NOR** (qui est le contraire de la porte OU).

Symbole	Table de vérité	Notations
	A | B | A NOR B 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0	NON(A OU B) NON(A + B) NON A ET NON B NON(A).NON(B)

3. Déterminer : le symbole (américain), la table de vérité et la notation en algèbre de Boole de la porte **NON OU EXCLUSIF** ou **XNOR** (qui est le contraire de la porte XOR).

Symbole	Table de vérité	Notations
	A | B | A XNOR B 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1	NON(A XOR B) NON(A ⊕ B) NON A ET NON B + A.B NON(A).NON(B)

4. Déterminer la table de vérité de l'expression suivante : NOT((A AND B)XOR A).

A	B	(A AND B)	((A AND B) XOR A)	NOT((A AND B) XOR A)
0	0	0	0	1
0	1	0	0	1
1	0	0	1	0
1	1	1	0	1

Exercice 2 [se référer à la dernière page du cours sur l'assembleur ARM]

Expliquez les instructions suivantes :

1) `ADD R0, R1, #42`
additionne la valeur contenue dans le registre R1 et le nombre 42 et stocke le résultat dans le registre R0

2) `LDR R5,98`
charge la valeur contenue dans la RAM à l'adresse 98 et la stocke dans le registre R5

3) `STR R0,15`
stocke le contenu du registre R0 dans la RAM à l'adresse 15

4) `CMP R4, #18`
`BGT 77`
si la valeur contenue dans le registre R4 est plus grande que la valeur 18, exécuter la ligne de programme située à l'adresse 77 dans la RAM

5) `B 100`
saut inconditionnel à l'adresse 100 de la RAM : on va lire directement l'instruction écrite dans la RAM à l'adresse 100

Exercice 3 [se référer à la dernière page du cours sur l'assembleur ARM]

Écrire les instructions en assembleur correspondant aux phrases suivantes :

1. Additionne la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1, le résultat est stocké dans le registre R5 ADD R5, R0, R1
2. Place la valeur stockée à l'adresse mémoire 878 dans le registre R0 LDR R0,878
3. Place le contenu du registre R0 en mémoire vive à l'adresse 124 STR R0,124
4. Si la valeur stockée dans le registre R0 est égale 42 alors la prochaine instruction à exécuter se situe à l'adresse mémoire 85 , sinon la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 478

```
CMP R0,#42
BEQ 85
B 478
```

Exercice 4 [se référer à la dernière page du cours sur l'assembleur ARM]

Des entiers sont stockés aux adresses 7,8,9,10 dans la mémoire vive. Écrire une suite d'instructions en assembleur qui additionne le contenu de ces mémoires et le stocke à l'adresse 11.

Pour cela, vous n'avez le droit d'utiliser que deux registres : R0 et R1

```
LDR R0,7
LDR R1,8
ADD R1,R0,R1
LDR R0,9
ADD R1,R0,R1
LDR R0,10
ADD R1,R0,R1
STR R1,11
```

Exercice 5 [se référer à la dernière page du cours sur l'assembleur ARM]

Les cases mémoires 11 12 13 de la mémoire vive contiennent des entiers. Que fait le programme suivant :

100	102	104	106	108	110	112	114	116
LDR R0,11	CMP R0,#2	BEQ 112	LDR R2,13	STR R2,30	B 116	LDR R2,12	STR R2,30	HALT

100 : stocke dans le registre R0 la valeur stockée à l'adresse 11 de la RAM
102 : compare la valeur dans R0 et le nombre 2
104 : si les valeurs comparées précédemment sont égales, aller à l'instruction n°112

si les 2 valeurs sont égales

si les 2 valeurs sont différentes

112 : place la valeur stockée dans la RAM à l'adresse 12 dans le registre R2
114 : place la valeur stockée dans R2 dans la RAM à l'adresse mémoire 30

106 : place la valeur stockée à l'adresse mémoire 13 de la RAM dans la registre R2
108 : place la valeur du registre R2 à l'adresse mémoire 30 de la RAM
110 : la prochaine instruction à lire est la n°116

116 : arrêter l'exécution du programme

Exercice 6

Voici un programme Python, et son équivalent en assembleur.

Après avoir analysé très attentivement le programme en assembleur ci-contre, établir une correspondance entre les lignes du programme en Python et les lignes du programme en assembleur.

À quoi sert la ligne "B endif" ?

À quoi correspondent les adresses mémoires 23, 75 et 30 ?

```
x=4
y = 8
if x == 10:
    y = 9
else :
    x=x+1
z=6
```

```
1)    MOV R0, #4
2)    STR R0,30
3)    MOV R0, #8
4)    STR R0,75
5)    LDR R0,30
6)    CMP R0, #10
7)    BNE else
8)    MOV R0, #9
9)    STR R0,75
10)    B endif
11)else:
12)    LDR R0,30
13)    ADD R0, R0, #1
14)    STR R0,30
15)endif:
16)    MOV R0, #6
17)    STR R0,23
18)    HALT
```

« *x*=4 »
« *y*=8 »
« *x*==10 »
si « False », aller à 11
y=9, puis aller en « endif »
x=*x*+1
« *z*=6 »

La ligne B endif sert à sauter le else (branchement fait avec BNE else)
L'adresse 30 stocke *x* ; adresse 75 stocke *y* ; adresse 23 stocke *z*