Exercice 1

Activité

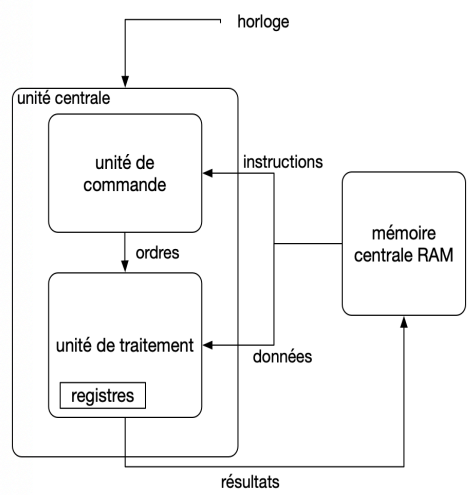


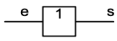
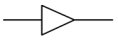
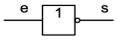
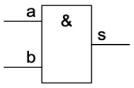
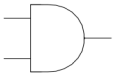
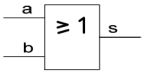
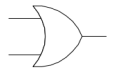
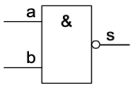
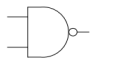
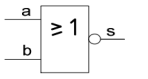
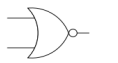
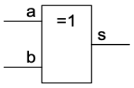
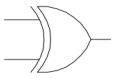
Figure 1 – processeur et architecture Von Neumann

1. Dans l'architecture Von Neumann, les données et programmes sont écrits sur une même mémoire (en binaire). Une instruction en mémoire combine souvent des parties opératoires, et des données. Comment ces deux parties sont-elles distribuées dans le processeur ?
2. Comment sont liées les mémoires non volatiles (disques durs), les mémoires volatiles (RAM), et celles du processeur (registre) ?
3. Quel est le rôle de l'horloge ?

Exercice 2

Exercices

2.1 Portes logiques

Fonction logique	Symbole européen	Symbole américain
OUI s = e		
NON (NO) $s = \overline{e}$		
ET (AND) s = a · b		
OU (OR) s = a + b		
NON ET (NAND) $s = \overline{a \cdot b}$		
NON OU (NOR) $s = \overline{a + b}$		
OU exclusif (EXOR) s = a ⊕ b		

2.1.1 Donner la table de verité des portes logiques ET, OU et XOR.

2.1.2 On dispose d'une porte logique inconnue P1. La table de vérité est donnée ci-dessous :

E1	E2	S
0	0	1
0	1	1
1	0	1
1	1	0

a. Cette porte logique se trouve t-elle dans le document ci-dessus ? Quel est sa fonction ?

b. Comment peut-on réaliser la même fonction à l'aide des portes logiques ET et NON ?

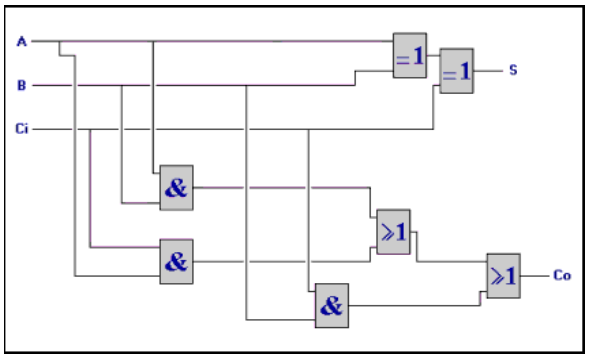
2.1.3 Expliquer pourquoi la fonction logique ET est représentée par `S = E1 . E2`, et la fonction OU par `S = E1 + E2`

2.1.4 Addition sur 1 bit

Lorsque l'on additionne les nombres $A_1A_2A_3$ et $B_1B_2B_3$, les bits sont additionnés 2 à 2 : $A3$ est additionné à B_3, A_2 à B_2, etc...

Pour la fonction logique OU, l'addition n'est pas exactement réalisée entre 2 bits à cause de la possible retenue (1 +

1). On donne ci-dessous le circuit réalisant l'addition sur 1 bit de 2 entiers A et B. C_i représente la retenue sur les bits précédents A et B. C_0 est la retenue sortante.



Compléter la table de vérité correspondante :

Ci	A	B	Co	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Exercice 3

Extrait du sujet de bac : Sujet Centres étrangers 2 2021

Thèmes abordés : conversion décimal/binaire, table de vérité, codage des caractères

L'objectif de l'exercice est d'étudier une méthode de cryptage d'une chaîne de caractères à l'aide du codage ASCII et de la fonction logique XOR.

1. Le nombre 65, donné ici en écriture décimale, s'écrit 01000001 en notation binaire. En détaillant la méthode utilisée, donner l'écriture binaire du nombre

2.

3. La fonction logique OU EXCLUSIF, appelée XOR et représentée par le symbole $\oplus$, fournit une sortie égale à 1 si l'une ou l'autre des deux entrées vaut 1 mais pas les deux.

On donne ci-contre la table de vérité de la fonction XOR :

E_1	E_2	$E_1 \oplus E_2$
0	0	0
0	1	1
1	0	1
1	1	0

Si on applique cette fonction à un nombre codé en binaire, elle opère bit à bit.

```
    1100
⊕   1010
-------
=   0110
```

Poser et calculer l'opération : $11001110 \oplus 01101011$

3. On donne, ci-dessous, un extrait de la table ASCII qui permet d'encoder les caractères de A à Z. On peut alors considérer l'opération XOR entre deux caractères en effectuant le XOR entre les codes ASCII des deux caractères. Par exemple : 'F' XOR 'S' sera le résultat de $01000110 \oplus 01010011$.

Code ASCII Décimal	Code ASCII Binaire	Caractère
65	01000001	A
66	01000010	B
67	01000011	C
68	01000100	D
69	01000101	E
70	01000110	F
71	01000111	G
72	01001000	H
73	01001001	I
74	01001010	J
75	01001011	K
76	01001100	L
77	01001101	M

Code ASCII Décimal	Code ASCII Binaire	Caractère
78	01001110	N
79	01001111	O
80	01010000	P
81	01010001	Q
82	01010010	R
83	01010011	S
84	01010100	T
85	01010101	U
86	01010110	V
87	01010111	W
88	01011000	X
89	01011001	Y
90	01011010	Z

On souhaite mettre au point une méthode de cryptage à l'aide de la fonction XOR.

Pour cela, on dispose d'un message à crypter et d'une clé de cryptage de même longueur que ce message. Le message et la clé sont composés uniquement des caractères du tableau ci-dessus et on applique la fonction XOR caractère par caractère entre les lettres du message à crypter et les lettres de la clé de cryptage.

Par exemple, voici le cryptage du mot ALPHA à l'aide de la clé YAKYA :

Message à crypter	A	L	P	H	A
Clé de cryptage	Y	A	K	Y	A
	↓	↓	↓	↓	↓
Message crypté	'A' XOR 'Y'	'L' XOR 'A'	'P' XOR 'K'	…	…

Ecrire une fonction `xor_crypt(message, cle)` qui prend en paramètres deux chaînes de caractères et qui renvoie la liste des entiers correspondant au message crypté.

Aide :

- On pourra utiliser la fonction native du langage Python `ord(c)` qui prend en paramètre un caractère c et qui renvoie un nombre représentant le code ASCII du caractère c.
- On considère également que l'on dispose d'une fonction écrite `xor(n1,n2)` qui prend en paramètre deux nombres n1 et n2 et qui renvoie le résultat de $n1 \oplus n2$.

Exercice 4

Travaux pratiques : Chiffrage-dechiffrage XOR

Ce travail pratique s'appuie sur l'exercice de bac Centres étrangers 2 2021

4.1 fonction ascii

```
ascii('YAKA')
[89, 65, 75, 65]
```

Aide :

- on utilisera la fonction native python ord qui retourne le rang d'un caractere dans la table ascii :

```
ord('Y')
89
```

- Cette fonction òrdfait l'inverse de celle appeléechr‘ :

```
chr(89)
'Y'
```

4.2 fonction xor

Ecrire d'abord une fonction xor qui prend en paramètres 2 nombres entiers n1 et n2, les convertit en binaire, et applique bit à bit la fonction XOR sur ces 2 nombres. Puis la fonction retourne l'entier correspondant. Il y a donc 3 étapes :

- conversion de n1 et n2 en binaire
- appliquer bit à bit XOR
- convertir le binaire obtenu en entier

Exemple :

```
xor(89,102)
63
```

Aides :

- Pour convertir le nombre lentiern1en binaire, on pourra s'inspirer de l'instruction suivante en python, qui utilise la fonction nativebin‘ :

```
b1 = bin(n1)[2:]
```

- Pour convertir un binaire b en entier :

```
int(b,2)
```

4.3 fonction xor_crypt

Exemple :

```
xor_crypt('debarquement', 'YAKYAKYAKYAK')
=$)83:,$&</?
```