Exercice 1

Questions de cours

1.1 Architecture

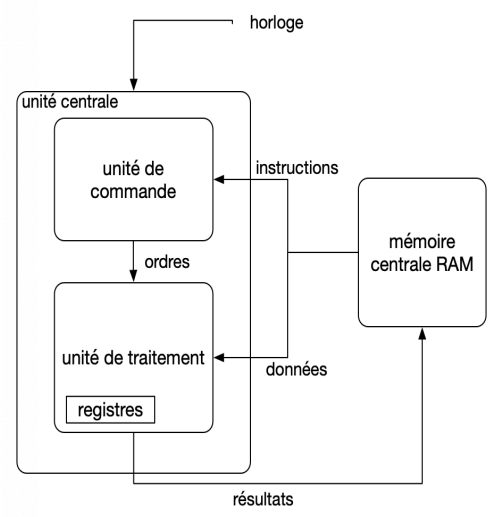


FIGURE 1 – processeur et architecture Von Neumann

1. Quels éléments font partie du processeur : l'Unité arithmetique et logique, la mémoire vive, la mémoire de masse, l'unité de contrôle, le bus de données, le bus d'adresse ?
2. 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 ? Quel registre en particulier est utilisé ?
3. Comment sont reliés la RAM et le processeur ?
4. Quel est le rôle de l'horloge ?
5. Mettre les actions du cycle d'éxecution dans l'ordre :

- l'adresse du code opération est envoyée à la mémoire
- le code opération est envoyé à l'UAL
- le compteur de programme est incrémenté
- le contenu de la mémoire est envoyé à l'UAL
- le contenu du compteur de programme est envoyé à la mémoire
- l'instruction est lue de la mémoire
- l'UAL effectue l'opération sur les 2 données lues de la mémoire

1.2 Encodage et nombre binaire

1. (vu en 1ere NSI) Calculer le poids d'un fichier texte de 1300 caractères, encodés en ASCII étendu (1 octet par caractère).

2. (vu en 1ere NSI) Calculer le poids d'une image non compressée en couleur RVB (24 octets par pixel) de dimension 600 * 400 pixels.

3. (vu en 1ere NSI) Calculer le poids d'une video de 10s, de 24 images par seconde, constituée d'un assemblage des images non compressées de la question précédente.

4. (vu en 1ere NSI) Quel est le principe d'encodage par *utf-8* ? Pourquoi dit-on qu'il s'agit du meilleur compromis entre le code ASCII et l'Unicode ?

5. Donner une valeur possible pour un ordinateur de bureau moderne, pour chacun des composants suivants : la mémoire vive, la fréquence de l'horloge, la mémoire de masse.

1.3 Fréquence du processeur

Un programme traite des données qui sont en mémoire. La partie du programme qui traite ces données est constituée de 100 instructions et l'horloge du processeur a une vitesse de 1 GHz. Combien de temps faut-il pour traiter un million de données ?

Exercice 2

Exercices sur le langage assembleur

2.1 Les 3 familles d'instructions

Parmi les instructions suivantes, quelles sont celles qui sont de type :

- transfert de données
- arithmétiques
- rupture de séquence ?

Instructions :

```
x: 23 // affectation
LDR R0, x // charge dans R0 la valeur de x
MOV R1, #23 // place dans R1 la valeur 23
STR R1, x // place dans x la valeur de R1
ADD R0, R1, #16 // add R1 + 16
SUB R0, R0, #16 // soustraction R0 - 16
LSL R0, R0, #1  // multiplier par 2
LSR R0, R0, #1 // diviser par 2
CMP R1, R2 // compare
B label // toujours revenir à la ligne d'etiquette label
BGT label // aller à label si CMP R1,R2 donne R1 > R2 (Greater Than)
BLT label // aller à label si CMP R1,R2 donne R1 < R2 (Lower Than)
```

2.2 Boucle infinie

Ecrire le plus petit programme en assembleur qui boucle indéfiniment.

2.3 programme qui affiche toutes les puissances de 2 jusqu'à 128 :

```
0       MOV R0, #1
1       MOV R1, #0
  multiplie:
2       LSL R0, R0, #1
3       OUT R0,4
4       ADD R1,R1,#1
5       CMP R1, ...
6       BLT multiplie
7       HALT
```

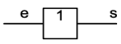
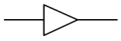
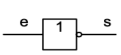
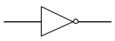
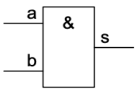
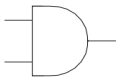
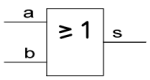
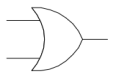
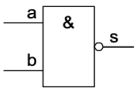
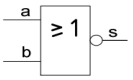
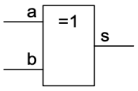
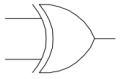
1. Que fait le programme à la ligne 4 : `ADD R1,R1,#1` ?
2. Compléter le programme à la ligne 5 pour que celui-ci affiche les puissances de 2, jusqu'à 128 inclus.

2.4 somme des n premiers nombres

1. Écrire un programme assembleur qui calcule al somme des **n** premiers nombres. **n** est stocké à l'adresse X et le résultat doit être stocké à l'adresse Z.
2. Quel est le nombre d'instructions à exécuter en fonction de **n** ?

Exercice 3

Exercice sur les 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		

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

3.0.2 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 ?

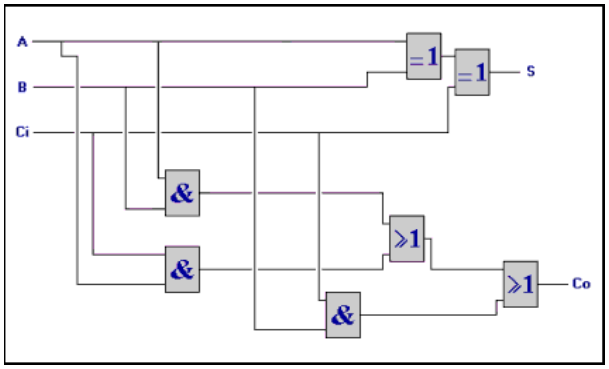
3.0.3 Expliquer pourquoi la fonction logique ET est représentée par `S = E1 . E2`, et la fonction OU par `S = E1 + E2`

3.0.4 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 4

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 ⊕ n2.

4. On souhaite maintenant générer une clé de la taille du message à partir d'un mot quelconque. On considère que le mot choisi est plus court que le message, il faut donc le reproduire un certain nombre de fois pour créer une clé de la même longueur que le message.

Par exemple, si le mot choisi est YAK pour crypter le message ALPHABET, la clé sera YAKYAKYA.)

Créer une fonction `generer_cle(mot,n)` qui renvoie la clé de longueur n à partir de la chaîne de caractères mot. 5. Recopier et compléter la table de vérité de $(\oplus)\oplus$.

E_1	E_2	$E_1 \oplus E_2$	$(E_1 \oplus E_2) \oplus E_2$
0	0	0	
0	1	1	
1	0	1	
1	1	0	

A l'aide de ce résultat, proposer une démarche pour décrypter un message crypté.

Exercice 5

Travaux pratiques : Chiffrage-dechiffrage XOR

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

5.1 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 ord fait l'inverse de celle appelée chr :

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

5.2 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)
```

5.3 fonction xor_crypt

Exemple :

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

5.4 Correction

```
def xor(n1,n2):
    b1 = bin(n1)[2:]
    print(b1)
    b2 = bin(n2)[2:]
    print(b2)
    b = ""
    for i in range(len(b1)):
        if b1[i] != b2[i]:
            b+='1'
        else:
            b+='0'
    print(int(b,2))
    return int(b,2)

def ascii (mot):
    L=[]
    for c in mot:
        L.append(ord(c))
    return L

def xor_crypt(message,cle):
    L1= ascii(message)
    L2 = ascii(cle)
    L3=[]
    for i in range(len(message)):
        L3.append(chr(xor(L1[i],L2[i])))
    return "".join(L3)
    #return L3

#xor(ascii('l')[0],ascii('Y')[0])
#print(ascii('MOI'))

print(xor_crypt('THE essager', 'YAKYAKYAKYAK'))
```

Exercice 6

Cours

6.1 L'architecture Von Neumann

Un ordinateur se définit comme une instance matérielle, concrète, d'une Machine de Turing Universelle.

Un ordinateur est une *machine* qui traite de *l'information*, stockée dans la *mémoire*, par le *processeur*. Ces traitements sont aussi stockés dans la mémoire, sous forme de *programmes*.

6.2 Rappels

Dans l'architecture Von Neumann :

- une machine universelle est contrôlée par un programme.
- Les données et programmes sont écrits sur une même mémoire (en binaire). L'ordinateur enregistre les instructions des programmes qu'il exécute dans sa mémoire vive.
- Les instructions sont executée de manière séquentielle, par un **processeur**.
- Les mémoires RAM d'une part et les mémoires non volatiles d'autre part (Disques durs, mémoires flash) sont reliées au processeur (donc aux registres du processeur) par une liaison appelée Bus. En réalité, seule la RAM est directement accessible au processeur.

6.3 Liaisons entre composants : les Bus

Il existe des Bus de différentes nature pour relier les composants :

- **bus d'adresse** : permet au processeur d'indiquer à la mémoire l'emplacement pour lire/écrire.
- **bus de données** : transporte les données entre la mémoire et le processeur.
- **bus de contrôle** : permet de coordonner le fonctionnement du processeur et de la mémoire.

Les bus peuvent être bi-directionnels : les données transitent dans les 2 sens. Mais aussi directionnel : c'est le cas du bus d'adresse entre le processeur et la RAM, qui n'est parcouru que dans le sens processeur => RAM.

Exercice 7

Description d'un ordinateur

7.1 CPU et MCU : des architectures différentes

- Certaines machines comportent des composants séparés sur la carte mère. Le microprocesseur (CPU) est relié aux autres composants par des bus.
- Pour d'autres machines, les composants occupent une même puce (un même *chip*). Ces machines sont moins chères et consomment moins d'électricité.

7.2 Echelle macro : périphériques

Un ordinateur comprend des périphériques d'entrée et sortie :

- écran, haut-parleur, souris, clavier...

Ces périphériques sont connectés aux 3 *Bus* définis plus loin et se comportent comme une mémoire. On leur affecte des adresses (différentes de celles de la RAM) et on communique avec eux en écrivant à ces adresses. Le Bus de contrôle va detecter un changement sur l'une de ces adresses et interrompre le traitement en cours.

7.3 Echelle intermédiaire

A une echelle plus proche du processeur : Un ordinateur a une carte mère sur laquelle on trouve différents ports et supports de cartes :

- port de branchement des mémoires, disques durs, port RJ45 (reseau), carte wifi, carte son, carte graphique, ...

Certains ports vont ajouter des extensions à l'ordinateur (carte son, video, wifi). Certains vont être essentiels au fonctionnement de l'ordinateur (mémoires).

7.4 Echelle micro

Les constituants de l'ordinateur sont composés de circuits intégrés. On trouve 2 grandes catégories de circuits intégrés :

- les circuits combinatoires : l'état de sortie de ces circuits ne dépend que des états d'entrée.
- les circuits séquentiels : l'état de sortie dépend des états d'entrée, mais aussi de l'état courant du circuit. Ces circuits peuvent ainsi stocker une valeur appelée état courant. Les registres sont typiquement des circuits intégrant des circuits séquantiels.

Exercice 8

Le fonctionnement du processeur

8.1 Généralités

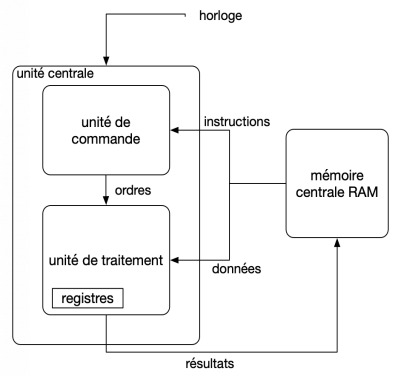


FIGURE 2 – Architecture de Von-Neumann - processeur

L'architecture d'un **processeur** (CPU) comporte 2 parties :

- L'**UC** : C'est l'*unité de commande*, **UC**, qui traduit chacune des instructions successives du programme en opérations élémentaires. C'est cette partie du processeur qui est chargée d'obtenir les instructions successives du programme et de les executer. L'**UC** contient :
  - le registre Program Counter, PC
  - le registre d'instruction RI
  - une horloge.
- La **partie opérative** qui comprend les outils permettent de réaliser les actions élémentaires ordonnés par l'UC. Cette **unité de traitement** est constituée de :
  - l'**UAL** (Unité Arithmétique et Logique) qui réalise les opérations arithmétiques sur des entiers.
  - des **registres internes**, pour stocker de manière très provisoire les retenues des calculs par exemple.

L'UC traite une séquence d'instructions, alors que l'UAL traite une séquence de données. L'**UC** est chargée de la reconnaissance des instructions et de leur exécution par l'unité de traitement au *rythme de l'horloge.*

Le processeur réalise le **cycle de Von Neumann** :

1. Lire une case mémoire d'adresse PC (envoyer l'adresse à la mémoire, et recevoir en retour la donnée à cette adresse).
2. Interpréter cette donnée comme une instruction, et l'exécuter
3. Ajouter 1 au *program Counter* (PC), qui stocke l'adresse de la prochaine instruction.
4. Recommencer

8.2 Cycle d'execution

Le processeur ne manipule *pas directement* les données mais leurs *adresses.*

Des registres servent à lire et écrire les données en mémoire. Ce sont les registres d'adresse et de données.

8.2.1 Utilisation en lecture

- registre d'adresse : stocke l'adresse de la donnée, avant d'envoyer le signal de lecture
- registre de données : la mémoire va chercher le contenu de l'adresse qui lui est envoyée, et retourne la donnée vers le registre de données.

La donnée peut être interprétée comme une instruction. Celle-ci est alors décomposée en un code d'opération (OP) ainsi qu'en 2 opérandes, qui sont les adresses d'une données (A et B). OP, A et B sont stockés dans le registre d'instruction (RI).

8.2.2 Execution de l'opération

La figure ci-contre représente un schéma classique d'UAL. Celle-ci possède deux entrées A et B sur lesquelles on présente les données à traiter. L'entrée F désigne l'opération à effectuer (vient du code OP).

Enfin, l'UAL possède deux sorties, R qui est le résultat de l'opération, et D les éventuels drapeaux. (information d'erreur ou de dépassement)

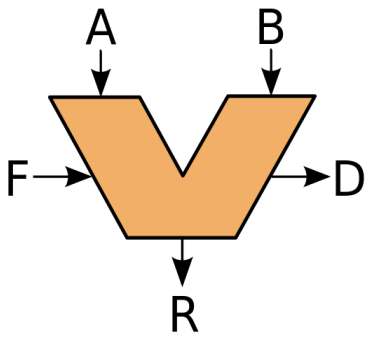


Figure 3 – UAL - Unité Arithmétique et Logique

8.2.3 Utilisation en écriture

- registre d'adresse : envoie à la mémoire l'adresse pour l'écriture
- registre de données : envoie la donnée à la mémoire, qui va écrire celle-ci à l'adresse reçue.