Exercice 3 (8 points)

Cet exercice porte sur l'architecture matérielle (dont le langage assembleur), les structures linéaires de données et la programmation orientée objet.

Les deux parties peuvent être traitées indépendamment.

Partie A

L'architecture moderne des ordinateurs est basée sur un modèle conçu en 1945 dans lequel la mémoire vive de l'ordinateur (la RAM) est utilisée pour stocker à la fois les instructions et les données demandées ou produites par les programmes.

1. Donner le nom de l'inventeur de ce modèle, ayant donné son nom au modèle.
2. Sur la copie, associer un des termes parmi les propositions suivantes à chaque numéro sur le schéma ci-dessous représentant cette architecture : 'Mémoire RAM', 'Processeur', 'Unité arithmétique et logique' et 'Unité de contrôle'.

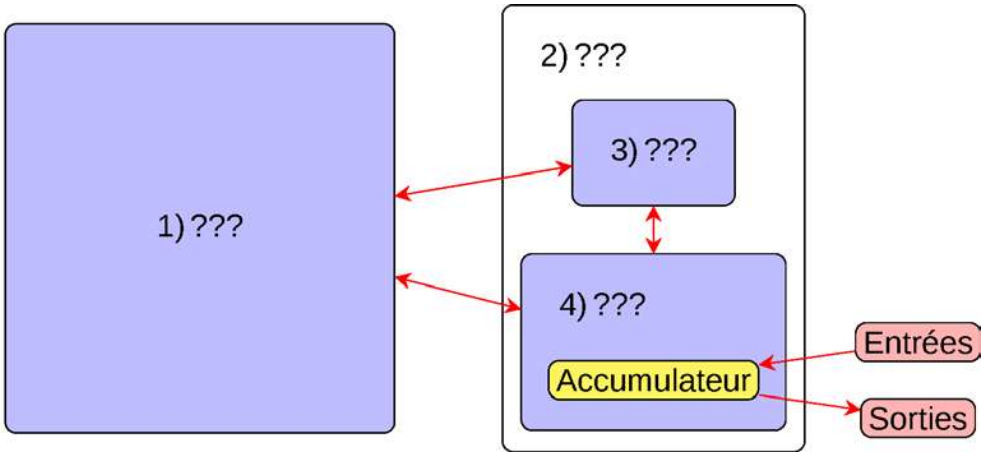


Figure 1. Modèle d'architecture moderne des ordinateurs.

3. Expliquer ce que signifie le fait que la mémoire RAM soit 'volatile'.
4. Classer les types de mémoire suivants selon leur rapidité d'accès aux données, de la plus rapide à la plus lente.
   - Mémoire cache
   - Disque dur
   - Mémoire vive
   - Registre

Les instructions d'un programme Python ne peuvent pas être exécutées telles quelles par l'unité de contrôle, c'est pourquoi elles sont transformées en langage machine. Ainsi, un programme écrit en Python qui contient une centaine de lignes peut correspondre à quelques dizaines de milliers d'octets en *langage machine* à faire exécuter par l'unité de contrôle.

Pour pallier la difficulté d'écrire de très grandes suites d'octets, des langages très bas niveau (c'est-à-dire proches du langage machine) mais lisibles par les humains ont été conçus. C'est le cas du langage assembleur RISC-V sur 32 bits qui est étudié ici.

Dans un code écrit en assembleur, chaque ligne est soit une instruction qui permet de manipuler des variables appelées *registres*, soit une étiquette qui permet d'identifier une position dans le code. Par exemple, le code ci-dessous permet de mettre dans le registre `r5` l'addition entre la valeur du registre `r2` et la valeur `3` si la valeur du registre `r7` est strictement supérieure à la valeur du registre `r6`, puis ajoute `1` au registre `r5` dans tous les cas. (On suppose que l'unité de contrôle dispose de 8 registres notés `r1` à `r8`).

```
    ble r7, r6, SUITE
    addi r5, r2, 3
SUITE:
    addi r5, r5, 1
```

Voici un extrait des instructions disponibles en RISC-V sur 32 bits :

- `addi rA, rB, v` : permet de placer dans le registre `rA` la somme de la valeur du registre `rB` et de la valeur `v` ;
- `add rA, rB, rC` : permet de placer dans le registre `rA` la somme de la valeur du registre `rB` et de celle du registre `rC` ;
- `sub rA, rB, rC` : permet de placer dans le registre `rA` la valeur du registre `rB` moins celle du registre `rC` ;
- `beq rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est égale à la valeur du registre `rB` ;
- `bne rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` n'est pas égale à la valeur du registre `rB` ;
- `bgt rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est strictement plus grande que la valeur du registre `rB` ;
- `ble rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est plus petite ou égale à la valeur du registre `rB`.

5. Écrire une instruction en assembleur permettant de mettre dans le registre `r2` le résultat de la soustraction entre la valeur du registre `r5` et celle du registre `r4`.

6. Expliquer ce que réalise l'instruction `add r1 r2 0`.

7. Écrire une suite d'instructions en assembleur permettant d'échanger les contenus des registres `r1` et `r2`. On pourra utiliser le registre `r3` pour stocker temporairement des valeurs.

Au niveau matériel, les opérations sont effectuées dans l'unité arithmétique et logique qui contient des circuits logiques booléens combinatoires. On donne ci-dessous les portes logiques effectuant les opérations booléennes classiques.

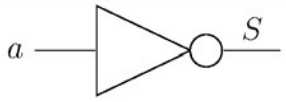
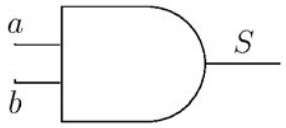
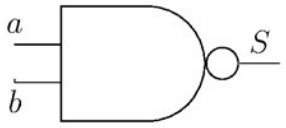
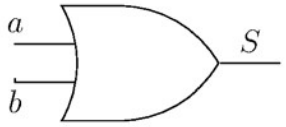
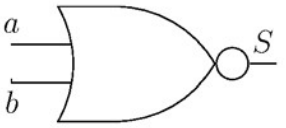
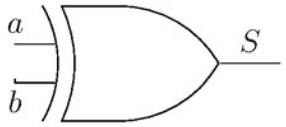
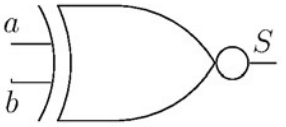
Nom	Symbole	Table de vérité
NOT		a | S 0 | 1 1 | 0
AND		a | b | S 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1
NAND		a | b | S 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
OR		a | b | S 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1
NOR		a | b | S 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0
XOR		a | b | S 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
NXOR		a | b | S 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1

Figure 2. Portes logiques.

Dans une table de vérité, a et b représentent chacun une valeur binaire d'entrée de la porte et S représente la valeur binaire de sortie de la porte.

8. Donner pour chaque lettre A, B, C et D sa valeur binaire dans la table de vérité du circuit combinatoire ci-après.

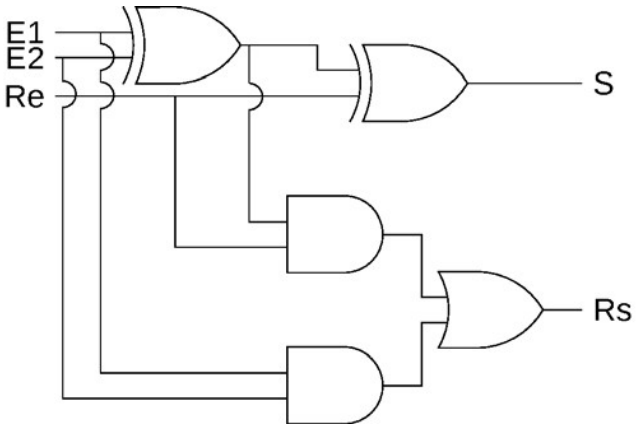


Figure 3. Circuit logique

Table de vérité				
E1	E2	Re	**Rs**	**S**
0	0	0	0	0
0	0	1	**A**	**B**
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	**C**	0
1	1	0	1	0
1	1	1	1	**D**

9. En observant le tableau complété à la question précédente, expliquer ce que fait le circuit étudié.

Partie B

Une structure souvent utilisée dans les mémoires des systèmes modernes est la *mémoire tampon*. Il s'agit d'une zone de la mémoire réelle utilisée pour entreposer temporairement des données, par exemple par deux processus ne travaillant pas au même rythme et ayant pourtant besoin d'être synchronisés.

Cette mémoire tampon fonctionne comme une file, ainsi les données sont stockées dans l'ordre chronologique de leur enregistrement et sont sorties de la mémoire tampon dans le même ordre.

10. Entre FIFO et LIFO, donner le terme correspondant le mieux à la structure de file.

La file représentant la mémoire tampon est un tableau circulaire, c'est-à-dire que lorsque l'on souhaite ajouter un élément après la dernière case du tableau, on le place dans la première case du tableau.

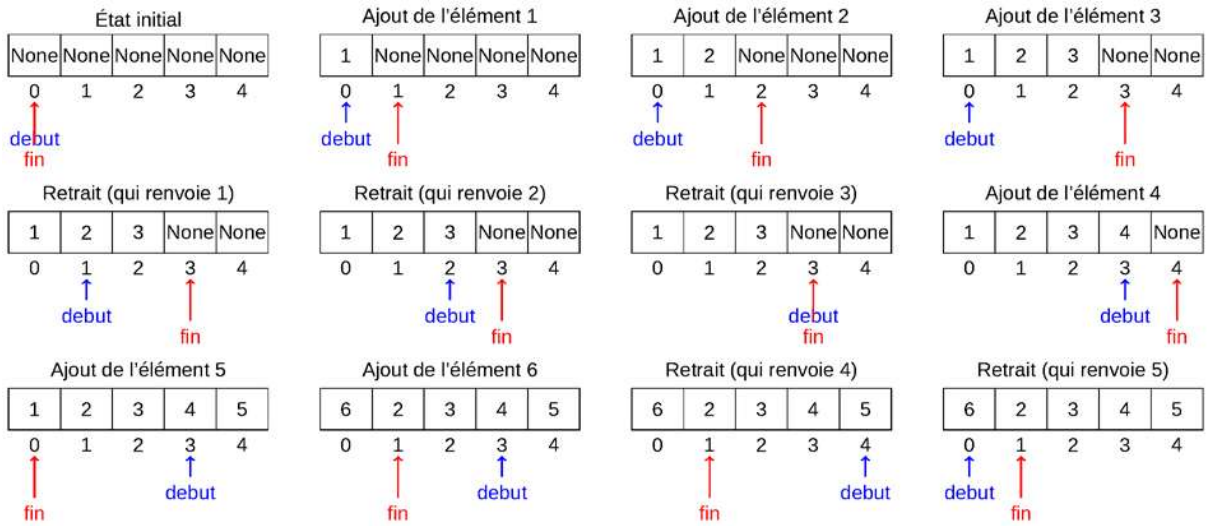


Figure 4. Exemple de manipulation d'une mémoire tampon de capacité 5.

On remarque que le retrait d'un élément de la mémoire tampon consiste simplement au décalage de l'indice de début, sans retirer réellement l'élément du tableau. On suppose que la capacité maximale d'une file n'est jamais atteinte, c'est-à-dire que les indices de début et de fin ne sont égaux que lorsque la file est vide.

La classe `Tampon` ci-dessous implémente une mémoire tampon à l'aide d'une liste Python. La longueur de cette liste est fixée définitivement à l'initialisation à l'aide du paramètre `capacite` du constructeur.

```
 1 class Tampon:
 2     def __init__(self, capacite):
 3         self.capacite = capacite
 4         self.contenu = [None for i in range(capacite)]
 5         self.debut = 0 # Indice de la tête de file
 6         self.fin = 0   # Indice de la queue de file exclu
 7
 8     def ajouter_element(self, element):
 9         self.contenu[self.fin] = element
10         self.fin = self.fin + 1
11         if self.fin == self.capacite:
12             self.fin = 0
13
14     def retirer_element(self):
15         ...
16
```

```
17      def est_vide(self):
18          return ...
```

11. Donner la valeur de l'attribut `contenu` de l'objet `memoire` à l'issue de l'appel suivant :

```
memoire = Tampon(5)
```

On considère avoir à disposition une mémoire tampon dont l'état est représenté ci-dessous.

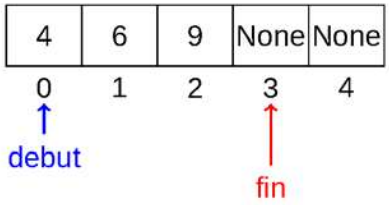


Figure 5. État de la mémoire tampon à considérer pour la question suivante.

12. Donner l'état de cette mémoire tampon, avec l'éventuel élément renvoyé, après chacune des opérations successives suivantes :

    a) retrait d'un élément ;
    b) ajout de l'élément 10 ;
    c) retrait d'un élément ;
    d) retrait d'un élément ;
    e) ajout de l'élément 3 ;
    f) retrait d'un élément ;
    g) retrait d'un élément.

13. Donner la valeur de l'attribut `contenu` de la mémoire tampon `memoire` après l'exécution du programme suivant.

```
memoire = Tampon(5)
memoire.ajouter_element('n')
memoire.ajouter_element('s')
memoire.ajouter_element('i')
memoire.retirer_element()
memoire.ajouter_element('n')
memoire.ajouter_element('f')
memoire.retirer_element()
memoire.ajouter_element('o')
```

14. Recopier et compléter la méthode `est_vide` de la classe `Tampon`.

15. Écrire le code de la méthode `retirer_element` de la classe Tampon qui renvoie l'élément en tête de file et le retire de la file. On suppose que la file est non-vide. On rappelle que l'élément n'est pas réellement retiré de la liste, seul l'indice de début est décalé.