TD 2 : Architecture des ordinateurs

Exercice 1 : Déterminer la valeur approchée de 2^{24} sans l'utilisation de la calculatrice.

Déduire une valeur en puissance de 2 du 1 Giga et 1 Téra.

On sait que : $2^{24} = 2^{10} \times 2^{10} \times 2^4$.

$2^{10} = 1024 \approx 1000 = 10^3 = 1$ kilo

$2^{24} = 2^{20} \times 2^4$

$2^{10} \times 2^{10} = 1$ Million $= 1$ Méga

$2^{24} = 16$ Méga

1 Giga = 1 Milliard $= 1024 \times 2^{20} = 2^{10} \times 2^{20} = 2^{30}$

1 Téra = 1000 Milliards = 1 Billion $= 1024 \times 2^{30} = 2^{10} \times 2^{30} = 2^{40}$

Exercice 2 :

1.1 Dans l'architecture CISC, pourquoi les instructions sont appelées complexes ?
1.2 Quels sont les différents types des instructions ?
1.3 Quelles sont les phases de traitement d'une instruction ?
1.4 Etant donnée l'instruction suivante :

Move Dest, Src ;
add Rd, Rsrc1, Rsrc2 ;

- Déterminer le code opération (opcode) et l'opérande de chaque instruction.
- Déduire le type de chaque instruction.

Réponse :

1.1 Comme son nom l'indique, les processeurs CISC utilisent des instructions complexes. Par exemple, l'addition de deux nombres entiers est considérée comme une instruction simple. Mais, une instruction qui copie un élément d'un tableau (emplacement mémoire par exemple) à l'autre et met automatiquement à jour les indices du tableau est considérée comme une instruction complexe. Au contraire, les systèmes RISC utilisent des instructions simples. En outre, les systèmes RISC supposent que les opérandes nécessaires à l'opération sont chargés dans les registres du processeur, et non dans la mémoire principale comme c'est le cas pour l'architecture CISC.

1.2 Trois adresses ; deux adresses, une seule adresse, zéro adresse.

1.3 Première phase : Fetch ; deuxième phase : Exécute

1.4 1ère instruction :

- Opcode : Move ; opérande : Dest, Src. Instruction de type deux adresses
- Opcode : Add ; opérande : Rd, Rsrc1, Rsrc2. Instruction de type trois adresses.

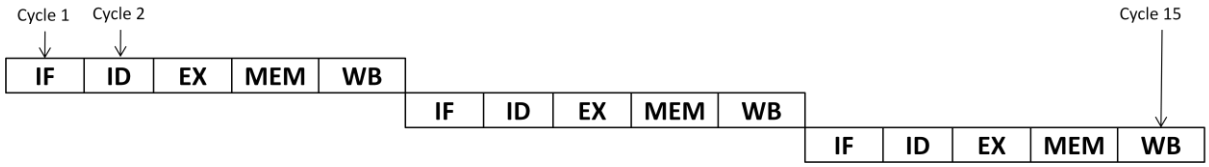
Exercice 3 :

Deux ordinateurs A et B ayant un temps d'exécution égal à 10 ns chacun. L'ordinateur A dispose de la technique de traitement *pipeline* et l'ordinateur B ne l'est pas. Que signifie le traitement *pipeline*. Pouvez-vous déduire lequel des ordinateurs A et B soit le plus rapide ?

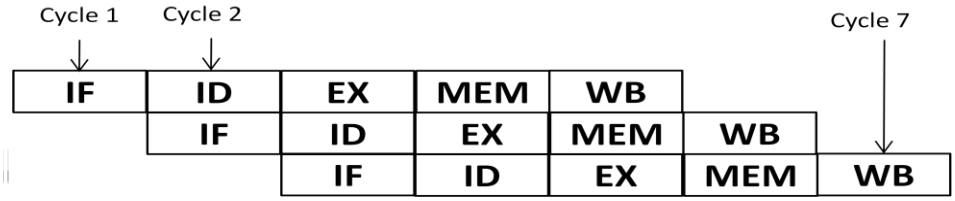
Traitement pipeline : consiste à traiter les instructions en parallèle. Etant donné que le traitement d'une instruction passe par deux phases essentielles (Fetch et Execute), il est possible d'effectuer le chargement d'une instruction avant que l'exécution de la précédente soit terminée**.** Un pipeline à 5 étages traite une instruction en 5 cycles**.** La figure ci-dessous illustre l'avantage du pipeline :

Abréviations

IF	Instruction Fetch
ID	Instruction Décode
EX	Exécution ou calcul d'adresses
MEM	Memory Access. Transfert depuis un registre vers la mémoire (STORE) ou depuis la mémoire vers le registre (LOAD)
WB	Write Back. Stock le résultat dans un registre



L'exécution de trois instructions dans un ordinateur sans pipeline nécessite 15 cycles



L'exécution de trois instructions dans un ordinateur avec un traitement pipeline nécessite seulement 7 cycles.
On peut conclure bien évidemment que l'ordinateur A et plus performant que B.

Exercice 4:

Supposons que le registre R0 de taille 32bits d'un microprocesseur supportant le système de stockage des données *big* et *little-endian*, est initialisé par la valeur 0x23456789. Donner le contenu du R0 après exécution du programme ci-dessous en *big-endian (gros-boutiste).* Même question dans le cas où le programme soit exécuté en *little-endain (petit-boutise).*

LOAD Word R0, MEM[$0] ➔ charger dans R0 le mot mémoire qui se trouve à l'adresse ($0)

LOAD BYTE R0, MEM[1 + $0] ➔ charger le byte qui se trouve dans la mémoire à l'adresse (1 + $0)

Réponse :

Big-Endian

Mot d'adresse 0

Adresse du Byte	0	1	2	3
Valeur	23	45	67	89
	MSB			LSB

Little-Endian

Mot d'adresse 0

Adresse du Byte	3	2	1	0
Valeur	23	45	67	89
	MSB			LSB

Après exécution de l'instruction (load Byte), le registre R0 devrait contenir la valeur 0x00000045 en gros-boutiste et la valeur 0x00000067 en petit-boutiste.

Exercice 5 :

1. Quel est l'avantage de l'alignement des mots mémoires ?
2. Etant donné les adresses mémoires suivantes :

(a) 12345678 (c) 9128ADCC

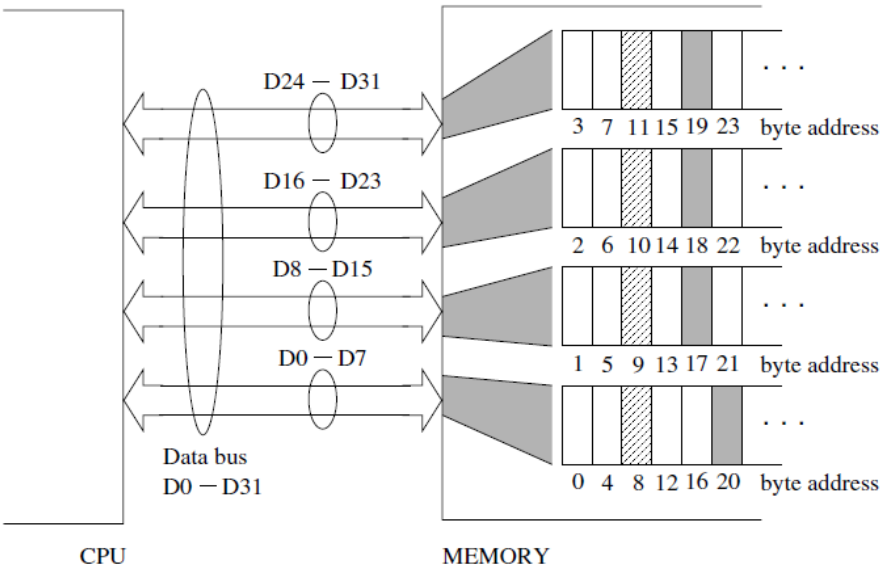
(b) ABCD755A (d) 38B0F050

- Déduire si les 32-bits stockés en mémoire sont alignés ou non.
- Même question dans le cas des 64-bits.

Réponse :

1- L'alignement des mots mémoires contribue à l'amélioration des performances du CPU, c.-à-d. la réduction du temps d'exécution des programmes.

En effet, supposons que nous voulons lire un mot mémoire de taille 32-bits, par hypothèse que la taille du bus de données est égale à 32-bits. Si l'adresse de cette variable (mot) est un multiple de 4, les 32-bits sont stockés dans une seule colonne de la mémoire (alignement). Ainsi le processeur peut lire les 32-bits en un seul cycle. Au contraire, si les 32-bits ne sont pas alignés, c.-à-d., les 32-bits se trouve dans colonne de la mémoire, le CPU lit deux colonnes de données mémoire et effectue l'assemblage des 32-bits du mot mémoire. Ce scénario est illustré par la figure ci-dessous.



Dans cette figure les 32-bits stockés à l'adresse 8 (représentés par des cases hachée) sont alignés. Grâce à cet alignement, le CPU peut lire ces données en un seul cycle. D'autre part, les données stockées à l'adresse 17 (représentées par des cases grises) ne sont pas alignées. La lecture des ces données nécessite 2 cycles : un cycle pour lire les 32-bits à l'adresse 16 et le deuxième cycle pour lire les 32-bits à l'adresse 20 (où se trouve le dernier octet).

2. Alignement des données

Le tableau ci-dessous récapitule les conditions d'alignement des données mémoires.

Taille des données	**Condition d'alignement**
16-bits (2 octets)	Adresse est un multiple de 2 c.-à-d., le bit de poids faible de l'adresse devrait être 0.
32-bits (4 octets)	Adresse est un multiple de 4 c.-à-d., les 2 bits de poids faible de l'adresse devraient être 00.
64-bits (8 octets)	Adresse est un multiple de 8 c.-à-d., les 3 bits de poids faible de l'adresse devraient être 000.

Par conséquent,

Adresse	**32-bits**	**64-bits**
(a) 12345678	Aligné	Aligné
(b) ABCD755A	Non-aligné	Non-aligné
(c) 9128ADCC	Aligné	Non-aligné
(d) 38B0F050	Aligné	Aligné